Ngôn ngữ lập trình C

Biên soạn: Đỗ Bình Nguyên

Mục tiêu bài học

- Trang bị cho học viên các kiến thức
  - Cấu trúc một chương trình C
  - Các loại hàm, biến trong ngôn ngữ C
  - Các cấu trúc điều khiển, vòng lặp của ngôn ngữ C

Cấu trúc chương trình C

Cấu trúc chương trình C

- Ví dụ:

```
#include <p89v51rx2.h> // thư viện dùng cho P8951RD2
void delay (unsigned int n) //chương trình delay
{
        unsigned int i,j;
        for(i = 0; i < n; i ++)
                for(j = 0; j < 100; j ++);
}
/*----------------------------
      Chương trình chính
-----------------------------*/
void main(void)
{
        while(1)
        {
                P1 = 0;
                delay(1000);
                P1 = 255;
                delay(1000);
        }
}
```

Cấu trúc chương trình C t.t.

```
#include <p89v51rx2.h>

unsigned char x;
int y, Y;

void func1 (void);

void func2 (void)
{
… //các câu lệnh
}
char func3 (void)
{
 long tmp = 1;
… //các câu lệnh
}

void main (void)
{
…
}

void func1 (void)
{
…
}
```

Khai báo tập tin thư viện

Khai báo biến toàn cục

Khai báo prototype cho chương trình con

Khai báo chương trình con

Khai báo biến cục bộ

Khai báo chương trình chính

Mỗi chương trình bắt buộc phải có một hàm main

Khai báo chương trình con

Cấu trúc chương trình C t.t.

- Các câu lệnh trong hàm chính có thể gọi các hàm con đã khai báo hoặc không.
- Hàm chính và hàm con chỉ có thể gọi các hàm được khai báo phía trên nó.
- Các câu lệnh trong C kết thúc bằng dấu **;**
- Khi có lời gọi hàm con nào thì chương trình sẽ nhảy đến thực hiện hàm con đó. Sau khi thực hiện xong sẽ nhảy về thực hiện tiếp các hàm hoặc câu lệnh trong chương trình chính.
- Đặt các lời giải thích bằng dấu // hoặc /* …*/

Biến trong C

Các kiểu biến

Kiểu biến	Độ dài (bit)	Miền giá trị
bit	1	0, 1
signed char	8	-128 … +127
unsigned char	8	0 … 255
enum	8/16	-128 … +127 hoặc -32,768 … 32,767
signed short int	16	-32,768 … 32,767
unsigned short int	16	0 … 65,535
signed int	16	-32,768 … 32,767
unsigned int	16	0 … 65,535
signed long int	32	-2,147,483,648 … +2,147,483,647
unsigned long int	32	0 … 4,294,967,295
float	32	±1.175494E-38 … ±3.402823E+38
double	32	±1.175494E-38 … ±3.402823E+38
sbit	1	0, 1
sfr	8	0 … 255
sfr16	16	0 … 65,535

Vùng chứa biến

Tên	Mô tả
code	Bộ nhớ chương trình (64KB) Truy cập bằng lệnh MOVC @A +DPTR
data	Vùng RAM nội truy cập trực tiếp
idata	Vùng RAM nội truy cập gián tiếp
xdata	Vùng RAM ngoài (64KB) Truy cập bằng lệnh MOVX @DPTR

Khai báo biến

Kiểu biến Vùng chứa ***Tên biến***

- Ví dụ:
  - **unsigned char** tmp; // biến kiểu char không dấu, chứa trong vùng RAM nội truy cập trực tiếp
- Có thể gán giá trị ban đầu cho biến ngay khi khai báo.
- Ví dụ:
  - **unsigned int xdata** day_of_week = 7; // biến kiểu int có dấu, chứa trong RAM ngoài
  - **bit** flagRun = 1; // biến kiểu bit
- Có thể khai báo cùng lúc nhiều biến có cùng kiểu.
- Ví dụ:
  - **signed char** hour, min, sec;
- Biến phải được khai báo trước khi sử dụng.

Khai báo biến t.t.

- Để biểu diễn một dãy số hay một bảng dữ liệu, ta có dữ liệu kiểu mảng.
- Mảng là một tập hợp nhiều phần tử có cùng kiểu giá trị, cùng một tên. Mỗi phần tử được truy cập bằng chỉ số của phần tử đó. Chỉ số mảng bắt đầu tính từ 0.
- Khai báo mảng như sau:

Loại mảng Vùng chứa Tên mảng [Kích thước] ...[Kích thước]

- Ví dụ:
  - **int** arrSin[10]; // mảng kiểu int, một chiều, có 10 phần tử, chứa trong RAM nội.
  - **char xdata** arrLed [8][16]; //mảng kiểu char không dấu, 2 chiều, kích thước 8 x 16, chứa trong RAM ngoài.
  - arrSin[5] = 3; // gán giá trị 3 cho phần tử thứ 6 của mảng arrSin
  - **char** strName[] = “Lac Hong University”; // khai báo chuỗi

Các toán tử

Toán tử gán (=)

- Toán tử gán dùng để gán một giá trị nào đó cho một biến.
- Vế trái bắt buộc phải là một biến còn vế phải có thể là bất kì hằng, biến hay kết quả của một biểu thức.
- Ví dụ:
  - a = 5; //gán giá trị nguyên 5 cho biến a.
  - a = b = c = 5; //gán giá trị 5 cho cả ba biến a, b, c
  - a = b; //gán giá trị của biến b cho biến a, sự thay đổi của b sau đó sẽ không ảnh hưởng đến giá trị của a.

Toán tử số học

- Năm toán tử số học được hỗ trợ bởi ngôn ngữ là: cộng (+), trừ (-), nhân (*), chia (/), chia lấy phần dư (%)
- Thứ tự thực hiện các toán tử này giống như trong toán học.
- Ví dụ:
  - a = 11 % 3; // biến a sẽ mang giá trị 2 sau khi thực hiện

Toán tử quan hệ

- = = Bằng
- != Khác
- > Lớn hơn
- < Nhỏ hơn
- >= Lớn hơn hoặc bằng
- <= Nhỏ hơn hoặc bằng
- Ví dụ:
  - (7 = = 5) //trả về giá trị false
  - (6 >= 6) sẽ trả giá trị true
  - Cho a=2, b=3 và c=6
    - (a * b >= c) //sẽ trả giá trị true
    - (b + 4 < a * c) //sẽ trả giá trị false

Toán tử logic

- **!** Toán tử NOT
- **&&** :Toán tử AND
- || :Toán tử OR
- Toán tử logic && và || được sử dụng khi tính toán hai biểu thức để lấy ra một kết quả duy nhất.
- Ví dụ:
- ((5 == 5) && (3 > 6)) trả về false (true && false).
- ((5 == 5) || (3 > 6)) trả về true (true || false).

Toán tử thao tác bit

- Các toán tử thao tác bit thay đổi các bit biểu diễn một biến, có nghĩa là thay đổi biểu diễn nhị phân của chúng
- **&** :AND
- | :OR
- ^ :Exclusive OR
- ~ :Đảo bit
- << :Dịch bit sang trái
- >> :Dịch bit sang phải
- VD:
  - (0x0F & 0xAA)=0x0A //0000.1111 & 1010.1010 = 0000.1010
  - ~ 0xAA = 0x55 // ~ 1010.1010 = 0101.0101

Toán tử điều kiện (?)

- Toán tử điều kiện tính toán một biểu thức và trả về một giá trị khác tuỳ thuộc vào biểu thức đó là đúng hay sai.

Điều kiện ? Kết quả 1 : Kết quả 2

- Nếu **Điều kiện** là *true* thì giá trị trả về sẽ là **Kết quả 1**, nếu không giá trị trả về là **Kết quả 2**.
- Ví dụ:
  - 7 = = 5 ? 4 : 3 //trả về 3 vì 7 ≠ 5.
  - 7 = = 5 + 2 ? 4 : 3 //trả về 4 vì 7 = 5 + 2.
  - 5 > 3 ? a : b //trả về a, vì 5 > 3.
  - a > b ? a : b //trả về giá trị lớn hơn, a hoặc b.

Toán tử gán phức hợp

- Các toán tử gán phức hợp (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
- Ví dụ:
  - value += increase; //value = value + increase
  - a -= 5; // a = a - 5
  - a /= b; //a = a / b
  - price *= units + 1; //price = price * (units + 1)

Tăng (++) và giảm (--)

- Ví dụ:

  a ++; a += 1; a = a + 1;

- Tính chất tiền tố - hậu tố
  - Toán tử này có thể viết trước tên biến (++ a) hoặc sau (a ++)
  - Trong trường hợp tiền tố (++ a) giá trị biến được tăng trước khi biểu thức được tính, giá trị đã tăng được sử dụng trong biểu thức.
  - Trường hợp ngược lại (a ++) giá trị của biến được tăng sau khi đã tính toán.
- Ví dụ :
  - Giả sử B = 3;
  - A = ++ B; //Kết quả: A = 4, B = 4
  - A = B ++; // Kết quả: A = 3, B = 4

Các cấu trúc rẽ nhánh

Cấu trúc **if**

```
if(biểu thức điều kiện)
{
      <khối lệnh> ;
}
```

- Nếu biểu thức điều kiện cho kết quả khác không thì thực hiện khối lệnh.
- Ví dụ:

```
if (sensor == 1) // nếu cảm biến tác động
{
      turn_relay_on(); // bật relay
      stop_motor(); // dừng động cơ
}
```

Cấu trúc **if .. else**

```
if (biểu thức điều kiện)
{
        <khối lệnh 1>;
}
else
{
        <khối lệnh 2>;
}
```

- Nếu biểu thức điều kiện cho kết quả khác không thì thực hiện khối lệnh 1, ngược lại thì cho thực hiện khối lệnh 2.

Cấu trúc **switch … case**

- **switch** (biểu thức)
  {
  case n1:
  các câu lệnh;
  break;
  case n2:
  các câu lệnh;
  break;
  ……
  case ni:
  <các câu lệnh>;
  break;
  [**default**: các câu lệnh]
  }

Cấu trúc **switch … case** t.t.

- i là các hằng số nguyên hoặc ký tự.
- Phụ thuộc vào giá trị của biểu thức viết sau **switch**, nếu:
  - Giá trị này bằng **ni** thì thực hiện câu lệnh sau **case ni**
  - Khi giá trị biểu thức không thỏa tất cả các **ni** thì thực hiện câu lệnh sau **default** nếu có, hoặc thoát khỏi câu lệnh **switch**.
- Khi chương trình đã thực hiện xong câu lệnh của **case ni** nào đó thì nó sẽ thực hiện luôn các lệnh thuộc **case** bên dưới nó mà không xét lại điều kiện. Vì vậy, để chương trình thoát khỏi lệnh **switch** sau khi thực hiện xong một trường hợp, ta dùng lệnh **break**.

Cấu trúc **switch … case** t.t.

- Ví dụ:

```
switch (button) //nút nào được nhấn ??
{
        case 1:
                LCD_string("Turning left") ;
                turn_left();
                break ;
        case 2:
                LCD_string("Turning right") ;
                turn_right();
                break ;
        default:
                LCD_string("Go straight");
                go_straight();
}
```

Các cấu trúc lặp

Cấu trúc **for**

```
for (<bt1>; <bt2>; <bt3>)
{
        <khối lệnh>;
}
```

- Bất kỳ biểu thức nào trong 3 biểu thức trên cũng có thể vắng mặt (có thể vắng mặt cả 3) nhưng phải giữ dấu **;**
- bt1: thường là phép gán để tạo giá trị ban đầu cho biến điều khiển
- bt2: là điều kiện để tiếp tục vòng lặp
- bt3: là một phép gán để thay đổi giá trị của biến điều khiển

Cấu trúc **for** t.t.

- Hoạt động của cấu trúc điều khiển for:
  - Bước 1: Thực hiện bt1
  - Bước 2: Xác định giá trị bt2
  - Bước 3:
    - Nếu bt2 sai: kết thúc vòng lặp for
    - Nếu bt2 đúng: thực hiện <khối lệnh>. Sau đó sẽ thực hiện bước 4.
  - Bước 4: Xác định giá trị bt3, sau đó quay lại bước 2 để bắt đầu vòng lặp mới

Cấu trúc **for** t.t.

- Ví dụ:

```
for (i = 0; i < 10; i ++) //thực hiện 10 lần
{
      P1_0 = 0; //tắt led
      delay(100);
      P1_0 = 1; //bật led
      delay(100);
}
```

Cấu trúc **while**

```
while (bieu_thuc)
{
        <khối lệnh>;
}
```

- Hoạt động của cấu trúc while:
  - Bước 1: Kiểm tra giá trị bieu_thuc
  - Bước 2:
    - Nếu bieu_thuc cho giá trị đúng: thực hiện <khối lệnh> và quay về bước 1.
    - Nếu bieu_thuc sai: thoát khỏi vòng lặp

Cấu trúc **while** t.t.

- Ví dụ:

```
i = 0;
while (i < 10) //thực hiện 10 lần
{
      P1_0 = 0; //tắt led
      delay(100);
      P1_0 = 1; //bật led
      delay(100);
      i ++;
}
```

Cấu trúc **do … while**

```
do
{
        <khối lệnh>;
} while (bieu_thuc)
```

- Hoạt động của cấu trúc while:
  - Bước 1: thực hiện <khối lệnh>
  - Bước 2: Kiểm tra giá trị bieu_thuc
    - Nếu bieu_thuc cho giá trị đúng: quay về bước 1.
    - Nếu bieu_thuc sai: thoát khỏi vòng lặp

Cấu trúc **do … while** t.t.

- Ví dụ:

```
i = 0;
do
{
      P1_0 = 0; //tắt led
      delay(100);
      P1_0 = 1; //bật led
      delay(100);
      i ++;
} while (i < 10) //thực hiện 10 lần
```

Lệnh **break**

- Cho phép thoát sớm ra khỏi một vòng lặp (for, while, do … while) mà không cần xét đến điều kiện lặp
- Ví dụ:

```
while (1)
{
      send_data_to_PC(); //truyền dữ liệu về máy tính
      if (button == 0) break; //nếu người dùng nhấn nút => thoát khỏi vòng lặp
}
```

Lệnh **continue**

- Làm cho chương trình thực hiện tiếp vòng lặp mới, bỏ qua các câu lệnh khác nằm sau lệnh **continue** này
- Chỉ được dùng trong các vòng lặp
- Ví dụ:

```
unsigned char strSource= "01203405678";
unsigned char i;
for (i = 0; i < 9; i ++)
{
      if (strSource[i] == '0') continue; //
nếu ký tự là '0', bỏ qua
      blink(); // nếu ký tự '0', chớp led
}
```

Hàm

Hàm cơ bản

- Khai báo hàm như sau:

Kiểu giá trị trả về Tên hàm (*Kiểu tham số nhận vào* Tên tham số [, …])

- *Kiểu giá trị trả về* & *Kiểu tham số*: Là một trong các kiểu biến đã trình bày ở trên
- Nếu hàm không có giá trị trả về, không nhận tham số vào thì khai báo **void**
- Nếu có nhiều tham số thì các tham số được khai báo cách nhau bởi dấu **,**
- Các lệnh xử lý trong hàm phải nằm giữa cặp dấu **{ … }**

Hàm cơ bản t.t.

- Ví dụ:
  - **char** getTemp (**unsigned char** channel) {…}
  - **void** openLamp (**unsigned char** num) {…}
  - **void** turn_left (**void**) {…}
  - **unsigned int** CountDown (**void**) {…}

Lệnh **return**

- Lệnh **return** cho phép thoát ra khỏi một hàm để trở về hàm đã gọi nó.
- Khi gặp lệnh **return**, chương trình sẽ bỏ qua các lệnh sau nó để thoát ra khỏi hàm.
- Các dạng của lệnh **return**:
  - `return;`
  - `return (bieu_thuc);`
  - `return bieu_thuc;`

Lệnh **return** t.t.

- Trong thân hàm có thể sử dụng một, hoặc một vài lệnh **return**. Hoặc cũng có thể không sử dụng lệnh này.
- Nếu kiểu giá trị trả về của hàm không phải **void** (Hàm có trả về giá trị) thì trong thân hàm phải sử dụng lệnh **return** để trả giá trị về cho chương trình.

Lệnh **return** t.t.

- Ví dụ 1:

```
void blink_led (void)
{
      while (1)
      { //chớp led và kiểm tra nút nhấn
            P1_0 = 0;
            delay();
            P1_1 = 1;
            delay();
            if (button == 0) return; // nếu có
nút nhấn => thoát khỏi vòng lặp
      }
}
```

Lệnh **return** t.t.

- Ví dụ 2:

```
/*
Hàm tìm giá trị lớn nhất trong hai số
*/
char max (char a, char b)
{
      if (a > b) return a;
      return b;
}
```

Hàm ngắt

- Hàm ngắt là hàm do Keil C định nghĩa thêm để dùng riêng cho vi điều khiển. Đó là hàm sẽ được thực hiện khi có ngắt xảy ra.
- Khai báo hàm ngắt như sau

void Tên hàm (**void**) **interrupt** interrupt_number **using** bank thanh ghi

- Hàm ngắt không được trả về giá trị và không được nhận tham số vào.

Hàm ngắt t.t.

- Interrupt_number là các giá trị tương ứng với nguồn ngắt được cho trong bảng sau

Interrupt number	Nguồn ngắt
0	Ngắt ngoài 0
1	Timer/Counter0
2	Ngắt ngoài 1
3	Timer/Counter 1
4	Ngắt nối tiếp
5	Timer/Count 2 (Đối với họ 8052)

- Bank thanh ghi: là bank thanh ghi trong RAM mà hàm ngắt sẽ thực hiện trên đó. Nhận các giá trị từ 0 … 3

Hàm ngắt t.t.

- Ví dụ:

```
unsigned int intCnt;
unsigned char second;

void timer0 (void) interrupt 1 using 2
{
      if (++intCnt == 4000) //đếm đến 4000
      {
            second ++; // đếm giây
            intCnt = 0; // reset intCnt
      }
}
```

NGÔN NGỮ LẬP C

Giáo viên

Vũ Văn Định

Bài 1: Tổng quan về ngôn ngữ lập trình C

- Ngôn ngữ C có một số các đặc điểm nổi bật sau :
  - Bộ lệnh phù hợp với phương pháp lập trình cấu trúc.
  - Kiểu dữ liệu phong phú.
  - Một chương trình C bao giờ cũng gồm một hoặc nhiều hàm và các hàm rời nhau.
  - Là ngôn ngữ linh động về cú pháp, chấp nhận nhiều cách thể hiện chương trình .

I. Hướng dẫn sử dụng môi trương kết hợp Turbo C

1. **Khởi động**

C1: Từ DOS [đường dẫn]\ TC.EXE

C2: Từ Win C -> TC -> BIN -> TC.EXE

C3: Start -> Run -> C:\TC\BIN\TC.EXE

2. **Mở File**

Mở file mới : File -> New

Mở file đã có: File -> Open

3. **Ghi File**

Save (F2) : Ghi tệp mới đang soạn thảo vào đĩa

Save as : Ghi tệp đang soạn thảo vào đĩa theo tên mới hoặc đe lên tệp đã có

- Chạy một chương trình
  - F9 : Biên dịch
  - Ctrl F9 : Thực thi chương trình
  - Alt F5 : Xem kết quả
- Thoát khỏi C
  - Thoát tạm thời về DOS : Dos Shell
  - Thoát hẳn khỏi C: File \ Quit (Alt + X)

II. Giới thiệu ngôn ngữ lập trình C

1. Các thành phần của NNLT C

- Tập các ký tự
  - Chữ cái: A .. Z, a .. z
  - Chữ số : 0..9
  - Ký hiệu toán học : + - * / = ( )
  - Ký tự gạch nối: _
  - Các ký hiệu đặc biệt khác như : . , ; : [ ] { } ? ! \ & | % # $,…
- Từ khoá
  - Là những từ có một ý nghĩa hoàn toàn xác định
  - Asm, char, do, int, float, for, do, While,…
- Tên
  - Dùng để xác định các đại lượng khác nhau trong một chương trình
  - Bắt đầu bằng chữ cái hoặc gạch nối
  - Độ dài cực đại mặc định là 32

2. Các kiểu dữ liệu cơ sở trong C

- Kiểu số ký tự (char)
- Kiểu số nguyên (int)
- Kiểu dấu phẩy động (chính xác đơn (float), chính xác kép (double))
- Kiểu void

2.1 Kiểu ký tự (char)

- Một giá trị kiểu ký tự (char) chiếm 1 byte trong bộ nhớ và biểu diễn một ký tự thông qua bảng mã ASCII.
- *Ví dụ*

Ký tự	Mã ASCII
0	48
1	49
2	50
A	65
a	97

- Trong ngôn ngữ C cung cấp hai kiểu ký tự (char) là signed char và unsigned char

Phạm vi	Số ký tự	Kích thước
signed char	-128..127	1 byte
unsigned char	0..255	1 byte

Ví dụ : char ch, ch1;

ch= ‘a’ ; ch1= 97;

2.2 Kiểu số nguyên (int)

Kiểu số nguyên trong C gồm các kiểu sau:

Kiểu	Phạm vi biểu diễn	Kích thước
int	-32768 -> 32767	2 byte
Unsigned int	0 -> 65535	2 byte
	-2147483648 -	4 byte
		4 byte

2.3 Kiểu số thực hay còn gọi là kiểu dấu phẩy động

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
float	3.4-38E -> 3.4E+38	7-8	4 byte
double	1.7E-308 -> 1.7E+308	15-16	8 byte
laong double	3.4E-4932 -> 1.1E+4932	17-18	10 byte

3 Hằng và biến

3.1 Hằng:

- Khái niệm: hằng là giá trị bất biến trong chương trình không thay đổi, không biến đổi về mặt giá trị. Các loại hằng được sử dụng trong C tương ứng với các kiểu dữ liệu nhất định
- Trong C có ba loại hằng :
  - Hằng số
  - Hằng chuỗi
  - Hằng ký tự

Hằng số: là các giá trị số đã xác định, có thể là kiểu nguyên hay kiểu thực

- Hằng nguyên: Giá trị chỉ bao gồm các chữ số, dấu +, - được lưu trữ theo kiểu int. Ví dụ: 12,-12
- Nếu giá trị vượt quá miền giá trị của int hoặc có ký tự l (hay L) theo sau giá trị thì lưu theo kiểu long int. Ví dụ: 43L hoặc 43l là hằng nguyên lưu theo kiểu long int.
- Hằng thực: Trong giá trị có dấu chấm thập phân, hoặc ghi dưới dạng số có mũ, và được lưu theo kiểu float, double, long double. Ví dụ: 1.2 , 2.1E -3 (2.1E-3=0.0021) hoặc 3.1e-2 (3.1e-2=0.031).

Hằng ký tự

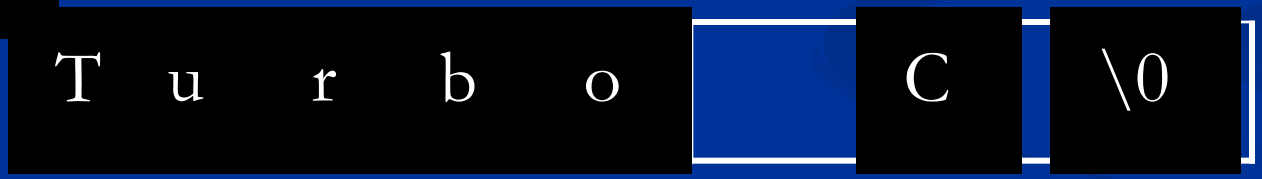
- Một hằng kiểu ký tự được viết trong dấu ngoặc đơn (') như 'A' hoặc 'z'.
- Hằng ký tự 'A' thực sự đồng nghĩa với giá trị nguyên 65, là giá trị trong bảng mã ASCII của chữ hoa 'A' (Như vậy giá trị của hằng chính là mã ASCII của nó). Đối với một vài hằng ký tự đặc biệt, ta cần sử dụng cách viết thêm dấu \ , như '\t' tương ứng với phím tab:
- Hằng ký tự có thể tham gia vào phép toán như mọi số nguyên khác:

  VD: '8' - '1'= 56-49=7.

Cách viết	Ký tự
‘\n’	Xuống hàng
‘\t’	Tab
‘\o’	“nul” tương ứng với giá trị nguyên 0 trong bảng mã ASCII
‘\b’	Backspacse
‘\r’	Về đầu dòng
‘\f’	Sang trái
‘\\’	\
‘ \” ’	”
‘ \’ ’	’

Hằng chuỗi

- Là chuỗi ký tự nằm trong cặp dấu nháy kép " ". Các ký tự này cũng có thể là các ký tự được biểu diễn bằng chuỗi thoát.
- *Ví dụ: "Turbo C", "Ngôn ngữ C++ \n\r"*
- *Một hằng chuỗi được lưu trữ tận cùng bằng một ký tự Nul (\0), ví dụ chuỗi "Turbo C" được lưu trữ trong bộ nhớ như sau:*



Cách định nghĩa hằng sử dụng trong chương trình

- Với các giá trị hằng thường được dùng trong một chương trình ta nên định nghĩa ở đầu chương trình (sau các dòng khai báo những thư viện chuẩn) theo cú pháp:

  #define <tên hằng> <giá trị>

  Ví dụ: #define PI 3.1415

3.2 Biến

- **Cách khai báo:** Mỗi biến trong chương trình đều phải được khai báo trước khi sử dụng với cú pháp khai:

Kiểu dữ liệu <danh_sách_tên_biến>;

- ***Lưu ý: nếu có nhiều tên biến thì giữa các tên biến phải có dấu , để ngăn cách***

Ví dụ: int a,b; float x;

- **Khởi đầu cho các biến**

Ngay trên dòng khai báo ta có thể gán cho biến một giá trị. Việc làm này gọi là khởi đầu cho biến.

Ví dụ: int a,b=6,d=1;

- Cách truy xuất đến địa chỉ của biến
- Một số hàm của C dùng đến địa chỉ của biến ví dụ như hàm *scanf*. Để nhận địa chỉ của biến dùng toán tử: &

Ví dụ: &tên_biến -→ &a : địa chỉ của biến a

3.3 Cấu trúc tổng quát của chương trình C

Một chương trình C chuẩn gồm có các thành phần sau:

1. Các chỉ thị tiền biên dịch
2. Khai báo các kiểu dữ liệu mới
3. Khai báo hằng, khai báo biến
4. Khai báo hàm
5. Chương trình chính

1. Chỉ thị tiền biên dịch: giúp trình biên dịch thực hiện một số công việc trước khi thực hiện một số công việc trước khi thực hiện biên dịch chính thức

   VD: #include <stdio.h>;
   #include <conio.h>;

2. Khai báo kiểu dữ liệu mới: dung từ khoá **typedef.**

   VD: typedef int songuyen;
   typedef float mang[10];

3. Khai báo hằng và biến: khai báo các hằng số và biến dùng trong chương trình
4. Khai báo hàm: khai báo các hàm tự viết
5. Chương trình chính: hàm main là hàm bắt buộc trong chương trình. Hàm main có thể trả về giá trị kiểu nguyên (int) hoặc không trả về giá trị nào (void)

```
/* Chương trình in ra dòng chữ Trung tâm đào tạo Trí Đức  trên màn hình */
# include <stdio.h>
void main () /* Ham chinh */
{
    printf(" \n Trung tâm đào tạo Trí Đức ");
    /*xuong dong in chu Trung tâm đào tạo Trí Đức */
    }
```

```
* Chương trình tính chu vi và diện tích hình tròn, biết bán kính r là một hằng số có giá trị =3.1 */
# include <stdio.h> /* khai báo thư viện hàm nhập xuất chuẩn */
# include <math.h> /* khai báo thư viện hàm toán học */
#define r 3.1
void main ()
{
    float cv,dt; /* khai bao bien chu vi va dien tich kieu so thuc */
    cv=2*r*M_PI; /* tinh chu vi */
    dt=M_PI*r*r; /* Tinh dien tich */
    printf("\nChu vi = %10.2f\nDien tich = %10.2f",cv,dt);
    /* In ket qua len man hinh */
    getch(); /* Tam dung chuong trinh */
    }
```

```
*Chương trình này minh họa cách vừa khai báo, vừa khởi đầu một biến trong C */
#include <stdio.h>
void main()
{
    char ki_tu = 'a'; /* Khai báo/khởi đầu kí tự. */
    int so_nguyen = 15; /* Khai báo khởi đầu  số nguyên */
    float so_thuc = 27.62; /* Khai báo/khởi đầu số thực/
    printf("%c la mot ki tu.\n",ki_tu);
    printf("%d la mot so nguyen.\n",so_nguyen);
    printf("%f la mot so thuc.\n",so_thuc);
    }
```

Bài 2:
Biểu thức và các phép toán

- I. Biểu thức
  - Là sự kết hợp các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó.
  - Biểu thức trong C gồm có biểu thức toán học và biểu thức logic
    - Biểu thức toán học bao gồm các phép toán số học và các hằng, các biến, các hàm
    - Biểu thức logic bao gồm các biến, hằng, hàm và phép toán logic (!: phép phủ định, &&: phép và, || : phép hoặc)

II. Các phép toán

- Phép toán số học

Phép toán	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Lấy phần dư

Chú ý:

-Phép toán chia 2 số nguyên sẽ chặt cụt phần thập phân.

-Phép toán lấy phần dư không áp dụng cho các giá trị float và double

II. Các phép toán

- Phép toán quan hệ

Phép toán	Ý nghĩa	Ví dụ
>	Có lớn hơn không?	a>b
>=	Có lớn hơn hay bằng không?	a>=b
<	Có nhỏ hơn không?	a<b
<=	Có nhỏ hơn hay bằng không?	a<=b
==	Có bằng hay không?	a==b
!=	Có khác nhau không?	a!=b

Các phép toán quan hệ có độ ưu tiên thấp hơn so với các phép toán số học

II. Các phép toán

- Phép toán logic
  - Phép phủ định !
  - Phép và (AND) &&
  - Phép hoặc (OR) ||
- Các phép toán quan hệ có độ ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với phép && và ||

- Câu lệnh gán và biểu thức

  Cú pháp của lệnh gán: **<tên biến> = <biểu thức>;**

  VD: x = -10;

  m = y + 2 – m;

  Trong C cho phép người sử dụng được gộp lệnh gán theo cú pháp :

  a = b = c = 7;

Phép toán tăng giảm

- Toán tử + + : dùng để tăng giá trị của các biến nguyên hay biến thực
- Toán tử - - dùng để giảm giá trị của biến nguyên hay biến thực
- Toán tử + + và -- đều có thể đứng trước hoặc sau toán hạng
- VD: + + n ; n+ +; --m ; m--;
- Khi các toán tử + +, -- đứng trước toán hạng thì giá trị của toán hạng được tăng hoặc giảm trước khi sử dụng và ngược lại nếu toán tử ++, -- đứng sau toán hạng thì toán hạng được tăng hay giảm sau khi thực hiện

Chuyển đổi kiểu giá trị

Việc chuyển đổi kiểu giá trị trong C thường diễn ra tự động trong trường hợp sau:

- Trong biểu thức có các toán hạng khác kiểu
- Khi gán một giá trị kiểu này cho một giá trị kiểu khác
- Ngoài ra ta có thể dùng phép chuyển kiểu để ép kiểu dữ liệu sang kiểu khác

(type) biểu_thức

Chú ý:

- Khi chuyển đổi kiểu tổng biểu thức thì đối với toán hạng có kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trược khi htực hiện phép toán và kết quả thu được sẽ có kết quả theo kiểu cao hơn

- Kiểu int và kiểu long thì int ----> long
- int và float thì int ---> float
- Float và double thì float -----> double
- Kiểu int có thể chuyển thành float và ngược lại

Bài 3
Các hàm vào ra dữ liệu

I. Hàm xuat nhap chuan trong thư viện <stdio.h>

- Hàm đưa kết quả ra màn hình

Cú pháp: **printf(<dòng điểu khiển>, bt1, bt2,...btk);**

Ý nghĩa:

bt1,...btk : là k biểu thức cần in kết quả ra màn hình

<dòng điểu khiển> là một hằng xâu ký tự bao gồm 3 loại:

- Ký tự diều khiển việc xuống dòng tiếp theo '\n'
- Ký tự hiển thị: là ký tự được in ra màn hình
- Ký tự mô tả cách đưa ra màn hình của các biến(đặc tả của kiểu). Mỗi biểu thức có đặc tả tương ứng

- Đặc tả kiểu nguyên đối với biểu thức có giá trị kiểu số nguyên **%[n]d** . Trong đó

  n là số nguyên xác định độ rộng tối thiểu dành cho giá trị biểu thức in ra màn hình

- Đặc tả kiểu số thực **%[n][.m]f.** Trong đó

  m là số chữ số sau dấu phẩy, n là một số nguyên xác định độ rộng tối thiểu trên màn hình cho giá trị của biểu thức:

  VD:

- ✓ %c : in một ký tự có mã ASCII tương ứng
- ✓ %[n]d : in một số nguyên với chiều dài tối thiểu là n
- ✓ %[n]ld: in một số nguyên (long int)
- ✓ %[n.m]f : in một số thực vối chiều dài n và lấy m số thập phân
- ✓ %s : in ra chuỗi ký tự

- Hàm hiển thị một xâu ký tự ra màn hình

Cú pháp: **int puts(char *s);**

Ý nghĩa: Hiển thị một xâu ký tự s lên màn hình, sau khi in xong thì con trỏ sẽ được chuyển xuống dòng. Trong đó s là con trỏ kiểu char trỏ tới vùng chứa xâu ký tự

- Hàm đưa một ký tự ra màn hình

Cú pháp: **int putchar(int ch);**

Ý nghĩa: Hàm sẽ ký tự ch ra màn hình với ch là mã của lý tự cần in

- Hàm nhận dữ liệu từ bàn phím

Cú pháp: **scanf("dt1dt2..dtk", &biến1, ...&biếnk);**

Ý nghĩa:

- dt1,...dtk là một hằng xâu ký tự đặc tả của k biến
- &biến1, ..&biếnk: là địa chỉcủa biến trong bộ nhớ

- Hàm nhận từ bàn phím một xâu ký tự

Cú pháp: **int *getchar(char *s);**

Ý nghĩa:

- Hàm nhận dãy ký tự từ bàn phím vào cho đến khi gặp ký tự '\n' thì dừng lại
- s là con trỏ trỏ tới vùng nhớ sẽ chứa xâu vừa nhận

- Hàm nhận một ký tự từ bàn phím

Cú pháp: **getchar(void);**

Ý nghĩa: nhận ký tự được nhập từ bàn phím

II. Hàm xuất nhập chuẩn trong thư viện <conio.h>

- Hàm getch() và getche

Cú pháp : ***int getch(void)***

int getche(void)

- Hai hàm trên chờ nhận một ký tự trực tiếp từ bộ đệm bàn phím. Nếu bộ đệm rỗng thì chờ. Khi một phím được ấn thì nhận ngay ký tự đó mà không cần phải enter như các hàm nhập từ stdio.h
- Hàm getche() cho hiện ký tự lên màn hình còn getch() thì không
- Kết quả trả về của hàm là ký tự được ấn trên bàn phím.

- Xuất ký tự có màu

Cú pháp : ***cprintf***

Ý nghĩa: in ra ký tự có màu được ấn định bởi hàm textcolor.

- Nhập ký tự có màu

Cú pháp: **cscanf**

Ý nghĩa:

+ Nội dung nhập có màu được ấn định bởi hàm textcolor

+ Nhận nội dung trực tiếp từ bộ đệm bàn phím. Vì vậy với hàm cscanf ta cũng phải khử ký tự \n trong bộ đệm bằng %*c hoặc bằng hàm getch()

III.Một số hàm thao tác trên màn hình

- *Hàm xóa màn hình:* **clrscr();**

  Có tác dụng xóa toàn bộ m àn hình và sau khi xóa con trỏ sẽ ở vị trí góc phía bên trái.

- *Hàm đặt tọa độ con trỏ:* **gotoxy(int x, int y);**

  Đặt con trỏ tại vị trí x, y

- *Hàm đặt mầu nền textbackground*

  void textbackground(int color);

  Đặt mầu nền Color là một biểu thức nguyên có giá trị từ 0 đến 7 tương ứng với một trong 8 hằng số màu đầu tiên của bảng mầu văn bản.

- *Hàm đặt mầu chữ textcolor*

void textcolor(int newColor);

Lựa chọn màu ký tự mới newColor.Trong đó newColor là một biểu thức nguyên có giá trị từ 0 đến 15 tương ứng với một trong các hằng số màu của bảng mầu văn bản.

Bài 4: Cấu trúc điều khiển

- Câu lệnh, khối lệnh
  - Câu lệnh: mỗi câu lệnh thực hiện một công việc và được kết thúc bởi dấu ;
  - Khối lệnh: là tập hợp các câu lệnh bắt đầu bằng dấu “{“ và kết thúc bằng dấu “}”

I. Cấu trúc điều khiển if

1. Cấu trúc if dạng 1

Cú pháp: **if (bt) s ;**

Ý nghĩa: bt là biểu thức lôgic, s là lệnh đơn hoặc lệnh phức. nếu bt nhân giá trị true thì thực hiện s, ngược lại s được bỏ qua

2. Cấu trúc if dạng 2

Cú pháp:

```
if(bt)      s;
else        s1 ;
```

Ý nghĩa: bt là biểu thức lôgic, nếu bt nhận giá trị true thì thực hiện s bỏ qua s1, ngược lại nếu bt nhận giá trị flase thì thực hiện s1 bỏ qua s(s và s1 có thể là lệnh đơn hoặc lệnh phức)

- Chú ý : trong C cho phép sử dụng các cấu trúc if lồng nhau để giải quyết bài toán

3. Bài tập

- Nhập 2 số thực a, b từ bàn phím. Tìm và in ra màn hình số lớn nhất và số bé nhất
- Giải hệ phương trình bậc nhất hai ẩn số

$$ax + by = c$$

$$dx + ey = f$$

II. Cấu trúc rẽ nhánh switch

1. Cấu trúc tổng quát

Cú pháp:

```
switch (bt)
{       case n1 : s1
        case n2 : s2
                ....
        case nk : sk
        [default : s(k+1 )]
}
```

Ý nghĩa:

- Bt: là biểu thức toán học có giá trị kiểu nguyên
- Ni(i=1..k): là các số kiểu nguyên, kiểu hằng ký tự, hoặc biểu thức
- Si(i=1..k): là các lệnh đơn hoặc lệnh phức
- [default : s(k+1)] : là phần tuỳ chọn có thể có hoặc không

Hoạt động: lệnh switch phụ thuộc vào giá trị của biểu thức bt viết sau switch, nếu:

- Giá trị bt = ni thì thực hiện câu lệnh sau case ni;
- Khi giá trị biểu thức khác tất cả các ni thì thực hiện câu lệnh sau default nếu có, hoặc thoát khỏi câu lệnh switch.
- Khi chương trình đã thực hiện xong câu lệnh của case ni nào đó thì nó sẽ thực hiện luôn các lệnh thuộc case bên dưới nó mà không xét lại điều kiện (do các ni còn được xem như các nhãn). Vì vậy, để chương trình thoát khỏi lệnh switch sau khi thực hiện xong một trường hợp, ta dùng lệnh break.

3. Bài tập

- Viết chương trình nhập vào từ bàn phím một mã số nguyên và đưa ra đánh gía trình độ theo yêu cầu:
  - 1: trình độ sơ cấp
  - 2 : trình độ trung cấp
  - 3: trình độ Đại học
  - 4: trình độ Cao học
  - 5: trình độ Tiến sỹ
  - Các số khác: Không xác định
- Cho một số tự nhiên, in ra màn hình tên gọi của số lên màn hình(Bài số 12)

III. Câu lệnh lặp for

1. Cú pháp:

for(<bt1> ; <bt2> ; <bt3>) S ;

Ý nghĩa:

- S là lệnh đơn hoặc lệnh phức
- bt1 : thường là một lệnh gán khởi tạo cho biến điều khiển
- bt2: là biểu thức logic, giá trị của biểu thức lôgic này quyết định vòng lặp tiếp tục hay kết thúc
- bt3: thường là lệnh gán có tác dụng làm thay đổi giá trị của biến điều khiển

Hoạt động:

- Bước 1: Thực hiện bt1
- Bước 2: Tính toán, xác định giá trị của bt2
- Bước 3: Nếu bt2 có giá trị false thì thoát khỏi vòng lặp. Ngược lại bt2 có giá trị true thì s được thực hiện
- Bước 4: sau khi thực hiện s thực hiện bt3 và quay lại bước 2

Nhận xét:

- <t1> chỉ được thực hiện duy nhất một lần khi bắt đầu vòng lặp
- <bt2>, <bt3> và S có thể được tính toán và thực hiện lặp nhiều lần

Chú ý khi sử dụng vòng lặp for

- <bt1>, <bt2>, <bt3> đều có thể vắng mặt nhưng vẫn phải giữ lại dấu (;)
- Trường hợp đặc biệt <bt2> không có thì luôn được xem là nhận giá trị true, muốn thoát khỏi vòng lặp phải dùng lệnh break, goto hoặc return
- Có thể dùng cấu trúc các vòng for lồng nhau
- Khi gặp lệnh break thì chương trình sẽ thoát khỏi vòng for sâu nhất còn chứa lệnh break
- Trong vòng for có thể sử dụng lệnh continue để chuyển tới chu trình mới của vòng lặp

2. Bài tập

- Viết chương trình tính tổng của n số đầu tiên của dãy số sau:

$$S = 1+1/2+1/3+1/4+...+1/n.$$

- Viết chương trình tìm tất cả các số nguyên có ba chữ số sao cho tổng tam thừa của ba chữ số hàng trăm, hàng chục, hàng đơn vị sẽ bằng số nguyên đó. Ví dụ: $1^3+5^3+3^3=153$

IV. Câu lệnh while

1. Cú pháp :

While (bt)

S;

Ý nghĩa: bt là biểu thức lôgic, S là một lệnh hoặc một dãy lệnh

Hoạt động

- Xác định giá trị của bt. Nếu giá trị của bt= true(<>0) thì chuyển sang bước 2, ngược lại thì thoát khỏi vòng lặp
- Thực hiện S sau đó quay về bước 1(Lệnh S có thể được thực hiện nhiều lần hoặc không được thực hiện lần nào nếu bt =false ngay từ đầu)

Chú ý : trong câu lệnh lặp while ta có thể dùng câu lệnh break để thoát khỏi vòng lặp theo ý muốn

2. Bài tập :

- Nhập hai số nguyên từ bàn phím, tìm và in ra màn hình ước số chung lơn nhất của hai số
- Tìm hình chữ nhật có diện tích lớn nhất khi biết chu vi của nó(bài số 25)

V. Câu lệnh do.. while

1. Cú pháp

do S

while (bt);

Ý nghĩa: S là một câu lệnh đơn hoặc phức, bt là biểu thức lôgic

Hoạt động:

- (1) Thực hiện lệnh S
- (2) Xác định giá trị của bt. Nếu giá trị của bt = true thì chuyển sang bước (1), ngược lại thì thoát khỏi vòng lặp
- Lệnh S luôn được thực hiện ít nhất 1 lần trong câu lệnh

Bài 5: Dữ liệu kiểu mảng

1. Khái niệm:

- Mảng được hiểu là một tập hợp các giá trị có cùng kiểu dữ liệu nằm liên tiếp nhau trong bộ nhớ máy tính
- Mảng được coi như một biến mảng và tên mảng được đặt theo quy tắc đặt tên biến
- Mảng có những thành phần sau:
  - Kiểu dữ liệu của các phần tử trong mảng
  - Tên mảng
  - Số chiều và kích thước của mỗi chiều

2. Cách khai báo biến mảng

<kiểu_dl> <tên_mảng><ds các chiều của mảng>

VD:

```
int A[10];
//mảng 1 chiều A gồm 10 phần tử kiểu số nguyên
float B[2] [3];
// Mảng 2 chiều B gồm 2 hàng và 3 cột, các phần tử có kiểu số thực
```

3. Cách tổ chức và truy xuất đến phần tử mảng

- Phần tử của mảng được xác định thông qua chỉ số. Chỉ số của phần tử trong mảng luôn là một số nguyên không vượt qua kích thước của mảng
- Các phần tử của mảng được sắp xếp liền nhau trong bộ nhớ của máy tính và chỉ cho phép truy cập đến địa chỉ trực tiếp của phần tử đối với mảng một chiều. Cách truy cập theo địa chỉ

&tên_biến[i]

trong đó i là chỉ số của phần tử

- VD: a= &a[0]

//Tên mảng chỉ tới địa chỉ phần tử đầu tiên của mảng

4. Cách xuất nhập dữ liệu trên mảng

- Nhập xuất trực tiếp ứng dụng cho mảng một chiều và mảng hai chiều có phần tử kiểu int thông qua địa chỉ
- Nhập dữ liệu cho mảng

```
for( i=0;i<5;i++) {
        printf("Phan tu thu %d= ",i);
        scanf("%d", &a[i]);
    }
```

- In các phần tử của mảng ra màn hình

```
for(i=0;i<n;i++)  printf("%6d",a[i])
```

- Nhập xuất dữ liệu gián tiếp thông qua một biến trung gian đối với mảng một chiều và mảng đa chiều

```
for(i=0;i<2;i++)
    for(j=0;j<3;j++) {
        printf("a[%d,%d]", i, j);
        scantf("%f",&temp);
        a[i] [j] = temp;
    }
```

- Bài tập
- Nhập vào từ bàn phím n số nguyên, tìm và in ra màn hình số nguyên lớn nhất và số nguyên nhỏ nhất
- Nhập ma trận các số thực kích thước n hàng và m cột. Tìm và in ra số thực lớn nhất trong ma trận

Bài 6: Con trỏ

1. Khái niệm con trỏ và địa chỉ

- Địa chỉ: Dựa vào kiểu dữ liệu khi khai báo biến máy sẽ cấp phát cho biến một địa chỉ để lưu trữ biến đó trên vùng nhớ. Mỗi biến có kiểu khác nhau thì được lưu vào các địa chỉ khác nhau

- Con trỏ là một biến dùng để chứa địa chỉ. Mỗi loại địa chỉ thì có loại con trỏ tương ứng. Trước khi sử dụng biến con trỏ ta phải khai báo trước khi sử dụng

- Khai báo: <kiểu_DL> * <tên_biến_con_trỏ>;

- VD1: int x, y, *p, *c;

x, y là hai biến kiểu nguyên, p, c là hai biến con trỏ kiểu nguyên

VD2: float *t, *d ;

//Khai báo biến con trỏ t và d có kiểu thực

Biến con trỏ được dùng theo hai trường hợp sau:

- Tên con trỏ chỉ đến địa chỉ của biến được lưu trong con trỏ:

```
float a,*p,*q;
p=&a; /* lưu địa chỉ của biến a vào con trỏ p */
q= p; /* lưu địa chỉ trong p vào con trỏ q*/
```

- Dạng khai báo của con trỏ chỉ đến giá trị lưu tại vùng nhớ mà con trỏ trỏ tới.

VD: *float x=5, y , z=20, *px, *pz;*py;*

px=& x; / khi đó *px = x =5*/*

pz=&z; / *pz=z=20*/*

khi đó ba biểu thức sau là tương đương:

*y=3*x+z;* **py=3*x+z;* **py=3*(*px)+*pz;*

2. Con trỏ và mảng một chiều

Các phần tử của mảng có thể được xác định thông qua con trỏ. Ta có khai báo : float a[10];

//Khai báo mảng gồm 10 phần tử kiểu thực

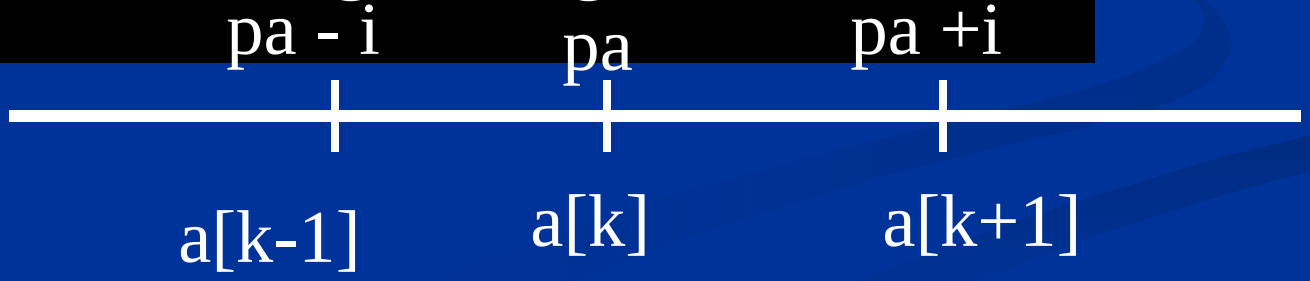
Ta có tên mảng chính là một hằng địa chỉ trỏ tới địa chỉ phần tử đầu tiên của mảng và

a tương đương với &a[0]

a+i tương đương với &a[i]

*(a+i) tương đương với a[i]

Vậy



- Các cách viết a[i], *(a+i), *(p+i), p[i] là tương đương nhau
- VD: Nhập từ bàn phím các phần tử của mảng và tính tổng các phần tử đó

```
#include<stdio.h>
#include<stdio.h>
void main()
{       float a[5], s ; int i;
        for(i=0;i<5;i++) {
                printf("\na[%d]= ",i); scanf("%f",&a[i]); }
        s=0;
   for (i=0;i<5;i++)          s+=a[i];
   printf("\n Tong =%8.2f",s);
   getch();
}
```

Ví dụ: Tro1

3. Con trỏ với mảng nhiều chiều

Phép toán lấy địa chỉ nói chung không dùng được đối với các thành phần của mảng nhiều chiều (trừ trường hợp mảng hai chiều các số nguyên).

Để tính toán địa chỉ của thành phần a[i][j] chúng ta sử dụng công thức sau :

(float *)a+i*n+j.

a là một hằng con trỏ trỏ đến các dòng của một ma trân hai chiều, vì vậy

a trỏ đến dòng thứ nhất

a+1 trỏ đến dòng thứ hai

a+2 trỏ đến dòng thứ ba

- Để tính toán được địa chỉ của phần tử ở dòng i cột j chúng ta phải dùng phép chuyển đổi kiểu bắt buộc đối với a: (float *)a

  a là con trỏ trỏ đến thành phần a[0][0] của ma trận.

  a[i][j] sẽ có địa chỉ là (float *a) +i*n+j

  Xét VD nhập giá trị của ma trận hai chiều: Tro2

```
#include <stdio.h>
#include <stdio.h>
void main()
{       float a[10][20];        int i,j,n;
        printf("Nhap vao kich thuoc ma tran n=");
        scanf("%n",&n);
   for(i=0;i<n;i++)
        for(j=0;j<n;j++)
   {    printf("a[%d][%d] = ",i,j);
        scanf("%f",(float *)a+i*20+j);
        }
   getch();
}
```

4. Phép toán trên con trỏ

- Phép gán: chỉ nên thực hiện trên các con trỏ có cùng kiểu, khi thực hiện trên con trỏ phải thực hiện phép ép kiểu:

  Vd: int x;

  char *p;

  p=(char*)(&x);

- Phép tăng giảm địa chỉ

  VD: float x[30], *px;

  px=&x[10];// p là con trỏ thực trỏ tới phần tử x[10]

  px+i trỏ tới phần tử x[10+i]

  px – i trỏ tới phần tử x[10-i]

- Phép so sánh: dùng so sánh các con trỏ cùng kiểu, giả sử p1 và p2 là hai con trỏ kiểu float thì tồn tại phép so sánh

  p1 < p2 // địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới

  p1==p2

5. Con trỏ kiểu void

   Là con trỏ đặc biệt không có kiểu, nó có thể nhận bất kỳ địa chỉ nào. Con trỏ kiểu void thường dùng làm đối để nhận bất kỳ địa chỉ nào thông qua phép ép kiểu trong thân hàm

   Các phép toán tăng giảm địa chỉ, so sánh không dùng được con trỏ kiểu void

6. Mảng con trỏ

Mảng con trỏ là một mảng mà mỗi phần tử của nó có thể chứa một địa chỉ nào đó. Mảng con trỏ có nhiều kiểu, mỗi phần tử của mảng kiểu nào thì sẽ chứa địa chỉ kiểu tương ứng với nó. Mảng con trỏ được khai báo theo mẫu sau:

<kiểu Dl> *<tênmảng>[N]

Khi gặp khai báo mảng con trỏ thì máy sẽ cấp phát N khoảng nhớ liên tiếp cho N phần tử tương ứng trong mảng

Chú ý: Mảng con trỏ không dùng để lưu số liệu, trước khi sử dụng mảng con trỏ cần gán cho mỗi phần tử một giá trị là địa chỉ của một biến hoặc của một phần tử trong mảng

Bài 7: Hàm và chương trình

1. Khái niệm

Chương trình: Một chương trình C bao gồm một hoặc nhiều hàm. Hàm main() là thành phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện từ câu lệnh đầu tiên của hàm main( ) cho đến khi gặp dấu } cuối cùng của hàm này.

Hàm: Là một đoạn chương trình độc lập thực hiện trọn vẹn một công việc rồi trả về một giá trị cho chương trình đã gọi nó.

Đặc điểm của hàm:

- Là một đơn vị độc lập của chương trình.
- Không cho phép xây dựng một hàm bên trong một hàm khác.

2. Quy tắc xây dựng hàm: Một hàm gồm có các thành phần sau

- **Nguyên mẫu của hàm:** Bao gồm

  *<kiểu dl của hàm> <**tên hàm**(ds các tham số)>;*

  Có thể có hoặc không khai báo nguyên mẫu của hàm, khi không khai báo nguyên mẫu thì bộ biên dịch sẽ kiểm tra việc truyền tham số, giá trị trả về có phù hợp hay không rồi mới cho thực hiện hàm.

  Tất cả nguyên mẫu của các hàm có trong chương trình nên đặt trước hàm main().

- **Kiểu giá trị của hàm**

  Giá trị trả về của hàm được xác định dựa vào mục đích của hàm. Nếu các hàm không trả về giá trị ta phải khai báo kiểu void.
- **Tên hàm**

  Đặt theo qui định đối với danh định. Tên hàm trong nguyên mẫu và khi khai báo phải giống nhau.
- **Tham số của hàm**

  Khi viết một hàm ta phải xác định xem hàm có bao nhiêu tham số ?
- **Nội dung của hàm**

Cấu trúc của một hàm

```
<Kiểu trả về><Tên hàm>(<ds tham số hình thức hay đối số>)
{       <Khai báo biến cục bộ>;
        <Các câu lệnh trong thân hàm>;
        [return<bt trả về giá trị hàm>];
   };
```

Chú ý:

- Đối với các hàm không có kiểu trả về ta có hàm kiểu void
- Hàm không có đối thì dùng kiểu void để khai báo đối. VD

```
void bell(void)
{    int i;
     for(i=0;i<10;i++) putch(7);
     }
```

- Cách sử dụng hàm: Hàm được sử dụng thông qua lời gọi hàm.

  <tên hàm> ([ds tham số thực])

- Tham số thực phải bằng tham số hình thức
- Kiểu của tham số thực phải phù hợp với kiểu của tham số hình thức

- Hoạt động của hàm khi có lời gọi hàm
  - Cấp phát bộ nhớ cho tham số hình thức và biến cục bộ
  - Gán giá trị của tham số thực cho tham số hình thức
  - Thực hiện các lệnh trong thân hàm
  - Khi gặp câu lệnh return hoặc dấu hiệu kết thúc hàm thì bộ nhớ sẽ xoá các tham số hình thức và biến cục bộ sau đó thoát khỏi hàm quay về chương trình gọi hàm

3. Các tham số trong hàm

3.1 Phân loại tham số theo cách sử dụng

- Tham số hình thức: Các tham số mà ta ghi trong nguyên mẫu hay ghi lúc khai báo hàm gọi là tham số hình thức.
- Tham số thực:Các giá trị, biến mà ta ghi sau tên hàm khi gọi hàm đó để thực hiện gọi là tham số thực. Trong C, các tham số thực lại chia ra làm hai loại:

- Tham chiếu: Là các tham số thực mà ta truyền cho Hàm dưới dạng con trỏ (dạng địa chỉ). Tham chiếu mới ghi nhận lại được những kết quả vừa tính toán trong Hàm khi Hàm kết thúc.
- Tham trị : Là các tham số thực mà ta truyền cho Hàm dưới dạng biến. Tham trị không bảo lưu lại những kết quả thay đổi của nó được tính toán trong Hàm khi Hàm kết thúc.

3.2 Phân loại theo công dụng

- Tham số của một hàm có hai công dụng:
  - Cung cấp các giá trị cho hàm khi ta gọi nó thực hiện .
  - Lưu các kết quả tính toán được trong quá trình hàm hoạt động
  - Tương ứng với công dụng ta có các loại tham số:
    - Tham số vào: Cung cấp giá trị cho hàm.
    - Tham số ra: Lưu kết quả tính toán được trong hàm.
    - Tham số vừa vào, vừa ra: vừa cung cấp giá trị cho hàm, vừa lưu kết quả tính toán được trong hàm.

4. Hàm có đối con trỏ

Đối số của hàm là con trỏ kiểu *int (float,double,.)* thì tham số thực tương ứng phải là địa chỉ của biến kiểu *int (float,double,.)*. Khi đó địa chỉ của biến được truyền cho đối con trỏ tương ứng.

Khi muốn bảo lưu lại kết quả tính toán được của các đối số trong hàm để sử dụng cho chương trình gọi hàm có đối số thì chúng ta phải khai báo đối số của hàm là tham chiếu (con trỏ hay dạng địa chỉ).

VD:

Bài 8: Chuỗi ký tự

1. Khái niệm

- Chuỗi ký tự là một dãy các ký tự đặt trong cặp dấu nháy kép. Chuỗi rỗng được ký hiệu bằng hai dấu nháy kép đi liền nhau. Một chuỗi ký tự được cấp phát một khoảng nhớ cho một mảng kiểu char chứa các ký tự của chuỗi và chứa thêm ký tự '\0' là ký tự kết thúc chuỗi.
- Mỗi ký tự của chuỗi được chứa trong một phần tử của mảng. Chuỗi ký tự là một trường hợp riêng của mảng một chiều khi mỗi thành phần của mảng là ký tự
- Chuỗi ký tự thường được khai báo theo khai báo theo hai mẫu:

*char ten_chuoi[] ; hoặc char *ten_chuoi;*

2. Các thao tác trên chuỗi

Trong C không tồn tại các phép toán so sánh, gán nội dung của chuỗi này cho chuỗi khác.

Để thực hiện các thao tác này ta sử dụng một thư viện các hàm chuẩn là <**string.h**>.

- **Hàm strlen:** int strlen(char s[])

  Trả về độ dài của chuỗi s, chính là chỉ số của ký tự NULL trong chuỗi.

- **Hàm strcpy:** strcpy(char dest[], char source[])

  Sao chép nội dung chuỗi source vào chuỗi dest.

- **Hàm strchr:** char *strchr(char s[], char c)

  Tìm lần xuất hiện đầu tiên của ký tự c trong chuỗi s, trả về địa chỉ của ký tự này.

- **Hàm strncpy:** strncpy(char dest[], char source[], int n)

  Sao chép n ký tự trong chuỗi source vào chuỗi dest. Trong trường hợp không có đủ n ký tự trong source thì hàm sẽ điền thêm các ký tự trắng vào chuỗi dest.
- **Hàm strcat :** strcat(char ch1[], char ch2[])

  Nối chuỗi ch2 vào cuối chuỗi ch1. Sau lời gọi hàm này độ dài chuỗi ch1 bằng tổng độ dài của cả hai chuỗi ch1 và ch2 trước lời gọi hàm.
- **Hàm strncat :** strncat(char ch1[], char ch2[],int n)

  Nối n ký tự đầu tiên của ch2 vào ch1

- **Hàm strstr :** char *strstr(char s1[], char s2[])

  Tìm kiếm chuỗi s2 trong chuỗi s1, Trả về địa chỉ của lần xuất hiện đầu tiên của s2 trong s1 hoặc NULL khi không tìm thấy.
- **Hàm strcmp :** int strcmp(char ch1[], char ch2[])

  So sánh hai chuỗi ch1 và ch2. Nguyên tắc so sánh theo kiểu từ điển. Giá trị trả về:
  - = 0 nếu chuỗi ch1 bằng chuỗi ch2
  - > 0 nếu chuỗi ch1 lớn hơn chuỗi ch2
  - < 0 nếu chuỗi ch1 nhỏ hơn chuỗi ch2

  VD: Đếm số lần xuất hiện của ký tự a trong một xâu ký tự cho trước

```
#include<stdio.h>
#include<conio.h>
#define HANG 128
void main()
{  char xau[HANG];      int i,na;
   clrscr();
   printf("\nNhap mot xau ky tu:");gets(xau);
   na=i=0;
   while(xau[i])
        if (xau[i++]=='a') na++;
        printf("\nXau co %d chu  a",na);
        getch();
   return;
}
```

3. Mảng và chuỗi ký tự

- Một dạng sử dụng con trỏ đặc biệt là việc sử dụng một mảng các biến con trỏ. Khai báo theo mẫu

*type *pointer_array[size];*

- VD: khai báo *char *temp[10];*

sẽ khai báo một mảng 10 con trỏ *char* có thể được dùng để khai báo một mảng để lưu trữ địa chỉ của mười chuỗi ký tự nào đó.

- Bài tập:viết chương trình nhập nhiều tên người vào từ bàn phím, sắp xếp lại theo thứ tự và in kết quả đã sắp xếp ra.

+ Sắp xếp lại các tên này theo thứ tự alphabet

+ In các tên ra theo thứ tự đó.

- VD: xét một mảng các con trỏ ptr_array được gán các địa chỉ của các biến int có giá trị và vị trí bất kỳ. Dùng một hàm để sắp xếp lại các địa chỉ này trong mảng để sao cho các địa chỉ của các số bé được xếp trước địa chỉ của các số lớn hơn. Lúc đó dù chúng ta không làm thay đổi vị trí hoặc thay đổi các giá trị của các biến nhưng mảng vẫn giống như một mảng chỉ đến các giá trị đã sắp xếp có thứ tự.

Bài tập:

- Viết chương trình đếm số lần xuất hiện của một ký tự trong một xâu ký tự
- Viết chương trình nhập một chữ, xuất ra chữ đó nhiều lần dùng con trỏ

Bài 9: Cấp phát và giải phóng bộ nhớ động

1. Khái niệm

- **Biến động:** Là các biến *được tạo ra lúc chạy chương trình*, tùy theo nhu cầu. Số biến này hoàn toàn không được xác định từ trước. Các biến động không có tên (việc đặt tên thực chất là gán cho nó một địa chỉ xác định).
- **Cách tạo ra biến động và truy nhập đến biến động được tiến hành như sau**

Việc tạo ra biến động và xóa nó đi (để thu hồi lại bộ nhớ) được thực hiện nhờ các hàm như malloc() và free() đã có sẵn trong thư viện stdlib.h

- Việc truy nhập đến biến động được tiến hành nhờ các biến con trỏ. Các biến con trỏ được định nghĩa như các biến tĩnh (được khai báo ngay từ đầu trong phần khai báo biến) và được dùng để chứa địa chỉ các biến động
- *VD1:* *int *p; /* Khai báo biến con trỏ p*/*

  *p= (int *) malloc(100);/* Tạo biến động*/*

Đoạn chương trình trên sẽ cấp phát 100 bytes trong bộ nhớ và gán địa chỉ khối bộ nhớ này cho p

- *VD2: cấp phát bộ nhớ chính xác cho 70 ký tự:*

```
/* Khai báo biến con trỏ kiểu char */
    char *cp;
/* Tạo biến động */
    cp=(char *) malloc(70);
```

2. Cấp phát và giải phóng bộ nhớ động (các hàm thuộc stdlib.h và alloc.h)

2.1 Cấp phát bộ nhớ động bằng hàm malloc( )

Cú pháp **void *malloc(kiểu _dl size)**

- Chức năng: Hàm malloc cấp phát một vùng nhớ có kích thước là size.
- size là một giá trị kiểu_dl (là một kiểu dữ liệu định sẵn trong thư viện stdlib.h).
- Hàm malloc trả về con trỏ kiểu void chứa địa chỉ ô nhớ đầu của vùng nhớ được cấp phát. Nếu không đủ vùng nhớ để cấp phát hàm trả về giá trị **NULL**, vì vậy phải kiểm tra giá trị trả về khi sử dụng hàm malloc.

2.2 Cấp phát bộ nhớ động bằng hàm calloc

Cú pháp

(datatype *) calloc(n, sizeof(object));

Hàm calloc cấp phát bộ nhớ động cho các kiểu dữ liệu
Trong đ ó:

- (datatype *) là kiểu con trỏ trỏ tới kiểu dữ liệu datatype.
- n là số lượng object thuộc kiểu datatype cần cấp phát bộ nhớ.
- datatype có thể là kiểu dữ liệu cơ sở hoặc kiểu dữ liệu mới

2.3 Cấp phát bộ nhớ động bằng hàm relloc

Cú pháp

(datatype *) realloc(buf _p, newsize);

Hàm có chức năng cấp phát lại bộ nhớ

Trong đó:

- buf_p là con trỏ đang trỏ đến vùng ô nhớ đã được cấp phát từ trước.
- newsize là kích thước mới cần cấp phát, có thể lớn hoặc nhỏ hơn.

2.4 Giải phóng bộ nhớ bằng hàm free

Cú pháp

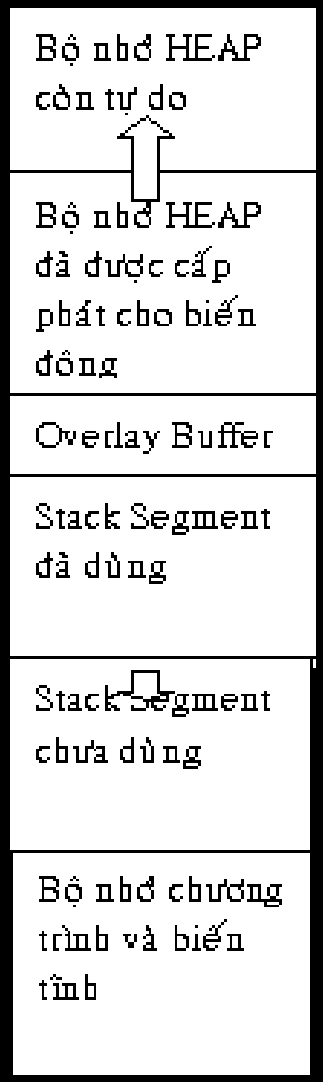
void free(void *prt)

Hàm free giải phóng vùng nhớ được trỏ đến bởi con trỏ ptr.

Nếu con trỏ ptr = NULL thì hàm free không làm gì cả.

3. Bộ nhớ HEAP và cơ chế tạo biến động

- Các biến động do malloc tạo ra được C xếp vào một vùng ô nhớ tự do theo kiểu xếp chồng và được gọi là HEAP (bộ nhớ cấp phát động). Ngôn ngữ C quản lý HEAP thông qua một con trỏ của HEAP là HEAPPTR. Nó luôn trỏ vào byte tự do đầu tiên của vùng ô nhớ còn tự do của HEAP. Mỗi lần gọi malloc(), con trỏ của HEAP được dịch chuyển về phía đỉnh của vùng ô nhớ tự do một số byte tương ứng với kích thước của biến động mới tạo ra.
- Ngược lại, mỗi khi giải phóng bộ nhớ biến động, bộ nhớ biến động được thu hồi



Hình ảnh khái quát việc sử dụng bộ nhớ và Heap

Bài 10: Kiểu cấu trúc

1. Kiểu enum

- Câu lệnh khai báo kiểu enum có thể viết theo bốn cách

```
enum tk {pt1,pt2,...} tb1,tb2,...;
enum  tk {pt1,pt2,...};
enum  {pt1,pt2,...} tb1,tb2,...;
enum  {pt1,pt2,...};
```

- Trong đó :
  - Tk là tên kiểu enum (một kiểu dữ liệu mới),
  - pt1,pt2,... là tên các phần tử,
  - tb1,tb2,... là tên biến kiểu enum.

- Ví dụ: khai báo kiểu dữ liệu làm việc với các ngày trong tuần ta có thể dùng kiểu weekday và biến day như sau:

  enum weekday{SUNDAY,MONDAY,TUESDAY,WEDSDAY,THURSDAY, FRIDAY, SATURDAY} day;

- Chú ý biến kiểu enum thực chất là biến nguyên, nó được cấp phát 2 byte bộ nhớ và nó có thể nhận một giá trị nguyên bất kỳ.

2. Kiểu cấu trúc

2.1 Định nghĩa

Cấu trúc là một kiểu dữ liệu bao gồm nhiều thành phần có thể thuộc nhiều kiểu dữ liệu khác nhau. Các thành phần được truy nhập thông qua một tên

2.2 Cú pháp tổng quát

```
struct [tên_cấu_trúc]
{
  khai báo các thành phần
  } [danh sách các biến cấu trúc];
```

trong đó:

- *struct* là từ khóa đứng trước một khai báo cấu trúc,
- *tên_cấu_trúc* là một tên hợp lệ được dùng làm tên cấu trúc;
- *[danh sách các biến cấu trúc]* liệt kê các biến có kiểu cấu trúc vừa khai báo
- ***VD:***

```
struct hoc_sinh {
char ho_ten[20];
float diem;
} hs,        dshs[100];
```

2.3 **Cú pháp định nghĩa kiểu dữ liệu mới**

Ngôn ngũ C cho phép ta đặt lại tên kiểu dữ liệu mới bằng câu lệnh:

typedef kiểu_đã_có tên_kiểu_mới;

trong đó :

- *kiểu_đã_có là kiểu dữ liệu mà ta muốn đổi tên.*
- *tên_kiểu_mới là tên mới mà ta muốn đặt.*

2.4 Nguyên tắc truy cập đến thành phần của cấu trúc

- Các thành phần của cấu trúc được truy nhập thông qua tên biến cấu trúc và tên thành phần.

  tên_biến_cấu_trúc.tên_thành_phần

  Để truy nhập đến các thành phần của biến hs chúng ta viết như sau:

  hs.ho_ten

  hs.diem

  Chú ý:Không nên sử dụng toán tử & đối với các thành phần cấu trúc (đặc biệt đối với các thành phần không nguyên) trong khi nhập dữ liệu

2.5 Con trỏ cấu trúc

Cách khai báo

- Một biến cấu trúc cũng là một biến trong bộ nhớ, có thể lấy địa chỉ của một biến cấu trúc bằng toán tử lấy địa chỉ &. Giá trị trả lại là địa chỉ đến trường đầu của cấu trúc.
- Có thể khai báo một biến con trỏ chỉ đến một cấu trúc để có thể lưu địa chỉ của một biến cấu trúc nào đó. Cú pháp khai báo một biến con trỏ cấu trúc như sau:

*struct tên_cấu_trúc *tên_con_trỏ;*

*VD: struct hoc_sinh *ptrhs;*

- Việc truy xuất đến một thành phần của cấu trúc thông qua một con trỏ được thực hiện bằng phép toán kép ->

VD:

```
printf("\nHo va ten hoc sinh %s",ptrhs->ho_ten);
printf("\nDiem %6.3f",ptrhs->diem);
```

kết quả thực hiện hai câu lệnh này tương đương với hai câu lệnh sau:

```
printf("\nHo va ten hoc sinh %s",hs.ho_ten);
printf("\nDiem %6.3f",hs.diem);
```

- Việc sử dụng con trỏ chỉ đến cấu trúc thường được sử dụng để truyền cấu trúc đến cho một hàm
- Một ứng dụng khác của con trỏ cấu trúc là dùng để xây dựng các cấu trúc tự trỏ như: danh sách liên kết (còn gọi là danh sách móc nối).

2.6 Mảng có cấu trúc

Mảng mà gồm các thành phần có kiểu cấu trúc được gọi là mảng cấu trúc. Khai báo một mảng các cấu trúc hoàn toàn tương tự như đối với khai báo một mảng bình thường, chỉ có một điểm khác là thay cho tên các kiểu dữ liệu bình thường là một tên kiểu dữ liệu cấu trúc.

Ví dụ về khai báo một mảng có cấu trúc:

struct hoc_sinh dshs[100]; //hoc_sinh là kiểu cấu trúc

Việc sử dụng các mảng cấu trúc sẽ làm cho việc xử lý một tập hợp các biến cấu trúc trở nên dễ nhìn hơn. Các quy định về mảng cũng được áp dụng đối với mảng các cấu trúc.

3. Cấu trúc tự trỏ

Cấu trúc có ít nhất một thành phần là con trỏ chỉ đến bản thân cấu trúc được gọi là cấu trúc tự trỏ.

Ví dụ

```
struct h_sinh{
char ho_ten[20];
float diem;
}
struct h_sinh *next ;/*con trỏ chỉ đến học sinh tiếp theo trong danh sách*/
```

khai báo này định nghĩa một cấu trúc tự trỏ có thể dùng để quản lý danh sách họ tên học sinh và điểm số học sinh. Danh sách này chỉ truy cập được theo một chiều

3.1 Danh sách liên kết: danh sách liên kết gồm các phần tử, mỗi phần tử có hai vùng chính: vùng dữ liệu danh sách và vùng liên kết. Vùng liên kết là một hoặc nhiều con trỏ chỉ đến các phần tử trước hoặc sau phần tử đang được xem xét tùy thuộc vào yêu cầu của công việc cụ thể

Cú pháp chung cho khai báo danh sách liên kết sử dụng kiểu dữ liệu con trỏ như sau:

```
typedef struct kiểu_dữ_liệu{
<khai báo phần dữ liệu>
<khai báo các con trỏ liên kết>
}t_kiểu_dữ_liệu
```

Bài 11: Kiểu tập tin(File)

- File là loại dữ liệu có thể ghi lên đĩa để dùng nhiều lần.Trong C chỉ có một loại File, nhưng cấu trúc của mỗi File có thể khác nhau. Cấu trúc này được hình thành khi ta ghi dữ liệu lên File, nó phụ thuộc vào hàm mà ta dùng để ghi dữ liệu lên đĩa.
- Trong C có hai loại hàm thao tác trên file:
- Dùng những hàm cấp thấp làm việc với tập tin thông qua một số hiệu tập tin (file handle).
- Dùng những hàm được xây dựng từ những hàm cấp thấp, dễ sử dụng hơn. Có các hàm phục vụ cho việc đọc ghi trên từng loại dữ liệu (số, chuỗi, ký tự, cấu trúc...). Các hàm này làm việc với tập tin thông qua một con trỏ tập tin. Con trỏ này được xác định khi ta mở tập tin.

1. Các kiểu xuất nhập dữ liệu trong tập tin

1.1 Xuất nhập kiểu nhị phân

Dữ liệu ghi lên tập tin không bị thay đổi và khi đóng tập tin thì mã kết thúc tập tin sẽ được ghi lên đĩa là -1.

1.2 Xuất nhập kiểu văn bản

- Chỉ khác kiểu nhập xuất nhị phân khi xử lý ký tự xuống dòng và khi ta đóng tập tin thì mã kết thúc tập tin sẽ được ghi lên đĩa là 26.
- Khi ghi một ký tự chuyển dòng lên đĩa (mã 10) sẽ ghi thành 2 ký tự mã 13 và mã 10.
- Khi đọc nếu gặp hai ký tự liên tiếp là mã 10 và mã 13 sẽ gom lại thành một ký tự là mã 10.

Chú ý:

- Tập tin khi ghi lên đĩa dưới dạng nào thì phải đọc dưới dạng đó. Nếu không việc xử lý sẽ không chính xác.
- Trong C có hàm dùng để nhập xuất cho cả hai kiểu, có hàm chỉ dùng để nhập xuất cho một kiểu nào đó.

2. Các hàm thao tác trên tập tin

- Các hàm sau đây dùng chung cho cả hai kiểu nhị phân và văn bản.

2.1 Mở file

FILE *fopen(const char *tên_tập_tin,const char *kiểu);

- Mở một tập tin. Nếu thành công trả về kết quả là con trỏ FILE tương ứng với file vừa mở, ngược lại trả về giá trị NULL.Sau khi mở file phải kiểm tra xem thao tác mở tập tin thành công hay không.
- *** tên tập tin:** Là một hằng chuỗi, hoặc một con trỏ chỉ đến vùng nhớ chứa tên tập tin.
- *** kiểu:** là hằng chuỗi cho biết kiểu truy nhập:

2.2 Đóng file

int fclose(FILE *f)

- Đóng tập tin được chỉ đến bởi con trỏ f. Nếu thành công thì giá trị của hàm = 0 ngược lại có giá trị EOF. Sau khi đóng con trỏ f sẽ không còn trỏ đến file trước đó nữa

2.3 Làm sạch vùng đệm

int fflush(FILE *f)

- Làm sạch vùng đệm của tập tin được chỉ đến bởi con trỏ f. Nếu thành công cho giá trị 0, ngược lại cho giá trị EOF.

int flushalll(void)

- Làm sạch vùng đệm của tất cả các tập tin đang mở. Nếu thành công giá trị của hàm bằng số tập tin đang mở, ngược lại cho giá trị EOF

2.4 Xoá tập tin

int unlink(const char *tên_tập_tin)

- Xóa một tập tin trên đĩa. Nếu thành công giá trị của hàm bằng 0 , ngược lại cho giá trị EOF

2.5 Đổi tên tập tin

int rename(const char *tên_cũ,const char *tên_mới)

- Đổi một tập tin trên đĩa. Nếu thành công giá trị của hàm bằng 0 , ngược lại cho giá trị EOF

2.6 Kiểm tra kết thúc tập tin

int feof(FILE *f)

- Cho giá trị khác không nếu ở cuối tập tin, ngược lại =0

3. Xuất nhập dữ liệu cho file

3.1 Nhập xuất ký tự : (file kiểu nhị phân và văn bản)

- **Ghi ký tự lên tập tin:**

  int putc(int ch, FILE *f)

  int fputc(int ch, FILE *f)

  Ghi lên file f ký tự có mã = ch % 256 Nếu thành công kết quả = mã của ký tự đã ghi, ngược lại =EOF (-1)

  Trong trường hợp ghi theo văn bản thì khi gặp mã 10 sẽ ghi thành 13 và 10

- **Đọc ký tự từ tập tin:**

  int getc(FILE *f)

  int fgetc(FILE *f)

  Đọc một ký tự từ file f . Nếu thành công kết quả = mã của ký tự đọc được, ngược lại = -1

3.2 Nhập xuất chuỗi: (Dùng cho kiểu văn bản)

- **Ghi một chuỗi:**

  int fputs(const char *s, FILE *f)

  Ghi một chuỗi được chỉ tới bởi con trỏ s vào file f.

  Kết quả = ký tự cuối được ghi nếu thành công, ngược lại =EOF

- **Đọc một chuỗi:**

  char *fgets(const char *s, int n, FILE *f)

  Đọc một chuỗi từ File f và đưa vào vùng nhớ do s trỏ đến.

  Việc đọc kết thúc khi đã đọc được n-1 ký tự , hoặc gặp ký tự xuống dòng , hoặc gặp ký tự kết thúc File.

  Nếu việc đọc có lỗi kết quả của hàm =NULL.

3.3 Đọc ghi dữ liệu theo khuôn dạng: (Dùng cho kiểu văn bản)

- **Ghi dữ liệu theo khuôn dạng:**

  int fprintf(FILE *f , const char *đặc tả,....)

  ... là danh sách các đối số tương ứng với các đặc tả.

  Sử dụng giống như hàm printf, dữ liệu sẽ được ghi lên file.

- **Đọc dữ liệu theo khuôn dạng:**

  fscanf(FILE *f , const char *đặc tả,....)

  ... là danh sách các đối số tương ứng với các đặc tả.

  Sử dụng giống như hàm scanf, dữ liệu sẽ được đọc từ File f rồi đưa vào các đối số tương ứng.

TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP
KHOA ĐIỆN TỬ
BỘ MÔN KỸ THUẬT PHẦN MỀM

BÀI GIẢNG MÔN HỌC

NGÔN NGỮ LẬP TRÌNH BẬC CAO

Theo chương trình 150 TC

Số tín chỉ: 03

(Lưu hành nội bộ)

THÁI NGUYÊN 2010

BỘ MÔN KỸ THUẬT PHẦN MỀM

BÀI GIẢNG MÔN HỌC
NGÔN NGỮ LẬP TRÌNH BẬC CAO

Theo chương trình 150 TC

Số tín chỉ: 03

(Lưu hành nội bộ)

Thái Nguyên, ngày……tháng 12 năm 2010

TRƯỞNG BỘ MÔN **TRƯỞNG KHOA ĐIỆN TỬ**

Ths. Nguyễn Thị Hương **PGS. TS Nguyễn Hữu Công**

MỤC LỤC

ĐỀ CƯƠNG CHI TIẾT HỌC PHẦN

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC
KỸ THUẬT CÔNG NGHIỆP

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

CHƯƠNG TRÌNH GIÁO DỤC ĐẠI HỌC

NGÀNH ĐÀO TẠO: ĐIỆN TỬ

CHUYÊN NGÀNH: CÁC KHỐI NGÀNH KỸ THUẬT

ĐỀ CƯƠNG CHI TIẾT HỌC PHẦN NGÔN NGỮ LẬP TRÌNH BẬC CAO

(HỌC PHẦN BẮT BUỘC)

1. **Tên học phần: Ngôn ngữ lập trình bậc cao**
2. **Số tín chỉ: 03; 3(3;1,5;6)/12**
3. **Trình độ:**
4. **Phân bổ thời gian:**
   - Lên lớp lý thuyết: 3 (tiết/tuần) x 12 (tuần) = 36 tiết.
   - Thảo luận, thực hành: 1,5 (tiết/tuần) x 12 (tuần) = 18 tiết.
     + Thảo luận: 10 tiết
     + Thực hành: 8 tiết
   - Hướng dẫn bài tập lớn (dài):
   - Khác: Không.
   - Tổng số tiết thực dạy: (3+1,5)x12 = 54 tiết thực hiện.
   - Tổng số tiết chuẩn: 3x12+1,5x12/2 = 45 tiết chuẩn.
5. **Các học phần học trước: Toán cao cấp**
6. **Học phần thay thế, học phần tương đương: Không**
7. **Mục tiêu của học phần:**

Trang bị cho sinh viên kiến thức nâng cao trong lĩnh vực tin học, cụ thể: giúp cho sinh viên nắm chắc được quy trình xây dựng chương trình để giải quyết một bài toán cụ thể, đặc biệt là trong lĩnh vực kỹ thuật. Từ khâu đặt vấn đề của bài toán, phân tích yêu cầu của bài toán, xây dựng thuật toán, mã hóa chương trình trên ngôn ngữ bậc cao (C++), kiểm thử và khai thác sử dụng.

8. **Mô tả vắn tắt nội dung học phần:**

Môn học cung cấp các kiến thức chi tiết về ngôn ngữ lập trình C++ nhằm giải quyết các bài toán kỹ thuật. Cụ thể:

- Các thành phần của ngôn ngữ.
- Cấu trúc của một chương trình C++.
- Biến và các kiểu dữ liệu đơn giản trong C++.

- Biểu thức, câu lệnh và các phép toán.
- Câu lệnh đơn giản và câu lệnh có cấu trúc.
- Hàm, đệ quy và truyền tham số.
- Các kiểu dữ liệu có cấu trúc: mảng, xâu, cấu trúc, file.

9. Nhiệm vụ của sinh viên:

- Dự lớp [redacted]80 % tổng số thời lượng của học phần.
- Làm bài tập ở nhà.
- Chuẩn bị thảo luận..

10. Tài liệu học tập:

- **Sách, giáo trình chính:**

[1]. Tống Đình Quỳ, *Ngôn ngữ lập trình C++*, NXB Thống kê 2000.
[2]. Tống Đình Quỳ, *Bài tập ngôn ngữ lập trình C++*, NXB Thống kê 2000.

- **Tài liệu tham khảo:**

[3]. Quách Tuấn Ngọc, *Ngôn ngữ lập trình C*, NXB Giáo Dục, 1998.
[4]. GS. Phạm Văn Ất, *Kỹ thuật lập trình C*, NXB KH&KT, 1999.
[5]. Leendert Ammeraal, *Programs and Data Structures in C*, John Willey & Sons Press.
[6]. N. Wirth, *Cẩm nang lập trình tập 1, tập 2*, NXB Thống kê 1981.
[7]. Đỗ Xuân Lôi, *Cấu trúc dữ liệu và giải thuật*, NXB Thống kê 1996.

11. Tiêu chuẩn đánh giá sinh viên:

- Dự lớp: ≥ 80% tổng số giờ môn học.
- Thảo luận.
- Kiểm tra giữa học phần.
- Thi kết thúc học phần.

*** Thang điểm**

- Thực hành: Trọng số 0.1
- Kiểm tra giữa học phần: Trọng số 0.2
- Thi kết thúc học phần: Trọng số 0.8

12. Nội dung chi tiết học phần:

- Người biên soạn:

KS Võ Phúc Nguyên
KS Đỗ Duy Cốp
ThS Nguyễn Tuấn Anh

CHƯƠNG 1. GIỚI THIỆU NGÔN NGỮ C++

(Tổng số tiết 1; Lý thuyết 1)

1.1. Lịch sử ngôn ngữ C và C++

1.2. Cài đặt C++

1.3. Môi trường Borland C++

1.4. Thiết lập cấu hình cho môi trường

CHƯƠNG 2. CÁC THÀNH PHẦN CƠ BẢN, CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ CÁC PHÉP TOÁN

(Tổng số tiết 9; Lý thuyết 6; Thảo luận 2)

2.1. Các thành phần cơ bản

2.1.1. Bộ ký tự

2.1.2. Tên

2.1.3. Từ khoá

2.1.4. Lời giải thích

2.1.5. Cấu trúc của một chương trình C++ và quy tắc viết chương trình

2.2. Các kiểu dữ liệu và cách khai báo

2.2.1. Kiểu dữ liệu cơ sở

2.2.1.1. Kiểu số nguyên

2.2.1.2. Kiểu số thực

2.2.1.3. Kiểu ký tự

2.2.2. Sự tương thích giữa các kiểu

2.2.3. Định nghĩa và khai báo hằng

2.2.4. Các biến tham chiếu

2.2.5. Biến con trỏ

2.3. Biểu thức, câu lệnh và các phép toán

2.3.1. Biểu thức và các phép toán

2.3.2. Thứ tự thực hiện các phép toán

2.3.3. Câu lệnh

2.3.4. Lệnh hợp thành

2.3.5. Một số hàm số học

CHƯƠNG 3. CÁC THAO TÁC XỬ LÝ INPUT/OUTPUT

(Tổng số tiết 6; Lý thuyết 3; Thảo luận 2 Thực hành 1)

3.1. Hàm in ra màn hình printf()

3.2. Hàm đọc ký tự từ bàn phím scanf()

3.3. Thực hiện Input/Output với dòng tin trong C++

3.3.1. Input

3.3.2. Output

3.4. Thiết lập khuôn dạng - trình bày màn hình

CHƯƠNG 4. CÁC CẤU TRÚC ĐIỀU KHIỂN

(Tổng số tiết 15; Lý thuyết 12; Thảo luận 2, Thực hành 1)

4.1. Cấu trúc if

4.2. Cấu trúc switch

4.3. Cấu trúc for

4.4. Cấu trúc while

4.5. Cấu trúc do

4.6. Câu lệnh break

4.7. Câu lệnh continue

CHƯƠNG 5. HÀM TRONG C++

(Tổng số tiết 8; Lý thuyết 5; Thảo luận 2, Thực hành 1)

5.1. Hàm trong C++

5.2. Truyền tham số cho hàm

5.3. Đệ quy

5.4. Hàm inline

CHƯƠNG 6. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

(Tổng số tiết 15; Lý thuyết 12; Thảo luận 2, Thực hành 1)

6.1. Mảng dữ liệu

6.1.1. Mảng một chiều

6.1.2. Mảng nhiều chiều

6.2. Xâu ký tự và các hàm xử lý xâu

6.3. Cấu trúc (structure)

6.4. Cấu trúc động của dữ liệu (union)

6.5. Các kiểu dữ liệu tự định nghĩa khác

13. Lịch trình giảng dạy

- Số tuần dạy lý thuyết: 08 tuần
- Số tuần thảo luận, bài tập: 04 tuần
- Số tuần thực dạy: 12 tuần

+ 6 Tuần 5 tiết/tuần (4 tuần lý thuyết, 2 tuần thảo luận)
+ 6 Tuần 4 tiết/tuần (4 tuần lý thuyết, 2 tuần thực hành)

Tuần thứ	Nội dung	Tài liệu học tập, tham khảo	Hình thức học
1	**CHƯƠNG 1. GIỚI THIỆU NGÔN NGỮ C++** **1.1. Lịch sử ngôn ngữ C và C++** **1.2. Cài đặt C++** **1.3. Môi trường Borland C++** **1.4. Thiết lập cấu hình cho môi trường** **CHƯƠNG 2. CÁC THÀNH PHẦN CƠ BẢN, CÁC KIỂU DỮ LIỆU CƠ SỞ VÀ CÁC PHÉP TOÁN** **2.1. Các thành phần cơ bản** 2.1.1. Bộ ký tự 2.1.2. Tên 2.1.3. Từ khoá 2.1.4. Lời giải thích 2.1.5. Cấu trúc của một chương trình C++ và quy tắc viết chương trình **2.2. Các kiểu dữ liệu và cách khai báo** 2.2.1. Kiểu dữ liệu cơ sở 2.2.1.1. Kiểu số nguyên 2.2.1.2. Kiểu số thực 2.2.1.3. Kiểu ký tự 2.2.2. Sự tương thích giữa các kiểu 2.2.3. Định nghĩa và khai báo hằng 2.2.4. Các biến tham chiếu 2.2.5. Biến con trỏ	[1] - [8]	Giảng
2	**2.3. Biểu thức, câu lệnh và các phép toán** 2.3.1. Biểu thức và các phép toán 2.3.2. Thứ tự thực hiện các phép toán 2.3.3. Câu lệnh 2.3.4. Lệnh hợp thành 2.3.5. Một số hàm số học **CHƯƠNG 3. CÁC THAO TÁC XỬ LÝ INPUT/OUTPUT** **3.1. Hàm in ra màn hình printf()** **3.2. Hàm đọc ký tự từ bàn phím scanf()** **3.3. Thực hiện Input/Output với dòng tin trong C++**	[1] - [8]	Giảng

	3.3.1. Input 3.3.2. Output **3.4. Thiết lập khuôn dạng - trình bày màn hình**		
3	**CHƯƠNG 4. CÁC CẤU TRÚC ĐIỀU KHIỂN** **4.1. Cấu trúc if** **4.2. Cấu trúc switch** 4.3. **Cấu trúc for**	[1] - [8]	Giảng
4	**4.4. Cấu trúc while** **4.5. Cấu trúc do** **4.6. Câu lệnh break** 4.7. **Câu lệnh continue**	[1] - [8]	Giảng
5	**CHƯƠNG 5. HÀM TRONG C++** **5.1. Hàm trong C++** **5.2. Truyền tham số cho hàm** **5.3. Đệ quy** **5.4. Hàm inline**	[1] - [8]	Thảo luận
6	**Thảo luận và làm các bài tập từ chương 1 đến chương 4**	[1] - [8]	Thảo luận
7	**Kiểm tra giữa kỳ**		
8	**CHƯƠNG 6. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC** **6.1. Mảng dữ liệu** 6.1.1. Mảng một chiều 6.1.2. Mảng nhiều chiều	[1] - [8]	Giảng
9	**6.2. Xâu ký tự và các hàm xử lý xâu** **6.3. Cấu trúc (structure)**	[1] - [8]	Giảng
10	**6.4. Cấu trúc động của dữ liệu (union)** **6.5. Các kiểu dữ liệu tự định nghĩa khác**	[1] - [8]	Giảng
11	**Thực hành**	[1] - [8]	Phòng máy
12	**Thực hành**	[1] - [8]	Phòng máy
13	**Thảo luận và làm các bài tập từ chương 5 đến chương 6**	[1] - [8]	Giảng

14. Ngày phê duyệt:

15. Cấp phê duyệt:

Đề cương chi tiết học phần đã được Hội đồng khối ngành **Điện – Điện tử và SPKT Điện – Tin học** phê duyệt.

Trưởng bộ môn Kỹ thuật phần mềm	Chủ tịch Hội đồng KH&GD Khoa Điện tử	Chủ tịch Hội đồng Khối ngành Điện - Điện tử và SPKT Điện – Tin học
ThS. Nguyễn Thị Hương	**PGS.TS. Nguyễn Hữu Công**	**PGS.TS Nguyễn Như Hiển**

CHƯƠNG 1. GIỚI THIỆU NGÔN NGỮ C++

1.1. Lịch sử ngôn ngữ C++

Lịch sử ngôn ngữ lập trình C: là một ngôn ngữ mệnh lệnh được phát triển từ đầu thập niên 1970 bởi Ken Thompson và Dennis Ritchie tại phòng thí nghiệm Bell Telephone để dùng trong hệ điều hành UNIX; theo Ritchie thì thời gian sáng tạo nhất là vào năm 1972. Nó được đặt tên là C vì nhiều đặc tính của nó rút ra từ một ngôn ngữ trước đó là B. Từ dó, ngôn ngữ này đã lan rộng ra nhiều hệ điều hành khác và trở thành một những ngôn ngữ phổ dụng nhất. C là ngôn ngữ rất có hiệu quả và được ưa chuộng nhất để viết các phần mềm hệ thống, mặc dù nó cũng được dùng cho việc viết các ứng dụng. Ngoài ra, C cũng thường được dùng làm phương tiện giảng dạy trong khoa học máy tính mặc dù ngôn ngữ này không dược thiết kế dành cho người nhập môn.

C++ là gì? Có thể thấy rằng ngôn ngữ lập trình C được phát triển đầu tiên sau đó C++ mới được phát triển. Vậy C++ nó là cái gì? Nó có mối quan hệ thế nào với C. Câu trả lời là: C++ cơ bản là C ở một mức độ mới. Sự khác biệt quan trọng duy nhất là C++ hỗ trợ hướng đối tượng. Các đoạn mã viết bằng C được dịch và chạy tốt với hầu hết các chương trình dịch của C++ nhưng điều ngược lại không đúng. C++ hỗ trợ tất cả các lệnh của C và có mở rộng.

Lịch sử ngôn ngữ lập trình C++: Bjarne Stroustrup của Bell Labs đã phát triển C++ (mà tên nguyên thủy là "C với các lớp" trong suốt thập niên 1980 như là một bản nâng cao của ngôn ngữ C. Những bổ sung nâng cao bắt đầu với sự thêm vào của khái niệm lớp, tiếp theo đó là các khái niệm hàm ảo, toán tử quá tải, đa kế thừa, tiêu bản, và xử lý ngoại lệ. Tiêu chuẩn của ngôn ngữ C++ đã được thông qua trong năm 1998 như là ISO/IEC 14882: 1998. Phiên bản hiện đang lưu hành là phiên bản 2003, ISO/IEC 14882: 2003.

Năm 1983, thì tên C với các lớp được đổi thành C++. các chức năng mới được thêm vào bao gồm hàm ảo, quá tải hàm và toán tử, tham chiếu, hằng, khả năng kiểm soát bộ nhớ của lưu trữ tự do, nâng cao việc kiểm soát kiểu, và lệnh chú giải kiểu (//).

Năm 1989 phiên bản C++ 2.0 phát hành. Các tính năng mới bao gồm đa kế thừa, lớp trừu tượng, hàm tĩnh, hàm thành viên hằng, và thành viên bảo tồn.

Phiên bản xuất bản sau đó có thêm các chức năng tiêu bản, ngoại lệ, không gian tên, chuyển kiểu cho toán tử new, và kiểu Boolean.

Khi C++ hình thành, thì thư viện chuẩn hoàn thiện với nó. Thư viện C++ đầu tiên thêm vào là IOSttream cung cấp cơ sở để thay thế các hàm C truyền thống như là printf và scanf. Sau này, trong những thư viện chuẩn quan trọng nhất được thêm vào là Thư viện Tiêu bản Chuẩn.

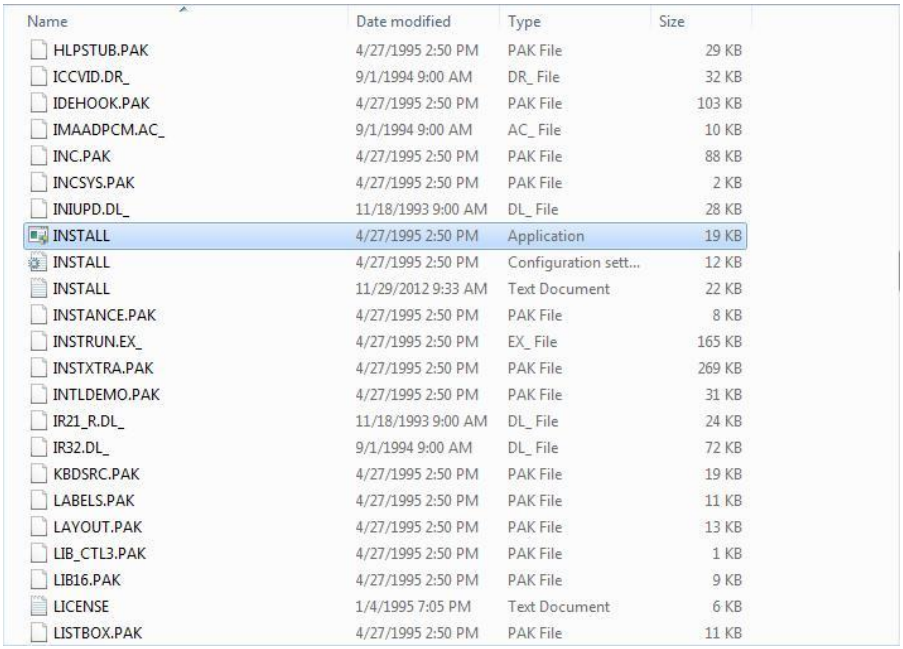
Sau nhiều năm làm việc, có sự cộng tác giữa ANSI và hội đồng tiêu chuẩn hoá C++ của ISO để soạn thảo tiêu chuẩn ISO/IEC 14882: 1998. Phiên bản tiêu chuẩn này được phát hành năm 1989, hội đồng tiếp tục xử lí các báo cáo trục trặc, và ấn hành một phiên bản sửa sai của chuẩn C++ trong năm 2003.

Không ai là chủ nhân của ngôn ngữ C++, nó hoàn toàn miễn phí khi dùng. Mặc dù vậy, các văn bản tiêu chuẩn thì không miễn phí.

C++ về bản chất được xây dựng trên nền của ngôn ngữ lập trình C. C++ nguyên là sự kết thừa từ C. Mặc dù vậy, không phải mọi chương trình trong C đều hợp lệ trong C++. Vì là hai ngôn ngữ độc lập, số lượng không tương thích giữa hai ngôn ngữ này đã tăng lên. [2]. Phiên bản cuối cùng C99 đã tạo ra thêm nhiều tính năng xung đột (giữa C và C++). Các sự khác nhau này tạo ra khó khăn để viết các chương trình và thư viện để có thể được dịch và hoạt động chính xác trong cả hai loại mã C hay C++, đồng thời gây nhầm lẫn cho những người lập trình dùng cả hai ngôn ngữ này. Sự chênh lệch này cũng gây khó khăn cho ngôn ngữ này có thể tiếp thu các tính năng của ngôn ngữ kia.

1.2. Cài đặt Borland C++ 4.5

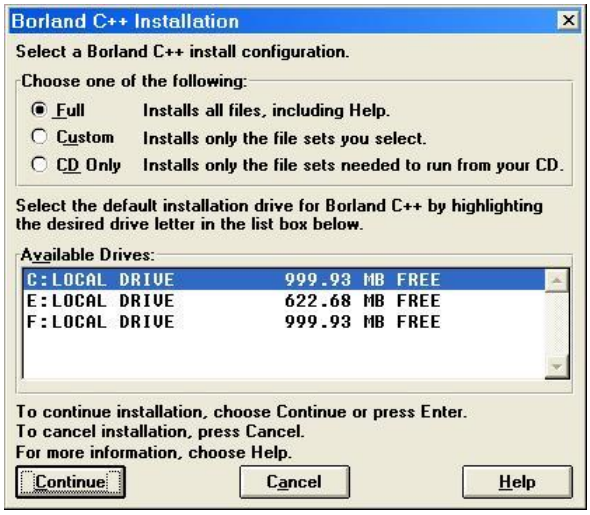
Bước 1: Chạy file install.exe



Name	Date modified	Type	Size
HLPSTUB.PAK	4/27/1995 2:50 PM	PAK File	29 KB
ICCVID.DR_	9/1/1994 9:00 AM	DR_ File	32 KB
IDEHOOK.PAK	4/27/1995 2:50 PM	PAK File	103 KB
IMAADPCM.AC_	9/1/1994 9:00 AM	AC_ File	10 KB
INC.PAK	4/27/1995 2:50 PM	PAK File	88 KB
INCSYS.PAK	4/27/1995 2:50 PM	PAK File	2 KB
INIUPD.DL_	11/18/1993 9:00 AM	DL_ File	28 KB
INSTALL	4/27/1995 2:50 PM	Application	19 KB
INSTALL	4/27/1995 2:50 PM	Configuration sett...	12 KB
INSTALL	11/29/2012 9:33 AM	Text Document	22 KB
INSTANCE.PAK	4/27/1995 2:50 PM	PAK File	8 KB
INSTRUN.EX_	4/27/1995 2:50 PM	EX_ File	165 KB
INSTXTRA.PAK	4/27/1995 2:50 PM	PAK File	269 KB
INTLDEMO.PAK	4/27/1995 2:50 PM	PAK File	31 KB
IR21_R.DL_	11/18/1993 9:00 AM	DL_ File	24 KB
IR32.DL_	9/1/1994 9:00 AM	DL_ File	72 KB
KBDSRC.PAK	4/27/1995 2:50 PM	PAK File	19 KB
LABELS.PAK	4/27/1995 2:50 PM	PAK File	11 KB
LAYOUT.PAK	4/27/1995 2:50 PM	PAK File	13 KB
LIB_CTL3.PAK	4/27/1995 2:50 PM	PAK File	1 KB
LIB16.PAK	4/27/1995 2:50 PM	PAK File	9 KB
LICENSE	1/4/1995 7:05 PM	Text Document	6 KB
LISTBOX.PAK	4/27/1995 2:50 PM	PAK File	11 KB

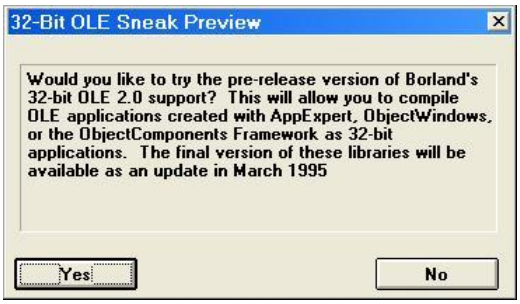
Bước 2: Chọn nút Continue

Bước 3: Chọn nút Continue

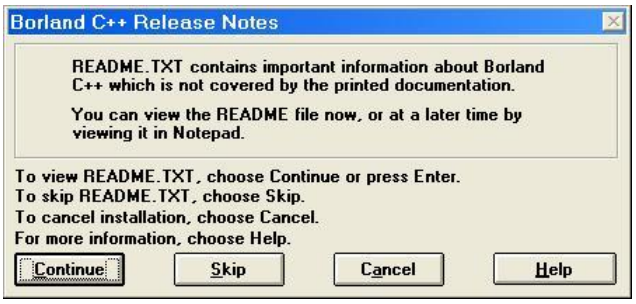


Bước 4: Chọn nút Continue

Bước 5: Chọn nút Yes

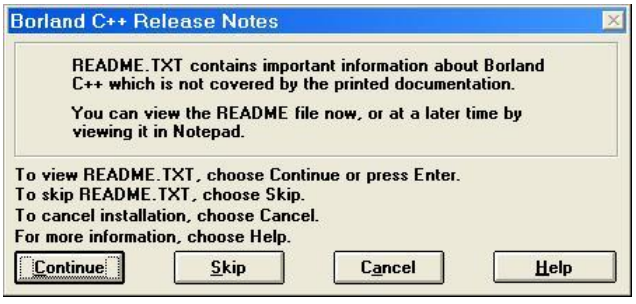


Bước 6: Chọn nút Continue

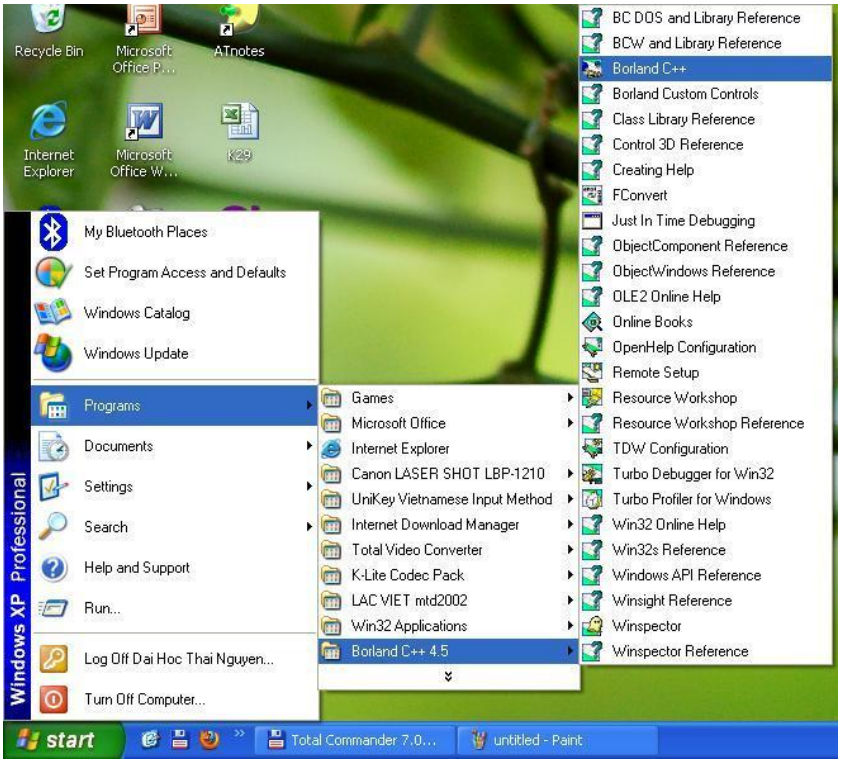


Bước 7: Chọn nút Install

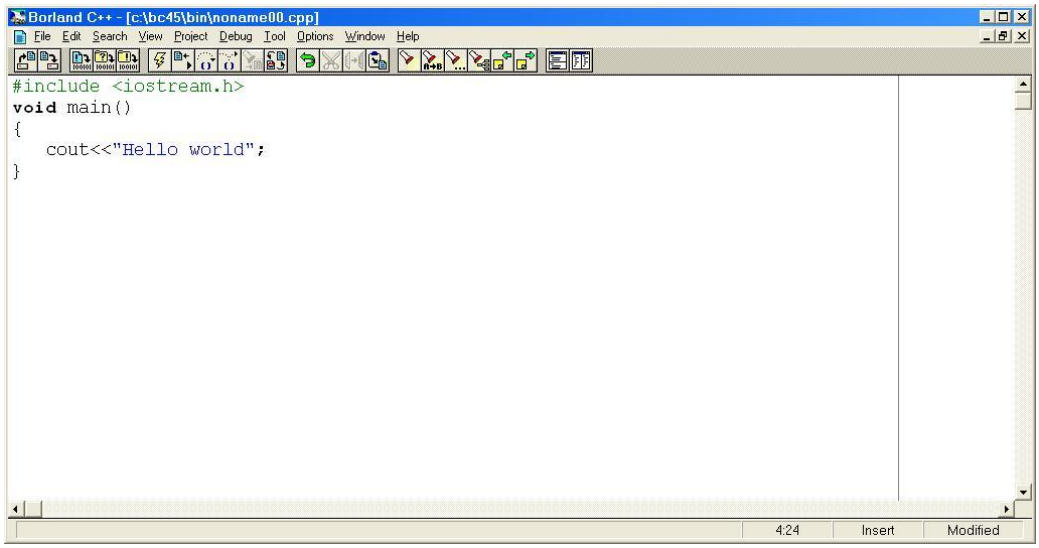
Bước 8: Chọn nút Continue



Bước 9: Sau khi chọn nút OK ta đã cài xong chương trình Borland C++ 4.5.



Giao diện chương trình Borland C++ 4.5



CHƯƠNG 2. THÀNH PHẦN CƠ BẢN, KIỂU DỮ LIỆU CƠ SỞ VÀ PHÉP TOÁN

A. Phần lý thuyết

2.1. Các thành phần cơ bản

2.1.1. Bộ ký tự (Character Set)

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Các từ lại được liên kết với nhau theo một qui tắc nào đó để tạo nên các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và thể hiện một thuật toán để giải một bài toán nào đó. Ngôn ngữ C++ được xây dựng trên bộ ký tự sau:

- 26 chữ cái hoa: A B C.. Z
- 26 chữ cái thường: a b c… z
- 10 chữ số: 0 1 2… 9
- Các ký hiệu toán học: + - * / =()
- Ký tự gạch nối: _
- Các ký tự khác:.,: ; [ ] {} ! \ & % # $...

Dấu cách (space) dùng để tách các từ. Ví dụ chữ VIET NAM có 8 ký tự, còn VIETNAM chỉ có 7 ký tự.

Chú ý: Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên.

Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2 + bx + c = 0$, ta cần tính biệt thức Delta $\Delta = b^2 - 4ac$, trong ngôn ngữ C++ không cho phép dùng ký tự Δ vì vậy ta phải dùng ký hiệu khác để thay thế.

2.1.2. Tên (Identifier)

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn,...

Tên được đặt theo qui tắc sau:

Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng: Option-Compiler-Source-Identifier length khi dùng TURBO C++.

Các tên đúng: a_1 delta x1 _step GAMA

Các tên sai:

3MN	Ký tự đầu tiên là số
m#2	Sử dụng ký tự #
f(x)	Sử dụng các dấu ()
do	Trùng với từ khoá
te ta	Sử dụng dấu trắng
Y-3	Sử dụng dấu -

Chú ý:

Trong TURBO C++, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. Trong C++, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

2.1.3. Từ khoá (Keywords)

Từ khoá là những từ được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Bảng dưới đây liệt kê các từ khoá của TURBO C++:

asm	break	case	cdecl
char	const	continue	default
do	double	else	enum
extern	far	float	for
goto	huge	if	int
interrupt	long	near	pascal
register	return	short	signed
sizeof	static	struct	switch
tipedef	union	unsigned	void
volatile	while		

Chú ý:

- Không được dùng các từ khoá để đặt tên cho các hằng, biến, mảng, hàm...

- Từ khoá phải được viết bằng chữ thường, ví dụ: viết từ khoá khai báo kiểu nguyên là int chứ không phải là int.

2.1.4. Lời giải thích (Comments)

Các chú thích được các lập trình viên sử dụng để ghi chú hay mô tả trong các phần của chương trình. Trong C++ có hai cách để chú thích:

//Chú thích theo dòng

/*Chú thích theo khối

Dòng chú thích 1

Dòng chú thích 2

*/

Chú thích theo dòng bắt đầu từ cặp dấu xổ (//) cho đến cuối dòng. Chú thích theo khối bắt đầu bằng /*và kết thúc bằng */ và có thể bao gồm nhiều dòng. Chúng ta sẽ thêm các chú thích cho chương trình:

Ví dụ 2.1

```
//Ví dụ về ghi chú trong C++
#include <iostream.h>
void main(){
     //In ra màn hình dòng Hello World!
     cout<<"Hello World! ";
     //In ra màn hình dòng I'm a C++ program
     cout<<"I'm a C++ program";
}
```

2.1.5. Cấu trúc của một chương trình C++

Chúng ta xem xét ví dụ sau đây:

Ví dụ 2.2

```
//My first program in C++
#include <iostream.h>
int main()
{
     cout<<"Hello World!";
     return 0;
}
```

Chương trình trên đây là chương trình đầu tiên mà hầu hết những người học nghề lập trình viết đầu tiên và kết quả của nó là viết câu "Hello, World" lên màn hình. Đây là một trong những chương trình đơn giản nhất có thể viết bằng C++ nhưng nó

đã bao gồm những phần cơ bản mà mọi chương trình C++ có. Hãy cùng xem xét từng dòng một:

//My first program in C++

Đây là dòng chú thích. Tất cả các dòng bắt đầu bằng hai dấu sổ (//) được coi là chút thích mà chúng không có bất kì một ảnh hưởng nào đến hoạt động của chương trình. Chúng có thể được các lập trình viên dùng để giải thích hay bình phẩm bên trong mã nguồn của chương trình. Trong trường hợp này, dòng chú thích là một giải thích ngắn gọn những gì mà chương trình chúng ta làm.

#include <iostream.h>

Câu lệnh #include <iostream.h> báo cho trình dịch biết cần phải "include" thư viện iostream. Đây là một thư viện vào ra cơ bản trong C++ và nó phải được "include" vì nó sẽ được dùng trong chương trình. Đây là cách cổ điển để sử dụng thư viện iostream

int main()

Dòng này tương ứng với phần bắt đầu khai báo hàm main. Hàm main là điểm mà tất cả các chương trình C++ bắt đầu thực hiện. Nó không phụ thuộc vào vị trí của hàm này (ở đầu, cuối hay ở giữa của mã nguồn) mà nội dung của nó luôn được thực hiện đầu tiên khi chương trình bắt đầu. Thêm vào đó, do nguyên nhân nói trên, mọi chương trình C++ đều phải tồn tại một hàm main.

Theo sau main là một cặp ngoặc đơn bởi vì nó là một hàm. Trong C++, tất cả các hàm mà sau đó là một cặp ngoặc đơn () thì có nghĩa là nó có thể có hoặc không có tham số (không bắt buộc). Nội dung của hàm main tiếp ngay sau phần khai báo chính thức được bao trong các ngoặc nhọn ({ }) như trong ví dụ của chúng ta.

cout<<"Hello World";

Dòng lệnh này làm việc quan trọng nhất của chương trình. cout là một dòng (stream) output chuẩn trong C++ được định nghĩa trong thư viện iostream và những gì mà dòng lệnh này làm là gửi chuỗi kí tự "Hello World" ra màn hình.

Chú ý rằng dòng này kết thúc bằng dấu chấm phẩy (;). Kí tự này được dùng để kết thúc một lệnh và bắt buộc phải có sau mỗi lệnh trong chương trình C++ của bạn (một trong những lỗi phổ biến nhất của những lập trình viên C++ là quên mất dấu chấm phẩy).

return 0;

Lệnh return kết thúc hàm main và trả về mã đi sau nó, trong trường hợp này là 0. Đây là một kết thúc bình thường của một chương trình không có một lỗi nào trong quá trình thực hiện. Như bạn sẽ thấy trong các ví dụ tiếp theo, đây là một cách phổ biến nhất để kết thúc một chương trình C++.

Chương trình được cấu trúc thành những dòng khác nhau để nó trở nên dễ đọc hơn nhưng hoàn toàn không phải bắt buộc phải làm vậy. Ví dụ, thay vì viết

```
int main()
{
     cout<<" Hello World ";
     return 0;
}
```

ta có thể viết

```
int main() { cout<<" Hello World "; return 0; }
```

cũng cho một kết quả chính xác như nhau.

Trong C++, các dòng lệnh được phân cách bằng dấu chấm phẩy (Việc chia chương trình thành các dòng chỉ nhằm để cho nó dễ đọc hơn mà thôi).

2.2. Các kiểu dữ liệu và cách khai báo

2.2.1. Khái niệm về kiểu dữ liệu

C++ chia thành hai tập hợp kiểu dữ liệu chính:

Kiểu xây dựng sẵn (built- in) mà ngôn ngữ cung cấp cho người lập trình và ***kiểu được người dùng định nghĩa*** (user-defined) do người lập trình tạo ra. 2 tập hợp kiểu dữ liệu này phân làm 2 loại kiểu dữ liệu giá trị (value) và kiểu dữ liệu tham chiếu(reference). Trong đó Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng và chuỗi. Và tất cả các kiểu do người dùng định nghĩ ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu.

Các đối tượng thuộc 1 kiểu dữ liệu có thể được chuyển đổi sang một kiểu dữ liệu khác qua cơ chế chuyển đổi tường minh hoặc chuyển đổi ngầm định. Việc chuyển đổi ngầm định được thực hiện tự động và phải đảm bảo không mất thông tin. Nghĩa là kiểu được chuyển đổi phải có kích thước lơn hơn.

Ta có thể gán ngầm định một số kiểu short vào một số kiểu int nhưng không thể làm ngược lại vì kích thước của kiểu int lớn hơn kích thước của kiểu short.

2.2.2. Kiểu dữ liệu cơ sở

- ***Kiểu số nguyên***

Trong C++ cho phép sử dụng số nguyên kiểu int, số nguyên dài kiểu long và số nguyên không dấu kiểu unsigned. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây:

Kiểu	Phạm vi biểu diễn	Kích thước
unsigned int	0 đến 65535	2 byte
Short int	-32768 đến 32767	2 byte
int	-32768 đến 32767	2 byte
long	-2147483648 đến 2147483647	4 byte
unsigned long	0 đến 4294967295	4 byte

Chú ý: Kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên.

- ***Kiểu số thực***

Trong C++ cho phép sử dụng ba loại dữ liệu dấu phảy động, đó là float, double và long double. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây:

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
float	3.4E-38 đến 3.4E+38	7 đến 8	4 byte
double	1.7E-308 đến 1.7E+308	15 đến 16	8 byte
long double	3.4E-4932 đến 1.1E4932	17 đến 18	10 byte

Giải thích: Máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ 3.4E-38 đến 3.4E+38. Các số có giá trị tuyệt đối nhỏ hơn3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

- ***Kiểu ký tự***

Một giá trị kiểu char chiếm 1 byte (8 bit) và biểu diễn được một ký tự thông qua bảng mã ASCII.

Ví dụ 2.3

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
B	066
A	097
B	098

Có hai kiểu dữ liệu char: kiểu signed char và unsigned char.

Kiểu	Phạm vi biểu diễn	Số ký tự	Kích thước	
char (signed char)	-128 đến 127	256	1 byte	
unsigned char	0 đến 255	256	1 byte	

Ví dụ sau minh hoạ sự khác nhau giữa hai kiểu dữ liệu trên: Xét đoạn chương trình sau:

```
char ch1;
unsigned char ch2;
......
ch1=200; ch2=200;
```

Khi đó thực chất:

```
ch1=-56;
ch2=200;
```

Nhưng cả ch1 và ch2 đều biểu diễn cùng một ký tự có mã 200.

Có thể chia 256 ký tự làm ba nhóm:

Nhóm 1: Nhóm các ký tự điều khiển có mã từ 0 đến 31. Chẳng hạn ký tự mã 13 dùng để chuyển con trỏ về đầu dòng, ký tự 10 chuyển con trỏ xuống dòng dưới (trên cùng một cột). Các ký tự nhóm này nói chung không hiển thị ra màn hình.

Nhóm 2: Nhóm các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể được đưa ra màn hình hoặc máy in.

Nhóm 3: Nhóm các ký tự đồ hoạ có mã số từ 127 đến 255. Các ký tự này có thể đưa ra màn hình nhưng không in ra được (bằng các lệnh DOS).

- ***Biểu boolean***

C++ không có kiểu Boolean xây dựng sẵn. Vì vậy ta có thể sử dụng kiểu int cho mục đích này với "0" là **false** và các số khác là **true.**

2.2.3. Sự tương thích giữa các kiểu

Việc chuyển đổi một kiểu đã xác định thành một kiểu khác về cơ bản sẽ thay đổi một hay cả hai tính chất của kiểu nhưng không làm thay đổi hình mẫu bit nền tảng. Kích cỡ kiểu có thể rộng hơn hay hẹp hơn và tất nhiên cách hiểu về kiểu sẽ thay đổi. Một số phép chuyển đổi là không an toàn, về cơ bản trình biên dịch sẽ cảnh báo điều đó, việc chuyển đổi từ kiểu dữ liệu rộng hơn sang một kiểu dữ liệu hẹp hơn là một hình thái chuyển đổi kiểu không an toàn.

2.2.4. Định nghĩa và khai báo hằng

a) Định nghĩa và khai báo hằng

Định nghĩa hằng:

Bạn có thể định nghĩa các hằng với tên mà bạn muốn để có thể sử dụng thường xuyên mà không mất tài nguyên cho các biến bằng cách sử dụng chỉ thị #define. Đây là dạng của nó:

#define <tên hằng> [giá trị]

Ví dụ 2.4

```
#define PI 3.14159265
#define NEWLINE '\n'
#define WIDTH 100
```

Chúng định nghĩa ba hằng số mới. Sau khi khai báo bạn có thể sử dụng chúng như bất kì các hằng số nào khác, ví dụ:

```
circle = 2 * PI * r;
cout<<NEWLINE;
```

Trong thực tế việc duy nhất mà trình dịch làm khi nó tìm thấy một chỉ thị #define là thay thế các tên hằng tại bất kì chỗ nào chúng xuất hiện (như trong ví dụ trước, PI, NEWLINE hay WIDTH) bằng giá trị mà chúng được định nghĩa.

Chỉ thị #define không phải là một lệnh thực thi, nó là chỉ thị tiền xử lý (preprocessor), đó là lý do trình dịch coi cả dòng là một chỉ thị và dòng đó không cần kết thúc bằng dấu chấm phẩy. Nếu bạn thêm dấu chấm phẩy vào cuối dòng, nó sẽ được coi là một phần của giá trị định nghĩa hằng.

Khai báo các hằng (const):

Với tiền tố const bạn có thể khai báo các hằng với một kiểu xác định như là bạn làm với một biến:

```
const int width = 100;
```

```
const to char tab = '\t';
const zip = 12440;
```

b) Các loại hằng

Hằng int:

Hằng int là số nguyên có giá trị trong khoảng từ -32768 đến 32767.

Ví dụ 2.5

#define number1 -50	Định nghiã hằng int number1 có giá trị là -50	
#define sodem 2732	Định nghiã hằng int sodem có giá trị là 2732	

Chú ý:

Cần phân biệt hai hằng 5056 và 5056.0: ở đây 5056 là số nguyên còn 5056.0 là hằng thực.

Hằng long:

Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647. Hằng long được viết theo cách:

1234L hoặc 1234l (thêm L hoặc l vào đuôi)

Một số nguyên vượt ra ngoài miền xác định của int cũng được xem là long.

Ví dụ 2.6

#define sl 8865056L	Định nghĩa hằng long sl có giá trị là 8865056
#define sl 8865056	Định nghiã hằng long sl có giá trị là 8865056

Hằng ký tự:

Hằng ký tự là một ký tự riêng biệt được viết trong hai dấu nháy đơn, ví dụ 'a'.

Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác.

Ví dụ: '9'-'0'=57-48=9

Chú thích:

#define kt 'a' Định nghĩa hằng ký tự kt có giá trị là 97

Hằng ký tự còn có thể được viết theo cách sau: '\c1c2c3', trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

Ví dụ: chữ a có mã hệ 10 là 97, đổi ra hệ 8 là 0141. Vậy hằng ký tự 'a' có thể viết dưới dạng '\141'. Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu \):

Cách viết	Ký tự
'\''	'
'\"'	"
'\\'	\
'\n'	\n (chuyển dòng)
'\0'	\0 (null)
'\t'	Tab
'\b'	Backspace
'\r'	CR (về đầu dòng)
'\f'	LF (sang trang)

Chú ý:

Cần phân biệt hằng ký tự '0' và '\0'. Hằng '0' ứng với chữ số 0 có mã ASCII là 48, còn hằng '\0' ứng với ký tự \0 (thường gọi là ký tự null) có mã ASCII là 0.

Hằng ký tự thực sự là một số nguyên, vì vậy có thể dùng các số nguyên hệ 10 để biểu diễn các ký tự, ví dụ lệnh printf("%c%c",65,66) sẽ in ra AB.

Hằng xâu ký tự:

Hằng xâu ký tự là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép.

Ví dụ 2.7

```
#define xau1 "Ha noi"
#define xau2 "My name is Giang"
```

Xâu ký tự được lưu trữ trong máy dưới dạng một bảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự null \0 vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

Chú ý:

Cần phân biệt hai hằng 'a' và "a". 'a' là hằng ký tự được lưu trữ trong 1 byte, còn "a" là hằng xâu ký tự được lưu trữ trong 1 mảng hai phần tử: phần tử thứ nhất chứa chữ a còn phần tử thứ hai chứa \0. Các hình thức chuyển đổi kiểu giá trị.

Các toán tử chuyển đổi kiểu cho phép bạn chuyển đổi dữ liệu từ kiểu này sang kiểu khác. Có vài cách để làm việc này trong C++, cách cơ bản nhất được thừa kế từ

ngôn ngữ C++ là đặt trước biểu thức cần chuyển đổi tên kiểu dữ liệu được bọc trong cặp ngoặc đơn ().

Ví dụ 2.8

Cách 1:

```
int i;
float f = 3.14;
i = (int) f;
```

Cách 2:

```
i = int (f);
```

Cả hai cách chuyển đổi kiểu đều hợp lệ trong C++. Thêm vào đó ANSI-C++ còn có những toán tử chuyển đổi kiểu mới đặc trưng cho lập trình hướng đối tượng.

2.2.5. Biến

a) Khai báo

Để có thể sử dụng một biến trong C++, đầu tiên chúng ta phải khai báo nó, ghi rõ nó là kiểu dữ liệu nào. Chúng ta chỉ cần viết tên kiểu (như **int**, **short**, **float,**...) tiếp theo sau đó là một tên biến hợp lệ.

Ví dụ 2.9

```
int a;
float mynumber;
```

Dòng đầu tiên khai báo một biến kiểu int với tên là a. Dòng thứ hai khai báo một biến kiểu float với tên mynumber. Sau khi được khai báo, các biến trên có thể được dùng trong phạm vi của chúng trong chương trình.

Nếu bạn muốn khai báo một vài biến có cùng một kiểu và bạn muốn tiết kiệm công sức viết bạn có thể khai báo chúng trên một dòng, ngăn cách các tên bằng dấu phẩy.

Ví dụ 2.10

```
int a, b, c;
```

khai báo ba biến kiểu int (a,b và c) và hoàn toàn tương đương với:

```
int a;
int b;
int c;
```

Các kiểu số nguyên (char, short, long and int) có thể là số có dấu hay không dấu tuỳ theo miền giá trị mà chúng ta cần biểu diễn. Vì vậy khi xác định một kiểu số nguyên chúng ta đặt từ khoá signed hoặc unsigned trước tên kiểu dữ liệu. Ví dụ:

```
unsigned short NumberOfSons;
signed int MyAccountBalance;
```

Nếu ta không chỉ rõ signed or unsigned nó sẽ được coi là có dấu, vì vậy trong khai báo thứ hai chúng ta có thể viết:

```
int MyAccountBalance
```

cũng hoàn toàn tương đương với dòng khai báo ở trên. Trong thực tế, rất ít khi người ta dùng đến từ khoá signed. Ngoại lệ duy nhất của luật này kiểu char. Trong chuẩn ANSI-C++ nó là kiểu dữ liệu khác với signed char và unsigned char.

Để có thể thấy rõ hơn việc khai báo trong chương trình, chúng ta sẽ xem xét một đoạn mã C++ ví dụ như sau:

b) Khởi tạo các biến

Khi khai báo một biến, giá trị của nó mặc nhiên là không xác định. Nhưng có thể bạn sẽ muốn nó mang một giá trị xác định khi được khai báo. Để làm điều đó, bạn chỉ cần viết dấu bằng và giá trị bạn muốn biến đó sẽ mang:

type <tên biến> = <giá trị khởi tạo>;

Ví dụ, nếu chúng ta muốn khai báo một biến int là a chứa giá trị 0 ngay từ khi khởi tạo, chúng ta sẽ viết:

```
int a = 0;
```

Bổ xung vào cách khởi tạo kiểu C này, C++ còn có thêm một cách mới để khởi tạo biến bằng cách bọc một cặp ngoặc đơn sau giá trị khởi tạo. Ví dụ:

```
int a (0);
```

Cả hai cách đều hợp lệ trong C++.

c) Phạm vi hoạt động của các biến

Tất cả các biến mà chúng ta sẽ sử dụng đều phải được khai báo trước. Một điểm khác biết giữa C và C++ là trong C++ chúng ta có thể khai báo biến ở bất kì nơi nào trong chương trình, thậm chí là ngay ở giữa các lệnh thực hiện chứ không chỉ là ở đầu khối lệnh như ở trong C.

Biến toàn cục: có thể được sử dụng ở bất kì đâu trong chương trình, ngay sau khi nó được khai báo.

Biến cục bộ: Tầm hoạt động của biến cục bộ bị giới hạn trong phần mã mà nó được khai báo. Nếu chúng được khai báo ở đầu một hàm (như hàm main), tầm hoạt động sẽ là toàn bộ hàm main. Điều đó có nghĩa là trong ví dụ trên, các biến được khai báo trong hàm main() chỉ có thể được dùng trong hàm đó, không được dùng ở bất kì đâu khác.

Trong C++ tầm hoạt động của một biến chính là khối lệnh mà nó được khai báo (một khối lệnh là một tập hợp các lệnh được gộp lại trong một bằng các ngoặc nhọn **{ }**). Nếu nó được khai báo trong một hàm tầm hoạt động sẽ là hàm đó, còn nếu được khai báo trong vòng lặp thì tầm hoạt động sẽ chỉ là vòng lặp đó,....

2.2.6. Biến tham chiếu (reference)

a) Sự tham chiếu

Khi một con trỏ được khai báo, thì chưa có một địa chỉ nào gán cho nó cả. Địa chỉ tương quan với một con trỏ có thể được thay đổi (hay hình thành) qua phép gán. Trong thí dụ sau, biến ptr sẽ cài cho nó chỉ tới cùng một dữ liệu như là biến nguyên cơ bản a:

```
int *ptr;
int a;
ptr = &a;
```

Để làm được việc này, toán tử tham chiếu (hay còn gọi là tham chiếu ngược) & đã được dùng tới. Nó trả về vị trí của biến trong bộ nhớ (tức là địa chỉ) hay là chỗ chứa thực thể theo sau (trong thí dụ thì thực thể đó là a). Toán tử này như là công việc nó làm thường được gọi là toán tử "địa chỉ".

Trong trường hợp tổng quát, cụm từ giá trị tham chiếu được dùng để nói về địa chỉ trong bộ nhớ của sự tham chiếu (hay tham chiếu ngược).

b) Sự tham chiếu ngược

Cùng một biểu hiện, giá trị có thể đọc về từ một giá trị tham chiếu. Trong thí dụ sau, biến nguyên cơ bản b sẽ được gán đến dữ liệu mà dữ liệu đó được tham chiếu bởi ptr:

```
int *ptr;
int a, b;
//Gán cho a giá trị là 10
a = 10;
//Phép gán địa chỉ của a (tức là &a) lên con trỏ 'ptr'
//để ptr bây giờ chỉ đến địa chỉ có nội dung là 10
ptr = &a;
//Phép gán này cho b một giá trị nằm ở địa chỉ mà 'ptr'
```

```
//Chỉ tới tức là giá trị của b sẽ là 10
b = *ptr
```

Để hoàn tất được thao tác này, toán tử tham chiếu ngược (&) đã được dùng. Nó trả về dữ liệu cơ bản. Dữ liệu này có giá trị là một tham chiếu chỉ tới (tức là một địa chỉ). Biểu thức *ptr là một cách viết khác của giá trị 10 (gán cho b).

Việc quá tải của kí tự * có hai biểu hiện liên hệ mà có thể gây ra sự nhầm lẫn. Hiểu được sự khác nhau giữa việc dùng nó như là một tiền tố trong một khai báo (con trỏ) và việc xem nó là toán tử tham chiếu trong một biểu thức là rất quan trọng.

c) Sự tham chiếu tương đương và các mệnh đề cơ bản

Bảng sau đây là danh sách các mệnh đề tương đương giữa kiểu cơ bản và kiểu tham chiếu (hay tham chiếu ngược). Trong đó, biến cơ bản d và biến tham chiếu ptr được hiểu ngầm. Các mệnh đề cơ bản và tham chiếu tương đương

	Đến một giá trị cơ bản	Đến một giá trị tham chiếu
Từ một giá trị cơ bản	d	&d
Từ một giá trị tham chiếu	*ptr	ptr

2.2.7. Biến con trỏ (pointer)

Con trỏ đối tượng dùng để chứa địa chỉ của biến đối tượng, được khai báo như sau:

Kiểu_dữ_liệu * Tên_con_ trỏ;

Ví dụ 2.11

```
int *p1, *p2, *p3;   //Khai báo 3 con trỏ p1, p2, p3
int d1, d2;          //Khai báo hai đối tượng d1, d2
int d[20];           //Khai báo mảng đối tượng
```

Có thể thực hiện câu lệnh:

```
p1 = &d2;        //p1 chứa địa chỉ của d2, p1 trỏ tới d2
p2 =d;           //p2 trỏ tới đầu mảng d
```

2.2.8. Chuyển đổi kiểu dữ liệu

Khi tính toán một biểu thức phần lớn các phép toán đều yêu cầu các toán hạng phải cùng kiểu. Ví dụ để phép gán thực hiện được thì giá trị của biểu thức phải có cùng kiểu với biến. Trong trường hợp kiểu của giá trị biểu thức khác với kiểu của phép gán thì hoặc là chương trình sẽ tự động chuyển kiểu giá trị biểu thức về thành kiểu của biến được gán (nếu được) hoặc sẽ báo lỗi. Do vậy khi cần thiết người lập

trình phải sử dụng các câu lệnh để chuyển kiểu của biểu thức cho phù hợp với kiểu của biến.

a) Chuyển kiểu tự động: về mặt nguyên tắc, khi cần thiết các kiểu có giá trị thấp sẽ được chương trình tự động chuyển lên kiểu cao hơn cho phù hợp với phép toán. Cụ thể phép chuyển kiểu có thể được thực hiện theo sơ đồ như sau:

char -> int -> long ->float -> double

Ví dụ 2.12

```
int i = 3;
float f;
f = i + 2;
```

Trong ví dụ trên i có kiểu nguyên và vì vậy i+2 cũng có kiểu nguyên trong khi f có kiểu thực. Tuy vậy phép toán gán này là hợp lệ vì chương trình sẽ tự động chuyển kiểu của i+2 (bằng 5) sang kiểu thực (bằng 5.0) rồi mới gán cho f.

b) Ép kiểu: trong chuyển kiểu tự động, chương trình chuyển các kiểu từ thấp đến cao, tuy nhiên chiều ngược lại thì có thể gây mất dữ liệu. Do đó nếu cần thiết người lập trình phải ra lệnh cho chương trình.

Cú pháp tổng quát như sau:

(tên_kiểu)biểu_thức //Cú pháp cũ trong C

hoặc:

tên_kiểu(biểu_thức) //Cú pháp mới trong C++

Phép ép kiểu từ một số thực về số nguyên sẽ cắt bỏ tất cả phần thập phân của số thực, chỉ để lại phần nguyên. Như vậy để tính phần nguyên của một số thực x ta chỉ cần ép kiểu của x về thành kiểu nguyên, có nghĩa int(x) là phần nguyên của số thực x bất kỳ. Ví dụ để kiểm tra một số nguyên n có phải là số chính phương, ta cần tính căn bậc hai của n. Nếu căn bậc hai x của n là số nguyên thì n là số chính phương, tức nếu int(x) = x thì x nguyên và n là chính phương, ví dụ:

```
int n = 10;
float x = sqrt(n); //Hàm sqrt(n) tính căn bậc hai của n
if (int(x) == x)
 cout<<"n chính phương";
else
 cout<<"n không chính phương";
```

Để biết mã ASCII của một kí tự ta chỉ cần chuyển kí tự đó sang kiểu nguyên.

```
char c;
cin >> c;
```

```
cout<<"Mã của kí tự vừa nhập là " << int(c);
```

Xét ví dụ sau:

```
int i = 3, j = 5;
float x;
x = i / j * 10;       //x = 6?
cout<<x;
```

Trong ví dụ này mặc dù x được khai báo là thực nhưng kết quả in ra sẽ là 0 thay vì 6 như mong muốn. Lý do là vì phép chia giữa 2 số nguyên i và j sẽ cho lại số nguyên, tức i/j = 3/5 = 0. Từ đó x = 0*10 = 0. Để phép chia ra kết quả thực ta cần phải ép kiểu hoặc i hoặc j hoặc cả 2 thành số thực, khi đó phép chia sẽ cho kết quả thực và x được tính đúng giá trị. Cụ thể câu lệnh x = i/j*10 được đổi thành:

```
x = float(i) / j * 10;          //đúng
x = i / float(j) * 10;          //đúng
x = float(i) / float(j) * 10;   //đúng
x = float(i/j) * 10;            //sai
```

Phép ép kiểu: x = float(i/j) * 10; vẫn cho kết quả sai vì trong dấu ngoặc phép chia i/j vẫn là phép chia nguyên, kết quả x vẫn là 0.

2.3. Biểu thức, câu lệnh và các phép toán

2.3.1. Biểu thức

Biểu thức là một sự kết hợp giữa các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó. Mỗi biểu thức có sẽ có một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức.

Trong C++, ta có hai khái niệm về biểu thức :

- Biểu thức gán
- Biểu thức điều kiện

Biểu thức được phân loại theo kiểu giá trị : nguyên và thực. Trong các mệnh đề logic, biểu thức được phân thành đúng (giá trị khác 0) và sai (giá trị bằng 0).

Biểu thức thường được dùng trong:

- Vế phải của câu lệnh gán.
- Làm tham số thực sự của hàm.
- Làm chỉ số.
- Trong các toán tử của các cấu trúc điều khiển.

Ta đã có hai khái niệm chính tạo nên biểu thức đó là toán hạng và phép toán. Toán hạng gồm : hằng, biến, phần tử mảng và hàm.

Lệnh gán:

Biểu thức gán là biểu thức có dạng :

v=e;

Trong đó v là một biến (hay phần tử mảng), e là một biểu thức. Giá trị của biểu thức gán là giá trị của e, kiểu của nó là kiểu của v. Nếu đặt dấu ; vào sau biểu thức gán ta sẽ thu được phép toán gán có dạng:

v=e;

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác. Ví dụ như khi ta viết

a=b=5;

thì điều đó có nghĩa là gán giá trị của biểu thức b=5 cho biến a. Kết qủa là b=5 và a=5.

Hoàn toàn tương tự như:

a=b=c=d=6; gán 6 cho cả a, b, c và d

Ví dụ:

z=(y=2)*(x=6); // ở đây * là phép toán nhân

Gán 2 cho y, 6 cho x và nhân hai biểu thức lại cho ta z=12.

Lệnh gán mở rộng:

Một đặc tính của ngôn ngữ C++ làm cho nó nổi tiếng là một ngôn ngữ súc tích chính là các toán tử gán phức hợp cho phép chỉnh sửa giá trị của một biến với một trong những toán tử cơ bản sau:

```
    value += increase; tương đương với value = value +
increase;
    a -= 5; tương đương với a = a - 5;
    a /= b; tương đương với a = a / b;
    price *= units + 1; tương đương với price = price *
(units + 1);
```

và tương tự cho tất cả các toán tử khác.

2.3.2. Các phép toán

C++ có rất nhiều phép toán loại 1 ngôi, 2 ngôi và thậm chí cả 3 ngôi. Các thành phần tên gọi tham gia trong phép toán được gọi là hạng thức hoặc toán hạng, các kí hiệu phép toán được gọi là toán tử. Ví dụ trong phép toán a + b; a, b được gọi là toán hạng và + là toán tử. Phép toán 1 ngôi là phép toán chỉ có một toán hạng, ví dụ -a (đổi dấu số a), &x (lấy địa chỉ của biến x),… Một số kí hiệu phép toán cũng được sử dụng chung cho cả 1 ngôi lẫn 2 ngôi (hiển nhiên với ngữ nghĩa khác nhau).

*a) Các phép toán số học: +, -, *, /, %*

Các phép toán + (cộng), (trừ), * (nhân) được hiểu theo nghĩa thông thường trong số học.

Phép toán a / b (chia) được thực hiện theo kiểu của các toán hạng, tức nếu cả hai toán hạng là số nguyên thì kết quả của phép chia chỉ lấy phần nguyên, ngược lại nếu 1 trong 2 toán hạng là thực thì kết quả là số thực. Ví dụ:

Kết quả bằng 2 do 13 và 5 là 2 số nguyên:

```
13/5 = 2
```

Kết quả bằng 2.6 do có ít nhất 1 toán hạng là thực:

```
13.0/5 = 13/5.0 = 13.0/5.0 = 2.6
```

Phép toán a % b (lấy phần dư) trả lại phần dư của phép chia a/b, trong đó a và b là 2 số nguyên. Ví dụ:

```
13%5 = 3   //Phần dư của 13/5
5%13 = 5   //Phần dư của 5/13
```

b) Các phép toán tự tăng, giảm: i++, ++i, i--, --i

Phép toán ++i và i++ sẽ cùng tăng i lên 1 đơn vị tức tương đương với câu lệnh i = i+1. Tuy nhiên nếu 2 phép toán này nằm trong câu lệnh hoặc biểu thức thì ++i khác với i++. Cụ thể ++i sẽ tăng i, sau đó i mới được tham gia vào tính toán trong biểu thức. Ngược lại i++ sẽ tăng i sau khi biểu thức được tính toán xong (với giá trị i cũ). Điểm khác biệt này được minh hoạ thông qua ví dụ sau, giả sử i = 3, j = 15.

Phép toán	**Tương đương**	**Kết quả**
i = ++j;	= j + 1; i = j;	i= 16, j = 16
i= j++;	i = j; j = j + 1;	i=15, j = 16
j = ++i + 5;	i = i + 1; j = i + 5;	i=4, j = 9
j = i++ + 5;	j = i + 5; i = i + 1;	i=4, j = 8

c) Các phép toán so sánh và lôgic

Đây là các phép toán mà giá trị trả lại là đúng hoặc sai. Nếu giá trị của biểu thức là đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của 2 khái niệm "đúng", "sai". Mở rộng hơn C++ quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai".

Các phép toán so sánh:

== (bằng nhau), != (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Hai toán hạng của các phép toán này phải cùng kiểu. Ví dụ:

```
3 == 3 hoặc 3 == (4 -1) //nhận giá trị 1 vì đúng
3 == 5 //= 0 vì sai
3 != 5 //= 1
3 + (5 < 2) //= 3 vì 5<2 bằng 0
3 + (5 >= 2) //= 4 vì 5>=2 bằng 1
```

Chú ý: cần phân biệt phép toán gán (=) và phép toán so sánh (==). Phép gán vừa gán giá trị cho biến vừa trả lại giá trị bất kỳ (là giá trị của toán hạng bên phải), trong khi phép so sánh luôn luôn trả lại giá trị 1 hoặc 0.

Các phép toán lôgic: && (và), || (hoặc), ! (không, phủ định)

Hai toán hạng của loại phép toán này phải có kiểu lôgic tức chỉ nhận một trong hai giá trị "đúng" (được thể hiện bởi các số nguyên khác 0) hoặc "sai" (thể hiện bởi 0). Khi đó giá trị trả lại của phép toán là 1 hoặc 0 và được cho trong bảng sau:

a	**b**	**a && b**	**a || b**	**! a**
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Tóm lại:

Phép toán "và" đúng khi và chỉ khi hai toán hạng cùng đúng

Phép toán "hoặc" sai khi và chỉ khi hai toán hạng cùng sai

Phép toán "không" (hoặc "phủ định") đúng khi và chỉ khi toán hạng của nó sai.

Ví dụ 2.13

```
7 || 0              //có giá trị bằng 1
3<7  && 8>6         //có giá trị bằng 1
4 && 6              //có giá trị bằng 1
```

d) Các phép tính theo bit

Toán tử bit coi các toán hạng của nó như một tập hợp các bit. Mỗi bit có thể chứa hoặc giá trị 0 (tắt) hay 1 (bật). Toán tử bit cho phép người lập trình kiểm thử và đặt giá trị cho từng bit hay nhóm bit. Toán hạng của toán tử bit phải có kiểu nguyên. Việc dùng toán hạng không dấu làm cho chương trình phù hợp hơn với mọi cài đặt.

Bảng sau nói về các phép toán bit:

Toán tử	Chức năng	Cách dùng
~	Phủ định NOT	~bt
<<	dịch chuyển trái	bt1 << bt2
>>	dịch chuyển phải	bt1>>bt2
&	Và bit AND	bt1&bt2
^	hoặc bit XOR	bt1^bt2
|	hoặc bit OR	bt1|bt2

Toán tử bit NOT lật lại giá trị từng bit của toán hạng. Mỗi bit 1 đc đặt thành 0 và ngược lại.

Ví dụ 2.14

```
unsigned char bits = 0227;
 1   0   0   1   0   1   1   1
Bits  =   ~  Bits;
 0   1   1   0   1   0   0   0
```

Các phép toán dịch chuyển bit sẽ chuyển các bit của toán hạng bên trái đi một số vị trí hoặc sang trái hoặc phải.

Ví dụ 2.15

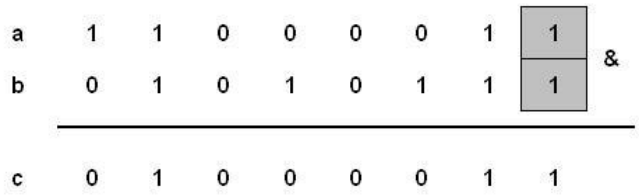
```
unsigned char bits = 1;
 0   0   0   0   0   0   0   1
Bits  =  Bits  <<  1;
 0   0   0   0   0   0   1   0
```

Ví dụ 2.16

```
unsigned char bits = 7;
 0   0   0   0   0   1   1   1
Bits  =  Bits  >>  2;
 0   0   0   0   0   0   0   1
```

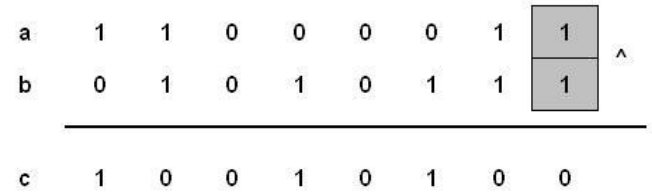
Toán tử AND bit nhận hai toán hạng, với mỗi vị trí bit, kết quả cho lại là bit 1 nếu cả hai toán hạng đều chứa bit 1, ngược lại kết quả là bit 0 (các bạn chú ý không nhầm toán tử này với toán tử AND logic (&&)).

Ví dụ 2.17



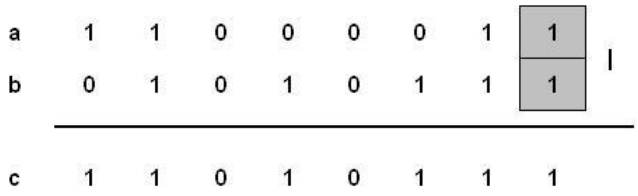
Toán tử XOR bit nhân hai toán hạng, với mỗi vị trí bit, kết quả cho lại là bit 1 nếu một trong hai nhưng không đồng thời cả hai toán hạng có chứa bit 1, ngược lại kết quả là bit 0.

Ví dụ 2.18



Toán tử OR bit nhận hai toán hạng, với mỗi vị trí bit, kết quả cho lại là bit 1 nếu một trong hai toán hạng có chứa bit 1, ngược lại kết quả là bit 0.

Ví dụ 2.19



e) Toán tử sizeof

Toán tử này có một tham số, đó có thể là một kiểu dữ liệu hay là một biến và trả về kích cỡ bằng byte của kiểu hay đối tượng đó.

```
a = sizeof (char);
```

a sẽ mang giá trị 1 vì kiểu char luôn có kích cỡ 1 byte trên mọi hệ thống. Giá trị trả về của sizeof là một hằng số vì vậy nó luôn luôn được tính trước khi chương trình thực hiện.

f) Toán tử điều kiện.

Toán tử điều kiện tính toán một biểu thức và trả về một giá trị khác tuỳ thuộc vào biểu thức đó là đúng hay sai.

Cú pháp:

<điều kiện>?<kết quả 1>: <kết quả 2>

Nếu condition là **true** thì giá trị trả về sẽ là result1, nếu không giá trị trả về là result2.

```
7==5?4: 3          //Trả về 3 vì 7 không bằng 5.
7==5+2?4: 3        //Trả về 4 vì 7 bằng 5+2.
5>3?a: b           //Trả về a, vì 5 lớn hơn 3.
a>b?a: b           //Trả về giá trị lớn hơn, a hoặc b.
```

g) Toán tử dãy

Mỗi câu lệnh C++ được kết thúc bằng dấu. Tuy nhiên khái niệm một biểu thức trong C++ được mở rộng hơn, nó được kết thúc bằng dấu chấm phẩy song nó có thể chứa một dãy các lệnh, dãy biểu thức tính toán độc lập chứ không phải chỉ có một biểu thức tính một giá trị.

Ví dụ 2.20

```
a*b, i+j;
```

Là một biểu thức tính a*b như là một giá trị, sau đó tính một giá trị khác là i+j. Hai tính toán này được đặt cách nhau bằng một dấu phẩy chứ không phải là dấu chấm phẩy.

2.3.3. Thứ tự ưu tiên các phép toán

Các phép toán có độ ưu tiên khác nhau, điều này có ý nghĩa trong cùng một biểu thức sẽ có một số phép toán này được thực hiện trước một số phép toán khác.

Thứ tự ưu tiên của các phép toán được trình bày trong bảng sau:

Thứ tự	Phép toán	Trình tự kết hợp
1	::	Trái qua phải
2	() [] . -> ++ -- dynamic_cast static_cast reinterpret_cast const_cast typeid	Trái qua phải
3	++ -- ~ ! sizeof new delete	Phải qua trái
	* &	
	+ -	
4	(type)	Phải qua trái
5	.* ->*	Trái qua phải
6	* / %	Trái qua phải
7	+ -	Trái qua phải
8	<< >>	Trái qua phải
9	< > <= >=	Trái qua phải
10	== !=	Trái qua phải
11	&	Trái qua phải

12	^	Trái qua phải
13	|	Trái qua phải
14	&&	Trái qua phải
15	||	Trái qua phải
16	?:	Phải qua trái
17	= *= /= %= += -= >>= <<= &= ^= |=	Phải qua trái
18	,	Trái qua phải

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, ~, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái. Ví dụ theo mức ưu tiên đã qui định, biểu thức tính x trong ví dụ trên sẽ được tính như x = 3 + (4 * 2) + 7 = 18. Phần lớn các trường hợp muốn tính toán theo một trật tự nào đó ta nên sử dụng cụ thể các dấu ngoặc (vì các biểu thức trong dấu ngoặc được tính trước).

2.3.4. Câu lệnh

Một câu lệnh trong C++ được thiết lập từ các từ khoá và các biểu thức … và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ vào/ra hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
cin >> x >> y;
x = 3 + x; y = (x = sqrt(x)) + 1; cout<<x;
cout<<y;
```

Lệnh hợp thành hay lệnh ghép (Compound statement)

Một dãy các câu lệnh được bao bởi các dấu { } gọi là một khối lệnh. Ví dụ:

```
{
      a=2;
      b=3;
      printf("\n%6d%6d",a,b);
}
```

C++ xem khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác, chỗ nào viết được một câu lệnh thì ở đó cũng có quyền đặt một khối lệnh.

Sự lồng nhau của các khối lệnh và phạm vi hoạt động của các biến:

Bên trong một khối lệnh lại có thể viết lồng khối lệnh khác. Sự lồng nhau theo cách như vậy là không hạn chế.

Khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại

trong thời gian máy làm việc bên trong khối lệnh và chúng lập tức biến mất ngay sau khi máy ra khỏi khối lệnh.

Vậy giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó.

Ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh

Nếu bên trong một khối ta dùng một biến hay một mảng có tên là a thì điều này không làm thay đổi giá trị của một biến khác cũng có tên là a (nếu có) được dùng ở đâu đó bên ngoài khối lệnh này.

Nếu có một biến đã được khai báo ở ngoài một khối lệnh và không trùng tên với các biến khai báo bên trong khối lệnh này thì biến đó cũng có thể sử dụng cả bên trong cũng như bên ngoài khối lệnh.

Ví dụ 2.21

Xét đoạn chương trình sau:

```
{
 int a=5,b=2;
 {
      int a=4;
      b=a+b;
      cout<<a << endl << b << endl;
 }
 cout<<a << endl << b << endl;
}
```

Khi đó đoạn chương trình sẽ in kết quả như sau:

```
a trong =4 b=6
a ngoài =5 b=6
```

Do tính chất biến a trong và ngoài khối lệnh.

2.3.5. Một số hàm số học

Tóm tắt một số các hàm toán học hay dùng. Các hàm này đều được khai báo trong file nguyên mẫu math.h.

- abs(x), labs(x), fabs(x): trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
- pow(x, y): hàm mũ, trả lại giá trị x lũy thừa y (xy).
- exp(x): hàm mũ, trả lại giá trị e mũ x (e^x).

- log(x), log10(x): trả lại lôgarit cơ số e và lôgarit thập phân của x (lnx, logx).
- sqrt(x): trả lại căn bậc 2 của x.
- atof(s_number): trả lại số thực ứng với số viết dưới dạng xâu kí tự s_number.
- sin(x), cos(x), tan(x): trả lại các giá trị sinx, cosx, tgx.

B. Phần thảo luận, bài tập

Bài 1. Nêu thứ tự thực hiện các phép toán trong biểu thức ở câu lệnh cout và cho biết kết quả in ra màn hình sau khi thực hiện chương trình sau:

```
#include <iostream.h>
void main(){
     cout<<(2+3*5/2-3<<1&5|7);
}
```

Bài 2. Nêu thứ tự thực hiện các phép toán trong biểu thức ở câu lệnh cout và cho biết kết quả in ra màn hình sau khi thực hiện chương trình sau:

```
#include <iostream.h>
void main(){
     cout<<(6^3||4+3-6&&7/3);
}
```

Bài 3. Nêu thứ tự thực hiện các phép toán trong biểu thức ở câu lệnh cout và cho biết kết quả in ra màn hình sau khi thực hiện chương trình sau:

```
#include <iostream.h>
void main(){
     int a=2,b=2;
     cout<<(--a-5+b++*4>>2&7);
}
```

Bài 4. Nêu tác dụng của từng câu lệnh trong hàm main và cho biết kết quả in ra màn hình sau khi thực hiện chương trình sau:

```
#include <iostream.h>
void main(){
     char *s="abcdefgh",*st=s;
     st+=4; *st+=4;
     s+=1; *s+=1;
     cout<<s;
}
```

Bài 5. Nêu tác dụng của từng câu lệnh trong hàm main và cho biết kết quả in ra màn hình sau khi thực hiện chương trình sau:

```
#include <iostream.h>
void main(){
     unsigned char c=200; float f=4.5;
     c+=100; f+=0.5;
     cout<<f/2+c/3;
}
```

CHƯƠNG 3. CÁC THAO TÁC XỬ LÝ INPUT/OUTPUT

A. Phần lý thuyết

3.1. Hàm in ra màn hình printf() và putchar() với các tham số

3.1.1. Hàm printf

Để in các giá trị bt_1, bt_2, …, bt_n ra màn hình theo một khuôn dạng mong muốn ta có thể sử dụng cú pháp sau đây:

printf(Chuỗi định dạng, bt_1, bt_2, ..., bt_n) ;

trong đó Chuỗi định dạng là một dãy kí tự đặt trong cặp dấu nháy kép (“”) qui định khuôn dạng cần in của các giá trị bt_1, bt_2, …, bt_n. Các bt_i có thể là các hằng, biến hay các biểu thức tính toán. Câu lệnh trên sẽ in giá trị của các bt_i này theo thứ tự xuất hiện của chúng và theo qui định được cho trong dòng định dạng.

Ví dụ, giả sử x = 4, câu lệnh:

printf(“%d %0.2f”, 3, x + 1) ;

sẽ in các số 3 và 5.00 ra màn hình, trong đó 3 được in dưới dạng số nguyên (được qui định bởi “%d”) và x + 1 (có giá trị là 5) được in dưới dạng số thực với 2 số lẻ thập phân (được qui định bởi “%0.2f”).

Các kí tự đi sau kí hiệu % dùng để định dạng việc in gồm có:

- d in số nguyên dưới dạng hệ thập phân
- o in số nguyên dạng hệ 8
- x, X in số nguyên dạng hệ 16
- u in số nguyên dạng không dấu
- c in kí tự
- s in xâu kí tự
- e, E in số thực dạng dấu phẩy động
- f in số thực dạng dấu phẩy tĩnh

Các kí tự trên phải đi sau dấu %. Các kí tự nằm trong dòng định dạng nếu không đi sau % thì sẽ được in ra màn hình. Muốn in % phải viết 2 lần (tức %%).

Ví dụ câu lệnh: printf(“Tỉ lệ học sinh giỏi: %0.2f %%”, 32.486) ;

sẽ in câu “Tỉ lệ học sinh giỏi: “, tiếp theo sẽ in số 32.486 được làm tròn đến 2 số lẻ thập phân lấp vào vị trí của “%0.2f”, và cuối cùng sẽ in dấu “%” (do có %% trong dòng định dạng). Câu được in ra màn hình sẽ là:

Tỉ lệ học sinh giỏi: 32.49%

Chú ý: Mỗi bt_i cần in phải có một định dạng tương ứng trong dòng định dạng.

Ví dụ câu lệnh trên cũng có thể viết:

printf(“%s %0.2f”, “Tỉ lệ học sinh giỏi: “, 32.486);

Trong câu lệnh này có 2 biểu thức cần in. Biểu thức thứ nhất là xâu kí tự “Tỉ lệ học sinh giỏi:” được in với khuôn dạng %s (in xâu kí tự) và biểu thức thứ hai là 32.486 được in với khuôn dạng %0.2f (in số thực với 2 số lẻ phần thập phân).

Nếu giữa kí tự % và kí tự định dạng có số biểu thị độ rộng cần in thì giá trị in ra sẽ được gióng cột sang lề phải, để trống các dấu cách phía trước. Nếu độ rộng âm (thêm dấu trừ − phía trước) sẽ gióng cột sang lề trái. Nếu không có độ rộng hoặc độ rộng bằng 0 (ví dụ %0.2f) thì độ rộng được tự điều chỉnh đúng bằng độ rộng của giá trị cần in.

Dấu + trước độ rộng để in giá trị số kèm theo dấu (dương hoặc âm)

Trước các định dạng số cần thêm kí tự l (ví dụ ld, lf) khi in số nguyên dài long hoặc số thực với độ chính xác gấp đôi double.

Các ký tự điều khiển:

- \n sang dòng mới
- \b lùi lại 1 tab.
- \f sang trang mới
- \t dấu tab
- \' In ra dấu '
- \" In ra dấu "
- \\: In ra dấu \

Ví dụ 3.1

```
#include <stdio.h>
void main()
{
     int i = 2, j = 3 ;
```

```
    printf("Tổng 2 số nguyên:\ni + j = %d", i+j);
}
```

3.1.2. Hàm putchar()

Để đưa một ký tự ra màn hình, sử dụng hàm putchar() với cú pháp như sau:

putchar(ch);

Hàm này sẽ đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Hàm putchar() nằm trong thư viện stdio.h. Ví dụ:

```
putchar('A'); ----> in ra ký tự A.
```

3.2. Hàm đọc ký tự từ bàn phím

Hàm getch(): nhận 1 ký tự trực tiếp từ bộ đệm bàn phím và trả về ký tự nhận được. Hàm getch() nằm trong thư viện conio.h.

Hàm getchc(): nhận 1 ký tự trực tiếp từ bộ đệm bàn phím và hiển thị trên monitor. Hàm getch() nằm trong thư viện stdio.h.

3.3. Thực hiện Input/Output

Để xuất dữ liệu ra màn hình và nhập dữ liệu từ bàn phím, trong C++ vẫn có thể dùng hàm printf() và scanf(), ngoài ra trong C++ ta có thể dùng dòng xuất/nhập chuẩn để nhập/xuất dữ liệu thông qua hai biến đối tượng của dòng (stream object) là **cout** và **cin**.

3.3.1. Nhập dữ liệu

a) Toán tử >>

Toán tử >> được sử dụng như sau để đọc dữ liệu từ dòng cin.

Cú pháp:

cin>>biến 1>>biến 2>>...>>biến n;

Toán tử cin được định nghĩa trước như một đối tượng biểu diễn cho thiết bị vào chuẩn của C++ là bàn phím, cin được sử dụng kết hợp với toán tử trích >> để nhập dữ liệu từ bàn phím cho các biến 1, 2,..., N.

b) Nhập ký tự và chuỗi ký tự

Có thể dùng các phương thức sau (định nghĩa trong lớp istream) để nhập ký tự và chuỗi: cin.get cin.getline cin.ignore

- ***Phương thức get():***

Dạng 1: int cin.get();

Dùng để đọc một ký tự (kể cả khoảng trắng).

Ví dụ 3.2

```
#include <iostream.h>
void main()
{
     int a; //Khai báo biến a kiểu int
     a=cin.get();//Đọc giá từ bàn phím trả về mã ACSII của ký tự
     cout<<a;  //Hiển thị ký tự vừa gõ ra màn hình.
}
```

Dạng 2: istream& cin.get(char &ch);

Dùng để đọc một ký tự (kể cả khoảng trắng) và đặt vào một biến kiểu char được tham chiếu bởi ch.

Ví dụ 3.3

```
#include <iostream.h>
void main()
{
     char b[10];
     cin.get(b,10);
     cout<<b;
}
```

Dạng 3: istream& cin.get(char *str, int n, char d = '\n');

Dùng để đọc một dãy ký tự (kể cả khoảng trắng) và đưa vào vùng nhớ do str trỏ tới. Quá trình đọc kết thúc khi xảy ra một trong hai tình huống sau:

- Gặp ký tự giới hạn (cho trong d). Ký tự giới hạn mặc định là '\n'.
- Đã nhận đủ (n-1) ký tự.

Chú ý:

- Ký tự kết thúc chuỗi '\0' được bổ sung vào cuối chuỗi nhận được.
- Ký tự giới hạn vẫn còn lại trên dòng nhập để dành cho các lệnh nhận tiếp theo.
- Ký tự <Enter> còn lại trên dòng nhập có thể làm trôi phương thức get() dạng 3.

Ví dụ xét đoạn chương trình:

Ví dụ 3.4

```
#include <iostream.h>
void main()
{
     char hoten[25], diachi[50], quequan[30];
     cout<<"\nHo ten: ";
     cin.get(hoten,25);
     cout<<"\nDia chi: ";
     cin.get(diachi,50);
     cout<<"\nQue quan: ";
     cin.get(quequan,30);
     cout<<"\n" << hoten << " " << diachi << " " << quequan;
}
```

Đoạn chương trình dùng để nhập họ tên, dịa chỉ và quê quán. Nếu gõ vào Nguyen van X <Enter> thì câu lệnh get đầu tiên sẽ nhận được chuỗi "Nguyen van X" cất vào mảng hoten. Ký tự <Enter> còn lại sẽ làm trôi 2 câu lệnh get tiếp theo. Do đó câu lệnh cuối cùng sẽ chỉ in ra Nguyen van X.

Để khắc phục tình trạng trên, có thể dùng một trong các cách sau:

- Dùng phương thức get() dạng 1 hoặc dạng 2 để lấy ra ký tự <Enter> trên dòng nhập trước khi dùng get (dạng 3).
- Dùng phương thức ignore để lấy ra một số ký tự không cần thiết trên dòng nhập trước khi dùng get dạng 3. Phương thức này viết như sau:
  - ***cin.ignore(n); Lấy ra (loại ra hay loại bỏ) n ký tự trên dòng nhập.***

Như vậy để có thể nhập được cả quê quán và cơ quan, cần sửa lại đoạn chương trình trên như sau:

Ví dụ 3.5

```
#include <iostream.h>
void main()
{
     char hoten[25], diachi[50], quequan[30];
     cout<<"\nHo ten: ";
     cin.get(hoten,25);
     cin.ignore(1);
     cout<<"\nDia chi: ";
     cin.get(diachi,50);
     cin.ignore(1);
     cout<<"\nQue quan: ";
     cin.get(quequan,30);
     cin.ignore(1);
     cout<<endl<< hoten<< " " << diachi << " " << quequan;
}
```

➢ ***Phương thức getline()***

Phương thức getline để nhập một dãy ký tự từ bàn phím.

Cú pháp:

istream& cin.getline(char *str, int n, char d ='\n');

Ví dụ 3.6

```
#include <iostream>
using namespace std;
void main () {
     char name[256], title[256];
     cout<<"Enter your name: ";
     cin.getline (name,256);
     cout<<"Enter your favourite movie: ";
     cin.getline (title,256);
     cout<<name << "'s favourite movie is " << title;
}
```

Chú ý:

- Để nhập một chuỗi không quá n ký tự và lưu vào mảng một chiều a (kiểu char) có thể dùng hàm cin.get như sau: cin.get(a, n);
- Toán tử nhập cin>> sẽ để lại ký tự chuyển dòng '\n' trong bộ đệm. Ký tự này có thể làm trôi phương thức cin.get. Để khắc phục tình trạng trên cần dùng phương thức cin.ignore(1) để bỏ qua một ký tự chuyển dòng.
- Để sử dụng các loại toán tử và phương thức nói trên cần khai báo tập tin dẫn hướng iostream.h

3.3.2. Xuất dữ liệu

Cú pháp:

cout<<biểu thức 1 <<... << biểu thức N;

Trong đó cout được định nghĩa trước như một đối tượng biểu diễn cho thiết bị xuất chuẩn của C++ là màn hình, cout được sử dụng kết hợp với toán tử chèn << để hiển thị giá trị các biểu thức 1, 2,..., N ra màn hình.

3.4. Thiết lập khuôn dạng - Trình bày màn hình

3.4.1. Các phương thức định dạng

Phương thức int cout.width(): Cho biết độ rộng quy định hiện tại.

- ***Phương thức int cout.width(int n):*** Thiết lập độ rộng quy định mới là n và trả về độ rộng quy định trước đó.

Chú ý: Độ rộng quy định n chỉ có tác dụng cho một giá trị xuất. Sau đó C++ lại áp dụng độ rộng quy định bằng 0. Ví dụ với các câu lệnh:

```
int m=1234, n=56;
cout<<"\nAB";
cout.width(6);
cout<<m;
cout<<n;
```

Thì kết quả là:

```
AB 123456
```

Giữa B và số 1 có 2 dấu cách.

Phương thức int cout.precision(): Cho biết độ chính xác hiện tại (đang áp dụng để xuất các giá trị thức).

Phương thức int cout.precision(int n): Thiết lập độ chính xác sẽ áp dụng là n và cho biết độ chính xác trước đó. Độ chính xác được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh thiết lập độ chính xác mới.

Phương thức char cout.fill(): Cho biết ký tự bù đầy dòng hiện tại đang được áp dụng.

Phương thức char cout.fill(char ch): Quy định ký tự độn mới sẽ được dùng là ch và cho biết ký tự độn đang dùng trước đó. Ký tự độn được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh chọn ký tự độn mới.

Ví dụ 3.7

```
#include <iostream.h>
#include <conio.h>
void main()
{
     clrscr();
     float x=-3.1551, y=-23.45421;
     cout.precision(2);
     cout.fill('*');
     cout<<"\n";
     cout.width(8);
     cout<<x;
     cout<<"\n";
     cout.width(8);
     cout<<y;
}
```

Sau khi thực hiện, chương trình in ra màn hình 2 dòng sau:

```
***-3.16
**-23.45
```

3.4.2. Cờ định dạng

Mỗi cờ định dạng chứa trong một bit. Cờ có 2 trạng thái: Bật (on) – có giá trị 1, Tắt (off) – có giá trị 0. Các cờ có thể chứa trong một biến kiểu long. Trong tập tin iostream.h đã định nghĩa các cờ sau:

ios::left	ios::right	ios::internal
ios::dec	ios::oct	ios::hex
ios::fixed	ios::scientific	ios::showpos
ios::uppercase	ios::showpoint	ios::showbase

Có thể chia cờ định dạng thành các nhóm:

❖ **Nhóm 1 gồm các cờ căn lề:**

ios::left ios::right ios::internal

- **Cờ ios::left**: khi bật cờ ios::left thì giá trị in ra nằm bên trái vùng quy định, các ký tự độn nằm sau.

- **Cờ ios::right**: khi bật cờ ios::right thì giá trị in ra nằm bên phải vùng quy định, các ký tự độn nằm trước. Chú ý mặc định cờ ios::right bật.

- **Cờ ios::internal**: cờ ios::internal có tác dụng giống như cờ ios::right chỉ khác là dấu (nếu có) in đầu tiên.

Chương trình sau minh hoạ cách dùng các cờ căn lề:

Ví dụ 3.8

```
#include <iostream.h>
#include <conio.h>
void main()
{
     clrscr();
     float x=-87.1551, y=23.45421;
     cout.precision(2);
     cout.fill('*');
     cout.setf(ios::left); //bat co ios::left
     cout<<"\n";
     cout.width(8);
     cout<<x;
     cout<<"\n";
```

```
    cout.width(8);
    cout<<y;
    cout.setf(ios::right); //bat co ios::right
    cout<<"\n";
    cout.width(8);
    cout<<x;
    cout<<"\n";
    cout.width(8);
    cout<<y;
    cout.setf(ios::internal); //bat co ios::internal
    cout<<"\n";
    cout.width(8);
    cout<<x;
    cout<<"\n";
    cout.width(8);
    cout<<y;
}
```

Sau khi thực hiện chương trình in ra 6 dòng như sau:

```
-87.16**
23.45**
**-87.16
***23.45
-**87.16
***23.45
```

❖ **Nhóm 2 gồm các cờ định dạng số nguyên:**

ios::dec ios::oct ios::hex

- Khi ios::dec bật (mặc định): số nguyên được in dưới dạng cơ số 10
- Khi ios::oct bật: số nguyên được in dưới dạng cơ số 8
- Khi ios::hex bật: số nguyên được in dưới dạng cơ số 16

Chương trình sau minh hoạ cách dùng các cờ định dạng số:

Ví dụ 3.9

```
#include <iostream>
int main()
{
    cout.setf(ios::hex, ios::basefield);
    cout<<100;
}
```

❖ **Nhóm 3 gồm các cờ định dạng số thực:**

ios::fixed ios::scientific ios::showpoint

Mặc định: cờ ios::fixed bật (on) và cờ ios::showpoint tắt (off).

- Khi ios::fixed bật và cờ ios::showpoint tắt thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n = 4 thì

Số thực -87.1500 được in: -87.15

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 678

- Khi ios::fixed bật và cờ ios::showpoint bật thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được in ra đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n = 4 thì

Số thực -87.1500 được in: -87.1500

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 678.0000

- Khi ios::scientific bật và cờ ios::showpoint tắt thì số thực in ra dưới dạng khoa học. Số chữ số phần phân (sau dấu chấm) được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n=4 thì

Số thực -87.1500 được in: -87.15e+01

Số thực 23.45425 được in: 23.4543e+01

Số thực 678.0 được in: 678e+02

- Khi ios::scientific bật và cờ ios::showpoint bật thì số thực in ra dưới dạng mũ. Số chữ số phần phân (sau dấu chấm) của phần định trị được in đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n=4 thì

Số thực -87.1500 được in: -87.150e+01

Số thực 23.45425 được in: 23.4543e+01

Số thực 678.0 được in: 67800e+01

Chương trình sau minh hoạ cách dùng các cờ định dạng số thực:

Ví dụ 3.10

```
#include <iostream>
```

```
#include <iomanip>
int main()
{
     cout.setf(ios::hex, ios::basefield);
     cout<<100 << endl;
     cout.setf(ios::oct | ios::showbase | ios::fixed);
     cout<<123 << " " << 123.23 << endl;
     cout<<setiosflags(ios::showpos);
     cout<<setiosflags(ios::scientific);
     cout<<123 << " " << 123.23;
}
```

Kết quả:

```
64
123  123.230000
+123 +123.23
```

Để quy định số thực được hiển thị ra màn hình với p chữ số sau dấu chấm thập phân, ta sử dụng đồng thời các hàm sau:

```
//Bật cờ hiệu showpoint(p)
setiosflags(ios:: showpoint);
//Thiết lập p chữ số phần thập phân
setprecision(p);
```

Các hàm này cần đặt trong toán tử xuất như sau:

```
cout<<setiosflag(ios:: showpoint) << setprecision(p);
```

Với p là số chữ số phần thập phân (kể cả dấu chấm).

Câu lệnh trên sẽ có hiệu lực đối với tất cả các toán tử xuất tiếp theo cho đến khi gặp một câu lệnh định dạng mới.

Ví dụ 3.11

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
     float a=5.3333333;
     cout<<setiosflags(ios:: showpoint) << setprecision(3);
     cout<<a;
}
```

❖ **Nhóm 4 gồm các hiển thị:**

ios::uppercase ios::showpos ios::showbase

- Cờ ios::showpos:

+ Nếu cờ ios::showpos tắt (mặc định) thì dấu cộng không được in trước số dương.

+ Nếu cờ ios::showpos tắt thì dấu cộng được in trước số dương.

+ Cờ ios::showbase bật thì số nguyên hệ 8 được in bắt đầu bằng ký tự 0 và số nguyên hệ 16 được bắt đầu bằng các ký tự 0x.

Ví dụ nếu a = 40 thì:

Dạng in hệ 8 là: 050

Dạng in hệ 16 là 0x28

- Cờ ios::showbase: tắt (mặc định) thì không in 0 trước số nguyên hệ 8 và không 0x trước số nguyên hệ 16.

Ví dụ nếu a = 40 thì:

Dạng in hệ 8 là: 50

Dạng in hệ 16 là 28

- Cờ ios::uppercase:

+ Nếu cờ ios::uppercase bật thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ hoa.

+ Nếu cờ ios::uppercase tắt (mặc định) thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ thường.

Chương trình sau minh hoạ cách dùng các cờ định dạng số thực:

Ví dụ 3.12

```
#include <iostream>
int main()
{
     cout.setf(ios::showpoint|ios::uppercase|ios::scientific);
     cout<<100.0;
}
```

3.4.3. Các phương thức bật tắt cờ

Các phương thức này định nghĩa trong lớp ios.

- Phương thức long cout.setf(long f): Sẽ bật các cờ liệt kê trong f và trả về một giá trị long biểu thị các cờ đang bật.

- Phương thức long cout.unsetf(long f): Sẽ tắt các cờ liệt kê trong f và trả về một giá trị long biểu thị các cờ đang bật.

- Phương thức long cout.flags(long f): Có tác dụng giống như cout.setf(long).

- Phương thức long cout.flags(): Sẽ trả về một giá trị long biểu thị các cờ đang bật.

B. Phần thảo luận, bài tập

Bài 1. Viết chương trình đọc 2 số nguyên và in ra kết quả của phép (+), phép trừ (-), phép nhân (*), phép chia (/) ra màn hình. Nhận xét kết quả chia 2 số nguyên.

Bài 2. Viết chương trình nhập vào bán kính r của một hình tròn. Tính chu vi và diện tích của hình tròn theo công thức:

Chu vi CV = 2*Pi*r

Diện tích S = Pi*r*r

In các kết quả lên màn hình.

Bài 3. Viết chương trình nhập vào bán kính hình cầu, tính và in ra diện tích, thể tích của hình cầu đó. Hướng dẫn: S = 4*PI*R^2, và V = (4/3)*PI*R^3

Bài 4. Viết chương trình nhập vào một số a bất kỳ và in ra giá trị bình phương (a^2), lập phương (a^3) của a và giá trị a^4.

Bài 5. Viết chương trình nhập vào điểm ba môn Toán, Lý, Hóa của một học sinh. In ra điểm trung bình của học sinh đó với hai số lẻ thập phân.

Bài 6. Viết chương trình đảo ngược một số nguyên dương có đúng 3 chữ số.

CHƯƠNG 4. CÁC CẤU TRÚC ĐIỀU KHIỂN

A. Phần lý thuyết

Một chương trình thường không chỉ bao gồm các lệnh tuần tự nối tiếp nhau. Trong quá trình chạy nó có thể rẽ nhánh hay lặp lại một đoạn mã nào đó. Để làm điều này chúng ta sử dụng các cấu trúc điều khiển.

Cùng với việc giới thiệu các cấu trúc điều khiển chúng ta cũng sẽ phải biết tới một khái niệm mới: khối lệnh, đó là một nhóm các lệnh được ngăn cách bởi dấu chấm phẩy (;) nhưng được gộp trong một khối giới hạn bởi một cặp ngoặc nhọn: { và }.

Hầu hết các cấu trúc điều khiển mà chúng ta sẽ xem xét trong chương này cho phép sử dụng một lệnh đơn hay một khối lệnh làm tham số, tuỳ thuộc vào chúng ta có đặt nó trong cặp ngoặc nhọn hay không.

4.1. Cấu trúc if

Cấu trúc này được dùng khi một lệnh hay một khối lệnh chỉ được thực hiện khi một điều kiện nào đó thoả mãn.

Cú pháp:

if (Điều kiện) < Khối lệnh 1>;

Nếu <*Biểu thức*> đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện <*Khối lệnh 1*> và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua <*Khối lệnh 1*> mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Ví dụ, đoạn mã sau đây sẽ viết "x is 100" chỉ khi biến x chứa giá trị 100:

```
if (x == 100)
 cout<<"x is 100";
```

Nếu chúng ta muốn có hơn một lệnh được thực hiện trong trường hợp <điều kiện> là true chúng ta có thể chỉ định một khối lệnh bằng cách sử dụng một cặp ngoặc nhọn {}:

```
if (x == 100){
 cout<<"x is ";
 cout<<x;
}
```

Chúng ta cũng có thể chỉ định điều gì sẽ xảy ra nếu điều kiện không được thoả mãn bằng cách sử dụng từ khoá else.

Cú pháp:

if (Biểu thức)

<Khối lệnh 1>;

else

<Khối lệnh 2>;

Máy tính giá trị của ***(Biểu thức)***. Nếu ***(Biểu thức)*** đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện <***Khối lệnh 1***> và sau đó sẽ thực hiện các lệnh tiếp sau <***Khối lệnh 2***> trong chương trình. Nếu ***(Biểu thức)*** sai (biểu thức có giá trị bằng 0) thì máy bỏ qua <***Khối lệnh 1***> mà thực hiện <***Khối lệnh 2***> sau đó thực hiện tiếp các lệnh tiếp sau <***Khối lệnh 2***> trong chương trình.

Ví dụ 4.1: Viết chương trình nhập hai số nguyên a, b từ bàn phím. In lên mà hình số lớn nhất của hai số đó.

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
     int a,b,Max;
     cout<<"nhap a ="; cin>>a;
     cout<<"nhap b ="; cin>>b;
     Max=a;
     if (Max<b)
          Max=b;
     cout<<"So lon nhat Max = " << Max;
     getch();
}
```

Ví dụ 4.2: Viết chương trình giải phương trình bậc nhất một ẩn ax + b = 0

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
     float a, b, x;
     cout<<"nhap a ="; cin>>a;
     cout<<"nhap b ="; cin>>b;
     if (a == 0.0)
          if (b == 0.0)
               cout<<"Phuong trinh vo dinh" ;
```

```
            else
                  cout<<"Phuong trinh vo nghiem";
      else
      {
            x = -b/a;
            cout<<"Nghiem cua pt: x =" << x;
      }
}
```

Cấu trúc if + else có thể được móc nối để kiểm tra nhiều giá trị. Ví dụ sau đây sẽ kiểm tra xem giá trị chứa trong biến x là dương, âm hay bằng không.

```
if (x > 0)
      cout<<"x is positive";
else if (x < 0)
      cout<<"x is negative";
else
      cout<<"x is 0";
```

4.2. Cấu trúc switch

Cú pháp của lệnh switch hơi đặc biệt một chút. Mục đích của nó là kiểm tra một vài giá trị hằng cho một biểu thức, tương tự với những gì chúng ta làm ở đầu bài này khi liên kết một vài lệnh if và else if với nhau.

Cú pháp:

switch (Biểu thức nguyên)

{

case n1

<Khối lệnh 1>;

case n2

<Khối lệnh 2>;

.....

case nk

<Khối lệnh k>;

[default

<Khối lệnh k+1>;]

}

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch.

default là một thành phần không bắt buộc phải có trong thân của switch.

Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc ( ) như sau:

Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni.

Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau:

Khi có default máy sẽ nhảy tới câu lệnh sau nhãn default.

Khi không có default máy sẽ nhảy ra khỏi cấu trúc switch.

Chú ý

Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Hai đoạn mã sau là tương đương:

switch	**if-else tương đương**

`switch (x) {`
`case 1:`
`  cout<<"x = 1";`
`  break;`
`case 2:`
`  cout<<"x = 2";`
`  break;`
`default:`
`  cout<<"x=?";`
`}`
	`if (x == 1) {`
	`  cout<<"x = 1";`
	`  }`
	`else if (x == 2) {`
	`  cout<<"x = 2";`
	`  }`
	`else {`
	`  cout<<"x=?";`
	`}`

Ví dụ 4.3: Viết chương trình nhập vào một tháng của năm (tính theo dương lịch), tính xem tháng đó có bao nhiêu ngày rồi in kết quả lên màn hình

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
     int t, n;
     cout<<"nhap thang t ="; cin>>t;
     cout<<"nhap nam ="; cin>>n;
     switch (t)
     {
          case 1:
          case 3:
          case 5:
          case 7:
          case 8:
          case 10:
          case 12:
               cout<<"Thang " << t << " nam " << n << " co 31 ngay";
               break;
          case 4:
          case 6:
          case 9:
          case 11:
               cout<<"Thang " << t << " nam " << n << " co 30 ngay";
               break;
          case 2:
               if (n % 4 == 0)
                    cout<<"Thang 2 nam " << n << " co 29 ngay";
               else
                    cout<<"Thang 2 nam co 28 ngay";
               break;
     }
     getch();
}
```

4.3. Cấu trúc for

Cú pháp:

for (<khởi tạo>; <điều kiện>; <tăng, giảm biến điều khiển>) <khối lệnh>;

Và chức năng chính của nó là lặp lại *<khối lệnh>* chừng nào <điều kiện> còn mang giá trị đúng, như trong vòng lặp while. Nhưng thêm vào đó, for cung cấp chỗ dành cho lệnh khởi tạo và lệnh tăng (giảm) . Vì vậy vòng lặp này được thiết kế đặc biệt lặp lại một hành động với một số lần xác định.

Cách thức hoạt động của nó như sau:

1. <khởi tạo> được thực hiện. Nói chung nó đặt một giá trị ban đầu cho biến điều khiển. Lệnh này được thực hiện chỉ một lần.

2. <điều kiện> được kiểm tra, nếu nó là đúng vòng lặp tiếp tục còn nếu không vòng lặp kết thúc và <khối lệnh> được bỏ qua.

3. <khối lệnh> được thực hiện. Nó có thể là một lệnh đơn hoặc là một khối lệnh được bao trong một cặp ngoặc nhọn.

4. Cuối cùng, <tăng, giảm biến điều khiển> được thực hiện để tăng biến điều khiển và vòng lặp quay trở lại bước 2.

Ví dụ 4.4: Tính tổng: S = $1^2 + 2^2 + 3^2 + ... + n^2$ (n nguyên dương)

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
     int i, n;
     double    s=0;
     cout<<"Nhap n ="; cin>>n;
     for (i=1;i<=n;i++)
          s=s+i*i;
     cout<<"Tong S = " << s;
}
```

Ví dụ 4.5: Sau đây là một ví dụ đếm ngược sử dụng vòng for.

```
#include <iostream.h>
void main()
{
     for (int n=10; n>0; n--){
          cout<<n << ", ";
     }
}
```

Kết quả:

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
```

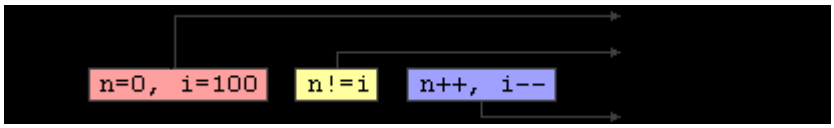
Phần khởi tạo và lệnh tăng không bắt buộc phải có. Chúng có thể được bỏ qua nhưng vẫn phải có dấu chấm phẩy ngăn cách giữa các phần. Vì vậy, chúng ta có thể viết for (;n<10;) hoặc for (;n<10;n++).

Bằng cách sử dụng dấu phẩy, chúng ta có thể dùng nhiều lệnh trong bất kì trường nào trong vòng for, như là trong phần khởi tạo. Ví dụ chúng ta có thể khởi tạo một lúc nhiều biến trong vòng lặp:

```
for (n=0, i=100; n!=i; n++, i--)
```

```
    {
          //Thân của vòng lặp
    }
```

Vòng lặp này sẽ thực hiện 50 lần nếu như n và i không bị thay đổi trong thân vòng lặp:



Ví dụ 4.6: Viết chương trình in hình chữ nhật như sau

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
     int m,n,i,j;
     cout<<"Nhap so cot m = "; cin>>m;
     cout<<"Nhap so hang n = "; cin>>n;
     for (i=1;i<=n;i++)
     {
          for (j=1;j<=m;j++)
               cout<<"A";
          cout<<"\n";
     }
}
```

4.4. Cấu trúc while

Cú pháp:

while (điều kiện) <khối lệnh>

Chức năng của nó đơn giản chỉ là lặp lại <khối lệnh> khi <điều kiện> còn thoả mãn.

Ví dụ 4.7: Viết một chương trình đếm ngược sử dụng vào lặp while:

```
#include <iostream.h>
void main()
```

```
{
	int n;
	cout<<"Nhap N >";
	cin >> n;
	while (n>0) {
		cout<<n << ", ";
		--n;
	}
}
```

Kết quả:

Nhap N >8
8, 7, 6, 5, 4, 3, 2, 1,

Khi chương trình chạy người sử dụng được yêu cầu nhập vào một số để đếm ngược. Sau đó, khi vòng lặp while bắt đầu nếu số mà người dùng nhập vào thoả mãn điều kiện điều kiện n>0 khối lệnh sẽ được thực hiện một số lần không xác định chừng nào điều kiện (n>0) còn được thoả mãn.

Chúng ta cần phải nhớ rằng vòng lặp phải kết thúc ở một điểm nào đó, vì vậy bên trong vòng lặp chúng ta phải cung cấp một phương thức nào đó để buộc <điều kiện> trở thành sai nếu không thì nó sẽ lặp lại mãi mãi. Trong ví dụ trên vòng lặp phải có lệnh --n; để làm cho <điều kiện> trở thành sai sau một số lần lặp.

Ví dụ 4.8: Nhập số tự nhiên n. Tính S = n!

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{
	int n,i,j;
	double	s=1;
	cout<<"Nhap n = "; cin>>n;
	i=1;
	if(n==0)
		s=1;
	else
		while(i<=n)
		{
			s=s*i;
			i=i+1;
		}
	cout<<"S = " << n << "! = " << s;
	getch();
}
```

4.5. Cấu trúc do while

Cú pháp:

do <khối lệnh> while (điều kiện);

Chức năng của nó là hoàn toàn giống vòng lặp while chỉ trừ có một điều là điều kiện điều khiển vòng lặp được tính toán sau khi <khối lệnh> được thực hiện, vì vậy <khối lệnh> sẽ được thực hiện ít nhất một lần ngay cả khi <điều kiện> không bao giờ được thoả mãn.

Ví dụ 4.9 : Chương trình dưới đây sẽ viết ra bất kì số nào mà bạn nhập vào cho đến khi bạn nhập số 0.

```
#include <iostream.h>
void main()
{
     unsigned long n;
     do {
          cout<<"Enter number (0 to end): ";
          cin >> n;
          cout<<"You entered: " << n << "\n";
     } while (n != 0);
}
```

Kết quả:

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

Vòng lặp do-while thường được dùng khi điều kiện để kết thúc vòng lặp nằm trong vòng lặp, như trong ví dụ trên, số mà người dùng nhập vào là điều kiện kiểm tra để kết thúc vòng lặp. Nếu bạn không nhập số 0 trong ví dụ trên thì vòng lặp sẽ không bao giờ chấm dứt.

4.6. Câu lệnh break

Sử dụng break chúng ta có thể thoát khỏi vòng lặp ngay cả khi điều kiện để nó kết thúc chưa được thoả mãn. Lệnh này có thể được dùng để kết thúc một vòng lặp không xác định hay buộc nó phải kết thúc giữa chừng thay vì kết thúc một cách bình thường. Ví dụ, chúng ta sẽ dừng việc đếm ngược trước khi nó kết thúc:

Ví dụ 4.10

```
#include <iostream.h>
void main()
{
     int n;
     for (n=10; n>0; n--) {
          cout<<n << ", ";
          if (n==3)
          {
               cout<<"countdown aborted!";
               break;
          }
     }
}
```

4.7. Câu lệnh continue

Lệnh continue làm cho chương trình bỏ qua phần còn lại của vòng lặp và nhảy sang lần lặp tiếp theo.

Ví dụ 4.11

```
#include <iostream.h>
void main()
{
     for (int n=10; n>0; n--)
     {
          if (n==5) continue;
          cout<<n << ", ";
     }
     cout<<"FIRE!";
}
```

B. Phần thảo luận, bài tập

Bài 1. Viết chương trình nhập vào số nguyên dương, in ra thông báo số chẵn hay lẻ.

Bài 2. Viết chương trình nhập vào 4 số thực a, b, c, d. Tìm và in ra số lớn nhất trong 4 số đó (sử dụng toán tử điều kiện, và cấu trúc if).

Bài 3. Viết chương trình giải phương trình bậc 2: $ax^2 + bx + c = 0$, với a, b, c nhập vào từ bàn phím (tính cả nghiệm phức).

Bài 4. Viết chương trình nhập vào tháng, in ra tháng đó có bao nhiêu ngày.

Bài 5. Viết chương trình nhập vào 2 số x, y và 1 trong 4 toán tử +, -, *, /. Nếu là + thì in ra kết quả x + y, nếu là – thì in ra x – y, nếu là * thì in ra x * y, nếu là / thì in ra x / y (nếu y = 0 thì thông báo không chia được).

Bài 7. Viết chương trình nhập vào 3 số thực a, b, c. Kiểm tra xem a, b, c có phải là 3 cạnh của tam giác không? Nếu là 3 cạnh của tam giác thì tính diện tích của tam giác theo công thức sau:

$s = \sqrt{p*(p-a)*(p-b)*(p-c)}$, với $p = (a+b+c)/2$

Hướng dẫn: a, b, c là 3 cạnh của tam giác phải thỏa điều kiện sau: (a + b) > c và (a + c) > b và (b + c) > a.

Bài 8. Viết chương trình tính giá trị của hàm f, với x là số thực được nhập từ bàn phím.

$$f(x) = \begin{cases} x^2 + x & x \le 0 \\ 0 & 0 < x \le 2 \\ \sin(x^2) & x > 2 \end{cases}$$

Bài 9. Viết chương trình tính tiền điện với chỉ số mới và chỉ số cũ được nhập vào từ bàn phím. In ra màn hình chỉ số cũ, chỉ số mới, và số tiền phải trả. Biết rằng 100 kWh đầu giá 550, từ KWh 101 - 150 giá 1.110, từ KWh 151 - 200 giá 1.470, từ KWh 201 - 300 giá 1.600, từ KWh 301 - 400 giá 1.720, từ KWh 401 trở lên giá 1.780.

Bài 10. Viết chương trình nhập vào một số nguyên rồi in ra tất cả các ước số của số đó.

Bài 11. Viết chương trình nhập vào một số và kiểm tra xem số đó có phải là số nguyên tố hay không?

CHƯƠNG 5. HÀM TRONG C++

A. Phần lý thuyết

5.1. Hàm trong C++

Hàm là một khối lệnh được thực hiện khi nó được gọi từ một điểm khác của chương trình.

Cú pháp:

type <tên hàm> ([tham số 1], [tham số 2],...) <khối lệnh>;

Trong đó:

- type là kiểu dữ liệu được trả về của hàm.
- <tên hàm> là tên gọi của hàm.
- [tham số i] là các tham số (có nhiều bao nhiêu cũng được tuỳ theo nhu cầu).

Một tham số bao gồm tên kiểu dữ liệu sau đó là tên của tham số giống như khi khai báo biến (ví dụ int x) và đóng vai trò bên trong hàm như bất kì biến nào khác. Chúng dùng để truyền tham số cho hàm khi nó được gọi. Các tham số khác nhau được ngăn cách bởi các dấu phẩy.

- <khối lệnh> là thân của hàm. Nó có thể là một lệnh đơn hay một khối lệnh.

Ví dụ 5.1

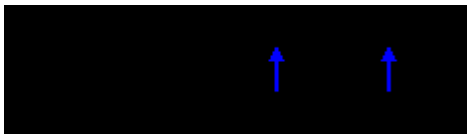
Dưới đây là ví dụ đầu tiên về hàm

```
#include <iostream.h>
int addition (int a, int b)
{
     int r;
     r=a+b;
     return (r);
}
void main ()
{
     int z;
     z = addition (5,3);
     cout<<"z = " << z;
}
```

Kết quả:

z = 8

Chúng ta có thể thấy hàm main bắt đầu bằng việc khai báo biến z kiểu int. Ngay sau đó là một lời gọi tới hàm addition. Nếu để ý chúng ta sẽ thấy sự tương tự giữa cấu trúc của lời gọi hàm với khai báo của hàm:



Các tham số có vai trò thật rõ ràng. Bên trong hàm main chúng ta gọi hàm addition và truyền hai giá trị: 5 và 3 tương ứng với hai tham số int a và int b được khai báo cho hàm addition.

Vào thời điểm hàm được gọi từ main, quyền điều khiển được chuyển sang cho hàm addition. Giá trị của c hai tham số (5 và 3) được copy sang hai biến cục bộ int a và int b bên trong hàm.

Dòng lệnh sau:

```
return (r);
```

kết thúc hàm addition, và trả lại quyền điều khiển cho hàm nào đã gọi nó (main) và tiếp tục chương trình ở cái điểm mà nó bị ngắt bởi lời gọi đến addition. Nhưng thêm vào đó, giá trị được dùng với lệnh return (r) chính là giá trị được trả về của hàm.

Giá trị trả về bởi một hàm chính là giá trị của hàm khi nó được tính toán. Vì vậy biến z sẽ có có giá trị được trả về bởi addition (5, 3), đó là 8.

Phạm vi hoạt động của các biến

Bạn cần nhớ rằng phạm vi hoạt động của các biến khai báo trong một hàm hay bất kì một khối lệnh nào khác chỉ là hàm đó hay khối lệnh đó và không thể sử dụng bên ngoài chúng. Trong chương trình ví dụ trên, bạn không thể sử dụng trực tiếp các biến a, b hay r trong hàm main vì chúng là các biến cục bộ của hàm addition. Thêm vào đó bạn cũng không thể sử dụng biến z trực tiếp bên trong hàm addition vì nó làm biến cục bộ của hàm main.

Tuy nhiên bạn có thể khai báo các biến toàn cục để có thể sử dụng chúng ở bất kì đâu, bên trong hay bên ngoài bất kì hàm nào. Để làm việc này bạn cần khai báo

chúng bên ngoài mọi hàm hay các khối lệnh, có nghĩa là ngay trong thân chương trình.

Ví dụ 5.2

Đây là một ví dụ khác về hàm:

```
#include <iostream.h>
int subtraction (int a, int b)
{
     int r;
     r=a-b;
     return (r);
}
int main ()
{
     int x=5, y=3, z;
     z = subtraction (7,2);
     cout<<"Ket qua 1: "<<z<<'\n';
     cout<<"Ket qua 2: "<<subtraction(7,2)<<'\n';
     cout<<"Ket qua 3: "<<subtraction (x,y)<<'\n';
     z= 4 + subtraction (x,y);
     cout << "Ket qua 4: " << z << '\n';
     return 0;
}
```

Kết quả:

```
Ket qua 1: 5
Ket qua 2: 5
Ket qua 3: 2
Ket qua 4: 6
```

Trong trường hợp này chúng ta tạo ra hàm subtraction. Chức năng của hàm này là lấy hiệu của hai tham số rồi trả về kết quả.

Tuy nhiên, nếu phân tích hàm main các bạn sẽ thấy chương trình đã vài lần gọi đến hàm subtraction. Tôi đã sử dụng vài cách gọi khác nhau để các bạn thấy các cách khác nhau mà một hàm có thể được gọi.

Để có hiểu cặn kẽ ví dụ này bạn cần nhớ rằng một lời gọi đến một hàm có thể hoàn toàn được thay thế bởi giá trị của nó. Ví dụ trong lệnh gọi hàm đầu tiên:

```
z = subtraction (7,2);
cout<<"Ket qua 1: "<<z<<'\n';
```

Nếu chúng ta thay lời gọi hàm bằng giá trị của nó (đó là 5), chúng ta sẽ có:

```
z = 5;
cout<<"Ket qua 1: "<<z<<'\n';
```

Tương tự như vậy

```
cout<<"Ket qua 2: "<<subtraction(7,2)<<'\n';
```

Cũng cho kết quả giống như hai dòng lệnh trên nhưng trong trường hợp này chúng ta gọi hàm subtraction trực tiếp như là một tham số của cout. Chúng ta cũng có thể viết:

```
cout<<"Ket qua 2: "<<5<<'\n';
```

Vì 5 là kết quả của subtraction (7,2). Còn với lệnh

```
cout<<"Ket qua 3: " << subtraction (x,y)<<'\n';
```

Điều mới mẻ duy nhất ở đây là các tham số của subtraction là các biến thay vì các hằng. Điều này là hoàn toàn hợp lệ. Trong trường hợp này giá trị được truyền cho hàm subtraction là giá trị của x and y.

Trường hợp thứ tư cũng hoàn toàn tương tự. Thay vì viết

```
z = 4 + subtraction (x,y);
```

chúng ta có thể viết:

```
z = subtraction (x,y) + 4;
```

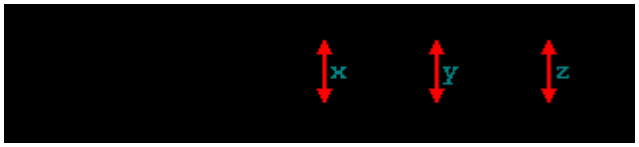
Cũng hoàn toàn cho kết quả tương đương.

5.2. Truyền tham số cho hàm

Cho đến nay, trong tất cả các hàm chúng ta đã biết, tất cả các tham số truyền cho hàm đều được truyền theo giá trị. Điều này có nghĩa là khi chúng ta gọi hàm với các tham số, những gì chúng ta truyền cho hàm là các giá trị chứ không phải bản thân các biến. Ví dụ, giả sử chúng ta gọi hàm addition như sau:

```
int x=5, y=3, z;
z = addition (x, y);
```

Trong trường hợp này khi chúng ta gọi hàm addition thì các giá trị 5 and 3 được truyền cho hàm, không phải là bản thân các biến.



Đến đây các bạn có thể hỏi tôi: Như vậy thì sao, có ảnh hưởng gì đâu? Điều đáng nói ở đây là khi các bạn thay đổi giá trị của các biến a hay b bên trong hàm thì các biến x và y vẫn không thay đổi vì chúng đâu có được truyền cho hàm chỉ có giá trị của chúng được truyền mà thôi.

Hãy xét trường hợp bạn cần thao tác với một biến ngoài ở bên trong một hàm. Vì vậy bạn sẽ phải truyền tham số dưới dạng tham số biến như ở trong hàm duplicate trong ví dụ dưới đây:

Ví dụ 5.3

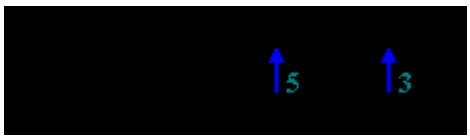
```
#include <iostream.h>
void duplicate (int& a, int& b, int& c)
{
     a*=2;
     b*=2;
     c*=2;
}
void main ()
{
     int x=1, y=3, z=7;
     duplicate (x, y, z);
     cout<<"x=" << x << ", y=" << y << ", z=" << z;
}
```

Kết quả:

```
x=2, y=6, z=14
```

Điều đầu tiên làm bạn chú ý là trong khai báo của duplicate theo sau tên kiểu của mỗi tham số đều là dấu và (&), để báo hiệu rằng các tham số này được truyền theo tham số biến chứ không phải tham số giá trị.

Khi truyền tham số dưới dạng tham số biến chúng ta đang truyền bản thân biến đó và bất kì sự thay đổi nào mà chúng ta thực hiện với tham số đó bên trong hàm sẽ ảnh hưởng trực tiếp đến biến đó.



Trong ví dụ trên, chúng ta đã liên kết a, b và c với các tham số khi gọi hàm (x, y và z) và mọi sự thay đổi với a bên trong hàm sẽ ảnh hưởng đến giá trị của x và hoàn toàn tương tự với b và y, c và z.

Kiểu khai báo tham số theo dạng tham số biến sử dụng dấu và (&) chỉ có trong C++. Trong ngôn ngữ C chúng ta phải sử dụng con trỏ để làm việc tương tự như thế.

Truyền tham số dưới dạng tham số biến cho phép một hàm trả về nhiều hơn một giá trị.

Ví dụ 5.4

Đây là một hàm trả về số liền trước và liền sau của tham số đầu tiên.

```
#include <iostream.h>
void prevnext (int x, int& prev, int& next)
{
     prev = x-1;
     next = x+1;
}
void main (){
     int x=100, y, z;
     prevnext (x, y, z);
     cout<<"Previous=" << y << ", Next=" << z;
}
```

Kết quả

```
Previous=99, Next=101
```

Giá trị mặc định của tham số.

Khi định nghĩa một hàm chúng ta có thể chỉ định những giá trị mặc định sẽ được truyền cho các đối số trong trường hợp chúng bị bỏ qua khi hàm được gọi. Để làm việc này đơn giản chỉ cần gán một giá trị cho đối số khi khai báo hàm. Nếu giá trị của tham số đó vẫn được chỉ định khi gọi hàm thì giá trị mặc định sẽ bị bỏ qua.

Ví dụ 5.5

```
//Giá trị mặc định trong hàm
#include <iostream.h>
int divide (int a, int b=2)
{
     int r;
     r=a/b;
     return (r);
}
void main ()
{
     cout<<divide (12);
     cout<<endl;
     cout<<divide (20,4);
}
```

Kết quả:

```
6
5
```

Nhưng chúng ta thấy trong thân chương trình, có hai lời gọi hàm divide. Trong lệnh đầu tiên:

```
divide (12)
```

Chúng ta chỉ dùng một tham số nhưng hàm divide cho phép đến hai. Bởi vậy hàm divide sẽ tự cho tham số thứ hai giá trị bằng 2 vì đó là giá trị mặc định của nó (chú ý phần khai báo hàm được kết thúc bởi int b=2). Vì vậy kết quả sẽ là 6 (12/2).

Trong lệnh thứ hai:

```
divide (20,4)
```

Có hai tham số, bởi vậy giá trị mặc định sẽ được bỏ qua. Kết quả của hàm sẽ là 5 (20/4).

5.3. Đệ quy

5.3.1. Khái niệm đệ qui

Một hàm gọi đến hàm khác là bình thường, nhưng nếu hàm lại gọi đến chính nó thì ta gọi hàm là đệ qui. Khi thực hiện một hàm đệ qui, hàm sẽ phải chạy rất nhiều lần, trong mỗi lần chạy chương trình sẽ tạo nên một tập biến cục bộ mới trên ngăn xếp (các đối, các biến riêng khai báo trong hàm) độc lập với lần chạy trước đó, từ đó dễ gây tràn ngăn xếp. Vì vậy đối với những bài toán có thể giải được bằng phương pháp lặp thì không nên dùng đệ qui.

Để minh hoạ ta hãy xét hàm tính n giai thừa. Để tính n! ta có thể dùng phương pháp lặp như sau:

Ví dụ 5.6

```
//Tính giai thừa
#include <iostream.h>
void main()
{
     int n;
     double kq = 1;
     cout<<"n = "; cin >> n;
     if(n==0)
          kq=1;
     else
          for (int i=1; i<=n; i++)
               kq *= i;
     cout<<n << "! = " << kq;
}
```

Mặt khác, n! giai thừa cũng được tính thông qua (n-1)! bởi công thức truy hồi

```
n! = 1          nếu n = 0
n! = (n-1)!n    nếu n > 0
```

Do đó ta có thể xây dựng hàm đệ qui tính n! như sau:

Ví dụ 5.7

//Tính giai thừa theo đệ qui

```
#include <iostream.h>
double gt(int n)
{
     if (n==0)
          return 1;
     else
          return gt(n-1)*n;
}
void main()
{
     int n;
     cout<<"n = "; cin >> n;
     cout<<gt(n);
}
```

Trong hàm main() giả sử ta nhập 3 cho n, khi đó để thực hiện câu lệnh cout<<gt(3) để in 3! đầu tiên chương trình sẽ gọi chạy hàm gt(3). Do 3 ≠ 0 nên hàm gt(3) sẽ trả lại giá trị gt(2)*3, tức lại gọi hàm gt với tham đối thực sự ở bước này là n = 2. Tương tự gt(2) = gt(1)*2 và gt(1) = gt(0)*1. Khi thực hiện gt(0) ta có đối n = 0 nên hàm trả lại giá trị 1, từ đó gt(1) = 1*1 = 1 và suy ngược trở lại ta có gt(2) = gt(1)*2 = 1*2 = 2, gt(3) = gt(2)*3 = 2*3 = 6, chương trình in ra kết quả 6.

Từ ví dụ trên ta thấy hàm đệ qui có đặc điểm: Chương trình viết rất gọn, việc thực hiện gọi đi gọi lại hàm rất nhiều lần phụ thuộc vào độ lớn của đầu vào. Chẳng hạn trong ví dụ trên hàm được gọi n lần, mỗi lần như vậy chương trình sẽ mất thời gian để lưu giữ các thông tin của hàm gọi trước khi chuyển điều khiển đến thực hiện hàm được gọi. Mặt khác các thông tin này được lưu trữ nhiều lần trong ngăn xếp sẽ dẫn đến tràn ngăn xếp nếu n lớn. Tuy nhiên, đệ qui là cách viết rất gọn, dễ viết và đọc chương trình, mặt khác có nhiều bài toán hầu như tìm một thuật toán lặp cho nó là rất khó trong khi viết theo thuật toán đệ qui thì lại rất dễ dàng.

5.3.2. Lớp các bài toán giải được bằng đệ qui

Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:

- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp suy biến hay cơ sở, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó,

- Đối với trường hợp tổng quát, bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham đối khác có kích thước nhỏ hơn tham đối ban đầu. Và sau một

số bước hữu hạn biến đổi cùng dạng, bài toán đưa được về trường hợp suy biến. Như vậy trong trường hợp tính n! nếu n = 0 hàm cho ngay giá trị 1 mà không cần phải gọi lại chính nó, đây chính là trường hợp suy biến. Trường hợp n > 0 hàm sẽ gọi lại chính nó nhưng với n giảm 1 đơn vị. Việc gọi này được lặp lại cho đến khi n=0.

Một lớp rất rộng của bài toán dạng này là các bài toán có thể định nghĩa được dưới dạng đệ qui như các bài toán lặp với số bước hữu hạn biết trước, các bài toán UCLN, tháp Hà Nội,...

5.3.3. Cấu trúc chung của hàm đệ qui

Cú pháp:

if (trường hợp suy biến)
{
trình bày cách giải //Giả định đã có cách giải
}
else //Trường hợp tổng quát
{
gọi lại hàm với tham đối "bé" hơn
}

Ví dụ 5.8

Tìm UCLN của 2 số a, b. Bài toán có thể được định nghĩa dưới dạng đệ qui như sau:

- nếu a = b thì UCLN = a
- nếu a > b thì UCLN(a, b) = UCLN(a-b, b)
- nếu a < b thì UCLN(a, b) = UCLN(a, b-a) Từ đó ta có chương trình đệ qui để tính UCLN của a và b như sau.

```
//Hàm tìm ước số chung lớn nhất của hai số a,b
#include <iostream.h>
int UCLN(int a, int b)   //qui uoc a, b > 0
{
     if (a < b) UCLN(a, b-a);
     if (a == b) return a;
     if (a > b) UCLN(a-b, b);
}
void main()
{
```

```
     int x,y;
     cout<<"X="; cin>>x;
     cout<<"Y="; cin>>y;
     cout<<"Uoc chung lon nhat: "<<UCLN(x,y);
}
```

Ví dụ 5.9

Tính số hạng thứ n của dãy Fibonaci là dãy f(n) được định nghĩa:

- f(0) = f(1) = 1
- f(n) = f(n-1) + f(n-2) với n ≥ 2.

```
//Hàm tính dãy Fibonaci
#include <iostream.h>
long Fib(int n)
{
     long kq;
     if (n==0 || n==1)
          kq = 1;
     else
          kq = Fib(n-1) + Fib(n-2);
     return kq;
}
void main()
{
     int n;
     cout<<"Nhap N="; cin>>n;
     cout<<"Fibonaci = "<<Fib(n);
}
```

5.4. Hàm inline

Chỉ thị inline có thể được đặt trước khao báo của một hàm để chỉ rõ rằng lời gọi hàm sẽ được thay thế bằng mã lệnh của hàm khi chương trình được dịch. Việc này tương đương với việc khai báo một macro, lợi ích của nó chỉ thể hiện với các hàm rất ngắn, tốc độ chạy chương trình sẽ được cải thiện vì nó không phải gọi một thủ tục con.

Cú pháp:

inline type <tên hàm> (<tham số>)

{

<thân của hàm>

}

Lời gọi hàm cũng như bất kì một hàm nào khác. Không cần thiết phải đặt từ khoá inline trong lệnh gọi, chỉ cần trong lời khai báo hàm là đủ.

5.5. Hàm tải bội

Hai hàm có thể có cũng tên nếu khai báo tham số của chúng khác nhau, điều này có nghĩa là bạn có thể đặt cùng một tên cho nhiều hàm nếu chúng có số tham số khác nhau hay kiểu dữ liệu của các tham số khác nhau (hay thậm chí là kiểu dữ liệu trả về khác nhau).

Ví dụ 5.10

```
#include <iostream.h>
int divide (int a, int b)
{
     return (a/b);
}
float divide (float a, float b)
{
     return (a/b);
}
void main ()
{
     int x=5,y=2;
     float n=5.0,m=2.0;
     cout<<divide (x,y);
     cout<<"\n";
     cout<<divide (n,m);
}
```

Kết quả:

```
2
2.5
```

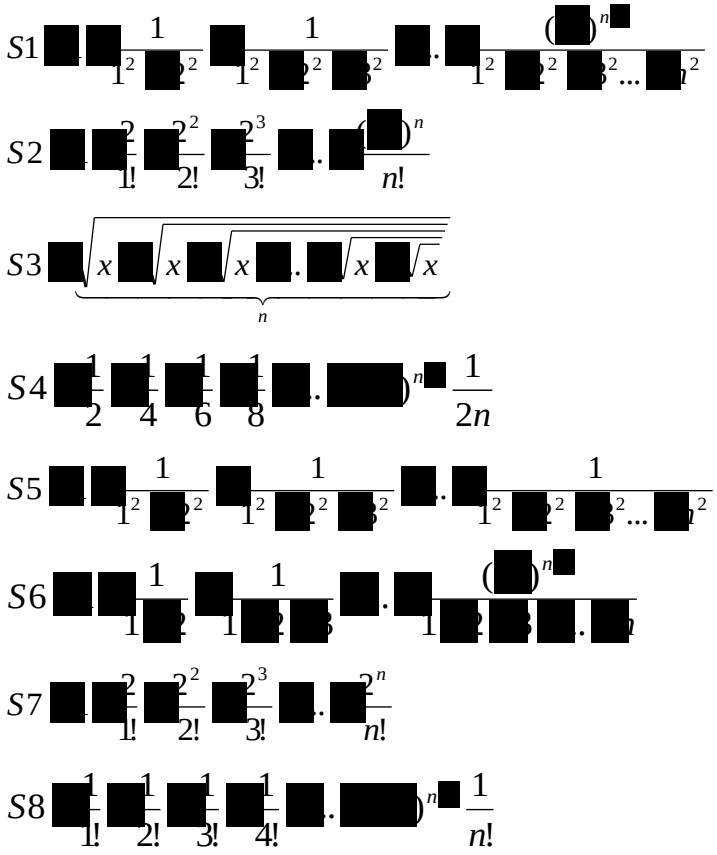
Trong ví dụ này chúng ta định nghĩa hai hàm có cùng tên nhưng một hàm dùng hai tham số kiểu int và hàm còn lại dùng kiểu float. Trình biên dịch sẽ biết cần phải gọi hàm nào bằng cách phân tích kiểu tham số khi hàm được gọi.

Để đơn giản tôi viết cả hai hàm đều có mã lệnh như nhau nhưng điều này không bắt buộc. Bạn có thể xây dựng hai hàm có cùng tên nhưng hoạt động hoàn toàn khác nhau.

B. Phần thảo luận, bài tập

Bài 1. Viết hàm tính N!, với N nguyên dương nhập vào từ bàn phím.

Bài 2. Xây dựng các hàm S1, S2, S3, S4, S5, S6, S7, S8 để tính giá trị của biểu thức tương ứng dưới đây (sử dụng các cấu trúc lặp for, while, do...while). Sử dụng các hàm nói trên để tính giá trị các biểu thức, với n>0 được nhập vào từ bàn phím.



CHƯƠNG 6. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

A. Phần lý thuyết

6.1. Mảng dữ liệu

6.1.1. Mảng một chiều

a) Khái niệm về mảng

Khi cần lưu trữ một dãy n phần tử dữ liệu chúng ta cần khai báo n biến tương ứng với n tên gọi khác nhau. Điều này sẽ rất khó khăn cho người lập trình để có thể nhớ và quản lý hết được tất cả các biến, đặc biệt khi n lớn. Trong thực tế, hiển nhiên chúng ta gặp rất nhiều dữ liệu có liên quan đến nhau về một mặt nào đó, ví dụ chúng có cùng kiểu và cùng thể hiện một đối tượng: như các toạ độ của một vectơ, các số hạng của một ma trận, các sinh viên của một lớp hoặc các dòng kí tự của một văn bản … Lợi dụng đặc điểm này toàn bộ dữ liệu (cùng kiểu và cùng mô tả một đối tượng) có thể chỉ cần chung một tên gọi để phân biệt với các đối tượng khác, và để phân biệt các dữ liệu trong cùng đối tượng ta sử dụng cách đánh số thứ tự cho chúng, từ đó việc quản lý biến sẽ dễ dàng hơn, chương trình sẽ gọn và có tính hệ thống hơn.

Giả sử ta có 2 vectơ trong không gian ba chiều, mỗi vec tơ cần 3 biến để lưu 3 toạ độ, vì vậy để lưu toạ độ của 2 vectơ chúng ta phải dùng đến 6 biến, ví dụ x1, y1, z1 cho vectơ thứ nhất và x2, y2, z2 cho vectơ thứ hai. Một kiểu dữ liệu mới được gọi là mảng một chiều cho phép ta chỉ cần khai báo 2 biến v1 và v2 để chỉ 2 vectơ, trong đó mỗi v1 hoặc v2 sẽ chứa 3 dữ liệu được đánh số thứ tự từ 0 đến 2, trong đó ta có thể ngầm định thành phần 0 biểu diễn toạ độ x, thành phần 1 biểu diễn toạ độ y và thành phần có số thứ tự 2 sẽ biểu diễn toạ độ z.

Tóm lại, mảng là một dãy các thành phần có cùng kiểu được sắp kề nhau liên tục trong bộ nhớ. Tất cả các thành phần đều có cùng tên là tên của mảng. Để phân biệt các thành phần với nhau, các thành phần sẽ được đánh số thứ tự từ 0 cho đến hết mảng. Khi cần nói đến thành phần cụ thể nào của mảng ta sẽ dùng tên mảng và kèm theo số thứ tự của thành phần đó.

Dưới đây là hình ảnh của một mảng gồm có 9 thành phần, các thành phần được đánh số từ 0 đến 8.

0 1 2 3 4 5 6 7 8

b) Khai báo mảng

Cú pháp:

<tên kiểu> <tên mảng>[số thành phần];

<tên kiểu> <tên mảng>[số thành phần]={ dãy giá trị };

<tên kiểu> <tên mảng>[ ] = { dãy giá trị };

- Tên kiểu là kiểu dữ liệu của các thành phần, các thành phần này có kiểu giống nhau. Thỉnh thoảng ta cũng gọi các thành phần là phần tử.

- Cách khai báo trên giống như khai báo tên biến bình thường nhưng thêm số thành phần trong mảng giữa cặp dấu ngoặc vuông [] còn được gọi là kích thước của mảng. Mỗi tên mảng là một biến và để phân biệt với các biến thông thường ta còn gọi là biến mảng.

- Một mảng dữ liệu được lưu trong bộ nhớ bởi dãy các ô liên tiếp nhau. Số lượng ô bằng với số thành phần của mảng và độ dài (byte) của mỗi ô đủ để chứa thông tin của mỗi thành phần. Ô đầu tiên được đánh thứ tự bởi 0, ô tiếp theo bởi 1, và tiếp tục cho đến hết. Như vậy nếu mảng có n thành phần thì ô cuối cùng trong mảng sẽ được đánh số là n – 1.

- Dạng khai báo thứ 2 cho phép khởi tạo mảng bởi dãy giá trị trong cặp dấu {}, mỗi giá trị cách nhau bởi dấu phảy (,), các giá trị này sẽ được gán lần lượt cho các phần tử của mảng bắt đầu từ phần tử thứ 0 cho đến hết dãy. Số giá trị có thể bé hơn số phần tử. Các phần tử mảng chưa có giá trị sẽ không được xác định cho đến khi trong chương trình nó được gán một giá trị nào đó.

- Dạng khai báo thứ 3 cho phép vắng mặt số phần tử, trường hợp này số phần tử được xác định bởi số giá trị của dãy khởi tạo. Do đó nếu vắng mặt cả dãy khởi tạo là không được phép (chẳng hạn khai báo int a[] là sai).

Ví dụ: Khai báo biến chứa 2 vectơ a, b trong không gian 3 chiều:

```
float a[3], b[3];
```

Khai báo 3 phân số a, b, c; trong đó a = 1/3 và b = 3/5:

```
int a[2] = {1, 3}, b[2] = {3, 5}, c[2];
```

ở đây ta ngầm qui ước thành phần đầu tiên (số thứ tự 0) là tử và thành phần thứ hai (số thứ tự 1) là mẫu của phân số.

Khai báo mảng L chứa được tối đa 100 số nguyên dài:

```
long L[100];
```

Khai báo mảng dong (dòng), mỗi dòng chứa được tối đa 80 kí tự:

```
char dong[80];
```

Khai báo dãy Data chứa được 5 số thực độ chính xác gấp đôi:

```
double Data[] = { 0,0,0,0,0 }; //Khởi tạo tạm thời bằng 0
```

c) Cách sử dụng

- Để chỉ thành phần thứ i (hay chỉ số i) của một mảng ta viết tên mảng kèm theo chỉ số trong cặp ngoặc vuông [].

- Tuy mỗi mảng biểu diễn một đối tượng nhưng chúng ta không thể áp dụng các thao tác lên toàn bộ mảng mà phải thực hiện thao tác thông qua từng thành phần của mảng. Ví dụ chúng ta không thể nhập dữ liệu cho mảng a[10] bằng câu lệnh:

```
cin >> a;  //sai
```

mà phải nhập cho từng phần tử từ a[0] đến a[9] của mảng a. Dĩ nhiên trong trường hợp này chúng ta phải cần đến lệnh lặp for:

```
int i;
for (i = 0; i < 10; i++)
 cin >> a[i];
```

Ví dụ 6.1

```
//Tìm tổng, tích 2 phân số.
#include <iostream.h>
void main()
{
     int a[2], b[2], tong[2], tich[2];
     cout<<"Nhập a. Tử = "; cin >> a[0];
     cout<<"mẫu = "; cin >> a[1];
     cout<<"Nhập b. Tử = "; cin >> b[0];
     cout<<"Mẫu = "; cin >> b[1];
     tong[0] = a[0]*b[1] + a[1]*b[0];
     tong[1] = a[1] * b[1];
     tich[0] = a[0]*b[0];
     tich[1] = a[1] * b[1];
     cout<<"Tổng = " << tong[0] << '/' << tong[1];
     cout<<"Tích = " << tich[0] << '/' << tich[1];
}
```

Ví dụ 6.2

Nhập dãy số nguyên, tính: số số hạng dương, âm, bằng không của dãy.

```
#include <iostream.h>
void main()
{
```

```
      float a[50], sd, sa, s0;      //a chứa tối đa 50 số
      int i, n;
      cout<<"Nhập số phần tử của dãy: ";
      cin >> n;      //nhập số phần tử
      for (i=0; i<n; i++) {
           cout<<"a[" << i << "] = ";
           cin >> a[i];
      }    //nhập dãy số
      sd = sa = s0 = 0;
      for (i=1; i<n; i++) {
           if (a[i] > 0) sd++;
           if (a[i] < 0) sa++;
           if (a[i] == 0) s0++;
      }
      cout<<"Số số dương = " << sd << " số số âm = " << sa;
      cout<<"Số số bằng 0 = " << s0;
}
```

6.1.2. Mảng nhiều chiều

Để thuận tiện trong việc biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu có nhiều chỉ tiêu, C++ đưa ra kiểu dữ liệu mảng nhiều chiều. Tuy nhiên, việc sử dụng mảng nhiều chiều rất khó lập trình vì vậy trong mục này chúng ta chỉ bàn đến mảng hai chiều. Đối với mảng một chiều m thành phần, nếu mỗi thành phần của nó lại là mảng một chiều n phần tử thì ta gọi mảng là hai chiều với số phần tử (hay kích thước) mỗi chiều là m và n. Ma trận là một minh hoạ cho hình ảnh của mảng hai chiều, nó gồm m dòng và n cột, tức chứa m x n phần tử, và hiển nhiên các phần tử này có cùng kiểu. Tuy nhiên, về mặt bản chất mảng hai chiều không phải là một tập hợp với m x n phần tử cùng kiểu mà là tập hợp với m thành phần, trong đó mỗi thành phần là một mảng một chiều với n phần tử. Điểm nhấn mạnh này sẽ được giải thích cụ thể hơn trong các phần trình bày về con trỏ của chương sau.

Ví dụ nhiệt độ trung bình theo mùa của ba thành phố: Hà Nội, Thái Nguyên, Cao Bằng

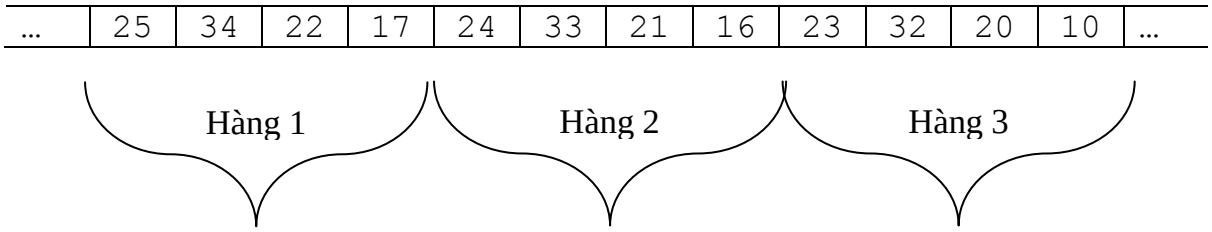
	Mùa xuân	Mùa hạ	Mùa thu	Mùa đông
Hà Nội	25	34	22	17
Thái Nguyên	24	33	21	16
Cao bằng	23	32	20	10

Điều này có thể được biểu diễn bằng một mảng hai chiều mà mỗi phần tử mảng là một số nguyên:

int a[3][4];

Cách tổ chức mảng này trong bộ nhớ như là 12 phần tử số nguyên liên tiếp nhau. Tuy nhiên, lập trình viên có thể tưởng tượng nó như là một mảng gồm ba hàng với mỗi hàng có bốn phần tử số nguyên.

Cách thức tổ chức mảng a trong bộ nhớ như sau:



Như trước, các phần tử được truy xuất thông qua chỉ số mảng. Một chỉ số riêng biệt được cần cho mỗi mảng. Ví dụ, nhiệt độ mùa hè trung bình của thành phố Nà Nội (hàng đầu tiên cột thứ hai) được cho bởi nhietDo[0][1].

a) Khai báo mảng

Cú pháp:

<kiểu thành phần > <tên mảng>[m][n];

- m, n là số hàng, số cột của mảng.
- Kiểu thành phần là kiểu của m x n phần tử trong mảng.
- Trong khai báo cũng có thể được khởi tạo bằng dãy các dòng giá trị, các dòng cách nhau bởi dấu phẩy, mỗi dòng được bao bởi cặp ngoặc {} và toàn bộ giá trị khởi tạo nằm trong cặp dấu {}.

b) Cách sử dụng

- Tương tự mảng một chiều các chiều trong mảng cũng được đánh số từ 0.
- Không sử dụng các thao tác trên toàn bộ mảng mà phải thực hiện thông qua từng phần tử của mảng.
- Để truy nhập phần tử của mảng ta sử dụng tên mảng kèm theo 2 chỉ số chỉ vị trí hàng và cột của phần tử. Các chỉ số này có thể là các biểu thức thực, khi đó C++ sẽ tự chuyển kiểu sang nguyên.

Ví dụ 6.3

Khai báo mảng nhiệt độ trung bình, mảng có thể được khởi tạo bằng cách sử dụng một bộ khởi tạo lồng nhau (xem bảng nhiệt độ trung bình trên):

```
int a [3][4] = {
 {26, 34, 22, 17},
```

```
    {24, 32, 19, 13},
    {28, 38, 25, 20}
   };
```

Ví dụ 6.4

Khai báo 2 ma trận 4 hàng 5 cột A, B chứa các số nguyên:

```
int A[3][4], B[3][4];
```

Khai báo có khởi tạo:

```
int A[3][4] = { {1,2,3,4}, {3,2,1,4}, {0,1,1,0} };
```

với khởi tạo này ta có ma trận:

1	2	3	4
3	2	1	4
0	1	1	0

trong đó: A[0][0] = 1, A[0][1] = 2, A[1][0] = 3, A[2][3] = 0 …

Trong khai báo có thể vắng số hàng (không được vắng số cột), số hàng này được xác định thông qua khởi tạo.

```
float A[][3] = { {1,2,3}, {0,1,0} };
```

Trong khai báo này chương trình tự động xác định số hàng là 2.

Phép khai báo và khởi tạo sau đây là cũng hợp lệ:

```
float A[][3] = { {1,2}, {0} };
```

Chương trình cũng xác định số hàng là 2 và số cột (bắt buộc phải khai báo) là 3 mặc dù trong khởi tạo không thể xác định được số cột. Các phần tử chưa khởi tạo sẽ chưa được xác định cho đến khi nào nó được nhập hoặc gán giá trị cụ thể. Trong ví dụ trên các phần tử A[0][2], A[1][1] và A[1][2] là chưa được xác định.

Ví dụ 6.5

Nhập một ma trận A gồm các số thực từ bàn phím.

- In ma trân A ra màn hình.
- Tìm phần tử lớn nhất của ma trận A.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main()
{
     float a[10][10];
     int m, n;                  //số hàng, cột của ma trận
     int i, j;                  //Các chỉ số trong vòng lặp
```

```
     int amax, imax, jmax;    //số lớn nhất và chỉ số của nó
     clrscr();
     cout<<"Nhập số hàng và cột: ";
     cin >> m >> n; for (i=0; i<m; i++)
     for (j=0; j<n; j++) {
          cout<<"a[" << i << "," << j << "] = ";
          cin >> a[i][j];
     }
     amax = a[0][0]; imax = 0; jmax = 0;
     for (i=0; i<m; i++)
          for (j=0; j<n; j++)
               if (amax < a[i][j]){
                    amax = a[i][j]; imax = i; jmax = j;
               }
     cout<<"Ma trận đã nhập\n";
     cout<<setiosflags(ios:: showpoint) << setprecision(1);
     for (i=0; i<m; i++)
     for (j=0; j<n; j++) {
          if (j==0) cout<<endl;
               cout<<setw(6) << a[i][j];
          }
     cout<<"Số lớn nhất là " << setw(6) << amax << endl;
     cout<<"tại vị trí (" << imax << "," << jmax << ")";
     getch();
}
```

Ghi chú: Khi làm việc với mảng (1 chiều, 2 chiều) do thói quen chúng ta thường tính chỉ số từ 1 (thay vì 0), do vậy trong mảng ta có thể bỏ qua hàng 0, cột 0 bằng cách khai báo số hàng và cột tăng lên 1 so với số hàng, cột thực tế của mảng và từ đó có thể làm việc từ hàng 1, cột 1 trở đi.

Ví dụ 6.6

Nhân 2 ma trận. Cho 2 ma trận A (m x n) và B (n x p). Tính ma trận C = A x B, trong đó C có kích thước là m x p. Ta lập vòng lặp tính từng phần tử của C. Giá trị của phần tử C tại hàng i, cột j chính là tích vô hướng của hàng i ma trận A với cột j ma trận B. Để tránh nhầm lẫn ta qui ước bỏ các hàng, cột 0 của các ma trận A, B, C (tức các chỉ số được tính từ 1 trở đi).

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main() {
     float A[10][10], B[10][10], C[10][10];
     int m, n, p;   //số hàng, cột của ma trận
     int i, j, k;   //Các chỉ số trong vòng lặp
     clrscr();
```

```
     cout<<"Nhập số hàng và cột của 2 ma trận: ";
     cin >> m >> n >> p; //Nhập ma trận A
     for (i=0; i<m; i++)
          for (j=0; j<n; j++) {
               cout<<"A[" << i << "," << j << "] = ";
               cin >> A[i][j];
          }
     //Nhập ma trận B
     for (i=0; i<n; i++)
          for (j=0; j<p; j++) {
               cout<<"B[" << i << "," << j << "] = ";
          cin>>B[i][j];
          }
     //Tính ma trận C = A x B
     for (i=0; i<m; i++)
          for (j=0; j<p; j++) {
               C[i][j] = 0;
               for (k=0; k<n; k++) C[i][j] +=
          A[i][k]*B[k][j];
          }
     //In kết quả
     cout<<"Ma trận kết quả\n";
     cout<<setiosflags(ios:: showpoint) << setprecision(2);
     for (i=0; i<m; i++)
          {
               for (j=0; j<n; j++)
                    cout<<setw(6) << A[i][j];
               cout<<endl;
          }
     getch();
}
```

6.2. Xâu ký tự

Một xâu kí tự là một dãy bất kỳ các kí tự (kể cả dấu cách) do vậy nó có thể được lưu bằng mảng kí tự. Tuy nhiên để máy có thể nhận biết được mảng kí tự này là một xâu, cần thiết phải có kí tự kết thúc xâu, theo qui ước là kí tự có mã 0 (tức '\0') tại vị trí nào đó trong mảng. Khi đó xâu là dãy kí tự bắt đầu từ phần tử đầu tiên (thứ 0) đến kí tự kết thúc xâu đầu tiên (không kể các kí tự còn lại trong mảng).

0	1	2	3	4	5	6	7
H	E	L	L	O	\0		
H	E	L	\0	L	O	\0	
\0	H	E	L	L	O	\0	

Hình vẽ trên minh hoạ 3 xâu, mỗi xâu được chứa trong mảng kí tự có độ dài tối đa là 8. Nội dung xâu thứ nhất là "Hello" có độ dài thực tế là 5 kí tự, chiếm 6 ô trong mảng (thêm ô chứa kí tự kết thúc '\0'). Xâu thứ hai có nội dung "Hel" với độ dài 3 (chiếm 4 ô) và xâu cuối cùng biểu thị một xâu rỗng (chiếm 1 ô). Chú ý mảng kí tự được khai báo với độ dài 8 tuy nhiên các xâu có thể chỉ chiếm một số kí tự nào đó trong mảng này và tối đa là 7 kí tự.

6.2.1. Khai báo

char <tên xâu>[độ dài]; ***//không khởi tạo***

char <tên xâu>[độ dài] = xâu kí tự; ***//Có khởi tạo***

char <tên xâu>[] = xâu kí tự; ***//Có khởi tạo***

- Độ dài mảng là số kí tự tối đa có thể có trong xâu. Độ dài thực sự của xâu chỉ tính từ đầu mảng đến dấu kết thúc xâu (không kể dấu kết thúc xâu '\0').
- Do một xâu phải có dấu kết thúc xâu nên trong khai báo độ dài của mảng cần phải khai báo thừa ra một phần tử. Thực chất độ dài tối đa của xâu = độ dài mảng - 1. Ví dụ nếu muốn khai báo mảng s chứa được xâu có độ dài tối đa 80 kí tự, ta cần phải khai báo char s[81].
- Cách khai báo thứ hai có kèm theo khởi tạo xâu, đó là dãy kí tự đặt giữa cặp dấu nháy kép.

Ví dụ 6.7

```
char hoten[26]; //xâu họ tên chứa tối đa 25 kí tự
char monhoc[31] = "NNLT C++";
```

Xâu môn học chứa tối đa 30 kí tự, được khởi tạo với nội dung "NNLT C++" với độ dài thực sự là 10 kí tự (chiếm 11 ô đầu tiên trong mảng monhoc[31]).

- Cách khai báo thứ 3 tự chương trình sẽ quyết định độ dài của mảng bởi xâu khởi tạo (bằng độ dài xâu + 1).

Ví dụ 6.8

```
char thang[] = "Mười hai";      //Độ dài mảng = 9
```

6.2.2. Cách sử dụng

Tương tự như các mảng dữ liệu khác, xâu kí tự có những đặc trưng như mảng, tuy nhiên chúng cũng có những điểm khác biệt. Dưới đây là các điểm giống và khác nhau đó.

- Truy cập một kí tự trong xâu: cú pháp giống như mảng.

Ví dụ 6.9

```
char s[50] = "I\'m a student";
//Chú ý kí tự ' phải được viết là \'
cout<<s[0];//In kí tự đầu tiên, tức kí tự 'I'
s[1] = 'a';     //đặt lại kí tự thứ 2 là 'a'
```

- Không được thực hiện các phép toán trực tiếp trên xâu như:

```
char s[20] = "Hello", t[20];
//khai báo hai xâu s và t
t = "Hello"; //sai, chỉ gán được khi khai báo
t = s;     //sai, không gán được toàn bộ mảng
if (s < t) …    //sai, không so sánh được hai mảng
…
```

- Toán tử nhập dữ liệu >> vẫn dùng được nhưng có nhiều hạn chế.

Ví dụ 6.10

```
char s[60];
cin >> s;
cout<<s;
```

Nếu xâu nhập vào là "Tin học hoá" chẳng hạn thì toán tử >> chỉ nhập "Tin" cho s (bỏ tất cả các kí tự đứng sau dấu trắng), vì vậy khi in ra trên màn hình chỉ có từ "Tin".

Vì các phép toán không dùng được trực tiếp trên xâu nên các chương trình dịch đã viết sẵn các hàm thư viện được khai báo trong file nguyên mẫu string.h. Các hàm này giải quyết được hầu hết các công việc cần thao tác trên xâu. Nó cung cấp cho NSD phương tiện để thao tác trên xâu như gán, so sánh, sao chép, tính độ dài xâu, nhập, in, … Để sử dụng được các hàm này đầu chương trình cần có khai báo string.h. Phần lớn các hàm này sẽ được giới thiệu trong phần tiếp sau.

6.2.3. Phương thức nhập xâu

Do toán tử nhập >> có hạn chế đối với xâu kí tự nên C++ đưa ra hàm riêng (còn gọi là phương thức) cin.getline(s,n) để nhập xâu kí tự. Hàm có 2 đối với s là xâu cần nhập nội dung và n-1 là số kí tự tối đa của xâu. Giống phương thức nhập kí tự cin.get(c), khi gặp hàm cin.getline(s,n) chương trình sẽ nhìn vào bộ đệm bàn phím lấy ra n-1 kí tự (nếu đủ hoặc lấy tất cả kí tự còn lại, trừ kí tự enter) và gán cho s. Nếu tại thời điểm đó bộ đệm đang rỗng, chương trình sẽ tạm dừng chờ NSD nhập dữ liệu (dãy kí tự) vào từ bàn phím. NSD có thể nhập vào dãy với độ dài bất kỳ cho đến khi nhấn Enter, chương trình sẽ lấy ra n-1 kí tự đầu tiên gán cho s, phần còn lại

vẫn được lưu trong bộ đệm (kể cả kí tự Enter) để dùng cho lần nhập sau. Hiển nhiên, sau khi gán các kí tự cho s, chương trình sẽ tự động đặt kí tự kết thúc xâu vào ô tiếp theo của xâu s.

Ví dụ xét đoạn lệnh sau

```
char s[10];
cin.getline(s, 10);
cout<<s << endl;
cin.getline(s, 10);
cout<<s << endl;
```

Giả sử ta nhập vào bàn phím dòng kí tự: 1234567890abcd ↵. Khi đó lệnh cin.getline(s,10) đầu tiên sẽ gán xâu "123456789" (9 kí tự) cho s, phần còn lại vẫn lưu trong bộ đệm bàn phím. Tiếp theo s được in ra màn hình. Đến lệnh cin.getline(s,10) thứ hai NSD không phải nhập thêm dữ liệu, chương trình tự động lấy nốt số dữ liệu còn lại (vì chưa đủ 9 kí tự) "0abcd" để gán cho s. Sau đó in ra màn hình. Như vậy trên màn hình sẽ xuất hiện hai dòng:

```
123456789
0abcd
```

Ví dụ 6.11

Nhập một ngày tháng dạng Mỹ (mm/dd/yy), đổi sang ngày tháng dạng Việt Nam rồi in ra màn hình.

```
#include <iostream.h>
main(){
     char US[9], VN[9] = " / / ";
     //khởi tạo trước hai dấu /
     cin.getline(US, 9);
     //nhập ngày tháng, ví dụ "05/01/99"
     VN[0] = US[3]; VN[1] = US[4];      //ngày
     VN[3] = US[0]; VN[4] = US[1];      //Tháng
     VN[6] = US[6]; VN[7] = US[7];      //năm
     cout<<VN << endl;
}
```

6.2.4. Một số hàm xử lí xâu

- ***Hàm strcpy(s, t)***

Gán nội dung của xâu t cho xâu s (thay cho phép gán = không được dùng). Hàm sẽ sao chép toàn bộ nội dung của xâu t (kể cả kí tự kết thúc xâu) vào cho xâu s. Để sử dụng hàm này cần đảm bảo độ dài của mảng s ít nhất cũng bằng độ dài của mảng

t. Trong trường hợp ngược lại kí tự kết thúc xâu sẽ không được ghi vào s và điều này có thể gây treo máy khi chạy chương trình.

Ví dụ 6.12

```
char s[10], t[10];
t = "Face";                 //Không được dùng
s = t;                      //Không được dùng
strcpy(t, "Face");          //Được, gán "Face" cho t
strcpy(s, t);               //Được, sao chép t sang s
cout<<s << " to " << t;     //In ra: Face to Face
```

- ***Hàm strncpy(s, t, n)***

Sao chép n kí tự của t vào s. Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn kí tự kết thúc xâu cho s. Do vậy NSD phải thêm câu lệnh đặt kí tự '\0' vào cuối xâu s sau khi sao chép xong.

Ví dụ 6.13

```
char s[10], t[10] = "Steven";
strncpy(s, t, 5);           //Copy 5 kí tự "Steve" vào s
s[5] = '\0';                //Đặt dấu kết thúc xâu
//In câu: Steve is young brother of Steven
cout<<s << " is young brother of " << t;
```

Một sử dụng có ích của hàm này là copy một xâu con bất kỳ của t và đặt vào s. Ví dụ cần copy xâu con dài 2 kí tự bắt đầu từ kí tự thứ 3 của xâu t và đặt vào s, ta viết strncpy(s, t+3, 2). Ngoài ra xâu con được copy có thể được đặt vào vị trí bất kỳ của s (không nhất thiết phải từ đầu xâu s) chẳng hạn đặt vào từ vị trí thứ 5, ta viết: strncpy(s+5, t+3, 2). Câu lệnh này có nghĩa: lấy 2 kí tự thứ 3 và thứ 4 của xâu t đặt vào 2 ô thứ 5 và thứ 6 của xâu s. Trên cơ sở này chúng ta có thể viết các đoạn chương trình ngắn để thay thế một đoạn con bất kỳ nào đó trong s bởi một đoạn con bất kỳ (có độ dài tương đương) trong t. Ví dụ các dòng lệnh chuyển đổi ngày tháng trong ví dụ trước có thể viết lại bằng cách dùng hàm strncpy như sau:

```
strncpy(VN+0, US+3, 2);     //ngày
strncpy(VN+3, US+0, 2);     //Tháng
strncpy(VN+6, US+6, 2);     //năm
```

- ***Hàm strcat(s, t)***

Nối một bản sao của t vào sau s (thay cho phép +). Hiển nhiên hàm sẽ loại bỏ kí tự kết thúc xâu s trước khi nối thêm t. Việc nối sẽ đảm bảo lấy cả kí tự kết thúc của xâu t vào cho s (nếu s đủ chỗ) vì vậy NSD không cần thêm kí tự này vào cuối xâu. Tuy nhiên, hàm không kiểm tra xem liệu độ dài của s có đủ chỗ để nối thêm nội dung, việc kiểm tra này phải do NSD đảm nhiệm. Ví dụ:

```
char a[100] = "Man", b[4] = "tôi";
strcat(a, " và ");
strcat(a, b);
cout<<a    //Man va toi
char s[100], t[100] = "Steve";
strncpy(s, t, 3); s[3] = '\0';
strcat(s, "p");      //s = "Step"
cout<<t << " goes " << s << " by " << s
```

- ***Hàm strncat(s, t, n)***

Nối bản sao n kí tự đầu tiên của xâu t vào sau xâu s. Hàm tự động đặt thêm dấu kết thúc xâu vào s sau khi nối xong (tương phản với strncpy()). Cũng giống strcat hàm đòi hỏi độ dài của s phải đủ chứa kết quả. Tương tự, có thể sử dụng cách viết strncat(s, t+k, n) để nối n kí tự từ vị trí thứ k của xâu t cho s. Ví dụ:

```
char s[20] = "Nha ";
char t[] = "vua chua"
strncat(s, t, 3);    //s = "Nha vua"
```

Hoặc:

```
strncat(s, t+4, 4);  //s = "Nha chua"
```

- ***Hàm strcmp(s, t)***

Hàm so sánh 2 xâu s và t (thay cho các phép toán so sánh). Giá trị trả lại là hiệu 2 kí tự khác nhau đầu tiên của s và t. Từ đó, nếu s1 < s2 thì hàm trả lại giá trị âm, bằng 0 nếu s1==s2, và dương nếu s1 > s2. Trong trường hợp chỉ quan tâm đến so sánh bằng,
nếu hàm trả lại giá trị 0 là 2 xâu bằng nhau và nếu giá trị trả lại khác 0 là 2 xâu khác nhau. Ví dụ:

```
if (strcmp(s,t))
 cout<<"s khác t";
else
 cout<<"s bằng t";
```

- ***Hàm strncmp(s, t);***

Giống hàm strcmp(s, t) nhưng chỉ so sánh tối đa n kí tự đầu tiên của hai xâu. Ví dụ:

```
char s[] = "Ha Noi", t[] = "Ha noi";
cout<<strcmp(s,t);
//-32 (vì 'N' = 78, 'n' = 110)
cout<<strncmp(s, t, 3);
//0 (vì 3 kí tự đầu của s và t là như nhau)
```

- ***Hàm strcmpi(s, t)***

Như strcmp(s, t) nhưng không phân biệt chữ hoa, thường. Ví dụ:

```
char s[] = "Ha Noi", t[] = "Ha noi";
cout<<strcmpi(s, t); //0 (vì s = t)
```

- ***Hàm strupr(s)***

Hàm đổi xâu s thành in hoa, và cũng trả lại xâu in hoa đó. Ví dụ:

```
char s[10] = "Ha noi";
cout<<strupr(s);     //HA NOI
cout<<s;             //HA NOI (s cũng thành in hoa)
```

- ***Hàm strlwr(s)***

Hàm đổi xâu s thành in thường, kết quả trả lại là xâu s. Ví dụ:

```
char s[10] = "Ha Noi";
cout<<strlwr(s);     //ha noi
cout<<s;             //ha noi (s cũng thành in thường)
```

- ***Hàm strlen(s)***

Hàm trả giá trị là độ dài của xâu s.

Ví dụ:

```
char s[10] = "Ha Noi";
cout<<strlen(s);     //5
```

Sau đây là một số ví dụ sử dụng tổng hợp các hàm trên.

Ví dụ 6.14

Thống kê số chữ 'a' xuất hiện trong xâu s.

```
#include <iostream.h>

void main() {

     const int MAX = 100;
     char s[MAX+1];
     int sokitu = 0;
     cout<<"Nhap xau:";
     cin.getline(s, MAX+1);
     for (int i=0; i < strlen(s); i++)
          if (s[i] == 'a') sokitu++;
               cout<<"So ky tu= " << sokitu << endl;
}
```

Ví dụ 6.15

Tính độ dài xâu bằng cách đếm từng kí tự (tương đương với hàm strlen())

```
#include <iostream.h>
void main()
```

```
{
     char s[100];
     cin.getline(s, 100);
     int i=0;
     while(s[i++]!='\0');
     i--;
     cout<<"Do dai xau= " << i;
}
```

Ví dụ 6.16

Sao chép xâu s sang xâu t (tương đương với hàm strcpy(t,s))

```
#include <iostream.h>
void main()
{
     char s[100], t[100];
     cin.getline(s, 100);     //nhập xâu s
     int i=0;
     while ((t[i] = s[i]) != '\0') i++;
     //Copy cả dấu kết thúc xâu '\0'
     cout<<t << endl;
}
```

Ví dụ 6.17

Cắt dấu cách 2 đầu của xâu s. Chương trình sử dụng biến i chạy từ đầu xâu đến vị trí đầu tiên có kí tự khác dấu trắng. Từ vị trí này sao chép từng kí tự còn lại của xâu về đầu xâu bằng cách sử dụng thêm biến j để làm chỉ số cho xâu mới. Kết thúc sao chép j sẽ ở vị trí cuối xâu (mới). Cho j chạy ngược về đầu xâu cho đến khi gặp kí tự đầu tiên khác dấu trắng. Đặt dấu kết thúc xâu tại đây.

```
#include <iostream.h>
void main()
{
     char s[100];
     cout<<"Nhap xau: ";
     cin.getline(s, 99); //Nhập xâu
     int i=0, j=0;
     //Bỏ qua các khoảng trắng đầu xâu
     while (s[i++] == ' '); i--;
     //Sao chép phần còn lại vào s
     while (s[i] != '\0') s[j++] = s[i++];
     //Bỏ qua các dấu cách cuối xâu
     while (s[--j] == ' ');
     s[j+1] = '\0';
     cout<<s;
}
```

Ví dụ 6.18

Chạy dòng chữ quảng cáo vòng tròn từ phải sang trái giữa màn hình.

Giả sử hiện 30 kí tự của xâu quảng cáo. Ta sử dụng vòng lặp. Cắt 30 kí tự đầu tiên của xâu cho vào biến hien, hiện biến này ra màn hình. Bước lặp tiếp theo cắt ra 30 kí tự của xâu nhưng dịch sang phải 1 kí tự cho vào biến hien và hiện ra màn hình. Quá trình tiếp tục, mỗi bước lặp ta dịch chuyển nội dung cần hiện ra màn hình 1 kí tự, do hiệu ứng của mắt ta thấy dòng chữ sẽ chạy từ biên phải về biên trái của màn hình. Để quá trình chạy theo vòng tròn (khi hiện đến kí tự cuối của xâu sẽ hiện quay lại từ kí tự đầu của xâu) chương trình sử dụng biến i đánh dấu điểm đầu của xâu con cần cắt cho vào biến hien, khi i bằng độ dài của xâu chương trình đặt lại i = 0 (cắt lại từ đầu xâu). Ngoài ra, để phần cuối xâu nối với phần đầu (tạo thành vòng tròn) ngay từ đầu chương trình, xâu quảng cáo sẽ được nối thành gấp đôi.

Vòng lặp tiếp tục đến khi nào NSD ấn phím bất kỳ (chương trình nhận biết điều này nhờ vào hàm kbhit() thuộc file nguyên mẫu conio.h) thì dừng. Để dòng chữ chạy không quá nhanh chương trình sử dụng hàm trễ delay(n) (thuộc dos.h, tạm dừng trong n phần nghìn giây) với n được điều chỉnh thích hợp theo tốc độ của máy. Hàm gotoxy(x, y) (thuộc conio.h) trong chương trình đặt con trỏ màn hình tại vị trí cột x dòng y để đảm bảo dòng chữ luôn luôn hiện ra tại đúng một vị trí trên màn hình.

```
#include <iostream.h>
#include <conio.h>
#include <dos.h>
void main() {
     char qc[100] = "Quảng cáo miễn phí: Không có tiền thì \
     không có kem. ";
     int dd = strlen(qc);
     char tam[100];
     strcpy(tam, qc);
     strcat(qc, tam);    //nhân đôi dòng quảng cáo
     clrscr();           //xoá màn hình
     char hien[31];      //Chứa xâu dài 30 kí tự để hiện
     i = 0;
     while (!kbhit()) { //Trong khi chưa ấn phím bất kỳ
          strncpy(hien, s+i, 30);
          hien[30] = '\0';//Copy 30 kí tự từ qc[i] sang hien
          gotoxy(20,10); cout<<hien;
          //In hien tại dòng 10 cot 20
          delay(100);              //Tạm dừng 1/10 giây
          i++; if (i==dd) i = 0;   //Tăng i
     }
}
```

Ví dụ 6.19

Nhập mật khẩu (không quá 10 kí tự). In ra "đúng" nếu là "HaNoi2000", "sai" nếu ngược lại. Chương trình cho phép nhập tối đa 3 lần. Nhập riêng rẽ từng kí tự (bằng hàm getch()) cho mật khẩu. Hàm getch() không hiện kí tự NSD gõ vào, thay vào đó chương trình chỉ hiện kí tự 'X' để che giấu mật khẩu. Sau khi NSD đã gõ xong (9 kí tự) hoặc đã Enter, chương trình so sánh xâu vừa nhập với "HaNoi2000", nếu đúng chương trình tiếp tuc, nếu sai tăng số lần nhập (cho phép không quá 3 lần).

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
void main() {
     char pw[11]; int solan = 0; //Cho phep nhap 3 lan
     do {
          clrscr(); gotoxy(30,12);
          int i = 0;
          while ((pw[i]=getch()) != 13 && ++i < 10)
               cout<<'X'; //13 = Enter pw[i] = '\0';
          cout<<endl;
          if (!strcmp(pw, "HaNoi2000")) {
               cout<<"Mời vào"; break; }
          else { cout<<"Sai mật khẩu. Nhập lại");
          solan++; }
     }while (solan < 3);
}
```

6.3. Cấu trúc (structure)

6.3.1. Khai báo

Để tạo ra một kiểu cấu trúc NSD cần phải khai báo tên của kiểu (là một tên gọi do NSD tự đặt), tên cùng với các thành phần dữ liệu có trong kiểu cấu trúc này. Một kiểu cấu trúc được khai báo theo mẫu sau:

struct <tên kiểu> {

các thành phần;

} <danh sách biến>;

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.

- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).

- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.

- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

- Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

struct <tên cấu trúc> <danh sách biến>; //Trong C

hoặc

<tên cấu trúc> <danh sách biến>; //Trong C++

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> biến = { giá trị khởi tạo };

Ví dụ 6.20

- Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số.

```
struct Phanso {
 int tu;
 int mau;
};
```

hoặc:

```
struct Phanso { int tu, mau; }
```

- Kiểu ngày tháng gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Ngaythang {
 int ng;
 int th;
 int nam;
} holiday = { 1,5,2000 };
```

Một biến holiday cũng được khai báo kèm cùng kiểu này và được khởi tạo bởi bộ số 1. 5. 2000. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 1, th = 5 và nam = 2000.

- Kiểu Lop dùng chứa thông tin về một lớp học gồm tên lớp và sĩ số sinh viên. Các biến kiểu Lop được khai báo là daihoc và caodang, trong đó daihoc được khởi tạo bởi bộ giá trị {"K41T", 60} với ý nghĩa tên lớp đại học là K41T và sĩ số là 60 sinh viên.

```
struct Lop {
 char tenlop[10],
 int soluong;
};
```

```
struct Lop daihoc = {"K41T", 60}, caodang;
```

hoặc:

```
Lop daihoc = {"K41T", 60}, caodang;
```

- Kiểu Sinhvien gồm có các trường hoten để lưu trữ họ và tên sinh viên, ns lưu trữ ngày sinh, gt lưu trữ giới tính dưới dạng số (qui ước 1: nam, 2: nữ) và cuối cùng trường diem lưu trữ điểm thi của sinh viên. Các trường trên đều có kiểu khác nhau.

```
struct Sinhvien {
 char hoten[25];
 Ngaythang ns;
 int gt;
float diem;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1};
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng K41T với 60 phần tử kiểu Sinhvien. Một biến y được khai báo thêm và kèm theo khởi tạo giá trị {"NVA", {1,1,1980}, 1}, tức họ tên của sinh viên y là "NVA", ngày sinh là 1/1/1980, giới tính nam và điểm thi để trống. Đây là kiểu khởi tạo thiếu giá trị, giống như khởi tạo mảng, các giá trị để trống phải nằm ở cuối bộ giá trị khởi tạo (tức các thành phần bỏ khởi tạo không được nằm xen kẽ giữa những thành phần được khởi tạo).Ví dụ này còn minh hoạ cho các cấu trúc lồng nhau, cụ thể trong kiểu cấu trúc Sinhvien có một thành phần cũng kiểu cấu trúc là thành phần ns.

6.3.2. Truy nhập các thành phần kiểu cấu trúc

Để truy nhập vào các thành phần kiểu cấu trúc ta sử dụng cú pháp: tên biến.tên thành phần hoặc tên biến → tên thành phần đối với biến con trỏ cấu trúc. Cụ thể:

- Đối với biến thường: tên biến.tên thành phần

Ví dụ 6.21

```
struct Lop {
 char tenlop[10];
 int siso;
};
Lop daihoc = "K41T", caodang;
caodang.tenlop = daihoc.tenlop;
//Gán tên lớp cđẳng bởi tên lớp đhọc
caodang.siso++; //Tăng sĩ số lớp caodang lên 1
```

- Đối với biến con trỏ: tên biến → tên thành phần. Ví dụ:

```
struct Sinhvien {
 char hoten[25];
```

```
 Ngaythang ns;
 int gt;
float diem;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1};
y.diem = 5.5;        //Gán điểm thi cho sinh viên y
p = new Sinhvien;    //Cấp bộ nhớ chứa 1 sinh viên
strcpy(p→hoten, y.hoten);
//Gán họ tên của y cho sv trỏ bởi p
cout<<p→hoten << y.hoten;
//In hoten của y và con trỏ p
```

- Đối với biến mảng: truy nhập thành phần mảng rồi đến thành phần cấu trúc.

Ví dụ 6.22

```
//Gán họ tên cho sv đầu tiên của lớp
strcpy(K41T[1].hoten, p→hoten);
K41T[1].diem = 7.0;  //Gán điểm cho sv đầu tiên
```

- Đối với cấu trúc lồng nhau. Truy nhập thành phần ngoài rồi đến thành phần của cấu trúc bên trong, sử dụng các phép toán. hoặc → (các phép toán lấy thành phần) một cách thích hợp.

```
x.ngaysinh.ng = y.ngaysinh.ng; //Gán ngày,
x.ngaysinh.th = y.ngaysinh.th; //Tháng,
//Năm sinh của y cho x.
x.ngaysinh.nam = y.ngaysinh.nam;
```

6.3.3. Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết câu lệnh vào/ra từng cho từng thành phần. Nhận xét này được minh họa trong ví dụ sau:

```
struct Sinhvien {
 char hoten[25];
 Ngaythang ns;
 int gt;
 float diem;
} x, y;
cout<<" Nhập dữ liệu cho sinh viên x: " << endl;
in.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem
cout<<"Thông tin về sinh viên x là: " << endl;
cout<<"Họ và tên: " << x.hoten << endl;
```

```
cout<<"Sinh ngày:"<< x.ns.ng<<"/"<<x.ns.th<< "/" << x.ns.nam;
cout<<"Giới tính: " << (x.gt == 1)?"Nam": "Nữ;
cout<<x.diem
```

Tuy nhiên, khác với biến mảng, đối với cấu trúc chúng ta có thể gán giá trị của 2 biến cho nhau. Phép gán này cũng tương đương với việc gán từng thành phần của cấu trúc. Ví dụ:

```
struct Sinhvien {
 char hoten[25];
 Ngaythang ns;
 int gt;
 float diem;
} x, y, *p;
cout<<" Nhập dữ liệu cho sinh viên x: " << endl;
in.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam; cin >> x.gt;
cin >> x.diem
y = x;
//Đối với biến mảng, phép gán này là không thực hiện được
p = new Sinhvien[1]; *p = x;
cout<<"Thông tin về sinh viên y là: " << endl;
cout<<"Họ và tên: " << y.hoten << endl;
cout<<"Sinh ngày: "<< y.ns.ng << "/" <<y.ns.th<<"/"<<y.ns.nam;
cout<<"Giới tính: " << (y.gt = 1)?"Nam": "Nữ;
cout<<y.diem
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc. Cách gán này chỉ thực hiện được khi khởi tạo.

Ví dụ 6.23

```
Sinhvien x = { "NVA", {1,1,1980}, 1, 7.0}, y; //được
y = { "NVA", {1,1,1980}, 1, 7.0};   //không được
y = x;      //được
```

Ví dụ 6.24

Cộng, trừ, nhân chia hai phân số được cho dưới dạng cấu trúc.

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
     int tu;
     int mau;
} a, b, c;
void main()
{
     clrscr();
     cout<<"Nhập phân số a: " << endl;  //nhập a
```

```
     cout<<"Tử: "; cin >> a.tu; cout<<"Mẫu: ";
     cin >> a.mau;
     cout<<"Nhập phân số b: " << endl;  //nhập b
     cout<<"Tử: "; cin >> b.tu; cout<<"Mẫu: ";
     cin >> b.mau;
     c.tu = a.tu*b.mau + a.mau*b.tu; //Tính và in a+b
     c.mau = a.mau*b.mau;
     cout<<"a + b = " << c.tu << "/" << c.mau;
     c.tu = a.tu*b.mau - a.mau*b.tu; //Tính và in a-b
     c.mau = a.mau*b.mau;
     cout<<"a - b = " << c.tu << "/" << c.mau;
     c.tu = a.tu*b.tu;   //Tính và in axb
     c.mau = a.mau*b.mau;
     cout<<"a + b = " << c.tu << "/" << c.mau;
     c.tu = a.tu*b.mau;  //Tính và in a/b
     c.mau = a.mau*b.tu;
     cout<<"a + b = " << c.tu << "/" << c.mau; getch();
}
```

6.4. Cấu trúc động của dữ liệu (Union)

Union cho phép một phần bộ nhớ có thể được truy xuất dưới dạng nhiều kiểu dữ liệu khác nhau mặc dù tất cả chúng đều nằm cùng một vị trí trong bộ nhớ. Phần khai báo và sử dụng nó tương tự với cấu trúc nhưng chức năng thì khác hoàn toàn:

```
union model_name {
 type1 element1;
 type2 element2;
 type3 element3;
 .
 .
} object_name;
```

Tất cả các phần tử của union đều chiếm cùng một chỗ trong bộ nhớ. Kích thước của nó là kích thước của phần tử lớn nhất. Ví dụ:

```
union mytypes_t {
 char c;
 int i;
 float f;
}mytypes;
```

định nghĩa ba phần tử

```
mytypes.c
mytypes.i
mytypes.f
```

mỗi phần tử có một kiểu dữ liệu khác nhau. Nhưng vì tất cả chúng đều nằm cùng một chỗ trong bộ nhớ nên bất kì sự thay đổi nào đối với một phần tử sẽ ảnh hưởng tới tất cả các thành phần còn lại.

Một trong những công dụng của union là dùng để kết hợp một kiểu dữ liệu cơ bản với một mảng hay các cấu trúc gồm các phần tử nhỏ hơn. Ví dụ:

```
union mix_t{
 long l;
 struct {
 short hi;
      short lo;
 } s;
 char c[4];
} mix;
```

định nghĩa ba phần tử cho phép chúng ta truy xuất đến cùng một nhóm 4 byte: mix.l, mix.s và mix.c mà chúng ta có thể sử dụng tuỳ theo việc chúng ta muốn truy xuất đến nhóm 4 byte này như thế nào. Tôi dùng nhiều kiểu dữ liệu khác nhau, mảng và cấu trúc trong union để bạn có thể thấy các cách khác nhau mà chúng ta có thể truy xuất dữ liệu.

Các union vô danh

Trong C++ chúng ta có thể sử dụng các unions vô danh. Nếu chúng ta đặt một union trong một cấu trúc mà không đề tên (phần đi sau cặp ngoặc nhọn { }) union sẽ trở thành vô danh và chúng ta có thể truy xuất trực tiếp đến các phần tử của nó mà không cần đến tên của union (có cần cũng không được). Ví dụ, hãy xem xét sự khác biệt giữa hai phần khai báo sau đây:

union	union vô danh

`struct {`
`  char title[50];`
`  char author[50];`
`  union {`
`  float dollars;`
`  int yens;`
`  } price;`
`} book;`
	`struct {`
	`  char title[50];`
	`  char author[50];`
	`  union {`
	`  float dollars;`
	`  int yens;`
	`  };`
	`} book;`

Sự khác biệt duy nhất giữa hai đoạn mã này là trong đoạn mã đầu tiên chúng ta đặt tên cho union (price) còn trong cái thứ hai thì không. Khi truy nhập vào các phần tử dollars và yens, trong trường hợp thứ nhất chúng ta viết:

```
book.price.dollars
```

```
book.price.yens
```

còn trong trường hợp thứ hai:

```
book.dollars
book.yens
```

Một lần nữa tôi nhắc lại rằng vì nó là một union, hai trường dollars và yens đều chiếm cùng một chỗ trong bộ nhớ nên chúng không thể giữ hai giá trị khác nhau.

6.5. Các kiểu dữ liệu tự định nghĩa khác

C++ cho phép chúng ta định nghĩa các kiểu dữ liệu của riêng mình dựa trên các kiểu dữ liệu đã có. Để có thể làm việc đó chúng ta sẽ sử dụng từ khoá typedef, dạng thức như sau:

typedef existing_type new_type_name;

Trong đó existing_type là một kiểu dữ liệu cơ bản hay bất kì một kiểu dữ liệu đã định nghĩa và new_type_name là tên của kiểu dữ liệu mới. Ví dụ

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

Trong trường hợp này chúng ta đã định nghĩa bốn kiểu dữ liệu mới: C, WORD, string_t và field kiểu char, unsigned int, char* kiểu char[50], chúng ta hoàn toàn có thể sử dụng chúng như là các kiểu dữ liệu hợp lệ:

```
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

typedef có thể hữu dụng khi bạn muốn định nghĩa một kiểu dữ liệu được dùng lặp đi lặp lại trong chương trình hoặc kiểu dữ liệu bạn muốn dùng có tên quá dài và bạn muốn nó có tên ngắn hơn.

Kiểu liệt kê (enum)

Kiểu dữ liệu liệt kê dùng để tạo ra các kiểu dữ liệu chứa một cái gì đó hơi đặc biệt một chút, không phải kiểu số hay kiểu kí tự hoặc các hằng true và false. Dạng thức của nó như sau:

enum model_name {
value1,
value2,

value3,

.

.

} object_name;

Ví dụ, chúng ta có thể tạo ra một kiểu dữ liệu mới có tên color để lưu trữ các màu với phần khai báo như sau:

```
   enum colors_t {black, blue, green, cyan, red, purple,
yellow, white};
```

Chú ý rằng chúng ta không sử dụng bất kì một kiểu dữ liệu cơ bản nào trong phần khai báo. Chúng ta đã tạo ra một kiểu dữ liệu mới mà không dựa trên bất kì kiểu dữ liệu nào có sẵn: kiểu color_t, những giá trị có thể của kiểu color_t được viết trong cặp ngoặc nhọn {}. Ví dụ, sau khi khai báo kiểu liệt kê, biểu thức sau sẽ là hợp lệ:

```
colors_t mycolor;
mycolor = blue;
if (mycolor == green)
 mycolor = red;
```

Trên thực tế kiểu dữ liệu liệt kê được dịch là một số nguyên và các giá trị của nó là các hằng số nguyên được chỉ định. Nếu điều này không đươc chỉ định, giá trị nguyên tương đương với phần tử đầu tiên là 0 và các giá trị tiếp theo cứ thế tăng lên 1, Vì vậy, trong kiểu dữ liệu colors_t mà chúng ta định nghĩa ở trên, white tương đương với 0, blue tương đương với 1, green tương đương với 2 và cứ tiếp tục như thế.

Nếu chúng ta chỉ định một giá trị nguyên cho một giá trị nào đó của kiểu dữ liệu liệt kê (trong ví dụ này là phần tử đầu tiên) các giá trị tiếp theo sẽ là các giá trị nguyên tiếp theo, ví dụ:

```
enum months_t
{
 january=1, february, march, april, may, june, july,
 august,september, october, november, december
} y2k;
```

trong trường hợp này, biến y2k có kiểu dữ liệu liệt kê months_t có thể chứa một trong 12 giá trị từ january đến december và tương đương với các giá trị nguyên từ 1 đến 12, không phải 0 đến 11 vì chúng ta đã đặt january bằng 1.

B. Phần thảo luận, bài tập

Bài 1. Viết chương trình nhập mảng n phần tử số nguyên, tìm phần tử lớn nhất của mảng.

Bài 2. Viết hàm tìm phần tử lớn nhất trong mảng (trả về giá trị và chỉ số).

Bài 3. Viết hàm tìm phần tử nhỏ nhất trong mảng (trả về giá trị và chỉ số).

Bài 4. Viết chương trình nhập mảng n phần tử số nguyên, đếm, tính tổng và liệt kê các số nguyên tố trong mảng.

Bài 5. Viết chương trình nhập mảng n phần tử số nguyên, đếm, tính tổng và liệt kê các số chính phương trong mảng.

Bài 6. Viết chương trình nhập n phần tử số nguyên. Nhập phần tử cần tìm kiếm X. Nếu trong n phần tử đã nhập có X thì báo "tìm thấy", "số lần tìm thấy" và "các vị trí tìm thấy", ngược lại báo "không tìm thấy".

Bài 7. Viết chương trình nhập vào ma trận A có kích thước mxn, hãy tính:

a) Tính tổng các phần tử âm, dương của ma trận.

b) Tính tổng các phần tử chẵn, lẻ của ma trận.

c) Tìm hàng có tổng các phần tử lớn nhất.

d) Tìm cột có tổng các phần tử lớn nhất.

Bài 8. Viết chương trình nhập vào hai ma trận vuông A, B có kích thước N x N. Hãy tính tổng, hiệu, tích của hai ma trân A và B.

Bài 9. Nhập xâu ký tự từ bàn phím, đếm số từ trong xâu (từ là dãy ký tự không chứa ký tự trắng)

Bài 12. Nhập xâu ký tự từ bàn phím, đếm xem trong xâu ký tự nào xuất hiện nhiều nhất.

Bài 10. Viết hàm tính độ dài xâu (không sử dụng thư viện). Viết chương trình nhập xâu từ bàn phím, sử dụng hàm vừa xây dựng đưa ra độ dài xâu.

Bài 13. Nhập xâu ký tự từ bàn phím, chuẩn hoá xâu đó:

a) Loại bỏ khoảng trắng bên trái xâu.

b) Loại bỏ khoảng trắng bên phải xâu

c) Loại bỏ các khoảng trắng thừa giữa các từ trong xâu.

Bài 14. Cho một xây bất kỳ, không sử dụng các hàm thư viện về xâu, hãy xây dựng một hàm đổi tất cả các ***chữ thường*** thành ***chữ hoa*** (các ký tự khác giữ nguyên) và in cả hai ra màn hình. Viết một chương trình nhập một xây bất kỳ từ bàn phím, sau đó sử dụng hàm đã xây dựng ở trên để in kết quả ra màn hình.

Bài 15. Nhập mảng n sinh viên gồm các thông tin: tên, giới tính, điểm toán, điểm lý, điểm hoá. In danh sách (số thứ tự, tên, tổng điểm) các sinh viên nữ có thi lại theo thứ tự tăng dần của tổng điểm.

Bài 16. Cho cấu trúc

```
struct thisinh{
     int       sbd;          //Số báo danh
     char      hoten[25];    //Họ và tên
     float     m1,m2,m3l;    //Điểm ba môn thi
     float     tong;         //Tổng điểm ba môn
} danhsach[100];
```

Viết chương trình (có sử dụng các hàm) để sắp xếp các thí sinh theo thứ tự giảm dần của tổng điểm, và in ra màn hình danh sách đã sắp.

NMLT
MỞ ĐẦU

Trần Phước Tuấn
tranphuoctuan.khoatoan.dhsp@gmail.com
http://baigiang.tranphuoctuan.com

Đề cương bài giảng

- **Mục tiêu môn học:** Cung cấp cho sinh viên các kỹ năng cơ bản để lập trình giải quyết các vấn đề, bài toán. Các chương trình được thể hiện bằng NNLT C. Riêng về ngôn ngữ lập trình C, các sinh viên được cung cấp các kỹ năng:
  - *Đọc và viết được chương trình đơn giản*
  - *Hiểu cấu trúc ngôn ngữ*
  - *Sử dụng thành thạo các thư viện chuẩn*
  - *Nhận biết và sửa chữa các lỗi thường gặp khi lập trình*
- **Các môn học tiên quyết:** không.
- **Nội dung bài giảng**:

Nội dung môn học

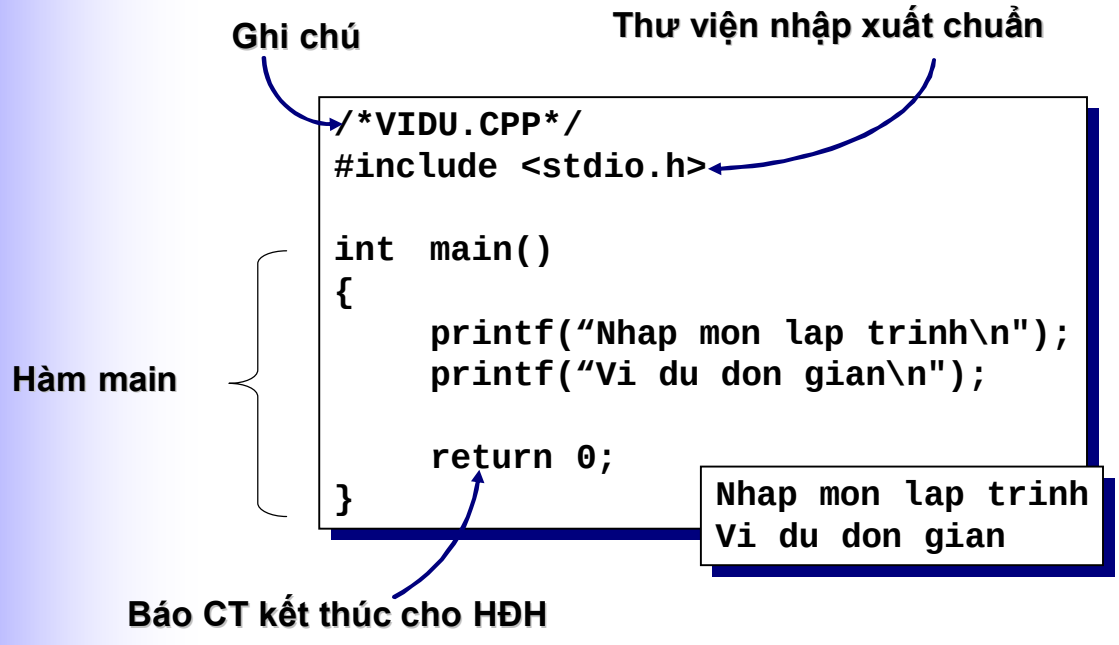
- Tổng quan
- Các kiểu dữ liệu cơ bản
- Lệnh nhập, xuất dữ liệu
- Các cấu trúc điều khiển
- Hàm
- Struct
- Mảng
- Con trỏ
- Chuỗi ký tự

Tổng quan

- Khái niệm chương trình – lập trình
- Cấu trúc một chương trình đơn giản
- Khái niệm Thuật toán – biểu diễn thuật toán
- Khái niệm NNLT, sơ lược lịch sử phát triển NNLT
- Ngôn ngữ lập trình C

Các thành phần của chương trình C

Ví dụ chương trình C



Một số lưu ý từ ví dụ

- Phần ghi chú được trình biên dịch bỏ qua
- Phân biệt chữ in hoa và chữ in thường
- Câu lệnh luôn được kết thúc bằng dấu ;
- Chuỗi ký tự phải ghi giữa cặp nháy kép “
- In xuống dòng dùng ký tự \n
- Chương trình nên thông báo kết quả thực hiện với hệ thống: Tốt – 0, có lỗi – 1, 2, 3 …
- Chương trình có một hàm main

Ví dụ 2

Khai báo 2 biến số nguyên, “a” và “b”

Nhập 2 số nguyên vào a và b

Viết các biểu thức “a”, “b” và “a-b” theo định dạng %i

```
#include <stdio.h>

int  main(void)
{
     int   a, b;

     printf(“Nhap 2 so ngguyen: ");
     scanf("%i %i", &a, &b);

     printf("%i - %i = %i\n", a, b, a - b);

     return 0;
}
```

```
Nhap 2 so nguyen: 21 17
21 - 17 = 4
```

Biến – Variable

```
int a, b;
```

- Chứa dữ liệu có thể thay đổi được trong chương trình.
- Muốn sử dụng phải khai báo.
- Tên: gồm chữ cái, ký số, dấu nối (_), không được bắt đầu bằng ký số.
- Biến khai báo trong khối được gọi là biến cục bộ, không thuộc khối nào được gọi là biến toàn cục
- Có tác dụng trong toàn khối kể từ lúc được khai báo.

Lệnh xuất - printf

Xuất dữ liệu ra màn hình:

```
printf("%i - %i = %i\n", a, b, a - b);
```

- Các ký tự hằng được in nguyên văn
- Các ký tự định dạng được thay bằng giá trị của biểu thức tương ứng:
- %i: ký tự định dạng số nguyên kiểu int
- Các ký tự điều khiển: \n – xuống dòng; \t – dấu tab; \\ – dấu \; \"– dấu " …
- Thư viện: **stdio.h**

Lệnh nhập - scanf

Nhập dữ liệu từ bàn phím

```
scanf("%i %i", &a, &b);
```

- Trong chuỗi định dạng chỉ có ký tự định dạng và khoảng trắng.
- Dữ liệu phải được nhập vào các biến.
- Trước tên biến phải ghi dấu & - toán tử địa chỉ. Nếu không có toán tử địa chỉ, giá trị của biến sẽ không được cập nhật
- Thư viện: **stdio.h**

Kiểu dữ liệu cơ sở trong C

Các kiểu số nguyên của C

- C hỗ trợ khá nhiều kiểu số nguyên
- Các giá trị lớn nhất và nhỏ nhất được định nghĩa trong thư viện "`limits.h`"

Kiểu	định dạng	kích thước	nhỏ nhất	lớn nhất
`char`	%c	1	CHAR_MIN	CHAR_MAX
`unsigned char`	%c	1	0	UCHAR_MAX
`short` [int]	%hi	2	SHRT_MIN	SHRT_MAX
`unsigned short`	%hu	2	0	USHRT_MAX
`int`	%i	**2 or 4**	INT_MIN	INT_MAX
`unsigned int`	%u	**2 or 4**	0	UINT_MAX
`long` [int]	%li	4	LONG_MIN	LONG_MAX
`unsigned long`	%lu	4	0	ULONG_MAX

Ví dụ về số nguyên

```
#include <stdio.h>
#include <limits.h>

int  main()
{
     unsigned long  big = ULONG_MAX;

     printf("minimum int = %i, ", INT_MIN);
     printf("maximum int = %i\n", INT_MAX);
     printf("maximum unsigned = %u\n", UINT_MAX);
     printf("maximum long int = %li\n", LONG_MAX);
     printf("maximum unsigned long = %lu\n", big);

     return 0;
}
```

```
minimum int = -32768, maximum int = 32767
maximum unsigned = 65535
maximum long int = 2147483647
maximum unsigned long = 4294967295
```

Ví dụ kiểu ký tự

In ra mã ASCII của ký tự

```
#include <stdio.h>
#include <limits.h>

int main()
{
    char  lower_a = 'a';
    char  lower_m = 'm';

    printf("minimum char = %i, ", CHAR_MIN);
    printf("maximum char = %i\n", CHAR_MAX);

    printf("Sau '%c' la '%c'\n", lower_a, lower_a + 1);
    printf("Ky tu in hoa '%c'\n", lower_m - 'a' + 'A');

    return 0;
}
```

Trong NNLT C, ký tự chính là số nguyên

```
minimum char = -128, maximum char = 127
Sau 'a' la 'b'
Ky tu in hoa 'M'
```


Số nguyên trong các cơ số khác

- Các hệ cơ số có thể thực hiện được: cơ số 8 (octal), cơ số 10 (decimal), cơ số 16 (hexadecimal)

Số 0: số octal

0x: số hexadecimal

```
#include <stdio.h>

int main(void)
{
    int   dec = 20, oct = 020, hex = 0x20;

    printf("dec=%d, oct=%d, hex=%d\n", dec, oct, hex);
    printf("dec=%d, oct=%o, hex=%x\n", dec, oct, hex);

    return 0;
}
```

```
dec=20, oct=16, hex=32
dec=20, oct=20, hex=20
```

Số thực

- C hỗ trợ nhiều kiểu số thực lưu trữ dấu chấm động.
- Các giá trị lớn nhất và nhỏ nhất được định nghĩa trong thư viện "`float.h`"

Kiểu	định dạng	kích thước	nhỏ nhất	lớn nhất
`float`	`%f %e %g`	4	`FLT_MIN`	`FLT_MAX`
`double`	`%lf %le %lg`	8	`DBL_MIN`	`DBL_MAX`
`long double`	`%Lf %Le %Lg`	10	`LDBL_MIN`	`LDBL_MAX`

Ví dụ số thực:

```
#include <stdio.h>
#include <float.h>

int  main(void)
{
     double f = 3.1416, g = 1.2e-5, h = 5000000000.0;

     printf("f=%lf\tg=%lf\th=%lf\n", f, g, h);
     printf("f=%le\tg=%le\th=%le\n", f, g, h);
     printf("f=%lg\tg=%lg\th=%lg\n", f, g, h);

     printf("f=%7.2lf\tg=%.2le\th=%.4lg\n", f, g, h);

     return 0;
}
```

```
f=3.141600          g=0.000012          h=5000000000.000000
f=3.141600e+00      g=1.200000e-05      h=5.000000e+09
f=3.1416            g=1.2e-05           h=5e+09
f=   3.14           g=1.20e-05          h=5e+09
```

Hằng – Constant

```
const int days_in_week = 7;
```

Chứa dữ liệu không thể thay đổi được trong chương trình.

- Muốn sử dụng phải khai báo.
- Phải có kiểu (tương tự như biến)
- Hằng số có chứa "." hoặc "e" có kiểu `double` (3.5, 1e-7, -1.29e15)
- Hằng số kiểu `float` kết thúc bởi "F" (3.5F, 1e-7F)
- Hằng số kiểu `long double` kết thúc bởi "L" (-1.29e15L, 1e-7L)
- Hằng số không có ".", "e" hoặc "F" có kiểu `int`. Ví dụ: 10000, -35. (Một vài trình biên dịch tự động chuyển thành `long int` nếu giá trị hằng tràn kiểu `int`)
- Khai báo hằng `long int` phải thêm vào cuối "L" (9000000L)

Ví dụ về hằng

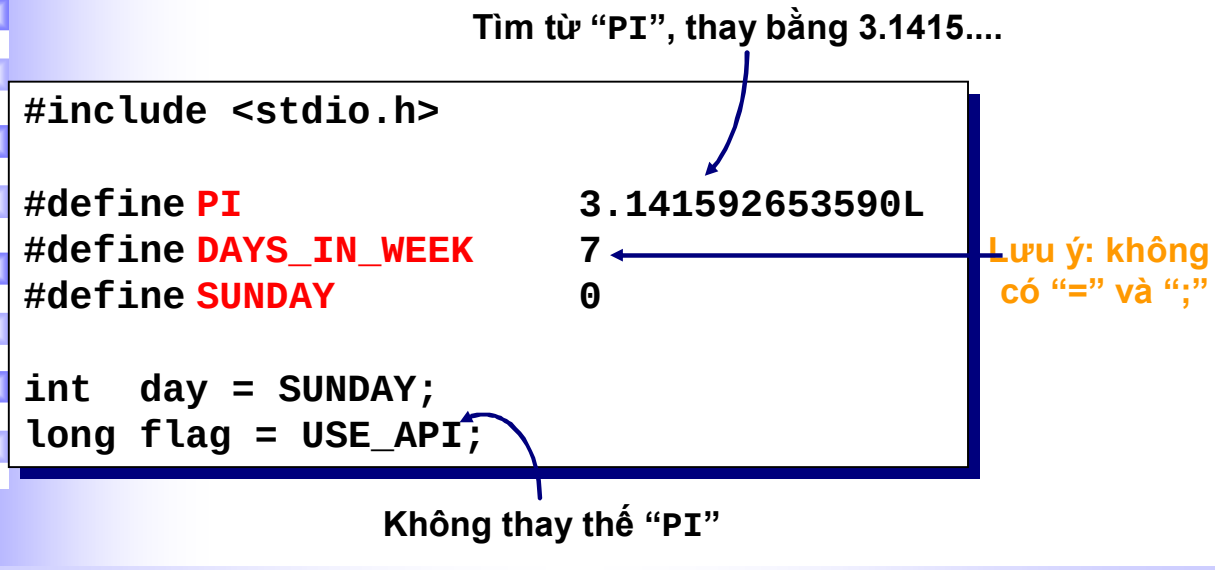
Các hằng pi, days_in_week, sunday được tạo với từ khóa const

```
#include <stdio.h>

int main(void)
{
    const long double pi = 3.141592653590L;
    const int days_in_week = 7;
    const sunday = 0;

    days_in_week = 5;

    return 0;
}
```

Lỗi

Hằng xử lý trước biên dịch

- Các hằng có thể được xác lập trước khi biên dịch
  - Bản chất là tìm kiếm và thay thế
  - Thường được đặt tên với các chữ cái in hoa



Toán tử trong C

Toán tử trong C

- Phép toán số học
- Ép kiểu
- Các toán tử trên bit
- Các toán tử so sánh
- Phép gán
- Toán tử **`sizeof`**
- Biểu thức điều kiện

Toán tử số học

- NNLT C hỗ trợ các phép toán số học:

+	cộng
-	trừ
*	nhân
/	chia
%	chia lấy dư

Lưu ý:

- *"/" cho kết quả phụ thuộc vào kiểu của các toán hạng*
- *"%" không thực hiện được với các số thực*

Ví dụ về toán tử chia "/"

- Trình biên dịch dựa vào kiểu của các toán hạng để quyết định phép chia tương ứng

"i", "j" kiểu int, "/" là phép chia lấy nguyên → k nhận giá trị 1

"f", "g" kiểu double, "/" là phép chia số thực → h nhận giá trị 1.25

Phép chia nguyên, bất kể "h" có kiểu double. Kết quả là 1.00000

```
int  main(void)
{
    int     i = 5,   j = 4,   k;
    double  f = 5.0, g = 4.0, h;

    k = i / j;
    h = f / g;
    h = i / j;

    return 0;
}
```


Ép kiểu

- Ép kiểu làm thay đổi *tạm thời* kiểu của một biến trong một biểu thức.

Phép chia số nguyên được thực hiện, kết quả, 1, được đổi sang kiểu double, 1.00000

```
int  main(void)
{
    int i = 5, j = 4;
    double f;

    f = (double)i / j;
    f = i / (double)j;
    f = (double)i / (double)j;
    f = (double)(i / j);

    return 0;
}
```

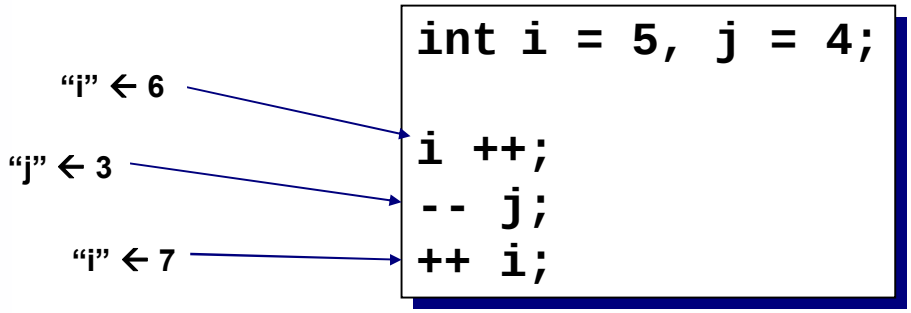
Phép tăng (giảm) 1

- NNLT C có 2 toán tử đặc biệt hỗ trợ việc tăng (giảm) giá trị của một biến thay đổi 1 đơn vị

  ++ tăng 1

  -- giảm 1

- Các toán tử này có thể đặt ở trước hoặc sau biến.



Trước hay sau ?

Thứ tự thực hiện các toán tử ++ và -- phụ thuộc vào vị trí của chúng (trước hay sau) so với biến:

```
#include <stdio.h>

int main(void)
{
    int   i, j = 5;
    i = ++j;
    printf("i=%d, j=%d\n", i, j);
    j = 5;
    i = j++;
    printf("i=%d, j=%d\n", i, j);

    return 0;
}
```

Tương đương:
1. j++;
2. i = j;

Tương đương:
1. i = j;
2. j++;

```
i=6, j=6
i=5, j=6
```

Kiểu luận lý trong C

- Trong C không có kiểu dữ liệu luận lý (thể hiện các giá trị ĐÚNG – SAI), thay vào đó các biểu thức so sánh sẽ cho kết quả là **SỐ**
- Giá trị 0 (0.0) ứng với kết quả SAI (FALSE)
- Các giá trị khác như 1, -3.5, -7, 10.4, … (khác không) đều được xem là ĐÚNG (TRUE)

Các toán tử so sánh

- NNLT C hỗ trợ các phép so sánh:

<	bé hơn
<=	bé hơn hay bằng
>	lớn hơn
>=	lớn hơn hay bằng
==	bằng
!=	không bằng

- Tất cả đều cho kết quả 1 khi so sánh đúng và 0 trong trường hợp ngược lại.

Toán tử luận lý

- NNLT C hỗ trợ các toán tử luận lý:

&&	và (and)
||	hoặc (or)
!	phủ định (not)

- Tất cả đều cho kết quả 1 hoặc 0 tương ứng các trường hợp ĐÚNG hoặc SAI

```
int i, j = 10, k = 28;

i = ((j > 5) && (k < 100)) || (k > 24);
```


Toán tử luận lý

- Lưu ý khi sử dụng các toán tử luận lý:

Nếu không có các dấu (), các phép toán được thực hiện từ trái sang phải

```
if((i < 10) && (a[i] > 0))
    printf("%i\n", a[i]);
```

Nên viết
"i < 10" được kiểm tra trước, nếu không đúng giá trị của biểu thức sẽ là 0 và "a[i] > 0" sẽ không được tính

Không nên: `(a < b < c)`

Nên: `((a < b) && (b< c)`

Toán tử trên bit

- Các toán tử trên bit chỉ có tác dụng trên các kiểu số nguyên:

&	And
|	Or
^	XOr
>>	Đẩy phải
<<	Đẩy trái

Ví dụ về các toán tử trên bit

```
#include <stdio.h>

int main(void)
{
    short a = 0x6eb9;
    short b = 0x5d27;
    unsigned short c = 7097;

    printf("0x%x, ", a & b);
    printf("0x%x, ", a | b);
    printf("0x%x\n", a ^ b);

    printf("%u, ", c << 2);
    printf("%u\n", c >> 1);

    return 0;
}
```

```
0x4c21, 0x7fbf, 0x339e
28388, 3548
```

```
0x6eb9   0110 1110 1011 1001
0x5d27   0101 1101 0010 0111
0x4c21   0100 1100 0010 0001
```

```
0x6eb9   0110 1110 1011 1001
0x5d27   0101 1101 0010 0111
0x7fbf   0111 1111 1011 1111
```

```
0x6eb9   0110 1110 1011 1001
0x5d27   0101 1101 0010 0111
0x339e   0011 0011 1001 1110
```

```
 7097   0001 1011 1011 1001
28388   0110 1110 1110 0100
```

```
 7097   0001 1011 1011 1001
 3548   0000 1101 1101 1100
```

Phép gán

- Có thể sử dụng liên tiếp nhiều phép gán
- Giá trị được gán sẽ sẵn sàng cho lệnh kế tiếp

```
int  i, j, k, l, m, n;

i = j = k = l = m = n = 22;

printf("%i\n", j = 93);
```

"n = 22" gán trước, lại gán "n" cho "m", "m" cho "l", … → i, j, k, l, m, n đều nhận giá trị 22.

"j" được gán 93, giá trị 93 sẽ được in ra màn hình

Phép gán

Lưu ý:

- Vế trái phép gán luôn phải là biến
- Không được nhầm lẫn giữa so sánh bằng "==" và gán "="

```
#include <stdio.h>

int main(void)
{
    int  i = 0;

    if(i = 0)
        printf("i nhan gia tri khong\n");
    else
        printf("i khac khong\n");

    return 0;
}
```

```
i khac khong
```

Một số phép gán đặc biệt

- Các phép gán kết hợp toán tử khác:

  `+=` `-=` `*=` `/=` `%=`

  `&=` `|=` `^=`

  `<<=` `>>=`

- Tổng quát:

  biến ***op*=** **biểu thức**

  tương đương:

  biến = biến *op* (biểu thức)

```
a += 27;
```
```
a = a + 27;
```

```
f /= 9.2;
```
```
f = f / 9.2;
```

```
i *= j + 2;
```
```
i = i * (j + 2);
```


Toán tử `sizeof`

`sizeof(Obj)`

- Cho biết kích thước của đối tượng theo đơn vị byte

```
#include <stdio.h>

int main(void)
{
    long  big;

    printf("\"big\" is %u bytes\n", sizeof(big));
    printf("a short is %u bytes\n", sizeof(short));
    printf("a double is %u bytes\n", sizeof (double));

    return 0;
}
```

```
"big" is 4 bytes
a short is 2 bytes
a double is 8 bytes
```

Biểu thức chọn theo điều kiện

(điều kiện) ? BT1 : BT2

- Biểu thức nhận giá trị BT1 nếu điều kiện khác 0 (ĐÚNG), các trường hợp khác nhận giá trị BT2

```
int i, j = 100, k = -1;

i = (j > k) ? j : k;
```

```
Nếu (j > k)
    i = j;
Ngược lại
    i = k;
```

```
int i, j = 100, k = -1;

i = (j < k) ? j : k;
```

```
Nếu (j < k)
    i = j;
Ngược lại
    i = k;
```

- Có thể định nghĩa sẵn một macro để tìm số lớn:

```
#define max(x, y) ((x>y) ? x : y)
```


Độ ưu tiên của toán tử

- Thứ tự thực hiện các toán tử trong một biểu thức phụ thuộc vào *độ ưu tiên* của chúng.
- Có 15 mức ưu tiên.
- Thông thường, toán tử một ngôi có độ ưu tiên cao hơn toán tử hai ngôi.
- Các cặp dấu ngoặc đơn () thường được dùng để chỉ rõ thứ tự các toán tử.

```
#include <stdio.h>
int main(void)
{
    int  j = 3 * 4 + 48 / 7;
    printf("j = %i\n", j);

    return 0;
}
```

```
j = 18
```

Bảng thứ tự thực hiện các toán tử

Toán tử	Thứ tự (nếu cùng ĐƯT)
() [] -> .	→
! ++ -- - + (cast) * & sizeof	←
* / %	→
+ -	→
<< >>	→
< <= >= >	→
== !=	→
&	→
|	→
^	→
&&	→
||	→
?:	←
= += -= *= /= %= …	←

Luyện tập

```
#include <stdio.h>

int main(void)
{
    int  i = 0, j, k = 7, m = 5, n;
    j = m += 2;
    printf("j = %d\n", j);
    j = k++ > 7;
    printf("j = %d\n", j);
    j = i == 0 & k;
    printf("j = %d\n", j);
    n = !i > k >> 2;
    printf("n = %d\n", n);
    return 0;
}
```

Tóm lược

- Một số thành phần của chương trình trong C
- Hàm main
- Lệnh xuất / nhập – printf / scanf
- Biến
- Các kiểu số nguyên và số thực
- Hằng – 2 cách khai báo
- Sử dụng toán tử trong C

NGÔN NGỮ LẬP TRÌNH C#

NGUYỄN NGỌC BÌNH PHƯƠNG
THÁI THANH PHONG

CÁC GIẢI PHÁP LẬP TRÌNH C#

NXB GIAO THÔNG VẬN TẢI

Visual Studio.net

NGÔN NGỮ

LẬP TRÌNH C#

Mục Lục

Tham Khảo

Giáo trình "*Ngôn ngữ Lập trình C#*" được biên dịch và tổng hợp từ:

Programming C#, Jesse Liberty, O'Reilly.

C# in 21 Days, Bradley L.Jones, SAMS.

Windows Forms Programming with C#, Erik Brown, Manning.

MSDN Library – April 2002.

Quy ước

Giáo trình sử dụng một số quy ước như sau:

Các thuật ngữ được giới thiệu lần đầu tiên sẽ *in nghiêng*.

Mã nguồn của chương trình minh họa dùng font Verdana -10.

Các từ khóa của C# dùng font **Verdana-10, đậm** hoặc Verdana-10, bình thường.

Tên namspace, lớp, đối tượng, phương thức, thuộc tính, sự kiện... dùng font Verdana-10.

Kết quả của chương trình xuất ra màn hình console dùng font `Courier New-10.`

Chương 1

MICROSOFT .NET

- **Tình hình trước khi MS.NET ra đời**
  - **Nguồn gốc của .NET**
- **Microsoft .NET**
  - **Tổng quan**
  - **Kiến trúc .NET Framework**
  - **Common Language Runtime (CLR)**
  - **Thư viện .NET Framework**
  - **Phát triển ứng dụng client**
- **Biên dịch và MSIL**
- **Ngôn ngữ C#**

Tình hình trước khi MS.NET ra đời

Trong lĩnh vực công nghệ thông tin của thế giới ngày nay, với sự phát triển liên tục và đa dạng nhất là phần mềm, các hệ điều hành, các môi trường phát triển, các ứng dụng liên tục ra đời. Tuy nhiên, đôi khi việc phát triển không đồng nhất và nhất là do lợi ích khác nhau của các công ty phần mềm lớn làm ảnh hưởng đến những người xây dựng phần mềm.

Cách đây vài năm Java được Sun viết ra, đã có sức mạnh đáng kể, nó hướng tới việc chạy trên nhiều hệ điều hành khác nhau, độc lập với bộ xử lý (Intel, Risc,…). Đặc biệt là Java rất thích hợp cho việc viết các ứng dụng trên Internet. Tuy nhiên, Java lại có hạn chế về mặt tốc độ và trên thực tế vẫn chưa thịnh hành. Mặc dù Sun Corporation và IBM có đẩy mạnh Java, nhưng Microsoft đã dùng ASP để làm giảm khả năng ảnh hưởng của Java.

Để lập trình trên Web, lâu nay người ta vẫn dùng CGI-Perl và gần đây nhất là PHP, một ngôn ngữ giống như Perl nhưng tốc độ chạy nhanh hơn. Ta có thể triển khai Perl trên Unix/Linux hay MS Windows. Tuy nhiên có nhiều người không thích dùng do bản thân ngôn ngữ hay các qui ước khác thường và Perl không được phát triển thống nhất, các công cụ được xây dựng cho Perl tuy rất mạnh nhưng do nhiều nhóm phát triển và người ta không đảm bảo rằng tương lai của nó ngày càng tốt đẹp hơn.

Trong giới phát triển ứng dụng trên Windows ta có thể viết ứng dụng bằng Visual C++, Delphi hay Visual Basic, đây là một số công cụ phổ biến và mạnh. Trong đó Visual C++ là một ngôn ngữ rất mạnh và cũng rất khó sử dụng. Visual Basic thì đơn giản dễ học, dễ dùng nhất nên rất thông dụng. Lý do chính là Visual Basic giúp chúng ta có thể viết chương trình trên Windows dễ dàng mà không cần thiết phải biết nhiều về cách thức MS Windows hoạt động, ta chỉ cần biết một số kiến thức căn bản tối thiểu về MS Windows là có thể lập trình được. Do đó theo quan điểm của Visual Basic nên nó liên kết với Windows là điều tự nhiên và dễ hiểu, nhưng hạn chế là Visual Basic không phải ngôn ngữ hướng đối tượng (Object Oriented).

Delphi là hậu duệ của Turbo Pascal của Borland. Nó cũng giống và tương đối dễ dùng như Visual Basic. Delphi là một ngôn ngữ hướng đối tượng. Các điều khiển dùng trên Form của Delphi đều được tự động khởi tạo mã nguồn. Tuy nhiên, chức năng khởi động mã nguồn này của Delphi đôi khi gặp rắc rối khi có sự can thiệp của người dùng vào. Sau này khi công ty Borland bị bán và các chuyên gia xây dựng nên Delphi đã chạy qua bên Microsoft, và Delphi không còn được phát triển tốt nữa, người ta không dám đầu tư triển khai phần mềm vào Delphi. Công ty sau này đã phát triển dòng sản phẩm Jbuilder (dùng Java) không còn quan tâm đến Delphi.

Tuy Visual Basic bền hơn do không cần phải khởi tạo mã nguồn trong Form khi thiết kế nhưng Visual Basic cũng có nhiều khuyết điểm :

Không hỗ trợ thiết kế hướng đối tượng, nhất là khả năng thừa kế (inheritance).

Giới hạn về việc chạy nhiều tiểu trình trong một ứng dụng, ví dụ ta không thể dùng Visual Basic để viết một Service kiểu NT.

Khả năng xử lý lỗi rất yếu, không thích hợp trong môi trường Multi- tier

Khó dùng chung với ngôn ngữ khác như C++.

Không có User Interface thích hợp cho Internet.

Do Visual Basic không thích hợp cho viết các ứng Web Server nên Microsoft tạo ra ASP (Active Server Page). Các trang ASP này vừa có tag HTML vừa chứa các đoạn script (VBScript, JavaScript) nằm lẫn lộn nhau. Khi xử lý một trang ASP, nếu là tag HTML thì sẽ được gởi thẳng qua Browser, còn các script thì sẽ được chuyển thành các dòng HTML rồi gởi đi, ngoại trừ các function hay các sub trong ASP thì vị trí các script khác rất quan trọng.

Khi một số chức năng nào được viết tốt người ta dịch thành ActiveX và đưa nó vào Web Server. Tuy nhiên vì lý do bảo mật nên các ISP (Internet Service Provider) làm máy chủ cho Web site thường rất dè đặt khi cài ActiveX lạ trên máy của họ. Ngoài ra việc tháo gỡ các phiên bản của ActiveX này là công việc rất khó, thường xuyên làm cho Administrator nhức đầu. Những người đã từng quản lý các version của DLL trên Windows điều than phiền tại sao phải đăng ký các DLL và nhất là chỉ có thể đăng ký một phiên bản của DLL mà thôi. Và từ "*DLL Hell*" xuất hiện tức là địa ngục DLL…

Sau này để giúp cho việc lập trình ASP nhanh hơn thì công cụ Visual InterDev, một IDE (Integrated Development Environment) ra đời. Visual InterDev tạo ra các Design Time Controls cho việc thiết kế các điều khiển trên web,… Tiếc thay Visual InterDev không bền vững lắm nên sau một thời gian thì các nhà phát triển đã rời bỏ nó.

Tóm lại bản thân của ASP hãy còn một số khuyết điểm quan trọng, nhất là khi chạy trên Internet Information Server với Windows NT 4, ASP không đáng tin cậy lắm.

Tóm lại trong giới lập trình theo Microsoft thì việc lập trình trên desktop cho đến lập trình hệ phân tán hay trên web là không được nhịp nhàng cho lắm. Để chuyển được từ lập trình client hay desktop đến lập trình web là một chặng đường dài.

Nguồn gốc .NET

Đầu năm 1998, sau khi hoàn tất phiên bản Version 4 của Internet Information Server (IIS), các đội ngũ lập trình ở Microsoft nhận thấy họ còn rất nhiều sáng kiến để kiện toàn IIS. Họ bắt đầu xây dựng một kiến trúc mới trên nền tảng ý tưởng đó và đặt tên là Next Generation Windows Services (NGWS).

Sau khi Visual Basic được trình làng vào cuối 1998, dự án kế tiếp mang tên Visual Studio 7 được xác nhập vào NGWS. Đội ngũ COM+/MTS góp vào một universal runtime cho tất cả ngôn ngữ lập trình chung trong Visual Studio, và tham vọng của họ cung cấp cho các ngôn ngữ lập trình của các công ty khác dùng chung luôn. Công việc này được xúc tiến một cách hoàn toàn bí mật mãi cho đến hội nghị Professional Developers' Conference ở Orlado vào tháng 7/2000. Đến tháng 11/2000 thì Microsoft đã phát hành bản Beta 1 của .NET gồm 3 đĩa CD. Tính đến lúc này thì Microsoft đã làm việc với .NET gần 3 năm rồi, do đó bản Beta 1 này tương đối vững chắc.

.NET mang dáng dấp của những sáng kiến đã được áp dụng trước đây như p-code trong UCSD Pascal cho đến Java Virtual Machine. Có điều là Microsoft góp nhặt những sáng kiến của người khác, kết hợp với sáng kiến của chính mình để làm nên một sản phẩm hoàn chỉnh từ bên trong lẫn bên ngoài. Hiện tại Microsoft đã công bố phiên bản release của .NET.

Thật sự Microsoft đã đặt cược vào .NET vì theo thông tin của công ty, đã tập trung 80% sức mạnh của Microsoft để nghiên cứu và triển khai .NET (bao gồm nhân lực và tài chính ?), tất cả các sản phẩm của Microsoft sẽ được chuyển qua .NET.

Microsoft .NET

Tổng quan

Microsoft .NET gồm 2 phần chính : Framework và Integrated Development Environment (IDE). Framework cung cấp những gì cần thiết và căn bản, chữ Framework có nghĩa là khung hay khung cảnh trong đó ta dùng những hạ tầng cơ sở theo một qui ước nhất định để công việc được trôi chảy. IDE thì cung cấp một môi trường giúp chúng ta triển khai dễ dàng, và nhanh chóng các ứng dụng dựa trên nền tảng .NET. Nếu không có IDE chúng ta cũng có thể

dùng một trình soạn thảo ví như Notepad hay bất cứ trình soạn thảo văn bản nào và sử dụng command line để biên dịch và thực thi, tuy nhiên việc này mất nhiều thời gian. Tốt nhất là chúng ta dùng IDE phát triển các ứng dụng, và cũng là cách dễ sử dụng nhất.

Thành phần Framework là quan trọng nhất .NET là cốt lõi và tinh hoa của môi trường, còn IDE chỉ là công cụ để phát triển dựa trên nền tảng đó thôi. Trong .NET toàn bộ các ngôn ngữ C#, Visual C++ hay Visual Basic.NET đều dùng cùng một IDE.

Tóm lại Microsoft .NET là nền tảng cho việc xây dựng và thực thi các ứng dụng phân tán thế hệ kế tiếp. Bao gồm các ứng dụng từ client đến server và các dịch vụ khác. Một số tính năng của Microsoft .NET cho phép những nhà phát triển sử dụng như sau:

Một mô hình lập trình cho phép nhà phát triển xây dựng các ứng dụng dịch vụ web và ứng dụng client với Extensible Markup Language (XML).

Tập hợp dịch vụ XML Web, như Microsoft .NET My Services cho phép nhà phát triển đơn giản và tích hợp người dùng kinh nghiệm.

Cung cấp các server phục vụ bao gồm: Windows 2000, SQL Server, và BizTalk Server, tất cả điều tích hợp, hoạt động, và quản lý các dịch vụ XML Web và các ứng dụng.

Các phần mềm client như Windows XP và Windows CE giúp người phát triển phân phối sâu và thuyết phục người dùng kinh nghiệm thông qua các dòng thiết bị.

Nhiều công cụ hỗ trợ như Visual Studio .NET, để phát triển các dịch vụ Web XML, ứng dụng trên nền Windows hay nền web một cách dể dàng và hiệu quả.

Kiến trúc .NET Framework

.NET Framework là một platform mới làm đơn giản việc phát triển ứng dụng trong môi trường phân tán của Internet. .NET Framework được thiết kế đầy đủ để đáp ứng theo quan điểm sau:

Để cung cấp một môi trường lập trình hướng đối tượng vững chắc, trong đó mã nguồn đối tượng được lưu trữ và thực thi một cách cục bộ. Thực thi cục bộ nhưng được phân tán trên Internet, hoặc thực thi từ xa.

Để cung cấp một môi trường thực thi mã nguồn mà tối thiểu được việc đóng gói phần mềm và sự tranh chấp về phiên bản.

Để cung cấp một môi trường thực thi mã nguồn mà đảm bảo việc thực thi an toàn mã nguồn, bao gồm cả việc mã nguồn được tạo bởi hãng thứ ba hay bất cứ hãng nào mà tuân thủ theo kiến trúc .NET.

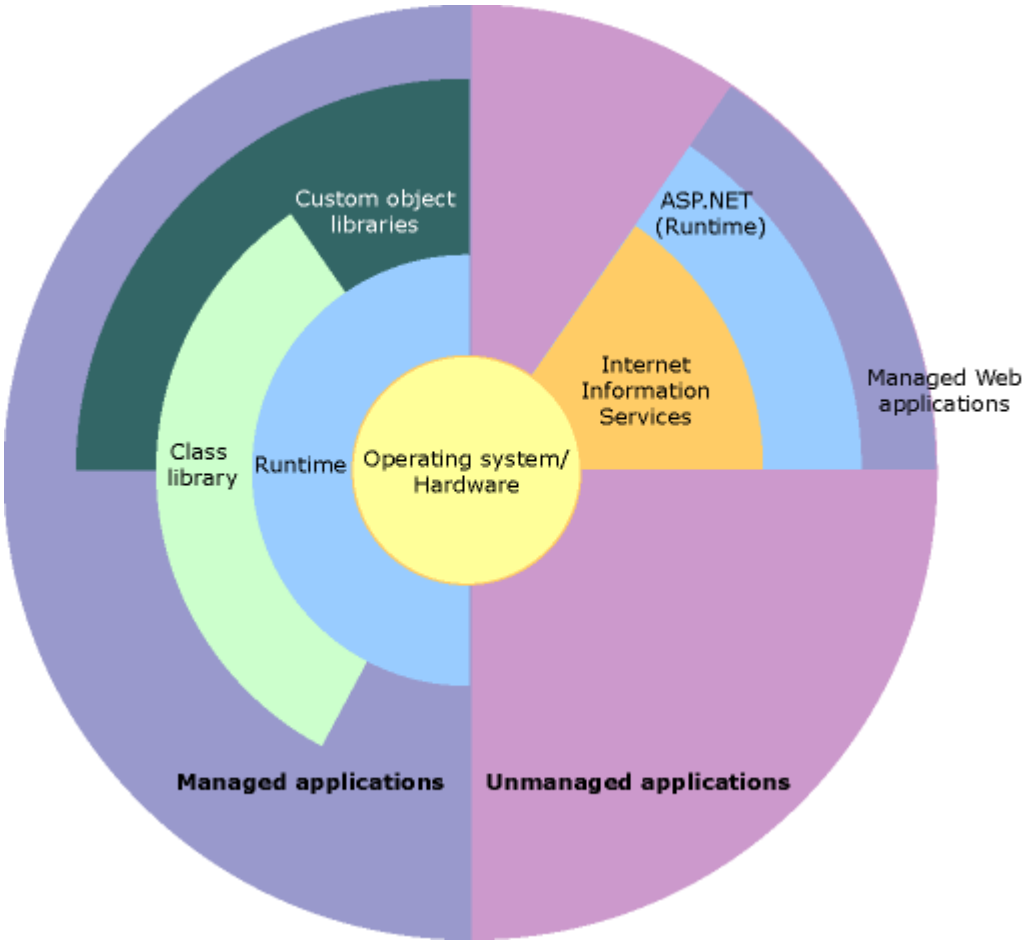
Để cung cấp một môi trường thực thi mã nguồn mà loại bỏ được những lỗi thực hiện các script hay môi trường thông dịch.

Để làm cho những người phát triển có kinh nghiệm vững chắc có thể nắm vững nhiều kiểu ứng dụng khác nhau. Như là từ những ứng dụng trên nền Windows đến những ứng dụng dựa trên web.

Để xây dựng tất cả các thông tin dựa triên tiêu chuẩn công nghiệp để đảm bảo rằng mã nguồn trên .NET có thể tích hợp với bất cứ mã nguồn khác.

.NET Framework có hai thành phần chính: Common Language Runtime (CLR) và thư viện lớp .NET Framework. CLR là nền tảng của .NET Framework. Chúng ta có thể hiểu runtime như là một agent quản lý mã nguồn khi nó được thực thi, cung cấp các dịch vụ cốt lõi như: quản lý bộ nhớ, quản lý tiểu trình, và quản lý từ xa. Ngoài ra nó còn thúc đẩy việc sử dụng kiểu an toàn và các hình thức khác của việc chính xác mã nguồn, đảm bảo cho việc thực hiện được bảo mật và mạnh mẽ. Thật vậy, khái niệm quản lý mã nguồn là nguyên lý nền tảng của runtime. Mã nguồn mà đích tới runtime thì được biết như là mã nguồn được quản lý (managed code). Trong khi đó mã nguồn mà không có đích tới runtime thì được biết như mã nguồn không được quản lý (unmanaged code).

Thư viện lớp, một thành phần chính khác của .NET Framework là một tập hợp hướng đối tượng của các kiểu dữ liệu được dùng lại, nó cho phép chúng ta có thể phát triển những ứng dụng từ những ứng dụng truyền thống command-line hay những ứng dụng có giao diện đồ họa (GUI) đến những ứng dụng mới nhất được cung cấp bởi ASP.NET, như là Web Form và dịch vụ XML Web.



Hình 1.1: Mô tả các thành phần trong .NET Framework.

Common Language Runtime (CLR)

Như đã đề cập thì CLR thực hiện quản lý bộ nhớ, quản lý thực thi tiểu trình, thực thi mã nguồn, xác nhận mã nguồn an toàn, biên bịch và các dịch vụ hệ thống khác. Những đặc tính trên là nền tảng cơ bản cho những mã nguồn được quản lý chạy trên CLR.

Do chú trọng đến bảo mật, những thành phần được quản lý được cấp những mức độ quyền hạn khác nhau, phụ thuộc vào nhiều yếu tố nguyên thủy của chúng như: liên quan đến Internet, hệ thống mạng trong nhà máy, hay một máy tính cục bộ. Điều này có nghĩa rằng, một thành phần được quản lý có thể có hay không có quyền thực hiện một thao tác truy cập tập tin, thao tác truy cập registry, hay các chức năng nhạy cảm khác.

CLR thúc đẩy việc mã nguồn thực hiện việc truy cập được bảo mật. Ví dụ, người sử dụng giới hạn rằng việc thực thi nhúng vào trong một trang web có thể chạy được hoạt hình trên màn hình hay hát một bản nhạc, nhưng không thể truy cập được dữ liệu riêng tư, tập tin hệ thống, hay truy cập mạng. Do đó, đặc tính bảo mật của CLR cho phép những phần mềm đóng gói trên Inernet có nhiều đặc tính mà không ảnh hưởng đến việc bảo mật hệ thống.

CLR còn thúc đẩy cho mã nguồn được thực thi mạnh mẽ hơn bằng việc thực thi mã nguồn chính xác và sự xác nhận mã nguồn. Nền tảng của việc thực hiện này là Common Type System (CTS). CTS đảm bảo rằng những mã nguồn được quản lý thì được tự mô tả (self-describing). Sự khác nhau giữa Microsoft và các trình biên dịch ngôn ngữ của hãng thứ ba là việc tạo ra các mã nguồn được quản lý có thể thích hợp với CTS. Điều này thì mã nguồn được quản lý có thể sử dụng những kiểu được quản lý khác và những thể hiện, trong khi thúc đẩy nghiêm ngặt việc sử dụng kiểu dữ liệu chính xác và an toàn.

Thêm vào đó, môi trường được quản lý của runtime sẽ thực hiện việc tự động xử lý layout của đối tượng và quản lý những tham chiếu đến đối tượng, giải phóng chúng khi chúng không còn được sử dụng nữa. Việc quản lý bộ nhớ tự động này còn giải quyết hai lỗi chung của ứng dụng: thiếu bộ nhớ và tham chiếu bộ nhớ không hợp lệ.

Trong khi runtime được thiết kế cho những phần mềm của tương lai, nó cũng hỗ trợ cho phân mềm ngày nay và trước đây. Khả năng hoạt động qua lại giữa mã nguồn được quản lý và mã nguồn không được quản lý cho phép người phát triển tiếp tục sử dụng những thành phần cần thiết của COM và DLL.

Rutime được thiết kế để cải tiến hiệu suất thực hiện. Mặc dù CLR cung cấp nhiều các tiêu chuẩn dịch vụ runtime, nhưng mã nguồn được quản lý không bao giờ được dịch. Có một đặc tính gọi là Just-in-Time (JIT) biên dịch tất cả những mã nguồn được quản lý vào trong ngôn ngữ máy của hệ thống vào lúc mà nó được thực thi. Khi đó, trình quản lý bộ nhớ xóa bỏ những phân mảnh bộ nhớ nếu có thể được và gia tăng tham chiếu bộ nhớ cục bộ, và kết quả gia tăng hiệu quả thực thi.

Thư viện lớp .NET Framework

Thư viện lớp .NET Framework là một tập hợp những kiểu dữ liệu được dùng lại và được kết hợp chặt chẽ với Common Language Runtime. Thư viện lớp là hướng đối tượng cung cấp những kiểu dữ liệu mà mã nguồn được quản lý của chúng ta có thể dẫn xuất. Điều này không chỉ làm cho những kiểu dữ liệu của .NET Framework dễ sử dụng mà còn làm giảm thời gian liên quan đến việc học đặc tính mới của .NET Framework. Thêm vào đó, các thành phần của các hãng thứ ba có thể tích hợp với những lớp trong .NET Framework.

Cũng như mong đợi của người phát triển với thư viện lớp hướng đối tượng, kiểu dữ liệu . NET Framework cho phép người phát triển thiết lập nhiều mức độ thông dụng của việc lập trình, bao gồm các nhiệm vụ như: quản lý chuỗi, thu thập hay chọn lọc dữ liệu, kết nối với cơ cở dữ liệu, và truy cập tập tin. Ngoài những nhiệm vụ thông dụng trên. Thư viện lớp còn đưa vào những kiểu dữ liệu để hỗ trợ cho những kịch bản phát triển chuyên biệt khác. Ví dụ người phát triển có thể sử dụng .NET Framework để phát triển những kiểu ứng dụng và dịch vụ như sau:

Ứ ng dụng Console
Ứ ng dụng giao diện GUI trên Windows (Windows Forms)
Ứ ng dụng ASP.NET
Dịch vụ XML Web
Dịch vụ Windows

Trong đó những lớp Windows Forms cung cấp một tập hợp lớn các kiểu dữ liệu nhằm làm đơn giản việc phát triển các ứng dụng GUI chạy trên Windows. Còn nếu như viết các ứng dụng ASP.NET thì có thể sử dụng các lớp Web Forms trong thư viện .NET Framework.

Phát triển ứng dụng Client

Những ứng dụng client cũng gần với những ứng dụng kiểu truyền thống được lập trình dựa trên Windows. Đây là những kiểu ứng dụng hiển thị những cửa sổ hay những form trên desktop cho phép người dùng thực hiện một thao tác hay nhiệm vụ nào đó. Những ứng dụng client bao gồm những ứng dụng như xử lý văn bản, xử lý bảng tính, những ứng dụng trong lĩnh vực thương mại như công cụ nhập liệu, công cụ tạo báo cáo...Những ứng dụng client này thường sử dụng những cửa sổ, menu, toolbar, button hay các thành phần GUI khác, và chúng thường truy cập các tài nguyên cục bộ như là các tập tin hệ thống, các thiết bị ngoại vi như máy in.

Một loại ứng dụng client khác với ứng dụng truyền thống như trên là ActiveX control (hiện nay nó được thay thế bởi các Windows Form control) được nhúng vào các trang web trên Internet. Các ứng dụng này cũng giống như những ứng dụng client khác là có thể truy cập tài nguyên cục bộ.

Trong quá khứ, những nhà phát triển có thể tạo các ứng dụng sử dụng C/C++ thông qua kết nối với MFC hoặc sử dụng môi trường phát triển ứng dụng nhanh (RAD: Rapid

Application Development). .NET Framework tích hợp diện mạo của những sản phẩm thành một. Môi trường phát triển cố định làm đơn giản mạnh mẽ sự phát triển của ứng dụng client.

Những lớp .NET Framework chứa trong .NET Framework được thiết kế cho việc sử dụng phát triển các GUI. Điều này cho phép người phát triển nhanh chóng và dễ dàng tạo các cửa sổ, button, menu, toolbar, và các thành phần khác trong các ứng dụng được viết phục vụ cho lĩnh vực thương mại. Ví dụ như, .NET cung cấp những thuộc tính đơn giản để hiệu chỉnh các hiệu ứng visual liên quan đến form. Trong vài trường hợp hệ điều hành không hỗ trợ việc thay đổi những thuộc tính này một cách trực tiếp, và trong trường hợp này .NET tự động tạo lại form. Đây là một trong nhiều cách mà .NET tích hợp việc phát triển giao diện làm cho mã nguồn đơn giản và mạnh mẽ hơn.

Không giống như ActiveX control, Windows Form control có sự truy cập giới hạn đến máy của người sử dụng. Điều này có nghĩa rằng mà nguồn thực thi nhị phân có thể truy cập một vài tài nguyên trong máy của người sử dụng (như các thành phần đồ họa hay một số tập tin được giới hạn) mà không thể truy cập đến những tài nguyên khác. Nguyên nhân là sự bảo mật truy cập của mã nguồn. Lúc này các ứng dụng được cài đặt trên máy người dùng có thể an toàn để đưa lên Internet

Biên dịch và MSIL

Trong .NET Framework, chương trình không được biên dịch vào các tập tin thực thi mà thay vào đó chúng được biên dịch vào những tập tin trung gian gọi là Microsoft Intermediate Language (MSIL). Những tập tin MSIL được tạo ra từ C# cũng tương tự như các tập tin MSIL được tạo ra từ những ngôn ngữ khác của .NET, platform ở đây không cần biết ngôn ngữ của mã nguồn. Điều quan trọng chính yếu của CLR là chung (common), cùng một runtime hỗ trợ phát triển trong C# cũng như trong VB.NET.

Mã nguồn C# được biên dịch vào MSIL khi chúng ta build project. Mã MSIL này được lưu vào trong một tập tin trên đĩa. Khi chúng ta chạy chương trình, thì MSIL được biên dịch một lần nữa, sử dụng trình biên dịch Just-In-Time (JIT). Kết quả là mã máy được thực thi bởi bộ xử lý của máy.

Trình biên dịch JIT tiêu chuẩn thì thực hiện theo yêu cầu. Khi một phương thức được gọi, trình biên dịch JIT phân tích MSIL và tạo ra sản phẩm mã máy có hiệu quả cao, mã này có thể chạy rất nhanh. Trình biên dịch JIT đủ thông minh để nhận ra khi một mã đã được biên dịch, do vậy khi ứng dụng chạy thì việc biên dịch chỉ xảy ra khi cần thiết, tức là chỉ biên dịch mã MSIL chưa biên dịch ra mã máy. Khi đó một ứng dụng .NET thực hiện, chúng có xu hướng là chạy nhanh và nhanh hơn nữa, cũng như là những mã nguồn được biên dịch rồi thì được dùng lại.

Do tất cả các ngôn ngữ .NET Framework cùng tạo ra sản phẩm MSIL giống nhau, nên kết quả là một đối tượng được tạo ra từ ngôn ngữ này có thể được truy cập hay được dẫn xuất từ

một đối tượng của ngôn ngữ khác trong .NET. Ví dụ, người phát triển có thể tạo một lớp cơ sở trong VB.NET và sau đó dẫn xuất nó trong C# một cách dễ dàng.

Ngôn ngữ C#

Ngôn ngữ C# khá đơn giản, chỉ khoảng 80 từ khóa và hơn mười mấy kiểu dữ liệu được xây dựng sẵn. Tuy nhiên, ngôn ngữ C# có ý nghĩa cao khi nó thực thi những khái niệm lập trình hiện đại. C# bao gồm tất cả những hỗ trợ cho cấu trúc, thành phần component, lập trình hướng đối tượng. Những tính chất đó hiện diện trong một ngôn ngữ lập trình hiện đại. Và ngôn ngữ C# hội đủ những điều kiện như vậy, hơn nữa nó được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

Ngôn ngữ C# được phát triển bởi đội ngũ kỹ sư của Microsoft, trong đó người dẫn đầu là Anders Hejlsberg và Scott Wiltamuth. Cả hai người này điều là những người nổi tiếng, trong đó Anders Hejlsberg được biết đến là tác giả của Turbo Pascal, một ngôn ngữ lập trình PC phổ biến. Và ông đứng đầu nhóm thiết kế Borland Delphi, một trong những thành công đầu tiên của việc xây dựng môi trường phát triển tích hợp (IDE) cho lập trình client/server.

Phần cốt lõi hay còn gọi là trái tim của bất cứ ngôn ngữ lập trình hướng đối tượng là sự hỗ trợ của nó cho việc định nghĩa và làm việc với những lớp. Những lớp thì định nghĩa những kiểu dữ liệu mới, cho phép người phát triển mở rộng ngôn ngữ để tạo mô hình tốt hơn để giải quyết vấn đề. Ngôn ngữ C# chứa những từ khóa cho việc khai báo những kiểu lớp đối tượng mới và những phương thức hay thuộc tính của lớp, và cho việc thực thi đóng gói, kế thừa, và đa hình, ba thuộc tính cơ bản của bất cứ ngôn ngữ lập trình hướng đối tượng.

Trong ngôn ngữ C# mọi thứ liên quan đến khai báo lớp điều được tìm thấy trong phần khai báo của nó. Định nghĩa một lớp trong ngôn ngữ C# không đòi hỏi phải chia ra tập tin header và tập tin nguồn giống như trong ngôn ngữ C++. Hơn thế nữa, ngôn ngữ C# hỗ trợ kiểu XML, cho phép chèn các tag XML để phát sinh tự động các document cho lớp.

C# cũng hỗ trợ giao diện interface, nó được xem như một cam kết với một lớp cho những dịch vụ mà giao diện quy định. Trong ngôn ngữ C#, một lớp chỉ có thể kế thừa từ duy nhất một lớp cha, tức là không cho đa kế thừa như trong ngôn ngữ C++, tuy nhiên một lớp có thể thực thi nhiều giao diện. Khi một lớp thực thi một giao diện thì nó sẽ hứa là nó sẽ cung cấp chức năng thực thi giao diện.

Trong ngôn ngữ C#, những cấu trúc cũng được hỗ trợ, nhưng khái niệm về ngữ nghĩa của nó thay đổi khác với C++. Trong C#, một cấu trúc được giới hạn, là kiểu dữ liệu nhỏ gọn, và khi tạo thể hiện thì nó yêu cầu ít hơn về hệ điều hành và bộ nhớ so với một lớp. Một cấu trúc thì không thể kế thừa từ một lớp hay được kế thừa nhưng một cấu trúc có thể thực thi một giao diện.

Ngôn ngữ C# cung cấp những đặc tính hướng thành phần (component-oriented), như là những thuộc tính, những sự kiện. Lập trình hướng thành phần được hỗ trợ bởi CLR cho phép lưu trữ metadata với mã nguồn cho một lớp. Metadata mô tả cho một lớp, bao gồm những

phương thức và những thuộc tính của nó, cũng như những sự bảo mật cần thiết và những thuộc tính khác. Mã nguồn chứa đựng những logic cần thiết để thực hiện những chức năng của nó.. Do vậy, một lớp được biên dịch như là một khối self-contained, nên môi trường hosting biết được cách đọc metadata của một lớp và mã nguồn cần thiết mà không cần những thông tin khác để sử dụng nó.

Một lưu ý cuối cùng về ngôn ngữ C# là ngôn ngữ này cũng hỗ trợ việc truy cập bộ nhớ trực tiếp sử dụng kiểu con trỏ của C++ và từ khóa cho dấu ngoặc [] trong toán tử. Các mã nguồn này là không an toàn (unsafe). Và bộ giải phóng bộ nhớ tự động của CLR sẽ không thực hiện việc giải phóng những đối tượng được tham chiếu bằng sử dụng con trỏ cho đến khi chúng được giải phóng.

Chương 2

NGÔN NGỮ C#

- **Tại sao phải sử dụng ngôn ngữ C#**
  - **C# là ngôn ngữ đơn giản**
  - **C# là ngôn ngữ hiện đại**
  - **C# là ngôn ngữ hướng đối tượng**
  - **C# là ngôn ngữ mạnh mẽ**
  - **C# là ngôn ngữ ít từ khóa**
  - **C# là ngôn ngữ module hóa**
  - **C# sẽ là ngôn ngữ phổ biến**
- **Ngôn ngữ C# và những ngôn ngữ khác**
- **Các bước chuẩn bị cho chương trình**
- **Chương trình C# đơn giản**
- **Phát triển chương trình minh họa**
- **Câu hỏi & bài tập**

Tại sao phải sử dụng ngôn ngữ C#

Nhiều người tin rằng không cần thiết có một ngôn ngữ lập trình mới. Java, C++, Perl, Microsoft Visual Basic, và những ngôn ngữ khác được nghĩ rằng đã cung cấp tất cả những chức năng cần thiết.

Ngôn ngữ C# là một ngôn ngữ được dẫn xuất từ C và C++, nhưng nó được tạo từ nền tảng phát triển hơn. Microsoft bắt đầu với công việc trong C và C++ và thêm vào những đặc tính mới để làm cho ngôn ngữ này dễ sử dụng hơn. Nhiều trong số những đặc tính này khá giống với những đặc tính có trong ngôn ngữ Java. Không dừng lại ở đó, Microsoft đưa ra một số mục đích khi xây dựng ngôn ngữ này. Những mục đích này được được tóm tắt như sau:

C# là ngôn ngữ đơn giản

C# là ngôn ngữ hiện đại

C# là ngôn ngữ hướng đối tượng

C# là ngôn ngữ mạnh mẽ và mềm dẻo

C# là ngôn ngữ có ít từ khóa
C# là ngôn ngữ hướng module
C# sẽ trở nên phổ biến

C# là ngôn ngữ đơn giản

C# loại bỏ một vài sự phức tạp và rối rắm của những ngôn ngữ như Java và c++, bao gồm việc loại bỏ những macro, những template, đa kế thừa, và lớp cơ sở ảo (virtual base class). Chúng là những nguyên nhân gây ra sự nhầm lẫn hay dẫn đến những vấn đề cho các người phát triển C++. Nếu chúng ta là người học ngôn ngữ này đầu tiên thì chắc chắn là ta sẽ không trải qua những thời gian để học nó! Nhưng khi đó ta sẽ không biết được hiệu quả của ngôn ngữ C# khi loại bỏ những vấn đề trên.

Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++. Nếu chúng ta thân thiện với C và C++ hoặc thậm chí là Java, chúng ta sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và những chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để làm cho ngôn ngữ đơn giản hơn. Một vài trong các sự cải tiến là loại bỏ các dư thừa, hay là thêm vào những cú pháp thay đổi. Ví dụ như, trong C++ có ba toán tử làm việc với các thành viên là ::, . , và ->. Để biết khi nào dùng ba toán tử này cũng phức tạp và dễ nhầm lẫn. Trong C#, chúng được thay thế với một toán tử duy nhất gọi là . (dot). Đối với người mới học thì điều này và những việc cải tiến khác làm bớt nhầm lẫn và đơn giản hơn.

Ghi chú: Nếu chúng ta đã sử dụng Java và tin rằng nó đơn giản, thì chúng ta cũng sẽ tìm thấy rằng C# cũng đơn giản. Hầu hết mọi người đều không tin rằng Java là ngôn ngữ đơn giản. Tuy nhiên, C# thì dễ hơn là Java và C++.

C# là ngôn ngữ hiện đại

Điều gì làm cho một ngôn ngữ hiện đại? Những đặc tính như là xử lý ngoại lệ, thu gom bộ nhớ tự động, những kiểu dữ liệu mở rộng, và bảo mật mã nguồn là những đặc tính được mong đợi trong một ngôn ngữ hiện đại. C# chứa tất cả những đặc tính trên. Nếu là người mới học lập trình có thể chúng ta sẽ cảm thấy những đặc tính trên phức tạp và khó hiểu. Tuy nhiên, cũng đừng lo lắng chúng ta sẽ dần dần được tìm hiểu những đặc tính qua các chương trong cuốn sách này.

Ghi chú: Con trỏ được tích hợp vào ngôn ngữ C++. Chúng cũng là nguyên nhân gây ra những rắc rối của ngôn ngữ này. C# loại bỏ những phức tạp và rắc rối phát sinh bởi con trỏ. Trong C#, bộ thu gom bộ nhớ tự động và kiểu dữ liệu an toàn được tích hợp vào ngôn ngữ, sẽ loại bỏ những vấn đề rắc rối của C++.

C# là ngôn ngữ hướng đối tượng

Những đặc điểm chính của ngôn ngữ hướng đối tượng (Object-oriented language) là sự đóng gói (encapsulation), sự kế thừa (inheritance), và đa hình (polymorphism). C# hỗ trợ tất

cả những đặc tính trên. Phần hướng đối tượng của C# sẽ được trình bày chi tiết trong một chương riêng ở phần sau.

C# là ngôn ngữ mạnh mẽ và cũng mềm dẻo

Như đã đề cập trước, với ngôn ngữ C# chúng ta chỉ bị giới hạn ở chính bởi bản thân hay là trí tưởng tượng của chúng ta. Ngôn ngữ này không đặt những ràng buộc lên những việc có thể làm. C# được sử dụng cho nhiều các dự án khác nhau như là tạo ra ứng dụng xử lý văn bản, ứng dụng đồ họa, bản tính, hay thậm chí những trình biên dịch cho các ngôn ngữ khác.

C# là ngôn ngữ ít từ khóa

C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì sẽ mạnh hơn. Điều này không phải sự thật, ít nhất là trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào. Bảng sau liệt kê các từ khóa của ngôn ngữ C#.

abstract	default	foreach	object	sizeof	unsafe
as	delegate	goto	operator	stackalloc	ushort
base	do	if	out	static	using
bool	double	implicit	override	string	virtual
break	else	in	params	struct	volatile
byte	enum	int	private	switch	void
case	event	interface	protected	this	while
catch	explicit	internal	public	throw	
char	extern	is	readonly	true	
checked	false	lock	ref	try	
class	finally	long	return	typeof	
const	fixed	namespace	sbyte	uint	
continue	float	new	sealed	ulong	
decimal	for	null	short	unchecked	

Bảng 1.2: Từ khóa của ngôn ngữ C#.

C# là ngôn ngữ hướng module

Mã nguồn C# có thể được viết trong những phần được gọi là những lớp, những lớp này chứa các phương thức thành viên của nó. Những lớp và những phương thức có thể được sử dụng lại trong ứng dụng hay các chương trình khác. Bằng cách truyền các mẫu thông tin đến những lớp hay phương thức chúng ta có thể tạo ra những mã nguồn dùng lại có hiệu quả.

C# sẽ là một ngôn ngữ phổ biến

C# là một trong những ngôn ngữ lập trình mới nhất. Vào thời điểm cuốn sách này được viết, nó không được biết như là một ngôn ngữ phổ biến. Nhưng ngôn ngữ này có một số lý do để trở thành một ngôn ngữ phổ biến. Một trong những lý do chính là Microsoft và sự cam kết của .NET

Microsoft muốn ngôn ngữ C# trở nên phổ biến. Mặc dù một công ty không thể làm một sản phẩm trở nên phổ biến, nhưng nó có thể hỗ trợ. Cách đây không lâu, Microsoft đã gặp sự thất bại về hệ điều hành Microsoft Bob. Mặc dù Microsoft muốn Bob trở nên phổ biến nhưng thất bại. C# thay thế tốt hơn để đem đến thành công sơ với Bob. Thật sự là không biết khi nào mọi người trong công ty Microsoft sử dụng Bob trong công việc hằng ngày của họ. Tuy nhên, với C# thì khác, nó được sử dụng bởi Microsoft. Nhiều sản phẩm của công ty này đã chuyển đổi và viết lại bằng C#. Bằng cách sử dụng ngôn ngữ này Microsoft đã xác nhận khả năng của C# cần thiết cho những người lập trình.

Micorosoft .NET là một lý do khác để đem đến sự thành công của C#. .NET là một sự thay đổi trong cách tạo và thực thi những ứng dụng.

Ngoài hai lý do trên ngôn ngữ C# cũng sẽ trở nên phổ biến do những đặc tính của ngôn ngữ này được đề cập trong mục trước như: đơn giản, hướng đối tượng, mạnh mẽ...

Ngôn ngữ C# và những ngôn ngữ khác

Chúng ta đã từng nghe đến những ngôn ngữ khác như Visual Basic, C++ và Java. Có lẽ chúng ta cũng tự hỏi sự khác nhau giữa ngôn ngữ C# và nhưng ngôn ngữ đó. Và cũng tự hỏi tại sao lại chọn ngôn ngữ này để học mà không chọn một trong những ngôn ngữ kia. Có rất nhiều lý do và chúng ta hãy xem một số sự so sánh giữa ngôn ngữ C# với những ngôn ngữ khác giúp chúng ta phần nào trả lời được những thắc mắc.

Microsoft nói rằng C# mang đến sức mạnh của ngôn ngữ C++ với sự dễ dàng của ngôn ngữ Visual Basic. Có thể nó không dễ như Visual Basic, nhưng với phiên bản Visual Basic.NET (Version 7) thì ngang nhau. Bởi vì chúng được viết lại từ một nền tảng. Chúng ta có thể viết nhiều chương trình với ít mã nguồn hơn nếu dùng C#.

Mặc dù C# loại bỏ một vài các đặc tính của C++, nhưng bù lại nó tránh được những lỗi mà thường gặp trong ngôn ngữ C++. Điều này có thể tiết kiệm được hàng giờ hay thậm chí hàng ngày trong việc hoàn tất một chương trình. Chúng ta sẽ hiểu nhiều về điều này trong các chương của giáo trình.

Một điều quan trọng khác với C++ là mã nguồn C# không đòi hỏi phải có tập tin header. Tất cả mã nguồn được viết trong khai báo một lớp.

Như đã nói ở bên trên. .NET runtime trong C# thực hiện việc thu gom bộ nhớ tự động. Do điều này nên việc sử dụng con trỏ trong C# ít quan trọng hơn trong C++. Những con trỏ cũng có thể được sử dụng trong C#, khi đó những đoạn mã nguồn này sẽ được đánh dấu là không an toàn (unsafe code).

C# cũng từ bỏ ý tưởng đa kế thừa như trong C++. Và sự khác nhau khác là C# đưa thêm thuộc tính vào trong một lớp giống như trong Visual Basic. Và những thành viên của lớp được gọi duy nhất bằng toán tử "." khác với C++ có nhiều cách gọi trong các tình huống khác nhau.

Một ngôn ngữ khác rất mạnh và phổ biến là Java, giống như C++ và C# được phát triển dựa trên C. Nếu chúng ta quyết định sẽ học Java sau này, chúng ta sẽ tìm được nhiều cái mà học từ C# có thể được áp dụng.

Điểm giống nhau C# và Java là cả hai cùng biên dịch ra mã trung gian: C# biên dịch ra MSIL còn Java biên dịch ra bytecode. Sau đó chúng được thực hiện bằng cách thông dịch hoặc biên dịch just-in-time trong từng máy ảo tương ứng. Tuy nhiên, trong ngôn ngữ C# nhiều hỗ trợ được đưa ra để biên dịch mã ngôn ngữ trung gian sang mã máy. C# chứa nhiều kiểu dữ liệu cơ bản hơn Java và cũng cho phép nhiều sự mở rộng với kiểu dữ liệu giá trị. Ví dụ, ngôn ngữ C# hỗ trợ kiểu liệt kệ (enumerator), kiểu này được giới hạn đến một tập hằng được định nghĩa trước, và kiểu dữ liệu cấu trúc đây là kiểu dữ liệu giá trị do người dùng định nghĩa. Chúng ta sẽ được tìm hiểu kỹ hơn về kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị sẽ được trình bày trong phần sau

Tương tự như Java, C# cũng từ bỏ tính đa kế thừa trong một lớp, tuy nhiên mô hình kế thừa đơn này được mở rộng bởi tính đa kế thừa nhiều giao diện. ***Các bước chuẩn bị cho chương trình***

Thông thường, trong việc phát triển phần mềm, người phát triển phải tuân thủ theo quy trình phát triển phần mềm một cách nghiêm ngặt và quy trình này đã được chuẩn hóa. Tuy nhiên trong phạm vi của chúng ta là tìm hiểu một ngôn ngữ mới và viết những chương trình nhỏ thì không đòi hỏi khắt khe việc thực hiện theo quy trình. Nhưng để giải quyết được những vấn đề thì chúng ta cũng cần phải thực hiện đúng theo các bước sau. Đầu tiên là phải xác định vấn đề cần giải quyết. Nếu không biết rõ vấn đề thì ta không thể tìm được phương pháp giải quyết. Sau khi xác định được vấn đề, thì chúng ta có thể nghĩ ra các kế hoạch để thực hiện. Sau khi có một kế hoạch, thì có thể thực thi kế hoạch này. Sau khi kế hoạch được thực thi, chúng ta phải kiểm tra lại kết quả để xem vấn đề được giải quyết xong chưa. Logic này thường được áp dụng trong nhiều lĩnh vực khác nhau, trong đó có lập trình.

Khi tạo một chương trình trong C# hay bất cứ ngôn ngữ nào, chúng ta nên theo những bước tuần tự sau:

Xác định mục tiêu của chương trình.

Xác định những phương pháp giải quyết vấn đề.

Tạo một chương trình để giải quyết vấn đề.

Thực thi chương trình để xem kết quả.

Ví dụ mục tiêu để viết chương trình xử lý văn bản đơn giản, mục tiêu chính là xây dựng chương trình cho phép soạn thảo và lưu trữ những chuỗi ký tự hay văn bản. Nếu không có mục tiêu thì không thể viết được chương trình hiệu quả.

Bước thứ hai là quyết định đến phương pháp để viết chương trình. Bước này xác định những thông tin nào cần thiết được sử dụng trong chương trình, các hình thức nào được sử dụng. Từ những thông tin này chúng ta rút ra được phương pháp để giải quyết vấn đề.

Bước thứ ba là bước cài đặt, ở bước này có thể dùng các ngôn ngữ khác nhau để cài đặt, tuy nhiên, ngôn ngữ phù hợp để giải quyết vấn đề một cách tốt nhất sẽ được chọn. Trong phạm vi của sách này chúng ta mặc định là dùng C#, đơn giản là chúng ta đang tìm hiểu nó! Và bước cuối cùng là phần thực thi chương trình để xem kết quả.

Chương trình C# đơn giản

Để bắt đầu cho việc tìm hiểu ngôn ngữ C# và tạo tiền đề cho các chương sau, chương đầu tiên trình bày một chương trình C# đơn giản nhất.

Ví dụ 2.1 : Chương trình C# đầu tiên.

```
---------------------------------------------------------------------------
class ChaoMung
{
    static void Main( )
    {
        // Xuat ra man hinh
        System.Console.WriteLine("Chao Mung");
    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Chao Mung
---------------------------------------------------------------------------
```

Sau khi viết xong chúng ta lưu dưới dạng tập tin có phần mở rộng *.cs (C sharp). Sau đó biên dịch và chạy chương trình. Kết quả là một chuỗi "`Chao Mung`" sẽ xuất hiện trong màn hình console.

Các mục sau sẽ giới thiệu xoay quanh ví dụ 2.1, còn phần chi tiết từng loại sẽ được trình bày trong các chương kế tiếp.

Lớp, đối tượng và kiểu dữ liệu (type)

Điều cốt lõi của lập trình hướng đối tượng là tạo ra các kiểu mới. Kiểu là một thứ được xem như trừu tượng. Nó có thể là một bảng dữ liệu, một tiểu trình, hay một nút lệnh trong một cửa sổ. Tóm lại kiểu được định nghĩa như một dạng vừa có thuộc tính chung (properties) và các hành vi ứng xử (behavior) của nó.

Nếu trong một ứng dụng trên Windows chúng ta tạo ra ba nút lệnh OK, Cancel, Help, thì thực chất là chúng ta đang dùng ba thể hiện của một kiểu nút lệnh trong Windows và các nút này cùng chia xẻ các thuộc tính và hành vi chung với nhau. Ví dụ, các nút có các thuộc tính như kích thước, vị trí, nhãn tên (label), tuy nhiên mỗi thuộc tính của một thể hiện không nhất thiết phải giống nhau, và thường thì chúng khác nhau, như nút OK có nhãn là "OK", Cancel có nhãn là "Cancel"...Ngoài ra các nút này có các hành vi ứng xử chung như khả năng vẽ, kích hoạt, đáp ứng các thông điệp nhấn,…Tùy theo từng chức năng đặc biệt riêng của từng loại thì nội dung ứng xử khác nhau, nhưng tất cả chúng được xem như là cùng một kiểu.

Cũng như nhiều ngôn ngữ lập trình hướng đối tượng khác, kiểu trong C# được định nghĩa là một lớp (class), và các thể hiện riêng của từng lớp được gọi là đối tượng (object). Trong các chương kế tiếp sẽ trình bày các kiểu khác nhau ngoài kiểu lớp như kiểu liệt kê, cấu trúc và kiểu ủy quyền (delegates).

Quay lại chương trình ChaoMung trên, chương trình này chỉ có một kiểu đơn giản là lớp ChaoMung. Để định nghĩa một kiểu lớp trong C# chúng ta phải dùng từ khoá class, tiếp sau là tên lớp trong ví dụ trên tên lớp là ChaoMung. Sau đó định nghĩa các thuộc tính và hành động cho lớp. Thuộc tính và hành động phải nằm trong dấu { }.

Ghi chú: Khai báo lớp trong C# không có dấu ; sau ngoặc } cuối cùng của lớp. Và khác với lớp trong C/C++ là chia thành 2 phần header và phần định nghĩa. Trong C# , định nghĩa một lớp được gói gọn trong dấu { } sau tên lớp và trong cùng một tập tin.

Phương thức

Hai thành phần chính cấu thành một lớp là thuộc tính hay tính chất và phương thức hay còn gọi là hành động ứng xử của đối tượng. Trong C# hành vi được định nghĩa như một phương thức thành viên của lớp.

Phương thức chính là các hàm được định nghĩa trong lớp. Do đó, ta còn có thể gọi các phương thức thành viên là các hàm thành viên trong một lớp. Các phương thức này chỉ ra rằng các hành động mà lớp có thể làm được cùng với cách thức làm hành động đó. Thông thường, tên của phương thức thường được đặt theo tên hành động, ví dụ như DrawLine() hay GetString().

Tuy nhiên trong ví dụ 2.1 vừa trình bày, chúng ta có hàm thành viên là Main() hàm này là hàm đặc biệt, không mô tả hành động nào của lớp hết, nó được xác định là hàm đầu vào của lớp (entry point) và được CRL gọi đầu tiên khi thực thi.

Ghi chú: Trong C#, hàm Main() được viết ký tự hoa đầu, và có thể trả về giá trị void hay int

Khi chương trình thực thi, CLR gọi hàm Main() đầu tiên, hàm Main() là đầu vào của chương trình, và mỗi chương trình phải có một hàm Main(). Đôi khi chương trình có nhiều hàm Main() nhưng lúc này ta phải xác định các chỉ dẫn biên dịch để CLR biết đâu là hàm Main() đầu vào duy nhất trong chương trình.

Việc khai báo phương thức được xem như là một sự giao ước giữa người tạo ra lớp và người sử dụng lớp này. Người xây dựng các lớp cũng có thể là người dùng lớp đó, nhưng không hoàn toàn như vậy. Vì có thể các lớp này được xây dựng thành các thư viện chuẩn và cung cấp cho các nhóm phát triển khác…Do vậy việc tuân thủ theo các qui tắc lá rất cần thiết.

Để khai báo một phương thức, phải xác định kiểu giá trị trả về, tên của phương thức, và cuối cùng là các tham số cần thiết cho phương thức thực hiện.

Chú thích

Một chương trình được viết tốt thì cần phải có chú thích các đoạn mã được viết. Các đoạn chú thích này sẽ không được biên dịch và cũng không tham gia vào chương trình. Mục đích chính là làm cho đoạn mã nguồn rõ ràng và dễ hiểu.

Trong ví dụ 2.1 có một dòng chú thích :

```
// Xuat ra man hinh.
```

Một chuỗi chú thích trên một dòng thì bắt đầu bằng ký tự "//". Khi trình biên dịch gặp hai ký tự này thì sẽ bỏ qua dòng đó.

Ngoài ra C# còn cho phép kiểu chú thích cho một hay nhiều dòng, và ta phải khai báo "/*" ở phần đầu chú thích và kết thúc chú thích là ký tự "*/".

Ví dụ 2.2 : Minh họa dùng chú thích trên nhiều dòng.

```
-------------------------------------------------------------------------
class ChaoMung
{

        static void Main()
        {
            /* Xuat ra man hinh chuoi 'chao mung'
             Su dung ham WriteLine cua lop System.Console
             */
             System.Console.WriteLine("Chao Mung");
        }
}
-------------------------------------------------------------------------
```

Kết quả:

```
Chao Mung
-------------------------------------------------------------------------
```

Ngoài hai kiểu chú thích trên giống trong C/C++ thì C# còn hỗ trợ thêm kiểu thứ ba cũng là kiểu cuối cùng, kiểu này chứa các định dạng XML nhằm xuất ra tập tin XML khi biên dịch để tạo sưu liệu cho mã nguồn. Chúng ta sẽ bàn kiểu này trong các chương trình ở các phần tiếp.

Ứng dụng Console

Ví dụ đơn giản trên được gọi là ứng dụng console, ứng dụng này giao tiếp với người dùng thông quan bàn phím và không có giao diện người dùng (UI), giống như các ứng dụng thường thấy trong Windows. Trong các chương xây dựng các ứng dụng nâng cao trên Windows hay Web thì ta mới dùng các các giao diện đồ họa. Còn để tìm hiểu về ngôn ngữ C# thuần tuý thì cách tốt nhất là ta viết các ứng dụng console.

Trong hai ứng dụng đơn giản trên ta đã dùng phương thức WriteLine() của lớp Console. Phương thức này sẽ xuất ra màn hình dòng lệnh hay màn hình DOS chuỗi tham số đưa vào, cụ thể là chuỗi "`Chao Mung`".

Namespace

Như chúng ta đã biết .NET cung cấp một thư viện các lớp đồ sộ và thư viện này có tên là FCL (Framework Class Library). Trong đó Console chỉ là một lớp nhỏ trong hàng ngàn lớp trong thư viện. Mỗi lớp có một tên riêng, vì vậy FCL có hàng ngàn tên như ArrayList, Dictionary, FileSelector,…

Điều này làm nảy sinh vấn đề, người lập trình không thể nào nhớ hết được tên của các lớp trong .NET Framework. Tệ hơn nữa là sau này có thể ta tạo lại một lớp trùng với lớp đã có chẳng hạn. Ví dụ trong quá trình phát triển một ứng dụng ta cần xây dựng một lớp từ điển và lấy tên là Dictionary, và điều này dẫn đến sự tranh chấp khi biên dịch vì C# chỉ cho phép một tên duy nhất.

Chắc chắn rằng khi đó chúng ta phải đổi tên của lớp từ điển mà ta vừa tạo thành một cái tên khác chẳng hạn như myDictionary. Khi đó sẽ làm cho việc phát triển các ứng dụng trở nên phức tạp, cồng kềnh. Đến một sự phát triển nhất định nào đó thì chính là cơn ác mộng cho nhà phát triển.

Giải pháp để giải quyết vấn đề này là việc tạo ra một namespace**,** namsespace sẽ hạn chế phạm vi của một tên, làm cho tên này chỉ có ý nghĩa trong vùng đã định nghĩa.

Giả sử có một người nói Tùng là một kỹ sư, từ kỹ sư phải đi kèm với một lĩnh vực nhất định nào đó, vì nếu không thì chúng ta sẽ không biết được là anh ta là kỹ sư cầu đường, cơ khí hay phần mềm. Khi đó một lập trình viên C# sẽ bảo rằng Tùng là CauDuong.KySu phân biệt với CoKhi.KySu hay PhanMem.KySu. Namespace trong trường hợp này là CauDuong, CoKhi, PhanMem sẽ hạn chế phạm vi của những từ theo sau. Nó tạo ra một vùng không gian để tên sau đó có nghĩa.

Tương tự như vậy ta cứ tạo các namespace để phân thành các vùng cho các lớp trùng tên không tranh chấp với nhau.

Tương tự như vậy, .NET Framework có xây dựng một lớp Dictionary bên trong namespace System.Collections, và tương ứng ta có thể tạo một lớp Dictionary khác nằm trong namespace ProgramCSharp.DataStructures, điều này hoàn toàn không dẫn đến sự tranh chấp với nhau.

Trong ví dụ minh họa 1.2 đối tượng Console bị hạn chế bởi namespace bằng việc sử dụng mã lệnh:

```
System.Console.WriteLine();
```

Toán tử '.'

Trong vídụ 2.2 trên dấu '.' được sử dụng để truy cập đến phương thức hay dữ liệu trong một lớp (trong trường hợp này phương thức là WriteLine()), và ngăn cách giữa tên lớp đến một namespace xác nhận (namspace System và lớp là Console). Việc thực hiện này theo hướng từ trên xuống, trong đó mức đầu tiên namespace là System, tiếp theo là lớp Console, và cuối cùng là truy cập đến các phương thức hay thuộc tính của lớp.

Trong nhiều trường hợp namespace có thể được chia thành các namespace con gọi là subnamespace. Ví dụ trong namespace System có chứa một số các subnamesapce như Configuration, Collections, Data, và còn rất nhiều nữa, hơn nữa trong namespace Collection còn chia thành nhiều namesapce con nữa.

Namespace giúp chúng ta tổ chức và ngăn cách những kiểu. Khi chúng ta viết một chương trình C# phức tạp, chúng ta có thể phải tạo một kiến trúc namespace riêng cho mình, và không giới hạn chiều sâu của cây phân cấp namespace. Mục đích của namespace là giúp chúng ta chia để quản lý những kiến trúc đối tượng phức tạp.

Từ khóa using

Để làm cho chương trình gọn hơn, và không cần phải viết từng namespace cho từng đối tượng, C# cung cấp từ khóa là using, sau từ khóa này là một namespace hay subnamespace với mô tả đầy đủ trong cấu trúc phân cấp của nó.

Ta có thể dùng dòng lệnh :

```
using System;
```

ở đầu chương trình và khi đó trong chương trình nếu chúng ta có dùng đối tượng Console thì không cần phải viết đầy đủ : System.Console. mà chỉ cần viết Console. thôi.

Ví dụ 2.3: Dùng khóa using

```
---------------------------------------------------------------------------
using System;
class ChaoMung
{
    static void Main()
    {
        //Xuat ra man hinh chuoi thong bao
        Console.WriteLine("Chao Mung");
    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Chao Mung
```

Lưu ý rằng phải đặt câu using System trước định nghĩa lớp ChaoMung.
Mặc dù chúng ta chỉ định rằng chúng ta sử dụng namespace System, và không giống như các ngôn ngữ khác, không thể chỉ định rằng chúng ta sử dụng đối tượng System.Console.

Ví dụ 2.4: Không hợp lệ trong C#.

```
using System.Console;
class ChaoMung
{
    static void Main()
    {
        //Xuat ra man hinh chuoi thong bao
        WriteLine("Chao Mung");
    }
}
```

Đoạn chương trình trên khi biên dịch sẽ được thông báo một lỗi như sau:
error CS0138: A using namespace directive can only be applied to namespace;
'System.Console' is a class not a namespace.
Cách biểu diễn namespace có thể làm giảm nhiều thao tác gõ bàn phím, nhưng nó có thể sẽ không đem lại lợi ích nào bởi vì nó có thể làm xáo trộn những namespace có tên không khác nhau. Giải pháp chung là chúng ta sử dụng từ khóa **using** với các namespace đã được xây dựng sẵn, các namespace do chúng ta tạo ra, những namespace này chúng ta đã nắm chắc sưu liệu về nó. Còn đối với namespace do các hãng thứ ba cung cấp thì chúng ta không nên dùng từ khóa **using**.

Phân biệt chữ thường và chữ hoa

Cũng giống như C/C++, C# là ngôn ngữ phân biệt chữ thường với chữ hoa, điều này có nghĩa rằng hai câu lệnh writeLine thì khác với WriteLine và cũng khác với WRITELINE.
Đáng tiếc là C# không giống như VB, môi trường phát triển C# sẽ không tự sửa các lỗi này, nếu chúng ta viết hai chữ với cách khác nhau thì chúng ta có thể đưa vào chương trình gỡ rối tìm ra các lỗi này.

Để trách việc lãnh phí thời gian và công sức, người ta phát triển một số qui ước cho cách đặt tên biến, hằng, hàm, và nhiều định danh khác nữa. Qui ước trong giáo trình này dùng cú pháp lạc đà (camel notation) cho tên biến và cú pháp Pascal cho hàm, hằng, và thuộc tính.
Ví dụ :

Biến myDictionary theo cách đặt tên cú pháp lạc đà.

Hàm DrawLine, thuộc tính ColorBackground theo cách đặt tên cú pháp Pascal.

Từ khóa static

Hàm Main() trong ví dụ minh họa trên có nhiều hơn một cách thiết kế. Trong minh họa này hàm Main() được khai báo với kiểu trả về là void, tức là hàm này không trả về bất cứ giá trị nào cả. Đôi khi cần kiểm tra chương trình có thực hiện đúng hay không, người lập trình có thể khai báo hàm Main() trả về một giá trị nào đó để xác định kết quả thực hiện của chương trình. Trong khai báo của ví dụ trên có dùng từ khóa **static**:

```
static void Main()
{
    .....
}
```

Từ khóa này chỉ ra rằng hàm Main() có thể được gọi mà không cần phải tạo đối tượng ChaoMung. Những vấn đề liên quan đến khai báo lớp, phương thức, hay thuộc tính sẽ được trình bày chi tiết trong các chương tiếp theo.

Phát triển chương trình minh họa

Có tối thiểu là hai cách để soạn thảo, biên dịch và thực thi chương trình trong cuốn sách này:

Sử dụng môi trường phát triển tích hợp (IDE) Visual Studio .NET

Sử dụng chương trình soạn thảo văn bản bất kỳ như Notepad rồi dùng biên dịch dòng lệnh.

Mặc dù chúng ta có thể phát triển phần mềm bên ngoài Visual Studio .NET, IDE cung cấp nhiều các tiện ích hỗ trợ cho người phát triển như: hỗ trợ phần soạn thảo mã nguồn như canh lề, màu sắc, tích hợp các tập tin trợ giúp, các đặc tính intellisense,...Nhưng điều quan trọng nhất là IDE phải có công cụ debug mạnh và một số công cụ trợ giúp phát triển ứng dụng khác.

Trong cuốn sách này giả sử rằng người đọc đang sử dụng Visual Studio .NET. Phần trình này sẽ tập trung vào ngôn ngữ và platform hơn là công cụ phát triển. Chúng ta có thể sao chép tất cả những mã nguồn ví dụ vào trong một chương trình soạn thảo văn bản như Notepad hay Emacs, lưu chúng dưới dạng tập tin văn bản, và biên dịch chúng bằng trình biên dịch dòng lệnh C#, chương trình này được phân phối cùng .NET Framework SDK. Trong những chương cuối về xây dựng các ứng dụng trên Windows và Web, chúng ta sẽ sử dụng công cụ Visual Studio .NET để tạo ra các Windows Form và Web Form, tuy nhiên chúng ta cũng có thể viết bằng tay trong Notepad nếu chúng ta quyết định sử dụng cách làm bằng tay thay vì dùng công cụ thiết kế.

Sử dụng Notepad soạn thảo

Đầu tiên chúng ta sẽ mở chương trình Notepad rồi soạn thảo chương trình minh họa trên, lưu ý là ta có thể sử dụng bất cứ trình soạn thảo văn bản nào chứ không nhất thiết là Notepad. Sau khi soạn thảo xong thì lưu tập tin xuống đĩa và tập tin này có phần mở rộng là *.cs, trong ví dụ này là chaomung.cs. Bước tiếp theo là biên dịch tập tin nguồn vừa tạo ra. Để biên dịch ta dùng trình biên dịch dòng lệnh C# (csc.exe) chương trình này được chép vào máy trong quá trình cài .NET Framework. Để biết csc.exe nằm chính xác vị trí nào trong đĩa ta có thể dùng chức năng tìm kiếm của Windows.

Để thực hiện biên dịch chúng ta mở một cửa sổ dòng lệnh rồi đánh vào lệnh theo mẫu sau:

```
csc.exe [/out: <file thực thi>]  <file nguồn>
```

Ví dụ: `csc.exe /out:d:\chaomung.exe  d:\chaomung.cs`

Thường thì khi biên dịch ta chỉ cần hai phần là tên của trình biên dịch và tên tập tin nguồn mà thôi. Trong mẫu trên có dùng một trong nhiều tùy chọn khi biên dịch là /out, theo sau là tên của chương trình thực thi hay chính là kết quả biên dịch tập tin nguồn.

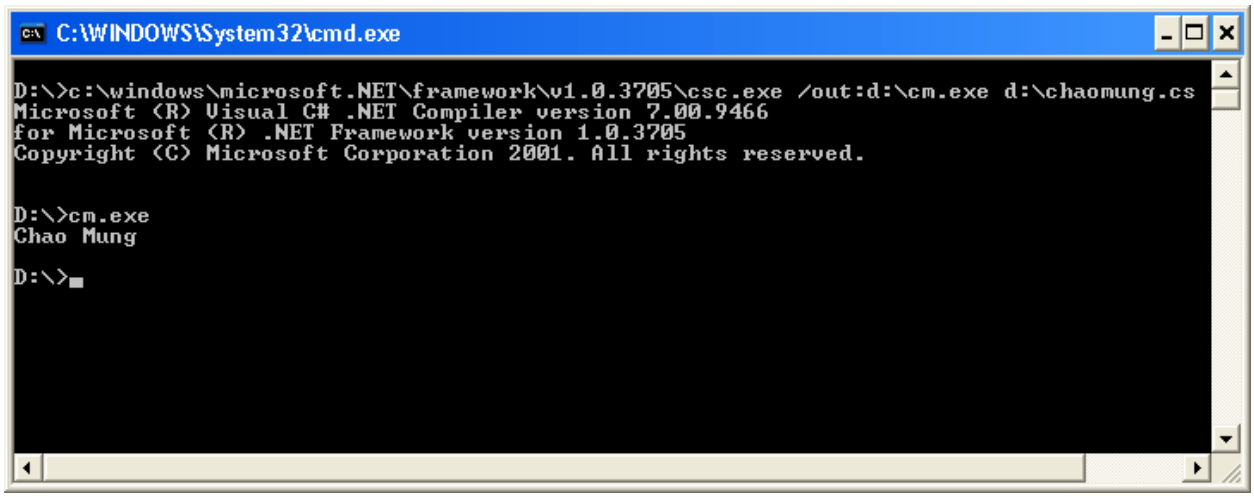
Các tham số tùy chọn có rất nhiều nếu muốn tìm hiểu chúng ta có thể dùng lệnh:

```
csc.exe /?
```

Lệnh này xuất ra màn hình toàn bộ các tùy chọn biên dịch và các hướng dẫn sử dụng.

Hai hình sau minh họa quá trình nhập mã nguồn chương trình C# bằng một trình soạn thảo văn bản đơn giản như Notepad trong Windows. Và sau đó biên dịch tập tin mã nguồn vừa tạo ra bằng chương trình csc.exe một trình biên dịch dòng lệnh của C#. Kết quả là một tập tin thực thi được tạo ra và ta sẽ chạy chương trình này.

Hình 2.2: Mã nguồn được soạn thảo trong Notepad.



Hình 2.3: Biên dịch và thực thi chương trình.

Sử dụng Visual Studio .NET để tạo chương trình

Để tạo chương trình chào mừng trong IDE, lựa chọn mục Visual Studio .NET trong menu Start hoặc icon của nó trên desktop, sau khi khởi động xong chương trình, chọn tiếp chức năng File New Project trong menu. Chức năng này sẽ gọi cửa sổ New Project (hình 2.4 bên dưới). Nếu như chương trình Visual Studio .NET được chạy lần đầu tiên, khi đó cửa sổ New Project sẽ xuất hiện tự động mà không cần phải kích hoạt.

Để tạo ứng dụng, ta lựa chọn mục Visual C# Projects trong cửa sổ Project Type bên trái. Lúc này chúng ta có thể nhập tên cho ứng dụng và lựa chọn thư mục nơi lưu trữ các tập tin này. Cuối cùng, kích vào OK khi mọi chuyện khởi tạo đã chấm dứt và một cửa sổ mới sẽ xuất hiện (hình 2.4 bên dưới), chúng ta có thể nhập mã nguồn vào đây.

Lưu ý rằng Visual Studio .NET tạo ra một namespace dựa trên tên của project mà ta vừa cung cấp (ChaoMung), và thêm vào chỉ dẫn sử dụng namespace System bằng lệnh **using**, bởi hầu như mọi chương trình mà chúng ta viết đều cần sử dụng các kiểu dữ liệu chứa trong namespace System.

Hình 2.4: Tạo ứng dụng C# console trong Visual Studio .NET.

Hình 2.5: Phần soạn thảo mã nguồn cho project.

Visual Studio .NET tạo một lớp tên là Class1, lớp này chúng ta có thể tùy ý đổi tên của chúng. Khi đổi tên của lớp, tốt nhất là đổi tên luôn tập tin chứa lớp đó (Class1.cs). Giả sử

trong ví dụ trên chúng ta đổi tên của lớp thành ChaoMung, và đổi tên tập tin Class1.cs (đổi tên tập tin trong cửa sổ Solution Explorer).

Cuối cùng, Visual Studio .NET tạo một khung sườn chương trình, và kết thúc với chú thích TODO là vị trí bắt đầu của chúng ta. Để tạo chương trình chào mừng trong minh họa trên, ta bỏ tham số string[] args của hàm Main() và xóa tất cả các chú thích bên trong của hàm. Sau đó nhập vào dòng lệnh sau bên trong thân của hàm Main()

```
// Xuat ra man hinh
System.Console.WriteLine("Chao Mung");
```

Sau tất cả công việc đó, tiếp theo là phần biên dịch chương trình từ Visual Studio .NET. Thông thường để thực hiện một công việc nào đó ta có thể chọn kích hoạt chức năng trong menu, hay các button trên thanh toolbar, và cách nhanh nhất là sử dụng các phím nóng hay các phím kết hợp để gọi nhanh một chức năng.

Trong ví dụ, để biên dịch chương trình nhấn *Ctrl-Shift-B* hoặc chọn chức năng:

Build Build Solution. Một cách khác nữa là dùng nút lệnh trên thanh toolbar: 

Để chạy chương trình vừa được tạo ra mà không sử dụng chế độ debug chúng ta có thể nhấn *Ctrl-F5* hay chọn Debug Start Without Debugging hoặc nút lệnh  trên thanh toolbar của Visual Studio . NET

Ghi chú: Tốt hơn hết là chúng ta nên bỏ ra nhiều thời gian để tìm hiểu hay khám phá môi trường phát triển Visual Studio .NET. Đây cũng là cách thức tốt mà những người phát triển ứng dụng và chúng ta nên thực hiện. Việc tìm hiểu Visual Studio .NET và thông thạo nó sẽ giúp cho chúng ta rất nhiều trong quá trình xây dựng và phát triển ứng dụng sau này.

Câu hỏi và trả lời

Câu hỏi 1: Một chương trình C# có thể chạy trên bất cứ máy nào?

Trả lời 1: Không phải tất cả. Một chương trình C# chỉ chạy trên máy có Common Language Runtime (CLR) được cài đặt. Nếu chúng ta copy một chương trình exe của C# qua một máy không có CLR thì chúng ta sẽ nhận được một lỗi. Trong những phiên bản của Windows không có CLR chúng ta sẽ được bảo rằng thiếu tập tin DLL.

Câu hỏi 2: Nếu muốn đưa chương trình mà ta viết cho một người bạn thì tập tin nào mà chúng ta cần đưa?

*Trả lời 2: Thông thường cách tốt nhất là đưa chương trình đã biên dịch. Điều này có nghĩa rằng sau khi mã nguồn được biên dịch, chúng ta sẽ có một chương trình thực thi (tập tin có phần mở rộng *.exe). Như vậy, nếu chúng ta muốn đưa chương trình Chaomung cho tất cả những người bạn của chúng ta thì chỉ cần đưa tập tin Chaomung.exe. Không cần thiết phải đưa tập tin nguồn Chaomung.cs. Và những người bạn của chúng ta không cần thiết phải có trình biên dịch C#. Họ chỉ cần có C# runtime trên máy tính (như CLR của Microsoft) là có thể chạy được chương trình của chúng ta.*

Câu hỏi 3: Sau khi tạo ra được tập tin thực thi .exe. Có cần thiết giữ lại tập tin nguồn không?

Trả lời 3: Nếu chúng ta từ bỏ tập tin mã nguồn thì sau này sẽ rất khó khăn cho việc mở rộng hay thay đổi chương trình, do đó cần thiết phải giữ lại các tập tin nguồn. Hầu hết các IDE tạo ra các các tập tin nguồn (.cs) và các tập tin thực thi. Cũng như giữ các tập tin nguồn chúng ta cũng cần thiết phải giữ các tập tin khác như là các tài nguyên bên ngoài các icon, image, form.. Chúng ta sẽ lưu giữ những tập tin này trong trường hợp chúng ta cần thay đổi hay tạo lại tập tin thực thi.

Câu hỏi 4: Nếu trình biên dịch C# đưa ra một trình soạn thảo, có phải nhất thiết phải sử dụng nó?

Trả lời 4: Không hoàn toàn như vậy. Chúng ta có thể sử dụng bất cứ trình soạn thảo văn bản nào và lưu mã nguồn dưới dạng tập tin văn bản. Nếu trình biên dịch đưa ra một trình soạn thảo thì chúng ta nên sử dụng nó. Nếu chúng ta có môt trình soạn thảo khác tốt hơn chúng ta có thể sử dụng nó. Một số các tiện ích soạn thảo mã nguồn có thể giúp cho ta dễ dàng tìm các lỗi cú pháp, giúp tạo một số mã nguồn tự động đơn giản...Nói chung là tùy theo chúng ta nhưng theo tôi thì Visual Studio .NET cũng khá tốt để sử dụng

Câu hỏi 5: Có thể không quan tâm đến những cảnh báo khi biên dịch mã nguồn

Trả lời 5: Một vài cảnh báo không ảnh hưởng đến chương trình khi chạy, nhưng một số khác có thể ảnh hưởng đến chương trình chạy. Nếu trình biên dịch đưa ra cảnh báo, tức là tín hiệu cho một thứ gì đó không đúng. Hầu hết các trình biên dịch cho phép chúng ta thiết lập mức độ cảnh báo. Bằng cách thiết lập mức độ cảnh báo chúng ta có thể chỉ quan tâm đến những cảnh báo nguy hiểm, hay nhận hết tất cả những cảnh báo. Nói chung cách tốt nhất là chúng ta nên xem tất cả những cảnh báo để sửa chữa chúng, một chương trình tạm gọi là đạt yêu cầu khi không có lỗi biên dịch và cũng không có cảnh báo (nhưng chưa chắc đã chạy đúng kết quả!).

Câu hỏi thêm

Câu hỏi 1: Hãy đưa ra 3 lý do tại sao ngôn ngữ C# là một ngôn ngữ lập trình tốt?

Câu hỏi 2: IL và CLR viết tắt cho từ nào và ý nghĩa của nó?

Câu hỏi 3: Đưa ra các bước cơ bản trong chu trình xây dựng chương trình?

Câu hỏi 4: Trong biên dịch dòng lệnh thì lệnh nào được sử dụng để biên dịch mã nguồn .cs và lệnh này gọi chương trình nào?

Câu hỏi 5: Phần mở rộng nào mà chúng ta nên sử dụng cho tập tin mã nguồn C#?

Câu hỏi 6: Một tập tin .txt chứa mã nguồn C# có phải là một tập tin mã nguồn C# hợp lệ hay không? Có thể biên dịch được hay không?

Câu hỏi 7: Ngôn ngữ máy là gì? Khi biên dịch mã nguồn C# ra tập tin .exe thì tập tin này là ngôn ngữ gì?

Câu hỏi 8: Nếu thực thi một chương trình đã biên dịch và nó không thực hiện đúng như mong đợi của chúng ta, thì điều gì chúng ta cần phải làm?

Câu hỏi 9: Một lỗi tương tự như bên dưới thường xuất hiện khi nào?

mycode.cs(15,5): error CS1010: NewLine in constan

Câu hỏi 10: Tại sao phải khai báo static cho hàm Main của lớp?

Câu hỏi 11: Một mã nguồn C# có phải chứa trong các lớp hay là có thể tồn tại bên ngoài lớp như C/C++?

Câu hỏi 12: So sánh sự khác nhau cơ bản giữa C# và C/C++, C# với Java, hay bất cứ ngôn ngữ cấp cao nào mà bạn đã biết?

Câu hỏi 13: Con trỏ có còn được sử dụng trong C# hay không? Nếu có thì nó được quản lý như thế nào?

Câu hỏi 14: Khái niệm và ý nghĩa của namespace trong C#? Điều gì xảy ra nếu như ngôn ngữ lập trình không hỗ trợ namespace?

Bài tập

Bài tập 1: Dùng trình soạn thảo văn bản mở chương trình exe mà ta đã biên dịch từ các chương trình nguồn trước và xem sự khác nhau giữa hai tập tin này, lưu ý sao khi đóng tập tin này ta không chọn lưu tập tin.

Bài tập 2: Nhập vào chương trình sau và biên dịch nó. Cho biết chương trình thực hiện điều gì?

```
---------------------------------------------------------------------------
using System;
class variables
{
    public static void Main()
    {
        int radius = 4;
        const double PI = 3.14159;
        double circum, area;
        area = PI * radius* radius;
        circum = 2 * PI * radius;
        // in kết quả
        Console.WriteLine("Ban kinh = {0}, PI = {1}", radius, PI);
        Console.WriteLine("Dien tich {0}", area);
        Console.WriteLine("Chu vi {0}", circum);
    }
}
---------------------------------------------------------------------------
```

Bài tập 3: Nhập vào chương trình sau và biên dịch. Cho biết chương trình thực hiện điều gì?

```
---------------------------------------------------------------------------
class AClass
```

```
{
    static void Main()
    {
        int x, y;
        for( x = 0; x < 10; x++, System.Console.Write("\n"));
        for( y = 0 ; y < 10; y++, System.Console.WriteLine("{0}",y));
    }
}
```

Bài tập 4: Chương trình sau có chứa lỗi. Nhập vào và sửa những lỗi đó

Bài tập 5: Sửa lỗi và biên dịch chương trình sau

```
class Test
{
    pubic static void Main()
    {

        Console.WriteLine("Xin chao");
        Consoile.WriteLine("Tam biet");

    }
}
```

Bài tập 6: Sửa lỗi và biên dịch chương trình sau

```
class Test
{
    pubic void Main()
    {

        Console.WriteLine('Xin chao');
        Consoile.WriteLine('Tam biet');

    }
}
```

Bài tập 7: Viết chương trình xuất ra bài thơ:

Rằm Tháng Giêng

Rằm xuân lồng lộng trăng soi,
Sông xuân nước lẫn màu trời thêm xuân.
Giữa dòng bàn bạc việc quân
Khuya về bát ngát trăng ngân đầy thuyền.

Hồ Chí Minh.

Chương 3

NỀN TẢNG NGÔN NGỮ C#

- **Kiểu dữ liệu**
    - **Kiểu dữ liệu xây dựng sẵn**
    - **Chọn kiểu dữ liệu**
    - **Chuyển đổi các kiểu dữ liệu**
- **Biến và hằng**
    - **Gán giá trị xác định cho biến**
    - **Hằng**
    - **Kiểu liệt kê**
    - **Kiểu chuỗi ký tự**
    - **Định danh**
- **Biểu thức**
- **Khoảng trắng**
- **Câu lệnh**
    - **Phân nhánh không có điều kiện**
    - **Phân nhánh có điều kiện**
    - **Câu lệnh lặp**
- **Toán tử**
- **Namespace**
- **Các chỉ dẫn biên dịch**
- **Câu hỏi & bài tập**

Trong chương trước chúng ta đã tìm hiểu một chương trình C# đơn giản nhất. Chương trình đó chưa đủ để diễn tả một chương trình viết bằng ngôn ngữ C#, có quá nhiều phần và chi tiết đã bỏ qua. Do vậy trong chương này chúng ta sẽ đi sâu vào tìm hiểu cấu trúc và cú pháp của ngôn ngữ C#.

Chương này sẽ thảo luận về hệ thống kiểu dữ liệu, phân biệt giữa kiểu dữ liệu xây dựng sẵn (như int, bool, string…) với kiểu dữ liệu do người dùng định nghĩa (lớp hay cấu trúc do người lập trình tạo ra...). Một số cơ bản khác về lập trình như tạo và sử dụng biến dữ liệu hay hằng cũng được đề cập cùng với cấu trúc liệt kê, chuỗi, định danh, biểu thức và cậu lệnh.

Trong phần hai của chương hướng dẫn và minh họa việc sử dụng lệnh phân nhánh **if, switch**, **while**, **do**...**while**, **for**, và **foreach**. Và các toán tử như phép gán, phép toán logic, phép toán quan hệ, và toán học...

Như chúng ta đã biết C# là một ngôn ngữ hướng đối tượng rất mạnh, và công việc của người lập trình là kế thừa để tạo và khai thác các đối tượng. Do vậy để nắm vững và phát triển tốt người lập trình cần phải đi từ những bước đi dầu tiên tức là đi vào tìm hiểu những phần cơ bản và cốt lõi nhất của ngôn ngữ.

Kiểu dữ liệu

C# là ngôn ngữ lập trình mạnh về kiểu dữ liệu, một ngôn ngữ mạnh về kiểu dữ liệu là phải khai báo kiểu của mỗi đối tượng khi tạo (kiểu số nguyên, số thực, kiểu chuỗi, kiểu điều khiển...) và trình biên dịch sẽ giúp cho người lập trình không bị lỗi khi chỉ cho phép một loại kiểu dữ liệu có thể được gán cho các kiểu dữ liệu khác. Kiểu dữ liệu của một đối tượng là một tín hiệu để trình biên dịch nhận biết kích thước của một đối tượng (kiểu int có kích thước là 4 byte) và khả năng của nó (như một đối tượng button có thể vẽ, phản ứng khi nhấn,...).

Tương tự như C++ hay Java, C# chia thành hai tập hợp kiểu dữ liệu chính: Kiểu xây dựng sẵn (built- in) mà ngôn ngữ cung cấp cho người lập trình và kiểu được người dùng định nghĩa (user-defined) do người lập trình tạo ra.

C# phân tập hợp kiểu dữ liệu này thành hai loại: Kiểu dữ liệu giá trị (value) và kiểu dữ liệu tham chiếu (reference). Việc phân chi này do sự khác nhau khi lưu kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu trong bộ nhớ. Đối với một kiểu dữ liệu giá trị thì sẽ được lưu giữ kích thước thật trong bộ nhớ đã cấp phát là stack. Trong khi đó thì địa chỉ của kiểu dữ liệu tham chiếu thì được lưu trong stack nhưng đối tượng thật sự thì lưu trong bộ nhớ heap.

Nếu chúng ta có một đối tượng có kích thước rất lớn thì việc lưu giữ chúng trên bộ nhớ heap rất có ích, trong chương 4 sẽ trình bày những lợi ích và bất lợi khi làm việc với kiểu dữ liệu tham chiếu, còn trong chương này chỉ tập trung kiểu dữ kiểu cơ bản hay kiểu xây dựng sẵn.

Ghi chú: Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng và chuỗi. Và tất cả các kiểu do người dùng định nghĩa ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu.

Ngoài ra C# cũng hỗ trợ một kiểu con trỏ C++, nhưng hiếm khi được sử dụng, và chỉ khi nào làm việc với những đoạn mã lệnh không được quản lý (unmanaged code). Mã lệnh không được quản lý là các lệnh được viết bên ngoài nền .MS.NET, như là các đối tượng COM.

Kiểu dữ liệu xây dựng sẵn

Ngôn ngữ C# đưa ra các kiểu dữ liệu xây dựng sẵn rất hữu dụng, phù hợp với một ngôn ngữ lập trình hiện đại, mỗi kiểu dữ liệu được ánh xạ đến một kiểu dữ liệu được hỗ trợ bởi hệ thống xác nhận ngôn ngữ chung (Common Language Specification: CLS) trong MS.NET. Việc ánh xạ các kiểu dữ liệu nguyên thuỷ của C# đến các kiểu dữ liệu của .NET sẽ đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng đồng thời với các đối tượng được tạo bởi bất cứ ngôn ngữ khác được biên dịch bởi .NET, như VB.NET.

Mỗi kiểu dữ liệu có một sự xác nhận và kích thước không thay đổi, không giống như C++, int trong C# luôn có kích thước là 4 byte bởi vì nó được ánh xạ từ kiểu Int32 trong . NET.

Bảng 3.1 sau sẽ mô tả một số các kiểu dữ liệu được xây dựng sẵn

Kiểu C#	Số byte	Kiểu .NET	Mô tả
byte	1	Byte	Số nguyên dương không dấu từ 0-255
char	2	Char	Ký tự Unicode
bool	1	Boolean	Giá trị logic true/ false
sbyte	1	Sbyte	Số nguyên có dấu (từ -128 đến 127)
short	2	Int16	Số nguyên có dấu giá trị từ -32768 đến 32767.
ushort	2	Uint16	Số nguyên không dấu 0 – 65.535
int	4	Int32	Số nguyên có dấu –2.147.483.647 và 2.147.483.647
uint	4	Uint32	Số nguyên không dấu 0 – 4.294.967.295
float	4	Single	Kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38, với 7 chữ số có nghĩa..
double	8	Double	Kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308, với 15,16 chữ số có nghĩa.
decimal	8	Decimal	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố “m” hay “M” theo sau giá trị.

long	8	Int64	Kiểu số nguyên có dấu có giá trị trong khoảng : -9.223.370.036.854.775.808 đến 9.223.372.036.854.775.807
ulong	8	Uint64	Số nguyên không dấu từ 0 đến 0xffffffffffffffff

Bảng 3.1 : Mô tả các kiểu dữ liệu xây dựng sẵn.

Ghi chú: Kiểu giá trị logic chỉ có thể nhận được giá trị là true hay false mà thôi. Một giá trị nguyên không thể gán vào một biến kiểu logic trong C# và không có bất cứ chuyển đổi ngầm định nào. Điều này khác với C/C++, cho phép biến logic được gán giá trị nguyên, khi đó giá trị nguyên 0 là false và các giá trị còn lại là true.

Chọn kiểu dữ liệu

Thông thường để chọn một kiểu dữ liệu nguyên để sử dụng như short, int hay long thường dựa vào độ lớn của giá trị muốn sử dụng. Ví dụ, một biến ushort có thể lưu giữ giá trị từ 0 đến 65.535, trong khi biến ulong có thể lưu giữ giá trị từ 0 đến 4.294.967.295, do đó tùy vào miền giá trị của phạm vi sử dụng biến mà chọn các kiểu dữ liệu thích hợp nhất. Kiểu dữ liệu int thường được sử dụng nhiều nhất trong lập trình vì với kích thước 4 byte của nó cũng đủ để lưu các giá trị nguyên cần thiết.

Kiểu số nguyên có dấu thường được lựa chọn sử dụng nhiều nhất trong kiểu số trừ khi có lý do chính đáng để sử dụng kiểu dữ liệu không dấu.

Stack và Heap

Stack là một cấu trúc dữ liệu lưu trữ thông tin dạng xếp chồng tức là vào sau ra trước (Last In First Out : LIFO), điều này giống như chúng ta có một chồng các đĩa, ta cứ xếp các đĩa vào chồng và khi lấy ra thì đĩa nào nằm trên cùng sẽ được lập ra trước, tức là đĩa vào sau sẽ được lấy ra trước.

Trong C#, kiểu giá trị như kiểu số nguyên được cấp phát trên stack, đây là vùng nhớ được thiết lập để lưu các giá trị, và vùng nhớ này được tham chiếu bởi tên của biến.

Kiểu tham chiếu như các đối tượng thì được cấp phát trên heap. Khi một đối tượng được cấp phát trên heap thì địa chỉ của nó được trả về, và địa chỉ này được gán đến một tham chiếu.

Thỉnh thoảng cơ chế thu gom sẽ hũy đối tượng trong stack sau khi một vùng trong stack được đánh dấu là kết thúc. Thông thường một vùng trong stack được định nghĩa bởi một hàm. Do đó, nếu chúng ta khai báo một biến cục bộ trong một hàm là một đối tượng thì đối tượng này sẽ đánh dấu để hũy khi kết thúc hàm.

Những đối tượng trên heap sẽ được thu gom sau khi một tham chiếu cuối cùng đến đối tượng đó được gọi.

Cách tốt nhất khi sử dụng biến không dấu là giá trị của biến luôn luôn dương, biến này thường thể hiện một thuộc tính nào đó có miền giá trị dương. Ví dụ khi cần khai báo một biến lưu giữ tuổi của một người thì ta dùng kiểu byte (số nguyên từ 0-255) vì tuổi của người không thể nào âm được.

Kiểu float, double, và decimal đưa ra nhiều mức độ khác nhau về kích thước cũng như độ chính xác.Với thao tác trên các phân số nhỏ thì kiểu float là thích hợp nhất. Tuy nhiên lưu ý rằng trình biên dịch luôn luôn hiểu bất cứ một số thực nào cũng là một số kiểu double trừ khi chúng ta khai báo rõ ràng. Để gán một số kiểu float thì số phải có ký tự f theo sau.

```
float soFloat = 24f;
```

Kiểu dữ liệu ký tự thể hiện các ký tự Unicode, bao gồm các ký tự đơn giản, ký tự theo mã Unicode và các ký tự thoát khác được bao trong những dấu nháy đơn. Ví dụ, A là một ký tự đơn giản trong khi \u0041 là một ký tự Unicode. Ký tự thoát là những ký tự đặc biệt bao gồm hai ký tự liên tiếp trong đó ký tự dầu tiên là dấu chéo '\'. Ví dụ, \t là dấu tab. Bảng 3.2 trình bày các ký tự đặc biệt.

Ký tự	Ý nghĩa
\'	Dấu nháy đơn
\"	Dấu nháy kép
\\	Dấu chéo
\0	Ký tự null
\a	Alert

\b	Backspace
\f	Sang trang form feed
\n	Dòng mới
\r	Đầu dòng
\t	Tab ngang
\v	Tab dọc

Bảng 3.2 : Các kiểu ký tự đặc biệt.

Chuyển đổi các kiểu dữ liệu

Những đối tượng của một kiểu dữ liệu này có thể được chuyển sang những đối tượng của một kiểu dữ liệu khác thông qua cơ chế chuyển đổi tường minh hay ngầm định. Chuyển đổi nhầm định được thực hiện một cách tự động, trình biên dịch sẽ thực hiện công việc này. Còn chuyển đổi tường minh diễn ra khi chúng ta gán ép một giá trị cho kiểu dữ liệu khác.

Việc chuyển đổi giá trị ngầm định được thực hiện một cách tự động và đảm bảo là không mất thông tin. Ví dụ, chúng ta có thể gán ngầm định một số kiểu short (2 byte) vào một số kiểu int (4 byte) một cách ngầm định. Sau khi gán hoàn toàn không mất dữ liệu vì bất cứ giá trị nào của short cũng thuộc về int:

```
short x = 10;
int y = x;  // chuyển đổi ngầm định
```

Tuy nhiên, nếu chúng ta thực hiện chuyển đổi ngược lại, chắc chắn chúng ta sẽ bị mất thông tin. Nếu giá trị của số nguyên đó lớn hơn 32.767 thì nó sẽ bị cắt khi chuyển đổi. Trình biên dịch sẽ không thực hiện việc chuyển đổi ngầm định từ số kiểu int sang số kiểu short:

```
short x;
int y = 100;
x = y;      // Không biên dịch, lỗi !!!
```

Để không bị lỗi chúng ta phải dùng lệnh gán tường minh, đoạn mã trên được viết lại như sau:

```
short x;
int y = 500;
x = (short) y;      // Ép kiểu tường minh, trình biên dịch không báo lỗi
```

Biến và hằng

Một biến là một vùng lưu trữ với một kiểu dữ liệu. Trong ví dụ trước cả x, và y điều là biến. Biến có thể được gán giá trị và cũng có thể thay đổi giá trị khi thực hiện các lệnh trong chương trình.

Để tạo một biến chúng ta phải khai báo kiểu của biến và gán cho biến một tên duy nhất. Biến có thể được khởi tạo giá trị ngay khi được khai báo, hay nó cũng có thể được gán một giá trị mới vào bất cứ lúc nào trong chương trình. Ví dụ 3.1 sau minh họa sử dụng biến.

Ví dụ 3.1: Khởi tạo và gán giá trị đến một biến.

```
class MinhHoaC3
{
    static void Main()
    {
        int bien1 = 9;
        System.Console.WriteLine("Sau khi khoi tao: bien1 ={0}", bien1);
        bien1 = 15;
        System.Console.WriteLine("Sau khi gan: bien1 ={0}", bien1);
    }
}
```

Kết quả:

```
Sau khi khoi tao: bien1 = 9
Sau khi gan: bien1 = 15
```

Ngay khi khai báo biến ta đã gán giá trị là 9 cho biến, khi xuất biến này thì biến có giá trị là 9. Thực hiện phép gán biến cho giá trị mới là 15 thì biến sẽ có giá trị là 15 và xuất kết quả là 15.

Gán giá trị xác định cho biến

C# đòi hỏi các biến phải được khởi tạo trước khi được sử dụng. Để kiểm tra luật này chúng ta thay đổi dòng lệnh khởi tạo biến bien1 trong ví dụ 3.1 như sau:

```
int  bien1;
```

và giữ nguyên phần còn lại ta được ví dụ 3.2:

Ví dụ 3.2: Sử dụng một biến không khởi tạo.

```
class MinhHoaC3
{
    static void Main()
    {
        int bien1;
        System.Console.WriteLine("Sau khi khoi tao: bien1 ={0}", bien1);
        bien1 = 15;
        System.Console.WriteLine("Sau khi gan: bien1 ={0}", bien1);
    }
}
```

Khi biên dịch đoạn chương trình trên thì trình biên dịch C# sẽ thông báo một lỗi sau:

```
...error CS0165: Use of unassigned local variable 'bien1'
```

Việc sử dụng biến khi chưa được khởi tạo là không hợp lệ trong C#. Ví dụ 3.2 trên không hợp lệ.

Tuy nhiên không nhất thiết lúc nào chúng ta cũng phải khởi tạo biến. Nhưng để dùng được thì bắt buộc phải gán cho chúng một giá trị trước khi có một lệnh nào tham chiếu đến biến đó. Điều này được gọi là gán giá trị xác định cho biến và C# bắt buộc phải thực hiện điều này.

Ví dụ 3.3 minh họa một chương trình đúng.

Ví dụ 3.3: Biến không được khi tạo nhưng sau đó được gán giá trị.

```
---------------------------------------------------------------------------
class MinhHoaC3
{
    static void Main()
    {
        int bien1;
        bien1 = 9;
        System.Console.WriteLine("Sau khi khoi tao: bien1 ={0}", bien1);
        bien1 = 15;
        System.Console.WriteLine("Sau khi gan: bien1 ={0}", bien1);
    }
}
---------------------------------------------------------------------------
```

Hằng

Hằng cũng là một biến nhưng giá trị của hằng không thay đổi. Biến là công cụ rất mạnh, tuy nhiên khi làm việc với một giá trị được định nghĩa là không thay đổi, ta phải đảm bảo giá trị của nó không được thay đổi trong suốt chương trình. Ví dụ, khi lập một chương trình thí nghiệm hóa học liên quan đến nhiệt độ sôi, hay nhiệt độ đông của nước, chương trình cần khai báo hai biến là DoSoi và DoDong, nhưng không cho phép giá trị của hai biến này bị thay đổi hay bị gán. Để ngăn ngừa việc gán giá trị khác, ta phải sử dụng biến kiểu hằng.

Hằng được phân thành ba loại: giá trị hằng (literal), biểu tượng hằng (symbolic constants), kiểu liệu kê (enumerations).

Giá trị hằng: ta có một câu lệnh gán như sau:

```
x = 100;
```

Giá trị 100 là giá trị hằng. Giá trị của 100 luôn là 100. Ta không thể gán giá trị khác cho 100 được.

Biểu tượng hằng: gán một tên cho một giá trị hằng, để tạo một biểu tượng hằng dùng từ khóa **const** và cú pháp sau:

```
<const> <type> <tên hằng> = <giá trị>;
```

Một biểu tượng hằng phải được khởi tạo khi khai báo, và chỉ khởi tạo duy nhất một lần trong suốt chương trình và không được thay đổi. Ví dụ:

```
const int DoSoi = 100;
```

Trong khai báo trên, 32 là một hằng số và DoSoi là một biểu tượng hằng có kiểu nguyên. Ví dụ 3.4 minh họa việc sử dụng những biểu tượng hằng.

Ví dụ 3.4: Sử dụng biểu tượng hằng.

```
---------------------------------------------------------------------------
class MinhHoaC3
{
    static void Main()
    {
        const int DoSoi = 100;   // Độ C
        const int DoDong = 0;    // Độ C
        System.Console.WriteLine( "Do dong cua nuoc {0}", DoDong );
        System.Console.WriteLine( "Do soi cua nuoc {0}", DoSoi );
    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Do dong cua nuoc 0
Do soi cua nuoc 100
---------------------------------------------------------------------------
```

Ví dụ 3.4 tạo ra hai biểu tượng hằng chứa giá trị nguyên: DoSoi và DoDong, theo qui tắc đặt tên hằng thì tên hằng thường được đặt theo cú pháp Pascal, nhưng điều này không đòi hỏi bởi ngôn ngữ nên ta có thể đặt tùy ý.

Việc dùng biểu thức hằng này sẽ làm cho chương trình được viết tăng thêm phần ý nghĩa cùng với sự dễ hiểu. Thật sự chúng ta có thể dùng hằng số là 0 và 100 thay thế cho hai biểu tượng hằng trên, nhưng khi đó chương trình không được dễ hiểu và không được tự nhiên lắm. Trình biên dịch không bao giờ chấp nhận một lệnh gán giá trị mới cho một biểu tượng hằng. Ví dụ 3.4 trên có thể được viết lại như sau

```
...
class MinhHoaC3
{
    static void Main()
    {
        const int DoSoi = 100;   // Độ C
        const int DoDong = 0;    // Độ C
        System.Console.WriteLine( "Do dong cua nuoc {0}", DoDong );
```

```
        System.Console.WriteLine( "Do soi cua nuoc {0}", DoSoi );
        DoSoi = 200;

    }
}
```

Khi đó trình biên dịch sẽ phát sinh một lỗi sau:

```
    error CS0131: The left-hand side of an assignment must be a variable,
property or indexer.
```

Kiểu liệt kê

Kiểu liệt kê đơn giản là tập hợp các tên hằng có giá trị không thay đổi (thường được gọi là danh sách liệt kê).

Trong ví dụ 3.4, có hai biểu tượng hằng có quan hệ với nhau:

```
const int DoDong = 0;
const int DoSoi = 100;
```

Do mục đích mở rộng ta mong muốn thêm một số hằng số khác vào danh sách trên, như các hằng sau:

```
const int DoNong  = 60;
const int DoAm = 40;
const int DoNguoi = 20;
```

Các biểu tượng hằng trên điều có ý nghĩa quan hệ với nhau, cùng nói về nhiệt độ của nước, khi khai báo từng hằng trên có vẻ cồng kềnh và không được liên kết chặt chẽ cho lắm. Thay vào đó C# cung cấp kiểu liệt kê để giải quyết vấn đề trên:

```
enum NhietDoNuoc
{
    DoDong = 0,
    DoNguoi = 20,
    DoAm  = 40,
    DoNong = 60,
    DoSoi  = 100,


}
```

Mỗi kiểu liệt kê có một kiểu dữ liệu cơ sở, kiểu dữ liệu có thể là bất cứ kiểu dữ liệu nguyên nào như int, short, long... tuy nhiên kiểu dữ liệu của liệt kê không chấp nhận kiểu ký tự. Để khai báo một kiểu liệt kê ta thực hiện theo cú pháp sau:

```
[thuộc tính] [bổ sung] enum <tên liệt kê> [:kiểu cơ sở] {danh sách các thành phần
liệt kê};
```

Thành phần thuộc tính và bổ sung là tự chọn sẽ được trình bày trong phần sau của sách. Trong phần này chúng ta sẽ tập trung vào phần còn lại của khai báo. Một kiểu liệt kê bắt đầu với từ khóa **enum**, tiếp sau là một định danh cho kiểu liệt kê:

enum NhietDoNuoc

Thành phần kiểu cơ sở chính là kiểu khai báo cho các mục trong kiểu liệt kê. Nếu bỏ qua thành phần này thì trình biên dịch sẽ gán giá trị mặc định là kiểu nguyên int, tuy nhiên chúng ta có thể sử dụng bất cứ kiểu nguyên nào như ushort hay long,..ngoại trừ kiểu ký tự. Đoạn ví dụ sau khai báo một kiểu liệt kê sử dụng kiểu cơ sở là số nguyên không dấu uint:

```
enum KichThuoc :uint
{
    Nho = 1,
    Vua = 2,
    Lon = 3,

}
```

Lưu ý là khai báo một kiểu liệt kê phải kết thúc bằng một danh sách liệt kê, danh sách liệt kê này phải có các hằng được gán, và mỗi thành phần phải phân cách nhau dấu phẩy.

Ta viết lại ví dụ minh họa 3-4 như sau.

Ví dụ 3.5: Sử dụng kiểu liệt kê để đơn giản chương trình.

```
---------------------------------------------------------------------------
class MinhHoaC3
{
    // Khai báo kiểu liệt kê
    enum NhietDoNuoc
    {
        DoDong = 0,
        DoNguoi = 20,
        DoAm  = 40,
        DoNong = 60,
        DoSoi  = 100,


    }
    static void Main()
    {
        System.Console.WriteLine( "Nhiet do dong: {0}", NhietDoNuoc.DoDong);
        System.Console.WriteLine( "Nhiet do nguoi: {0}", NhietDoNuoc.DoNguoi);
        System.Console.WriteLine( "Nhiet do am: {0}", NhietDoNuoc.DoAm);
        System.Console.WriteLine( "Nhiet do nong: {0}", NhietDoNuoc.DoNong);
        System.Console.WriteLine( "Nhiet do soi: {0}", NhietDoNuoc.DoSoi);


    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Nhiet do dong: 0
Nhiet do nguoi: 20
Nhiet do am: 40
Nhiet do nong: 60
Nhiet do soi: 100
--------------------------------------------------------------------------
```

Mỗi thành phần trong kiểu liệt kê tương ứng với một giá trị số, trong trường hợp này là một số nguyên. Nếu chúng ta không khởi tạo cho các thành phần này thì chúng sẽ nhận các giá trị tiếp theo với thành phần đầu tiên là 0.

Ta xem thử khai báo sau:

```
enum Thutu
{
    ThuNhat,
    ThuHai,
    ThuBa = 10,
    ThuTu


}
```

Khi đó giá trị của ThuNhat là 0, giá trị của ThuHai là 1, giá trị của ThuBa là 10 và giá trị của ThuTu là 11.

Kiểu liệt kê là một kiểu hình thức do đó bắt buộc phải thực hiện phép chuyển đổi tường minh với các kiêu giá trị nguyên:

```
int x = (int) ThuTu.ThuNhat;
```

Kiểu chuỗi ký tự

Kiểu dữ liệu chuỗi khá thân thiện với người lập trình trong bất cứ ngôn ngữ lập trình nào, kiểu dữ liệu chuỗi lưu giữ một mảng những ký tự.

Để khai báo một chuỗi chúng ta sử dụng từ khoá string tương tự như cách tạo một thể hiện của bất cứ đối tượng nào:

```
string chuoi;
```

Một hằng chuỗi được tạo bằng cách đặt các chuỗi trong dấu nháy đôi:

```
"Xin chao"
```

Đây là cách chung để khởi tạo một chuỗi ký tự với giá trị hằng:

```
string chuoi = "Xin chao"
```

Kiểu chuỗi sẽ được đề cập sâu trong chương 10.

Định danh

Định danh là tên mà người lập trình chỉ định cho các kiểu dữ liệu, các phương thức, biến, hằng, hay đối tượng.... Một định danh phải bắt đầu với một ký tự chữ cái hay dấu gạch dưới, các ký tự còn lại phải là ký tự chữ cái, chữ số, dấu gạch dưới.

Theo qui ước đặt tên của Microsoft thì đề nghị sử dụng *cú pháp lạc đà* (camel notation) bắt đầu bằng ký tự thường để đặt tên cho các biến là *cú pháp Pascal* (Pascal notation) với ký tự đầu tiên hoa cho cách đặt tên hàm và hầu hết các định danh còn lại. Hầu như Microsoft không còn dùng cú pháp Hungary như iSoNguyen hay dấu gạch dưới Bien_Nguyen để đặt các định danh.

Các định danh không được trùng với các từ khoá mà C# đưa ra, do đó chúng ta không thể tạo các biến có tên như class hay int được. Ngoài ra, C# cũng phân biệt các ký tự thường và ký tự hoa vì vậy C# xem hai biến bienNguyen và bienguyen là hoàn toàn khác nhau.

Biểu thức

Những câu lệnh mà thực hiện việc đánh giá một giá trị gọi là biểu thức. Một phép gán một giá trị cho một biến cũng là một biểu thức:

```
var1 = 24;
```

Trong câu lệnh trên phép đánh giá hay định lượng chính là phép gán có giá trị là 24 cho biến var1. Lưu ý là toán tử gán ('=') không phải là toán tử so sánh. Do vậy khi sử dụng toán tử này thì biến bên trái sẽ nhận giá trị của phần bên phải. Các toán tử của ngôn ngữ C# như phép so sánh hay phép gán sẽ được trình bày chi tiết trong mục toán tử của chương này.

Do var1 = 24 là một biểu thức được định giá trị là 24 nên biểu thức này có thể được xem như phần bên phải của một biểu thức gán khác:

```
var2 = var1 = 24;
```

Lệnh này sẽ được thực hiện từ bên phải sang khi đó biến var1 sẽ nhận được giá trị là 24 và tiếp sau đó thì var2 cũng được nhận giá trị là 24. Do vậy cả hai biến đều cùng nhận một giá trị là 24. Có thể dùng lệnh trên để khởi tạo nhiều biến có cùng một giá trị như:

```
a = b = c = d = 24;
```

Khoảng trắng (whitespace)

Trong ngôn ngữ C#, những khoảng trắng, khoảng tab và các dòng được xem như là khoảng trắng (whitespace), giống như tên gọi vì chỉ xuất hiện những khoảng trắng để đại diện cho các ký tự đó. C# sẽ bỏ qua tất cả các khoảng trắng đó, do vậy chúng ta có thể viết như sau:

```
var1 = 24;
```

hay

```
var1   =  24 ;
```

và trình biên dịch C# sẽ xem hai câu lệnh trên là hoàn toàn giống nhau.

Tuy nhiên lưu ý là khoảng trắng trong một chuỗi sẽ không được bỏ qua. Nếu chúng ta viết:

```
System.WriteLine("Xin chao!");
```

mỗi khoảng trắng ở giữa hai chữ "Xin" và "chao" đều được đối xử bình thường như các ký tự khác trong chuỗi.

Hầu hết việc sử dụng khoảng trắng như một sự tùy ý của người lập trình. Điều cốt yếu là việc sử dụng khoảng trắng sẽ làm cho chương trình dễ nhìn dễ đọc hơn Cũng như khi ta viết một văn bản trong MS Word nếu không trình bày tốt thì sẽ khó đọc và gây mất cảm tình cho người xem. Còn đối với trình biên dịch thì việc dùng hay không dùng khoảng trắng là không khác nhau.

Tuy nhiên, củng cần lưu ý khi sử dụng khoảng trắng như sau:

```
int x = 24;
```

tương tự như:

```
int x=24;
```

nhưng không giống như:

```
intx=24;
```

Trình biên dịch nhận biết được các khoảng trắng ở hai bên của phép gán là phụ và có thể bỏ qua, nhưng khoảng trắng giữa khai báo kiểu và tên biến thì không phải phụ hay thêm mà bắt buộc phải có tối thiểu một khoảng trắng. Điều này không có gì bất hợp lý, vì khoảng trắng cho phép trình biên dịch nhận biết được từ khoá int và không thể nào nhận được intx.

Tương tự như C/C++, trong C# câu lệnh được kết thúc với dấu chấm phẩy ';'. Do vậy có thể một câu lệnh trên nhiều dòng, và một dòng có thể nhiều câu lệnh nhưng nhất thiết là hai câu lệnh phải cách nhau một dấu chấm phẩy.

Câu lệnh (statement)

Trong C# một chỉ dẫn lập trình đầy đủ được gọi là câu lệnh. Chương trình bao gồm nhiều câu lệnh tuần tự với nhau. Mỗi câu lệnh phải kết thúc với một dấu chấm phẩy, ví dụ như:

```
int x; // một câu lệnh
x = 32; // câu lệnh khác
int y =x; // đây cũng là một câu lệnh
```

Những câu lệnh này sẽ được xử lý theo thứ tự. Đầu tiên trình biên dịch bắt đầu ở vị trí đầu của danh sách các câu lệnh và lần lượt đi từng câu lệnh cho đến lệnh cuối cùng, tuy nhiên chỉ đúng cho trường hợp các câu lệnh tuần tự không phân nhánh.

Có hai loại câu lệnh phân nhánh trong C# là : phân nhánh không có điều kiện (unconditional branching statement) và phân nhánh có điều kiện (conditional branching statement).

Ngoài ra còn có các câu lệnh làm cho một số đoạn chương trình được thực hiện nhiều lần, các câu lệnh này được gọi là câu lệnh lặp hay vòng lặp. Bao gồm các lệnh lặp **for**, **while**, **do**, **in**, và **each** sẽ được đề cập tới trong mục tiếp theo.

Sau đây chúng ta sẽ xem xét hai loại lệnh phân nhánh phổ biến nhất trong lập trình C#.

Phân nhánh không có điều kiện

Phân nhánh không có điều kiện có thể tạo ra bằng hai cách: gọi một hàm và dùng từ khoá phân nhánh không điều kiện.

Gọi hàm

Khi trình biên dịch xử lý đến tên của một hàm, thì sẽ ngưng thực hiện hàm hiện thời mà bắt đầu phân nhánh dể tạo một gọi hàm mới. Sau khi hàm vừa tạo thực hiện xong và trả về một giá trị thì trình biên dịch sẽ tiếp tục thực hiện dòng lệnh tiếp sau của hàm ban đầu. ví dụ 3.6 minh họa cho việc phân nhánh khi gọi hàm.

Ví dụ 3.6: Gọi một hàm.

```
---------------------------------------------------------------------------
using System;
class GoiHam
{

    static void Main()
    {
        Console.WriteLine( "Ham Main chuan bi goi ham Func()..." );
        Func();
        Console.WriteLine( "Tro lai ham Main()" );
    }
    static void Func()
    {
        Console.WriteLine( "---->Toi la ham Func()...");
    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Ham Main chuan bi goi ham Func()...
---->Toi la ham Func()...
Tro lai ham Main()
---------------------------------------------------------------------------
```

Luồng chương trình thực hiện bắt đầu từ hàm Main xử lý đến dòng lệnh Func(), lệnh Func() thường được gọi là một lời gọi hàm. Tại điểm này luồng chương trình sẽ rẽ nhánh để thực hiện hàm vừa gọi. Sau khi thực hiện xong hàm Func, thì chương trình quay lại hàm Main và thực hiện câu lệnh ngay sau câu lệnh gọi hàm Func.

Từ khoá phân nhánh không điều kiện

Để thực hiện phân nhánh ta gọi một trong các từ khóa sau: **goto**, **break**, **continue**, **return**, **statementthrow**. Việc trình bày các từ khóa phân nhánh không điều kiện này sẽ được đề cập trong chương tiếp theo. Trong phần này chỉ đề cập chung không đi vào chi tiết.

Phân nhánh có điều kiện

Phân nhánh có điều kiện được tạo bởi các lệnh điều kiện. Các từ khóa của các lệnh này như : **if**, **else**, **switch**. Sự phân nhánh chỉ được thực hiện khi biểu thức điều kiện phân nhánh được xác định là đúng.

Câu lệnh if...else

Câu lệnh phân nhánh **if**...**else** dựa trên một điều kiện. Điều kiện là một biểu thức sẽ được kiểm tra giá trị ngay khi bắt đầu gặp câu lệnh đó. Nếu điều kiện được kiểm tra là đúng, thì câu lệnh hay một khối các câu lệnh bên trong thân của câu lệnh **if** được thực hiện.

Trong câu điều kiện **if**...**else** thì **else** là phần tùy chọn. Các câu lệnh bên trong thân của **else** chỉ được thực hiện khi điều kiện của **if** là sai. Do vậy khi câu lệnh đầy đủ **if**...**else** được dùng thì chỉ có một trong hai **if** hoặc **else** được thực hiện. Ta có cú pháp câu điều kiện **if**... **else** sau:

```
if (biểu thức điều kiện)
    <Khối lệnh thực hiện khi điều kiện đúng>
    [else
    <Khối lệnh thực hiện khi điều kiện sai>]
```

Nếu các câu lệnh trong thân của **if** hay **else** mà lớn hơn một lệnh thì các lệnh này phải được bao trong một khối lệnh, tức là phải nằm trong dấu khối { }:

```
if (biểu thức điều kiện)
    {
        <lệnh 1>
        <lệnh 2>
        ....
    }
[else
    {
        <lệnh 1>
        <lệnh 2>
        ...
    }]
```

Như trình bày bên trên do **else** là phần tùy chọn nên được đặt trong dấu ngoặc vuông [...]. Minh họa 3.7 bên dưới cách sử dụng câu lệnh **if**...**else**.

Ví dụ 3.7: Dùng câu lệnh điều kiện if...else.

```
---------------------------------------------------------------------------
using System;
class ExIfElse
{
    static void Main()
    {
        int var1 = 10;
        int var2 = 20;
        if ( var1 > var2)
```

```
        {
            Console.WriteLine( "var1: {0} > var2:{1}", var1, var2);
        }
        else
        {
            Console.WriteLine( "var2: {0} > var1:{1}", var2, var1);
        }
        var1 = 30;
        if ( var1 > var2)
        {
            var2 = var1++;
            Console.WriteLine( "Gan gia tri var1 cho var2");
            Console.WriteLine( "Tang bien var1 len mot ");
            Console.WritelLine( "Var1 = {0}, var2 = {1}", var1, var2);
        }
        else
        {
            var1 = var2;
            Console.WriteLine( "Thiet lap gia tri var1 = var2" );
            Console.WriteLine( "var1 = {0}, var2 = {1}", var1, var2 );
        }
    }
}
```

Kết quả:

```
Gan gia tri var1 cho var2
Tang bien var1 len mot
Var1 = 31, var2 = 30
```

Trong ví dụ 3.7 trên, câu lệnh **if** đầu tiên sẽ kiểm tra xem giá trị của var1 có lớn hơn giá trị của var2 không. Biểu thức điều kiện này sử dụng toán tử quan hệ lớn hơn (>), các toán tử khác như nhỏ hơn (<), hay bằng (==). Các toán tử này thường xuyên được sử dụng trong lập trình và kết quả trả là giá trị đúng hay sai.

Việc kiểm tra xác định giá trị var1 lớn hơn var2 là sai (vì var1 = 10 trong khi var2 = 20), khi đó các lệnh trong **else** sẽ được thực hiện, và các lệnh này in ra màn hình:

```
var2: 20 > var1: 10
```

Tiếp theo đến câu lệnh **if** thứ hai, sau khi thực hiện lệnh gán giá trị của var1 = 30, lúc này điều kiện **if** đúng nên các câu lệnh trong khối **if** sẽ được thực hiện và kết quả là in ra ba dòng sau:

```
Gan gia tri var1 cho var2
Tang bien var1 len mot
Var1 = 31, var2 = 30
```

*Câu lệnh **if** lồng nhau*

Các lệnh điều kiện **if** có thể lồng nhau để phục vụ cho việc xử lý các câu điều kiện phức tạp. Việc này cũng thường xuyên gặp khi lập trình. Giả sử chúng ta cần viết một chương trình có yêu cầu xác định tình trạng kết hôn của một công dân dựa vào các thông tin như tuổi, giới tính, và tình trạng hôn nhân, dựa trên một số thông tin như sau:

- Nếu công dân là nam thì độ tuổi có thể kết hôn là 20 với điều kiện là chưa có gia đình.
- Nếu công dân là nữ thì độ tuổi có thể kết hôn là 19 cũng với điều kiện là chưa có gia đình.
- Tất cả các công dân có tuổi nhỏ hơn 19 điều không được kết hôn.

Dựa trên các yêu cầu trên ta có thể dùng các lệnh **if** lồng nhau để thực hiện. Ví dụ 3.8 sau sẽ minh họa cho việc thực hiện các yêu cầu trên.

*Ví dụ 3.8: Các lệnh **if** lồng nhau.*

```
---------------------------------------------------------------------------
using System;
class TinhTrangKetHon
{
    static void Main()
    {
        int tuoi;
        bool coGiaDinh; // 0: chưa có gia đình; 1: đã có gia đình
        bool gioiTinh; // 0: giới tính nữ; 1: giới tính nam
        tuoi = 24;
        coGiaDinh = false; // chưa có gia đình
        gioiTinh = true; // nam

        if ( tuoi >= 19)
        {
            if ( coGiaDinh == false)
            {
                if ( gioiTinh == false) // nu
                    Console.WriteLine(" Nu co the ket hon");
                else // nam
```

```
                if (tuoi >19) // phải lớn hơn 19 tuoi mới được kết hôn
                    Console.WriteLine(" Nam co the ket hon");
            }
            else // da co gia dinh
                Console.WriteLine(" Khong the ket hon nua do da ket hon");
        }
        else // tuoi < 19
            Console.WriteLine(" Khong du tuoi ket hon" );
    }
}
```

Kết quả:

```
Nam co the ket hon
```

Theo trình tự kiểm tra thì câu lệnh **if** đầu tiên được thực hiện, biểu thức điều kiện đúng do tuổi có giá trị là 24 lớn hơn 19. Khi đó khối lệnh trong **if** sẽ được thực thi. Ở trong khối này lại xuất hiện một lệnh **if** khác để kiểm tra tình trạng xem người đó đã có gia đình chưa, kết quả điều kiện **if** là đúng vì coGiaDinh = false nên biểu thức so sánh coGiaDinh == false sẽ trả về giá trị đúng. Tiếp tục xét xem giới tính của người đó là nam hay nữ, vì chỉ có nam trên 19 tuổi mới được kết hôn. Kết quả kiểm tra là nam nên câu lệnh **if** thứ ba được thực hiện và xuất ra kết quả : "`Nam co the ket hon`".

Câu lệnh switch

Khi có quá nhiều điều kiện để chọn thực hiện thì dùng câu lệnh **if** sẽ rất rối rắm và dài dòng, Các ngôn ngữ lập trình cấp cao đều cung cấp một dạng câu lệnh **switch** liệt kê các giá trị và chỉ thực hiện các giá trị thích hợp. C# cũng cung cấp câu lệnh nhảy **switch** có cú pháp sau:

```
switch (biểu thức điều kiện)
{
    case <giá trị>:
        <Các câu lệnh thực hiện>
        <lệnh nhảy>
    [default:
        <Các câu lệnh thực hiện mặc định>]
}
```

Cũng tương tự như câu lệnh if, biểu thức để so sánh được đặt sau từ khóa **switch**, tuy nhiên giá trị so sánh lại được đặt sau mỗi các từ khóa **case**. Giá trị sau từ khóa **case** là các giá trị hằng số nguyên như đã đề cập trong phần trước.

Nếu một câu lệnh **case** được thích hợp tức là giá trị sau **case** bằng với giá trị của biểu thức sau **switch** thì các câu lệnh liên quan đến câu lệnh **case** này sẽ được thực thi. Tuy nhiên phải có một câu lệnh nhảy như **break**, **goto** để điều khiển nhảy qua các **case** khác.Vì nếu không có các lệnh nhảy này thì khi đó chương trình sẽ thực hiện tất cả các **case** theo sau. Để dễ hiểu hơn ta sẽ xem xét ví dụ 3.9 dưới đây.

Ví dụ 3.9: Câu lệnh switch.

```
---------------------------------------------------------------------------
using System;
class MinhHoaSwitch
{

        static void Main()
        {
            const int mauDo = 0;
            const int mauCam = 1;
            const int mauVang = 2;
            const int mauLuc = 3;
            const int mauLam = 4;
            const int mauCham = 5;
            const int mauTim = 6;
            int chonMau = mauLuc;




            switch ( chonMau )
            {
                case mauDo:
                    Console.WriteLine( "Ban cho mau do" );
                    break;
                case mauCam:
                    Console.WriteLine( "Ban cho mau cam" );
                    break;
                case mauVang:
                    //Console.WriteLine( "Ban chon mau vang");
                case mauLuc:
                    Console.WriteLine( "Ban chon mau luc");
                    break;
                case mauLam:
                    Console.WriteLine( "Ban chon mau lam");
                    goto case mauCham;
                case mauCham:
```

```
                Console.WriteLine( "Ban cho mau cham");
                goto case mauTim;
            case mauTim:
                Console.WriteLine( "Ban chon mau tim");
                goto case mauLuc;
            default:
                Console.WriteLine( "Ban khong chon mau nao het");
                break;
        }
        Console.WriteLine( "Xin cam on!");
    }
}
```

Trong ví dụ 3.9 trên liệt kê bảy loại màu và dùng câu lệnh **switch** để kiểm tra các trường hợp chọn màu.Ở đây chúng ta thử phân tích từg câu lệnh **case** mà không quan tâm đến giá trị biến chonMau.

Giá trị chonMau	Câu lệnh case thực hiện	Kết quả thực hiện
mauDo	case mauDo	Ban chon mau do
mauCam	case mauCam	Ban chon mau cam
mauVang	case mauVang case mauLuc	Ban chon mau luc
mauLuc	case mauLuc	Ban chon mau luc
mauLam	case mauLam case mauCham case mauTim case mauLuc	Ban chon mau lam Ban chon mau cham Ban chon mau tim Ban chon mau luc
mauCham	case mauCham	Ban chon mau cham
mauTim	case mauTim case mauLuc case mauTim	Ban chon mau tim Ban chon mau luc Ban chon mau tim
	case mauLuc	Ban chon mau luc

Bảng 3.3: Mô tả các trường hợp thực hiện câu lệnh switch.

Trong đoạn ví dụ do giá trị của biến chonMau = mauLuc nên khi vào lệnh **switch** thì **case** mauLuc sẽ được thực hiện và kết quả như sau:

Kết quả ví dụ 3.9

```
Ban chon mau luc
Xin cam on!
--------------------------------------------------------------------------------
```

Ghi chú: Đối với người lập trình C/C++, trong C# chúng ta không thể nhảy xuống một trường hợp **case** tiếp theo nếu câu lệnh **case** hiện tại không rỗng. Vì vậy chúng ta phải viết như sau:

```
case 1:     // nhảy xuống
case 2:
```

Như minh họa trên thì trường hợp xử lý **case** 1 là rỗng, tuy nhiên chúng ta không thể viết như sau:

```
case 1:
    DoAnything();
    // Trường hợp này không thể nhảy xuống case 2
case 2:
```

trong đoạn chương trình thứ hai trường hợp **case** 1 có một câu lệnh nên không thể nhảy xuống được. Nếu muốn trường hợp case1 nhảy qua **case** 2 thì ta phải sử dụng câu lệnh **goto** một các tường minh:

```
case 1:
    DoAnything();
    goto case 2;
case 2:
```

Do vậy khi thực hiện xong các câu lệnh của một trường hợp nếu muốn thực hiện một trường hợp **case** khác thì ta dùng câu lệnh nhảy **goto** với nhãn của trường hợp đó:

```
goto case <giá trị>
```

Khi gặp lệnh thoát **break** thì chương trình thoát khỏi **switch** và thực hiện lệnh tiếp sau khối **switch** đó.

Nếu không có trường hợp nào thích hợp và trong câu lệnh **switch** có dùng câu lệnh **defalut** thì các câu lệnh của trường hợp **default** sẽ được thực hiện. Ta có thể dùng **default** để cảnh báo một lỗi hay xử lý một trường hợp ngoài tất cả các trường hợp **case** trong **switch**.

Trong ví dụ minh họa câu lệnh **switch** trước thì giá trị để kiểm tra các trường hợp thích hợp là các hằng số nguyên. Tuy nhiên C# còn có khả năng cho phép chúng ta dùng câu lệnh **switch** với giá trị là một chuỗi, có thể viết như sau:

```
switch (chuoi1)
{
    case "mau do":
        ....
        break;
    case "mau cam":
```

```
            ...
            break;

        ...
    }
```

Câu lệnh lặp

C# cung cấp một bộ mở rộng các câu lệnh lặp, bao gồm các câu lệnh lặp **for**, **while** và **do**... **while**. Ngoài ra ngôn ngữ C# còn bổ sung thêm một câu lệnh lặp **foreach**, lệnh này mới đối với người lập trình C/C++ nhưng khá thân thiện với người lập trình VB. Cuối cùng là các câu lệnh nhảy như **goto**, **break, continue**, và **return**.

Câu lệnh nhảy goto

Lệnh nhảy **goto** là một lệnh nhảy đơn giản, cho phép chương trình nhảy vô điều kiện tới một vị trí trong chương trình thông qua tên nhãn. Tuy nhiên việc sử dụng lệnh **goto** thường làm mất đi tính cấu trúc thuật toán, việc lạm dụng sẽ dẫn đến một chương trình nguồn mà giới lập trình gọi là "*mì ăn liền*" rối như mớ bòng bong vậy. Hầu hết các người lập trình có kinh nghiệm đều tránh dùng lệnh **goto**. Sau đây là cách sử dụng lệnh nhảy **goto**:

- Tạo một nhãn
- **goto** đến nhãn

Nhãn là một định danh theo sau bởi dấu hai chấm (:). Thường thường một lệnh **goto** gắn với một điều kiện nào đó, ví dụ 3.10 sau sẽ minh họa các sử dụng lệnh nhảy **goto** trong chương trình.

Ví dụ 3.10: Sử dụng goto.

```
-------------------------------------------------------------------------
using System;
public class UsingGoto
{
    public static int Main()
    {
        int i = 0;
        lap:    // nhãn
        Console.WriteLine("i:{0}",i);
        i++;
        if ( i < 10 )
            goto lap;  // nhãy về nhãn lap
        return 0;
    }
}
-------------------------------------------------------------------------
```

Kết quả:

```
i:0
i:1
i:2
i:3
i:4
i:5
i:6
i:7
i:8
i:9
```

Nếu chúng ta vẽ lưu đồ của một chương trình có sử dụng nhiều lệnh **goto**, thì ta sẽ thấy kết quả rất nhiều đường chồng chéo lên nhau, giống như là các sợi mì vậy. Chính vì vậy nên những đoạn mã chương trình có dùng lệnh **goto** còn được gọi là "*spaghetti code*".

Việc tránh dùng lệnh nhảy **goto** trong chương trình hoàn toàn thực hiện được, có thể dùng vòng lặp **while** để thay thế hoàn toàn các câu lệnh **goto**.

Vòng lặp while

Ý nghĩa của vòng lặp **while** là: "*Trong khi điều kiện đúng thì thực hiện các công việc này*".

Cú pháp sử dụng vòng lặp **while** như sau:

```
while (Biểu thức)
    <Câu lệnh thực hiện>
```

Biểu thức của vòng lặp **while** là điều kiện để các lệnh được thực hiện, biểu thức này bắt buộc phải trả về một giá trị kiểu bool là true/false. Nếu có nhiều câu lệnh cần được thực hiện trong vòng lặp **while** thì phải đặt các lệnh này trong khối lệnh. Ví dụ 3.11 minh họa việc sử dụng vòng lặp **while**.

Ví dụ 3.11: Sử dụng vòng lặp while.

```
using System;
public class UsingWhile
{
    public static int Main()
    {
        int i = 0;
        while ( i < 10 )
        {
            Console.WriteLine(" i: {0} ",i);
            i++;
```

```
        }
        return 0;
    }
}
```

Kết quả:

```
i:0
i:1
i:2
i:3
i:4
i:5
i:6
i:7
i:8
i:9
```

Đoạn chương trình 3.11 cũng cho kết quả tương tự như chương trình minh họa 3.10 dùng lệnh **goto**. Tuy nhiên chương trình 3.11 rõ ràng hơn và có ý nghĩa tự nhiên hơn. Có thể diễn giải ngôn ngữ tự nhiên đoạn vòng lặp **while** như sau: "*Trong khi i nhỏ hơn 10, thì in ra giá trị của i và tăng i lên một đơn vị*".

Lưu ý rằng vòng lặp **while** sẽ kiểm tra điều kiện trước khi thực hiện các lệnh bên trong, điều này đảm bảo nếu ngay từ đầu điều kiện sai thì vòng lặp sẽ không bao giờ thực hiện. do vậy nếu khởi tạo biến i có giá trị là 11, thì vòng lặp sẽ không được thực hiện.

Vòng lặp do...while

Đôi khi vòng lặp **while** không thoả mãn yêu cầu trong tình huống sau, chúng ta muốn chuyển ngữ nghĩa của **while** là "*chạy trong khi điều kiện đúng*" thành ngữ nghĩa khác như "*làm điều này trong khi điều kiện vẫn còn đúng*". Nói cách khác thực hiện một hành động, và sau khi hành động được hoàn thành thì kiểm tra điều kiện. Cú pháp sử dụng vòng lặp **do**...**while** như sau:

```
do
    <Câu lệnh thực hiện>
while ( điều kiện )
```

Ở đây có sự khác biệt quan trọng giữa vòng lặp **while** và vòng lặp **do**...**while** là khi dùng vòng lặp **do**...**while** thì tối thiểu sẽ có một lần các câu lệnh trong **do**...**while** được thực hiện. Điều này cũng dễ hiểu vì lần đầu tiên đi vào vòng lặp **do**...**while** thì điều kiện chưa được kiểm tra.

Ví dụ 3.12: Minh hoạ việc sử dụng vòng lặp do..while.

```
-------------------------------------------------------------------------
using System;
public class UsingDoWhile
{
    public static int Main( )
    {
        int i = 11;
        do
        {
            Console.WriteLine("i: {0}",i);
            i++;
        } while ( i < 10 )
        return 0;
    }
}
-------------------------------------------------------------------------
```

Kết quả:

```
i: 11
-------------------------------------------------------------------------
```

Do khởi tạo biến i giá trị là 11, nên điều kiện của **while** là sai, tuy nhiên vòng lặp **do**...**while** vẫn được thực hiện một lần.

Vòng lặp for

Vòng lặp **for** bao gồm ba phần chính:

- Khởi tạo biến đếm vòng lặp
- Kiểm tra điều kiện biến đếm, nếu đúng thì sẽ thực hiện các lệnh bên trong vòng **for**
- Thay đổi bước lặp.

Cú pháp sử dụng vòng lặp **for** như sau:

```
for ([ phần khởi tạo] ; [biểu thức điều kiện]; [bước lặp])
    <Câu lệnh thực hiện>
```

Vòng lặp **for** được minh họa trong ví dụ sau:

Ví dụ 3.13: Sử dụng vòng lặp for.

```
-------------------------------------------------------------------------
using System;
public class UsingFor
{
    public static int Main()
    {
        for (int i = 0; i < 30; i++)
```

```
        {
            if (i %10 ==0)
            {
                Console.WriteLine("{0} ",i);
            }
            else
            {
                Console.Write("{0} ",i);
            }
        }
        return 0;
    }
}
```

Kết quả:

```
0
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29
```

Trong đoạn chương trình trên có sử dụng toán tử chia lấy dư modulo, toán tử này sẽ được đề cập đến phần sau. Ý nghĩa lệnh i%10 == 0 là kiểm tra xem i có phải là bội số của 10 không, nếu i là bội số của 10 thì sử dụng lệnh WriteLine để xuất giá trị i và sau đó đưa cursor về đầu dòng sau. Còn ngược lại chỉ cần xuất giá trị của i và không xuống dòng.

Đầu tiên biến i được khởi tạo giá trị ban đầu là 0, sau đó chương trình sẽ kiểm tra điều kiện, do 0 nhỏ hơn 30 nên điều kiện đúng, khi đó các câu lệnh bên trong vòng lặp **for** sẽ được thực hiện. Sau khi thực hiện xong thì biến i sẽ được tăng thêm một đơn vị (i++).

Có một điều lưu ý là biến i do khai bao bên trong vòng lặp **for** nên chỉ có phạm vi hoạt động bên trong vòng lặp. Ví dụ 3.14 sau sẽ không được biên dịch vì xuất hiện một lỗi.

Ví dụ 3.14: Phạm vi của biến khai báo trong vòng lặp.

```
using System;
public class UsingFor
{
    public static int Main()
    {
        for (int i = 0; i < 30; i++)
        {
```

```
            if (i %10 ==0)
            {
                Console.WriteLine("{0} ",i);
            }
            else
            {
                Console.Write("{0} ",i);
            }
        }
        // Lệnh sau sai do biến i chỉ được khai báo bên trong vòng lặp
        Console.WriteLine(" Ket qua cuoi cung cua i:{0}",i);
        return 0;
    }
}
```

Câu lệnh lặp foreach

Vòng lặp **foreach** cho phép tạo vòng lặp thông qua một tập hợp hay một mảng. Đây là một câu lệnh lặp mới không có trong ngôn ngữ C/C++. Câu lệnh **foreach** có cú pháp chung như sau:

```
foreach ( <kiểu tập hợp> <tên truy cập thành phần > in < tên tập hợp>)
    <Các câu lệnh thực hiện>
```

Do lặp dựa trên một mảng hay tập hợp nên toàn bộ vòng lặp sẽ duyệt qua tất cả các thành phần của tập hợp theo thứ tự được sắp. Khi duyệt đến phần tử cuối cùng trong tập hợp thì chương trình sẽ thoát ra khỏi vòng lặp **foreach**.

Ví dụ 3.15 minh họa việc sử dụng vòng lặp foreach.

```
using System;
public class UsingForeach
{
        public static int Main()
        {
            int[] intArray = {1,2,3,4,5,6,7,8,9,10};
            foreach( int item in intArray)
            {
                Console.Write("{0} ", item);
            }
            return 0;
        }
```

}

Kết quả:

```
1 2 3 4 5 6 7 8 9 10
```

---

*Câu lệnh nhảy **break** và **continue***

Khi đang thực hiện các lệnh trong vòng lặp, có yêu cầu như sau: không thực hiện các lệnh còn lại nữa mà thoát khỏi vòng lặp, hay không thực hiện các công việc còn lại của vòng lặp hiện tại mà nhảy qua vòng lặp tiếp theo. Để đáp ứng yêu cầu trên C# cung cấp hai lệnh nhảy là **break** và **continue** để thoát khỏi vòng lặp.

Break khi được sử dụng sẽ đưa chương trình thoát khỏi vòng lặp và tiếp tục thực hiện các lệnh tiếp ngay sau vòng lặp.

Continue ngừng thực hiện các công việc còn lại của vòng lặp hiện thời và quay về đầu vòng lặp để thực hiện bước lặp tiếp theo

Hai lệnh **break** và **continue** tạo ra nhiều điểm thoát và làm cho chương trình khó hiểu cũng như là khó duy trì. Do vậy phải cẩn trọng khi sử dụng các lệnh nhảy này.

Ví dụ 3.16 sẽ được trình bày bên dưới minh họa cách sử dụng lệnh **continue** và **break**. Đoạn chương trình mô phỏng hệ thống xử lý tín hiệu giao thông đơn giản. Tín hiệu mô phỏng là các ký tự chữ hoa hay số được nhập vào từ bàn phím, sử dụng hàm ReadLine của lớp Console để đọc một chuỗi ký tự từ bàn phím.

Thuật toán của chương trình khá đơn giản: Khi nhận tín hiệu '0' có nghĩa là mọi việc bình thường, không cần phải làm bất cứ công việc gì cả, kể cả việc ghi lại các sự kiện. Trong chương trình này đơn giản nên các tín hiệu được nhập từ bàn phím, còn trong ứng dụng thật thì tín hiệu này sẽ được phát sinh theo các mẫu tin thời gian trong cơ sở dữ liệu. Khi nhận được tín hiệu thoát (mô phỏng bởi ký tự 'T') thì ghi lại tình trạng và kết thúc xử lý. Cuối cùng, bất cứ tín hiệu nào khác sẽ phát ra một thông báo, có thể là thông báo đến nhân viên cảnh sát chẳng hạn...Trường hợp tín hiệu là 'X' thì cũng sẽ phát ra một thông báo nhưng sau vòng lặp xử lý cũng kết thúc.

*Ví dụ 3.16: Sử dụng **break** và **continue**.*

```
using System;
public class TrafficSignal
{
        public static int Main()
        {
            string signal = "0"; // Khởi tạo tín hiệu
            // bắt đầu chu trình xử lý tín hiệu
            while ( signal != "X")
```

```
        {
            //nhập tín hiệu
            Console.Write("Nhap vao mot tin hieu: ");
            signal = Console.ReadLine();
            // xuất tín hiệu hiện thời
            Console.WriteLine("Tin hieu nhan duoc: {0}", signal);
            // phần xử lý tín hiệu
            if (signal == "T")
            {
                // Tín hiệu thoát được gởi
                // lưu lại sự kiện và thoát
                Console.WriteLine("Ngung xu ly! Thoat\n");
                break;
            }
            if ( signal == "0")
            {
                // Tín hiệu nhận được bình thường
                // Lưu lại sự kiện và tiếp tục
                Console.WriteLine("Tat ca dieu tot!\n");
                continue;
            }
            // Thực hiện một số hành động nào đó
            // và tiếp tục
            Console.WriteLine("---bip bip bip\n");
        }
        return 0;
    }
}
```

Kết quả: sau khi nhập tuần tự các tín hiệu : "0", "B", "T"

```
Nhap vao mot tin hieu: 0
Tin hieu nhan duoc: 0
Tat ca dieu tot!


Nhap vao mot tin hieu: B
Tin hieu nhan duoc: B
---bip bip bip
```

```
Nhap vao mot tin hieu: T
Tin hieu nhan duoc: T
Ngung xu ly! Thoat

--------------------------------------------------------------------------
```

Điểm chính yếu của đoạn chương trình trên là khi nhập vào tín hiệu "T" thì sau khi thực hiện một số hành động cần thiết chương trình sẽ thoát ra khỏi vòng lặp và không xuất ra câu thông báo bip bip bip. Ngược lại khi nhận được tín hiệu 0 thì sau khi xuất thông báo chương trình sẽ quay về đầu vòng lặp để thực hiện tiếp tục và cũng không xuất ra câu thông báo bip bip bip.

Toán tử

Toán tử được kí hiệu bằng một biểu tượng dùng để thực hiện một hành động. Các kiểu dữ liệu cơ bản của C# như kiểu nguyên hỗ trợ rất nhiều các toán tử như toán tử gán, toán tử toán học, logic....

Toán tử gán

Đến lúc này toán tử gán khá quen thuộc với chúng ta, hầu hết các chương trình minh họa từ đầu sách đều đã sử dụng phép gán. Toán tử gán hay phép gán làm cho toán hạng bên trái thay đổi giá trị bằng với giá trị của toán hạng bên phải. Toán tử gán là toán tử hai ngôi. Đây là toán tử đơn giản nhất thông dụng nhất và cũng dễ sử dụng nhất.

Toán tử toán học

Ngôn ngữ C# cung cấp năm toán tử toán học, bao gồm bốn toán tử đầu các phép toán cơ bản. Toán tử cuối cùng là toán tử chia nguyên lấy phần dư. Chúng ta sẽ tìm hiểu chi tiết các phép toán này trong phần tiếp sau.

Các phép toán số học cơ bản (+,-,,/)*

Các phép toán này không thể thiếu trong bất cứ ngôn ngữ lập trình nào, C# cũng không ngoại lệ, các phép toán số học đơn giản nhưng rất cần thiết bao gồm: phép cộng (+), phép trừ (-), phép nhân (*), phép chia (/) nguyên và không nguyên.

Khi chia hai số nguyên, thì C# sẽ bỏ phần phân số, hay bỏ phần dư, tức là nếu ta chia 8/3 thì sẽ được kết quả là 2 và sẽ bỏ phần dư là 2, do vậy để lấy được phần dư này thì C# cung cấp thêm toán tử lấy dư sẽ được trình bày trong phần kế tiếp.

Tuy nhiên, khi chia cho số thực có kiểu như `float`, `double`, hay `decimal` thì kết quả chia được trả về là một số thực.

Phép toán chia lấy dư

Để tìm phần dư của phép chia nguyên, chúng ta sử dụng toán tử chia lấy dư (%). Ví dụ, câu lệnh sau `8%3` thì kết quả trả về là 2 (đây là phần dư còn lại của phép chia nguyên).

Thật sự phép toán chia lấy dư rất hữu dụng cho người lập trình . Khi chúng ta thực hiện một phép chia dư n cho một số khác, nếu số này là bội số của n thì kết quả của phép chia dư là 0. Ví dụ `20 % 5 = 0` vì 20 là một bội số của 5. Điều này cho phép chúng ta ứng dụng trong

vòng lặp, khi muốn thực hiện một công việc nào đó cách khoảng n lần, ta chỉ cần kiểm tra phép chia dư n, nếu kết quả bằng 0 thì thực hiện công việc. Cách sử dụng này đã áp dụng trong ví dụ minh họa sử dụng vòng lặp **for** bên trên. Ví dụ 3.17 sau minh họa sử dụng các phép toán chia trên các số nguyên, thực...

Ví dụ 3.17: Phép chia và phép chia lấy dư.

```
---------------------------------------------------------------------------
using System;
class Tester
{

        public static void Main()
        {
        int i1, i2;
        float f1, f2;
        double d1, d2;
        decimal dec1, dec2;

        i1 = 17;
        i2 = 4;
        f1 = 17f;
        f2 =  4f;
        d1 = 17;
        d2 = 4;
        dec1 = 17;
        dec2 = 4;
        Console.WriteLine("Integer: \t{0}", i1/i2);
        Console.WriteLine("Float: \t{0}", f1/f2);
        Console.WriteLine("Double: \t{0}", d1/d2);
        Console.WriteLine("Decimal: \t{0}", dec1/dec2);
        Console.WriteLine("\nModulus: : \t{0}", i1%i2);
        }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Integer: 4
float:      4.25
double:     4.25
decimal: 4.25
```

```
Modulus: 1
---------------------------------------------------------------------------
```

Toán tử tăng và giảm

Khi sử dụng các biến số ta thường có thao tác là cộng một giá trị vào biến, trừ đi một giá trị từ biến đó, hay thực hiện các tính toán thay đổi giá trị của biến sau đó gán giá trị mới vừa tính toán cho chính biến đó.

Tính toán và gán trở lại

Giả sử chúng ta có một biến tên Luong lưu giá trị lương của một người, biến Luong này có giá trị hiện thời là 1.500.000, sau đó để tăng thêm 200.000 ta có thể viết như sau:

```
Luong = Luong + 200.000;
```

Trong câu lệnh trên phép cộng được thực hiện trước, khi đó kết quả của vế phải là 1.700.000 và kết quả này sẽ được gán lại cho biến Luong, cuối cùng Luong có giá trị là 1.700.000. Chúng ta có thể thực hiện việc thay đổi giá trị rồi gán lại cho biến với bất kỳ phép toán số học nào:

```
Luong = Luong * 2;
Luong = Luong – 100.000;
…
```

Do việc tăng hay giảm giá trị của một biến rất thường xảy ra trong khi tính toán nên C# cung cấp các phép toán tự gán (self- assignment). Bảng sau liệt kê các phép toán tự gán.

Toán tử	Ý nghĩa
+=	Cộng thêm giá trị toán hạng bên phải vào giá trị toán hạng bên trái
-=	Toán hạng bên trái được trừ bớt đi một lượng bằng giá trị của toán hạng bên phải
*=	Toán hạng bên trái được nhân với một lượng bằng giá trị của toán hạng bên phải.
/=	Toán hạng bên trái được chia với một lượng bằng giá trị của toán hạng bên phải.
%=	Toán hạng bên trái được chia lấy dư với một lượng bằng giá trị của toán hạng bên phải.

Bảng 3.4: Mô tả các phép toán tự gán.

Dựa trên các phép toán tự gán trong bảng ta có thể thay thế các lệnh tăng giảm lương như sau:

```
Luong  += 200.000;
Luong  *= 2;
Luong  -=  100.000;
```

Kết quả của lệnh thứ nhất là giá trị của Luong sẽ tăng thêm 200.000, lệnh thứ hai sẽ làm cho giá trị Luong nhân đôi tức là tăng gấp 2 lần, và lệnh cuối cùng sẽ trừ bớt 100.000 của Luong.

Do việc tăng hay giảm 1 rất phổ biến trong lập trình nên C# cung cấp hai toán tử đặc biệt là tăng một (++) hay giảm một (--).

Khi đó muốn tăng đi một giá trị của biến đếm trong vòng lặp ta có thể viết như sau:

```
bienDem++;
```

Toán tử tăng giảm tiền tố và tăng giảm hậu tố

Giả sử muốn kết hợp các phép toán như gia tăng giá trị của một biến và gán giá trị của biến cho biến thứ hai, ta viết như sau:

```
var1 = var2++;
```

Câu hỏi được đặt ra là gán giá trị trước khi cộng hay gán giá trị sau khi đã cộng. Hay nói cách khác giá trị ban đầu của biến var2 là 10, sau khi thực hiện ta muốn giá trị của var1 là 10, var2 là 11, hay var1 là 11, var2 cũng 11?

Để giải quyết yêu cầu trên C# cung cấp thứ tự thực hiện phép toán tăng/giảm với phép toán gán, thứ tự này được gọi là *tiền tố* (prefix) hay *hậu tố* (postfix). Do đó ta có thể viết:

```
var1 = var2++; // Hậu tố
```

Khi lệnh này được thực hiện thì phép gán sẽ được thực hiện trước tiên, sau đó mới đến phép toán tăng. Kết quả là var1 = 10 và var2 = 11. Còn đối với trường hợp tiền tố:

```
var1 = ++var2;
```

Khi đó phép tăng sẽ được thực hiện trước tức là giá trị của biến var2 sẽ là 11 và cuối cùng phép gán được thực hiện. Kết quả cả hai biến var1 và var2 điều có giá trị là 11.

Để hiểu rõ hơn về hai phép toán này chúng ta sẽ xem ví dụ minh họa 3.18 sau

Ví dụ 3.18: Minh hoạ sử dụng toán tử tăng trước và tăng sau khi gán.

```
---------------------------------------------------------------------------
using System;
class Tester
{
    static int Main()
    {
        int valueOne = 10;
        int valueTwo;
        valueTwo = valueOne++;
        Console.WriteLine("Thuc hien tang sau: {0}, {1}",
                valueOne, valueTwo);
        valueOne = 20;
```

```
            valueTwo = ++valueOne;
            Console.WriteLine("Thuc hien tang truoc: {0}, {1}",
                    valueOne, valueTwo);
            return 0;
        }
}
```

Kết quả:

```
Thuc hien tang sau: 11, 10
Thuc hien tang truoc: 21, 21
```

Toán tử quan hệ

Những toán tử quan hệ được dùng để so sánh giữa hai giá trị, và sau đó trả về kết quả là một giá trị logic kiểu bool (true hay false). Ví dụ toán tử so sánh lớn hơn (>) trả về giá trị là true nếu giá trị bên trái của toán tử lớn hơn giá trị bên phải của toán tử. Do vậy 5 > 2 trả về một giá trị là true, trong khi 2 > 5 trả về giá trị false.

Các toán tử quan hệ trong ngôn ngữ C# được trình bày ở bảng 3.4 bên dưới. Các toán tử trong bảng được minh họa với hai biến là value1 và value2, trong đó value1 có giá trị là 100 và value2 có giá trị là 50.

Tên toán tử	Kí hiệu	Biểu thức so sánh	Kết quả so sánh
So sánh bằng	==	value1 == 100 value1 == 50	true false
Không bằng	!=	value2 != 100 value2 != 90	false true
Lớn hơn	>	value1 > value2 value2 > value1	true false
Lớn hơn hay bằng	>=	value2 >= 50	true
Nhỏ hơn	<	value1 < value2 value2 < value1	false true
Nhỏ hơn hay bằng	<=	value1 <= value2	false

Bảng 3.4: Các toán tử so sánh (giả sử value1 = 100, và value2 = 50).

Như trong bảng 3.4 trên ta lưu ý toán tử so sánh bằng (==), toán tử này được ký hiệu bởi hai dấu bằng (=) liền nhau và cùng trên một hàng , không có bất kỳ khoảng trống nào xuất hiện giữa chúng. Trình biên dịch C# xem hai dấu này như một toán tử.

Toán tử logic

Trong câu lệnh **if** mà chúng ta đã tìm hiểu trong phần trước, thì khi điều kiện là true thì biểu thức bên trong **if** mới được thực hiện. Đôi khi chúng ta muốn kết hợp nhiều điều kiện với nhau như: bắt buộc cả hai hay nhiều điều kiện phải đúng hoặc chỉ cần một trong các điều kiện đúng là đủ hoặc không có điều kiện nào đúng...C# cung cấp một tập hợp các toán tử logic để phục vụ cho người lập trình.

Bảng 3.5 liệt kệ ba phép toán logic, bảng này cũng sử dụng hai biến minh họa là x, và y trong đó x có giá trị là 5 và y có giá trị là 7.

Tên toán tử	Ký hiệu	Biểu thức logic	Giá trị	Logic
and	&&	(x == 3) && (y == 7)	false	Cả hai điều kiện phải đúng
or	||	(x == 3) || (y == 7)	true	Chỉ cần một điều kiện đúng
not	!	! (x == 3)	true	Biểu thức trong ngoặc phải sai.

Bảng 3.5: Các toán tử logic (giả sử x = 5, y = 7).

Toán tử and sẽ kiểm tra cả hai điều kiện. Trong bảng 3.5 trên có minh họa biểu thức logic sử dụng toán tử and:

```
(x == 3) && (y == 7)
```

Toàn bộ biểu thức được xác định là sai vì có điều kiện (x == 3) là sai.

Với toán tử or, thì một hay cả hai điều kiện đúng thì đúng, biểu thức sẽ có giá trị là sai khi cả hai điều kiện sai. Do vậy ta xem biểu thức minh họa toán tử or:

```
(x == 3) || (y == 7)
```

Biểu thức này được xác định giá trị là đúng do có một điều kiện đúng là (y == 7) là đúng.

Đối với toán tử not, biểu thức sẽ có giá trị đúng khi điều kiện trong ngoặc là sai, và ngược lại, do đó biểu thức:

```
!( x == 3)
```

có giá trị là đúng vì điều kiện trong ngoặc tức là (x == 3) là sai.

Như chúng ta đã biết đối với phép toán logic and thì chỉ cần một điều kiện trong biểu thức sai là toàn bộ biểu thức là sai, do vậy thật là dư thừa khi kiểm tra các điều kiện còn lại một khi có một điều kiện đã sai. Giả sử ta có đoạn chương trình sau:

```
int x = 8;
if ((x == 5) && (y == 10))
```

Khi đó biểu thức **if** sẽ đúng khi cả hai biểu thức con là (x == 5) và (y == 10) đúng. Tuy nhiên khi xét biểu thức thứ nhất do giá trị x là 8 nên biểu thức (x == 5) là sai. Khi đó không cần thiết để xác định giá trị của biểu thức còn lại, tức là với bất kỳ giá trị nào của biểu thức (y == 10) thì toàn bộ biểu thức điều kiện **if** vẫn sai.

Tương tự với biểu thức logic or, khi xác định được một biểu thức con đúng thì không cần phải xác định các biểu thức con còn lại, vì toán tử logic or chỉ cần một điều kiện đúng là đủ:

```
int x =8;
if ( (x == 8) || (y == 10))
```

Khi kiểm tra biểu thức (x == 8) có giá trị là đúng, thì không cần phải xác định giá trị của biểu thức (y == 10) nữa.

Ngôn ngữ lập trình C# sử dụng logic như chúng ta đã thảo luận bên trên để loại bỏ các tính toán so sánh dư thừa và cũng không logic nữa!

Độ ưu tiên toán tử

Trình biên dịch phải xác định thứ tự thực hiện các toán tử trong trường hợp một biểu thức có nhiều phép toán, giả sử, có biểu thức sau:

```
var1 = 5+7*3;
```

Biểu thức trên có ba phép toán để thực hiện bao gồm (=, +,*). Ta thử xét các phép toán theo thứ tự từ trái sang phải, đầu tiên là gán giá trị 5 cho biến var1, sau đó cộng 7 vào 5 là 12 cuối cùng là nhân với 3, kết quả trả về là 36, điều này thật sự có vấn đề, không đúng với mục đích yêu cầu của chúng ta. Do vậy việc xây dựng một trình tự xử lý các toán tử là hết sức cần thiết. Các luật về độ ưu tiên xử lý sẽ bảo trình biên dịch biết được toán tử nào được thực hiện trước trong biểu thức.Tương tự như trong phép toán đại số thì phép nhân có độ ưu tiên thực hiện trước phép toán cộng, do vậy 5+7*3 cho kết quả là 26 đúng hơn kết quả 36. Và cả hai phép toán cộng và phép toán nhân điều có độ ưu tiên cao hơn phép gán. Như vậy trình biên dịch sẽ thực hiện các phép toán rồi sau đó thực hiện phép gán ở bước cuối cùng. Kết quả đúng của câu lệnh trên là biến var1 sẽ nhận giá trị là 26.

Trong ngôn ngữ C#, dấu ngoặc được sử dụng để thay đổi thứ tự xử lý, điều này cũng giống trong tính toán đại số. Khi đó muốn kết quả 36 cho biến var1 có thể viết:

```
var1 = (5+7) * 3;
```

Biểu thức trong ngoặc sẽ được xử lý trước và sau khi có kết quả là 12 thì phép nhân được thực hiện.

Bảng 3.6: Liệt kê thứ tự độ ưu tiên các phép toán trong C#.

STT	Loại toán tử	Toán tử	Thứ tự
1	Phép toán cơ bản	(x) x.y f(x) a[x] x++ x—new typeof sizeof checked unchecked	Trái
2		+ - ! ~ ++x –x (T)x	Trái
3	Phép nhân	* / %	Trái
4	Phép cộng	+ -	Trái
5	Dịch bit	<< >>	Trái
6	Quan hệ	< > <= >= is	Trái

7	So sánh bằng	== !=	Phải
8	Phép toán logic AND	&	Trái
9	Phép toán logic XOR	^	Trái
10	Phép toán logic OR	|	Trái
11	Điều kiện AND	&&	Trái
12	Điều kiện OR	||	Trái
13	Điều kiện	?:	Phải
14	Phép gán	= *= /= %= += -= <<= >>= &= ^= |=	Phải

Bảng 3.6: Thứ tự ưu tiên các toán tử.

Các phép toán được liệt kê cùng loại sẽ có thứ tự theo mục thứ thự của bảng: thứ tự trái tức là độ ưu tiên của các phép toán từ bên trái sang, thứ tự phải thì các phép toán có độ ưu tiên từ bên phải qua trái. Các toán tử khác loại thì có độ ưu tiên từ trên xuống dưới, do vậy các toán tử loại cơ bản sẽ có độ ưu tiên cao nhất và phép toán gán sẽ có độ ưu tiên thấp nhất trong các toán tử.

Toán tử ba ngôi

Hầu hết các toán tử đòi hỏi có một toán hạng như toán tử (++, --) hay hai toán hạng như (+,-,*,/,...). Tuy nhiên, C# còn cung cấp thêm một toán tử có ba toán hạng (?:). Toán tử này có cú pháp sử dụng như sau:

<Biểu thức điều kiện > ? <Biểu thức thứ 1> : <Biểu thức thứ 2>

Toán tử này sẽ xác định giá trị của một biểu thức điều kiện, và biểu thức điều kiện này phải trả về một giá trị kiểu bool. Khi điều kiện đúng thì <biểu thức thứ 1> sẽ được thực hiện, còn ngược lại điều kiện sai thì <biểu thức thứ 2> sẽ được thực hiện. Có thể diễn giải theo ngôn ngữ tự nhiên thì toán tử này có ý nghĩa : "*Nếu điều kiện đúng thì làm công việc thứ nhất, còn ngược lại điều kiện sai thì làm công việc thứ hai*". Cách sử dụng toán tử ba ngôi này được minh họa trong ví dụ 3.19 sau.

Ví dụ 3.19: Sử dụng toán tử bao ngôi.

```
---------------------------------------------------------------------------
using System;
class Tester
{

    public static int Main()
    {
        int value1;
```

```
            int value2;
            int maxValue;
            value1 = 10;
            value2 = 20;
            maxValue = value1 > value2 ? value1 : value2;
            Console.WriteLine("Gia tri thu nhat {0}, gia tri thu hai {1},
                        gia tri lon nhat {2}", value1, value2, maxValue);
            return 0;
        }
}
```

Kết quả:

```
Gia tri thu nhat 10, gia tri thu hai 20, gia tri lon nhat 20
```

Trong ví dụ minh họa trên toán tử ba ngôi được sử dụng để kiểm tra xem giá trị của value1 có lớn hơn giá trị của value2, nếu đúng thì trả về giá trị của value1, tức là gán giá trị value1 cho biến maxValue, còn ngược lại thì gán giá trị value2 cho biến maxValue.

Namespace

Chương 2 đã thảo luận việc sử dụng đặc tính *namespace* trong ngôn ngữ C#, nhằm tránh sự xung đột giữa việc sử dụng các thư viện khác nhau từ các nhà cung cấp. Ngoài ra, namespace được xem như là tập hợp các lớp đối tượng, và cung cấp duy nhất các định danh cho các kiểu dữ liệu và được đặt trong một cấu trúc phân cấp. Việc sử dụng namespace trong khi lập trình là một thói quen tốt, bởi vì công việc này chính là cách lưu các mã nguồn để sử dụng về sau. Ngoài thư viện namespace do MS.NET và các hãng thứ ba cung cấp, ta có thể tạo riêng cho mình các namespace. C# đưa ra từ khóa **using** đề khai báo sử dụng namespace trong chương trình:

```
using < Tên namespace >
```

Để tạo một namespace dùng cú pháp sau:

```
namespace <Tên namespace>
{
    < Định nghĩa lớp A>
    < Định nghĩa lớp B >
    .....
}
```

Đoạn ví dụ 3.20 minh họa việc tạo một namespace.

Ví dụ 3.20: Tạo một namespace.

```
namespace MyLib
{
        using System;
        public class Tester
        {
            public static int Main()
            {
                for (int i =0; i < 10; i++)
                {
                    Console.WriteLine( "i: {0}", i);
                }
                return 0;
            }
        }
}
```

Ví dụ trên tạo ra một namespace có tên là MyLib, bên trong namespace này chứa một lớp có tên là Tester. C# cho phép trong một namespace có thể tạo một namespace khác lồng bên trong và không giới hạn mức độ phân cấp này, việc phân cấp này được minh họa trong ví dụ 3.21.

Ví dụ 3.21: Tạo các namespace lồng nhau.

```
namespace MyLib
{
        namespace Demo
        {
            using System;
            public class Tester
            {
                public static int Main()
                {
                    for (int i =0; i < 10; i++)
                    {
                        Console.WriteLine( "i: {0}", i);
                    }
                    return 0;
                }
            }
```

```
    }
}
```

Lớp Tester trong ví dụ 3.21 được đặt trong namespace Demo do đó có thể tạo một lớp Tester khác bên ngoài namespace Demo hay bên ngoài namespace MyLib mà không có bất cứ sự tranh cấp hay xung đột nào. Để truy cập lớp Tester dùng cú pháp sau:

```
MyLib.Demo.Tester
```

Trong một namespace một lớp có thể gọi một lớp khác thuộc các cấp namespace khác nhau, ví dụ tiếp sau minh họa việc gọi một hàm thuộc một lớp trong namespace khác.

Ví dụ 3.22: Gọi một namespace thành viên.

```
using System;
namespace MyLib
{
    namespace Demo1
    {
        class Example1
        {
            public static void Show1()
            {
                Console.WriteLine("Lop Example1");
            }
        }
    }
    namespace Demo2
    {
        public class Tester
        {
            public static int Main()
            {
                Demo1.Example1.Show1();
                Demo1.Example2.Show2();
                return 0;

            }
        }
    }
}
// Lớp Example2 có cùng namespace MyLib.Demo1 với
```

```
//lớp Example1 nhưng hai khai báo không cùng một khối.
namespace MyLib.Demo1
{
        class Example2
        {
            public static void Show2()
            {
                Console.WriteLine("Lop Example2");
            }
        }
}
```

Kết quả:

```
Lop Exemple1
Lop Exemple2
```

Ví dụ 3.22 trên có hai điểm cần lưu ý là cách gọi một namespace thành viên và cách khai báo các namspace. Như chúng ta thấy trong namespace MyLib có hai namespace con cùng cấp là Demo1 và Demo2, hàm Main của Demo2 sẽ được chương trình thực hiện, và trong hàm Main này có gọi hai hàm thành viên tĩnh của hai lớp Example1 và Example2 của namespace Demo1.

Ví dụ trên cũng đưa ra cách khai báo khác các lớp trong namespace. Hai lớp Example1 và Example2 điều cùng thuộc một namespace MyLib.Demo1, tuy nhiên Example2 được khai báo một khối riêng lẻ bằng cách sử dụng khai báo:

```
namespace MyLib.Demo1
{
        class Example2
        {
            ....
        }
}
```

Việc khai báo riêng lẻ này có thể cho phép trên nhiều tập tin nguồn khác nhau, miễn sao đảm bảo khai báo đúng tên namspace thì chúng vẫn thuộc về cùng một namespace.

Các chỉ dẫn biên dịch

Đối với các ví dụ minh họa trong các phần trước, khi biên dịch thì toàn bộ chương trình sẽ được biên dịch. Tuy nhiên, có yêu cầu thực tế là chúng ta chỉ muốn một phần trong

chương trình được biên dịch độc lập, ví dụ như khi debug chương trình hoặc xây dựng các ứng dụng...

Trước khi một mã nguồn được biên dịch, một chương trình khác được gọi là chương trình tiền xử lý sẽ thực hiện trước và chuẩn bị các đoạn mã nguồn để biên dịch. Chương trình tiền xử lý này sẽ tìm trong mã nguồn các kí hiệu chỉ dẫn biên dịch đặc biệt, tất cả các chỉ dẫn biên dịch này đều được bắt đầu với dấu rào (#). Các chỉ dẫn cho phép chúng ta định nghĩa các định danh và kiểm tra các sự tồn tại của các định danh đó.

Định nghĩa định danh

Câu lệnh tiền xử lý sau:

```
#define DEBUG
```

Lệnh trên định nghĩa một định danh tiền xử lý có tên là DEBUG. Mặc dù những chỉ thị tiền xử lý khác có thể được đặt bất cứ ở đâu trong chương trình, nhưng với chỉ thị định nghĩa định danh thì phải đặt trước tất cả các lệnh khác, bao gồm cả câu lệnh using.

Để kiểm tra một định danh đã được định nghĩa thì ta dùng cú pháp #if <định danh>. Do đó ta có thể viết như sau:

```
#define DEBUG
//...Các đoạn mã nguồn bình thường, không bị tác động bởi trình tiền xử lý
...
#if DEBUG
    // Các đoạn mã nguồn trong khối if debug được biên dịch
#else
    // Các đoạn mã nguồn không định nghĩa debug và không được biên dịch
#endif
//...Các đoạn mã nguồn bình thường, không bị tác động bởi trình tiền xử lý
```

Khi chương trình tiền xử lý thực hiện, chúng sẽ tìm thấy câu lệnh #define DEBUG và lưu lại định danh DEBUG này. Tiếp theo trình tiền xử lý này sẽ bỏ qua tất cả các đoạn mã bình thường khác của C# và tìm các khối #if, #else, và #endif.

Câu lệnh #if sẽ kiểm tra định danh DEBUG, do định danh này đã được định nghĩa, nên đoạn mã nguồn giữa khối #if đến #else sẽ được biên dịch vào chương trình. Còn đoạn mã nguồn giữa #else và #endif sẽ không được biên dịch. Tức là đoạn mã nguồn này sẽ không được thực hiện hay xuất hiện bên trong mã hợp ngữ của chương trình.

Trường hợp câu lệnh #if sai tức là không có định nghĩa một định danh DEBUG trong chương trình, khi đó đoạn mã nguồn ở giữa khối #if và #else sẽ không được đưa vào chương trình để biên dịch mà ngược lại đoạn mã nguồn ở giữa khối #else và #endif sẽ được biên dịch.

Lưu ý: Tất cả các đoạn mã nguồn bên ngoài #if và #endif thì không bị tác động bởi trình tiền xử lý và tất cả các mã này đều được đưa vào để biên dịch.

Không định nghĩa định danh

Sử dụng chỉ thị tiền xử lý #undef để xác định trạng thái của một định danh là không được định nghĩa. Như chúng ta đã biết trình tiền xử lý sẽ thực hiện từ trên xuống dưới, do vậy một định danh đã được khai báo bên trên với chỉ thị #define sẽ có hiệu quả đến khi một gọi câu lệnh #undef định danh đó hay đến cuối chương trình:

```
#define DEBUG
#if DEBUG
    // Đoạn code này được biên dịch
#endif
....
#undef DEBUG
....
#if DEBUG
    // Đoạn code này không được biên dịch
    #endif
.....
```

#if đầu tiên đúng do DEBUG được định nghĩa, còn #if thứ hai sai không được biên dịch vì DEBUG đã được định nghĩa lại là #undef.

Ngoài ra còn có chỉ thị #elif và #else cung cấp các chỉ dẫn phức tạp hơn. Chỉ dẫn #elif cho phép sử dụng logic "else-if". Ta có thể diễn giải một chỉ dẫn như sau: "*Nếu DEBUG thì làm công việc 1, ngược lại nếu TEST thì làm công việc 2, nếu sai tất cả thì làm trường hợp 3*":

```
....
#if DEBUG
      // Đoạn code này được biên dịch nếu DEBUG được định nghĩa
#elif TEST
      //Đoạn code này được biên dịch nếu DEBUG không được định nghĩa
      // và TEST được định nghĩa
#else
      //Đoạn code này được biên dịch nếu cả DEBUG và
       //TEST không được định nghĩa.
#endif
....
```

Trong ví dụ trên thì chỉ thị tiền xử lý #if đầu tiên sẽ kiểm tra định danh DEBUG, nếu định danh DEBUG đã được định nghĩa thì đoạn mã nguồn ở giữa #if và #elif sẽ được biên dịch, và tất cả các phần còn lại cho đến chỉ thị #endif đều không được biên dịch. Nếu DEBUG không được định nghĩa thì #elif sẽ kiểm tra định danh TEST, đoạn mã ở giữa #elif và #else sẽ được

thực thi khi TEST được định nghĩa. Cuối cùng nếu cả hai DEBUG và TEST đều không được định nghĩa thì các đoạn mã nguồn giữa #else và #endif sẽ được biên dịch.

Câu hỏi và trả lời

Câu hỏi 1: Sự khác nhau giữa dựa trên thành phần (Component-Based) và hướng đối tượng (Object- Oriented)?

Trả lời 1: Phát triển dựa trên thành phần có thể được xem như là mở rộng của lập trình hướng đối tượng. Một thành phần là một khối mã nguồn riêng có thể thực hiện một nhiệm vụ đặc biệt. Lập trình dựa trên thành phần bao gồm việc tạo nhiều các thành phần tự hoạt động có thể được dùng lại. Sau đó chúng ta có thể liên kết chúng lại để xây dựng các ứng dụng.

Câu hỏi 2: Những ngôn ngữ nào khác được xem như là hướng đối tượng?

Trả lời 2: Các ngôn ngữ như là C++, Java, SmallTalk, Visual Basic.NET cũng có thể được sử dụng cho lập trình hướng đối tượng. Còn rất nhiều những ngôn ngữ khác nhưng không được phổ biến lắm.

Câu hỏi 3: Tại sao trong kiểu số không nên khai báo kiểu dữ liệu lớn thay vì dùng kiểu dữ liệu nhỏ hơn?

Trả lời 3: Mặc dù điều có thể xem là khá hợp lý, nhưng thật sự không hiệu quả lắm. Chúng ta không nên sử dụng nhiều tài nguyên bộ nhớ hơn mức cần thiết. Khi đó vừa lãng phí bộ nhớ lại vừa hạn chế tốc độ của chương trình.

Câu hỏi 4: Chuyện gì xảy ra nếu ta gán giá trị âm vào biến kiểu không dấu?

Trả lời 4: Chúng ta sẽ nhận được lỗi của trình biên dịch nói rằng không thể gán giá trị âm cho biến không dấu trong trường hợp ta gán giá trị hằng âm. Còn nếu trong trường hợp kết quả là âm đươc tính trong biểu thức khi chạy chương trình thì chúng ta sẽ nhận được lỗi dữ liệu. Việc kiểm tra và xử lý lỗi dữ liệu sẽ đươc trình bày trong các phần sau.

Câu hỏi 5: Những ngôn ngữ nào khác hỗ trở Common Type System (CTS) trong Common Language Runtime (CLR)?

Trả lời 5: Microsoft Visual Basic (Version 7), Visual C++.NET cùng hỗ trợ CTS. Thêm vào đó là một số phiên bản của ngôn ngữ khác cũng được chuyển vào CTS. Bao gồm Python, COBOL, Perl, Java. Chúng ta có thể xem trên trang web của Microsoft để biết thêm chi tiết.

Câu hỏi 6: Có phải còn những câu lệnh điều khiển khác?

Trả lời 6: Đúng, các câu lệnh này như sau: throw, try, catch và finally. Chúng ta sẽ được học trong chương xử lý ngoại lệ.

Câu hỏi 7: Có thể sử dụng chuỗi với câu lệnh switch?

Trả lời 7: Hoàn toàn được, chúng ta sử dụng biến giá trị chuỗi trong ***switch*** *rồi sau đó dùng giá trị chuỗi trong câu lệnh case. Lưu ý là chuỗi là những ký tự đơn giản nằm giữa hai dấu ngoặc nháy.*

Câu hỏi thêm

Câu hỏi 1: *Có bao nhiêu cách khai báo comment trong ngôn ngữ C#, cho biết chi tiết?*

Câu hỏi 2: *Những từ theo sau từ nào là từ khóa trong C#: field, cast, as, object, throw, football, do, get, set, basketball.*

Câu hỏi 3: *Những khái niệm chính của ngôn ngữ lập trình hướng đối tượng?*

Câu hỏi 4: *Sự khác nhau giữa hai lệnh Write và WriteLine?*

Câu hỏi 5: *C# chia làm mấy kiểu dữ liệu chính? Nếu ta tạo một lớp tên myClass thì lớp này được xếp vào kiểu dữ liệu nào?*

Câu hỏi 6: *Kiểu chuỗi trong C# là kiểu dữ liệu nào?*

Câu hỏi 7: *Dữ liệu của biến kiểu dữ liệu tham chiếu được lưu ở đâu trong bộ nhớ?*

Câu hỏi 8: *Sự khác nhau giữa lớp và cấu trúc trong C#? Khi nào thì dùng cấu trúc tốt hơn là dùng class?*

Câu hỏi 8: *Sự khác nhau giữa kiểu unsigned và signed trong kiểu số nguyên?*

Câu hỏi 9: *Kiểu dữ liệu nào nhỏ nhất có thể lưu trữ được giá trị 45?*

Câu hỏi 10: *Số lớn nhất, và nhỏ nhất của kiểu int là số nào?*

Câu hỏi 11: *Có bao nhiêu bit trong một byte?*

Câu hỏi 12: *Kiểu dữ liệu nào trong .NET tương ứng với kiểu int trong C#?*

Câu hỏi 13: *Những từ khóa nào làm thay đổi luồng của chương trình?*

Câu hỏi 14: *Kết quả của 15%4 là bao nhiêu?*

Câu hỏi 15: *Sự khác nhau giữa chuyển đổi tường minh và chuyển đổi ngầm định?*

Câu hỏi 16: *Có thể chuyển từ một giá trị long sang giá trị int hay không?*

Câu hỏi 17: *Số lần tối thiểu các lệnh trong* ***while*** *được thực hiện?*

Câu hỏi 18: *Số lần tối thiểu các lệnh trong do* ***while*** *được thực hiện?*

Câu hỏi 19: *Lệnh nào dùng để thoát ra khỏi vòng lặp?*

Câu hỏi 20: *Lệnh nào dùng để qua vòng lặp kế tiếp?*

Câu hỏi 21: *Khi nào dùng biến và khi nào dùng hằng?*

Câu hỏi 22: *Cho biết giá trị CanhCut trong kiểu liệt kê sau:*

```
enum LoaiChim
{
    HaiAu,
    BoiCa,
    DaiBang = 50,
    CanhCut
}
```

Câu hỏi 23: *Cho biết các lệnh phân nhánh trong C#?*

Bài tập

Bài tập 1: *Nhập vào, biên dịch và chạy chương trình. Hãy cho biết chương trình làm điều gì?*

```
class BaiTap3_1
{
    public static void Main()
    {
        int x = 0;
        for(x = 1; x < 10; x++)
        {
            System.Console.Write("{0:03}", x);
        }
    }
}
```

Bài tập 2: Tìm lỗi của chương trình sau? sửa lỗi và biên dịch chương trình.

```
class BaiTap3_2
{
    public static void Main()
    {
        for(int i=0; i < 10 ; i++)
            System.Console.WriteLine("so :{1}", i);
    }
}
```

Bài tập 3: Tìm lỗi của chương trình sau. Sửa lỗi và biên dịch lại chương trình.

```
using System;
class BaiTap3_3
{
    public static void Main()
    {
        double myDouble;
        decimal myDecimal;
        myDouble = 3.14;
        myDecimal = 3.14;
        Console.WriteLine("My Double: {0}", myDouble);
        Console.WriteLine("My Decimal: {0}", myDecimal);

    }
}
```

Bài tập 4: Tìm lỗi của chương trình sau. Sửa lỗi và biên dịch lại chương trình.

```
class BaiTap3_4
{
    static void Main()
    {
        int value;
        if (value > 100);
            System.Console.WriteLine("Number is greater than 100");
    }
}
```

Bài tập 5: Viết chương trình hiển thị ra màn hình 3 kiểu sau:

```
*                $ $ $ $ $ $              *
* *               $ $ $ $ $                * * *
* * *             $ $ $ $                * * * * *
* * * *           $ $ $                * * * * * * *
* * * * *         $ $                * * * * * * * * *
* * * * * *       $                * * * * * * * * * * *
```

a) b) c)

Bài tập 6: Viết chương trình hiển ra trên màn hình.

1
2 3 2
3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
6 7 8 9 0 1 0 9 8 7 6
7 8 9 0 1 2 3 2 1 0 9 8 7
8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
9 0 1 2 3 4 5 6 7 6 5 4 3 2 1 0 9
0 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 0

Bài tập 7: *Viết chương trình in ký tự số (0..9) và ký tự chữ (a..z) với mã ký tự tương ứng của từng ký tự*

Ví dụ:

'0' : 48

'1' : 49

....

Bài tập 8: *Viết chương trình giải phương trình bậc nhất, cho phép người dùng nhập vào giá trị a, b.*

Bài tập 9: *Viết chương trình giải phương trình bậc hai, cho phép người dùng nhập vào giá trị a, b, c.*

Bài tập 10: *Viết chương trình tính chu vi và diện tích của các hình sau: đường tròn, hình chữ nhật, hình thang, tam giác.*

Chương 4

XÂY DỰNG LỚP - ĐỐI TƯỢNG

- **Định nghĩa lớp**
    - **Thuộc tính truy cập**
    - **Tham số của phương thức**
- **Tạo đối tượng**
    - **Bộ khởi dựng**
    - **Khởi tạo biến thành viên**
    - **Bộ khởi dựng sao chép**
    - **Từ khóa this**
- **Sử dụng các thành viên static**
    - **Gọi phương thức static**
    - **Sử dụng bộ khởi dựng static**
    - **Sử dụng bộ khởi dựng private**
    - **Sử dụng thuộc tính static**
- **Hủy đối tượng**
- **Truyền tham số**
- **Nạp chồng phương thức**
- **Đóng gói dữ liệu với thành phần thuộc tính**
- **Thuộc tính chỉ đọc**
- **Câu hỏi & bài tập**

Chương 3 thảo luận rất nhiều kiểu dữ liệu cơ bản của ngôn ngữ C#, như int, long and char. Tuy nhiên trái tim và linh hồn của C# là khả năng tạo ra những kiểu dữ liệu mới, phức

tạp. Người lập trình tạo ra các kiểu dữ liệu mới bằng cách xây dựng các lớp đối tượng và đó cũng chính là các vấn đề chúng ta cần thảo luận trong chương này.

Đây là khả năng để tạo ra những kiểu dữ liệu mới, một đặc tính quan trọng của ngôn ngữ lập trình hướng đối tượng. Chúng ta có thể xây dựng những kiểu dữ liệu mới trong ngôn ngữ C# bằng cách khai báo và định nghĩa những lớp. Ngoài ra ta cũng có thể định nghĩa các kiểu dữ liệu với những giao diện (interface) sẽ được bàn trong Chương 8 sau. Thể hiện của một lớp được gọi là những đối tượng (object). Những đối tượng này được tạo trong bộ nhớ khi chương trình được thực hiện.

Sự khác nhau giữa một lớp và một đối tượng cũng giống như sự khác nhau giữa khái niệm giữa loài mèo và một con mèo Mun đang nằm bên chân của ta. Chúng ta không thể đụng chạm hay đùa giỡn với khái niệm mèo nhưng có thể thực hiện điều đó được với mèo Mun, nó là một thực thể sống động, chứ không trừu tượng như khái niệm họ loài mèo.

Một họ mèo mô tả những con mèo có các đặc tính: có trọng lượng, có chiều cao, màu mắt, màu lông,...chúng cũng có hành động như là ăn ngủ, leo trèo,...một con mèo, ví dụ như mèo Mun chẳng hạn, nó cũng có trọng lượng xác định là 5 kg, chiều cao 15 cm, màu mắt đen, lông đen...Nó cũng có những khả năng như ăn ngủ leo trèo,..

Lợi ích to lớn của những lớp trong ngôn ngữ lập trình là khả năng đóng gói các thuộc tính và tính chất của một thực thể trong một khối đơn, tự có nghĩa, tự khả năng duy trì . Ví dụ khi chúng ta muốn sắp nội dung những thể hiện hay đối tượng của lớp điều khiển ListBox trên Windows, chỉ cần gọi các đối tượng này thì chúng sẽ tự sắp xếp, còn việc chúng làm ra sao thì ta không quan tâm, và cũng chỉ cần biết bấy nhiêu đó thôi.

Đóng gói cùng với đa hình (polymorphism) và kế thừa (inheritance) là các thuộc tính chính yếu của bất kỳ một ngôn ngữ lập trình hướng đối tượng nào.

Chương 4 này sẽ trình bày các đặc tính của ngôn ngữ lập trình C# để xây dựng các lớp đối tượng. Thành phần của một lớp, các hành vi và các thuộc tính, được xem như là thành viên của lớp (class member). Tiếp theo chương cũng trình này khái niệm về phương thức (method) được dùng để định nghĩa hành vi của một lớp, và trạng thái của các biến thành viên hoạt động trong một lớp. Một đặc tính mới mà ngôn ngữ C# đưa ra để xây dựng lớp là khái niệm thuộc tính (property), thành phần thuộc tính này hoạt động giống như cách phương thức để tạo một lớp, nhưng bản chất của phương thức này là tạo một lớp giao diện cho bên ngoài tương tác với biến thành viên một cách gián tiếp, ta sẽ bàn sâu vấn đề này trong chương.

Định nghĩa lớp

Để định nghĩa một kiểu dữ liệu mới hay một lớp đầu tiên phải khai báo rồi sau đó mới định nghĩa các thuộc tính và phương thức của kiểu dữ liệu đó. Khai báo một lớp bằng cách sử dụng từ khoá **class**. Cú pháp đầy đủ của khai báo một lớp như sau:

```
[Thuộc tính] [Bổ sung truy cập] class <Định danh lớp> [: Lớp cơ sở]
{
```

```
    <Phần thân của lớp: bao gồm định nghĩa các thuộc tính và
      phương thức hành động >
}
```

Thành phần thuộc tính của đối tượng sẽ được trình bày chi tiết trong chương sau, còn thành phần bổ sung truy cập cũng sẽ được trình bày tiếp ngay mục dưới. Định danh lớp chính là tên của lớp do người xây dựng chương trình tạo ra. Lớp cơ sở là lớp mà đối tượng sẽ kế thừa để phát triển ta sẽ bàn sau. Tất cả các thành viên của lớp được định nghĩa bên trong thân của lớp, phần thân này sẽ được bao bọc bởi hai dấu ({}).

Ghi chú: Trong ngôn ngữ C# phần kết thúc của lớp không có đấu chấm phẩy giống như khai báo lớp trong ngôn ngữ C/C++. Tuy nhiên nếu người lập trình thêm vào thì trình biên dịch C# vẫn chấp nhận mà không đưa ra cảnh báo lỗi.

Trong C#, mọi chuyện đều xảy ra trong một lớp. Như các ví dụ mà chúng ta đã tìm hiểu trong chương 3, các hàm điều được đưa vào trong một lớp, kể cả hàm đầu vào của chương trình (hàm Main()):

```
public class Tester
{

    public static int Main()
    {
        //....
    }
}
```

Điều cần nói ở đây là chúng ta chưa tạo bất cứ thể hiện nào của lớp, tức là tạo đối tượng cho lớp Tester. Điều gì khác nhau giữa một lớp và thể hiện của lớp? để trả lới cho câu hỏi này chúng ta bắt đầu xem xét sự khác nhau giữa kiểu dữ liệu int và một biến kiểu int . Ta có viết như sau:

```
int var1 = 10;
```

tuy nhiên ta không thể viết được

```
int = 10;
```

Ta không thể gán giá trị cho một kiểu dữ liệu, thay vào đó ta chỉ được gán dữ liệu cho một đối tượng của kiểu dữ liệu đó, trong trường hợp trên đối tượng là biến var1.

Khi chúng ta tạo một lớp mới, đó chính là việc định nghĩa các thuộc tính và hành vi của tất cả các đối tượng của lớp. Giả sử chúng ta đang lập trình để tạo các điều khiển trong các ứng dụng trên Windows, các điều khiển này giúp cho người dùng tương tác tốt với Windows, như là ListBox, TextBox, ComboBox,...Một trong những điều khiển thông dụng là ListBox, điều khiển này cung cấp một danh sách liệt kê các mục chọn và cho phép người dùng chọn các mục tin trong đó.

ListBox này cũng có các thuộc tính khác nhau nhu: chiều cao, bề dày, vị trí, và màu sắc thể hiện và các hành vi của chúng như: chúng có thể thêm bới mục tin, sắp xếp,...

Ngôn ngữ lập trình hướng đối tượng cho phép chúng ta tạo kiểu dữ liệu mới là lớp ListBox, lớp này bao bọc các thuộc tính cũng như khả năng như: các thuộc tính height, width, location, color, các phương thức hay hành vi như Add(), Remove(), Sort(),...

Chúng ta không thể gán dữ liệu cho kiểu ListBox, thay vào đó đầu tiên ta phải tạo một đối tượng cho lớp đó:

```
ListBox myListBox;
```

Một khi chúng ta đã tạo một thể hiện của lớp ListBox thì ta có thể gán dữ liệu cho thể hiện đó. Tuy nhiên đoạn lệnh trên chưa thể tạo đối tượng trong bộ nhớ được, ta sẽ bàn tiếp. Bây giờ ta sẽ tìm hiểu cách tạo một lớp và tạo các thể hiện thông qua ví dụ minh họa 4.1. Ví dụ này tạo một lớp có chức năng hiểu thị thời gian trong một ngày. Lớp này có hành vi thể hiện ngày, tháng, năm, giờ, phút, giây hiện hành. Để làm được điều trên thì lớp này có 6 thuộc tính hay còn gọi là biến thành viên, cùng với một phương thức như sau:

Ví dụ 4.1: Tạo một lớp Thoigian đơn giản như sau.

```
---------------------------------------------------------------------------
using System;
public class ThoiGian
{
    public void ThoiGianHienHanh()
    {
        Console.WriteLine("Hien thi thoi gian hien hanh");
    }
    // Các biến thành viên
    int Nam;
    int Thang;
    int Ngay;
    int Gio;
    int Phut;
    int Giay;

}
public class Tester
{
    static void Main()
    {
        ThoiGian t = new ThoiGian();
        t.ThoiGianHienHanh();
    }
}
```

Kết quả:

```
Hien thi thoi gian hien hanh
--------------------------------------------------------------------------
```

Lớp ThoiGian chỉ có một phương thức chính là hàm ThoiGianHienHanh(), phần thân của phương thức này được định nghĩa bên trong của lớp ThoiGian. Điều này khác với ngôn ngữ C++, C# không đòi hỏi phải khai báo trước khi định nghĩa một phương thức, và cũng không hỗ trợ việc khai báo phương thức trong một tập tin và sau đó định nghĩa ở một tập tin khác. C# không có các tập tin tiêu đề, do vậy tất cả các phương thức được định nghĩa hoàn toàn bên trong của lớp. Phần cuối của định nghĩa lớp là phần khai báo các biến thành viên: Nam, Thang, Ngay, Gio, Phut, va Giay.

Sau khi định nghĩa xong lớp ThoiGian, thì tiếp theo là phần định nghĩa lớp Tester, lớp này có chứa một hàm khá thân thiện với chúng ta là hàm Main(). Bên trong hàm Main có một thể hiện của lớp ThoiGian được tạo ra và gán giá trị cho đối tượng t. Bởi vì t là thể hiện của đối tượng ThoiGian, nên hàm Main() có thể sử dụng phương thức của t:

```
t.ThoiGianHienHanh();
```

Thuộc tính truy cập

Thuộc tính truy cập quyết định khả năng các phương thức của lớp bao gồm việc các phương thức của lớp khác có thể nhìn thấy và sử dụng các biến thành viên hay những phương thức bên trong lớp. Bảng 4.1 tóm tắt các thuộc tính truy cập của một lớp trong C#.

Thuộc tính	Giới hạn truy cập
public	Không hạn chế. Những thành viên được đánh dấu public có thể được dùng bởi bất kì các phương thức của lớp bao gồm những lớp khác.
private	Thành viên trong một lớp A được đánh dấu là **private** thì chỉ được truy cập bởi các phương thức của lớp A.
protected	Thành viên trong lớp A được đánh dấu là **protected** thì chỉ được các phương thức bên trong lớp A và những phương thức dẫn xuất từ lớp A truy cập.
internal	Thành viên trong lớp A được đánh dấu là **internal** thì được truy cập bởi những phương thức của bất cứ lớp nào trong cùng khối hợp ngữ với A.
protected internal	Thành viên trong lớp A được đánh dấu là **protected internal** được truy cập bởi các phương thức của lớp A, các phương thức của lớp dẫn xuất của A, và bất cứ lớp nào trong cùng khối hợp ngữ của A.

Bảng 4.1: Thuộc tính truy cập.

Mong muốn chung là thiết kế các biến thành viên của lớp ở thuộc tính **private**. Khi đó chỉ có phương thức thành viên của lớp truy cập được giá trị của biến. C# xem thuộc tính **private** là mặc định nên trong ví dụ 4.1 ta không khai báo thuộc tính truy cập cho 6 biến nên mặc định chúng là **private**:

```
// Các biến thành viên private
int Nam;
int Thang;
int Ngay;
int Gio;
int Phut;
int Giay;
```

Do lớp Tester và phương thức thành viên ThoiGianHienHanh của lớp ThoiGian được khai báo là public nên bất kỳ lớp nào cũng có thể truy cập được.

Ghi chú: Thói quen lập trình tốt là khai báo tường minh các thuộc tính truy cập của biến thành viên hay các phương thức trong một lớp. Mặc dù chúng ta biết chắc chắn rằng các thành viên của lớp là được khai báo **private** mặc định. Việc khai báo tường minh này sẽ làm cho chương trình dễ hiểu, rõ ràng và tự nhiên hơn.

Tham số của phương thức

Trong các ngôn ngữ lập trình thì tham số và đối mục được xem là như nhau, cũng tương tự khi đang nói về ngôn ngữ hướng đối tượng thì ta gọi một hàm là một phương thức hay hành vi. Tất cả các tên này điều tương đồng với nhau.

Một phương thức có thể lấy bất kỳ số lượng tham số nào, Các tham số này theo sau bởi tên của phương thức và được bao bọc bên trong dấu ngoặc tròn (). Mỗi tham số phải khai báo kèm với kiểu dữ liệu. ví dụ ta có một khai báo định nghĩa một phương thức có tên là Method, phương thức không trả về giá trị nào cả (khai báo giá trị trả về là void), và có hai tham số là một kiểu int và button:

```
void  Method( int param1, button param2)
{
    //...
}
```

Bên trong thân của phương thức, các tham số này được xem như những biến cục bộ, giống như là ta khai báo biến bên trong phương thức và khởi tạo giá trị bằng giá trị của tham số truyền vào. Ví dụ 4.2 minh họa việc truyền tham số vào một phương thức, trong trường hợp này thì hai tham số của kiểu là int và float.

Ví dụ 4.2: Truyền tham số cho phương thức.

```
using System;
public class Class1
{
    public void SomeMethod(int p1, float p2)
    {
        Console.WriteLine("Ham nhan duoc hai tham so: {0} va {1}",
            p1,p2);
    }
}
public class Tester
{
    static void Main()
    {
        int var1 = 5;
        float var2 = 10.5f;
        Class1 c = new Class1();
        c.SomeMethod( var1, var2 );
    }
}
```

Kết quả:

```
Ham nhan duoc hai tham so: 5 va 10.5
```

Phương thức SomeMethod sẽ lấy hai tham số int và float rồi hiển thị chúng ta màn hình bằng việc dùng hàm Console.WriteLine(). Những tham số này có tên là p1 và p2 được xem như là biến cục bộ bên trong của phương thức.

Trong phương thức gọi Main, có hai biến cục bộ được tạo ra là var1 và var2. Khi hai biến này được truyền cho phương thức SomeMethod thì chúng được ánh xạ thành hai tham số p1 và p2 theo thứ tự danh sách biến đưa vào.

Tạo đối tượng

Trong Chương 3 có đề cập đến sự khác nhau giữa kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu. Những kiểu dữ liệu chuẩn của C# như int, char, float,… là những kiểu dữ liệu giá trị, và các biến được tạo ra từ các kiểu dữ liệu này được lưu trên stack. Tuy nhiên, với các đối tượng kiểu dữ liệu tham chiếu thì được tạo ra trên heap, sử dụng từ khóa **new** để tạo một đối tượng:

```
ThoiGian t = new ThoiGian();
```

t thật sự không chứa giá trị của đối tượng ThoiGian, nó chỉ chứa địa chỉ của đối tượng được tạo ra trên heap, do vậy t chỉ chứa tham chiếu đến một đối tượng mà thôi.

Bộ khởi dựng

Thử xem lại ví dụ minh họa 4.1, câu lệnh tạo một đối tượng cho lớp ThoiGian tương tự như việc gọi thực hiện một phương thức:

```
ThoiGian t = new ThoiGian();
```

Đúng như vậy, một phương thức sẽ được gọi thực hiện khi chúng ta tạo một đối tượng. Phương thức này được gọi là bộ khởi dựng (constructor). Các phương thức này được định nghĩa khi xây dựng lớp, nếu ta không tạo ra thì CLR sẽ thay mặt chúng ta mà tạo phương thức khởi dựng một cách mặc định. Chức năng của bộ khởi dựng là tạo ra đối tượng được xác định bởi một lớp và đặt trạng thái này hợp lệ. Trước khi bộ khởi dựng được thực hiện thì đối tượng chưa được cấp phát trong bộ nhớ. Sau khi bộ khởi dựng thực hiện hoàn thành thì bộ nhớ sẽ lưu giữ một thể hiện hợp lệ của lớp vừa khai báo.

Lớp ThoiGian trong ví dụ 4.1 không định nghĩa bộ khởi dựng. Do không định nghĩa nên trình biên dịch sẽ cung cấp một bộ khởi dựng cho chúng ta. Phương thức khởi dựng mặc định được tạo ra cho một đối tượng sẽ không thực hiện bất cứ hành động nào, tức là bên trong thân của phương thức rỗng. Các biến thành viên được khởi tạo các giá trị tầm thường như thuộc tính nguyên có giá trị là 0 và chuỗi thì khởi tạo rỗng,..Bảng 4.2 sau tóm tắt các giá trị mặc định được gán cho các kiểu dữ liệu cơ bản.

Kiểu dữ liệu	**Giá trị mặc định**
int, long, byte,…	0
bool	false
char	'\0' (null)
enum	0
reference	null

Bảng 4.2: Giá trị mặc định của kiểu dữ liệu cơ bản.

Thường thường, khi muốn định nghĩa một phương thức khởi dựng riêng ta phải cung cấp các tham số để hàm khởi dựng có thể khởi tạo các giá trị khác ngoài giá trị mặc định cho các đối tượng. Quay lại ví dụ 4.1 giả sử ta muốn truyền thời gian hiện hành: năm, tháng, ngày,…để đối tượng có ý nghĩa hơn.

Để định nghĩa một bộ khởi dựng riêng ta phải khai báo một phương thức có tên giống như tên lớp đã khai báo. Phương thức khởi dựng không có giá trị trả về và được khai báo là public. Nếu phương thức khởi dựng này được truyền tham số thì phải khai báo danh sách tham số giống như khai báo với bất kỳ phương thức nào trong một lớp. Ví dụ 4.3 được viết lại từ ví dụ 4.1 và thêm một bộ khởi dựng riêng, phương phức khởi dựng này sẽ nhận một tham số là một đối tượng kiểu DateTime do C# cung cấp.

Ví dụ 4.3: Định nghĩa một bộ khởi dựng.

```
---------------------------------------------------------------------------
using System;
public class ThoiGian
{
    public void ThoiGianHienHanh()
    {
        Console.WriteLine(" Thoi gian hien hanh la : {0}/{1}/{2}
            {3}:{4}:{5}", Ngay, Thang, Nam, Gio, Phut, Giay);
    }
    // Hàm khởi dựng
    public ThoiGian( System.DateTime dt )
    {
        Nam = dt.Year;
        Thang = dt.Month;
        Ngay = dt.Day;
        Gio = dt.Hour;
        Phut = dt.Minute;
        Giay = dt.Second;

    }
    // Biến thành viên private
    int Nam;
    int Thang;
    int Ngay;
    int Gio;
    int Phut;
    int Giay;
}
public class Tester
{
    static void Main()
    {
        System.DateTime  currentTime = System.DateTime.Now;
        ThoiGian t = new ThoiGian( currentTime );
        t.ThoiGianHienHanh();

    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Thoi gian hien hanh la: 5/6/2002  9:10:20
--------------------------------------------------------------------------
```

Trong ví dụ trên phương thức khởi dựng lấy một đối tượng DateTime và khởi tạo tất cả các biến thành viên dựa trên giá trị của đối tượng này. Khi phương thức này thực hiện xong, một đối tượng ThoiGian được tạo ra và các biến của đối tượng cũng đã được khởi tạo. Hàm ThoiGianHienHanh được gọi trong hàm Main() sẽ hiển thị giá trị thời gian lúc đối tượng được tạo ra.

Chúng ta thử bỏ một số lệnh khởi tạo trong phương thức khởi dựng và cho thực hiện chương trình lại thì các biến không được khởi tạo sẽ có giá trị mặc định là 0, do là biến nguyên. Một biến thành viên kiểu nguyên sẽ được thiết lập giá trị là 0 nếu chúng ta không gán nó trong phương thức khởi dựng. Chú ý rằng kiểu dữ liệu giá trị không thể không được khởi tạo, nếu ta không khởi tạo thì trình biên dịch sẽ cung cấp các giá trị mặc định theo bảng 4.2.

Ngoài ra trong chương trình 4.3 trên có sử dụng đối tượng của lớp DateTime, lớp DateTime này được cung cấp bởi thư viện System, lớp này cũng cung cấp các biến thành viên public như: Year, Month, Day, Hour, Minute, và Second tương tự như lớp ThoiGian của chúng ta. Thêm vào đó là lớp này có đưa ra một phương thức thành viên tĩnh tên là Now, phương thức Now sẽ trả về một tham chiếu đến một thể hiện của một đối tượng DateTime được khởi tạo với thời gian hiện hành.

Theo như trên khi lệnh :

```
System.DataTime  currentTime = System.DateTime.Now();
```

được thực hiện thì phương thức tĩnh Now() sẽ tạo ra một đối tượng DateTime trên bộ nhớ heap và trả về một tham chiếu và tham chiếu này được gán cho biến đối tượng currentTime.

Sau khi đối tượng currentTime được tạo thì câu lệnh tiếp theo sẽ thực hiện việc truyền đối tượng currentTime cho phương thức khởi dựng để tạo một đối tượng ThoiGian:

```
ThoiGian t = new ThoiGian( currentTime );
```

Bên trong phương thức khởi dựng này tham số dt sẽ tham chiếu đến đối tượng DateTime là đối tượng vừa tạo mà currentTime cũng tham chiếu. Nói cách khác lúc này tham số dt và currentTime cùng tham chiếu đến một đối tượng DateTime trong bộ nhớ. Nhờ vậy phương thức khởi dựng ThoiGian có thể truy cập được các biến thành viên public của đối tượng DateTime được tạo trong hàm Main().

Có một sự nhấn mạnh ở đây là đối tượng DateTime được truyền cho bộ dựng ThoiGian chính là đối tượng đã được tạo trong hàm Main và là kiểu dữ liệu tham chiếu. Do vậy khi thực hiện truyền tham số là một kiểu dữ liệu tham chiếu thì con trỏ được ánh xạ qua chứ hoàn toàn không có một đối tượng nào được sao chép lại.

Khởi tạo biến thành viên

Các biến thành viên có thể được khởi tạo trực tiếp khi khai báo trong quá trình khởi tạo, thay vì phải thực hiện việc khởi tạo các biến trong bộ khởi dựng. Để thực hiện việc khởi tạo này rất đơn giản là việc sử dụng phép gán giá trị cho một biến:

```
private int Giay = 30;     // Khởi tạo
```

Việc khởi tạo biến thành viên sẽ rất có ý nghĩa, vì khi xác định giá trị khởi tạo như vậy thì biến sẽ không nhận giá trị mặc định mà trình biên dịch cung cấp. Khi đó nếu các biến này không được gán lại trong các phương thức khởi dựng thì nó sẽ có giá trị mà ta đã khởi tạo. Ví dụ 4.4 minh họa việc khởi tạo biến thành viên khi khai báo. Trong ví dụ này sẽ có hai bộ dựng ngoài bộ dựng mặc định mà trình biên dịch cung cấp, một bộ dựng thực hiện việc gán giá trị cho tất cả các biến thành viên, còn bộ dựng thứ hai thì cũng tương tự nhưng sẽ không gán giá trị cho biến Giay.

Ví dụ 4.4: Minh hoạ sử dụng khởi tạo biến thành viên.

```
---------------------------------------------------------------------------
public class ThoiGian
{
    public void ThoiGianHienHanh()
    {
        System.DateTime now = System.DateTime.Now;
        System.Console.WriteLine("\n Hien tai: \t {0}/{1}/{2}  {3}:{4}:{5}",
            now.Day, now.Month, now.Year, now.Hour, now.Minute, now.Second);
        System.Console.WriteLine(" Thoi Gian:\t {0}/{1}/{2}  {3}:{4}:{5}",
            Ngay, Thang, Nam, Gio, Phut, Giay);
    }
    public ThoiGian( System.DateTime dt)
    {
        Nam = dt.Year;
        Thang = dt.Month;
        Ngay = dt.Day;
        Gio = dt.Hour;
        Phut = dt.Minute;
        Giay = dt.Second;    // có gán cho biến thành viên Giay
    }
    public ThoiGian(int Year, int Month, int Date, int Hour, int Minute)
    {
        Nam = Year;
        Thang = Month;
        Ngay = Date;
```

```
        Gio = Hour;
        Phut = Minute;

    }

    private int Nam;
    private int Thang;
    private int Ngay;
    private int Gio;
    private int Phut;

    private int Giay = 30 ; // biến được khởi tạo.


}
public class Tester
{
    static void Main()
    {

        System.DateTime currentTime = System.DateTime.Now;
        ThoiGian t1 = new ThoiGian( currentTime );
        t1.ThoiGianHienHanh();

        ThoiGian t2 = new ThoiGian(2001,7,3,10,5);

        t2.ThoiGianHienHanh();


    }
}
```

Kết quả:

```
Hien tai:   5/6/2002   10:15:5
Thoi Gian:  5/6/2002   10:15:5


Hien tai:   5/6/2002   10:15:5
Thoi Gian:  3/7/2001   10:5:30
```

Nếu không khởi tạo giá trị của biến thành viên thì bộ khởi dựng mặc định sẽ khởi tạo giá trị là 0 mặc định cho biến thành viên có kiểu nguyên. Tuy nhiên, trong trường hợp này biến thành viên Giay được khởi tạo giá trị 30:

```
Giay = 30;      // Khởi tạo
```

Trong trường hợp bộ khởi tạo thứ hai không truyền giá trị cho biến Giay nên biến này vẫn lấy giá trị mà ta đã khởi tạo ban đầu là 30:

```
ThoiGian t2 = new ThoiGian(2001, 7, 3, 10, 5);
t2.ThoiGianHienHanh();
```

Ngược lại, nếu một giá trị được gán cho biến Giay như trong bộ khởi tạo thứ nhất thì giá trị mới này sẽ được chồng lên giá trị khởi tạo.

Trong ví dụ trên lần đầu tiên tạo đối tượng ThoiGian do ta truyền vào đối tượng DateTime nên hàm khởi dựng thứ nhất được thực hiện, hàm này sẽ gán giá trị 5 cho biến Giay. Còn khi tạo đối tượng ThoiGian thứ hai, hàm khởi dựng thứ hai được thực hiện, hàm này không gán giá trị cho biến Giay nên biến này vẫn còn lưu giữ lại giá trị 30 khi khởi tạo ban đầu.

Bộ khởi dựng sao chép

Bộ khởi dựng sao chép thực hiện việc tạo một đối tượng mới bằng cách sao chép tất cả các biến từ một đối tượng đã có và cùng một kiểu dữ liệu. Ví dụ chúng ta muốn đưa một đối tượng ThoiGian vào bộ khởi dựng lớp ThoiGian để tạo một đối tượng ThoiGian mới có cùng giá trị với đối tượng ThoiGian cũ. Hai đối tượng này hoàn toàn khác nhau và chỉ giống nhau ở giá trị biến thành viên sao khi khởi dựng.

Ngôn ngữ C# không cung cấp bộ khởi dựng sao chép, do đó chúng ta phải tự tạo ra. Việc sao chép các thành phần từ một đối tượng ban đầu cho một đối tượng mới như sau:

```
public ThoiGian( ThoiGian tg)
{

    Nam = tg.Nam;
    Thang = tg.Thang;
    Ngay = tg.Ngay;
    Gio = tg.Gio;
    Phut = tg.Phut;
    Giay = tg.Giay;


}
```

Khi đó ta có thể sao chép từ một đối tượng ThoiGian đã hiện hữu như sau:

```
ThoiGian t2 = new ThoiGian( t1 );
```

Trong đó t1 là đối tượng ThoiGian đã tồn tại, sau khi lệnh trên thực hiện xong thì đối tượng t2 được tạo ra như bản sao của đối tượng t1.

Từ khóa this

Từ khóa **this** được dùng để tham chiếu đến thể hiện hiện hành của một đối tượng. Tham chiếu **this** này được xem là con trỏ ẩn đến tất các phương thức không có thuộc tính tĩnh trong một lớp. Mỗi phương thức có thể tham chiếu đến những phương thức khác và các biến thành viên thông qua tham chiếu **this** này.

Tham chiếu **this** này được sử dụng thường xuyên theo ba cách:

Sử dụng khi các biến thành viên bị che lấp bởi tham số đưa vào, như trường hợp sau:

```
public void SetYear( int Nam)
{
    this.Nam = Nam;
```

```
}
```

Như trong đoạn mã trên phương thức SetYear sẽ thiết lập giá trị của biến thành viên Nam, tuy nhiên do tham số đưa vào có tên là Nam, trùng với biến thành viên, nên ta phải dùng tham chiếu **this** để xác định rõ các biến thành viên và tham số được truyền vào. Khi đó this.Nam chỉ đến biến thành viên của đối tượng, trong khi Nam chỉ đến tham số.

Sử dụng tham chiếu **this** để truyền đối tượng hiện hành vào một tham số của một phương thức của đối tượng khác:

```
public void Method1( OtherClass otherObject )
{
    // Sử dụng tham chiếu this đế truyền tham số là bản
    // thân đối tượng đang thực hiện.
    otherObject.SetObject( this );
}
```

Như trên cho thấy khi cần truyền một tham số là chính bản thân của đối tượng đang thực hiện thì ta bắt buộc phải dùng tham chiếu **this** để truyền.

Các thứ ba sử dụng tham chiếu **this** là *mảng chỉ mục* (indexer), phần này sẽ được trình bày chi tiết trong chương 9.

Sử dụng các thành viên tĩnh (static member)

Những thuộc tính và phương thức trong một lớp có thể là những thành viên thể hiện (instance members) hay những thành viên tĩnh (static members). Những thành viên thể hiện hay thành viên của đối tượng liên quan đến thể hiện của một kiểu dữ liệu. Trong khi thành viên tĩnh được xem như một phần của lớp. Chúng ta có thể truy cập đến thành viên tĩnh của một lớp thông qua tên lớp đã được khai báo. Ví dụ chúng ta có một lớp tên là Button và có hai thể hiện của lớp tên là btnUpdate và btnDelete. Và giả sử lớp Button này có một phương thức tĩnh là Show(). Để truy cập phương thức tĩnh này ta viết :

```
Button.Show();
```

Đúng hơn là viết:

```
btnUpdate.Show();
```

Ghi chú: Trong ngôn ngữ C# không cho phép truy cập đến các phương thức tĩnh và các biến thành viên tĩnh thông qua một thể hiện, nếu chúng ta cố làm điều đó thì trình biên dịch C# sẽ báo lỗi, điều này khác với ngôn ngữ C++.

Trong một số ngôn ngữ thì có sự phân chia giữa phương thức của lớp và các phương thức khác (toàn cục) tồn tại bên ngoài không phụ thuộc bất cứ một lớp nào. Tuy nhiên, điều này không cho phép trong C#, ngôn ngữ C# không cho phép tạo các phương thức bên ngoài của lớp, nhưng ta có thể tạo được các phương thức giống như vậy bằng cách tạo các phương thức tĩnh bên trong một lớp.

GIỚI THIỆU

Tin học là một ngành khoa học mũi nhọn phát triển hết sức nhanh chóng trong vài chục năm lại đây và ngày càng mở rộng lĩnh vực nghiên cứu, ứng dụng trong mọi mặt của đời sống xã hội.

Ngôn ngữ lập trình là một loại công cụ giúp con người thể hiện các vấn đề của thực tế lên máy tính một cách hữu hiệu. Với sự phát triển của tin học, các ngôn ngữ lập trình cũng dần tiến hoá để đáp ứng các thách thức mới của thực tế.

Khoảng cuối những năm 1960 đầu 1970 xuất hiện nhu cầu cần có các ngôn ngữ bậc cao để hỗ trợ cho những nhà tin học trong việc xây dựng các phần mềm hệ thống, hệ điều hành. Ngôn ngữ C ra đời từ đó, nó đã được phát triển tại phòng thí nghiệm Bell. Đến năm 1978, giáo trình " Ngôn ngữ lập trình C " do chính các tác giả của ngôn ngữ là Dennish Ritchie và B.W. Kernighan viết, đã được xuất bản và phổ biến rộng rãi.

C là ngôn ngữ lập trình vạn năng. Ngoài việc C được dùng để viết hệ điều hành UNIX, người ta nhanh chóng nhận ra sức mạnh của C trong việc xử lý cho các vấn đề hiện đại của tin học. C không gắn với bất kỳ một hệ điều hành hay máy nào, và mặc dầu nó đã được gọi là " ngôn ngữ lập trình hệ thống" vì nó được dùng cho việc viết hệ điều hành, nó cũng tiện lợi cho cả việc viết các chương trình xử lý số, xử lý văn bản và cơ sở dữ liệu.

Và bây giờ chúng ta đi tìm hiểu thế giới của ngôn ngữ C từ những khái niệm ban đầu cơ bản nhất.

Hà nội tháng 11 năm 1997

Nguyễn Hữu Tuấn

Chương 1

CÁC KHÁI NIỆM CƠ BẢN

1.1. Tập ký tự dùng trong ngôn ngữ C :

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Các từ lại được liên kết với nhau theo một qui tắc nào đó để tạo nên các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và thể hiện một thuật toán để giải một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau :

26 chữ cái hoa : A B C .. Z

26 chữ cái thường : a b c .. z

10 chữ số : 0 1 2 .. 9

Các ký hiệu toán học : + - * / = ( )

Ký tự gạch nối : _

Các ký tự khác : . , : ; [ ] {} ! \ & % # $ ...

Dấu cách (space) dùng để tách các từ. Ví dụ chữ VIET NAM có 8 ký tự, còn VIETNAM chỉ có 7 ký tự.

Chú ý :

Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên.

Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2+bx+c=0$, ta cần tính biệt thức Delta $\Delta= b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự Δ, vì vậy ta phải dùng ký hiệu khác để thay thế.

1.2. Từ khoá :

Từ khoá là những từ được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Bảng dưới đây liệt kê các từ khoá của TURBO C :

asm	break	case	cdecl
char	const	continue	default
do	double	else	enum
extern	far	float	for
goto	huge	if	int
interrupt	long	near	pascal
register	return	short	signed

sizeof	static	struct	switch
tipedef	union	unsigned	void
volatile	while		

Ý nghĩa và cách sử dụng của mỗi từ khoá sẽ được đề cập sau này, ở đây ta cần chú ý :

- Không được dùng các từ khoá để đặt tên cho các hằng, biến, mảng, hàm ...

- Từ khoá phải được viết bằng chữ thường, ví dụ : viết từ khoá khai báo kiểu nguyên là int chứ không phải là INT.

1.3. Tên :

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn,...

Tên được đặt theo qui tắc sau :

Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng : Option-Compiler-Source-Identifier length khi dùng TURBO C.

Ví dụ :

Các tên đúng :

a_1 delta x1 _step GAMA

Các tên sai :

3MN	Ký tự đầu tiên là số
m#2	Sử dụng ký tự #
f(x)	Sử dụng các dấu ( )
do	Trùng với từ khoá
te ta	Sử dụng dấu trắng
Y-3	Sử dụng dấu -

Chú ý :

Trong TURBO C, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. trong C, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho

hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

1.4. Kiểu dữ liệu :

Trong C sử dụng các các kiểu dữ liệu sau :

1.4.1. Kiểu ký tự (char) :

Một giá trị kiểu char chiếm 1 byte (8 bit) và biểu diễn được một ký tự thông qua bảng mã ASCII. Ví dụ :

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
B	066
a	097
b	098

Có hai kiểu dữ liệu char : kiểu signed char và unsigned char.

Kiểu	Phạm vi biểu diễn	Số ký tự	Kích thước
Char (Signed char)	-128 đến 127	256	1 byte
Unsigned char	0 đến 255	256	1 byte

Ví dụ sau minh hoạ sự khác nhau giữa hai kiểu dữ liệu trên : Xét đoạn chương trình sau :

```
char  ch1;
unsigned char ch2;
.....
ch1=200; ch2=200;
```

Khi đó thực chất :

```
ch1=-56;
ch2=200;
```

Nhưng cả ch1 và ch2 đều biểu diễn cùng một ký tự có mã 200.

Phân loại ký tự :

Có thể chia 256 ký tự làm ba nhóm :

Nhóm 1: Nhóm các ký tự điều khiển có mã từ 0 đến 31. Chẳng hạn ký tự mã 13 dùng để chuyển con trỏ về đầu dòng, ký tự 10 chuyển con trỏ xuống dòng dưới (trên cùng một cột). Các ký tự nhóm này nói chung không hiển thị ra màn hình.

Nhóm 2 : Nhóm các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể được đưa ra màn hình hoặc máy in.

Nhóm 3 : Nhóm các ký tự đồ hoạ có mã số từ 127 đến 255. Các ký tự này có thể đưa ra màn hình nhưng không in ra được (bằng các lệnh DOS).

1.4.2. Kiểu nguyên :

Trong C cho phép sử dụng số nguyên kiểu int, số nguyên dài kiểu long và số nguyên không dấu kiểu unsigned. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây :

Kiểu	Phạm vi biểu diễn	Kích thước
int	-32768 đến 32767	2 byte
unsigned int	0 đến 65535	2 byte
long	-2147483648 đến 2147483647	4 byte
unsigned long	0 đến 4294967295	4 byte

Chú ý :

Kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên.

1.4.3. Kiểu dấu phảy động :

Trong C cho phép sử dụng ba loại dữ liệu dấu phảy động, đó là float, double và long double. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây :

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
Float	3.4E-38 đến 3.4E+38	7 đến 8	4 byte
Double	1.7E-308 đến 1.7E+308	15 đến 16	8 byte
long double	3.4E-4932 đến 1.1E4932	17 đến 18	10 byte

Giải thích :

Máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ 3.4E-38 đến 3.4E+38. Các số có giá trị tuyệt đối nhỏ hơn3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

1.5. Định nghĩa kiểu bằng TYPEDEF :

1.5.1. Công dụng :

Từ khoá typedef dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ được dùng để khai báo dữ liệu sau này. Nên chọn tên kiểu ngắn và gọn để dễ nhớ. Chỉ cần thêm từ khoá typedef vào trước một khai báo ta sẽ nhận được một tên kiểu dữ liệu và có thể dùng tên này để khai báo các biến, mảng, cấu trúc, vv...

1.5.2. Cách viết :

Viết từ khoá typedef, sau đó kiểu dữ liệu (một trong các kiểu trên), rồi đến tên của kiểu.
Ví dụ câu lệnh :

```
typedef int nguyen;
```

sẽ đặt tên một kiểu int là nguyen. Sau này ta có thể dùng kiểu nguyen để khai báo các biến, các mảng int như ví dụ sau ;

```
nguyen x,y,a[10],b[20][30];
```

Tương tự cho các câu lệnh :

```
typedef float mt50[50];
```

Đặt tên một kiểu mảng thực một chiều có 50 phần tử tên là mt50.

```
typedef int m_20_30[20][30];
```

Đặt tên một kiểu mảng thực hai chiều có 20x30 phần tử tên là m_20_30.
Sau này ta sẽ dùng các kiểu trên khai báo :

```
mt50 a,b;
m_20_30 x,y;
```

1.6. Hằng :

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán.

1.6.1. Tên hằng :

Nguyên tắc đặt tên hằng ta đã xem xét trong mục 1.3.

Để đặt tên một hằng, ta dùng dòng lệnh sau :

#define tên hằng giá trị

Ví dụ :

#define MAX 1000

Lúc này, tất cả các tên MAX trong chương trình xuất hiện sau này đều được thay bằng 1000. Vì vậy, ta thường gọi MAX là tên hằng, nó biểu diễn số 1000.

Một ví dụ khác :

#define pi 3.141593

Đặt tên cho một hằng float là pi có giá trị là 3.141593.

1.6.2. Các loại hằng :

1.6.2.1. Hằng int :

Hằng int là số nguyên có giá trị trong khoảng từ -32768 đến 32767.

Ví dụ :

#define number1 -50	Định nghiã hằng int number1 có giá trị là -50
#define sodem 2732	Định nghiã hằng int sodem có giá trị là 2732

Chú ý :

Cần phân biệt hai hằng 5056 và 5056.0 : ở đây 5056 là số nguyên còn 5056.0 là hằng thực.

1.6.2.2. Hằng long :

Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647.

Hằng long được viết theo cách :

1234L hoặc 1234l

(thêm L hoặc l vào đuôi)

Một số nguyên vượt ra ngoài miền xác định của int cũng được xem là long.

Ví dụ :

#define sl 8865056L	Định nghiã hằng long sl có giá trị là 8865056
#define sl 8865056	Định nghiã hằng long sl có giá trị là 8865056

1.6.2.3. Hằng int hệ 8 :

Hằng int hệ 8 được viết theo cách 0c1c2c3....Ở đây ci là một số nguyên dương trong khoảng từ 1 đến 7. Hằng int hệ 8 luôn luôn nhận giá trị dương.

Ví dụ :

#define h8 0345 Định nghiã hằng int hệ 8 có giá trị là 3*8*8+4*8+5=229

1.6.2.4. Hằng int hệ 16 :

Trong hệ này ta sử dụng 16 ký tự : 0,1..,9,A,B,C,D,E,F.

Cách viết	Giá trị
a hoặc A	10
b hoặc B	11
c hoặc C	12
d hoặc D	13
e hoặc E	14
f hoặc F	15

Hằng số hệ 16 có dạng 0xc1c2c3... hặc 0Xc1c2c3... Ở đây ci là một số trong hệ 16.

Ví dụ :

```
#define h16 0xa5
#define h16 0xA5
#define h16 0Xa5
#define h16 0XA5
```

Cho ta các hắng số h16 trong hệ 16 có giá trị như nhau. Giá trị của chúng trong hệ 10 là :

10*16+5=165.

1.6.2.5. Hằng ký tự :

Hằng ký tự là một ký tự riêng biệt được viết trong hai dấu nháy đơn, ví dụ 'a'.

Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác. Ví dụ :

'9'-'0'=57-48=9

Ví dụ :

#define kt 'a' Định nghiã hằng ký tự kt có giá trị là 97

Hằng ký tự còn có thể được viết theo cách sau :

' \c1c2c3'

trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.
Ví dụ : chữ a có mã hệ 10 là 97, đổi ra hệ 8 là 0141. Vậy hằng ký tự 'a' có thể viết dưới dạng '\141'. Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu \ ) :

Cách viết	Ký tự
'\''	'
'\"'	"
'\\'	\
'\n'	\n (chuyển dòng)
'\0'	\0 (null)
'\t'	Tab
'\b'	Backspace
'\r'	CR (về đầu dòng)
'\f'	LF (sang trang)

Chú ý :

Cần phân biệt hằng ký tự '0' và '\0'. Hằng '0' ứng với chữ số 0 có mã ASCII là 48, còn hằng '\0' ứng với kýtự \0 (thường gọi là ký tự null) có mã ASCII là 0.

Hằng ký tự thực sự là một số nguyên, vì vậy có thể dùng các số nguyên hệ 10 để biểu diễn các ký tự, ví dụ lệnh printf("%c%c",65,66) sẽ in ra AB.

1.6.2.5. Hằng xâu ký tự :

Hằng xâu ký tự là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép.

Ví dụ :

#define xau1 "Ha noi"

#define xau2 "My name is Giang"

Xâu ký tự được lưu trữ trong máy dưới dạng một bảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự null \0 vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

Chú ý :

Cần phân biệt hai hằng 'a' và "a". 'a' là hằng ký tự được lưu trữ trong 1 byte, còn "a" là hằng xâu ký tự được lưu trữ trong 1 mảng hai phần tử : phần tử thứ nhất chứa chữ a còn phần tử thứ hai chứa \0.

1.7. Biến :

Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng. Việc khai báo biến được thực hiện theo mẫu sau :

Kiểu dữ liệu của biến tên biến ;

Ví dụ :

int a,b,c;	Khai báo ba biến int là a,b,c
long dai,mn;	Khai báo hai biến long là dai và mn
char kt1,kt2;	Khai báo hai biến ký tự là kt1 và kt2
float x,y	Khai báo hai biến float là x và y
double canh1, canh2;	Khai báo hai biến double là canh1 và canh2

Biến kiểu int chỉ nhận được các giá trị kiểu int. Các biến khác cũng có ý nghĩa tương tự. Các biến kiểu char chỉ chứa được một ký tự. Để lưu trữ được một xâu ký tự cần sử dụng một mảng kiểu char.

Vị trí của khai báo biến :

Các khai báo cần phải được đặt ngay sau dấu { đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác. Sau đây là một ví dụ về khai báo biến sai :
(Khái niệm về hàm và cấu trúc chương trình sẽ nghiên cứu sau này)

```
main()
    {
```

```
        int a,b,c;
        a=2;
        int d; /* Vị trí của khai báo sai */
        .....
}
```

Khởi đầu cho biến :

Nếu trong khai báo ngay sau tên biến ta đặt dấu = và một giá trị nào đó thì đây chính là cách vừa khai báo vừa khởi đầu cho biến.

Ví dụ :

```
        int a,b=20,c,d=40;
        float e=-55.2,x=27.23,y,z,t=18.98;
```

Việc khởi đầu và việc khai báo biến rồi gán giá trị cho nó sau này là hoàn toàn tương đương.

Lấy địa chỉ của biến :

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ của biến sẽ được sử dụng trong một số hàm ta sẽ nghiên cứu sau này (ví dụ như hàm scanf).

Để lấy địa chỉ của một biến ta sử dụng phép toán :

& tên biến

1.8 Mảng :

Mỗi biến chỉ có thể biểu diễn một giá trị. Để biểu diễn một dãy số hay một bảng số ta có thể dùng nhiều biến nhưng cách này không thuận lợi. Trong trường hợp này ta có khái niệm về mảng. Khái niệm về mảng trong ngôn ngữ C cũng giống như khái niệm về ma trận trong đại số tuyến tính.

Mảng có thể được hiểu là một tập hợp nhiều phần tử có cùng một kiểu giá trị và chung một tên. Mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu biến thì có bấy nhiêu kiểu mảng. Mảng cần được khai báo để định rõ :

Loại mảng : int, float, double...

Tên mảng.

Số chiều và kích thước mỗi chiều.

Khái niệm về kiểu mảng và tên mảng cũng giống như khái niệm về kiểu biến và tên biến. Ta sẽ giải thích khái niệm về số chiều và kích thước mỗi chiều thông qua các ví dụ cụ thể dưới đây.
Các khai báo :

```
int a[10],b[4][2];
float x[5],y[3][3];
```

sẽ xác định 4 mảng và ý nghĩa của chúng như sau :

Thứ tự	Tên mảng	Kiểu mảng	Số chiều	Kích thước	Các phần tử
1	A	Int	1	10	a[0],a[1],a[2]...a[9]
2	B	Int	2	4x2	b[0][0], b[0][1] b[1][0], b[1][1] b[2][0], b[2][1] b[3][0], b[3][1]
3	X	Float	1	5	x[0],x[1],x[2]...x[4]
4	Y	Float	2	3x3	y[0][0], y[0][1], y[0][2] y[1][0], y[1][1], y[1][2] y[2][0], y[2][1], y[1][2]

Chú ý :

Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác, các phần tử của mảng có địa chỉ liên tiếp nhau.

Trong bộ nhớ, các phần tử của mảng hai chiều được sắp xếp theo hàng.

Chỉ số mảng :

Một phần tử cụ thể của mảng được xác định nhờ các chỉ số của nó. Chỉ số của mảng phải có giá trị int không vượt quá kích thước tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử z,b,x,y đã được khai báo như trên, và giả sử i,j là các biến nguyên trong đó i=2, j=1. Khi đó :

a[j+i-1]	là	a[2]
b[j+i][2-i]	là	b[3][0]
y[i][j]	là	y[2][1]

Chú ý :

Mảng có bao nhiêu chiều thì ta phải viết nó có bấy nhiêu chỉ số. Vì thế nếu ta viết như sau sẽ là sai : y[i] (Vì y là mảng 2 chiều) vv..

Biểu thức dùng làm chỉ số có thể thực. Khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng.

Ví dụ :

a[2.5] là a[2]

b[1.9] là a[1]

* Khi chỉ số vượt ra ngoài kích thước mảng, máy sẽ vẫn không báo lỗi, nhưng nó sẽ truy cập đến một vùng nhớ bên ngoài mảng và có thể làm rối loạn chương trình.

Lấy địa chỉ một phần tử của mảng :

Có một vài hạn chế trên các mảng hai chiều. Chẳng hạn có thể lấy địa chỉ của các phần tử của mảng một chiều, nhưng nói chung không cho phép lấy địa chỉ của phần tử của mảng hai chiều. Như vậy máy sẽ chấp nhận phép tính : &a[i] nhưng không chấp nhận phép tính &y[i][j].

Địa chỉ đầu của một mảng :

Tên mảng biểu thị địa chỉ đầu của mảng. Như vậy ta có thể dùng a thay cho &a[0].

Khởi đầu cho biến mảng :

Các biến mảng khai báo bên trong thân của một hàm (kể cả hàm main()) gọi là biến mảng cục bộ.

Muốn khởi đầu cho một mảng cục bộ ta sử dụng toán tử gán trong thân hàm.

Các biến mảng khai báo bên ngoài thân của một hàm gọi là biến mảng ngoài.

Để khởi đầu cho biến mảng ngoài ta áp dụng các qui tắc sau :

Các biến mảng ngoài có thể khởi đầu (một lần) vào lúc dịch chương trình bằng cách sử dụng các biểu thức hằng. Nếu không được khởi đầu máy sẽ gán cho chúng giá trị 0.

Ví dụ :

```
....
float y[6]={3.2,0,5.1,23,0,42};
```

```
int z[3][2]={
                {25,31},
                {12,13},
                {45,15}
            {
....
main()
{
        ....
}
```

Khi khởi đầu mảng ngoài có thể không cần chỉ ra kích thước (số phần tử) của nó. Khi đó, máy sẽ dành cho mảng một khoảng nhớ đủ để thu nhận danh sách giá trị khởi đầu.

Ví dụ :

```
....
float a[]={0,5.1,23,0,42};
int m[][3]={
                {25,31,4},
                {12,13,89},
                {45,15,22}
            };
```

Khi chỉ ra kích thước của mảng, thì kích thước này cần không nhỏ hơn kích thước của bộ khởi đầu.

Ví dụ :

```
....
float m[6]={0,5.1,23,0};
int z[6][3]={
                {25,31,3},
                {12,13,22},
                {45,15,11}
            };
....
```

Đối với mảng hai chiều, có thể khởi đầu với số giá trị khởi đầu của mỗi hàng có thể khác nhau :

Ví dụ :

```
....
float z[][3]={
                    {31.5},
                    {12,13},
                    {-45.76}
                };
int z[13][2]={
                    {31.11},
                    {12},
                    {45.14,15.09}
                };
```

Khởi đầu của một mảng char có thể là

Một danh sách các hằng ký tự.

Một hằng xâu ký tự.

Ví dụ :

```
char ten[]={'h','a','g'}
char ho[]='tran'
char dem[10]    ="van"
```

Chương 2

CÁC LỆNH VÀO RA

Chương này giới thiệu thư viện vào/ra chuẩn là một tập các hàm được thiết kế để cung cấp hệ thống vào/ra chuẩn cho các chương trình C. Chúng ta sẽ không mô tả toàn bộ thư viện vào ra ở đây mà chỉ quan tâm nhiều hơn đến việc nêu ra những điều cơ bản nhất để viết chương trình C tương tác với môi trường và hệ điều hành.

2.1. Thâm nhập vào thư viện chuẩn :

Mỗi tệp gốc có tham trỏ tới hàm thư viện chuẩn đều phải chứa dòng :

```
#include <conio.h>  cho các hàm getch(), putch(), clrscr(), gotoxy() ...
#include <stdio.h>  cho các hàm khác như gets(), fflus(), fwrite(), scanf()...
```

ở gần chỗ bắt đầu chương trình. Tệp stdio.h định nghĩa các macro và biến cùng các hàm dùng trong thư viện vào/ra. Dùng dấu ngoặc < và > thay cho các dấu nháy thông thường để chỉ thị cho trình biên dịch tìm kiếm tệp trong danh mục chứa thông tin tiêu đề chuẩn.

2.2. Các hàm vào ra chuẩn - getchar() và putchar() - getch() và putch() :

2.2.1. Hàm getchar () :

Cơ chế vào đơn giản nhất là đọc từng ký tự từ thiết bị vào chuẩn, nói chung là bàn phím và màn hình của người sử dụng, bằng hàm getchar().

Cách dùng :

Dùng câu lệnh sau :

```
biến = getchar();
```

Công dụng :

Nhận một ký tự vào từ bàn phím và không đưa ra màn hình. Hàm sẽ trả về ký tự nhận được và lưu vào biến.

Ví dụ :

```
int c;
c = getchar()
```

2.2.2. Hàm putchar () :

Để đưa một ký tự ra thiết bị ra chuẩn, nói chung là màn hình, ta sử dụng hàm putchar()

Cách dùng :

Dùng câu lệnh sau :

```
putchar(ch);
```

Công dụng :

Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị với màu trắng.

Ví dụ :

```
int c;
c = getchar();
putchar(c);
```

2.2.3. Hàm getch() :

Hàm nhận một ký tự từ bộ đệm bàn phím, không cho hiện lên màn hình.

Cách dùng :

Dùng câu lệnh sau :

```
getch();
```

Công dụng :

Nếu có sẵn ký tự trong bộ đệm bàn phím thì hàm sẽ nhận một ký tự trong đó.

Nếu bộ đệm rỗng, máy sẽ tạm dừng. Khi gõ một ký tự thì hàm nhận ngay ký tự đó (không cần bấm thêm phím Enter như trong các hàm nhập khác). Ký tự vừa gõ không hiện lên màn hình.

Nếu dùng :

```
biến=getch();
```

Thì biến sẽ chứa ký tự đọc vào.

Ví dụ :

c = getch();

2..2.4. Hàm putch() :

Cách dùng :

Dùng câu lệnh sau :

putch(ch);

Công dụng :

Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị theo màu xác định trong hàm textcolor.

Hàm cũng trả về ký tự được hiển thị.

2.3. Đưa kết quả lên màn hình - hàm printf :

Cách dùng :

prinf(điều khiển, đối số 1, đối số 2, ...);

Hàm printf chuyển, tạo khuôn dạng và in các đối của nó ra thiết bị ra chuẩn dưới sự điều khiển của xâu *điều khiển.* Xâu *điều khiển* chứa hai kiểu đối tượng : các ký tự thông thường, chúng sẽ được đưa ra trực tiếp thiết bị ra, và các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng và in đối tiếp sau của printf.

Chuỗi *điều khiển* có thể có các ký tự điều khiển :

\n	sang dòng mới
\f	sang trang mới
\b	lùi lại một bước
\t	dấu tab

Dạng tổng quát của đặc tả :

%[-][fw][.pp]ký tự chuyển dạng

Mỗi đặc tả chuyển dạng đều được đưa vào bằng ký tự % và kết thúc bởi một ký tự chuyển dạng. Giữa % và ký tự chuyển dạng có thể có :

Dấu trừ :

Khi không có dấu trừ thì kết quả ra được dồn về bên phải nếu độ dài thực tế của kết quả ra nhỏ hơn độ rộng tối thiểu fw dành cho nó. Các vị trí dư thừa sẽ được lấp đầy bằng các khoảng trống. Riêng đối với các trường số, nếu dãy số fw bắt đầu bằng số 0 thì các vị trí dư thừa bên trái sẽ được lấp đầy bằng các số 0.

Khi có dấu trừ thì kết quả được dồn về bên trái và các vị trí dư thừa về bên phải (nếu có) luôn được lấp đầy bằng các khoảng trống.

fw :

Khi fw lớn hơn độ dài thực tế của kết quả ra thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số 0 và nội dung của kết quả ra sẽ được đẩy về bên phải hoặc bên trái.

Khi không có fw hoặc fw nhỏ hơn hay bằng độ dài thực tế của kết quả ra thì độ rộng trên thiết bị ra dành cho kết quả sẽ bằng chính độ dài của nó.

Tại vị trí của fw ta có thể đặt dấu *, khi đó fw được xác định bởi giá trị nguyên của đối tương ứng.

Ví dụ :

Kết quả ra	fw	Dấu -	Kết quả đưa ra
-2503	8	có	-2503
-2503	08	có	-2503
-2503	8	không	-2503
-2503	08	không	000-2503
"abcdef"	8	không	abcdef
"abcdef"	08	có	abcdef
"abcdef"	08	không	abcdef

pp :

Tham số pp chỉ được sử dụng khi đối tương ứng là một xâu ký tự hoặc một giá trị kiểu float hay double.

Trong trường hợp đối tương ứng có giá trị kiểu float hay double thì pp là độ chính xác của trường ra. Nói một cách cụ thể hơn giá trị in ra sẽ có pp chữ số sau số thập phân.

Khi vắng mặt pp thì độ chính xác sẽ được xem là 6.

Khi đối là xâu ký tự :

Nếu pp nhỏ hơn độ dài của xâu thì chỉ pp ký tự đầu tiên của xâu được in ra. Nếu không có pp hoặc nếu pp lớn hơn hay bằng độ dài của xâu thì cả xâu ký tự sẽ được in ra.

Ví dụ :

Kết quả ra	fw	pp	Dấu -	Kết quả đưa ra	Độ dài trường ra
-435.645	10	2	có	-435.65	7
-435.645	10	0	có	-436	4
-435.645	8	vắng	có	-435.645000	11
"alphabeta"	8	3	vắng	alp	3
"alphabeta"	vắng	vắng	vắng	alphabeta	9
"alpha"	8	6	có	alpha	5

Các ký tự chuyển dạng và ý nghĩa của nó :

Ký tự chuyển dạng là một hoặc một dãy ký hiệu xác định quy tắc chuyển dạng và dạng in ra của đối tương ứng. Như vậy sẽ có tình trạng cùng một số sẽ được in ra theo các dạng khác nhau. Cần phải sử dụng các ký tự chuyển dạng theo đúng qui tắc định sẵn. Bảng sau cho các thông tin về các ký tự chuyển dạng.

Ký tự chuyển dạng	Ý nghĩa
d	Đối được chuyển sang số nguyên hệ thập phân
o	Đối được chuyển sang hệ tám không dấu (không có số 0 đứng trước)
x	Đối được chuyển sang hệ mười sáu không dấu (không có 0x đứng trước)
u	Đối được chuyển sang hệ thập phân không dấu
c	Đối được coi là một ký tự riêng biệt
s	Đối là xâu ký tự, các ký tự trong xâu được in cho tới khi gặp ký tự không hoặc cho tới khi đủ số lượng ký tự được xác định bởi các đặc tả về độ chính xác pp.
e	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m.n..nE[+ hoặc -] với độ dài của xâu chứa n là pp.
f	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m..m.n..n với độ dài của xâu chứa n là pp. Độ chính

	xác mặc định là 6. Lưu ý rằng độ chính xác không xác định ra số các chữ số có nghĩa phải in theo khuôn dạng f.
g	Dùng %e hoặc %f, tuỳ theo loại nào ngắn hơn, không in các số 0 vô nghĩa.

Chú ý :

Mọi dãy ký tự không bắt đầu bằng % hoặc không kết thúc bằng ký tự chuyển dạng đều được xem là ký tự hiển thị.

Để hiển thị các ký tự đặc biệt :

Cách viết	Hiển thị
\'	'
\"	"
\\	\

Các ví dụ :

1	printf("\" Nang suat tang : %d % \" \n\\d"",30,-50);	"Nang suat tang ; 30 %" \d=-50

2	n=8
	float x=25.5, y=-47.335
	printf("\n%f\n%*.2f",x,n,y);
	Lệnh này tương đương với
	printf("\n%f\n%8.2f",x,n,y);
	Vì n=8 tương ứng với vị trí *
		25.500000
		-47.34

2.4. Vào số liệu từ bàn phím - hàm scanf :

Hàm scanf là hàm đọc thông tin từ thiết bị vào chuẩn (bàn phím), chuyển dịch chúng (thành số nguyên, số thực, ký tự vv..) rồi lưu trữ nó vào bộ nhớ theo các địa chỉ xác định.

Cách dùng :

scanf(điều khiển,đối 1, đối 2, ...);

Xâu *điều khiển* chứa các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng biến tiếp sau của scanf.

Đặc tả có thể viết một cách tổng quát như sau :

%[*][d...d]ký tự chuyển dạng

Việc có mặt của dấu * nói lên rằng trường vào vẫn được dò đọc bình thường, nhưng giá trị của nó bị bỏ qua (không được lưu vào bộ nhớ). Như vậy đặc tả chứa dấu * sẽ không có đối tương ứng.

d...d là một dãy số xác định chiều dài cực đại của trường vào, ý nghĩa của nó được giải thích như sau :

Nếu tham số d...d vắng mặt hoặc nếu giá trị của nó lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ trường vào sẽ được đọc, nội dung của nó được dịch và được gán cho địa chỉ tương ứng (nếu không có dấu *).

Nếu giá trị của d...d nhỏ hơn độ dài của trường vào thì chỉ phần đầu của trường có kích cỡ bằng d...d được đọc và gán cho địa chỉ của biến tương ứng. Phần còn lại của trường sẽ được xem xét bởi các đặc tả và đối tương ứng tiếp theo.

Ví dụ :

```
int a;
float x,y;
char ch[6],ct[6]
scanf("%f%5f%3d%3s%s",&x&y&a&ch&ct0;
```

Với dòng vào : 54.32e-1 25 12452348a

Kết quả là lệnh scanf sẽ gán

5.432 cho x

25.0 cho y

124 cho a

xâu "523" và dấu kết thúc \0 cho ch

xâu "48a" và dấu kết thúc \0 cho ct

Ký tự chuyển dạng :

Ký tự chuyển dạng xác định cách thức dò đọc các ký tự trên dòng vào cũng như cách chuyển dịch thông tin đọc được trước khi gán nó cho các địa chỉ tương ứng.

Cách dò đọc thứ nhất là đọc theo trường vào, khi đó các khoảng trắng bị bỏ qua. Cách này áp dụng cho hầu hết các trường hợp.

Cách dò đọc thứ hai là đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét bình đẳng như các ký tự khác. Phương pháp này chỉ xảy ra khi ta sử dụng một trong ba ký tự chuyển dạng sau : C, [dãy ký tự], [^ dãy ký tự]

Các ký tự chuyển dạng và ý nghĩa của nó :

c	Vào một ký tự, đối tương ứng là con trỏ ký tự. Có xét ký tự khoảng trắng
d	Vào một giá trị kiểu int, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên
ld	Vào một giá trị kiểu long, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên
o	Vào một giá trị kiểu int hệ 8, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 8
lo	Vào một giá trị kiểu long hệ 8, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 8
x	Vào một giá trị kiểu int hệ 16, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 16
lx	Vào một giá trị kiểu long hệ 16, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 16
f hay e	Vào một giá trị kiểu float, đối tương ứng là con trỏ float, trường vào phải là số dấu phảy động
lf hay le	Vào một giá trị kiểu double, đối tương ứng là con trỏ double, trường vào phải là số dấu phảy động
s	Vào một giá trị kiểu double, đối tương ứng là con trỏ kiểu char, trường vào phải là dãy ký tự bất kỳ không chứa các dấu cách và các dấu xuống dòng

[Dãy ký tự], [^Dãy ký tự] Các ký tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một ký tự không thuộc tập các ký tự đặt trong[]. Đối tương ứng là con trỏ kiểu char. Trường vào là dãy ký tự bất kỳ (khoảng trắng được xem như một ký tự).

Ví dụ :

```
int a,b;
char ch[10], ck[10];
scanf("%d%[0123456789]%[^0123456789]%3d",&a,ch,ck,&b);
```

Với dòng vào :

35 13145 xyz 584235

Sẽ gán :

35 cho a

xâu "13145" cho ch

xâu "xyz' cho ck

584 cho b

Chú ý :

Xét đoạn chương trình dùng để nhập (từ bàn phím) ba giá trị nguyên rồi gán cho ba biến a,b,c như sau :

```
int a,b,c;
scanf("%d%d%d",&a,&b,&c);
```

Để vào số liệu ta có thể thao tác theo nhiều cách khác nhau:

Cách 1 :

Đưa ba số vào cùng một dòng, các số phân cách nhau bằng dấu cách hoặc dấu tab.

Cách 2 :

Đưa ba số vào ba dòng khác nhau.

Cách 3 :

Hai số đầu cùng một dòng (cách nahu bởi dấu cách hoặ tab), số thứ ba trên dòng tiếp theo.

Cách 4 :

Số thứ nhất trên một dòng, hai số sau cùng một dòng tiếp theo (cách nahu bởi dấu cách hoặ tab), số thứ ba trên dòng tiếp theo.

Khi vào sai sẽ báo lỗi và nhảy về chương trình chứa lời gọi nó.

2.5. Đưa kết quả ra máy in :

Để đưa kết quả ra máy in ta dùng hàm chuẩn fprintf có dạng sau :

fprintf(stdprn, điều khiển, biến 1, biến 2,...);

Tham số stdprn xác định thiết bị đưa ra là máy in.

Điều khiển có dạng đặc tả như lệnh printf.

Dùng giống như lệnh printf, chỉ khác là in ra máy in.

Ví dụ :

Đoạn chương trình in ma trận A, cỡ 8x6. Mỗi hàng của ma trận được in trên một dòng :

```
float a[8][6];
int i,j;
fprintf(stdprn,"\n%20c MA TRAN A\n\n\n",' ');
for (i=0;i<8;++i)
        { for (j=0;j<6;++j)
          fprintf(stdprn,"%10.2f",a[i][j]);
          fprintf(stdprn,"\n");
        }
```

Chương 3

BIỂU THỨC

Toán hạng có thể xem là một đại lượng có một giá trị nào đó. Toán hạng bao gồm hằng, biến, phần tử mảng và hàm.

Biểu thức lập nên từ các toán hạng và các phép tính để tạo nên những giá trị mới. Biểu thức dùng để diễn đạt một công thức, một qui trình tính toán, vì vậy nó là một thành phần không thể thiếu trong chương trình.

3.1. Biểu thức :

Biểu thức là một sự kết hợp giữa các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó. Mỗi biểu thức có sẽ có một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức.

Trong C, ta có hai khái niệm về biểu thức :

Biểu thức gán.

Biểu thức điều kiện .

Biểu thức được phân loại theo kiểu giá trị : nguyên và thực. Trong các mệnh đề logic, biểu thức được phân thành đúng (giá trị khác 0) và sai (giá trị bằng 0).

Biểu thức thường được dùng trong :

Vế phải của câu lệnh gán.

Làm tham số thực sự của hàm.

Làm chỉ số.

Trong các toán tử của các cấu trúc điều khiển.

Tới đây, ta đã có hai khái niệm chính tạo nên biểu thức đó là toán hạng và phép toán. Toán hạng gồm : hằng, biến, phần tử mảng và hàm trước đây ta đã xét. Dưới đây ta sẽ nói đến các phép toán. Hàm sẽ được đề cập trong chương 6.

3.2. Lệnh gán và biểu thức:

Biểu thức gán là biểu thức có dạng :

v=e

Trong đó v là một biến (hay phần tử mảng), e là một biểu thức. Giá trị của biểu thức gán là giá trị của e, kiểu của nó là kiểu của v. Nếu đặt dấu ; vào sau biểu thức gán ta sẽ thu được phép toán gán có dạng :

v=e;

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác. Ví dụ như khi ta viết

a=b=5;

thì điều đó có nghĩa là gán giá trị của biểu thức b=5 cho biến a. Kết qủa là b=5 và a=5.

Hoàn toàn tương tự như :

a=b=c=d=6; gán 6 cho cả a, b, c và d

Ví dụ :

z=(y=2)*(x=6); { ở đây * là phép toán nhân }

gán 2 cho y, 6 cho x và nhân hai biểu thức lại cho ta z=12.

3.3. Các phép toán số học :

Các phép toán hai ngôi số học là

Phép toán	Ý nghiã	Ví dụ
+	Phép cộng	a+b
-	Phép trừ	a-b
*	Phép nhân	a*b
/	Phép chia	a/b (Chia số nguyên sẽ chặt phần thập phân)
%	Phép lấy phần dư	a%b (Cho phần dư của phép chia a cho b)

Có phép toán một ngôi - ví du -(a+b) sẽ đảo giá trị của phép cộng (a+b).

Ví dụ :

11/3=3

11%3=2

-(2+6)=-8

Các phép toán + và - có cùng thứ tự ưu tiên, có thứ tự ưu tiên nhỏ hơn các phép * , / , % và cả ba phép này lại có thứ tự ưu tiên nhỏ hơn phép trừ một ngôi.

Các phép toán số học được thực hiện từ trái sang phải. Số ưu tiên và khả năng kết hợp của phép toán được chỉ ra trong một mục sau này

3.4. Các phép toán quan hệ và logic :

Phép toán quan hệ và logic cho ta giá trị đúng (1) hoặc giá trị sai (0). Nói cách khác, khi các điều kiện nêu ra là đúng thì ta nhận được giá trị 1, trái lại ta nhận giá trị 0.

Các phép toán quan hệ là :

Phép toán	Ý nghiã	Ví dụ
>	So sánh lớn hơn	a>b 4>5 có giá trị 0
>=	So sánh lớn hơn hoặc bằng	a>=b 6>=2 có giá trị 1
<	So sánh nhỏ hơn	a<b 6<=7 có giá trị 1
<=	So sánh nhỏ hơn hoặc bằng	a<=b 8<=5 có giá trị 0
==	So sánh bằng nhau	a==b 6==6 có giá trị 1
!=	So sánh khác nhau	a!=b 9!=9 có giá trị 0

Bốn phép toán đầu có cùng số ưu tiên, hai phép sau có cùng số thứ tự ưu tiên nhưng thấp hơn số thứ tự của bốn phép đầu.

Các phép toán quan hệ có số thứ tự ưu tiên thấp hơn so với các phép toán số học, cho nên biểu thức :

i<n-1

được hiểu là i<(n-1).

Các phép toán logic :

Trong C sử dụng ba phép toán logic :

Phép phủ định một ngôi !

a	!a
khác 0	0
bằng 0	1

Phép và (AND) &&

Phép hoặc (OR) ||

a	b	a&&b	a||b
khác 0	khác 0	1	1
khác 0	bằng 0	0	1
bằng 0	khác 0	0	1
bằng 0	bằng 0	0	0

Các phép quan hệ có số ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và ||, vì vậy biểu thức như :

(a<b)&&(c>d)

có thể viết lại thành :

a<b&&c>d

Chú ý :

Cả a và b có thể là nguyên hoặc thực.

3.5. Phép toán tăng giảm :

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng là ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm -- thì sẽ trừ toán hạng đi 1.

Ví dụ :

n=5

++n Cho ta n=6

--n Cho ta n=4

Ta có thể viết phép toán ++ và -- trước hoặc sau toán hạng như sau : ++n, n++, --n, n--.

Sự khác nhau của ++n và n++ ở chỗ : trong phép n++ thì tăng sau khi giá trị của nó đã được sử dụng, còn trong phép ++n thì n được tăng trước khi sử dụng. Sự khác nhau giữa n-- và --n cũng như vậy.

Ví dụ :

n=5

x=++n Cho ta x=6 và n=6

x=n++ Cho ta x=5 và n=6

3.6. Thứ tự ưu tiên các phép toán :

Các phép toán có độ ưu tiên khác nhau, điều này có ý nghĩa trong cùng một biểu thức sẽ có một số phép toán này được thực hiện trước một số phép toán khác.

Thứ tự ưu tiên của các phép toán được trình bày trong bảng sau :

TT	Phép toán	Trình tự kết hợp
1	() [] ->	Trái qua phải
2	! ~ & * - ++ -- (type) sizeof	Phải qua trái
3	* (phép nhân) / %	Trái qua phải
4	+ -	Trái qua phải
5	<< >>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	&	Trái qua phải
9	^	Trái qua phải
10	|	Trái qua phải
11	&&	Trái qua phải
12	||	Trái qua phải
13	?:	Phải qua trái
14	= += -= *= /= %= <<= >>= &= ^= |=	Phải qua trái
15	,	Trái qua phải

Chú thích :

Các phép toán tên một dòng có cùng thứ tự ưu tiên, các phép toán ở hàng trên có số ưu tiên cao hơn các số ở hàng dưới.

Đối với các phép toán cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại được chỉ ra trong cột *trình tự kết hợp*.

Ví dụ :

--px=(--px) (Phải qua trái)

8/4*6=(8/4)*6 (Trái qua phải)

Nên dùng các dấu ngoặc tròn để viết biểu thức một cách chính xác.

Các phép toán lạ :

Dòng 1

[ ] Dùng để biểu diễn phần tử mảng, ví dụ : a[i][j]

. Dùng để biểu diễn thành phần cấu trúc, ví dụ : ht.ten

-> Dùng để biểu diễn thành phần cấu trúc thông qua con trỏ

Dòng 2

* Dùng để khai báo con trỏ, ví dụ : int *a

& Phép toán lấy địa chỉ, ví dụ : &x

(type) là phép chuyển đổi kiểu, ví dụ : (float)(x+y)

Dòng 15

Toán tử **,** thường dùng để viết một dãy biểu thức trong toán tử for.

3.7. Chuyển đổi kiểu giá trị :

Việc chuyển đổi kiểu giá trị thường diễn ra một cách tự động trong hai trường hợp sau :

Khi gán biểu thức gồm các toán hạng khác kiểu.

Khi gán một giá trị kiểu này cho một biến (hoặc phần tử mảng) kiểu khác. Điều này xảy ra trong toán tử gán, trong việc truyền giá trị các tham số thực sự cho các đối.

Ngoài ra, ta có thể chuyển từ một kiểu giá trị sang một kiểu bất kỳ mà ta muốn bằng phép chuyển sau :

(type) biểu thức

Ví dụ :

(float) (a+b)

Chuyển đổi kiểu trong biểu thức :

Khi hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán. Kết quả thu được là một giá trị kiểu cao hơn. Chẳng hạn :

Giữa int và long thì int chuyển thành long.

Giữa int và float thì int chuyển thành float.

Giữa float và double thì float chuyển thành double.

Ví dụ :

1.5*(11/3)=4.5

1.5*11/3=5.5

(11/3)*1.5=4.5

Chuyển đổi kiểu thông qua phép gán :

Giá trị của vế phải được chuyển sang kiểu vế trái đó là kiểu của kết quả. Kiểu int có thể được được chuyển thành float. Kiểu float có thể chuyển thành int do chặt đi phần thập phân. Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int bằng cách cắt bỏ một vài chữ số.

Ví dụ :

int n;

n=15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức :

Theo cách này, kiểu của biểu thức được đổi thành kiểu type theo nguyên tắc trên.

Ví dụ :

Phép toán : (int)a

cho một giá trị kiểu int. Nếu a là float thì ở đây có sự chuyển đổi từ float sang int. Chú ý rằng bản thân kiểu của a vẫn không bị thay đổi. Nói cách khác, a vẫn có kiểu float nhưng (int)a có kiểu int.

Đối với hàm toán học của thư viện chuẩn, thì giá trị của đối và giá trị của hàm đều có kiểu double, vì vậy để tính căn bậc hai của một biến nguyên n ta phải dùng phép ép kiểu để chuyển kiểu int sang double như sau :

sqrt((double)n)

Phép ép kiểu có cùng số ưu tiên như các toán tử một ngôi.

Chú ý :

Muốn có giá trị chính xác trong phép chia hai số nguyên cần dùng phép ép kiểu :

((float)a)/b

Để đổi giá trị thực r sang nguyên, ta dùng :

(int)(r+0.5)

Chú ý thứ tự ưu tiên :

(int)1.4*10=1*10=10

(int)(1.4*10)=(int)14.0=14

Chương 4

CẤU TRÚC CƠ BẢN CỦA CHƯƠNG TRÌNH

4.1. Lời chú thích :

Các lời bình luận, các lời giải thích có thể đưa vào ở bất kỳ chỗ nào của chương trình để cho chương trình dễ hiểu, dễ đọc hơn mà không làm ảnh hưởng đến các phần khác. Lời giải thích được đặt giữa hai dấu /* và */.

Trong một chương trình cần (và luôn luôn cần) viết thêm những lời giải thích để chương trình thêm rõ ràng, thêm dễ hiểu.

Ví dụ :

```
#include "stdio.h"
 #include "string.h"
 #include "alloc.h"
 #include "process.h"
 int main()
 {
   char *str;
   /* Cấp phát bộ nhớ cho xâu ký tự */
   if ((str = malloc(10)) == NULL)
   {
     printf("Not enough memory to allocate buffer\n");
     exit(1);  /* Kết thúc chương trình nếu thiếu bộ nhớ */
   }
   /* copy "Hello" vào xâu */
   strcpy(str, "Hello");
   /* Hiển thị xâu */
   printf("String is %s\n", str);

   /* Giải phóng bộ nhớ */
   free(str);
   return 0;
 }
```

4.2. Lệnh và khối lệnh :

4.2.1. Lệnh :

Một biểu thức kiểu như x=0 hoặc ++i hoặc scanf(...) trở thành câu lệnh khi có đi kèm theo dấu ;

Ví dụ :

```
x=0;
++i;
scanf(...);
```

Trong chương trình C, dấu ; là dấu hiệu kết thúc câu lệnh.

4.2.2. Khối lệnh :

Một dãy các câu lệnh được bao bởi các dấu { } gọi là một khối lệnh. Ví dụ :

```
{
    a=2;
    b=3;
    printf("\n%6d%6d",a,b);
}
```

TURBO C xem khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác, chỗ nào viết được một câu lệnh thì ở đó cũng có quyền đặt một khối lệnh.

Khai báo ở đầu khối lệnh :

Các khai báo biến và mảng chẳng những có thể đặt ở đầu của một hàm mà còn có thể viết ở đầu khối lệnh :

```
{
    int a,b,c[50];
    float x,y,z,t[20][30];
    a==b==3;
    x=5.5; y=a*x;
    z=b*x;
    printf("\n y= %8.2f\n z=%8.2f",y,z);
}
```

Sự lồng nhau của các khối lệnh và phạm vi hoạt động của các biến và mảng :

Bên trong một khối lệnh lại có thể viết lồng khối lệnh khác. Sự lồng nhau theo cách như vậy là không hạn chế.

Khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại trong thời gian máy làm việc bên trong khối lệnh và chúng lập tức biến mất ngay sau khi máy ra khỏi khối lệnh. Vậy :

Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó.

ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh

Nếu bên trong một khối ta dùng một biến hay một mảng có tên là a thì điều này không làm thay đổi giá trị của một biến khác cũng có tên là a (nếu có) được dùng ở đâu đó bên ngoài khối lệnh này.

Nếu có một biến đã được khai báo ở ngoài một khối lệnh và không trùng tên với các biến khai báo bên trong khối lệnh này thì biến đó cũng có thể sử dụng cả bên trong cũng như bên ngoài khối lệnh.

Ví dụ :

Xét đoạn chương trình sau :

```
{
        int a=5,b=2;
        {
                int a=4;
                b=a+b;
                printf("\n a trong =%3d  b=%3d",a,b);
        }
        printf("\n a ngoai =%3d  b=%3d",a,b);
}
```

Khi đó đoạn chương trình sẽ in kết quả như sau :

a trong =4 b=6

a ngoài =5 b=6

Do tính chất biến a trong và ngoài khối lệnh.

4.3. Cấu trúc cơ bản của chương trình :

Cấu trúc chương trình và hàm là một trong các vấn đề quan trọng của C. Về hàm ta sẽ có một chương nói tỉ mỷ về nó. ở đây ta chỉ đưa ra một số qui tắc chung :

Hàm là một đơn vị độc lập của chương trình. Tính độc lập của hàm thể hiện ở hai điểm :

Không cho phép xây dựng một hàm bên trong các hàm khác.

Mỗi hàm có các biến, mảng .. riêng của nó và chúng chỉ được sử dụng nội bộ bên trong hàm. Nói cách khác hàm là đơn vị có tính chất khép kín.

Một chương trình bao gồm một hoặc nhiều hàm. Hàm main() là thành phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện các câu lệnh đầu tiên của hàm main() và kết thúc khi gặp dấu } cuối cùng của hàm này. Khi chương trình làm việc, máy có thể chạy từ hàm này sang hàm khác.

Các chương trình C được tổ chức theo mẫu :

.....

hàm 1

.....

hàm 2

.....

.....

hàm n

Bên ngoài các hàm ở các vị trí (.....) là chỗ đặt : các toán tử #include ... (dùng để khai báo sử dụng các hàm chuẩn), toán tử #define ... (dùng để định nghĩa các hằng), định nghĩa kiểu dữ liệu bằng typedef, khai báo các biến ngoài, mảng ngoài....

Việc truyền dữ liệu và kết quả từ hàm này sang hàm khác được thực hiện theo một trong hai cách :

Sử dụng đối của hàm.

Sử dụng biến ngoài, mảng ngoài ...

Vậy nói tóm lại cấu truc cơ bản của chương trình như sau :

- Các #include
- Các #define

- Khai báo các đối tượng dữ liệu ngoài (biến, mảng, cấu trúc vv..).
- Khai báo nguyên mẫu các hàm.
- Hàm main().
- Định nghĩa các hàm (hàm main có thể đặt sau hoặc xen vào giữa các hàm khác).

Ví dụ :

Chương trình tính x lũy thừa y rỗi in ra máy in kết quả :

```
#include "stdio.h"
#include "math.h"
main()
        {
                double x,y,z;
                printf("\n Nhap x va y");
                scanf("%lf%lf",&x,&y);
                z=pow(x,y); /* hàm lấy luỹ thừa y luỹ thừa x */
                fprintf(stdprn,"\n x= %8.2lf \n y=%8.2lf \n z=%8.2lf",x,y,z);
        }
```

4.4. Một số qui tắc cần nhớ khi viết chương trình :

Qui tắc đầu tiên cần nhớ là :

Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu ***;***

Qui tắc thứ hai là :

C*ác lời giải thích cần được đặt giữa các dấu /* và */ và có thể được viết*

Trên một dòng

Trên nhiều dòng

Trên phần còn lại của dòng

Qui tắc thứ ba là :

Trong chương trình, khi ta sử dụng các hàm chuẩn, ví dụ như printf(), getch() ,... mà các hàm này lại chứa trong file stdio.h trong thư mục của C, vì vậy ở đầu chương trình ta phải khai báo sử dụng ;

#include "stdio.h "

Qui tắc thứ tư là :

Một chương trình có thể chỉ có một hàm chính (hàm main()) hoặc có thể có thêm vài hàm khác.

Chương 5

CẤU TRÚC ĐIỀU KHIỂN

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các cấu trúc điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể nhảy thực hiện một câu lệnh khác ở một ví trí trước hoặc sau câu lệnh hiện thời.

Xét về mặt công dụng, có thể chia các cấu trúc điều khiển thành các nhóm chính :

Nhảy không có điều kiện.

Rẽ nhánh.

Tổ chức chu trình.

Ngoài ra còn một số toán tử khác có chức năng bổ trợ như break, continue.

5.1. Cấu trúc có điều kiện :

5.1.1. Lệnh if-else :

Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau :

```
if ( biểu thức )
khối lệnh 1;
/* Dạng một */
```

```
if ( biểu thức )
  khối lệnh 1;
else
  khối lệnh 2 ;
/* Dạng hai */
```

Hoạt động của biểu thức dạng 1 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu

biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ :

Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình. Chương trình có thể viết bằng cả hai cách trên như sau :

```
#include "stdio.h"
main()
        {
                float a,b,max;
                printf("\n Cho a=");
                scanf("%f",&a);
                printf("\n Cho b=");
                scanf("%f",&b);
                max=a;
                if (b>max) max=b;
                printf(" \n Max cua hai so a=%8.2f  va b=%8.2f la Max=%8.2f",a,b,max);
        }
#include "stdio.h"
main()
        {
                float a,b,max;
                printf("\n Cho a=");
                scanf("%f",&a);
                printf("\n Cho b=");
                scanf("%f",&b);
                if (a>b) max=a;
                else max=b;
                printf(" \n Max cua hai so a=%8.2f  va b=%8.2f la Max=%8.2f",a,b,max);
        }
```

Sự lồng nhau của các toán tử if :

C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else.

Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất. Chẳng hạn như đoạn chương trình ví dụ sau :

```
if ( n>0 )          /* if thứ nhất*/
        if ( a>b )          /* if thứ hai*/
                z=a;
        else
                z=b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với :

```
if ( n>0 )          /* if thứ nhất*/
         {
                if ( a>b )          /* if thứ hai*/
                        z=a;
                else
                        z=b;
        }
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau :

```
if ( n>0 )          /* if thứ nhất*/
         {
                if ( a>b )          /* if thứ hai*/
                        z=a;
        }
else
        z=b;
```

5.1.2. Lệnh else-if :

Khi muốn thực hiện một trong n quyết định ta có thể sử dụng cấu trúc sau :

```
if ( biểu thức 1 )
khối lệnh 1;
else if ( biểu thức 2 )
 khối lệnh 2;
.....
else if ( biểu thức n-1 )
 khối lệnh n-1;
else
 khối lệnh n;
```

Trong cấu trúc này, máy sẽ đi kiểm tra từ biểu thức 1 trở đi đến khi gặp biểu thức nào có giá trị khác 0.

Nếu biểu thức thứ i (1,2, ...n-1) có giá trị khác 0, máy sẽ thực hiện khối lệnh i, rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Nếu trong cả n-1 biểu thức không có biểu thức nào khác 0, thì máy sẽ thực hiện khối lệnh n rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Ví dụ :

Chương trình giải phương trình bậc hai.

```
#include "stdio.h"
main()
 {
        float a,b,c,d,x1,x2;
        printf("\n Nhap a, b, c:");
        scanf("%f%f%f,&a&b&c);
        d=b*b-4*a*c;
        if (d<0.0)
                printf("\n Phuong trinh vo nghiem ");
        else if (d==0.0)
                printf("\n Phuong trinh co nghiem kep x1,2=%8.2f",-b/(2*a));
        else
            {
                printf("\n Phuong trinh co hai nghiem ");
                printf("\n x1=%8.2f",(-b+sqrt(d))/(2*a));
```

```
        printf("\n x2=%8.2f",(-b-sqrt(d))/(2*a));
    }
```

5.2. Lệnh nhảy không điều kiện - toán tử goto :

Nhãn có cùng dạng như tên biến và có dấu : đứng ở phía sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình.

Ví dụ :

ts : s=s++;

thì ở đây **ts** là nhãn của câu lệnh gán s=s++.

Toán tử goto có dạng :

goto nhãn;

Khi gặp toán tử này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khoá goto.

Khi dùng toán tử goto cần chú ý :

Câu lệnh goto và nhãn cần nằm trong một hàm, có nghĩa là toán tử goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân một hàm và không thể dùng để nhảy từ một hàm này sang một hàm khác.

Không cho phép dùng toán tử goto để nhảy từ ngoài vào trong một khối lệnh. Tuy nhiên việc nhảy từ trong một khối lệnh ra ngoài là hoàn toàn hợp lệ. Ví dụ như đoạn chương trình sau là sai.

```
goto n1;

.......
{       .....
        n1: printf("\n Gia tri cua N la: ");
        .....
}
```

Ví dụ :

Tính tổng s=1+2+3+....+10

```
#include "stdio.h"
main()
```

```
{
        int s,i;
        i=s=0;
        tong:
        ++i;
        s=s+i;
        if (i<10) goto tong;
        printf("\n tong s=%d",s);
}
```

5.3. Cấu trúc rẽ nhánh - toán tử switch:

Là cấu trúc tạo nhiều nhánh đặc biệt. Nó căn cứ vào giá trị một biểu thức nguyên để để chọn một trong nhiều cách nhảy.

Cấu trúc tổng quát của nó là :

```
switch ( biểu thức nguyên )
  {
        case n1
                khối lệnh 1
        case n2
                khối lệnh 2
        .......
        case nk
        khối lệnh k
        [ default
                khối lệnh k+1    ]
}
```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch.

default là một thành phần không bắt buộc phải có trong thân của switch.

Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc ( ) như sau :

Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni.

Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau :

Khi có default máy sẽ nhảy tới câu lệnh sau nhãn default.

Khi không có default máy sẽ nhảy ra khỏi cấu trúc switch.

Chú ý :

Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ :

Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch :

```
#include "stdio.h"
main()
        {
                int diem;
                tt: printf("\nVao du lieu :");
                printf("\n Diem =");
                scanf("%d",&diem);
                switch (diem)
                {
                        case 0:
                        case 1:
                        case 2:
                        case 3:printf("Kem\n");break;
                        case 4:printf("Yeu\n");break;
```

```
                case 5:
                case 6:printf("Trung binh\n");break;
                case 7:
                case 8:printf("Kha\n");break;
                case 9:
                case 10:printf("Gioi\n");break;
                default:printf(Vao sai\n);
        }
        printf("Tiep tuc 1, dung 0 :")
        scanf("%d",&diem);
        if (diem==1) goto tt;
        getch();
        return;
}
```

5.4. Cấu trúc lặp :

5.4.1. Cấu trúc lặp với toán tử while và for :

5.4.1.1. Cấu trúc lặp với toán tử while :

Toán tử while dùng để xây dựng chu trình lặp dạng :

```
while ( biểu thức )
        Lệnh hoặc khối lệnh;
```

Như vậy toán tử while gồm một biểu thức và thân chu trình. Thân chu trình có thể là một lệnh hoặc một khối lệnh.

Hoạt động của chu trình như sau :

Máy xác định giá trị của biểu thức, tuỳ thuộc giá trị của nó máy sẽ chọn cách thực hiện như sau :

Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi chu trình và chuyển tới thực hiện câu lệnh tiếp sau chu trình trong chương trình.

Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh hoặc khối lệnh trong thân của while. Khi máy thực hiện xong khối lệnh này nó lại thực hiện xác định lại giá trị biểu thức rồi làm tiếp các bước như trên.

Chú ý :

Trong các dấu ngoặc ( ) sau while chẳng những có thể đặt một biểu thức mà còn có thể đặt một dãy biểu thức phân cách nhau bởi dấu phảy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

Bên trong thân của một toán tử while lại có thể sử dụng các toán tử while khác. bằng cách đó ta đi xây dựng được các chu trình lồng nhau.

Khi gặp câu lệnh break trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

Trong thân while có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân while để ra khỏi một hàm nào đó.

Ví dụ :

Chương trình tính tích vô hướng của hai véc tơ x và y :

Cách 1 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
        {
                float s=0;
                int i=-1;
                while (++i<4)
                s+=x[i]*y[i];
                printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
        }
```

Cách 2 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
        {
                float s=0;
                int i=0;
```

```
        while (1)
        {
                s+=x[i]*y[i];
                if (++i>=4) goto kt;
        }
        kt:printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
}
```

Cách 3 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
        {
                float s=0;
                int i=0;
                while (  s+=x[i]*y[i], ++i<=3 );
                printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
        }
```

5.4.1.2. Cấu trúc lặp với toán tử for :

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau :

```
for ( biểu thức 1; biểu thức 2; biểu thức 3)
        Lệnh hoặc khối lệnh ;
```

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ; .

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for :

Toán tử for hoạt động theo các bước sau :

Xác định biểu thức 1

Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh :

Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for.

Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Chú ý :

Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu ;. Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy ra sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một ví trí mong muốn bất kỳ.

Ví dụ 1:

Nhập một dãy số rồi đảo ngược thứ tự của nó.

Cách 1:

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
        int i,j;
```

```
        float c;
        for (i=0,j=n-1;i<j;++i,--j)
        {
                c=x[i];x[i]=x[j];x[j]=c;
        }
        fprintf(stdprn,"\n Day so dao la \n\n");
        for (i=0;i<n;++i)
        fprintf(stdprn,"%8.2f",x[i]);
}
```

Cách 2 :

```
#include "stdio.h"
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
        int i,j;
        float c;
        for (i=0,j=n-1;i<j;c=x[i],x[i]=x[j],x[j]=c,++i,--j)
        fprintf(stdprn,"\n Day so dao la \n\n");
        for (i=0;++i<n;)
        fprintf(stdprn,"%8.2f",x[i]);
}
```

Cách 3 :

```
#include "stdio.h"
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
        int i=0,j=n-1;
        float c;
        for ( ; ; )
```

```
        {
                c=x[i];x[i]=x[j];x[j]=c;
                if (++i>--j) break;
        }
        fprintf(stdprn,“\n Day so dao la \n\n”);
        for (i=-1;i++<n-1; fprintf(stdprn,“%8.2f”,x[i]));
}
```

Ví dụ 2:

Tính tích hai ma trận mxn và nxp.

```
#include "stdio.h"
float x[3][2],y[2][4],z[3][4],c;
main()
{
        int i,j;
        printf("\n nhap gia tri cho ma tran X ");
        for (i=0;i<=2;++i)
        for (j=0;j<=1;++j)
        {
                printf("\n x[%d][%d]=",i,j);
                scanf("%f",&c);
                x[i][j]=c;
        }
        printf("\n nhap gia tri cho ma tran Y ");
        for (i=0;i<=1;++i)
        for (j=0;j<=3;++j)
        {
                printf("\n y[%d][%d]=",i,j);
                scanf("%f",&c);
                y[i][j]=c;
        }
for (i=0;i<=3;++i)
for (j=0;j<=4;++j)
```

z[i][j]

}

5.4.2. Chu trình do-while

Khác với các toán tử while và for, việc kiểm tra điều kiện kết thúc đặt ở đầu chu trình, trong chu trình do while việc kiểm tra điều kiện kết thúc đặt cuối chu trình. Như vậy thân của chu trình bao giờ cũng được thực hiện ít nhất một lần.

Chu trình do while có dạng sau :

```
do
        Lệnh hoặc khối lệnh;
while ( biểu thức );
```

Lệnh hoặc khối lệnh là thân của chu trình có thể là một lệnh riêng lẻ hoặc là một khối lệnh.

Hoạt động của chu trình như sau :

Máy thực hiện các lệnh trong thân chu trình.

Khi thực hiện xong tất cả các lệnh trong thân của chu trình, máy sẽ xác định giá trị của biểu thức sau từ khoá while rồi quyết định thực hiện như sau :

Nếu biểu thức đúng (khác 0) máy sẽ thực hiện lặp lại khối lệnh của chu trình lần thứ hai rồi thực hiện kiểm tra lại biểu thức như trên.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc chu trình và chuyển tới thực hiện lệnh đứng sau toán tử while.

Chú ý :

Những điều lưu ý với toán tử while ở trên hoàn toàn đúng với do while.

Ví dụ :

Đoạn chương trình xác định phần tử âm đầu tiên trong các phần tử của mảng x.

```
#include "stdio.h"
float x[5],c;
main()
        {
                int i=0;
                printf("\n nhap gia tri cho ma tran x ");
```

```
        for (i=0;i<=4;++i)
        {
                printf("\n x[%d]=",i);
                scanf("%f",&c);
                y[i]=c;
        }
        do
                ++i;
        while (x[i]>=0 && i<=4);
        if (i<=4)
                printf("\n Phan tu am dau tien = x[%d]=%8.2f",i,x[i]);
        else
                printf("\n Mang khong có phan tu am ");
}
```

5.5. Câu lệnh break :

Câu lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình bên trong nhất chứa nó không cần điều kiện gì. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ví dụ :

Biết số nguyên dương n sẽ là số nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn bậc hai của n. Viết đoạn chương trình đọc vào số nguyên dương n, xem n có là số nguyên tố.

```
# include "stdio.h"
# include "math.h"
unsigned int n;
main()
        {
                int i,nt=1;
                printf("\n cho n=");
                scanf("%d",&n);
                for (i=2;i<=sqrt(n);++i)
```

```
        if ((n % i)==0)
        {
                nt=0;
                break;
        }
        if (nt)
                printf("\n %d la so nguyen to",n);
        else
                printf("\n %d khong la so nguyen to",n);
}
```

5.6. Câu lệnh continue :

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra, còn trong for điều khiển được chuyển về bước khởi đầu lại (tức là bước : tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình).

Chú ý :

Lệnh continue chỉ áp dụng cho chu trình chứ không áp dụng cho switch.

Ví dụ :

Viết chương trình để từ một nhập một ma trận a sau đó :

Tính tổng các phần tử dương của a.

Xác định số phần tử dương của a.

Tìm cực đại trong các phần tử dương của a.

```
#include "stdio.h"
float a[3[4];
main()
        {
                int i,j,soptd=0;
                float tongduong=0,cucdai=0,phu;
                for (i=0;i<3;++i)
                for (j=0;i<4;++j)
```

```
        {
                printf("\n a[%d][%d]=",i,j  );
                scanf("%f",&phu);
                a[i][j]=phu;
                if (a[i][j]<=0) continue;
                tongduong+=a[i][j];
                if (cucdai<a[i][j]) cucdai=a[i][j];
                ++soptd;
        }
        printf("\n So phan tu duong la : %d",soptd);
        printf("\n Tong cac phan tu duong la : %8.2f",tongduong);
        printf("\n Cuc dai phan tu duong la : %8.2f",cucdai);
}
```

Chương 6

HÀM

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó có một hàm chính (hàm main()). Hàm chia các bài toán lớn thành các công việc nhỏ hơn, giúp thực hiện những công việc lặp lại nào đó một cách nhanh chóng mà không phải viết lại đoạn chương trình. Thứ tự các hàm trong chương trình là bất kỳ, song chương trình bao giờ cũng đi thực hiện từ hàm main().

6.1. Cơ sở :

Hàm có thể xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không có phép xây dựng một hàm bên trong các hàm khác.

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau :

```
type tên hàm ( khai báo các đối )
{
        Khai báo các biến cục bộ
        Các câu lệnh
        [return[biểu thức];]
}
```

Dòng tiêu đề :

Trong dòng đầu tiên của hàm chứa các thông tin về : kiểu hàm, tên hàm, kiểu và tên mỗi đối.

Ví dụ :

```
        float max3s(float a, float b, float c)
```

khai báo các đối có dạng :

Kiểu đối 1 tên đối 1, kiểu đối 2 tên đối 2,..., kiểu đối n tên đối n

Thân hàm :

Sau dòng tiêu đề là thân hàm. Thân hàm là nội dung chính của hàm bắt đầu và kết thúc bằng các dấu { }.

Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đã đề ra cho hàm.

Thân hàm có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở các chỗ khác nhau, và cũng có thể không sử dụng câu lệnh này.

Dạng tổng quát của nó là :

return [biểu thức];

Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

Ví dụ :

Xét bài toán : Tìm giá trị lớn nhất của ba số mà giá trị mà giá trị của chúng được đưa vào bàn phím.

Xây dựng chương trình và tổ chức thành hai hàm : Hàm main() và hàm max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số đọc vào, giả sử là a,b,c. Nhiệm vụ của hàm main() là đọc ba giá trị vào từ bàn phím, rồi dùng hàm max3s để tính như trên, rồi đưa kết quả ra màn hình.

Chương trình được viết như sau :

```
#include "stdio.h"
float max3s(float a,float b,float c ); /* Nguyên mẫu hàm*/
main()
        {
                float x,y,z;
                printf("\n Vao ba so x,y,z:");
                scanf("%f%f%f",&x&y&z);
                printf("\n Max cua ba so x=%8.2f y=%8.2f z=%8.2f la : %8.2f",
                x,y,z,max3s(x,y,z));
        }       /* Kết thúc hàm main*/

float max3s(float a,float b,float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
```

```
        if (max<c) max=c;
        return(max);
} /* Kết thúc hàm max3s*/
```

Quy tắc hoạt động của hàm :

Một cách tổng quát lời gọi hàm có dạng sau :

tên hàm ([Danh sách các tham số thực])

Số các tham số thực tế thay vào trong danh sách các đối phải bằng số tham số hình thức và lần lượt chúng có kiểu tương ứng với nhau.

Khi gặp một lời gọi hàm thì nó sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau :

Cấp phát bộ nhớ cho các biến cục bộ.

Gán giá trị của các tham số thực cho các đối tương ứng.

Thực hiện các câu lệnh trong thân hàm.

Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối, biến cục bộ và ra khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

Các tham số thực, các đối và biến cục bộ :

Do đối và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối và biến cục bộ cần có tên khác nhau.

Đối và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được xét đến và bị xoá khi ra khỏi hàm nên ta không thể mang giá trị của đối ra khỏi hàm.

Đối và biến cục bộ có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gán cho các đối (trong ví dụ trên hàm max3s, các tham số thực là x,y,z, các đối tương ứng là a,b,c). Như vậy các đối chính là các bản sao của các tham số thực. Hàm chỉ làm việc trên các đối.

Các đối có thể bị biến đổi trong thân hàm, còn các tham số thực thì không bị thay đổi.

Chú ý :

Khi hàm khai báo không có kiểu ở trước nó thì nó được mặc định là kiểu int.

Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nói chung nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm hay tự động việc chuyển dạng.

Nguyên mẫu của hàm thực chất là dòng đầu tiên của hàm thêm vào dấu ;. Tuy nhiên trong nguyên mẫu có thể bỏ tên các đối.

Hàm thường có một vài đối. Ví dụ như hàm max3s có ba đối là a,b,c. cả ba đối này đều có giá trị float. Tuy nhiên, cũng có hàm không đối như hàm main.

Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị các đối.

6.2. Hàm không cho các giá trị :

Các hàm không cho giá trị giống như thủ tục (procedure) trong ngôn ngữ lập trình PASCAL. Trong trường hợp này, kiểu của nó là void.

Ví dụ hàm tìm giá trị max trong ba số là max3s ở trên có thể được viết thành thủ tục hiển thị số cực đại trong ba số như sau :

```
void htmax3s(float a, float b, float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
                if (max<c) max=c;
        }
```

Lúc này, trong hàm main ta gọi hàm htmax3s bằng câu lệnh :

```
htmax3s(x,y,z);
```

6.3. Hàm đệ qui :

6.3.3. Mở đầu :

C không những cho phép từ hàm này gọi tới hàm khác, mà nó còn cho phép từ một điểm trong thân của một hàm gọi tới chính hàm đó. Hàm như vậy gọi là hàm đệ qui.

Khi hàm gọi đệ qui đến chính nó, thì mỗi lần gọi máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với tập các biến cục bộ đã được tạo ra trong các lần gọi trước.

Để minh hoạ chi tiết những điều trên, ta xét một ví dụ về tính giai thừa của số nguyên dương n. Khi không dùng phương pháp đệ qui hàm có thể được viết như sau :

```
long int gt(int n) /* Tính n! với n>=0*/
 {
        long int gtphu=1;
        int i;
        for (i=1;i<=n;++i)
        gtphu*=i;
        return s;
 }
```

Ta nhận thấy rằng n! có thể tính theo công thức truy hồi sau :

n!=1 nếu n=0

n!=n*(n-1)! nếu n>0

Hàm tính n! theo phương pháp đệ qui có thể được viết như sau :

```
long int gtdq(int n)
  {
        if (n==0 || n==1)
          return 1;
        else
          return(n*gtdq(n-1));
  }
```

Ta đi giải thích hoạt động của hàm đệ qui khi sử dụng trong hàm main dưới đây :

```
#include "stdio.h"
main()
        {
        printf("\n 3!=%d",gtdq(3));
        }
```

Lần gọi đầu tiên tới hàm gtdq được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến tự động của hàm gtdq. Tập này chỉ gồm các đối n. Ta gọi đối n được tạo ra lần thứ nhất là n thứ nhất. Giá trị của tham số thực (số 3) được gán cho n thứ nhất. Lúc này biến n trong thân hàm được xem là n thứ nhất. Do n thứ nhất có giá trị bằng 3 nên điều kiện trong toán tử if là sai và do đó máy sẽ lựa chọn câu lệnh else. Theo câu lệnh này, máy sẽ tính giá trị biểu thức :

n*gtdq(n-1) (*)

Để tính biểu thức trên, máy cần gọi chính hàm gtdq vì thế lần gọi thứ hai sẽ thực hiện. Máy sẽ tạo ra đối n mới, ta gọi đó là n thứ hai. Giá trị của n-1 ở đây lại là đối của hàm , được truyền cho hàm và hiểu là n thứ hai, do vậy n thứ hai có giá trị là 2. Bây giờ, do n thứ hai vẫn chưa thoả mãn điều kiện if nên máy lại tiếp tục tính biểu thức :

n*gtdq(n-1) (**)

Biểu thức trên lại gọi hàm gtdq lần thứ ba. Máy lại tạo ra đối n lần thứ ba và ở đây n thứ ba có giá trị bằng 1. Đối n=1 thứ ba lại được truyền cho hàm, lúc này điều kiện trong lệnh if được thoả mãn, máy đi thực hiện câu lệnh :

return 1=gtdq(1) (***)

Bắt đầu từ đây, máy sẽ thực hiện ba lần ra khỏi hàm gtdq. Lần ra khỏi hàm thứ nhất ứng với lần vào thứ ba. Kết quả là đối n thứ ba được giải phóng, hàm gtdq(1) cho giá trị là 1 và máy trở về xét giá trị biểu thức

n*gtdq(1) đây là kết quả của (**)

ở đây, n là n thứ hai và có giá trị bằng 2. Theo câu lệnh return, máy sẽ thực hiện lần ra khỏi hàm lần thứ hai, đối n thứ hai sẽ được giải phóng, kết quả là biểu thức trong (**) có giá trị là 2.1. Sau đó máy trở về biểu thức (*) lúc này là :

n*gtdq(2)=n*2*1

n lại hiểu là thứ nhất, nó có giá trị bằng 3, do vậy giá trị của biểu thức trong (*) là 3.2.1=6. Chính giá trị này được sử dụng trong câu lệnh printf của hàm main() nên kết quả in ra trên màn hình là :

3!=6

Chú ý :

Hàm đệ qui so với hàm có thể dùng vòng lặp thì đơn giản hơn, tuy nhiên với máy tính khi dùng hàm đệ qui sẽ dùng nhiều bộ nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy khi gặp một bài toán mà có thể có cách giải lặp (không dùng đệ qui) thì ta nên dùng cách lặp này. Song vẫn tồn tại những bài toán chỉ có thể giải bằng đệ qui.

6.3.2. Các bài toán có thể dùng đệ qui :

Phương pháp đệ qui thường áp dụng cho các bài toán phụ thuộc tham số có hai đặc điểm sau :

Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Người ta thường gọi là trường hợp suy biến.

Trong trường hợp tổng quát, bài toán có thể qui về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Sau một số hữu hạn bước biến đổi dệ qui nó sẽ dẫn tới trường hợp suy biến.

Bài toán tính n giai thừa nêu trên thể hiện rõ nét đặc điểu này.

6.3.3. Cách xây dựng hàm đệ qui :

Hàm đệ qui thường được xây dựng theo thuật toán sau :

```
if ( trường hợp suy biến)
        {
                Trình bày cách giải bài toán khi suy biến
        }
else /* Trường hợp tổng quát */
        {
                Gọi đệ qui tới hàm ( đang viết ) với các giá
                trị khác của tham số
        }
```

6.3.4. Các ví dụ về dùng hàm đệ qui :

Ví dụ 1 :

Bài toán dùng đệ qui tìm USCLN của hai số nguyên dương a và b.

Trong trường hợp suy biến, khi a=b thì USCLN của a và b chính là giá trị của chúng.

Trong trường hợp chung :

uscln(a,b)=uscln(a-b,b) nếu a>b

uscln(a,b)=uscln(a,b-a) nếu a<b

Ta có thể viết chương trình như sau :

```
#include "stdio.h"
int uscln(int a,int b ); /* Nguyên mẫu hàm*/
main()
        {       int m,n;
                printf("\n Nhap cac gia tri cua a va b :");
                scanf("%d%d",&m,&n);
                printf("\n USCLN cua a=%d va b=%d la :%d",m,m,uscln(m,n))
```

```
        }
int uscln(int a,int b)
        {
                if (a==b)
                        return a;
                else
                  if (a>b)
                        return uscln(a-b,b);

                  else
                        return uscln(a,b-a);
        }
```

Ví dụ 2 :

Chương trình đọc vào một số rồi in nó ra dưới dạng các ký tự liên tiếp.

```
# include "stdio.h"
# include "conio.h"
void prind(int n);
main()
{
  int a;
  clrscr();
  printf("n=");
  scanf("%d",&a);
  prind(a);
  getch();
}
void prind(int n)
 {
  int i;
  if (n<0)
  { putchar('-');
    n=-n;
```

```
 }
 if ((i=n/10)!=0)
 prind(i);
 putchar(n%10+'0');
}
```

6.4. Bộ tiền sử lý C :

C đưa ra một số cách mở rộng ngôn ngữ bằng các bộ tiền sử lý macro đơn giản. Có hai cách mở rộng chính là #define mà ta đã học và khả năng bao hàm nội dung của các file khác vào file đang được dịch.

Bao hàm file :

Để dễ dàng xử lý một tập các #define và khai báo (trong các đối tượng khác), C đưa ra cách bao hàm các file khác vào file đang dịch có dạng :

#include "tên file"

Dòng khai báo trên sẽ được thay thế bởi nội dung của file có tên là tên file. Thông thường có vài dòng như vậy xuất hiện tại đầu mỗi file gốc để gọi vào các câu lệnh #define chung và các khai báo cho các biến ngoài. Các #include được phép lồng nhau. Thường thì các #include được dùng nhiều trong các chương trình lớn, nó đảm bảo rằng mọi file gốc đều được cung cấp cùng các định nghĩa và khai báo biến, do vậy tránh được các lỗi khó chịu do việc thiếu các khai báo định nghĩa. Tất nhiên khi thay đổi file được bao hàm vào thì mọi file phụ thuộc vào nó đều phải dịch lại.

Phép thế MACRO :

Định nghĩa có dạng :

#define biểu thức 1 [biểu thức 2]

sẽ gọi tới một macro để thay thế biểu thức 2 (nếu có) cho biểu thức 1.

Ví dụ :

#define YES 1

Macro thay biến YES bởi giá trị 1 có nghĩa là hễ có chỗ nào trong chương trình có xuất hiện biến YES thì nó sẽ được thay bởi giá trị 1.

Phạm vi cho tên được định nghĩa bởi #define là từ điểm định nghĩa đến cuối file gốc. Có thể định nghĩa lại tên và một định nghĩa có thể sử dụng các định nghĩa khác trước đó. Phép thế

không thực hiện cho các xâu dấu nháy, ví dụ như YES là tên được định nghĩa thì không có việc thay thế nào được thực hiện trong đoạn lệnh có "YES".

Vì việc thiết lập #define là một bước chuẩn bị chứ không phải là một phần của chương trình biên dịch nên có rất ít hạn chế về văn phạm về việc phải định nghĩa cái gì. Chẳng hạn như những người lập trình ưa thích PASCAL có thể định nghĩa :

```
#define then
#define begin {
#define end; }
```

sau đó viết đoạn chương trình :

```
if (i>0)  then
        begin
                a=i;
                .....
        end;
```

Ta cũng có thể định nghĩa các macro có đối, do vậy văn bản thay thế sẽ phụ thuộc vào cách gọi tới macro.

Ví dụ :

Định nghĩa macro gọi max như sau :

```
#define max(a,b) ((a)>(b) ?(a):(b))
```

Việc sử dụng :

```
x=max(p+q,r+s);
```

tương đương với :

```
x=((p+q)>(r+s) ? (p+q):(r+s));
```

Như vậy ta có thể có hàm tính cực đại viết trên một dòng. Chừng nào các đối còn giữ được tính nhất quán thì macro này vẫn có giá trị với mọi kiểu dữ liệu, không cần phải có các loại hàm max khác cho các kiểu dữ liệu khác nhưng vẫn phải có đối cho các hàm.

Tất nhiên nếu ta kiểm tra lại việc mở rộng của hàm max trên, ta sẽ thấy rằng nó có thể gây ra số bẫy. Biểu thức đã được tính lại hai lần và điều này là không tốt nếu nó gây ra hiệu quả phụ kiểu như các lời gọi hàm và toán tử tăng. Cần phải thận trọng dùng thêm dấu ngoặc để đảm bảo trật tự tính toán. Tuy vậy, macro vẫn rất có giá trị.

Chú ý :

Không được viết dấu cách giữa tên macro với dấu mở ngoặc bao quanh danh sách đối.

Ví dụ :

Xét chương trình sau :

```
main()
        {
                int x,y,z;
                x=5;
                y=10*5;
                z=x+y;
                z=x+y+6;
                z=5*x+y;
                z=5*(x+y);
                z=5*((x)+(y));
                printf("Z=%d",z);
                getch();
                return;
        }
```

Chương trình sử dụng MACRO sẽ như sau :

```
#define BEGIN {
#define END }
#define INTEGER int
#define NB 10
#define LIMIT NB*5
#define SUMXY x+y
#define SUM1 (x+y)
#define SUM2 ((x)+(y))
main()
        BEGIN
                INTEGER x,y,z;
                x=5;
                y=LIMIT;
```

```
        z=SUMXY;
        z=5*SUMXY;
        z=5*SUM1;
        z=5*SUM2;
        printf("\n Z=%d",z);
        getch();
        return;
END
```

Chương 7

CON TRỎ

Con trỏ là biến chứa địa chỉ của một biến khác. Con trỏ được sử dụng rất nhiều trong C, một phần là do chúng đôi khi là cách duy nhất để biểu diễn tính toán, và phần nữa do chúng thường làm cho chương trình ngắn gọn và có hiệu quả hơn các cách khác .

Con trỏ đã từng bị coi như có hại chẳng kém gì lệnh goto do cách sử dụng chúng đã tạo ra các chương trình khó hiểu. Điều này chắc chắn là đúng khi người ta sử dụng chúng một cách lôn xộn và do đó tạo ra các con trỏ trỏ đến đâu đó không biết trước được.

7.1. Con trỏ và địa chỉ :

Vì con trỏ chứa địa chỉ của đối tượng nên nó có thể xâm nhập vào đối tượng gián tiếp qua con trỏ. Giả sử x là một biến kiểu int, và giả sử px là con trỏ được tạo ra theo một cách nào đó. Phép toán một ngôi & sẽ cho địa chỉ của đối tượng, nên câu lệnh :

```
px=&x;
```

sẽ gán địa chỉ của biến x cho trỏ px, và px bây giờ được gọi là " trỏ tới biến x ". Phép toán & chỉ áp dụng được cho các biến và phần tử bảng, kết cấu kiểu &(x+1) và &3 là không hợp lệ. Lấy đại chỉ của biến register cũng là sai.

Phép toán một ngôi * coi là toán hạng của nó là đại chỉ cần xét và thâm nhập tới địa chỉ đó để lấy ra nội dung. Nếu biến y có kiểu int thì thì lệnh :

```
y=*px;
```

sẽ gán giá trị của biến mà trỏ px trỏ tới. Vậy dãy lệnh :

```
px=&x;
y=*px;
```

sẽ gán giá trị của x cho y như trong lệnh :

```
y=x;
```

Các khai báo cho các biến con trỏ có dạng :

```
tên kiểu *tên con trỏ
```

Ví dụ :

Như trong ví dụ trên, ta khai báo con trỏ px kiểu int :

```
int *px;
```

Trong khai báo trên ta đã ngụ ý nói rằng đó là một cách tượng trưng, rằng tổ hợp *px có kiểu int, tức là nếu px xuất hiện trong ngữ cảnh *px thì nó cũng tương đương với biến có kiểu int.

Con trỏ có thể xuất hiện trong các biểu thức. Chẳng hạn, nếu px trỏ tới số nguyên x thì *px có thể xuất hiện trong bất kỳ ngữ cảnh nào mà x có thể xuất hiện.

Ví dụ :

Lệnh y=*px+1;

sẽ đặt y lớn hơn x một đơn vị.

Lệnh printf("%d",*px);

sẽ in ra giá trị hiện tại của x

Lệnh :

d=sqrt((double) *px);

sẽ gán cho biến d căn bậc hai của x, giá trị này bị buộc phải chuyển sang double trước khi được chuyền cho sqrt (cách dùng hàm sqrt).

Trong các biểu thức kiểu như :

y=*px+1;

phép toán một ngôi * và & có mức ưu tiên cao hơn các phép toán số học, cho nên biểu thức này lấy bất ký giá trị nào mà px trỏ tới, cộng với 1 rồi gán cho y.

Con trỏ cũng có thể xuất hiện bên vế trái của phép gán. Nếu px trỏ tới x thì sau lệnh :

*px=0;

x sẽ có giá trị bằng 0. Cũng tương tự các lệnh:

*px+=1;

(*px)++;

sẽ tăng giá trị của x lên 1 đơn vị.

Các dấu ngoặc đơn ở câu lệnh cuối là cần thiết , nếu không thì biểu thức sẽ tăng px thay cho tăng ở biến mà nó trỏ tới vì phép toán một ngôi như * và ++ được tính từ phải sang trái.

Cuối cùng, vì con trỏ là biến nên ta có thao tác chúng như đối với các biến khác. Nếu py cũng là con trỏ int thì lệnh :

py=px;

sẽ sao nội dung của px vào py, nghĩa là làm cho py trỏ tới nơi mà px trỏ.

7.2. Con trỏ và mảng một chiều :

Trong C có mối quan hệ chặt chẽ giữa con trỏ và mảng : các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ.

7.2.1.Phép toán lấy địa chỉ :

Phép toán này chỉ áp dụng cho các phần tử của mảng một chiều. Giả sử ta có khai báo :

```
double b[20];
```

Khi đó phép toán :

```
&b[9]
```

sẽ cho địa chỉ của phần tử b[9].

7.2.2. Tên mảng là một hằng địa chỉ :

Khi khai báo :

```
float a[10];
```

máy sẽ bố trí bố trí cho mảng a mười khoảng nhớ liên tiếp, mỗi khoảng nhớ là 4 byte. Như vậy, nếu biết địa chỉ của một phần tử nào đó của mảng a, thì ta có thể dễ dàng suy ra địa chỉ của các phần tử khác của mảng.

Với C ta có :

a tương đương với &a[0]

a+i tương đương với &a[i]

*(a+i) tương đương với a[i]

7.2.3. Con trỏ trỏ tới các phần tử của mảng một chiều :

Khi con trỏ pa trỏ tới phần tử a[k] thì :

pa+i trỏ tới phần tử thứ i sau a[k], có nghĩa là nó trỏ tới a[k+i].

pa-i trỏ tới phần tử thứ i trước a[k], có nghĩa là nó trỏ tới a[k-i].

*(pa+i) tương đương với pa[i].

Như vậy, sau hai câu lệnh :

```
float a[20],*p;
p=a;
```

thì bốn cách viết sau có tác dụng như nhau :

```
a[i]    *(a+i)    p[i]    *(p+i)
```

Ví dụ :

Vào số liệu của các phần tử của một mảng và tính tổng của chúng :

Cách 1:

```
#include "stdio.h"
main()
	{
		float a[4],tong;
		int i;
		for (i=0;i<4;++i)
		 {
			printf("\n a[%d]=",i);
			scanf("%f",a+i);
		 }
		tong=0;
		for (i=0;i<4;++i)
			tong+=a[i];
		printf("\n Tong cac phan tu mang la :%8.2f ",tong);
	}
```

Cách 2 :

```
#include "stdio.h"
main()
	{
		float a[4],tong, *troa;
		int i;
		troa=a;
		for (i=0;i<4;++i)
		 {
			printf("\n a[%d]=",i);
			scanf("%f",&troa[i]);
		 }
		tong=0;
		for (i=0;i<4;++i)
```

```
                tong+=troa[i];
        printf("\n Tong cac phan tu mang la :%8.2f ",tong);
}
```

Cách 3 :

```
#include "stdio.h"
main()
{
        float a[4],tong,*troa;
        int i;
        troa=a;
        for (i=0;i<4;++i)
         {
                printf("\n a[%d]=",i);
                scanf("%f",troa+i);
         }
        tong=0;
        for (i=0;i<4;++i)
                tong+=*(troa+i);
        printf("\n Tong cac phan tu mang la :%8.2f ",tong);
}
```

Chú ý :

Mảng một chiều và con trỏ tương ứng phải cùng kiểu.

7.2.4. Mảng, con trỏ và xâu ký tự :

Như ta đã biết trước đây, xâu ký tự là một dãy ký tự đặt trong hai dấu nháy kép, ví dụ như :

```
"Viet nam"
```

Khi gặp một xâu ký tự, máy sẽ cấp phát một khoảng nhớ cho một mảng kiểu char đủ lớn để chứa các ký tự của xâu và chứa thêm ký tự '\0' là ký tự dùng làm ký tự kết thúc của một xâu ký tự. Mỗi ký tự của xâu được chứa trong một phần tử của mảng.

Cũng giống như tên mảng, xâu ký tự là một hàng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo biến **xau** như một con trỏ kiểu char :

```
char *xau;
```

thì phép gán :

```
xau="Ha noi"
```

là hoàn toàn có nghĩa. Sau khi thực hiện câu lệnh này trong con trỏ **xau** sẽ có địa chỉ đầu của mảng (kiểu char) đang chứa xâu ký tự bên phải. Khi đó các câu lệnh :

```
puts("Ha noi");
puts(xau);
```

sẽ có cùng một tác dụng là cho hiện lên màn hình dòng chữ **Ha noi.**

Mảng kiểu char thường dùng để chứa một dãy ký tự đọc vào bộ nhớ. Ví dụ, để nạp từ bàn phím tên của một người ta dùng một mảng kiểu char với độ dài 25, ta sử dụng các câu lệnh sau :

```
char ten[25];
printf("\n Ho ten :");
gets(ten);
```

Bây giờ ta xem giữa mảng kiểu char và con trỏ kiểu char có những gì giống và khác nhau. Để thấy được sự khác nhau của chúng, ta đưa ra sự so sánh sau :

```
char *xau, ten[15];
ten="Ha noi"
gets(xau);
```

Các câu lệnh trên là không hợp lệ. Câu lệnh thứ hai sai ở chỗ : ten là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác. Câu lệnh thứ ba không thực hiện được, mục đích của câu lệnh là đọc từ bàn phím một dãy ký tự và lưu vào một vùng nhớ mà con trỏ **xau** trỏ tới. Song nội dung của con trỏ **xau** còn chưa xác định. Nếu trỏ **xau** đã trỏ tới một vùng nhớ nào đó thì câu lệnh này hoàn toàn có ý nghĩa. Chẳng hạn như sau khi thực hiện câu lệnh :

```
xau=ten;
```

thì cách viết :

gets(ten) ; và gets(xau);

đều có tác dụng như nhau.

7.3. Con trỏ và mảng nhiều chiều :

Việc sử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều. Không phải mọi qui tắc đúng với mảng một chiều đều có thể áp dụng cho mảng nhiều chiều.

7.3.1.Phép lấy địa chỉ :

Phép lấy địa chỉ đối với các phần tử mảng hai chiều chỉ có thể áp dụng khi các phần tử mảng hai chiều có kiểu nguyên, còn lại thì phép lấy địa chỉ cho các phần tử mảng nhiều chiều là không thực hiện được .Ví dụ như ta có thể lấy địa chỉ &a[1][2] khi a là mảng nguyên.

Thủ thuật đọc từ bàn phím phần tử mảng hai chiều dùng lệnh scanf :

Chương trình đọc vào số liệu cho một ma trận hai chiều sẽ được thực hiện thông qua việc đọc vào một biến trung gian, đọc một giá trị và chứa tạm vào một biến trung gian sau đó ta gán biến cho phần tử mảng:

```
#include "stdio.h"
main()
 {
        float a[2][3], tg;
        int i,j;
        for (i=0;i<2;++i)
        for (j=0;j<2;++j)
          {
                printf("\n a[%d][%d]=",i,j);
                scanf("%8.2f",&tg);
                a[i][j]=tg;
          }
 }
```

7.3.2. Phép cộng địa chỉ trong mảng hai chiều:

Giả sử ta có mảng hai chiều a[2][3] có 6 phần tử úng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự sau :

Phần tử	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
Địa chỉ	1	2	3	4	5	6

Tên mảng a biểu thị địa chỉ đầu tiên của mảng. Phép cộng địa chỉ ở đây được thực hiện như sau :

C coi mảng hai chiều là mảng (một chiều) của mảng, như vậy khai báo

```
float a[2][3];
```

thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của mảng).

Vì vậy :

a trỏ phần tử thứ nhất của mảng : phần tử a[0][0]

a+1 trỏ phần tử đầu hàng thứ hai của mảng : phần tử a[1][0]

.....

7.3.3. Con trỏ và mảng hai chiều :

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta có thể dùng con trỏ như minh hoạ ở ví dụ sau :

```
float *pa,a[2][3];
pa=(float*)a;
```

lúc đó :

pa trỏ tới a[0][0]

pa+1 trỏ tới a[0][1]

pa+2 trỏ tới a[0][2]

pa+3 trỏ tới a[1][0]

pa+4 trỏ tới a[1][1]

pa+5 trỏ tới a[1][2]

Ví dụ :

Dùng con trỏ để vào số liệu cho mảng hai chiều.

Cách 1 :

```
#include "stdio.h"
main()
    {
        float a[2][3],*pa;
        int i;
        pa=(float*)a;
        for (i=0;i<6;++i)
        scanf("%f",pa+i);
    }
```

Cách 2 :

```
#include "stdio.h"
main()
        {
                float a[2][3],*pa;
                int i;
                for (i=0;i<6;++i)
                scanf("%f",(float*)a+i);
        }
```

7.4. Kiểu con trỏ, kiểu địa chỉ, các phép toán trên con trỏ :

7.4.1. Kiểu con trỏ và kiểu địa chỉ :

Con trỏ dùng để lưu địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán địa chỉ cho con trỏ chỉ có thể thực hiện được khi kiểu địa chỉ phù hợp với kiểu con trỏ.

Ví dụ theo khai báo :

```
float a[20][30],*pa,(*pm)[30];
```

thì :

pa là con trỏ float

pm là con trỏ kiểu float [30]

a là địa chỉ kiểu float [30]

Vì thế phép gán :

```
pa=a;
```

là không hợp lệ. Nhưng phép gán :

```
pm=a;
```

7.4.2. Các phép toán trên con trỏ:

Có 4 phép toán liên quan đến con trỏ và đại chỉ là :

Phép gán.

Phép tăng giảm địa chỉ.

Phép truy cập bộ nhớ.

Phép so sánh.

Phép gán :

Phép gán chỉ thực hiện với các con trỏ cùng kiểu. Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu như ví dụ sau :

```
int x;
char *pc;
pc=(char*)(&x);
```

Phép tăng giảm địa chỉ :

Để minh hoạ chi tiết cho phép toán này, ta xét ví dụ sau :

Các câu lệnh :

```
float x[30],*px;
px=&x[10];
```

cho con trỏ px là con trỏ float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ 4 byte, nên các phép tăng giảm địa chỉ được thực hiện trên 4 byte. Vì thế :

px+i trỏ tới phần tử x[10+i]

px-i trỏ tới phần tử x[10-i]

Xét ví dụ khác :

Giả sử ta khai báo :

```
float b[40][50];
```

Khai báo trên cho ta một mảng b gồm các dòng 50 phần tử thực. Kiểu địa chỉ của b là 50*4=200 byte.

Do vậy :

b trỏ tới đầu dòng thứ nhất (phần tử b[0][0]).

b+1 trỏ tới đầu dòng thứ hai (phần tử b[1][0]).

.....

b+i trỏ tới đầu dòng thứ i (phần tử b[i][0]).

Phép truy cập bộ nhớ :

Con trỏ float truy nhập tới 4 byte, con trỏ int truy nhập 2 byte, con trỏ char truy nhập 1 byte. Giả sử ta có cá khai báo :

```
float *pf;
int *pi;
```

```
char *pc;
```

Khi đó :

Nếu trỏ pi trỏ đến byte thứ 100 thì *pf biểu thị vùng nhớ 4 byte liên tiếp từ byte 100 đến 103.

Nếu trỏ pi trỏ đến byte thứ 100 thì *pi biểu thị vùng nhớ 2 byte liên tiếp từ byte 100 đến 101.

Nếu trỏ pc trỏ đến byte thứ 100 thì *pc biểu thị vùng nhớ 1 byte chính là byte 100.

Phép so sánh :

Cho phép so sánh các con trỏ cùng kiểu, ví dụ nếu p1 và p2 là các con trỏ cùng kiểu thì nếu :

p1<p2 nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới.

p1=p2 nếu địa chỉ p1 trỏ tới cũng là địa chỉ p2 trỏ tới.

p1>p2 nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

Ví dụ :

Ví dụ 1 :

Đoạn chương trình tính tổng các số thực dùng phép so sánh con trỏ :

```
float a[100],*p,*pcuoi,tong=0.0;
int n;
pcuoi=a+n-1;    /* Địa chỉ cuối dãy*/
for (p=a;p<=pcuoi;++p)
        s+=*p;
```

Ví dụ 2 :

Dùng con trỏ char để tách các byte của một biến nguyên, ta làm như sau :

Giả sử ta có lệnh :

```
unsigned int n=0xABCD; /* Số nguyên hệ 16*/
char *pc;
pc=(char*)(&n);
```

Khi đó :

*pc=0xAB (byte thứ nhất của n)

*pc+1=0xCD (byte thứ hai của n)

7.4.3. Con trỏ kiểu void :

Con trỏ kiểu void được khai báo như sau :

void *tên_con_trỏ;

Đây là con trỏ đặc biệt, con trỏ không kiểu, nó có thể nhận bất kỳ kiểu nào. Chẳng hạn câu lệnh sau là hợp lệ :

```
void *pa;
float a[20][30];
pa=a;
```

Con trỏ void thường dùng làm đối để nhận bất kỳ địa chỉ kiểu nào từ tham số thực. Trong thân hàm phải dùng phép chuyển đổi kiểu để chuyển sang dạng địa chỉ cần sử lý.

Chú ý :

Các phép toán tăng giảm địa chỉ, so sánh và truy cập bộ nhớ không dùng được trên con trỏ void.

Ví dụ :

Viết hàm thực hiện công ma trận :

```
void congmt(void *a,void *b,void *c,int N,int N, int m);
        {
                float *pa,*pb,*pc;
                int i,j;
                pa=(float*)a;
                pb=(float*)b;
                pc=(float*)c;
                for (i=1;i<m;++i)
                for (j=1;j<m;++j)
                *(pc+i*N+j)=*(pa+i*N+j)+*(pb+i*N+j);
```

}

Vì đối là con trỏ void nên nó có thể nhận được địa chỉ của các ma trận trong lời gọi hàm. Tuy nhiên ta không thể sử dụng trực tiếp các đối con trỏ void trong thân hàm mà phải chuyển kiểu của chúng sang thành float.

7.5. Mảng con trỏ :

Mảng con trỏ là sự mở rộng khái niệm con trỏ. Mảng con trỏ là một mảng mà mỗi phần tử của nó chứa được một địa chỉ nào đó. Cũng giống như con trỏ, mảng con trỏ có nhiều kiểu : Mỗi phần tử của mảng con trỏ kiểu int sẽ chứa được các địa chỉ kiểu int. Tương tự cho các mảng con trỏ của các kiểu khác.

Mảng con trỏ được khai báo theo mẫu :

Kiểu *Tên_mảng_con_trỏ[N];

Trong đó **Kiểu** có thể là int, float, double, char ... còn **Tên_mảng_con_trỏ** là tên của mảng, N là một hằng số nguyên xác định độ lớn của mảng.

Khi gặp khai báo trên, máy sẽ cấp phát N khoảng nhớ liên tiếp cho N phần tử của mảng **Tên_mảng_con_trỏ**.

Ví dụ :

Lệnh :

double *pa[100];

Khai báo một mảng con trỏ kiểu double gồm 100 phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double.

Chú ý :

Bản thân các mảng con trỏ không dùng để lưu trữ số liệu. Tuy nhiên mảng con trỏ cho phép sử dụng các mảng khác để lưu trữ số liệu một cách có hiệu quả hơn theo cách : chia mảng thành các phần và ghi nhớ địa chỉ đầu của mỗi phần vào một phần tử của mảng con trỏ.

Trước khi sử dụng một mảng con trỏ ta cần gán cho mỗi phần tử của nó một giá trị. Giá trị này phải là giá trị của một biến hoặc một phần tử mảng. Các phần tử của mảng con trỏ kiểu char có thể được khởi đầu bằng các xâu ký tự.

Ví dụ :

Xét một tổ lao động có 10 người, mã của mỗi người chính là số thứ tự. Ta lập một hàm để khi biết mã số của nhân viên thì xác định được họ tên của nhân viên đó.

```
#include "stdio.h"
#include "ctype.h"
void tim(int code);
main()
        {
                int i;
                tt:printf("\n Tim nguoi co so TT la :");
                scanf("%d",&i);
                tim(i);
                printf("Co tiep tuc nua khong C/K : ');
                if (tupper(getch())='C')
                        goto tt;
        }
void tim(int code);
        {
                static char *list[]= {
                                        "Khong co so thu tu nay "
                                        " Nguyen Van Toan"
                                        "Huynh Tuan Nghia"
                                        "Le Hong Son"
                                        "Tran Quang Tung"
                                        "Chu Thanh Tu"
                                        "Mac Thi Nga"
                                        "Hoang Hung"
                                        "Pham Trong Ha"
                                        "Vu Trung Duc"
                                        "Mai Trong Quat"
                                    };
                printf("\n\n Ma so : %d",code);
                printf(": %s",());
        }
```

7.6. Con trỏ tới hàm :

7.6.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm :

Ta sẽ trình bày quy tắc khai báo thông qua các ví dụ :

Ví dụ 1:

Câu lệnh :

```
float (*f)(float),(*mf[50])(int);
```

Để khai báo :

- f là con trỏ hàm kiểu float có đối là float
- mf là mảng con trỏ hàm kiểu float có đối kiểu int (có 50 phần tử)

Ví dụ 2:

Câu lệnh :

```
double (*g)(int, double),(*mg[30])(double, float);
```

Để khai báo :

- g là con trỏ hàm kiểu double có các đối kiểu int và double
- mg là mảng con trỏ hàm kiểu double có các đối kiểu double và float (có 30 phần tử)

7.6.2. Tác dụng của con trỏ hàm :

Con trỏ hàm dùng để chứa địa chỉ của hàm. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có ý nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm.

Ví dụ 1:

```
#include "stdio.h"
double fmax(double x, double y ) /* Tính max x,y */
{
	return(x>y ? x:y);
}
double (*pf)(double,double)=fmax; /*Khai báo và gán tên hàm cho con trỏ hàm */
```

```
main()  /* Sử dụng con trỏ hàm*/
        {
                printf("\n max=%f",pf(5.0,9.6));
        }
```

Ví dụ 2:

```
#include "stdio.h"
double fmax(double x, double y ) /* Tính max x,y */
{
        return(x>y ? x:y);
}
double (*pf)(double,double); /* Khai báo con trỏ hàm*/

main()  /* Sử dụng con trỏ hàm*/
        {
                pf=fmax;
                printf("\n max=%f",pf(5.0,9.6));

        }
```

7.6.3. Đối của con trỏ hàm :

C cho phép thiết kế các hàm mà tham số thực trong lời gọi tới nó lại là tên của một hàm khác. Khi đó tham số hình thức tương ứng phải là một con trỏ hàm.

Cách dùng con trỏ hàm trong thân hàm :

Nếu đối được khai báo :

```
double (*f)(double, int);
```

thì trong thân hàm ta có thể dùng các cách viết sau để xác định giá trị của hàm (do con trỏ f trỏ tới) :

f(x,m) hoặc (f)(x,m) hoặc (*f)(x,m)

ở đây x là biến kiểu double còn m là biến kiểu int.

Ví dụ :

Dùng mảng con trỏ để lập bảng giá trị cho các hàm : x*x, sin(x), cos(x), exp(x) và sqrt(x). Biến x chay từ 1.0 đến 10.0 theo bước 0.5

```
#include "stdio.h"
#include "math.h"
double bp(double x) /* Hàm tính x*x */
        {
                return x*x;
        }

main()
        {
                int i,j;
                double x=1.0;
                typedef double (*ham)(double);
                ham f[6]; /* Khai bao mảng con trỏ hàm*/
                /* Có thể khai báo như sau double (*f[6](double)*/
                f[1]=bp; f[2]=sin; f[3]=cos; f[4]=exp; f[5]=sqrt;
                /* Gán tên hàm cho các phần tử mẳng con trỏ hàm */
                while (x<=10.0)  /* Lập bảng giá trị */
                        {
                                printf("\n");
                                for (j=1;j<=5;++j)
                                        printf("%10.2f ",f[j](x));
                                x+=0.5;
                        }
        }
```

Chương 8
CẤU TRÚC

Cấu trúc là tập hợp của một hoặc nhiều biến, chúng có thể khác kiểu nhau, được nhóm lại dưới một cái tên duy nhất để tiện sử lý. Cấu trúc còn gọi là bản ghi trong một số ngôn ngữ khác, chẳng hạn như PASCAL.

Cấu trúc giúp cho việc tổ chức các dữ liệu phức tạp, đặc biệt trong những chương trình lớn vì trong nhiều tình huống chúng cho phép nhóm các biến có liên quan lại để xử lý như một đơn vị thay vì các thực thể tách biệt.

Một ví dụ được đề cập nhiều đến là cấu trúc phiếu ghi lương, trong đó mỗi nhân viên được mô tả bởi một tập các thuộc tính chẳng hạn như : tên, địa chỉ, lương, phụ cấp vv.. một số trong các thuộc tính này lại có thể là cấu trúc bởi trong nó có thể chứa nhiều thành phần : Tên (Họ, đệm, tên), Địa chỉ (Phố, số nhà) vv.

Trong chương này chúng ta sẽ minh hoạ cách sử dụng của các cấu trúc trong chương trình.

8.1. Kiểu cấu trúc :

Khi xây dựng cấu trúc, ta cần mô tả kiểu của nó. Điều này cũng tương tự như việc phải thiết kế ra một kiểu nhà trước khi ta đi xây dựng những căn nhà thực sự ở các địa điểm khác nhau. Công việc định nghĩa một kiểu cấu trúc bao gồm việc nêu ra tên của kiểu cấu trúc và các thành phần của nó theo mẫu sau :

```
struct tên_kiểu _cấu_trúc
        {
                Khai báo các thành phần của cấu trúc        (1)
        };
```

Trong đó :

- struct là từ khoá
- tên_kiểu _cấu_trúc là một tên bất kỳ do người lập trình tự đặt theo qui tắc đặt tên nêu ra trong chương 1.

Thành phần của cấu trúc có thể là : biến, mảng, cấu trúc khác đã được định nghĩa trước đó vv..

Ví dụ :

Ví dụ 1:

Đoạn chương trình :

```
struct ngay {
        int ngaythu;
        char thang[12];
        int nam;
};
```

mô tả một kiểu cấu trúc có tên là **ngay** gồm có ba thành phần : Biến nguyên **ngaythu**, mảng **thang**, và biến nguyên **nam**.

Ví dụ 2:

Đoạn chương trình :

```
struct nhancong
        {

                char ten[15];
                char diachi[20]
                double bacluong;
                struc ngay ngaysinh;
                struc ngay ngaybatdaucongtac;
};
```

tạo ra kiểu cấu trúc có tên là **nhancong** gồm có năm thành phần. Ba thành phần đầu không có gì cần nói thêm. Chỉ có hai thành phần còn lại là các cấu trúc **ngaysinh** và **ngaybatdaucongtac** được xây dựng theo cấu trúc **ngay** được định nghĩa trong ví dụ 1.

Định nghĩa cấu trúc bằng typedef :

Có thể dùng toán tử typedef để định nghĩa các kiểu cấu trúc **ngay** và **nhancong** ở trên như sau :

```
typedef struct
        {
                int ngaythu;
                char thang[12];
```

```
                int nam;

        } ngay;
typedef struct
        {

                char ten[15];
                char diachi[20]
                double bacluong;
                struc ngay ngaysinh;
                struc ngay ngaybatdaucongtac;
        } nhancong;
```

8.2. Khai báo theomột kiểu cấu trúc đã định nghĩa :

Xây dựng những cấu trúc thực sự theo các kiểu đã khai báo trước đó. Vấn đề này hoàn toàn giống như việc khai báo các biến và các mảng. Giả sử ta đã có các kiểu cấu trúc **ngay** và **nhancong** như trong mục trên. Khi đó ta khai báo :

Ví dụ 1 :

```
struct ngay ngaydi, ngayden;
```

sẽ cho ta hai cấu trúc với tên là **ngaydi** và **ngayden**. Cả hai cấu trúc đều được xây dựng theo cấu trúc kiểu **ngay**.

Ví dụ 2 :

```
struct nhancong nhom1,nhom2;
```

sẽ cho ta hai cấu trúc với tên là **nhom1** và **nhom2**. Cả hai cấu trúc đều được xây dựng theo cấu trúc kiểu **nhancong**.

Như vậy, một cách tổng quát, việc khai báo cấu trúc được thực hiện theo mẫu sau :

Cách 1 :

```
struct tên_kiểu_cấu_trúc_đã_khai_báo danh_sách_tên_các_cấu_trúc; (2)
```

Chú ý :

Các biến cấu trúc được khai báo theo mẫu trên sẽ được cấp phát bộ nhớ một cách đầy đủ cho tất cả các thành phần của nó.

Việc khai báo có thể thực hiện đồng thời với việc định nghĩa kiểu cấu trúc. Muốn vậy, chỉ cần đặt danh sách tên biến cấu trúc cần khai báo sau dấu } của (*) như trên .

Nói cách khác, để vừa khai báo kiểu vừa khai báo biến ta dùng cách sau :

Cách 2 :

```
struct tên_kiểu_cấu_trúc
        {
                Các thành phần của cấu trúc    (3)
        } danh_sách_tên_các_cấu_trúc;
```

Ví dụ :

Ví dụ 1 :

```
struct ngay
        {
                int ngaythu;
                char thang[12];
                int nam;
        } ngaydi,ngayden;
```

Ví dụ 2 :

```
struct nhancong
        {

                char ten[15];
                char diachi[20];
                double bacluong;
                struc ngay ngaysinh;
                struc ngay ngaybatdaucongtac;

        } nhom1,nhom2;
```

Khi vừa định nghĩa kiểu cấu trúc vừa khai báo cấu trúc như trong ví dụ trên, ta không thể không cần đến tên kiểu cấu trúc. Nói cách khác cấu trúc có thể được khai báo theo cách sau :

```
struct
        {
                Các thành phần của cấu trúc    (4)
        } danh_sách_tên_các_cấu_trúc;
```

Ví dụ :

```
struct
        {
                int ngaythu;
                char thang[12];
               int nam;
        } ngaydi,ngayden;
```

Sự khác nhau của các cách khai báo cấu trúc trong (3) và (4) là ở chỗ : Với (3) ta vừa khai báo được một kiểu cấu trúc vừa khai báo được các cấu trúc, và có thể dùng kiểu cấu trúc này để khai báo cho các cấu trúc khác như trong (2), còn (4) chỉ khai báo được các cấu trúc.

Chú ý :

Nếu dùng từ khoá typedef để định nghĩa kiểu cấu trúc như trong mục 8.1 thì khi khai báo các cấu trúc mới ta không cần dùng từ khoá struct, chỉ cần dùng tên kiểu.

Ví dụ như kiểu cấu trúc **ngay** được khai báo bằng typedef trong 8.1 thì khi khai báo các cấu trúc mới là **ngaydi** và **ngayden** có cùng kiểu **ngay** ta dùng dòng lệnh sau :

```
ngay ngaydi,ngayden;
```

8.3. Truy nhập đến các thành phần cấu trúc :

Ta đã khá quen với việc sử dụng các biến, các phần tử của mảng và tên mảng trong các câu lệnh. Trên đây ta cũng đã đề cập đến các thành phần của cấu trúc là biến và mảng. Việc xử lý một cấu trúc bao giờ cũng phải được thực hiện thông qua các thành phần của nó.

Để truy cập đến một thành phần cơ bản (là biến hoặc mảng) của một cấu trúc ta sử dụng một trong các cách viết sau :

tên_cấu_trúc.tên_thành_phần

tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần

tên_cấu_trúc. tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần

.....

Cách viết thứ nhất như trên được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc. Ví dụ như biến **ngaythu,** biến **nam** và mảng **thang** là các thành phần trực tiếp của các cấu trúc **ngaydi, ngayden**. Các biến **bacluong**, các mảng **ten, diachi** là các thành phần trực tiếp của các cấu trúc **nhancong**.

Các cách viết còn lại như trên được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc mà bản thân cấu trúc này lại là thành phần của các cấu trúc lớn hơn.

Ví dụ :

Ta xét phép toán trên các thành phần của cấu trúc **nhom1**, **nhom2** :

Câu lệnh :

```
printf("%s",nhom1.ten);
```

sẽ đưa lên màn hình tên của nhom1.

Câu lệnh :

```
tongluong=nhom1.bacluong+nhom2.bacluong;
```

sẽ gán tổng lương của **nhom1** và **nhom2** rồi gán cho biến **tongluong**.

Câu lệnh :

```
printf("%d",nhom1.ngaysinh.ten);
```

sẽ đưa lên màn hình ngày sinh của nhom1.

Câu lệnh :

```
printf("%d",nhom1. ngaybatdaucongtac.nam);
```

sẽ đưa lên màn hình ngày bắt đầu công tác của nhom1.

Chú ý :

- Có thể sử dụng phép toán lấy địa chỉ đối với các thành phần cấu trúc để nhập số liệu trực tiếp vào các thành phần cấu trúc. Ví dụ như ta viết :

```
scanf("%d",&nhom1. ngaybatdaucongtac.nam);
```

Nhưng đối với các thành phần không nguyên, việc làm trên có thể dẫn đến treo máy. Vì thế nên nhập số liệu vào một biến trung gian sau đó mới gán cho thành phần của cấu trúc.

Cách làm như sau :

```
int year;
scanf("%d",&year);
nhom1. ngaybatdaucongtac.nam=year;
```

- Để tránh dài dòng khi làm việc với các thành phần cấu trúc ta có thể dùng lệnh #define. Ví dụ trong câu lênh scanf ở ví dụ trên, ta có thể viết như sau :

```
#define p nhom1. ngaybatdaucongtac
.....
scanf("%d",&p.nam);
```

Ví dụ :

Giả sử ta lập trình quản lý thông tin cán bộ. Giả sử mỗi dữ liệu của một cán bộ gồm :

- Ngày tháng năm sinh.
- Ngày tháng năm vào cơ quan.
- Bậc lương.

Yêu cầu viết một chương trình để :

- Xây dựng cấu trúc cơ sở dữ liệu cho cán bộ.
- Vào số lệu của một cán bộ.
- Đưa số liệu đó ra máy in.

Chương trình được viết như sau :

```
#include "stdio.h"
typedef struct
        {
                int ngay;
                char thang[10];
                int nam;
        } date;
typedef struct
                {
```

```
                    date ngaysinh;
                    date ngayvaocq;
                    float luong;
                } canbo;
main()
    {
        canbo p;
        printf("\n Sinh ngay : ");
        scanf("%d",&p.ngaysinh.ngay);
        printf("\n Thang : ");
        scanf("%d",&p.ngaysinh.thang);
        printf("\n Nam : ");
        scanf("%d",&p.ngaysinh.nam);
        printf("\n Vao co quan ngay : ");
        scanf("%d",&p.ngayvaocq.ngay);
        printf("\n Thang : ");
        scanf("%d",&p.ngayvaocq.thang);
        printf("\n Nam : ");
        scanf("%d",&p.ngayvaocq.nam);
        printf("\n Luong : ");
        scanf("%d",&p.luong);
        fprintf(stdprn,"\n Ngay sinh:%d%s%d",p.ngaysinh.ngay,p.ngaysinh.thang,
        p.ngaysinh.nam);
        fprintf(stdprn,"\n Ngay vao co quan:%d%s%d",p.ngayvaocq.ngay,
        p.ngayvaocq.thang,p.ngayvaocq.nam);
        fprintf(stdprn,"\n Luong : %8.2f",p.luong);
    }
```

8.4. Mảng cấu trúc :

Như đã đề cập ở các chương trước, khi sử dụng một kiểu giá trị (ví dụ như kiểu int) ta có thể khai báo các biến và các mảng kiểu đó. Ví dụ như khai báo :

```
int a,b,c[10];
```

cho ta hai biến nguyên là a,b và một mảng nguyên c có 10 phần tử.

Hoàn toàn tương tự như vậy : ta có thể sử dụng một kiểu cấu trúc đã mô tả để khai báo các cấu trúc và mảng cấu trúc.

Cách khai báo mảng cấu trúc :

struct tên_kiểu_cấu_trúc_đã_định_nghĩa tên_mảng_cấu_trúc[số phần tử của mảng];

Ví dụ :

Ví dụ 1 :

Giả sử kiểu cấu trúc **canbo** đã được định nghĩa như mục trên. Khi đó dòng khai báo :

```
struct canbo cb1,cb2,nhom1[10],nhom2[7];
```

sẽ cho :

Hai biến cấu trúc cb1 và cb2.

Hai mảng cấu trúc nhom1 co 10 phần tử và nhom2 có 7 phần tử và mỗi phần tử của hai nhóm này có kiểu **canbo**.

Ví dụ 2 :

Đoạn chương trình sau sẽ tính tổng lương cho các phần tử nhóm 1:

```
double tongluong=0;
for (i=0;i<10;++i)
        tongluong+=nhom1[i].luong;
```

Chú ý :

Không cho phép sử dụng phép toán lấy địa chỉ đối với các thành phần của mảng cấu trúc khác kiểu nguyên. Chẳng hạn không cho phép sử dụng câu lệnh sau :

```
scanf("%f",&nhom1[5].luong);
```

Trong trường hợp này ta dùng biến trung gian.

8.5. Khởi đầu một cấu trúc :

Có thể khởi đầu cho một cấu trúc ngoài, cấu trúc tĩnh, mảng cấu trúc ngoài và mảng cấu trúc tĩnh

8.6. Phép gán cấu trúc :

Có thể thực hiện phép gán trên các biến và phần tử mảng cấu trúc cùng kiểu như sau :

- Gán hai biến cấu trúc cho nhau
- Gán biến cấu trúc cho phần tử mảng cấu trúc
- Gán phần tử mảng cấu trúc cho biến cấu trúc
- Gán hai phần tử mảng cấu trúc cho nhau

Mỗi một phép gán trên tương đương với một dãy phép gán các thành phần tương ứng.

Ví dụ :

Đoạn chương trình sau minh hoạ cách dùng phép gán cấu trúc để để sắp xếp n thí sinh theo thứ tự giảm của tổng điểm :

```
struct thisinh
        {
                char ht[25];
                float td;
        } tg,ts[100];
for (i=1;i<=n-1;++i)
for (j=1;j<=n;++j)
        if (ts[i].td<ts[j].td)
                {
                        tg=ts[i];
                        ts[i]=ts[j];
                        ts[j]=tg;
                }
```

8.7. Con trỏ cấu trúc và địa chỉ cấu trúc :

8.7.1. Con trỏ và địa chỉ :

Ta xét ví dụ sau :

```
struct ngay
        {
                int ngaythu;
                char thang[10];
                int nam;
        };
struct nhancong
```

```
{
        char ten[20];
        char diachi[25];
        double bacluong;
        struct ngay ngaysinh;
};
```

Nếu khai báo :

```
struct nhancong *p,*p1,*p2,nc1,nc2,ds[100];
```

ta có :

- p, p1, p2 là con trỏ cấu trúc
- nc1, nc2 là các biến cấu trúc
- ds là mảng cấu trúc

Con trỏ cấu trúc dùng để lưu trữ địa chỉ của biến cấu trúc và mảng cấu trúc.

Ví dụ :

```
p1=&nc1;        /* Gửi địa chỉ nc1 vào p1 */
p2=&ds[4];      /* Gửi địa chỉ ds[4] vào p2 */
p=ds;           /* Gửi địa chỉ ds[0] vào p */
```

8.7.2. Truy nhập qua con trỏ:

Có thể truy nhập đến các thành phần thông qua con trỏ theo một trong hai cách sau :

Cách một :

```
Tên_con_trỏ->Tên_thành_phần
```

Cách hai :

```
(*Tên_con_trỏ).Tên_thành_phần
```

Ví dụ :

```
nc1.ngaysinh.nam
p1-> ngaysinh.nam
ds[4].ngaysinh.thang
(*p2). ngaysinh.thang
```

8.7.3. Phép gán qua con trỏ:

Giả sử ta gán :

p1=&nc1;

p2=&ds[4];

Khi đó có thể dùng :

*p1 thay cho nc1

*p2 thay cho ds[4]

Tức là viết:

ds[5]=nc1;

ds[4]=nc2;

Tương đương với :

ds[5]=*p1;

*p2=nc2;

8.7.4. Phép cộng địa chỉ :

Sau các phép gán :

p=ds;

p2=&ds[4];

thì p trỏ thới ds[[0]] và p2 trỏ tới ds[4]. Ta có thể dùng các phép cộng, trừ địa chỉ để làm cho p và p2 trỏ tới các thành phần bất kỳ nào khác.

Ví dụ :

Sau các lệnh :

p=p+10;

p2=p2-4;

thì p trỏ tới ds[10] còn p2 trỏ tới ds[0]

8.7.5. Con trỏ và mảng :

Giả sử con trỏ p trỏ tới đầu mảng ds, khi đó :

- Ta có thể truy nhập tới các thành phần cấu trúc bằng các cách sau :

  + ds[i].thành_phần ds[i].ngaysinh.nam

  + p[i].thành_phần p[i].ngaysinh.nam

+ (p+i)->thành_phần (p+i)->ngaysinh.nam

- Khi ta sử dụng cả cấu trúc thì các cách viết sau là tương đương :

ds[i] p[i] *(p+i)

8.8. Cấu trúc tự trỏ và danh sách liên kết :

Khi ta lập một chương trình quản lý mà bản thân số biến (cấu trúc) chưa được biết trước, nếu ta sử dụng mảng (cấp phát bộ nhớ tĩnh) thì ta phải sử dụng số các phần tử là tối đa. Như vậy sẽ có rất nhiều vùng nhớ được cấp phát mà không bao giờ dùng đến. Lúc đó ta có cách để cấp phát bộ nhớ động. Số vùng nhớ cấp ra đủ số biến cần dùng.

Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa gọi là cấu trúc tự trỏ.

Ví dụ :

Các cách để định nghĩa cấu trúc tự trỏ person:

Cách 1 :

```
typedef struct pp
        {
                char ht[20];
                char qq[25];
                int tuoi;
                struct pp *tiep;
        } person;
```

Cách 2 :

```
typedef struct pp person
struct pp
        {
                char ht[20];
                char qq[25];
                int tuoi;
                person *tiep;
```

```
};
```

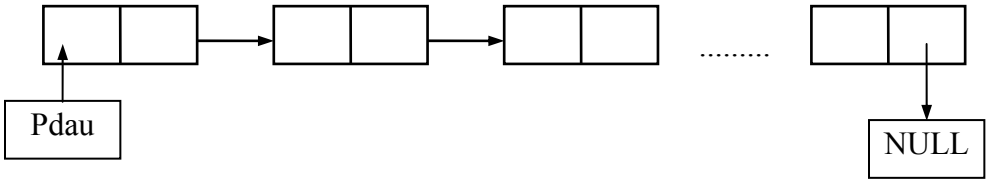
Cách 3 :

```
struct pp
    {
        char ht[20];
        char qq[25];
        int tuoi;
        struct pp *tiep;
    };
typedef pp person;
```

Cấu trúc tự trỏ được dùng để xây dựng danh sách liên kết (móc nối), đó là một nhóm các cấu trúc có tính chất sau : (Móc nối theo chiều thuận).

- Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó.
- Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp sau của danh sách.
- Cấu trúc cuối chứa hằng NULL.

Ví dụ :



Với danh sách này, ta có thể lần lượt từ cấu trúc đầu đến cấu trúc cuối theo chiều từ trên xuống dưới.

Nhóm cấu trúc móc nối theo chiều ngược có tính chất sau :

- Biết địa chỉ cấu trúc cuối.
- Trong mỗi cấu trúc (trừ cấu trúc đầu) đều chứ địa chỉ của cấu trúc trước.
- Cấu trúc đầu chứa hằng NULL.

Với danh sách này, ta có thể lần lượt từ cấu trúc cuối lên cấu trúc đầu theo chiều từ dưới lên trên.
Ngoài ra, ta có thể xây dựng các danh sách mà mỗi phần tử chứa hai địa chỉ của cấu trúc trước và cấu trúc sau. Với loại danh sách này, ta có thể truy nhập theo cả hai chiều trên.
Khi làm việc với danh sách móc nối, ta thường phải tiến hành các công việc sau sau :

(Giả sử ta có con trỏ **p**, trỏ **pdau** chỉ cấu trúc đầu của danh sách, con trỏ **tiep** là thành phần con trỏ của cấu trúc)

Tạo danh sách mới :

- Cấp phát bộ nhớ cho một cấu trúc
- Nhập một biến cấu trúc vào vùng nhớ vừa cấp
- Gán địa chỉ của cấu trúc sau cho thành phần con trỏ của cấu trúc trước

Duyệt qua tất cả các phần tử của danh sách :

- Đưa trỏ p về trỏ cùng cấu trúc với pdau bằng lệnh :

  p=pdau
- Để chuyển tiếp đến người tiếp theo ta dùng lệnh :

  p=p->tiep
- Dấu hiệu để biết đang xét cấu trúc cuối cùng của danh sách là :

  p->tiep==NULL

Loại một cấu trúc ra khỏi danh sách :

- Lưu trữ địa chỉ của cấu trúc cần loại vào một con trỏ (Để giải phóng bộ nhớ của cấu trúc này)
- Sửa để cấu trúc trước đó có địa chỉ của cấu trúc cần loại
- Giải phóng bộ nhớ cấu trúc cần loại

Bổ xung hoặc chèn một cấu trúc vào danh sách:

- Cấp phát bộ nhớ và nhập bổ xung
- Sửa thành phần con trỏ trong các cấu trúc có liên quan để đảm bảo mỗi cấu trúc chứa địa chỉ của cấu trúc tiếp theo

Hàm cấp phát bộ nhớ :

```
void *malloc(kichthuoc_t kichthuoc);
```

Hàm lấy trong thư viện alloc.h hoặc stdlib.h.

kichthuoc tính bằng số by te. Hàm sẽ đưa con trỏ về vị trí ô nhớ vừa được cấp hoặc về NULL nếu không đủ bộ nhớ cần thiết. Nếu **kichthuoc == 0** thì nó trả về NULL.

Ví dụ :

```
#include "stdio.h"
 #include "string.h"
 #include "alloc.h"
 #include "process.h"
 int main()
 {
   char *str;
   /* Cấp phát bộ nhớ cho xâu ký tự */
   if ((str = malloc(10)) == NULL)
   {
     printf("Not enough memory to allocate buffer\n");
     exit(1);  /* Kết thúc chương trình nếu thiếu bộ nhớ */
   }
   /* copy "Hello" vào xâu */
   strcpy(str, "Hello");
   /* Hiển thị xâu */
   printf("String is %s\n", str);
   /* Giải phóng bộ nhớ */
   free(str);
   return 0;
 }
```

Ví dụ :

Tạo một danh sách liên kết. Các biến cấu trúc gồm các trường : Họ tên, Quê quán, tuổi, và một trường con trỏ là Tiếp.

Móc nối theo chiều thuận (Vào trước ra trước FIFO first in first out):

```
#include "stdio.h"
#include "alloc.h"
#include "conio.h"
#include "string.h"
typedef struct pp
        {
                char ht[25];
```

```
                char qq[20];
                int tuoi;
                struct pp *tiep;
        } nhansu;
main()
        {
                char tt;
                nhansu *pdau,*pcuoi,*p;
                char tam[10];
                clrscr();
                pdau=NULL;
                do
                        {
            p=(nhansu*)malloc(sizeof(nhansu));
            printf("\n Ho ten : ");
                        gets(p->ht);
             printf(" Que quan : ");
                        gets(p->qq);
             printf(" Tuoi: ");
                        gets(tam);
                        p->tuoi=atoi(tam);
            if (pdau==NULL)
              {
                pdau=p;
                pcuoi=p;
                            p->tiep=NULL;
                                }
                              else
                                        {
                                         pcuoi->tiep=p;
                pcuoi=p;
                p->tiep=NULL;
                                        }
```

```
                printf("\nBam phim bat ky de tiep tuc, ESC de dung");
                tt=getch();
        } while(tt!=27) ;
            /* Đưa danh sách liên kết ra màn hình, trỏ pdau tro */
            printf("\n Danh sach nhu sau :\n");
            p=pdau;
            while (p!=NULL)
                {
                    printf("\n Ho ten: %25s  Que : %20s Tuoi :
                    %d",(*p).ht,(*p).qq,(*p).tuoi);
                    p=p->tiep;
                }
            getch();
    }
```

Móc nối theo chiều ngược (Vào sau ra trước LIFO last in first out):

```
#include "stdio.h"
#include "alloc.h"
#include "conio.h"
#include "string.h"
typedef struct pp
    {
        char ht[25];
        char qq[20];
        int tuoi;
        struct pp *tiep;
    } nhansu;
main()
    {
        char tt;
        nhansu *pdau,*pcuoi,*p;
         char tam[10];
        clrscr();
        pdau=NULL;
```

```
        do
                {
    p=(nhansu*)malloc(sizeof(nhansu));
    printf("\n Ho ten : ");
                gets(p->ht);
    printf(" Que quan : ");
                gets(p->qq);
     printf(" Tuoi: ");
                gets(tam);
                p->tuoi=atoi(tam);
    if (pdau==NULL)
      {
        pdau=p;
        pcuoi=p;
                    p->tiep=NULL;
                        }
                      else
                              {
                                p->tiep=pcuoi;
        pcuoi=p;
                              }
                    printf("\nBam phim bat ky de tiep tuc, ESC de dung");
                    tt=getch();
} while(tt!=27) ;
        /* Đưa danh sách liên kết ra màn hình, trỏ pdau tro */
        printf("\n Danh sach nhu sau :\n");
        p=pcuoi;
        while (p!=NULL)
                {
                        printf("\n Ho ten: %25s  Que : %20s Tuoi :
                        %d",(*p).ht,(*p).qq,(*p).tuoi);
                        p=p->tiep;
                }
```

```
        getch();
}
```

Chương 9

TẬP TIN - FILE

9.1. Khái niệm về tệp tin :

Tệp tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau và có cùng một kiểu được nhóm lại với nhau thành một dãy. Chúng thường được chứa trong một thiết bị nhớ ngoài của mấy tính (đĩa mềm, đĩa cứng...) dưới một cái tên nào đó.

Tên tiếng Anh của tệp là **file,** nó được dùng để chỉ ra một hộp đựng các phiếu hay thẻ ghi của thư viện. Một hình ảnh rõ nét giúp ta hình dung ra tệp là tủ phiếu của thư viện. Một hộp có nhiều phiếu giống nhau về hình thức và tổ chức, song lại khác nhau về nội dung. ở đây, tủ phiếu là tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng hoặc một đĩa mềm đóng vai trò chiếc tủ (để chứa nhiều tệp).

Tệp được chứa trong bộ nhớ ngoài, điều đó có nghĩa là tệp được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện. Chính vì lý do trên, chỉ những dữ liệu nào cần lưu trữ (như hồ sơ chẳng hạn) thì ta nên dùng đến tệp.

Tệp là một kiểu dữ liệu có cấu trúc. Định nghĩa tệp có phần nào giống mảng ở chỗ chúng đều là tập hợp của các phần tử dữ liệu cùng kiểu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong định nghĩa.

Trong C, các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia làm hai nhóm : nhóm 1 và nhóm 2. Các hàm cấp 1 là các hàm nhập / xuất hệ thống, chúng thực hiện việc đọc ghi như DOS. Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1 nên trong các chương trình viết trong C, các hàm cấp 2 hay được sử dụng hơn.

Một tệp tin dù được xây dựng bằng cách nào đi nữa cũng chỉ đơn giản là một dãy các byte ghi trên đĩa (có giá trị từ 0 đến 255). Số byte của dãy chính là độ dài của tệp.

Có hai kiểu nhập xuất dữ liệu lên tệp : Nhập xuất nhị phân và nhập xuất văn bản.

Nhập xuất nhị phân :

- Dữ liệu ghi lên tệp theo các byte nhị phân như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị biến đổi.
- Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (được định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác 0.

Nhập xuất văn bản:

- Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.
- Mã chuyển dòng :

  Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF

  Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF

Mã kết thúc tệp :

Trong khi đọc, nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (bằng -1) và hàm feof(fp) cho giá trị khác 0 (bằng 1).

9.2. Khai báo sử dụng tệp - một số hàm thường dùng khi thao tác trên tệp :

9.2.1. Khai báo sử dụng tệp :

Để khai báo sử dụng tệp, ta dùng lệnh sau :

FILE biến_con_trỏ_tệp;

Trong đó biến_con_trỏ_tệp có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phảy (dấu **,**).

Ví dụ :

```
FILE *vb, *np;  /* Khai báo hai biến con trỏ tệp */
```

9.2.2. Mở tệp - hàm fopen :

Cấu trúc ngữ pháp của hàm :

```
FILE *fopen(const char *tên_tệp, const char *kiểu);
```

Nguyên hàm trong : stdio.h .

Trong đó :

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Công dụng :

Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp hai sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng sau chỉ ra các giá trị của kiểu :

Tên kiểu	ý nghĩa
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w" "wt"	Mở một tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a" "at"	Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"ab"	Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w+" "w+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

Chú ý :

Trong các kiểu đọc ghi, ta nên lầm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc ngược lại. Ta sẽ đề cập đến các hàm với tính năng xoá sau này.

Ví dụ :

f=fopen("TEPNP","wb");

9.2.3. Đóng tệp - hàm fclose :

Cấu trúc ngữ pháp của hàm :

int fclose(FILE *fp);

Nguyên hàm trong : stdio.h .

Trong đó :

fp là con trỏ ứng với tệp cần đóng.

Công dụng :

Hàm dùng để đóng tệp khi kết thúc các thao tác trên nó. Khi đóng tệp, máy thực hiện các công việc sau :

- Khi đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
- Khi đang đọc dữ liệu thì máy sẽ xoá vùng đệm
- Giải phóng biến trỏ tệp.
- Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

fclose(f);

9.2.4. Đóng tất cả các tệp đang mở- hàm fcloseall :

Cấu trúc ngữ pháp của hàm :

int fcloseall(void);

Nguyên hàm trong : stdio.h .

Công dụng :

Hàm dùng để đóng tất cả các tệp đang mở . Nếu lệnh thành công, hàm sẽ cho giá trị bằng số là số tệp được đóng, trái lại nó cho hàm EOF.

Ví dụ :

fcloseall();

9.2.5. Làm sạch vùng đệm - hàm fflush :

Cấu trúc ngữ pháp của hàm :

```
int fflush(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tệp fp. Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

```
fflush(f);
```

9.2.6. Làm sạch vùng đệm của các tệp đang mở - hàm fflushall :

Cấu trúc ngữ pháp của hàm :

```
int fflushall(void);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tất cả các tệp đang mở. Nếu lệnh thành công, hàm sẽ cho giá trị bằng số các tệp đang mở, trái lại nó cho hàm EOF.

Ví dụ :

```
fflushall();
```

9.2.7. Kiểm tra lỗi file - hàm ferror :

Cấu trúc ngữ pháp của hàm :

```
int ferror(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra lỗi khi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không có lỗi, trái lại hàm cho giá trị khác 0.

9.2.8. Kiểmtra cuối tệp - hàm feof :

Cấu trúc ngữ pháp của hàm :

```
int feof(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

9.2.9. Truy nhập ngẫu nhiên - các hàm di chuyên con trỏ chỉ vị :

9.2.7.1. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind :

Cấu trúc ngữ pháp :

```
void rewind(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu.

Ví dụ :

```
rewind(f);
```

9.2.9.2. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek :

Cấu trúc ngữ pháp :

```
int fseek(FILE *fp, long sb, int xp);
```

Nguyên hàm trong : stdio.h .

Trong đó

fp là con trỏ tệp.

sb là số byte cần di chuyển.

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.

xp có thể nhận các giá trị sau :

xp=SEEK_SET hay 0 : Xuất phát từ đầu tệp.

xp=SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp=SEEK_END hay 2 : Xuất phát từ cuối tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp fp về vị trí xác định bởi xp qua một số byte xác định bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại nó sẽ di chuyển về đầu tệp. Khi thành công, hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

Chú ý :

Không nên dùng fseek trên tệp tin văn bản, do sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

Ví dụ :

```
fseek(stream, SEEK_SET, 0);
```

9.2.9.3. Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell :

Cấu trúc ngữ pháp :

```
int ftell(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó

fp là con trỏ tệp.

Công dụng :

Hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp **fp**) khi thành công. Số thứ tự tính từ 0. Trái lại hàm cho giá trị -1L.

Ví dụ :

Sau lệnh fseek(fp,0,SEEK_END);

ftell(fp) cho giá trị 3.

Sau lệnh fseek(fp,-1,SEEK_END);

ftell(fp) cho giá trị 2.

9.2.10. Ghi các mẫu tin lên tệp - hàm fwrite :

Cấu trúc ngữ pháp của hàm :

```
int fwrite(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm ghi n mẫu tin kích thước **size** byte từ vùng nhớ **ptr** lên tệp **fp**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự ghi được.

Ví dụ :

```
#include "stdio.h"
struct mystruct
{
 int i;
 char ch;
};
main()
{
  FILE *stream;
  struct mystruct s;
  stream = fopen("TEST.TXT", "wb") /* Mở tệp TEST.TXT */
  s.i = 0;
```

```
  s.ch = 'A';
  fwrite(&s, sizeof(s), 1, stream); /* Viết cấu trúc vào tệp */
  fclose(stream); /* Đóng tệp */
  return 0;
}
```

9.2.11. Đọc các mẫu tin từ tệp - hàm fread :

Cấu trúc ngữ pháp của hàm :

```
int fread(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm đọc n mẫu tin kích thước **size** byte từ tệp **fp** lên lên vùng nhớ **ptr**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ :

```
#include "string.h"
#include "stdio.h"
main()
{
  FILE *stream;
  char msg[] = "Kiểm tra";
  char buf[20];
 stream = fopen("DUMMY.FIL", "w+");
  /* Viết vài dữ liệu lên tệp */
  fwrite(msg, strlen(msg)+1, 1, stream);
 /* Tìm điểm đầu của file */
```

```
  fseek(stream, SEEK_SET, 0);
  /* Đọc số liệu và hiển thị */
  fread(buf, strlen(msg)+1, 1, stream);
  printf("%s\n", buf);
  fclose(stream);
  return 0;
}
```

9.2.10. Nhập xuất ký tự :

9.2.10.1. Các hàm putc và fputc :

Cấu trúc ngữ pháp :

```
int putc(int ch, FILE *fp);
int fputc(int ch, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ch là một giá trị nguyên

fp là một con trỏ tệp.

Công dụng :

Hàm ghi lên tệp fp một ký tự có mã bằng

m=ch % 256.

ch được xem là một giá trị nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại cho EOF

Ví dụ :

```
#include "stdio.h"
main()
{
  char msg[] = "Hello world\n";
  int i = 0;
  while (msg[i])
    putc(msg[i++], stdout);  /* stdout thiết bị ra chuẩn - Màn hình*/
```

```
  return 0;
}
```

9.2.12.2. Các hàm getc và fgettc :

Cấu trúc ngữ pháp :

int gretc(FILE *fp);

int fputc(FILE *fp);

Nguyên hàm trong : stdio.h .

Trong đó :

fp là một con trỏ tệp.

Công dụng :

Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ :

```
#include "string.h"
#include "stdio.h"
#include "conio.h"
main()
{
  FILE *stream;
  char string[] = "Kiem tra";
  char ch;
  /* Mở tệp để cập nhật*/
  stream = fopen("DUMMY.FIL", "w+");
  /*Viết một xâu ký tự vào tệp */
  fwrite(string, strlen(string), 1, stream);
  /* Tìm vị trí đầu của tệp */
  fseek(stream, 0, SEEK_SET);
```

```
  do
  {
    /* Đọc một ký tự từ tệp */
    ch = fgetc(stream);
    /* Hiển thị ký tự */
    putch(ch);
  } while (ch != EOF);
  fclose(stream);
  return 0;
}
```

9.2.13. Xoá tệp - hàm unlink:

Cấu trúc ngữ pháp :

```
int unlink(const char *tên_tệp)
```

Nguyên hàm trong : dos.h, io.h, stdio.h .

Trong đó

tên_tệp là tên của tệp cần xoá.

Công dụng :

Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ :

```
#include <stdio.h>
#include <io.h>
int main(void)
{
  FILE *fp = fopen("junk.jnk","w");
  int status;
  fprintf(fp,"junk");
  status = access("junk.jnk",0);
  if (status == 0)
    printf("Tệp tồn tại\n");
  else
```

```
    printf("Tệp không tồn tại\n");
  fclose(fp);
  unlink("junk.jnk");
  status = access("junk.jnk",0);
  if (status == 0)
    printf("Tệp tồn tại\n");
  else
    printf("Tệp không tồn tại\n");
  return 0;
}
```

Chương 10

ĐỒ HOẠ

Chương này sẽ giới thiệu các hàm và thủ tục để khởi động hệ đồ hoạ, vẽ các đường và hình cơ bản như hình tròn, cung elip, hình quạt, đường gãy khúc, đa giác, đường thẳng, hình chữ nhật, hình hộp chữ nhật....

Các hàm và thủ tục đồ hoạ được khai báo trong file graphics.h.

10.1. Khởi động đồ hoạ :

Mục đích của việc khởi động hệ thống đồ hoạ là xác định thiết bị đồ hoạ (màn hình) và mode đồ hoạ sẽ sử dụng trong chương trình. Để làm công việc này, ta có hàm sau :

void initgraph(int *graphdriver,int graphmode,char *driverpath);

Trong đó :

- driverpath là xâu ký tự chỉ đường dẫn đến thư mục chứa các tập tin điều khiển đồ hoạ.
- graphdriver cho biết màn hình đồ hoạ sử dụng trong chương trình.
- graphmode cho biết mode đồ hoạ sử dụng trong chương trình.

Bảng dưới đây cho các giá trị khả dĩ của graphdriver và graphmode :

graphdriver	**graphmode**	**Độ phân giải**
DETECT (0)		
CGA (1)	CGAC0 (0)	320x200
	CGAC1 (1)	320x200
	CGAC2 (2)	320x200
	CGAC3 (3)	320x200
	CGAHi (4)	640x200
MCGA (2)	MCGA0 (0)	320x200
	MCGA1 (1)	320x200
	MCGA2 (2)	320x200
	MCGA3 (3)	320x200
	MCGAMed (4)	640x200
	MCGAHi (5)	640x480
EGA (3)	EGAL0 (0)	640x200
	EGAHi (1)	640x350

EGA64 (4)	EGA64LO (0)	640x200
	EGA64Hi (1)	640x350
EGAMONO (5)	EGAMONOHi (0)	640x350
VGA (9)	VGALO (0)	640x200
	VGAMED (1)	640x350
	VGAHI (2)	640x480
HERCMONO (7)	HERCMONOHI	720x348
ATT400 (8)	ATT400C0 (0)	320x200
	ATT400C1 (1)	320x200
	ATT400C2 (2)	320x200
	ATT400C3 (3)	320x200
	ATT400MED (4)	640x400
	ATT400HI (5)	640x400
PC3270 (10)	PC3270HI (0)	720x350
IBM8514 (6)	PC3270LO (0)	640x480 256 mầu
	PC3270HI (1)	1024x768 256 mầu

Chú ý :

- Bảng trên cho ta các hằng và giá trị của chúng mà các biến graphdtriver và graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0 vv...

Khi lập trình ta có thể thay thế vào vị trí tương ứng của chúng trong hàm tên hằng hoặc giá trị của hằng đó.

Ví dụ :

Giả sử máy tính có màn hình VGA, các tập tin đồ hoạ chứa trong thư mục C:\TC \BGI, khi đó ta khởi động hệ thống đồ hoạ như sau :

```
#include "graphics.h"
main()
        {
                int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
                initgraph(&mh,&mode,"C:\\TC\\BGI");
                /* Vì kí tự \ trong C là kí tự đặc biệt nên ta phải gấp đôi nó */
```

}

- Bảng trên còn cho thấy độ phân giải còn phụ thuộc cả vào màn hình và mode. Ví dụ như trong màn hình EGA nếu dùng EGALo thì độ phân giải là 640x200 (Hàm getmaxx() cho giá trị cực đại của số điểm theo chiều ngang của màn hình. Với màn hình EGA trên : 639, Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình. Với màn hình EGA trên : 199).
- Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver bằng DETECT hay giá trị 0. Khi đó, kết quả của initgraph sẽ là :

  Kiểu màn hình đang sử dụng được phát hiện, giá trị của nó được gán cho biến graphdriver.

  Mode đồ hoạ ở độ phân giải cao nhất ứng với màn hành đang sử dụng cũng được phát hiện và trị số của nó được gán cho biến graphmode.

  Như vậy dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ hoạ với màn hình hiện có theo mode có độ phân giải cao nhất mà còn giúp ta xác định kiểu màn hình đang sử dụng.

Ví dụ :

Chương trình dưới đây xác định kiểu màn hình đang sử dụng :

```
#include "graphics.h"
#include "stdio.h"
main()
        {
                int mh=0, mode;
                initgraph(&mh,&mode,"C:\\TC\\BGI");
                printf("\n Gia tri so cua man hinh la : %d",mh);
                printf("\n Gia tri so mode do hoa la : %d",mode);
                closegraph();
        }
```

- Nếu chuỗi dùng để xác định driverpath là chuỗi rỗng thì chương trình dịch sẽ tìm kiếm các file điều khiển đồ hoạ trên thư mục chủ (Thư mục hiện thời).

10.2. Các hàm đồ hoạ :

10.2.1. Mẫu và màu :

- **Đặt màu nền :**

  Để đặt màu cho nền ta dùng thủ tục sau :

  void setbkcolor(int màu);

- **Đặt màu đường vẽ :**

  Để đặt màu vẽ đường ta dùng thủ tục sau :

  void setcolor(int màu);

- **Đặt mẫu (kiểu) tô và màu tô :**

  Để đặt mẫu (kiểu) tô và màu tô ta dùng thủ tục sau :

  void setfillstyle(int mẫu, int màu);

Trong cả ba trường hợp **màu** xác định mã của màu.

Các giá trị khả dĩ của **màu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của màu

Tên hằng	Giá trị số	Màu hiển thị
BLACK	0	Đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIGHTGRAY	7	Xám nhạt
DARKGRAY	8	Xám đậm
LIGHTBLUE	9	Xanh xa trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	16	Trắng

Các giá trị khả dĩ của **mẫu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của mẫu

Tên hằng	Giá trị số	Kiểu mẫu tô

EMPTY_FILL	0	Tô bằng mầu nền
SOLID_FILL	1	Tô bằng đường liền nét
LINE_FILL	2	Tô bằng đường --------
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

Chọn giải màu :

Để thay đổi giải màu đã được định nghĩa trong bảng trên, ta sử dụng hàm :

void setpalete(int số_thứ_tự_màu, int màu);

Ví dụ :

Câu lệnh :

setpalete(0,lightcyan);

biến màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng.

- **Lấy giải màu hiện thời :**

  + Hàm getcolor trả về mầu đã xác định bằng thủ tục setcolor ngay trước nó.

  + Hàm getbkcolor trả về mầu đã xác định bằng hàm setbkcolor ngay trước nó.

10.2.2. Vẽ và tô màu :

Có thể chia các đường và hình thành bốn nhóm chính :

- Cung tròn và hình tròn.
- Đường gấp khúc và đa giác.
- Đường thẳng.
- Hình chữ nhật.

10.2.2.1. Cung tròn và đường tròn :

Nhóm này bao gồm : Cung tròn, đường tròn, cung elip và hình quạt.

- **Cung tròn :**

Để vẽ một cung tròn ta dùng hàm :

void arc(int x, int y, int gd, int gc, int r);

Trong đó :

(x,y) là toạ độ tâm cung tròn.

gd là góc đầu cung tròn(0 đến 360 độ).

gc là góc cuối cung tròn (gd đến 360 độ).

r là bán kính cung tròn .

Ví dụ :

Vẽ một cung tròn có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán kính 30.

arc(100,50,0,180,30);

- **Đường tròn :**

Để vẽ đường tròn ta dùng hàm :

void circle(int x, int y, int r);

Trong đó :

(x,y) là toạ độ tâm cung tròn.

r là bán kính đường tròn.

Ví dụ :

Vẽ một đường tròn có tâm tại (100,50) và bán kính 30.

circle(100,50,30);

- **Cung elip**

Để vẽ một cung elip ta dùng hàm :

void ellipse(int x, int y, int gd, int gc, int xr, int yr);

Trong đó :

(x,y) là toạ độ tâm cung elip.

gd là góc đầu cung tròn(0 đến 360 độ).

gc là góc cuối cung tròn (gd đến 360 độ).

xr là bán trục nằm ngang.

yr là bán trục thẳng đứng.

Ví dụ :

Vẽ một cung elip có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán trục ngang 30, bán trục đứng là 20.

```
ellipse(100,50,0,180,30,20);
```

- **Hình quạt :**

Để vẽ và tô màu một hình quạt ta dùng hàm :

```
void pieslice(int x, int y, int gd, int gc, int r);
```

Trong đó :

(x,y) là toạ độ tâm hình quạt.

gd là góc đầu hình quạt (0 đến 360 độ).

gc là góc cuối hình quạt (gd đến 360 độ).

r là bán kính hình quạt .

Ví dụ :

Chương trình dưới đây sẽ vẽ một cung tròn ở góc phần tư thứ nhất, một cung elip ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
# include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
    {
        int md=0,mode;
        initgraph(&md,&mode,"C:\\TC\\BGI");
        setbkcolor(BLUE);
        setcolor(YELLOW);
        setfillstyle(SOLID_FILL,RED);;
        arc(160,50,0,90,45);
        circle(160,150,45);
```

```
        pieslice(480,150,90,360,45);
        getch();
        closegraph();
}
```

10.2.3. Vẽ đường gấp khúc và đa giác :

- **Vẽ đường gấp khúc :**

Muốn vẽ đường gấp khúc đi qua n điểm : (x1,y1), (x2,y2), ..., (xn,yn) thì trước hết ta phải gán các toạ độ (xi,yi) cho một mảng a kiểu int nào đó theo nguyên tắc sau :

Toạ độ x1 gán cho a[0]
Toạ độ y1 gán cho a[1]
Toạ độ x2 gán cho a[2]
Toạ độ y2 gán cho a[3]
....
Toạ độ xn gán cho a[2n-2]
Toạ độ yn gán cho a[2n-1]

Sau đó gọi hàm :

```
drawpoly(n,a);
```

Nếu điểm cuối cùng (xn,yn) trùng với điểm đầu (x1,y1) thì ta nhận được một đường gấp khúc khép kín.

- **Tô màu đa giác :**

Giả sử ta có a là mảng đã đề cập đến trong mục trên, khi đó ta gọi hàm :

```
fillpoly(n,a);
```

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm (x1,y1), (x2,y2), ..., (xn,yn)

Ví dụ :

Vẽ một đường gấp khúc và hai đường tam giác.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
```

```
int poly3[]={405,200,590,5,500,300,405,200};
main()
        {
                int md=0,mode;
                initgraph(&md,&mode,"C:\\TC\\BGI");
                setbkcolor(CYAN);
                setcolor(YELLOW);
                setfillstyle(SOLID_FILL,MAGENTA);
                drawpoly(3,poly1);
                fillpoly(3,poly2);
                fillpoly(4,poly3);
               getch();
                closegraph();
        }
```

- **Vẽ đường thẳng :**

Để vẽ đường thẳng nối hai điểm bất kỳ có toạ độ (x1,y1) và (x2,y2) ta sử dụng hàm sau :

```
void line(int x1, int y1, int x2, int y2);
```

Con chạy đồ hoạ giữ nguyên vị trí.

Để vẽ đường thẳng nối từ điểm con chạy đồ hoạ đến một điểm bất có toạ độ (x,y) ta sử dụng hàm sau :

```
void lineto(int x, int y);
```

Con chạy sẽ chuyển đến vị trí (x,y).

Để vẽ một đường thẳng từ ví trí con chạy hiện tại (giả sử là điểm x,y) đến điểm có toạ độ (x+dx,y+dy) ta sử dụng hàm sau :

```
void linerel(int dx, int dy);
```

Con chạy sẽ chuyển đến vị trí (x+dx,y+dy).

- **Di chuyển con chạy đồ hoạ :**

Để di chuyển con chạy đến vị trí (x,y), ta sử dụng hàm sau :

```
void moveto(int x, int y);
```

- **Chọn kiểu đường :**

Hàm void setlinestyle(int kiểu_đường, int mẫu, int độ_dày);

tác động đến nét vẽ của các thủ tục vẽ đường line, lineto,linerel , circle, rectangle (hàm vẽ hình chữ nhật, ta sẽ học trong phần vẽ miền ở dưới).

Hàm này sẽ cho phép ta xác định ba yếu tố khi vẽ đường thẳng, đó là : Kiểu đường, bề dày và mẫu tự tạo.

Dạng đường do tham số **kiểu_đường** xác định. Bảng dưới đây cho các giá trị khả dĩ của **kiểu_đường :**

Tên hằng	Giá trị số	Kiểu đường
SOLID_LINE	0	Nét liền
DOTTED_LINE	1	Nét chấm
CENTER_LINE	2	Nét chấm gạch
DASHED_LINE	3	Nét gạch
USERBIT_LINE	4	Mẫu tự tạo

Bề dày của đường vẽ do tham số **độ_dày** xác định,. bảng dưới đây cho các giá trị khả dĩ của **độ_dày** :

Tên hằng	Giá trị số	Bề dày
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Mẫu tự tạo : Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số **mẫu.** Ví dụ ta xét đoạn chương trình :

```
int pattern = 0x1010;
setlinestile(USERBIT_LINE,pattern,NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010, trong hệ 2 là :

0001 0000 0001 0000

Bit 1 sẽ cho điểm sáng, bit 0 sẽ làm tắt điểm ảnh.

Ví dụ :

Chương trình vẽ một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau :

(20,20),(620,20),(620,180),(20,180) và (320,100)

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
        {
                int mh=0, mode;
                initgraph(&mh,&mode,"C:\\TC\\BGI");
                setbkcolor(BLUE);
                setcolor(YELLOW);
                setlinestyle(SOLID-LINE,0,THICK_WIDTH);
                moveto(320,100);  /* con chạy ở vị trí ( 320,100 ) */
                line(20,20,620,20); /* con chạy vẫn ở vị trí ( 320,100 ) */
                linerel(-300,80);
                lineto(620,180);
                lineto(620,20);
                getch();
                closegraph();
        }
```

10.2.4. Vẽ điểm, miền :

- **Vẽ điểm :**

Hàm :

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo mầu xác định bởi **color.**

Hàm :

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y).

Chú ý :

Nếu điểm này chưa được tô màu bởi các hàm vẽ hoặc hàm putpixel (mà chỉ mới được tạo màu nền bởi setbkcolor) thì hàm cho giá trị 0.

Tô miền :

Để tô màu cho một miền nào đó trên màn hình, ta dùng hàm sau :

void floodfill(int x, int y, int border);

ở đây :

(x,y) là toạ độ của một điểm nào đó gọi là điểm gieo.

Tham số border chứa mã của màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x,y,border và trạng thái màn hình.

+ Khi trên màn hình có một đường cong khép kín hoặc đường gấp khúc khép kín mà mã màu của nó bằng giá trị của border thì :

- Nếu điểm gieo (x,y) nằm trong miền này thì miền giới hạn phía trong đường sẽ được tô màu.

- Nếu điểm gieo (x,y) nằm ngoài miền này thì miền phía ngoài đường sẽ được tô màu.

+ Trong trường hợp khi trên màn hình không có đường cong nào như trên thì cả màn hình sẽ được tô màu.

Ví dụ :

Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Toạ độ (x,y) của điểm gieo được nạp từ bàn phím. Tuỳ thuộc giá trị cụ thể của x,y chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include "graphics.h"
#include "stdio.h"
main()
        {
                int mh=mode=0, x, y;
                printf("\nVao toa do x,y:");
                scanf("%d%d",&x,&y);
                initgraph(&mh,&mode,"");
                if (graphresult != grOk) exit(1);
                setbkcolor(BLUE);
                setcolor(RED);
                setfillstyle(11,YELLOW);
                circle(320,100,50);
                moveto(1,150);
```

```
        floodfill(x,y,RED);
        closegraph();
}
```

10.2.5. Hình chữ nhật :

- Hàm :

```
void rectangle(int x1, int y1, int x2, int y2);
```

sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm :

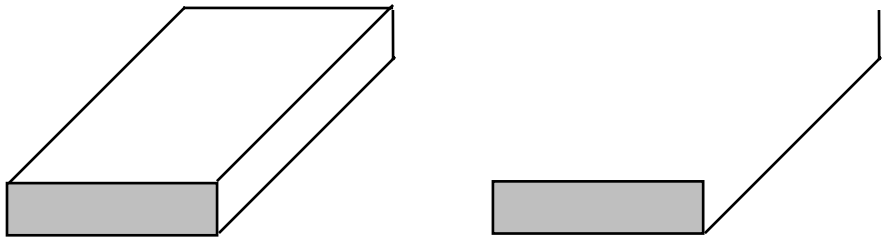
```
void bar(int x1, int y1, int x2, int y2);
```

sẽ vẽ và tô màu một hình chữ nhật. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm :

```
void bar3d(int x1, int y1, int x2, int y2, int depth, int top);
```

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các toạ độ (x1,y1), (x2,y2). Hình chữ nhật này được tô màu thông qua hàm setfillstyle . Tham số **depth** xác định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số **top** có thể nhận các giá trị 1 hay 0 và khối 3 chiều tương ứng sẽ có nắp hoặc không.



top=1 top=0

Ví dụ :

Chương trình dưới đây tạo nên một hình chữ nhật, một khối hình chữ nhật và một hình hộp có nắp :

```
#include "graphics.h"
main()
```

```
{
        int mh=mode=0;
        initgraph(&mh,&mode,"");
        if (graphresult != grOk) exit(1);
        setbkcolor(GREEN);
        setcolor(RED);
        setfillstyle(CLOSE_DOT_FILL,YELLOW);
        rectangle(5,5,300,160);
        bar(3,175,300,340);
        bar3d(320,100,500,340,100,1);
        closegraph();
}
```

10.2.6. Cửa sổ (Viewport) :

- **Thiết lập viewport :**

Viewport là một vùng chữ nhật trên màn hình đồ hoạ. Để thiết lập viewport ta dùng hàm :

```
void setviewport(int x1, int y1, int x2, int y2, int clip);
```

trong đó (x1,y1) là toạ độ góc trên bên trái, (x2,y2) là toạ độ góc dưới bên phải. Bốn giá trị này vì thế phải thoả mãn :

$$0 \le x1 \le x2$$

$$0 \le y1 \le y2$$

Tham số clip có thể nhận một trong hai giá trị :

clip=1 không cho phép vẽ ra ngoài viewport.

clip=0 cho phép vẽ ra ngoài viewport.

Ví dụ :

```
setviewport(100,50,200,150,1);
```

Lập nên một vùng viewport hình chữ nhật có toạ độ góc trái cao là (100,50) và toạ độ góc phải thấp là (200,150) (là toạ độ trước khi đặt viewport).

Chú ý :

Sau khi lập viewport, ta có hệ toạ độ mới mà góc trên bên trái sẽ có toạ độ (0,0).

- **Nhận diện viewport hiện hành :**

Để nhận viewport hiện thời ta dùng hàm :

```
void getviewsetting(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau :

```
struct viewporttype
    {
        int left,top,right,bottom;
        int clip;
    };
```

- **Xóa viewport :**

Sử dụng hàm :

```
void clearviewport(void);
```

- **Xoá màn hình, đưa con chạy về tạo độ (0,0) của màn hình :**

Sử dụng hàm :

```
void cleardevice(void);
```

- **Toạ độ âm dương :**

Nhờ sử dụng viewport có thể viết các chương trình đồ hoạ theo toạ độ âm dương. Muốn vậy ta thiết lập viewport và cho clip bằng 0 để có thể vẽ ra ngoài giới hạn của viewport.

Sau đây là đoạn chương trình thực hiện công việc trên :

```
int xc,yc;
xc=getmaxx()/2;
yc=getmaxy()/2;
setviewport(xc,yc,getmaxx(),getmaxy(),0);
```

Như thế, màn hình sẽ được chia làm bốn phần với toạ độ âm dương như sau :

Phần tư trái trên : x âm, y âm.

x : từ -getmaxx()/2 đến 0.

y : từ -getmaxy()/2 đến 0.

Phần tư trái dưới : x âm, y dương.

x : từ -getmaxx()/2 đến 0.

y : từ 0 đến getmaxy()/2.

Phần tư phải trên : x dương, y âm.

x : từ 0 đến getmaxx()/2.

y : từ -getmaxy()/2 đến 0.

Phần tư phải dưới : x dương, y dương.

x : từ 0 đến getmaxx()/2.

y : từ 0 đến getmaxy()/2.

Ví dụ :

Chương trình vẽ đồ thị hàm sin x trong hệ trục toạ độ âm dương. Hoành độ x lấy các giá trị từ -4π đến 4π. Trong chương trình có sử dụng hai hàm mới là settextjustify và outtextxy ta sẽ đề cập ngay trong phần sau.

```
#include "graphics.h"
#include "conio.h"
#include "math.h"
#define TYLEX 20
#define TYLEY 60
main()
        {
                int mh=mode=DETECT;
                int x,y,i;
                initgraph(mh,mode,"");
                if (graphresult!=grOK ) exit(1);
                setviewport(getmaxx()/2,getmaxy()/2,getmaxx(),getmaxy(),0);
                setbkcolor(BLUE);
                setcolor(YELLOW);
                line(-getmaxx()/2,0,getmaxx()/2,0);
                line(0,-getmaxy()/2,0,getmaxy()/2,0);
                settextjustify(1,1);
                setcolor(WHITE);
                outtextxy(0,0,"(0,0)");
                for (i=-400;i<=400;++i)
                        {
                                x=floor(2*M_PI*i*TYLEX/200);
```

```
                    y=floor(sin(2*M_PI*i/200)*TYLEY);
                    putpixel(x,y,WHITE);
                }
        getch();
        closegraph();
}
```

10.3. Xử lý văn bản trên màn hình đồ hoạ :

- **Hiển thị văn bản trên màn hình đồ hoạ :**

Hàm :

```
void outtext(char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí con trỏ đồ hoạ hiện thời.

Hàm :

```
void outtextxy(int x, int y,char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí (x,y).

Ví dụ :

Hai cách viết dưới đây :

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
outtext(" Say HELLO");
```

cho cùng kết quả.

- **Sử dụng các Fonts chữ :**

Các Fonts chữ nằm trong các tập tin *.CHR trên đĩa. Các Fonts này cho các kích thước và kiểu chữ khác nhau, chúng sẽ được hiển thị lên màn hình bằng các hàm outtext và outtextxy. Để chọn và nạp Fonts ta dùng hàm :

```
void settextstyle(int font, int direction, int charsize);
```

Tham số font để chọn kiểu chữ và nhận một trong các hằng sau :

DEFAULT_FONT=0

TRIPLEX_FONT=1

SMALL_FONT=2

SANS_SERIF_FONT=3

GOTHIC_FONT=4

Tham số derection để chọn hướng chữ và nhận một trong các hằng sau :

HORIZ_DIR=0 văn bản hiển thị theo hướng nằm ngang từ trái qua phải.

VERT_DIR=1 văn bản hiển thị theo hướng thẳng đứng từ dưới lên trên.

Tham số charsize là hệ số phóng to của ký tự và có giá trị trong khoảng từ 1 đến 10.

Khi charsize=1, font hiển thị trong hình chữ nhật 8*8 pixel.

Khi charsize=2 font hiển thị trong hình chữ nhật 16*16 pixel.

.....

Khi charsize=10, font hiển thị trong hình chữ nhật 80*80 pixel.

Các giá trị do settextstyle lập ra sẽ giữ nguyên tới khi gọi một settextstyle mới.

Ví dụ :

Các dòng lệnh :

```
settextstyle(3,VERT_DIR,2);
outtextxy(30,30,"GODS TRUST YOU");
```

sẽ hiển thị tại vị trí (30,30) dòng chữ GODS TRUST YOU theo chiều từ dưới lên trên, font chữ chọn là SANS_SERIF_FONT và cỡ chữ là 2.

- **Đạt vị trí hiển thị của các xâu ký tự cho bởi outtext và outtextxy :**

Hàm settextjustify cho phép chỉ định ra nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy và của outtextxy theo quan hệ với toạ độ (x,y);

Hàm này có dạng sau :

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau :

LEFT_TEXT=0 (Văn bản xuất hiện bên phải con chạy).

CENTER_TEXT (Chỉnh tâm văn bản theo vị trí con chạy).

RIGHT_TEXT (Văn bản xuất hiện bên trái con chạy).

Tham số vert có thể là một trong các hằng số sau :

BOTTOM_TEXT=0 (Văn bản xuất hiện phía trên con chạy).

CENTER_TEXT=1 (Chỉnh tâm văn bản theo vị trí con chạy).

TOP_TEXT=2 (Văn bản xuất hiện phía dưới con chạy).

Ví dụ :

```
settextjustify(1,1);
outtextxy(100,100,"ABC");
```

sẽ cho dòng chữ ABC trong đó điểm (100,100) sẽ nằm dưới chữ B.

- **Bề rộng và chiều cao văn bản :**

Chiều cao :

Hàm :

```
textheight(char *s);
```

cho chiều cao (tính bằng pixel) của chuỗi do con trỏ s trỏ tới.

Ví dụ 1 :

Với font bit map và hệ số phóng đại là 1 thì textheight("A") ch giá trị là 8.

Ví dụ 2 :

```
#include "stdio.h"
#include "graphics.h"
main()
    {
        int mh=mode=DETECT, y,size;
        initgraph(mh,mode,"C:\\TC\\BGI");
        y=10;
        settextjustify(0,0);
        for (size=1;size<5;++size)
            {
                settextstyle(0,0,size);
                outtextxy(0,y,"SACRIFICE");
                y+=textheight("SACRIFICE")+10;
            }
```

```
        getch();
        closegraph();
    }
```

Bề rộng :

Hàm :

```
        textwidth(char *s);
```

cho bề rộng chuỗi (tính theo pixel) mà con trỏ s trỏ tới dựa trên chiều dài chuỗi, kích thước font chữ, hệ số phóng đại.

MỤC LỤC

2.2.4. Hàm putch()

2.3. Đưa kết quả lên màn hình - hàm printf

2.4. Vào số liệu từ bàn phím - hàm scanf

2.5. Đưa kết quả ra máy in

Chương 3

BIỂU THỨC

3.1. Biểu thức

3.2. Lệnh gán và biểu thức

3.3. Các phép toán số học

3.4. Các phép toán quan hệ và logic

3.5. Phép toán tăng giảm

3.6. Thứ tự ưu tiên các phép toán

3.7. Chuyển đổi kiểu giá trị

Chương 4

CẤU TRÚC CƠ BẢN CỦA CHƯƠNG TRÌNH

4.1. Lời chú thích

4.2. Lệnh và khối lệnh

4.2.1. Lệnh

4.2.2. Khối lệnh

4.3. Cấu trúc cơ bản của chương trình

4.4. Một số qui tắc cần nhớ khi viết chương trình

Chương 5

CẤU TRÚC ĐIỀU KHIỂN

5.1. Cấu trúc có điều kiện

5.1.1. Lệnh if-else

5.1.2. Lệnh else-if

5.2. Lệnh nhảy không điều kiện - toán tử goto

5.3. Cấu trúc rẽ nhánh - toán tử switch

5.4. Cấu trúc lặp

5.4.1. Cấu trúc lặp với toán tử while và for

5.4.1.1. Cấu trúc lặp với toán tử while

5.4.1.2. Cấu trúc lặp với toán tử for :

5.4.2. Chu trình do-while

5.5. Câu lệnh break

5.6. Câu lệnh continue

Chương 6

HÀM

6.1. Cơ sở

6.2. Hàm không cho các giá trị

6.3. Hàm đệ qui

6.3.3. Mở đầu

6.3.2. Các bài toán có thể dùng đệ qui

6.3.3. Cách xây dựng hàm đệ qui

6.3.4. Các ví dụ về dùng hàm đệ qui

6.4. Bộ tiền sử lý C

Chương 7

CON TRỎ

7.1. Con trỏ và địa chỉ

7.2. Con trỏ và mảng một chiều

7.2.1.Phép toán lấy địa chỉ

7.2.2. Tên mảng là một hằng địa chỉ

7.2.3. Con trỏ trỏ tới các phần tử của mảng một chiều

7.2.4. Mảng, con trỏ và xâu ký tự

7.3. Con trỏ và mảng nhiều chiều

7.3.1.Phép lấy địa chỉ

7.3.2. Phép cộng địa chỉ trong mảng hai chiều

7.3.3. Con trỏ và mảng hai chiều

7.4. Kiểu con trỏ kiểu địa chỉ, các phép toán trên con trỏ

7.4.1. Kiểu con trỏ và kiểu địa chỉ

7.4.2. Các phép toán trên con trỏ

7.4.3. Con trỏ kiểu void

7.5. Mảng con trỏ

7.6. Con trỏ tới hàm

7.6.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm

7.6.2. Tác dụng của con trỏ hàm

7.6.3. Đối của con trỏ hàm

Chương 8

CẤU TRÚC

8.1. Kiểu cấu trúc

8.2. Khai báo theo một kiểu cấu trúc đã định nghĩa

8.3. Truy nhập đến các thành phần cấu trúc

8.4. Mảng cấu trúc

8.5. Khởi đầu một cấu trúc

8.6. Phép gán cấu trúc

8.7. Con trỏ cấu trúc và địa chỉ cấu trúc

8.7.1. Con trỏ và địa chỉ

8.7.2. Truy nhập qua con trỏ

8.7.3. Phép gán qua con trỏ

8.7.4. Phép cộng địa chỉ

8.7.5. Con trỏ và mảng

8.8. Cấu trúc tự trỏ và danh sách liên kết

Chương 9

TẬP TIN - FILE

9.1. Khái niệm về tệp tin

9.2. Khai báo sử dụng tệp - một số hàm thường dùng khi thao tác trên tệp

9.2.1. Khai báo sử dụng tệp

9.2.2. Mở tệp - hàm fopen

9.2.3. Đóng tệp - hàm fclose

9.2.4. Đóng tất cả các tệp đang mở- hàm fcloseall

9.2.5. Làm sạch vùng đệm - hàm fflush

9.2.6. Làm sạch vùng đệm của các tệp đang mở - hàm fflushall

9.2.7. Kiểm tra lỗi file - hàm ferror

9.2.8. Kiểmtra cuối tệp - hàm feof

9.2.9. Truy nhập ngẫu nhiên - các hàm di chuyên con trỏ chỉ vị

9.2.9.1. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind

9.2.9.2. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek

9.2.9.3. Vị trí hiện tại cuả con trỏ chỉ vị - Hàm ftell

9.2.10. Ghi các mẫu tin lên tệp - hàm fwrite

9.2.11. Đọc các mẫu tin từ tệp - hàm fread

9.2.12. Nhập xuất ký tự

9.2.12.1. Các hàm putc và fputc

9.2.12.2. Các hàm getc và fgettc

9.2.13. Xoá tệp - hàm unlink

Chương 10

ĐỒ HOẠ

10.1. Khởi động đồ hoạ

10.2. Các hàm đồ hoạ

10.2.1. Mẫu và màu

10.2.2. Vẽ và tô màu

10.2.3. Vẽ đường gấp khúc và đa giác

10.2.4. Vẽ điểm, miền

10.2.5. Hình chữ nhật

10.2.6. Cửa sổ (Viewport)

10.3. Sử lý văn bản trên màn hình đồ hoạ

BÀI TẬP.

Phần thứ nhất : *Nhóm các bài tập về tính toán,hàm và chu trình .*

Bài tập 1 :

Viết chương trình hiển thị tháp Pascal :

TÀI LIỆU THAM KHẢO

1. Các tài liệu tiếng Việt :

1.1. Ngô Trung Việt - Ngôn ngữ lập trình C và C++ - Bài giảng- Bài tập - Lời giải mẫu
NXB giao thông vận tải 1995

1.2. Viện tin học - Ngôn ngữ lập trình C
Hà nội 1990

1.3. Lê Văn Doanh - 101 thuật toán và chương trình bằng ngôn ngữ C

2. Các tài liệu tiếng Anh :

2.1. B. Kernighan and D. Ritchie - The C programming language
Prentice Hall 1989

2.2. Programmer's guide Borland C++ Version 4.0
Borland International, Inc 1993

2.3. Bile - Nabaiyoti - TURBO C++
The Waite Group's UNIX 1991

Bài tập
Ngôn ngữ lập trình C

Phần 1 : Nhóm các bài tập về tính toán, hàm và chu trình .

BàI tập 1 :
Viết chương trình hiển thị tháp PASCAL :

1
121
12321
1234321
123454321
12345654321
1234567654321
123456787654321
12345678987654321

Viết chương trình hiển thị tháp đảo ngược.

BàI tập 2 :

Viết chương trình nhập ba số thực. Kiểm tra xem ba số đó có thể là chiều dài của ba cạnh của một tam giác được không? Nếu được thì tính chu vi và diện tích tam giác đó.
BàI tập 3 :
Viết chương trình tính hàm số :

$$f(x) = \cfrac{K_0}{K_1 + \cfrac{x}{K_2 + \cfrac{x}{K_3 + \cfrac{x}{K_4 + \cfrac{x}{\cdots\cdots\cdots\cdots + \cfrac{x}{K_{n-1} + \cfrac{x}{K_n}}}}}}}$$

Bài tập 4 :
Viết chương trình tính tích hai ma trận $C_{mxn} = A_{mxn} * B_{nxk}$.

Bài tập 5 :
Viết chương trình nhập vào một dãy số sau đó tách dãy này thành hai dãy chỉ chứa các số dương và chỉ chứa các số âm. Tính tổng số phần tử của mỗi dãy sau đó sắp xếp để hai dãy có giá trị giảm dần.

Bài tập 6 :
Viết chương trình nhập vào một ma trận A nxm. Tìm giá trị cực đại và cực tiểu của các phần tử của mảng .

Bài tập 7 :

Trăm trâu,trăm cỏ
Trâu đứng ăn năm
Trâu nằm ăn ba
Lụ khụ trâu già
Ba con một bó.

Tính số trâu mỗi loại .

Bài tập 8 :

Vừa gà vừa chó
Bó lại cho tròn

Đúng ba sáu con
Một trăm chân chẵn .
Tính số gà, số chó .

Bài tập 9 :

Th.s. NGUYỄN VĂN LINH

NGÔN NGỮ LẬP TRÌNH

Được biên soạn trong khuôn khổ dự án ASVIET002CNTT
"Tăng cường hiệu quả đào tạo và năng lực tự đào tạo của sinh viên khoa Công nghệ Thông tin - Đại học Cần thơ"

ĐẠI HỌC CẦN THƠ - 12/2003

```
found := 0;
i := 1;
While (i<=n)and (found=0) do
  IF a[i]=x THEN found := 1
  ELSE i := i+1;
```

```
found := FALSE;
i := 1;
While(i<=n)and(NOT found) do
  IF a[i]=x THEN found:= TRUE
  ELSE i:=i+1;
```

4.- **Cú pháp**. Cú pháp của ngôn ngữ có ảnh hưởng lớn đến sự dễ đọc hiểu của chương trình. Chúng ta xét một số thí dụ sau để thấy rõ vấn đề này.

- Một số ngôn ngữ quy định độ dài tối đa của danh biểu quá ngắn, chẳng hạn trong FORTRAN 77 độ dài tối đa của danh biểu là 6, do đó tên biến nhiều khi phải viết tắt nên khó đọc hiểu.
- Việc sử dụng từ khóa cũng góp phần làm cho ngôn ngữ trở nên dễ đọc. Chẳng hạn trong ngôn ngữ Pascal chỉ sử dụng một từ khóa end để kết thúc một khối, kết thúc một lệnh case hay kết thúc một lệnh hợp thành do đó chương trình trở nên khó đọc, trong khi Ada dùng các từ khóa end if để kết thúc lệnh if, end loop để kết thúc lệnh vòng lặp thì chương trình dễ đọc hơn.

1.5.2 Tính dễ viết

Tính dễ viết của một ngôn ngữ là khả năng sử dụng ngôn ngữ đó để viết một chương trình cho một vấn đề nào đó một cách dễ dàng hay không. Thông thường các ngôn ngữ dễ đọc thì đều dễ viết. Tính dễ viết phải được xem xét trong ngữ cảnh của vấn đề mà ngôn ngữ được sử dụng để giải quyết. Theo đó không thể so sánh tính dễ viết của hai ngôn ngữ cho cùng một bài toán mà một trong hai được thiết kế để dành riêng giải quyết bài toán đó. Ví dụ để giải quyết bài toán quản trị dữ liệu, chúng ta không thể so sánh Pascal với một hệ quản trị cơ sở dữ liệu như Foxpro, Access hay Oracle.

Sau đây là một số yếu tố quan trọng nhất ảnh hưởng tới tính dễ viết của ngôn ngữ.

1.- **Sự giản dị**. Nếu một ngôn ngữ có quá nhiều cấu trúc thì một số người lập trình sẽ không quen sử dụng hết tất cả chúng. Tốt nhất là có một số nhỏ các cấu trúc ban đầu và một quy tắc để kết hợp chúng thành các cấu trúc phức tạp hơn.

2.- **Hỗ trợ cho trừu tượng**. Một cách ngắn gọn, trừu tượng (abstraction) là khả năng để định nghĩa và sử dụng các cấu trúc hoặc các phép toán phức tạp theo cách thức mà nó cho phép bỏ qua các chi tiết. Một ví dụ về trừu tượng là chương trình con, từ chương trình gọi, chúng ta gọi chương trình con để thực hiện một tác vụ nào đó mà không cần biết các cài đặt chi tiết bên trong chương trình con đó. Thực chất trừu tượng hóa chính là làm cho chương trình sáng sủa hơn.

3.- **Khả năng diễn đạt**. Là những công cụ của ngôn ngữ mà người lập trình có thể sử dụng để diễn đạt giải thuật một cách dễ dàng. Nói cách khác, một ngôn ngữ có khả năng diễn đạt là ngôn ngữ cung cấp cho người lập trình những công cụ sao cho người lập trình có thể nghĩ sao thì viết chương trình như vậy. Chẳng hạn lệnh lặp FOR trong Pascal dễ sử dụng cho cấu trúc lặp với số lần lặp xác định hơn là lệnh WHILE.

1.5.3 Độ tin cậy

Độ tin cậy của một ngôn ngữ lập trình là khả năng của ngôn ngữ hỗ trợ người lập trình tạo ra các chương trình đúng đắn. Độ tin cậy được thể hiện bởi các đặc trưng sau:

1.- **Kiểm tra kiểu**. Là kiểm tra lỗi về kiểu của chương trình trong giai đoạn dịch hoặc trong khi thực hiện. Kiểm tra kiểu là một yếu tố quan trọng đảm bảo độ tin cậy của ngôn ngữ. Kiểm tra kiểu sẽ báo cho người lập trình biết các lỗi về kiểu và yêu cầu họ có các sửa chữa cần thiết để có một chương trình đúng.

2.- **Xử lý ngoại lệ** (Exception Handing). Là một công cụ cho phép chương trình phát hiện các lỗi trong thời gian thực hiện, tạo khả năng để sửa chữa chúng và sau đó tiếp tục thực hiện mà không phải dừng chương trình.

3.- **Sự lắm tên** (Aliasing): Khi có hai hay nhiều tên cùng liên kết tới một ô nhớ ta gọi là sự lắm tên. Chẳng hạn các biến con trỏ trong ngôn ngữ Pascal cùng trỏ đến một ô nhớ. Sự lắm tên có thể làm giảm độ tin cậy do người lập trình không kiểm soát được giá trị được lưu trữ trong ô nhớ. Hãy xét ví dụ sau trong Pascal

```
Var p, q: ^integer;
Begin
     New(p);
     p^ := 50;
     q:= p;  {Cả q và p cùng trỏ đến một ô nhớ}
     writeln(p^, ' và ', q^);
     q^ := 20;
     writeln(p^, ' và ', q^);
end;
```

Kết quả thực hiện đoạn chương trình này là in ra hai dòng:
50 và 50
20 và 20

Trong khi nhiều người lầm tưởng hai dòng sẽ in ra là:
50 và 50
50 và 20

1.5.4 Chi phí

Chi phí của một ngôn ngữ cũng thường được quan tâm như là một tiêu chuẩn để đánh giá ngôn ngữ. Chi phí ở đây phải được hiểu là cả tiền bạc và thời gian. Chi phí này bao gồm:

- Chi phí đào tạo lập trình viên sử dụng ngôn ngữ. Chi phí này phụ thuộc vào sự giản dị của ngôn ngữ.
- Chi phí cài đặt chương trình. Chi phí này phụ thuộc vào tính dễ viết của ngôn ngữ.
- Chi phí dịch chương trình.
- Chi phí thực hiện chương trình.
- Chi phí bảo trì chương trình.

Xét về mặt nguồn gốc thì có thể phân ĐTDL làm hai loại: **ĐTDL tường minh** và **ĐTDL ẩn**.

ĐTDL tường minh *là một ĐTDL do người lập trình tạo ra* chẳng hạn như các biến, các hằng,… được người lập trình viết ra trong chương trình.

ĐTDL ẩn *là một ĐTDL được định nghĩa bởi hệ thống* như các ngăn xếp lưu trữ các giá trị trung gian, các mẩu tin kích hoạt chương trình con, các ô nhớ đệm của tập tin... Các ĐTDL này được phát sinh một cách tự động khi cần thiết trong quá trình thực hiện chương trình và người lập trình không thể truy cập đến chúng được.

2.2.3 Thuộc tính của ĐTDL

Thuộc tính của một ĐTDL là một tính chất đặc trưng của ĐTDL đó.

Mỗi ĐTDL có một tập hợp các thuộc tính để phân biệt ĐTDL này với ĐTDL khác.

Các ĐTDL sơ cấp chỉ có một thuộc tính duy nhất là kiểu dữ liệu của đối tượng đó. Các ĐTDL có cấu trúc có thêm các thuộc tính nhằm xác định số lượng, kiểu dữ liệu của các phần tử và các thuộc tính khác.

2.2.4 Giá trị dữ liệu

Giá trị dữ liệu (GTDL) của một ĐTDL sơ cấp có thể là một số, một ký tự hoặc là một giá trị logic tùy thuộc vào kiểu của ĐTDL đó.

Mỗi GTDL thường được biểu diễn bởi một dãy các *bit* trong bộ nhớ của máy tính.

Cần phân biệt hai khái niệm ĐTDL và GTDL. Một ĐTDL luôn luôn được biểu diễn bởi một khối ô nhớ trong bộ nhớ của máy tính trong khi một GTDL được biểu diễn bởi một dãy các *bit*. Khi nói rằng một ĐTDL A chứa một GTDL B có nghĩa là: khối ô nhớ biểu diễn cho A chứa dãy *bit* biểu diễn cho B.

GTDL của một ĐTDL có cấu trúc là một tập hợp các GTDL của các phần tử của ĐTDL có cấu trúc đó.

2.2.5 Thời gian tồn tại

Thời gian tồn tại (lifetime) của một ĐTDL là khoảng thời gian ĐTDL chiếm giữ bộ nhớ của máy tính. Thời gian này được tính từ khi ĐTDL được tạo ra cho đến khi nó bị hủy bỏ trong quá trình thực hiện chương trình.

2.2.6 Các mối liên kết

Một ĐTDL có thể tham gia vào nhiều mối liên kết trong thời gian tồn tại của nó. Các liên kết quan trọng nhất là:

- Sự liên kết của ĐTDL với một hoặc nhiều giá trị. Sự liên kết này có thể bị thay đổi bởi phép gán trị.
- Sự liên kết của một ĐTDL với một hoặc nhiều tên được tham chiếu trong quá trình thực hiện chương trình. Các liên kết này được thiết lập bởi sự khai báo và thay đổi bởi việc gọi và trả chương trình con.

- Các phép toán có thể thao tác trên các ĐTDL của kiểu.

Ví dụ, xét sự đặc tả kiểu dữ liệu mảng ta thấy:

1.- Các thuộc tính có thể bao gồm: số chiều, miền xác định của chỉ số đối với mỗi chiều và kiểu dữ liệu của các phần tử.

2.- Các giá trị có thể nhận của các phần tử mảng.

3.- Các phép toán có thể bao gồm: phép lựa chọn một phần tử mảng thông qua việc sử dụng chỉ số của phần tử đó, phép gán một mảng cho một mảng khác…

Phép toán

Các phép toán thao tác trên các ĐTDL là một **bộ phận không thể thiếu** của kiểu dữ liệu. Khi nói đến kiểu dữ liệu mà chúng ta không quan tâm đến các phép toán là chưa hiểu đầy đủ về kiểu dữ liệu đó. Mà dường như khiếm khuyết này lại hay xảy ra. Ví dụ khi nói đến kiểu integer trong ngôn ngữ Pascal, chúng ta chỉ nghĩ rằng đó là kiểu số nguyên, có các giá trị từ -32768 đến 32767, mà ít khi quan tâm đến các phép toán như +, -, *, … hay nói chính xác hơn chúng ta cứ nghĩ các phép toán này là mặc nhiên phải có. Trong tin học không có cái gì tự nhiên mà có cả, mọi cái hoặc do chúng ta tự tạo ra hoặc sử dụng cái có sẵn do người khác đã tạo ra. Nhấn mạnh việc có mặt các phép toán trong kiểu dữ liệu là để lưu ý chúng ta khi định nghĩa một kiểu dữ liệu mới, phải trang bị cho nó các phép toán cần thiết.

Có hai loại phép toán là các **phép toán nguyên thủy** được ngôn ngữ định nghĩa và các **phép toán do người lập trình định nghĩa** như là các chương trình con.

Phép toán trong NNLT về phương diện lôgic là một hàm toán học: đối với một đối số (argument) đã cho nó có một kết quả duy nhất và xác định.

Mỗi một phép toán có một miền xác định (domain) là tập hợp các đối số và một miền giá trị (range) là tập hợp các kết quả có thể tạo ra. Hoạt động của phép toán xác định kết quả được tạo ra đối với tập hợp bất kỳ các đối số đã cho. Giải thuật chỉ rõ làm thế nào để xác định kết quả đối với tập hợp bất kỳ các đối số đã cho là phương pháp phổ biến để xác định hoạt động của phép toán. Ngoài ra còn có những cách xác định khác chẳng hạn để xác định hoạt động của phép toán nhân chúng ta có thể cho một "bảng nhân" thay vì cho giải thuật của phép nhân hai số.

Để chỉ rõ miền xác định của phép toán, số lượng, thứ tự và kiểu dữ liệu của các đối số, tương tự miền giá trị, số lượng, thứ tự và kiểu dữ liệu của các kết quả người ta thường sử dụng các ký hiệu toán học.

Tên phép toán: Miền xác định -> Miền giá trị

Trong đó **Miền xác định = Kiểu đối số X Kiểu đối số X…**

(Miền xác định là tập tích Đề-các của các kiểu đối số)

Miền giá trị = Kiểu kết quả X Kiểu kết quả X ...

(Miền giá trị là tập tích Đề-các của các kiểu kết quả)

Khi nghiên cứu các phép toán trên các kiểu dữ liệu chúng ta cần lưu ý các vấn đề sau:

1.- Các phép toán không được xác định đầu vào một cách chắc chắn.

Một phép toán được xác định trên nhiều hơn một miền xác định thường chứa đựng các lỗi. Ví dụ các phép toán số học có thể xác định trên nhiều tập hợp số khác nhau có thể gây ra sự tràn số hoặc một kết quả sai lệch mà ta không thể kiểm soát được.

Ví dụ trong ngôn ngữ Pascal, phép cộng có thể xác định trên nhiều miền xác định khác nhau như *integer*, *real*,… nên có thể có những kết quả sai lệch như trong ví dụ sau:

```
var a, b : integer;
begin
        {1}     a:= 32767;
        {2}     b:= 30000;
        {3}     writeln(32767+30000);
        {4}     writeln(a+b);
end.
```

Kết quả của chương trình trên là 62767 và -2769.

Trong đó 62767 là kết quả của phép cộng 32767+30000. Đây là một kết quả đúng, do máy tính "hiểu" các số 32767 và 30000 là các số thực (real) và phép "+" trong lệnh {3} là "phép cộng các số thực". Ngược lại -2769 là kết quả sai của phép toán a+b. Về mặt toán học thì kết quả của a+b là 62767, nhưng kết quả của chương trình máy tính lại là -2769! Sở dĩ chương trình máy tính (ngôn ngữ Pascal) lại có kết quả này là do hai biến a và b được khai báo là các biến thuộc kiểu integer nên phép "+" trong lệnh {4} được hiểu là "phép cộng các số nguyên". Về nguyên tắc thì tổng a+b phải có giá trị thuộc kiểu integer nhưng do tập giá trị của kiểu integer là các số nguyên từ -32768 đến 32767 nên mới "sinh chuyện".

2.- Các đối số ẩn

Các phép toán trong chương trình thông thường sẽ được gọi với một tập hợp các đối số tường minh (explicit arguments). Tuy nhiên các phép toán có thể truy cập đến những đối số ẩn (implicit arguments) thông qua việc sử dụng các biến toàn cục hoặc tham chiếu các biến không cục bộ khác. Những đối số ẩn như thế sẽ gây khó khăn cho việc kiểm soát giá trị dữ liệu và do đó có thể ảnh hưởng đến kết quả của chương trình.

Ví dụ:

```
Var x: Integer;
Procedure P;
Begin
        x:= 0;
End;
Begin
        {1}      x:=10;
        {2}     P;
        {3}     Writeln(x);
End.
```

Trong ví dụ trên, chương trình con P thực hiện việc giá trị 0 cho biến toàn cục x. Trong chương trình chính, mặc dù ta mới gán 10 cho x (lệnh 1), nhưng sau khi gọi thủ tục P (lệnh 2) thì ở lệnh 3, x lại có giá trị 0. Việc chương trình con sử dụng biến không cục bộ như vậy sẽ dễ gây ngộ nhận cho người lập trình rằng x có giá trị 10, đặc biệt khi thủ tục P được định nghĩa ở một đoạn nào đó, xa đoạn chương trình chính.

3.- Hiệu ứng lề

Kiểm tra kiểu có thể được tiến hành trong lúc chạy chương trình (kiểm tra kiểu động) hoặc trong lúc biên dịch chương trình (kiểm tra kiểu tĩnh).

2.6.2 Kiểm tra kiểu động

Khái niệm:

Kiểm tra kiểu động là kiểm tra kiểu được thực hiện ***trong khi thực hiện chương trình.***

Thông thường kiểm tra kiểu động được thực hiện một cách tức thì trước khi thực hiện một phép toán.

Phương pháp thực hiện:

Để kiểm tra kiểu động người ta phải lưu trữ thông tin về kiểu của mỗi một ĐTDL cùng với ĐTDL đó. Trước khi thực hiện một phép toán thông tin về kiểu của mỗi một đối số được kiểm tra. Nếu kiểu của các đối số là đúng thì phép toán sẽ được thực hiện và kiểu của kết quả sẽ được ghi lại để dùng kiểm tra cho các phép toán sau, ngược lại sẽ có một thông báo lỗi về kiểu .

Ngôn ngữ sử dụng:

Kiểm tra kiểu động được sử dụng trong các **ngôn ngữ không khai báo** như SNOBOL4, LISP, APL. Trong các ngôn ngữ này không có sự khai báo kiểu cho biến. Kiểu dữ liệu của các biến A và B trong biểu thức "A+B" có thể thay đổi trong quá trình thực hiện chương trình. Trong những trường hợp như vậy, kiểu của A và B phải được kiểm tra động tại mỗi lần phép cộng được gọi thực hiện. Trong các ngôn ngữ không khai báo, các biến đôi khi được gọi là không định kiểu vì chúng không có kiểu cố định.

Ưu điểm:

Ưu điểm chủ yếu của kiểm tra kiểu động là tính mềm dẻo trong khi viết chương trình: không yêu cầu khai báo kiểu và kiểu của ĐTDL có thể thay đổi trong quá trình thực hiện chương trình. Người lập trình không phải lo lắng về kiểu dữ liệu.

Nhược điểm:

Tuy nhiên kiểm tra kiểu động cũng có một số yếu điểm như sau:

- Có khả năng bỏ sót lỗi về kiểu. Bởi vì việc kiểm tra động chỉ kiểm tra tại thời điểm thực hiện phép toán do đó các phép toán nằm trong nhánh chương trình không được thực hiện thì sẽ không được kiểm tra. Bất kỳ một nhánh chưa được kiểm tra nào đều có thể chứa các đối số có lỗi về kiểu và do đó các lỗi này có thể xuất hiện tại thời điểm sau đó. Ví dụ ta có một đoạn chương trình sau được viết trong một ngôn ngữ kiểm tra kiểu động:

  Nhập số a từ bàn phím;

  Nhập số b từ bàn phím;

  Nếu a > b Thì x := a + b

  Ngược lại x := a + “titi”;

  Nếu khi thực hiện đoạn chương trình này, người sử dụng luôn luôn nhập số a lớn hơn số b thì điều kiện a>b luôn luôn đúng nên không bao giờ chương trình

thực hiện lệnh x := a + "titi" do đó không bao giờ phát hiện lỗi về kiểu: a là một số, không thể cộng với "titi" là một chuỗi.

- Kiểm tra kiểu động đòi hỏi thông tin về kiểu phải được lưu giữ cho mỗi một ĐTDL trong quá trình thực hiện chương trình do đó yêu cầu về bộ nhớ phải lớn.
- Kiểm tra kiểu phải được tiến hành tức thì trước mỗi khi thực hiện một phép toán nên tốc độ thực hiện chương trình chậm.

2.6.3 Kiểm tra kiểu tĩnh

Khái niệm:

Kiểm tra kiểu tĩnh là sự kiểm tra kiểu ***được thực hiện trong quá trình dịch chương trình****.*

Phương pháp thực hiện:

Theo nguyên tắc kiểm tra kiểu tĩnh, thông tin về kiểu của ĐTDL phải được cung cấp cho bộ dịch. Thông tin này một phần được cung cấp bởi phép khai báo của người lập trình và một phần bởi ngôn ngữ .

Các thông tin bao gồm:

- Đối với mỗi một phép toán thì đó là số lượng, thứ tự và kiểu dữ liệu của đối số và kiểu của kết quả. Đối với các phép toán nguyên thuỷ thì việc định nghĩa ngôn ngữ sẽ cung cấp các thông tin này còn đối với chương trình con thì người lập trình phải xác định một cách tường minh.
- Đối với mỗi một biến thì đó là kiểu của biến.
- Đối với mỗi một hằng, thì đó là kiểu của đối tượng dữ liệu hằng. Ngữ nghĩa của một hằng trực kiện sẽ chỉ ra kiểu của nó, chẳng hạn "2" là một số nguyên, "2.3" là một số thực.

Kiểm tra kiểu tĩnh được thực hiện như sau: Thông qua đoạn đầu của chương trình, bộ biên dịch tập hợp thông tin từ sự khai báo trong chương trình vào trong bảng danh biểu (symbol table) nơi chứa thông tin về kiểu của các biến và chương trình con. Bộ biên dịch cũng sẽ có thông tin về các phép toán nguyên thuỷ được định nghĩa bởi ngôn ngữ, các hằng...Khi gặp một phép toán thì phải tra trong bảng danh biểu để xác định kiểu của mỗi một đối số có hợp lệ hay không. Chú ý rằng nếu phép toán là phép toán chung như đã nói ở trên thì có thể có nhiều kiểu hợp lệ cho một đối số. Nếu kiểu của đối số là hợp lệ thì kiểu kết quả được xác định và bộ biên dịch ghi lại thông tin này để kiểm tra các phép toán sau.

Ngôn ngữ sử dụng:

Kiểm tra kiểu tĩnh thường được sử dụng trong các **ngôn ngữ khai báo** tức là khi viết chương trình, các biến phải được khai báo kiểu trước khi sử dụng như Pascal, C…

Ưu điểm:

- Do phép kiểm tra kiểu tĩnh kiểm tra tất cả các phép toán có thể xuất hiện trong bất kỳ một lệnh nào của chương trình, tất cả các nhánh của chương trình đều được kiểm tra nên không thể có sự sót lỗi vê kiểu.

Nói chung các ngôn ngữ khác nhau thì phép gán cũng khác nhau.

Sự khác nhau đầu tiên là khác nhau về cú pháp, chẳng hạn ta có một số cú pháp lệnh gán như sau:

A := B	(Pascal hay Ada)
A = B	(C, C++, Fortran, PL/1 và SNOBOL4)
MOVE B TO A	(COBOL)
A <- B	(APL)
(SETQ A B)	(LISP)

Sự khác nhau thứ hai là kết quả trả về của phép gán trị. Nói chung trong các ngôn ngữ, lệnh gán trị không trả về kết quả. Chẳng hạn trong Pascal, đặc tả phép gán là **Phép gán (:=) Type1 x Type2 -> Void** với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và trả về một kết quả có kiểu void (có thể hiểu là không có kết quả trả về).

Trong một số ngôn ngữ như C, C++ và LISP, phép gán trả về trực tiếp một kết quả là một bản sao của giá trị được gán. Chẳng hạn trong C, sự đặc tả phép gán là

Phép gán (=) Type1 x Type2 -> Type3 với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và tạo ra một ĐTDL mới Type3 chứa bản sao giá trị của Type2, trả về Type3 như là một kết quả.

Vì phép gán trong Pascal không trả về một kết quả nên chúng ta chỉ sử dụng chức năng "gán trị" của nó mà thôi. Vì không có kết quả trả về nên mỗi một lệnh ta chỉ có thể viết một phép gán, chẳng hạn để gán giá trị 10 cho hai biến A và B ta phải viết hai lệnh B := 10; A := B; hoặc A := 10; B := 10;

Ngược lại khi lập trình bằng ngôn ngữ C, vì phép gán có trả về một kết quả nên ta có thể viết: A = B = 10; Trong đó phép gán B =10 vừa thực hiện chức năng "gán trị" giá trị 10 cho B vừa trả về 1 giá trị để gán tiếp cho A. Giá trị được trả về như là một kết quả của phép gán B cho A bị bỏ qua vì lệnh không chứa một phép toán nào sau đó nữa.

Phép gán trong ngôn ngữ C++ cũng có cùng cơ chế như trong C, vì vậy khi thiết kế toán tử gán cho một đối tượng nào đó (Overloading toán tử = trong khi xây dựng một lớp nào đó) ta phải viết:

```
<tên lớp> & operator= (const <tên lớp> & Obj)
{
  // Thực hiện việc gán dữ liệu;
  return *this;
}
```

Trong đó tên lớp là tên của lớp chúng ta đang định nghĩa. Phương thức này nhận vào một đối tượng Obj, thực hiện việc gán Obj cho đối tượng hiện hành và trả đối tượng hiện hành này về như một kết quả.

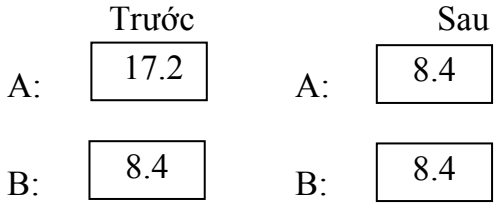
Ví dụ ta tạo một class có tên là point để biểu diễn cho một điểm trong mặt phẳng được đặc trưng bởi hai tọa độ x và y. Trong chương trình ta muốn gán các điểm cho nhau, nên trong khi định nghĩa class point, ta phải định nghĩa toán tử gán. Cụ thể như sau:

```
class point {
```

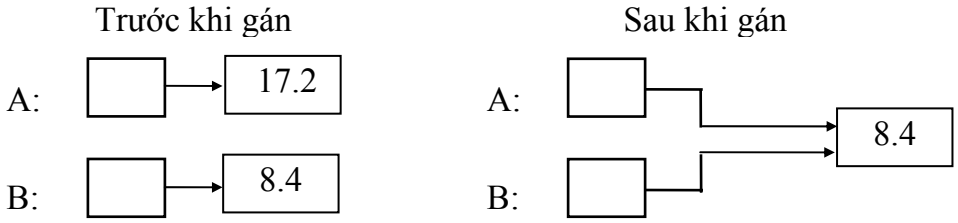
```
        float x;
        float y;
public:
        point() {x=0.0 ; y=0.0;}  // Phương thức xây dựng mặc nhiên
        point (float a, float b)  {x=a; y=b;} // Phương thức xây dựng bình thường
        point & operator= (const point & p ) //  Định nghĩa toán tử gán
        {
                x = p.x; y = p.y;  // Gán dữ liệu
                return * this;
        }
 }; // term
```

Sự khác nhau cuối cùng của phép gán là ở cách thức tiến hành gán trị. Xét lệnh gán của Pascal "A := B", ở Pascal cũng như một số ngôn ngữ khác, điều này có nghĩa là: "Gán bản sao của giá trị của biến B cho biến A". Bây giờ ta lại xét lệnh gán "A = B" của SNOBOL4. Trong SNOBOL4 thì nó có nghĩa là: "Tạo một biến tên A tham chiếu tới ĐTDL mà B đã tham chiếu". Trong SNOBOL4 cả A và B cùng trỏ tới một ĐTDL.

Pascal A := B (Sao chép ĐTDL khi gán)



SNOBOL4 A = B (Sao chép sự trỏ đến ĐTDL khi gán)



Cách thực hiện lệnh gán của SNOBOL4 rõ ràng là đã tạo ra một sự lắm tên.

2.8.2 Sự khởi tạo biến

Khởi tạo một biến là gán cho biến đó một giá trị đầu tiên.

Một biến khi được tạo ra thì sẽ được cấp phát ô nhớ nhưng nó vẫn chưa được khởi tạo. Khi nó được gán một giá trị đầu tiên thì mới được khởi tạo.

Các biến chưa được khởi tạo là nguồn gốc của các lỗi lập trình. Khi một biến được cấp phát ô nhớ mà chưa được khởi tạo thì trong ô nhớ của nó cũng có một giá trị ngẫu nhiên nào đó. Thường là một **giá trị rác** (Khi một ĐTDL nào trước đó đã bị hủy bỏ nhưng giá trị của ĐTDL này trong ô nhớ vẫn còn, giá trị này gọi là giá trị rác). Điều nguy hiểm là giá trị rác này vẫn là một giá trị hợp lệ. Vì thế chương trình có thể xử lý trên giá trị rác này một cách bình thường và chúng ta không thể kiểm sóat được kết quả xử lý đó.

17. Kiểm tra kiểu tĩnh được tiến hành trong lúc nào?
18. Kiểm tra kiểu động được tiến hành trong lúc nào?
19. Nêu các điểm mạnh của kiểm tra kiểu tĩnh.
20. Nêu các điểm yếu của kiểm tra kiểu tĩnh.
21. Nêu các điểm mạnh của kiểm tra kiểu động.
22. Nêu các điểm yếu của kiểm tra kiểu động.
23. Thông tin về kiểu trong kiểm tra kiểu tĩnh được lưu trữ ở đâu?
24. Thông tin về kiểu trong kiểm tra kiểu động được lưu trữ ở đâu?
25. Kiểm tra kiểu động được thực hiện trong ngôn ngữ nào?
26. Kiểm tra kiểu tĩnh được thực hiện trong ngôn ngữ nào?
27. Phép gán trị có trả về một kết quả không?
28. Thế nào là khởi tạo một biến?
29. Nếu trong một biểu thức có sử dụng một biến chưa được khởi tạo thì có thể đánh giá (định trị) được biểu thức đó không?

Giá trị nguyên lớn nhất đôi khi được biểu diễn như là một hằng xác định. Ví dụ trong Pascal là hằng MaxInt. Miền giá trị của kiểu số nguyên là tập các số nguyên từ -MaxInt đến MaxInt. Giá trị MaxInt được lựa chọn phản ánh giá trị nguyên lớn nhất có thể biểu diễn được trong phần cứng.

Đặc tả các phép toán: Trên ĐTDL nguyên thường có các nhóm phép toán chính như sau:

1.- Các phép tính số học

Các phép tính số học hai ngôi thường được định nghĩa là:

Bin_Op: Integer x Integer -> Integer.

Ở đây Bin_Op có thể là cộng (+), trừ (-), nhân (*), chia (/ hoặc DIV), lấy phần dư (MOD) hoặc một số phép toán tương tự khác.

Các phép tính số học một ngôi được định nghĩa: Unary_Op : Integer -> Integer

Ở đây Unary_Op có thể là âm (-), dương (+).

Các phép toán số học phổ biến khác thường được chứa trong thư viện chương trình con.

2.- Các phép toán quan hệ

Các phép toán quan hệ được định nghĩa là: Rel_Op : Integer x Integer -> Boolean

Ở đây Rel_Op có thể là bằng, khác, nhỏ hơn, lớn hơn, nhỏ hơn hoặc bằng, lớn hơn hoặc bằng. Phép toán quan hệ so sánh hai giá trị dữ liệu đối số và trả về kết quả là một đối tượng dữ liệu logic (đúng hoặc sai).

3.- Gán trị

Cũng như phép gán tổng quát, phép gán của số nguyên có thể trả về (với định nghĩa: **Assignment : Intger x Integer -> Integer)** hoặc không trả về một giá trị (với đinh nghĩa: **Assignment : Integer x Intger -> Void**) .

Sự cài đặt

Kiểu dữ liệu nguyên hầu hết được cài đặt một cách trực tiếp bằng cách dùng sự biểu diễn bộ nhớ được xác định bởi phần cứng và tập hợp các phép tính số học, các phép toán quan hệ nguyên thuỷ trong phần cứng cho các số nguyên. Thông thường sự biểu diễn này sử dụng một từ trong bộ nhớ hoặc một dãy các bytes để lưu trữ một số nguyên. Chẳng hạn ngôn ngữ Pascal đã sử dụng biểu diễn số nguyên bởi 1 từ (word) trong phần cứng của máy tính để biểu diễn cho một số integer.

3.5.2 Miền con của số nguyên

Sự đặc tả

Kiểu miền con của kiểu dữ liệu nguyên là một kiểu dữ liệu mà tập các giá trị của nó là một dãy các giá trị nguyên trong một khoảng giới hạn đã định.

Các dạng khai báo sau thường được sử dụng:

A : 1..10 (Pascal)

A : Integer Range 1..10 (Ada)

Như vậy về thuộc tính, kiểu miền con của kiểu số nguyên, có thuộc tính của kiểu số nguyên. Về giá trị, tập các giá trị của kiểu miền con được xác định rõ trong phép khai báo và cuối cùng, kiểu miền con cho phép sử dụng tập hợp phép toán như trong kiểu số nguyên bình thường.

Sự cài đặt

Kiểu miền con được cài đặt tương tự như cài đặt kiểu số nguyên.

Lợi ích của việc sử dụng kiểu miền con

Kiểu miền con có một ưu điểm nổi bật đó là kiểm tra kiểu tốt hơn kiểu số nguyên. Việc khai báo một biến kiểu miền con cho phép kiểm tra kiểu một cách nghiêm ngặt hơn khi thực hiện lệnh gán trị cho biến. Ví dụ để lưu trữ các tháng trong một năm ta có thể sử dung biến MONTH với khai báo kiểu miền con là MONTH: 1..12 thì lệnh gán MONTH := 0 là không hợp lệ và lỗi đó được tự động tìm thấy khi biên dịch. Nhưng nếu MONTH được khai báo là Integer thì lệnh gán trên là hợp lệ và lỗi chỉ có thể được tìm ra bởi người lập trình trong quá trình chạy thử.

3.5.3 Số thực dấu chấm động

Sự đặc tả

Tập hợp các giá trị thực dấu chấm động được xác định là một dãy số có thứ tự từ một số âm nhỏ nhất tới một số lớn nhất được xác định trong phần cứng của máy tính, nhưng các giá trị không được phân bố rời rạc đều trong giới hạn này.

Các phép tính số học, các phép toán quan hệ, phép gán đối với số thực cũng giống như đối với số nguyên. Một số phép toán khác cũng được các ngôn ngữ trang bị như là các hàm, chẳng hạn:

SIN : Real -> Real (Hàm SIN)

COS : Real -> Real (Hàm COSIN)

SQRT: Real -> Real (Hàm lấy căn bậc hai)

MAX : Real x Real -> Real (Hàm lấy giá trị lớn nhất)

Sự cài đặt

Sự biểu diễn bộ nhớ cho kiểu dữ liệu thực dấu chấm động dựa trên cơ sở biểu diễn phần cứng trong đó một ô nhớ được chia thành một phần định trị (mantissa) và một số mũ (exponent).

Các phép tính số học và các phép toán quan hệ trên kiểu số thực được hỗ trợ bởi phần cứng. Các phép toán khác phải được ngôn ngữ cài đặt như là các chương trình con.

Kiểu của mỗi một phần tử

Mỗi một phần tử của CTDL có một kiểu dữ liệu nào đó, ta gọi là kiểu phần tử. Kiểu phần tử có thể là một kiểu dữ liệu sơ cấp hoặc một CTDL. Các phần tử trong cùng một CTDL có thể có kiểu phần tử giống nhau hoặc khác nhau.

*Một CTDL được gọi là **đồng nhất** nếu tất cả các phần tử của nó đều có **cùng một kiểu.***

Ví dụ mảng, chuỗi ký tự, tập hợp và tập tin là các CTDL đồng nhất .

*Một CTDL được gọi là **không đồng nhất** nếu các phần tử của nó có kiểu **khác nhau.***

Ví dụ mẩu tin là CTDL không đồng nhất.

Tên để dùng cho các phần tử được lựa chọn

Để lựa chọn một phần tử của CTDL cho một xử lý nào đó người ta thường sử dụng một tên. Đối với cấu trúc mảng, tên có thể là một chỉ số nguyên hoặc một dãy chỉ số; đối với mẩu tin, tên là một tên trường. Một số kiểu cấu trúc dữ liệu như ngăn xếp và tập tin cho phép truy nhập đến chỉ một phần tử đặc biệt (phần tử đầu tiên hoặc phần tử hiện hành).

Số lượng lớn nhất các phần tử

Đối với CTDL có kích thước thay đổi như chuỗi ký tự hoặc ngăn xếp, đôi khi người ta quy định thuộc tính kích thước tối đa của cấu trúc để giới hạn số lượng các phần tử của cấu trúc.

Tổ chức cấu trúc

Tổ chức phổ biến nhất là một dãy tuần tự của các phần tử. Vector (mảng một chiều), mẩu tin, chuỗi ký tự, ngăn xếp, danh sách và tập tin là các CTDL có tổ chức kiểu này.

Một số cấu trúc còn được mở rộng thành dạng "nhiều chiều" ví dụ mảng nhiều chiều, mẩu tin mà các phần tử của nó là các mẩu tin, danh sách mà các phần tử của nó là danh sách.

4.3.2 Các phép toán trên cấu trúc dữ liệu

Một số các phép toán đặc thù của CTDL:

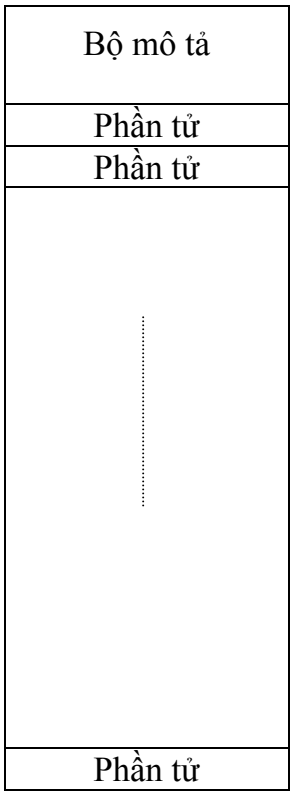
Phép toán lựa chọn phần tử của cấu trúc

Phép toán lựa chọn một phần tử là phép toán truy nhập đến một phần tử của CTDL và làm cho nó có thể được xử lý bởi một phép toán khác.

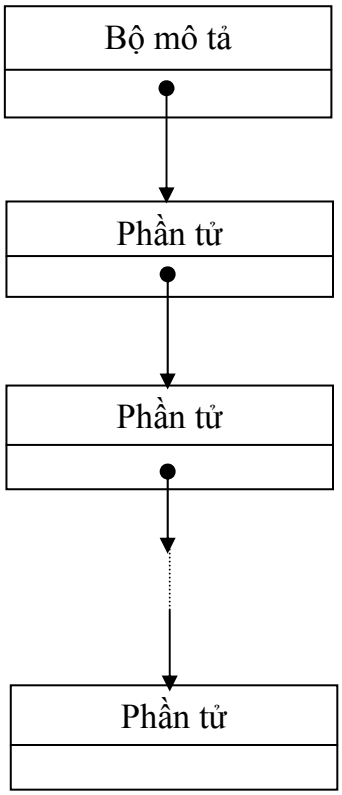
Có hai loại lựa chọn:

***Lựa chọn ngẫu nhiên** (hay còn gọi là lựa chọn trực tiếp) là sự lựa chọn một phần tử tùy ý của cấu trúc dữ liệu được truy nhập thông qua một cái tên.*

Ví dụ để lựa chọn một phần tử nào đó của mảng, ta chỉ ra chỉ số của phần tử đó, để lựa chọn một phần tử của mẩu tin ta sử dụng tên của phần tử đó.



Biểu diễn tuần tự



Biểu diễn liên kết

4.4.2 Cài đặt các phép toán trên cấu trúc dữ liệu

Phép toán lựa chọn một phần tử là phép toán cơ bản nhất trong các phép toán trên CTDL. Như trên đã trình bày, có hai cách lựa chọn là lựa chọn ngẫu nhiên và lựa chọn tuần tự và hai cách biểu diễn bộ nhớ là biểu diễn tuần tự và biêu diễn liên kết. Vì vậy ở đây chúng ta sẽ xét cách thực hiện mỗi một phương pháp lựa chọn với mỗi một phương pháp biểu diễn bộ nhớ.

Đối với biểu diễn tuần tự

Như trên đã trình bày, trong cách biểu diễn tuần tự, một khối ô nhớ liên tục sẽ được cấp phát để lưu trữ tòan bộ CTDL. Trong đó, vị trí đầu tiên của khối ô nhớ được gọi là **địa chỉ cơ sở**. Khoảng cách từ địa chỉ cơ sở đến vị trí của phần tử cần lựa chọn được gọi là **độ dời** của phần tử.

Cách thức truy xuất, được cho bởi tên hoặc chỉ số của phần tử (chẳng hạn chỉ số của một phần tử của mảng), sẽ xác định cách tính độ dời của phần tử như thế nào.

Để lựa chọn **ngẫu nhiên** một phần tử cần phải xác định vị trí thực của phần tử (tức là địa chỉ của ô nhớ lưu trữ phần tử đó) theo công thức:

Vị trí thực của phần tử = Địa chỉ cơ sở + độ dời của phần tử.

Lựa chọn **tuần tự** một dãy các phần tử của cấu trúc có thể theo các bước:

- Để chọn phần tử đầu tiên ta dùng cách tính địa chỉ cơ sở cộng với độ dời như đã nói ở trên.

Var b: ARRAY [-5..10] OF integer; Với khai báo này thì b là một véctơ có 16 phần tử (10 – (-5) + 1 = 16). Các phần tử được lựa chọn nhờ các chỉ số từ -5 đến 10.

Miền giá trị của chỉ số không nhất thiết là miền con của số nguyên, nó có thể là một liệt kê bất kỳ (hoặc 1 miền con của một liệt kê). Ví dụ:

```
Type
        Ngay = (Chu_nhat, Hai, Ba, Tu, Nam, Sau, Bay);
var
        c : ARRAY [Ngay] OF Integer ;
```

Khai báo này xác đinh véctơ c có 7 phần tử là các số integer, các phần tử của c được lựa chọn nhờ các "chỉ số" từ Chu_nhat đến Bay.

Khai báo véctơ trong ngôn ngữ C là **<kiểu phần tử> <tên biến> [<số lượng phần tử>]**.

Ví dụ int d[10];

Khai báo này xác định véctơ d có 10 phần tử các số int, các phần tử này được lựa chọn nhờ các chỉ số từ 0 đến 9.

Đặc tả các phép toán trên véctơ

Các phép toán trên véctơ bao gồm:

Phép toán **lựa chọn một phần tử** của véctơ là **phép lấy chỉ số**, được viết bằng tên của véctơ theo sau là chỉ số của phần tử được lựa chọn đặt trong cặp dấu []. Như vậy phép lựa chọn một phần tử của véctơ là phép **lựa chọn trực tiếp**.

Ví dụ, với các khai báo trong các ví dụ thuộc phần đặc tả thuộc tính nói trên,

Các phần tử của véctơ a được lựa chọn bằng cách viết a[1], a[2], …, a[10].

Các phần tử của véctơ b được lựa chọn bằng cách viết b[-5], b[-4], …, b[10].

Các phần tử của véctơ c được lựa chọn bằng cách viết c[Chu_nhat], c[Hai], …, c[Bay].

Các phần tử của véctơ d được lựa chọn bằng cách viết d[0], d[1], …, d[9].

Chỉ số có thể là một hằng hoặc một biến (nói chung là một biểu thức), ví dụ a[i] hay a[i+2]. Nhờ chỉ số là một biểu thức nên việc lập trình trở nên đơn giản hơn nhiều nhờ tính khái quát của chỉ số.

Ví dụ để in ra giá trị của 10 phần tử trong véctơ a, thay vì ta phải viết 10 lệnh in các phần tử cụ thể theo kiểu writeln(a[1]); writeln(a[2]); writeln(a[3]); … ta chỉ cần viết một lệnh for i:=1 to 10 do writeln(a[i]);

Các phép toán khác trên véctơ bao gồm các phép toán tạo và hủy bỏ véctơ, gán hai véctơ cho nhau và các phép toán thực hiện như các phép toán số học trên từng cặp 2 véctơ có cùng kích thước. Chẳng hạn phép cộng 2 véctơ (cộng các phần tử tương ứng). Tùy thuộc vào ngôn ngữ mà các phép toán này có hoặc không có.

4.5.3 Cài đặt một véctơ

Biểu diễn bộ nhớ

Biểu diễn bộ nhớ **tuần tự** được sử dụng để biểu diễn cho một véctơ.

4.6.1 Sự đặc tả và cú pháp

Đặc tả thuộc tính

Mảng nhiều chiều tương tự như véctơ nhưng chỉ có một thuộc tính khác véctơ là mỗi một chiều phải có một tập chỉ số tương ứng.

Chẳng hạn khai báo cho một mảng hai chiều có thể đươc viết dưới dạng

ARRAY[LB1..UB1, LB2..UB2] OF <Kiểu phần tử>

Trong đó tập chỉ số 1 có các giá trị từ LB1 đến UB1, tập chỉ số 2 có các giá trị từ LB2 đến UB2.

Như vậy số lượng các phần tử của mảng hai chiều sẽ là **(UB1-LB1+1)*(UB2-LB2+1)**

Ví dụ sự khai báo của Pascal:

M= array [1..3, -1..2] of Integer;

Sự khai báo này cho ta thấy mảng M có hai chiều, chiều thứ nhất được xác định bởi tập chỉ số 1..3 và chiều thứ hai được xác định bởi tập chỉ số -1..2. Có thể xem đây là một ma trận có 3 dòng và 4 cột, như vậy sẽ có 12 phần tử, mỗi phần tử có thể lưu trữ một số integer.

Đối với các mảng có số chiều nhiều hơn hai thì cách làm cũng tương tự như mảng hai chiều.

Đặc tả phép toán

Phép lựa chọn một phần tử được thực hiện bằng cách chỉ ra tên mảng và chỉ số của mỗi một chiều.

Chẳng hạn để lựa chọn một phân tử của ma trận ta viết tên ma trận, theo sau là cặp chỉ số dòng, cột phân cách nhau bởi dấu phẩy và đặt trong cặp dấu [], ví dụ M[2,0].

Như vậy phép lựa chọn một phần tử của mảng nhiều chiều là phép **lựa chọn trực tiếp**.

4.6.2 Sự cài đặt

Sự biểu diễn bộ nhớ

Sự biểu diễn bộ nhớ đối với mảng nhiều chiều tương tự như sự biểu diễn bộ nhớ đối với véctơ. Nghĩa là cũng sử dụng sự **biểu diễn tuần tự** và khố ô nhớ được chia làm hai phần: bộ mô tả và bộ nhớ cho các phần tử. Bộ mô tả của mảng giống bộ mô tả của véctơ ngoại trừ mỗi một chiều có một cận dưới và cận trên của tập chỉ số của chiều đó. Trong bộ nhớ dành cho các phần tử ta cũng lưu trữ liên tiếp các phần tử theo một trật tự nào đó.

Với ma trận, về mặt logic thì ma trận là một bảng gồm m dòng và n côt, mỗi một ô là một phần tử, nhưng bộ nhớ lại chỉ gồm các ô liên tiếp nhau, vì thế ta phải lưu trữ ma trận theo trật tự dòng hoặc theo trật tự cột.

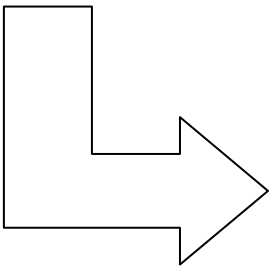
Lưu trữ theo **trật tự dòng** có nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong dòng thứ nhất, tiếp đến là các phần tử trong dòng thứ hai... cho đên dòng cuối cùng.

Lưu trữ theo **trật tự cột** nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong cột thứ nhất, tiếp đến là các phần tử trong cột thứ hai... cho đến cột cuối cùng.

Chẳng hạn với khai báo M: ARRAY [1..3,-1..2] OF Integer; ta có hình ảnh biểu diễn trong bộ nhớ như các hình sau:

Cấu trúc logic của ma trận M

M[1,-1]	M[1,0]	M[1,1]	M[1,2]
M[2,-1]	M[2,0]	M[2,1]	M[2,2]
M[3,-1]	M[3,0]	M[3,1]	M[3,2]

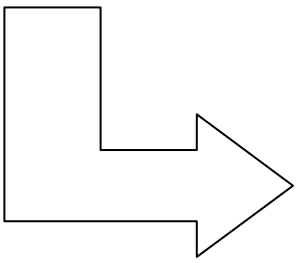


Lưu trữ ma trận M theo **trật tự dòng**

Bộ mô tả		
	Ma trận M	Kiểu dữ liệu
	LB1 (= 1)	Cận dưới của tập chỉ số thứ nhất
	UB1 (= 3)	Cận trên của tập chỉ số thứ nhất
	LB2 (= -1)	Cận dưới của tập chỉ số thứ hai
	UB2 (= 2)	Cận trên của tập chỉ số thứ hai
Bộ nhớ cho Các phần tử	M[1,-1]	
	M[1,0]	Dòng thứ nhất
	M[1,1]	
	M[1,2]	
	M[2,-1]	Dòng thứ hai
	M[2,0]	
	...	.....
	M[3,2]	

Cấu trúc logic của ma trận M

M[1,-1]	M[1,0]	M[1,1]	M[1,2]
M[2,-1]	M[2,0]	M[2,1]	M[2,2]
M[3,-1]	M[3,0]	M[3,1]	M[3,2]



Lưu trữ ma trận M theo **trật tự cột**

Bộ mô tả		
	Ma trận M	Kiểu dữ liệu
	LB1 (= 1)	Cận dưới của tập chỉ số thứ nhất
	UB1 (= 3)	Cận trên của tập chỉ số thứ nhất
	LB2 (= -1)	Cận dưới của tập chỉ số thứ hai
	UB2 (= 2)	Cận trên của tập chỉ số thứ hai
Bộ nhớ cho Các phần tử	M[1,-1]	
	M[2,-1]	Cột thứ nhất
	M[3,-1]	
	M[1,0]	
	M[2,0]	Cột thứ hai
	M[3,0]	
	...	.....
	M[3,2]	

Đặc tả phép toán

Lựa chọn một phần tử là phép toán cơ bản cuả mẩu tin. Phép toán này được thực hiện bằng cách chỉ ra tên trực kiện của phần tử.

Ví dụ để lựa chọn phần tử thứ 4 của mẩu tin Nhan_vien ta viết: Nhan_vien.Luong.

Phép toán lựa chọn một phần tử của mẩu tin là sự **lựa chọn trực tiếp**.

Mặc dù đều là lựa chọn trực tiếp, nhưng có khác biệt so với cách lựa chọn phần tử của véctơ. Điểm khác biệt ở đây là: đối với véctơ, ta có thể sử dụng giá trị của một biểu thức làm chỉ số, chẳng hạn VECTO[i+1], còn đối với mẩu tin thì bắt buộc phải chỉ rõ tên trực kiện, chứ không thể là biểu thức.

Ngoài phép toán lựa chọn phần tử, phép gán các mẩu tin có cùng cấu trúc là một phép toán phổ biến được các ngôn ngữ đưa vào. Chẳng hạn Nhan_vien := InputRec trong đó InputRec có các thuộc tính giống hệt Nhan_vien.

4.7.3 Sự cài đặt

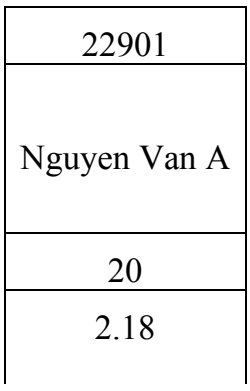
Biểu diễn bộ nhớ

Biểu diễn bộ nhớ **tuần tự** được sử dụng để lưu trữ một mẩu tin. Một khối liên tục các ô nhớ được dùng để lưu trữ cho một mẩu tin, trong khối đó, mỗi ô biểu diễn cho một trường. Có thể cũng cần sử dụng bộ mô tả riêng cho từng trường để lưu trữ thuộc tính của các trường đó. Do các trường có kiểu khác nhau nên ô nhớ dành cho chúng cũng có kích thước khác nhau.

Giải thuật thực hiên phép toán

Việc lựa chọn phần tử được thực hiện một cách dễ dàng vì tên trường được biết đến thông qua việc dịch chứ không phải được tính toán thông qua việc thực hiện như đối với véctơ. Việc khai báo mẩu tin còn cho phép xác định kích thước và vị trí của nó trong ô nhớ thông qua việc dịch. Kết quả là độ dời của phần tử bất kỳ có thể được tính thông qua việc dịch.

Chẳng hạn với mẩu tin Nhan_vien, các phần tử của nó được lưu trữ trong bộ nhớ như sau:



Vị trí của một phần tử bất kỳ được tính một cách dễ dàng. Chẳng hạn

Vị trí của Tuoi = α + Kích thước của Ma + Kích thước của Ho_ten.

Trong đó α là địa chỉ cơ sở của khối ô nhớ biểu diễn cho Nhan_vien.

Phép toán gán toàn bộ một mẩu tin cho một mẩu tin khác có cùng cấu trúc được thực hiện một cách đơn giản là copy nội dung khối ô nhớ biểu diễn cho mẩu tin thứ nhất sang khối ô nhớ biểu diễn cho mẩu tin thứ 2.

```
                    (gia_cong_nhat: Real);
    END;
```

Khai báo trên định nghĩa một mẩu tin có cấu trúc thay đổi. Mẩu tin luôn luôn có các trường Ho_Ten, Ngay_Cong, Luong và Loai. Khi giá trị của Loai = "bien_che" thì mẩu tin còn có các trường He_So và Nghi_Bhxh, trong khi đó nếu giá trị của Loai = "hop_dong" thì nó lại có trường Gia_Cong_Nhat.

Đặc tả phép toán

Phép toán lựa chọn các phần tử của mẩu tin có cấu trúc thay đổi cũng giống như mẩu tin bình thường. Chẳng hạn ta có thể sử dụng Cong_Nhan.Luong, Cong_Nhan.He_So hay Cong_Nhan.Gia_Cong_Nhat. Tuy nhiên các trường thuộc phần động chỉ tồn tại trong một thời điểm nhất định do đó khi chúng ta truy xuất tới một tên trường mà nó không tồn tại thì sẽ bị lỗi. Trường Loai trong ví dụ trên là rất quan trọng vì nó chỉ ra phần động nào của mẩu tin được sử dụng trong quá trình thực hiện chương trình. Người đọc có thể tham khảo ví dụ tương đối hoàn chỉnh viết bằng Pascal.

```
uses crt;
Const luong_toi_thieu = 290000;
Type
  Loai_cong_nhan = (bien_che, hop_dong);
  Cong_nhan = Record
            ho_ten : String[20];
            Ngay_cong : real;
            luong : real;
            Case loai: Loai_cong_nhan of
                 bien_che: (He_so, so_ngay_nghi_BHXH : real);
                 hop_dong: (don_gia: real);
  end;
  danh_sach_cong_nhan = Array[1..10] of cong_nhan;
Var
  n : integer; ho_so : danh_sach_cong_nhan;

{Nhập danh sách công nhân, và các thông tin liên quan đến lao động}
Procedure Nhap (var ho_so: danh_sach_cong_nhan; var n: integer);
Var
  i: integer;
  loaicn : char;
Begin
  write('So cong nhan: '); readln(n);
  For i:=1 to n do with ho_so[i] do begin
      Writeln('Cong nhan ',i);
      Write('Ho va Ten: '); readln(ho_ten);
      Write('Loai cong nhan: A la bien che, B la hop dong ');
```

```
      readln(loaicn);
      If Upcase(loaicn) ='A' then loai := bien_che else loai := hop_dong;
      write('So ngay cong: '); readln(ngay_cong);
      if loai = bien_che then begin
        write('He so: '); readln(he_so);
        write('So ngay nghi bao hiem: '); readln(so_ngay_nghi_BHXH);
      end else begin
        write('Don gia hop dong: '); readln(don_gia);
      end;
  end; { with Ho_so[i] }
end; {nhap}
{Tính lương cho từng công nhân, theo công thức của từng loại công nhân}
Procedure Tinh_luong (var ho_so: danh_sach_cong_nhan; n: integer);
Var
  i : integer; luong_binh_quan: real;
begin
  for i:=1 to n do with ho_so[i] do begin
      if loai = bien_che then begin {tính lương của công nhân biên chế}
        luong_binh_quan := he_so * luong_toi_thieu/20;
        luong :=  ngay_cong * luong_binh_quan +
                  so_ngay_nghi_BHXH * luong_binh_quan*0.80;
      end else {tính lương của công nhân hợp đồng}
        luong := ngay_cong * don_gia;
   end;  { with Ho_so[i] }
end; {Tinh_luong }

Procedure In_luong (ho_so: danh_sach_cong_nhan; n: integer);
Var
  i : integer;
begin
  for i:=1 to n do with ho_so[i] do begin
     Write(ho_ten:25);
     If loai = bien_che then write('Bien che':10)
     else write('Hop dong':10);
     write(ngay_cong:5:1);
     if loai = bien_che then begin
        write(he_so:5:1);
        write(so_ngay_nghi_BHXH:5:1);
     end else
        write(don_gia:10:2);
     writeln(luong:10:2);
```

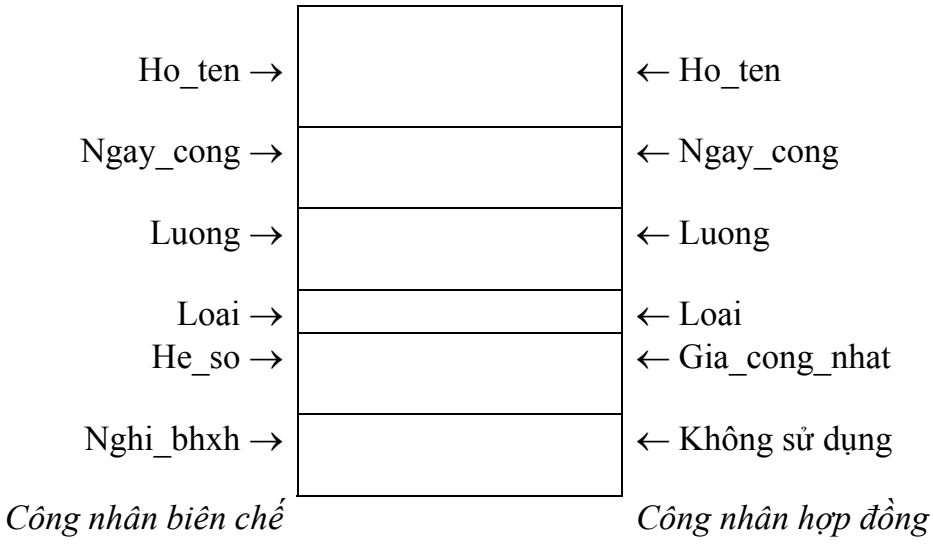
```
   end; { with Ho_so[i] }
   end; { In_luong }
begin {Chuong trinh chinh}
  nhap(ho_so,n);
  tinh_luong(ho_so,n);
  in_luong(ho_so,n);
  readln;
end.
```

4.8.2 Cài đặt mẩu tin có cấu trúc thay đổi

Biểu diễn bộ nhớ

Biểu diễn tuần tự sẽ được sử dụng để biểu diễn cho một mẩu tin có cấu trúc thay đổi.

Thông qua việc dịch, tổng bộ nhớ cần để lưu các phần tử của mỗi một phần động được xác định và bộ nhớ được cấp phát đủ để lưu trữ mẩu tin với phần động lớn nhất. Chẳng hạn với mẩu tin cong_nhan ta có mô hình lưu trữ như trong hình vẽ sau:



Vì khối ô nhớ đủ lớn để lưu trữ phần động lớn nhất nên có đủ chỗ cho bất kỳ một phần động nào nhưng đối với những phần động nhỏ hơn sẽ không sử dụng tới một số ô nhớ đã được cấp phát.

Với mẩu tin có cấu trúc thay đổi, rõ ràng ta đã tiết kiệm được một số ô nhớ so với mẩu tin bình thường.

Giải thuật thực hiện phép toán

Lựa chọn một phần tử của phần động cũng giống như lựa chọn một phần tử bình thường, qua việc dịch thì độ dời của phần tử được lựa chọn sẽ được tính toán và qua việc thực hiện thì độ dời được cọng vào địa chỉ cơ sở của khối để xác định vị trí của phần tử.

Ghép là phép toán nhập hai chuỗi ký tự tạo ra một chuỗi mới ví dụ nếu "//" là ký hiệu của phép ghép thì "BLOCK"//"HEAD" cho ra "BLOCKHEAD". Turbo Pascal sử dụng toán tử "+" cho phép toán ghép chuỗi.

b/ Các phép toán quan hệ trên chuỗi

Các phép toán quan hệ thông thường như bằng, nhỏ hơn, lớn hơn... trên kiểu ký tự có thể được mở rộng cho chuỗi ký tự. Tập hợp các ký tự cơ bản luôn luôn có một thứ tự. Mở rộng thứ tự này cho chuỗi ký tự thành thứ tự alphabe trong đó chuỗi A nhỏ hơn chuỗi B nếu ký tự đầu tiên của A nhỏ hơn ký tự đầu tiên của B hoặc hai ký tự đầu tiên tương ứng của chúng bằng nhau và ký tự thư hai của A nhỏ hơn ký tự thứ hai của B... Nếu chuỗi A ngắn hơn chuỗi B thì A được mở rộng bằng cách thêm vào các ký tự trắng cho dài bằng B để so sánh.

c/ Chọn chuỗi con dùng chỉ số chỉ vị trí của ký tự

Nhiều ngôn ngữ cung cấp một phép toán chọn chuỗi con của một chuỗi bằng cách cho vị trí của ký tự đầu tiên và ký tự cuối cùng của nó (hoặc vị trí của ký tự đầu tiên và chiều dài của chuỗi con). Ví dụ trong FORTRAN, lệnh NEXT = STR(6:10) là gán 5 ký tự, bắt đầu từ vị trí thứ 6 đến vị trị thứ 10 của chuỗi STR cho biến chuỗi NEXT.

d/ Định dạng nhập - xuất

Định dạng nhập xuất là phép toán dùng để thay đổi dạng nhập vào hoặc xuất ra của các chuỗi ký tự. Nhập xuất có định dạng là nét nổi bật của FORTRAN và PL/1.

e/ Chọn chuỗi con dùng so mẫu

Thông thường người ta không biết vị trí của một chuỗi con cần chọn trong một chuỗi lớn hơn nhưng quan hệ của nó với một chuỗi con khác thì có thể biết. Ví dụ chuỗi các chữ số sau dấu chấm thập phân hay chuỗi đứng sau một khoảng trống. Phép so mẫu có một đối số thứ nhất để xác định dạng của chuỗi con cần chọn (chẳng hạn độ dài của nó). Đối số thứ hai của phép toán so mẫu là chuỗi ký tự dùng để tìm trong chuỗi (chẳng hạn dấu chấm thập phân). Như vậy kết quả của phép toán so mẫu là chọn được một chuỗi con bắt đầu từ sau dấu chấm thập phân và có độ dài đã cho.

4.9.2 Cài đặt

Biểu diễn bộ nhớ

Mỗi một phương pháp đặc tả chuỗi có một cách biểu diễn bộ nhớ tương ứng.

Đối với chuỗi có độ dài được khai báo cố định thì dùng véctơ của các ký tự. Ví dụ chuỗi được khai báo có độ dài 8 và được dùng để lưu trữ chuỗi EINSTEIN (cũng có 8 ký tự):

E	I	N	S	T	E	I	N

Đối với chuỗi có độ dài thay đổi trong một giới hạn đã được khai báo thì vẫn dùng véctơ của các ký tự, trong đó sử dụng hai ô làm bộ mô tả chứa giá trị thể hiện độ dài lớn nhất đã được khai báo và độ dài hiện hành của chuỗi. Ví dụ chuỗi được khai báo có độ dài 12 và được dùng để lưu trữ chuỗi EINSTEIN (có 8 ký tự):

4.11.2 Sự đặc tả

Đặc tả thuộc tính

Có hai loại con trỏ khác nhau:

Con trỏ chỉ có thể tham chiếu tới các ĐTDL cùng kiểu

Đây là phương pháp được dùng trong Pascal và Ada.

Ví dụ trong Pascal:

Var p: ^integer chỉ ra rằng p là một biến con trỏ chứa địa chỉ của ô nhớ lưu trữ được một số integer.

Var q: ^VECT chỉ ra rằng q là một biến con trỏ chứa địa chỉ của khối ô nhớ của ĐTDL thuộc kiểu véctơ VECT nào đó.

Con trỏ có thể tham chiếu tới các ĐTDL khác kiểu nhau

Đây là cách được dùng trong các ngôn ngữ như SNOBOL4, nơi mà đối tượng dữ liệu mang bộ mô tả kiểu trong quá trình thực hiện và phép kiểm tra kiểu động được sử dụng.

Đặc tả phép toán

Các phép toán bao gồm:

Phép toán cấp phát ô nhớ động: Phép toán này dùng để cấp phát ô nhớ cho đối tượng dữ liệu mới và trả địa chỉ của ô nhớ đó về trong biến con trỏ. Đây là phép toán quan trọng nhất của kiểu con trỏ. Phép toán này có hai điểm khác biệt với việc tạo ra đối tượng dữ liệu tĩnh (bằng cách khai báo) là: Đối tượng dữ liệu được tạo ra không cần có tên vì nó được truy xuất thông qua con trỏ và đối tượng dữ liệu có thể được tạo ra một cách động trong quá trình thực hiện chương trình. Trong Pascal và Ada thì phép toán này có tên là NEW. Ví dụ NEW(p).

Phép toán truy xuất ô nhớ được cấp phát động: Để truy xuất đến giá trị dữ liệu lưu trong khối ô nhớ cấp phát động ta phải sử dụng địa chỉ của khối ô nhớ thông qua tên con trỏ (vì khối ô nhớ này không có tên). Ví dụ q^[5] là phần tử thứ 5 của véctơ Vect được trỏ bởi q.

Phép toán thu hồi ô nhớ được cấp phát động: Phép toán này cho phép giải phóng ô nhớ đã cấp phát. Trong Pascal, dùng phép toán DISPOSE.

Ví dụ sau trong Pascal minh hoạ tổng hợp các điều nói trên:

```
Type
Vect = ARRAY[1..10] of Integer;
{Lúc này bộ nhớ cho Vect chưa được cấp phát}
VAR
p: ^Vect;
{Khai báo p là một biến con trỏ chứa địa chỉ của khối ô
nhớ lưu trữ ĐTDL thuộc kiểu véctơ Vect. Khi dịch đến đây
thì ô nhớ cho p sẽ được cấp phát}
Begin
```

Đây là các phép toán được định nghĩa tương tự như trong toán học.

4.12.2 Cài đặt

Để cài đặt một tập hợp, ta có thể sử dụng một trong hai phương pháp sau:

Véctơ bit

Biểu diễn bộ nhớ

Tập hợp được biểu diễn bởi một chuỗi các bit. Cách tiếp cận này phù hợp cho một không gian nhỏ. Chẳng hạn ta có một không gian gồm n phần tử được đánh số thứ tự e_1, e_2, ... e_n. Một tập hợp các phần tử được chọn từ không gian này được biểu diễn bởi một véctơ có n bit, trong đó nếu bit thứ i có giá trị 1 thì phần tử e_i thuộc vào tập hợp, ngược lại bit thứ i có giá trị 0 thì e_i không thuộc tập hợp.

Giải thuật thực hiện các phép toán

Với cách biểu diễn này, việc thêm một phần tử vào trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 1. Việc xóa một phần tử trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 0. Phép kiểm tra một phần tử có thuộc tập hợp hay không được thực hiện bằng cách kiểm tra bit tương ứng có giá trị là 1 hay 0. Phép hợp của hai tập hợp tương ứng với phép toán logic OR của hai véctơ bit. Phép giao của hai tập hợp tương ứng với phép toán logic AND của hai véctơ bit. Hiệu của hai tập hợp tương ứng với phép toán logic AND của véctơ bit thứ nhất với phần bù của véctơ bit thứ hai. Các phép toán logic trên các véctơ bit đều được hỗ trợ bởi phần cứng.

Ví dụ Ta có một không gian bao gồm 5 phần tử 1,2,3,4,5. Khi đó

Tập hợp A = {1,2,4,5} được biểu diễn bởi véctơ (1,1,0,1,1)

Tập hợp B = {2,3,4} được biểu diễn bởi véctơ (0,1,1,1,0)

Do đó $A \cup B$ sẽ là tập {1,2,3,4,5} bởi vì (1,1,0,1,1) OR (0,1,1,1,0) = (1,1,1,1,1)

$A \cap B$ sẽ là tập hợp {2,4} bởi vì (1,1,0,1,1) AND (0,1,1,1,0) = (0,1,0,1,0)

A\B sẽ là tập hợp {1,5} bởi vì phần bù của (0,1,1,1,0) là (1,0,0,0,1) và

(1,1,0,1,1) AND (1,0,0,0,1) = (1,0,0,0,1)

Ưu điểm

Dễ dàng cài đặt các phép toán trên tập hợp với tốc độ thực hiện nhanh nhờ sử dụng các phép toán của phần cứng.

Nhược điểm

Không thể biểu diễn cho tập hợp mà các phần tử của nó có thể lấy từ một không gian lớn, có số lượng các phần tử bất kỳ.

Bảng băm

Biểu diễn bộ nhớ

Phương pháp này thích hợp cho các không gian lớn. Theo đó mỗi tập hợp được biểu diễn bởi một bảng băm (bảng băm mở). Mỗi phần tử của tập hợp được lưu trữ trong

Các phép toán chủ yếu đối với tập tin tuần tự là:

1/ OPEN

Thông thường một tập tin phải được mở trước khi sử dụng. Phép toán OPEN chỉ ra tên của tập tin và mode truy xuất tập tin (READ hoặc WRITE). Nếu mode là READ thì tập tin phải chắc chắn là đã tồn tại. Hệ diều hành cung cấp đặc tính của tập tin, cấp phát ô nhớ cần thiết cho vùng nhớ đệm và đặt con trỏ tập tin vào phần tử đầu tiên. Nếu mode là WRITE thì hệ điều hành tạo một tập tin rỗng, nếu tập tin đã tồn tại thì xóa tất cả các phần tử của tập tin để nó rỗng, con trỏ tập tin chỉ vào vị trí đầu tập tin rỗng.

Ví dụ trong Pascal thủ tục RESET mở một tập tin để READ và thủ tục REWRITE mở một tập tin để WRITE.

2/ READ

Phép toán READ chuyển nội dung của phần tử hiện hành của tập tin (được chỉ định bởi con trỏ tập tin) vào biến được chỉ định trong chương trình.

3/ WRITE

Phép toán WRITE tạo ra một phần tử mới của tập tin tại vị trí hiện hành và chuyển nội dung của biến chương trình được chỉ định vào phần tử mới.

4/ Kiểm tra cuối tập tin

Là phép toán xác định xem vị trí của con trỏ tập tin có nằm sau phần tử cuối cùng của tập tin hay không.

5/ CLOSE

Khi việc xử lý tập tin đã hoàn tất thì nó phải được đóng lại. Thông thường tập tin được đóng một cách tự động khi chương trình kết thúc. Tuy nhiên nếu muốn thay đổi mode truy nhập tập tin từ WRITE sang READ hoặc ngược lại thì tập tin phải được đóng một cách tường minh bằng phép toán CLOSE và sau đó mở lại cho mode mới.

Phép cài đặt

Trong hầu hết các hệ máy tính, thì hệ điều hành chịu trách nhiệm chủ yếu về việc cài đặt tập tin bởi vì tập tin được tạo ra và sử dụng bởi nhiều ngôn ngữ lập trình khác nhau. Ngôn ngữ lập trình chỉ làm một việc là cung cấp những cấu trúc dữ liệu cần thiết để giao diện với hệ điều hành.

Các phép toán trên tập tin được cài đặt một cách chủ yếu bằng cách gọi các phép toán của hệ điều hành.

Khi chương trình mở một tập tin, thì bộ nhớ lưu trữ một bảng thông tin về tập tin (FIT) (File Information Table) và một bộ nhớ đệm (buffer) được cung cấp. Phép toán OPEN của hệ diều hành sẽ lưu trữ thông tin về vị trí và các đặc tính của tập tin vào trong bảng FIT.

Nếu tập tin được mở để ghi thì khi phép toán WRITE chuyển một phần tử để nối vào cuối tập tin, thì dữ liệu được gửi cho phép toán WRITE của hệ điều hành. Phép toán WRITE của hệ điều hành sẽ lưu dữ liệu vào trong vị trí có thể của bộ nhớ đệm. Khi trong bộ nhớ đệm đã tích lũy được một khối các phần tử thì khối đó sẽ được chuyển sang bộ nhớ ngoài (đĩa hoặc băng từ). Quá trình tiếp tục của phép toán WRITE được

2. Nêu tên các thuộc tính của cấu trúc dữ liệu?
3. Thế nào là cấu trúc dữ liệu đồng nhất?
4. Thế nào là cấu trúc dữ liệu không đồng nhất?
5. Thế nào là cấu trúc dữ liệu có kích thước cố định?
6. Thế nào là cấu trúc dữ liệu có kích thước thay đổi?
7. Cho ví dụ về một cấu trúc dữ liệu đồng nhất.
8. Cho ví dụ về một cấu trúc dữ liệu không đồng nhất.
9. Cho ví dụ về một cấu trúc dữ có kích thước cố định.
10. Cho ví dụ về một cấu trúc dữ có kích thước không cố định.
11. Trên cấu trúc dữ liệu thường có các phép toán nào?
12. Kể tên các phương pháp lựa chọn một phần tử của cấu trúc dữ liêu?
13. Nêu tên các phương pháp biểu diễn cấu trúc dữ liệu trong bộ nhớ?
14. Phép toán lựa chọn trực tiếp (ngẫu nhiên) một phần tử của cấu trúc dữ liệu được biểu diễn tuần tự được thực hiện bằng cách nào?
15. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước cố định?
16. Cho biết công thức xác định số phần tử của một vectơ.
17. Cho biết công thức xác định địa chỉ (vị trí) của phần tử V[i] của véctơ V.
18. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước không cố định?
19. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu đồng nhất?
20. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu không đồng nhất?
21. Để lưu trữ một véctơ trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?
22. Cho biết công thức xác định số phần tử của một ma trận M[LB1..UB1, LB2..UB2] (mảng hai chiều).
23. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự dòng.
24. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự cột.
25. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột). Hãy vẽ mô hình biểu diễn sự lưu trữ này.
26. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòngt (trật tự dòng). Hãy vẽ mô hình biểu diễn sự lưu trữ này.

27. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòng (trật tự dòng), giả sử địa chỉ cơ sở của khối ô nhớ là ∝, kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
28. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột), giả sử địa chỉ cơ sở của khối ô nhớ là ∝, kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
29. Nêu tên các thuộc tính của kiểu mẩu tin.
30. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước cố định?
31. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước không cố định?
32. Có phải mẩu tin là một cấu trúc dữ liệu đồng nhất?
33. Có phải mẩu tin là một cấu trúc dữ liệu không đồng nhất?
34. Để lưu trữ một mẩu tin trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?
35. Việc lựa chọn một phần tử của mẩu tin được thực hiện bởi sự lựa chọn tuần tự hay trực tiếp?
36. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước cố định?
37. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước thay đổi?
38. Nêu tên các phương pháp đặc tả chuỗi ký tự.
39. Nêu tên các phép toán thường có trên kiểu chuỗi ký tự.
40. Cấp phát tĩnh được thực hiện vào lúc nào?
41. Cấp phát động được thực hiện vào lúc nào?
42. Cho biết các ưu nhược điểm của cấp phát động.
43. Sử dụng cấp phát tĩnh, người lập trình có thể chủ động giải phóng ô nhớ không?
44. Sử dụng cấp phát động, người lập trình có thể chủ động giải phóng ô nhớ không?
45. Biến con trỏ được cấp phát động hay cấp phát tĩnh?
46. Có những loại con trỏ nào?
47. Nêu tên các phép toán thường có trên tập hợp.
48. Nêu tên các phương pháp để biểu diễn một tập hợp.
49. Giả sử một tập hợp được biểu diễn bởi một vectơ bit, hãy cho biết giải thuật để thực hiện các phép toán Hợp, Giao và Hiệu hai tập hợp.
50. Sử dụng véctơ bit để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?

51. Sử dụng bảng băm để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?
52. Giả sử một không gian có 5 phần tử e1, e2, e3, e4, e5> Tập hợp { e2, e1, e5, e4} được biểu diễn bởi vector bit nào?
53. Giả sử có ba tập hợp A, B, C được biiểu diễn bởi ba vector bit tương ứng là (1, 0, 1, 1, 1); (1, 0, 1, 0, 1) và (1, 1, 1, 0, 1). Cho biết biểu thức liên hệ giữa các tập A,B và C?
54. Kể tên các phép toán thường có trên tập tin tuần tự.
55. Trong tập tin tuần tự, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
56. Trong tập tin truy xuất trực tiếp, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
57. Trong tập tin truy xuất trực tiếp, chúng ta có thể truy xuất các phần tử một cách tuần tự từ đầu đến cuối tập tin hay không?

5.3.2 Trừu tượng hóa quá trình

Trừu tượng hóa quá trình là việc phân chia chương trình thành những chương trình con. Mỗi chương trình con đảm nhiệm một tác vụ nào đó và được đặc trưng bởi một cái tên.

Ở cấp độ chương trình chính chúng ta chỉ gọi thực hiện các chương trình con, thông qua các tên chương trình con, để thực hiện các tác vụ mà chương trình con đó đảm trách. Như vậy, ở chương trình chính, chúng ta chỉ quan tâm đến kết quả của chương trình con mang lại mà không cần biết chi tiết cài đặt bên trong chương trình con đó.

Ví dụ để viết một chương trình quản lý, ta có thể viết theo hai cách, cách thứ nhất không phân chia thành các chương trình con và cách thứ hai có sử dụng chương trình con.

```
Program Quan_ly;
Begin
   {
   ----
   Đoạn chương trình dùng cho việc
   nhập dữ liệu
   ----
   }
   {
   ----
   Đoạn chương trình dùng cho việc xử
   lý dữ liệu
   ----
   }
   {
   ----
   Đoạn chương trình dùng cho việc
   xuất dữ liệu
   ----
   }
end.
```

Chương trình không có chương trình con

```
Program Quan_ly;
Prcedure nhap_du_lieu;
   {
   Chương trình con nhập dữ liệu
   }
Prcedure xu_ly_du_lieu;
   {
   Chương trình con xử lý dữ liệu
   }
Prcedure xu_ly_du_lieu;
   {
   Chương trình con xuất dữ liệu
   }

Begin  {Chương trình chính}
      nhap_du_lieu;
      xu_ly_du_lieu;
      xuat_du_lieu;
end.
```

Chương trình với chương trình con

Đối với phương pháp thứ nhất, ta thấy toàn bộ chương trình được viết trong chương trình chính, điều này làm cho chương trình chính rất rườm rà, khó đọc hiểu, khó kiểm soát, khó sửa lỗi,...

Đối với phương pháp thứ hai, trong chương trình chính ta chỉ thấy tên các chương trình con (nhap_du_lieu, xu_ly_du_lieu, xuat_du_lieu) và thông qua các tên này ta biết rõ chương trình chính làm những việc gì còn bản thân các việc ấy được làm như thế nào thì ta không cần biết.

Ưu điểm của trừu tượng hoá quá trình

Việc phân chia chương trình thành các chương trình con có các ưu điểm nổi bật như sau:

- Ở chương trình chính, cái tổng thể được làm nổi bật, các chi tiết bị che dấu nên chương trình sáng sủa, dễ đọc hiểu.

```
    RealVect = ARRAY[1..10] OF real;
```

Sau đó ta có thể dùng phép khai báo biến:

```
VAR
    A: RealVect;
    B,C:RealVect;
```

Ưu điểm của định nghĩa kiểu:

- Làm cho việc viết chương trinh trở nên ngắn gọn, sáng sủa hơn.
- Khi cần thay đổi cấu trúc dữ liệu, chỉ cần thay đổi một lần ở mức định nghĩa kiểu chứ không cần phải thay đổi nhiều lần ở mức khai báo từng biến riêng biệt.

Chúng ta thấy rằng kiểu do người dùng định nghĩa chính là một kiểu dữ liệu trừu tượng.

5.4.2 Tính tương đương của các kiểu định nghĩa

Kiểm tra kiểu dẫn tới sự so sánh giữa kiểu dữ liệu của đối số thực đã được cho của một phép toán và kiểu dữ liệu của đối số mà phép toán đó cần đến. Nếu kiểu giống nhau thì đối số được chấp nhận và phép toán được tiến hành, nếu kiểu khác nhau, thì một lỗi được xem xét hoặc một sự cưỡng bức chuyển đổi kiểu được dùng để đổi kiểu của đối số thực thành kiểu thích hợp.

Vấn đề ở đây là cần phải xác định hai kiểu như thế nào thì được coi là "giống nhau" hay tương đương. Xét ví dụ sau đây:

```
TYPE        Vect1 = ARRAY[1..10] OF REAL;
            Vect2 = ARRAY[1..10] OF REAL;
VAR   x,z : Vect1;
      y : Vect2;
PROCEDURE Sub(a:Vect1);
    .....
END;   { Sub }
BEGIN   { Chương trình chính }
    x := y;
    Sub(y);
    ......
END.
```

Vấn đề ở đây là các biến x, y và a có cùng kiểu do đó lệnh gán x := y và lời gọi chương trình con Sub(y) là đúng hay chúng có khác kiểu.

Có hai cách giải quyết cho vấn đề này: tương đương tên và tương đương cấu trúc.

1/ Tương đương tên

H ai kiểu dữ liệu được xem là tương đương chỉ khi chúng có tên giống nhau. Như vậy

Function Ten_ham(Danh sách các tham số cùng với kiểu dữ liệu tương ứng): Kiểu kết quả trả về

Ví dụ Function FN(x:real; y:integer) : real

Đặc tả này xác định hàm FN : REAL x INTEGER -> REAL

Nếu chương trình con trả về nhiều hơn một kết quả hoặc không có kết quả trả về trong tên chương trình con thường được gọi là thủ tục (procedure hoặc subroutine). Cú pháp điển hình đặc tả thủ tục được quy định trong ngôn ngữ lập trình Pascal:

Procedure Ten_thu_tuc(Danh sách các tham sốcùng với kiểu dữ liệu tương ứng)

Ví dụ Procedure SUB(X:real; Y:Integer; Var Z:Real; Var U:boolean);

Trong sự đặc tả này, tham số có tên đứng sau VAR biểu thị một kết quả hoặc một tham số có thể bị thay đổi. Cú pháp của sự đặc tả này trong Ada là:

Procedure SUB(X: IN Real; Y: IN Integer; Z: IN OUT Real; U: OUT Boolean)

Thủ tục này khai báo một chương trình con với sự xác định:

SUB : Real x Integer x Real -> Real x Boolean

Các từ IN, OUT và IN OUT phân biệt ba trường hợp sau đây: IN chỉ định một tham số không thể bị thay đổi bởi cương trình con, IN OUT chỉ định một tham số có thể bị thay đổi và OUT chỉ định một kết quả.

Mặc dù chương trình con biểu diễn một hàm toán học nhưng nó cũng có các vấn đề tương tự như đối với các phép toán nguyên thuỷ:

- Chương trình con có thể có các tham số ẩn trong dạng biến không địa phương mà nó tham chiếu.

- Chương trình con có thể có kết quả ẩn (hiệu ứng lề) được trả về thông qua sự thay đổi các biến không địa phương hoặc thông qua việc thay đổi các tham số IN-OUT của nó.

- Chương trình con có thể nhạy cảm với tiền sử (tự sửa đổi), vì vậy kết qủa của nó không chỉ phụ thuộc vào tham số được cho tại lần gọi đó mà còn phụ thuộc vào toàn bộ lịch sử các lần gọi trước đó. Nhạy cảm với tiền sử có thể do dữ liệu địa phương vẫn còn giữ lại giữa các lần gọi của chương trình con hoặc thông qua sự thay đổi mã riêng của nó (ít phổ biến hơn).

6.2.2 Cài đặt chương trình con

Các phép toán nguyên thuỷ được cài đặt bằng cách dùng cấu trúc dữ liệu và các phép toán được cung cấp bởi máy tính ảo bên dưới ngôn ngữ lập trình. Chương trình con biểu diễn một phép toán được xây dựng bởi người lập trình và do đó chương trình con được cài đặt bằng cách dùng cấu trúc dữ liệu và các phép toán được cung cấp bởi chính bản thân ngôn ngữ lập trình đó. Sự cài đặt được xác định bởi thân chương trình con, bao gồm cả việc khai báo dữ liệu cục bộ xác định cấu trúc dữ liệu được dùng cho chương trình con và các lệnh xác định hành động sẽ làm khi chương trình con thực hiện.

Sự khai báo và các lệnh thường được bao gói, người sử dụng chương trình con không thể truy xuất được tới dữ liệu cục bộ và các lệnh bên trong chương trình con. Người sử

```
VAR
      m : ARRAY[1..max] OF REAL;
      n : INTEGER;
BEGIN
     n := MAX;
     x := 2 * x + m[5];
    ......
END;
```

Sự định nghĩa này xác định các thành phần cần thiết cho một kích hoạt chương trình con:

1/ Bộ nhớ đối với các tham số, đối tượng dữ liệu x và y.

2/ Bộ nhớ cho kết quả hàm, đối tượng dữ liệu của kiểu REAL;

3/ Bộ nhớ cho biến cục bộ, mảng m và biến n.

4/ Bộ nhớ cho các hằng trực kiện và các hằng, 20, 2 và 5.

5/ Bộ nhớ cho mã có thể thực hiện phát sinh từ các lệnh trong thân chương trình con.

Định nghĩa chương trình con cho phép các vùng nhớ này được tổ chức và các mã có thể thực hiện được xác định thông qua việc dịch. Kết quả của việc dịch là khuôn mẫu dùng để xây dựng mỗi một kích hoạt riêng tại thời gian chương trình con được gọi trong quá trình thực hiện.

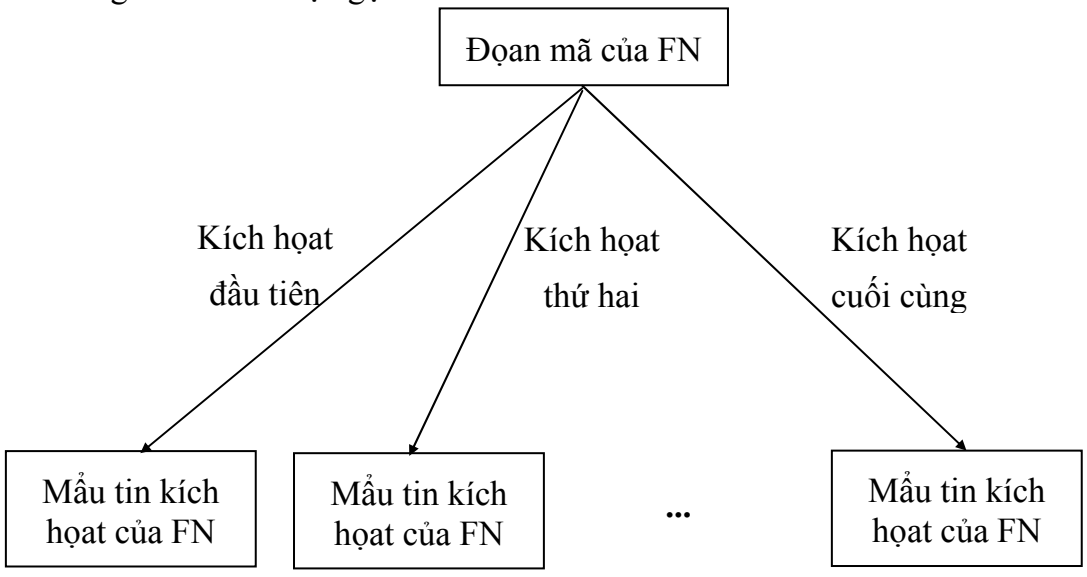
Để xây dựng một kích hoạt chương trình con từ khuôn mẫu của nó, cái khuôn mẫu ban đầu có thể phải được sao chép vào trong vùng mới của bộ nhớ. Tuy nhiên thay vì sao chép hoàn toàn, nó được tách ra thành hai phần:

1/ Phần tĩnh, gọi là đoạn mã (code segment) bao gồm các mục 4 (các hằng) và 5 (các mã có thể thực hiện được) đã nói ở trên. Phần tĩnh không thay đổi trong quá trình thực hiện chương trình con và do thế một bản sao có thể được dùng cho tất cả các kích hoạt.

2/ Phần động, gọi là mẩu tin kích hoạt (Activation record) bao gồm các mục 1 (Các tham số), 2 (Kết quả hàm) và 3 (dữ liệu cục bộ) và cộng thêm nhiều mục khác của dữ liệu ẩn như vùng nhớ tạm, điểm trở về, và sự tham khảo các biến không cục bộ. Phần động có cấu trúc giống nhau cho tất cả các kích hoạt nhưng nó chứa các giá trị dữ liệu khác nhau cho mỗi một kích họat. Do đó mỗi một kích hoạt cần thiết phải có một bản sao mẩu tin kích hoạt riêng của nó. Hình vẽ sau trình bày cấu trúc của một kích hoạt chương trình con của hàm FN nói trên

	Mở đầu	
	Mã lệnh có thể thực hiện	Khối mã có thể thực hiện
	Kết thúc	
Đoạn mã của FN	20 2 5 1	Các hằng
		Điểm trở về
	FN	Kết quả của hàm
	x	Các tham số
Mẩu tin kích hoạt của FN	y	
	m : : : n	Các biến cục bộ

Đối với mỗi chương trình con, một đoạn mã tồn tại thông qua sự thực hiện chương trình. Các mẩu tin kích hoạt được tạo ra và huỷ bỏ một cách động thông qua thực hiện mỗi lần chương trình con được gọi và mỗi lần nó kết thúc bởi RETURN.



Kích thước và cấu trúc của mẩu tin kích hoạt của chương trình con thông thường được xác định trong quá trình dịch, đó là trình biên dịch (compile) có thể xác định mẩu tin kích hoạt lớn như thế nào và vị trí của mỗi một phần tử trong đó. Để truy xuất đến các phần tử có thể sử dụng công thức tính địa chỉ cơ sở cộng độ dời như đã trình bày đối với mẩu tin bình thường.

6.5.3 Các phương pháp truyền tham số tham số

Nói chung một ngôn ngữ cung cấp nhiều phương pháp truyền tham số mà người lập trình có thể lựa chọn để xác định khai báo tham số hình thức lúc định nghĩa chương trình con và cung cấp các tham số thực tế lúc gọi thực hiện chương trình con. Các phương pháp truyền tham số chủ yếu bao gồm:

Truyền bằng giá trị (transmission by value)

- Tham số hình thức là tham số chỉ vào (IN-only parameters), tức là chỉ nhận giá trị vào cho chương trình con, không có nghĩa vụ trả kết quả về cho chương trình gọi. Tham số hình thức được xem như là một **biến cục bộ** của chương trình con và được cấp phát ô nhớ riêng.

- Tham số thực tế là một **biểu thức** (là một biến, một hằng, một hàm hoặc là một biểu thức thực sự).

- Phương pháp thực hiện: Tại thời điểm gọi, **giá trị** của tham số thực tế được sao chép vào trong ô nhớ của tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên ô nhớ riêng của nó, không ảnh hưởng đến tham số thực tế.

- Khi chương trình con kết thúc, sự thay đổi giá trị của tham số hình thức, **không làm ảnh hưởng** đến giá trị của tham số thực tế.

Truyền tham chiếu (transmission by reference)

- Tham số hình thức là tham số vào ra (IN-OUT parameters), tức là nó có nghĩa vụ nhận giá trị vào cho chương trình con và trả kết quả về cho chương trình gọi. Tham số hình thức là một **con trỏ**.

- Tham số thực tế phải là **một biến**, tức là một ĐTDL có ô nhớ.

- Phương pháp thực hiện: Tại thời điểm gọi, con trỏ của tham số thực tế được sao chép cho tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên **ô nhớ của tham số thực tế**.

- Khi chương trình con kết thúc, mọi thay đổi giá trị của tham số hình thức đều làm giá trị của tham số thực tế **thay đổi** theo.

Truyền bằng giá trị-kết quả (transmission by value-result)

- Tham số hình thức là tham số vào ra (IN-OUT parameters) nhưng là một biến cục bộ của chương trình con và được cấp phát ô nhớ riêng.

- Tham số thực tế phải là **một biến**, tức là một ĐTDL có ô nhớ.

- Phương pháp thực hiện: Tại thời điểm gọi, **giá trị** của tham số thực tế được sao chép vào trong ô nhớ của tham số hình thức. Trong quá trình thực hiện chương trình con, mọi thao tác trên tham số hình thức là sự thao tác trên ô nhớ riêng của nó, không ảnh hưởng đến tham số thực tế.

- Khi chương trình con kết thúc, giá trị cuối cùng của tham số hình thức được sao chép vào ô nhớ của tham số thực tế.

Trong ngôn ngữ cấp cao như FORTRAN, công thức này được viết như một biểu thức

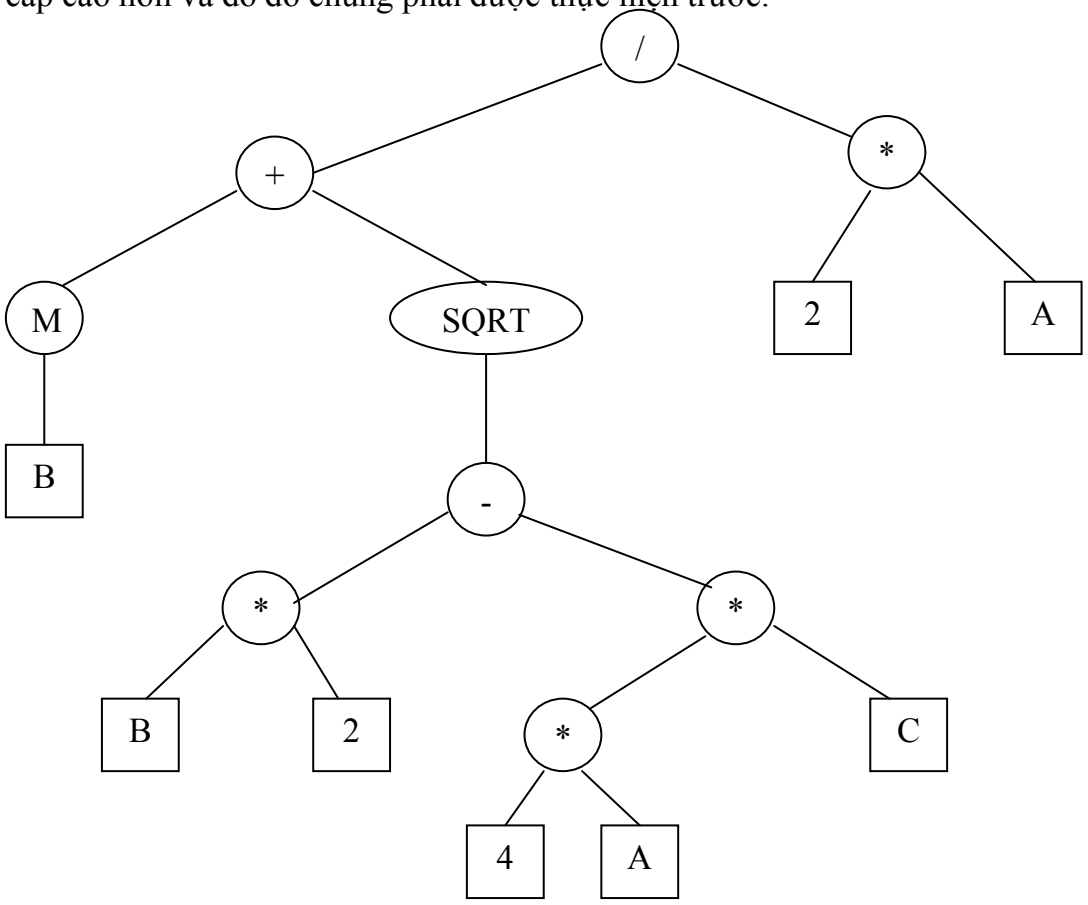
x = (-b + SQRT(b**2 - 4*a*c))/(2*a)

Biểu thức là một phương tiện tự nhiên và mạnh mẽ cho việc biểu diễn dãy các phép toán, tuy vậy chúng nảy sinh các vấn đề mới chẳng hạn như thứ tự thực hiện các toán tử.

7.3.2 Sự biểu diễn theo cấu trúc cây của biểu thức

Cơ chế điều khiển tuần tự cơ bản trong biểu thức là phép lấy hàm hợp: Một phép toán chính và các toán hạng của nó. Trong đó các toán hạng có thể là các hằng, biến hoặc các phép toán khác mà các toán hạng của chúng lại có thể là các hằng, biến hoặc các phép toán khác... Như vậy có thể xem biểu thức là một cấu trúc cây, trong đó nút gốc của cây biểu diễn cho phép toán chính, các nút giữa gốc và lá biểu diễn cho các phép toán trung gian và các nút lá biểu diễn các biến và các hằng. Ví dụ biểu thức nghiệm phương trình bậc hai được biểu diễn theo cấu trúc cây như sau (dùng M để biểu diễn cho phép toán một ngôi lấy số đối):

Sự biểu diễn cây làm sáng sủa cấu trúc điều khiển của biểu thức. Rõ ràng là các kết quả của biến hoặc phép toán ở cấp thấp trong cây được coi như là toán hạng của phép toán ở cấp cao hơn và do đó chúng phải được thực hiện trước.



7.3.3 Cú pháp của biểu thức

Nếu chúng ta xem biểu thức được biểu diễn bởi cây thì để dùng biểu thức trong chương trình, cây phải được tuyến tính hóa chẳng hạn phải có quy định để viết cây như là một dãy tuyến tính các ký hiệu. Chúng ta hãy xem các ký hiệu phổ biến nhất:

Ký hiệu tiền tố (prefix)

Theo ký hiệu Prefix, phép toán viết trước, sau đó là các toán hạng theo thứ tự từ trái sang phải. Nếu một toán hạng lại là một phép toán thì cũng theo quy tắc tương tự. Có ba loại ký hiệu prèix là ordinary, Polish, và Cambridge Polish.

Ký hiệu ordinary prefix sử dụng các dấu ngoặc để bao quanh các toán hạng và dấu phẩy để phân biệt các toán hạng. Ví dụ cấu trúc cây trong hình trên sẽ trở thành:

/(+M(B),SQRT(-(^(B,),*(*(4,A),C)))),*(2,A))

Một biến thể của ký hiệu này được dùng trong ngôn ngữ LISP đôi khi được gọi là Cambridge Polish. Theo ký hiệu Cambridge Polish thì các dấu ngoặc bên trái đứng sau một toán tử được chuyển ra trước toán tử đó và dấu phẩy ngăn cách các toán hạng bị xóa đi. Cấu trúc cây trên trở thành: (/(+(M B)(SQRT(-(^ B 2)(*(* 4 A)C)))) (* 2 A))

Biến thể thứ hai được gọi là ký hiệu Polish, cho phép bỏ hẳn các dấu ngoặc. Nếu chúng ta giả sử rằng số lượng các toán hạng của mỗi một phép toán là đã biết và cố định thì các dấu ngoặc là không cần thiết. Cấu trúc cây trên sẽ trở thành: / + M B SQRT - ^ B 2 * * 4 A C * 2 A

Bởi vì nhà toán học Ba lan Lukasiewiez đã phát minh ra ký hiệu không dấu ngoặc này nên thuật ngữ "Polish" được dùng cho ký hiệu này và các biến thể của nó.

Thực tế hiển nhiên là các biểu thức kiểu này rất khó giải. Trong thực tế, chúng ta không thể giải biểu thức dạng Polish. Các dạng ordinary prefix và Cambridge Polish đòi hỏi quá nhiều dấu ngoặc và dĩ nhiên là các ký hiệu này không gần gũi với những ký hiệu đã trở thành thói quen của chúng ta. Tuy nhiên ký hiệu ordinary prefix là một ký hiệu toán học chuẩn cho hầu hết các phép toán khác các phép toán số học và logic, chẳng hạn f(x,y,z) được viết theo ký hiệu prefix. Điều quan trọng hơn là ký hiệu prefix được dùng để biểu diễn một phép toán với số lượng toán hạng bất kỳ và do đó nói chung chỉ cần học một quy tắc để viết các biểu thức bất kỳ.

Ký hiệu hậu tố (postfix)

Ký hiệu postfix tương tự như ký hiệu Prefix ngoại trừ ký hiệu phép toán đứng sau danh sách các toán hạng. Ví dụ ((A,B)+,(C,A)-)* Hoặc A B + C A - *

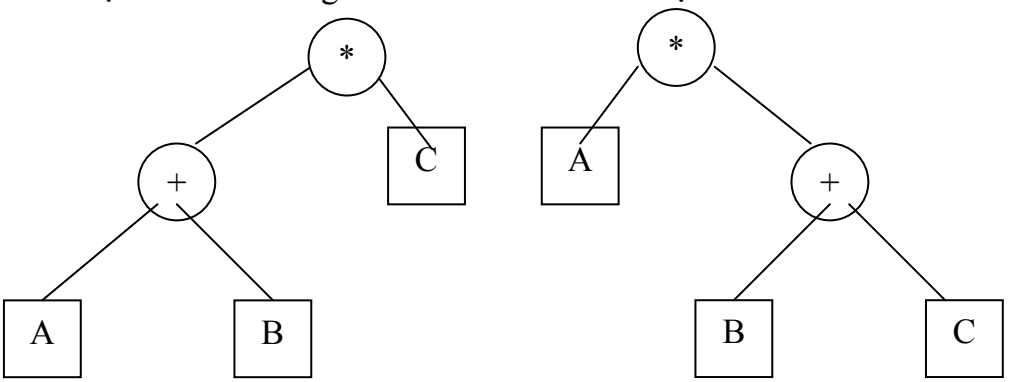
Postfix không phải là sự biểu diễn phổ biến cho biểu thức trong ngôn ngữ lập trình nhưng nó có tầm quan trọng như là cơ sở của sự biểu diễn tại thời gian thực hiện của biểu thức.

Kí hiệu trung tố (infix)

Ký hiệu trung tố thích hợp với phép toán hai ngôi tức là phép toán có hai toán hạng. Trong ký hiệu trung tố, ký hiệu phép toán được viết giữa hai toán hạng. Vì ký hiệu trung tố dùng cho các phép tính số học cơ bản, phép toán quan hệ và các phép toán logic trong toán hoc thông thường nên nó cũng được chọn để dùng một cách rộng rãi trong ngôn ngữ lập trình cho các phép toán đó và trong một số trường hợp còn được mở rộng cho các phép toán khác. Mặc dù ký hiệu trung tố được dùng một cách phổ biến, nhưng việc dùng nó trong ngôn ngữ lập trình cũng gây ra một số vấn đề nhất định:

1/ Vì ký hiệu trung tố chỉ thích hợp đối với phép toán hai ngôi nên một ngôn ngữ không chỉ dùng ký hiệu trung tố mà còn kết hợp với ký hiệu Prexfix hoặc Postfix. Điều này làm cho việc dịch trở nên phức tạp hơn.

2/ Khi có nhiều hơn một toán tử trung tố xuất hiện trong một biểu thức thì có thể xẩy ra tình trạng mập mờ, nghĩa là một biểu thức có thể biểu diễn bằng nhiều cây biểu thức. Ví dụ biểu thức trung tố: A * B + C có thể được biểu diễn thành hai cây như sau:



Dấu ngoặc có thể được dùng để chia các toán tử và toán hạng thành các nhóm, như (A * B) + C hoặc A * (B + C), nhưng trong các biểu thức phức tạp thì các dấu ngoặc lồng nhiều lớp là một trở ngại lớn cho người lập trình. Vì lý do này các ngôn ngữ thường sử dụng quy tắc điều khiển ẩn mà việc dùng dấu ngoặc là không cần thiết. Hai quy tắc ẩn phổ biến là:

a/ Quy tắc ưu tiên trước: Các phép toán xuất hiện trong biểu thức được sắp xếp theo một thứ bậc hoặc một thứ tự ưu tiên trước. Trong một biểu thức có nhiều phép toán, thứ bậc theo quy tắc ẩn là phép toán nào có bậc ưu tiên cao hơn sẽ được thực hiện trước. Ví dụ trong biểu thức A * B + C, phép nhân ưu tiên trước phép cộng nên sẽ được thực hiện trước.

b/ Quy tắc kết hợp: Trong một biểu thức có nhiều phép toán cùng cấp theo thứ tự ưu tiên thì nguyên tắc kết hợp là cần thiết để hoàn thiện việc xác định thứ tự các phép toán. Ví dụ trong biểu thức: A - B - C thì phép toán trừ thứ nhất hay phép trừ thứ hai được thực hiện trước?. Kết hợp trái (thực hiện từ trái qua phải) là nguyên tắc phổ biến nhất cho các phép toán số học, do đó A - B - C được xử lý như (A - B) - C. Tuy nhiên, có một số phép toán lại đòi hỏi sự kết hợp phải, chẳng hạn phép gán trong ngôn ngữ C. Trong ngôn ngữ C ta có thể viết a = b = 10, và thứ tự thực hiện là gán 10 cho b trước, kết quả trả về của phép gán này là 10 sẽ được gán tiếp cho a.

7.3.4 Dịch biểu thức thành biểu diễn cây

Dịch một biểu thức từ sự biểu diễn cú pháp của nó trong văn bản chương trình thành dạng có thể thực hiện là một qúa trình hai giai đoạn. Trước hết biểu thức được dịch thành biểu diễn cây của nó và sau đó cây được dịch thành một dãy các lệnh có thể thực hiện được. Giai đoạn 1 thông thường chỉ liên quan tới sự thành lập cấu trúc điều khiển cây cơ bản của biểu thức, lợi dụng quy tắc ẩn về ưu tiên trước và kết hợp khi biểu thức dùng ký hiệu trung tố. Giai đoạn thứ hai có những quyết định cụ thể liên quan tới thủ tục của sự định giá (evalution) được tạo ra bao gồm cả sự tối ưu hóa quá trình định giá.

Lệnh hợp thành	Lệnh lựa chọn	Lệnh lặp lại
S0 GOTO L1 L2: S2 GOTO L3 L1: S1 GOTO L2 L3 : S3	S0 IF A=0 THEN GOTO L1 S1 GOTO L2 L1: S2 L2: S3	S0 L1: IF A=0 THEN GOTO L2 S1 GOTO L1 L2: S2
Chuỗi lệnh thực hiện S0 S1 S2 S3	Chuỗi lệnh thực hiện S0 S1 S3 Hoặc S0 S2 S3	Chuỗi lệnh thực hiện S0 S2 Hoặc S0 S1 S2 Hoặc S0 S1 S1 S2 Hoặc S0 S1 S1 S1 S2

Lệnh GOTO có thuận tiện là dễ dùng, và có hiệu quả trong thực hiện vì nó phản ánh cấu trúc cơ bản của máy tính quy ước (conventional computers), trong đó mỗi một từ lệnh hoặc byte lệnh đều có địa chỉ, và trong phần cứng có các lệnh nhảy được xây dựng để chuyển điều kkhiển đến địa chỉ được chỉ định. Lệnh GOTO biểu thị một cấu trúc điều khiển tự nhiên để người lập trình chuyển ngôn ngữ cấp cao sang hợp ngữ. Hầu hết các ngôn ngữ cũ đều có cả lệnh GOTO cơ bản và nhiều dạng cải tiến đặt nền móng cho việc dùng nhãn như là dữ liệu. Trong các ngôn ngữ mới như Pascal điều khiển tuần tự trên cơ sở lệnh GOTO ít quan trong hơn mặc dù vẫn còn lệnh đó.

Trong một số ngôn ngữ mới, lệnh GOTO đã bị loại bỏ hoàn toàn. Vì sử dụng nhãn và lệnh GOTO thì chương trình trở nên rất khó đọc vì không có cấu trúc tổng thể và thứ tự các lệnh trong văn bản chương trình nguồn không tương ứng với thứ tự các lệnh khi thực hiện.

7.4.3 Các lệnh cấu trúc

Một lệnh có cấu trúc là một lệnh chứa các lệnh khác. Các lệnh thành phần của một lệnh có cấu trúc có thể là một lệnh cơ bản hoặc một lệnh có cấu trúc. Hầu hết ngôn ngữ cung cấp một tập hợp các lệnh có cấu trúc biểu thị các dạng điều khiển cơ bản (hợp thành, lựa chọn và lặp lại) mà không cần dùng lệnh GOTO.

Lệnh hợp thành (Compound Statements)

Lệnh hợp thành là một chuỗi các lệnh được đặt vào trong một cặp ký hiệu thể hiện sự mở đầu và kết thúc của chuỗi đó. Chẳng hạn trong Pascal, lệnh hợp thành là chuỗi các lệnh được đặt trong cặp từ khóa begin và end như sau:

```
Begin
        Lệnh 1;
```

```
        Lênh 2;
        .....
        Lệnh n
End
```

Cấu trúc lệnh hợp thành cho phép một tập hợp các lênh được trừu tượng hóa thành một lệnh đơn.

Lệnh hợp thành được cài đặt trong máy tính ảo bằng cách thiết lập một khối các mã lệnh có thể thực hiện được biểu diễn cho mỗi một lệnh của chuỗi lệnh trong bộ nhớ. Thứ tự mà chúng xuất hiện trong bộ nhớ xác định thứ tự trong đó chúng được thực hiện.

Lệnh điều kiện (Conditional Statements)

Lệnh điều kiện là một lệnh biểu thị sự lựa chọn của hai hoặc nhiều lệnh. Việc lựa chọn được điều khiển bằng cách kiểm tra một số điều kiện thường được viết trong dạng biểu thức của các phép toán quan hệ và logic. Các lệnh điều kiện phổ biến là lệnh IF và lệnh CASE.

Lệnh IF được cụ thể hóa thành các dạng IF một nhánh, IF hai nhánh và IF đa nhánh.

Chọn thực hiện một lệnh được biểu thị là IF một nhánh: IF <điều kiện> THEN <Lệnh> ENDIF

Chọn một trong hai dùng IF hai nhánh: IF <điều kiện> THEN <Lệnh1> ELSE <Lệnh2> ENDIF

Chọn một trong nhiều dùng các IF nối tiếp nhau hoặc dùng IF đa nhánh:

```
IF <điều kiện1> THEN <Lệnh1>
ELSIF<điều kiện2> THEN <Lệnh2>
   .
   .
   .
ELSIF <điều kiệnN> THEN <LệnhN>
ELSE <LệnhN+1> ENDIF
```

Lệnh CASE

Điều kiện trong lệnh If đa nhánh thường phải lặp lại việc kiểm tra giá trị của một biến, ví dụ:

```
IF TAG = 0 THEN
    <Lệnh 0>
ELSIF TAG = 1 THEN
    <Lệnh 1>
ELSIF TAG = 2 THEN
    <Lệnh 2>
ELSE
    <Lệnh 3>
ENDIF
```

Cấu trúc phổ biến này được biểu diễn một cách súc tích hơn bằng lệnh CASE

```
CASE TAG OF
    0: <Lệnh 0>
```

```
        ......
        average = sum/total;
        ...
        return ;
        when zero_divide {
                average = 0;
                printf(" error: cannot compute average, total is zero\n");
        }
        ......
} /** function example **/
```

7.5.3 Đề xuất một ngoại lệ

Một ngoại lệ có thể bị đề xuất bằng phép toán nguyên thuỷ được định nghĩa bởi ngôn ngữ chẳng hạn phép cộng, phép nhân có thể đề xuất ngoại lệ OVERFLOW. Ngoài ra, một ngoại lệ có thể bị đề xuất một cách tường minh bởi người lập trình bằng cách dùng một lệnh được cung cấp cho mục đích đó. Chẳng hạn trong Ada: raise BAD_DATA_VALUE;

Lệnh này có thể được thực hiện trong một chương trình con sau khi xác định một biến riêng hoặc tập tin nhập chứa giá trị không đúng.

7.5.4 Lan truyền một ngoại lệ (Propagating an exception)

Thông thường, khi xây dựng chương trình thì vị trí mà một ngoại lệ xuất hiện không phải là vị trí tốt nhất để xử lý nó. Khi một ngoại lệ được xử lý trong một chương trình con khác chứ không phải trong chương trình con mà nó được đề xuất thì ngoại lệ đó được gọi là được truyền (propagated) từ điểm mà tại đó nó được đề xuất đến điểm mà nó được xử lý.

Quy tắc để xác định việc xử lý một ngoại lệ đặc thù thường được gọi là chuỗi động (dynamic chain) của các kích hoạt chương trình con hướng tới chương trình con mà nó đề xuất ngoại lệ. Khi một ngoại lệ P được đề xuất trong chương trình con C, thì P được xử lý bởi một xử lý được định nghĩa trong C nếu có một cái xử lý như thế. Nếu không có thì C kết thúc. Nếu chương trình con B gọi C thì ngoại lệ được truyền đến B và một lần nữa được đề xuất tại điểm trong B nơi mà B gọi C. Nếu B không cung cấp một xử lý cho P thì B bị kết thúc và ngoại lệ lại được truyền tới chương trình gọi B vân vân... Nếu các chương trình con và bản thân chương trình chính không có xử lý cho P thì toàn bộ chương trình kết thúc và xử lý chuẩn được định nghĩa bởi ngôn ngữ sẽ được gọi tới.

Một hiệu quả quan trọng của quy tắc này đối với việc truyền các ngoại lệ là nó cho phép một chương trình con kế thừa (remain) như là một phép toán trừu tượng được định nghĩa bởi người lập trình ngay cả trong việc xử lý ngoại lệ. Một phép toán nguyên thuỷ có thể bất ngờ ngắt quá trình bình thường của nó và đề xuất một ngoại lệ. Tương tự, thông qua việc thực hiện lệnh RAISE, một chương trình con có thể bất ngờ ngắt quá trình bình thường của nó và đề xuất một ngoại lệ. Đến chương trình gọi thì

Lúc áp dụng, một phần tử cụ thể của miền xác định gọi là đối sẽ thay thế cho tham số trong định nghĩa hàm. Kết quả hàm thu được bằng cách đánh giá biểu thức hàm. Ví dụ lap_phuong(2.0) cho giá trị là 8.0. Trong định nghĩa hàm, x đại diện cho mọi phần tử của miền xác định. Trong lúc áp dụng, nó được cho một giá trị (chẳng hạn 2.0), giá trị của nó không thay đổi sau đó. Điều này trái ngược với biến trong lập trình có thể nhận các giá trị khác nhau trong quá trình thực hiện chương trình.

Trong định nghĩa hàm, ta bắt cặp tên hàm với biểu thức x*x*x. Đôi khi người ta sử dụng hàm không tên, trong trường hợp đó người ta sử dụng biểu thức lambda. Giá trị của biểu thức lambda chính là hàm của nó. Ví dụ λ(x)x*x*x. Tham số trong biểu thức lambda được gọi là biến kết ghép. Khi biểu thức lambda được đánh giá đối với một tham số đã cho, người ta nói rằng biểu thức được áp dụng cho tham số đó.

8.2.3 Dạng hàm

Dạng hàm là sự tổ hợp của các hàm. Dạng hàm phổ biến nhất là hàm hợp. Nếu f được định nghĩa là hàm hợp của g và h, được viết là f ≡ g.h thì việc áp dụng f được định nghĩa là sự áp dụng h sau đó áp dụng g lên kết quả.

Xây dựng (construction) là một dạng hàm mà các tham số của chúng là những hàm. Người ta ký hiệu một xây dựng bằng cách để các hàm tham số vào trong cặp dấu ngoặc vuông. Khi áp dụng vào một đối số thì các hàm tham số sẽ được áp dụng vào đối đó và tập hợp các kết quả vào trong một danh sách. Ví dụ: G(x) ≡ x*x, H(x) ≡ 2*x và I(x) ≡ x/2 thì [G,H,I](4) có kết quả là (16,8,2).

Áp dụng cho tất cả là một dạng hàm mà nó lấy một hàm đơn như là một tham số. Áp dụng cho tất cả được ký hiệu là ∝. Nếu áp dụng vào một danh sách các đối thì áp dụng cho tất cả sẽ áp dụng hàm tham số cho mỗi một giá trị và tập hợp các kết quả vào trong một danh sách. Ví dụ

Cho h(x) ≡ x*x thì ∝(h, (2,3,4)) có kết quả là (4,9,16)

8.2.4 Bản chất của ngôn ngữ lập trình hàm

Mục đich của việc thiết kế ngôn ngữ lập trình hàm là mô phỏng các hàm toán học một cách nhiều nhất có thể được. Trong ngôn ngữ ra lệnh, một biểu thức được đánh giá và kết quả của nó được lưu trữ trong ô nhớ được biểu diễn bởi một biến trong chương trình. Ngược lại, trong ngôn ngữ lập trình hàm không sử dụng biến và do đó không cần lệnh gán. Điều này giải phóng người lập trình khỏi mối quan tâm về ô nhớ của máy tính trong khi thực hiên chương trình. Không có biến cho nên không có cấu trúc lặp (vì cấu trúc lặp được điều khiển bởi biến). Các lệnh lặp lại sẽ được xử lý bằng giải pháp đệ quy. Chương trình là các định nghĩa hàm và các áp dụng hàm. Sự thực hiện là việc đánh giá các áp dụng hàm. Sự thực hiện một hàm luôn cho cùng một kết quả khi ta cho nó cùng một đối số. Điều này gọi là trong suốt tham khảo (referential transparancy). Nó cho thấy rằng ngữ nghĩa của ngôn ngữ lập trình hàm đơn giản hơn ngữ nghĩa của ngôn ngữ lập trình ra lệnh và ngôn ngữ hàm bao gồm cả những nét đặc biệt của ngôn ngữ ra lệnh.

Ngôn ngữ hàm cung cấp một tập hợp các hàm nguyên thủy, một tập các dạng hàm để xây dựng các hàm phức tạp từ các hàm đã có. Ngôn ngữ cũng cung cấp một phép toán áp dụng hàm và các cấu trúc lưu trữ dữ liệu. Một ngôn ngữ hàm được thiết kế tốt là

Biểu thức

Biểu thức là một nguyên tử hoặc một danh sách. Biểu thức luôn có một giá trị mà việc định trị nó theo nguyên tắc sau:

- Nếu biểu thức là một số, thì giá trị của biểu thức là giá trị của số đó.

  Ví dụ:

  > 25

  = 25

- Nếu biểu thức là một ký hiệu thì giá trị của biểu thức có thể là
  - Được xác định trước bởi LISP (chẳng hạn t có giá trị là T (TRUE) và nil có giá trị là NIL một danh sách rỗng) hoặc
  - Một giá trị dữ liệu của người sử dụng hoặc trong chương trình được gán cho một biến. Biến không cần phải khai báo.

  Ví du:

  > (setq a 3) ; Gán số 3 cho biến có tên a

  = 3

  > a ; hỏi giá trị của ký hiệu "a"

  = 3

- Nếu biểu thức là một danh sách có dạng (E_0 E_1 ... E_n) thì giá trị của biểu thức được xác định theo cách sau đây:
  - Phần tử đầu tiên E_0 phải là một hàm đã được LISP nhận biết.
  - Các phần tử E_1, E_2, ..., E_n được định trị tuần tự từ trái sang phải. Giả sử ta có các giá trị tương ứng là V_1, V_2, ..., V_n
  - Hàm E_0 được áp dụng cho các đối V_1, V_2, ..., V_n. Giá trị của hàm E_0 chính là giá trị của biểu thức.

  Ví dụ

  > (+ 5 3 6)

  = 14

  > (+ 4 (+ 3 5))

  = 12

  - **Chú ý:** Nếu biểu thức dùng hàm QUOTE hoặc dấu nháy đơn sẽ không được đánh giá

  Ví dụ:

  > '(+ 1 2)

  = (+ 1 2)

8.3.3 Các hàm

Một chương trình của LISP là một hàm hoặc một hàm hợp. Các hàm có thể do LISP định nghĩa trước hoặc do lập trình viên tự định nghĩa.

Một số hàm định nghĩa trước

- **Các hàm số học: +, -, *, /, 1+, 1-, MOD, SQRT tác động lên các biểu thức số và cho kết quả là một số.**

Ví dụ:

```
> (+  5  6  2)
= 13
> (-  8  3)
= 5
> (-  8  3  1)
= 4
>(1+  5)  ; Tương đương (+  5  1)
= 6
> (1-  5)   ; Tương đương (-  5  1)
= 4
>(MOD  14  3)
= 2
>(sqrt 9) ; Lấy căn bậc hai của 9
= 3
```

- **Các hàm so sánh các số <, >, <=, >=, = và /=, cho kết quả là T hoặc NIL**

  Ví dụ:

```
>(<  4  5)
= T
>(>  4  (*  2  3))
= NIL
```

- **(EQ s1 s2) so sánh xem hai ký hiệu s1 và s2 có giống nhau hay không?**

  Ví dụ:

```
>(eq  ‘tuong  ‘tuong)
= T
>(eq  ‘tuong  ‘duong)
= NIL
>(eq  ‘5  5  )
```

= T

- **(EQUAL o1 o2) so sánh xem đối tượng bất kỳ o1 và o2 có giống nhau hay không?**

Ví dụ:

>(equal ‘(a b c) ‘(a b c))

= T

>(equal ‘(a b c) ‘(b a c))

= NIL

>(equal ‘a ‘a)

= T

- **Các hàm thao tác trên danh sách: CAR, CDR, CONS và LIST**
  - (CAR L) nhận vào danh sách L, trả về phần tử đầu tiên của L.

Ví du:

> (CAR '(1 2 3))

= 1

> (CAR 3)

Error: bad argument type - 3

>(CAR nil)

= NIL

> (CAR '((a b) 1 2 3))

= (A B)

  - (CDR L) nhận vào danh sách L, trả về một danh sách bằng phần còn lại của danh sách L sau khi bỏ đi phần tử đầu tiên.

Ví dụ:

>(cdr '(1 2 3))

= (2 3)

>(cdr 3)

Error: bad argument type - 3

>(cdr nil)

= NIL

>(cdr '(1))

= NIL

>(CAR (CDR ‘(a b c)))

= B

- **Viết gộp các hàm: T**a có thể dùng hàm C..A/D..R để kết hợp nhiều CAR và CDR (có thể thay thế việc lồng nhau tới 4 cấp)

Ví du:

(CADR '(a b c))

= B

- (CONS x L) nhận vào phần tử x và danh sách L, trả về một danh sách, có được bằng cách thêm phần tử x vào đầu danh sách L

Ví du:

>(CONS 3 '(1 2 3))

= (3 1 2 3)

>(CONS 3 nil)

= (3)

>(CONS '(a b) '(1 2 3))

= ((A B) 1 2 3)

- (LIST E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2, ..., E_n, trả về danh sách bao gồm n phần tử V_1, V_2, ..., V_n, trong đó V_i là giá trị của biểu thức E_i (i=1..n) .

Ví du:

>(list 1 2)

= (1 2)

>(list 'a 'b)

= (A B)

>(list 'a 'b (+ 2 3 5))

= (A B 10)

- **Các vị từ kiểm tra**
  - (ATOM a) xét xem a có phải là một nguyên tử.
  - (NUMBERP n) xét xem n có phải là một số.
  - (LISTP L) xét xem L có phải là một danh sách.
  - (SYMBOLP S) xét xem S có phải là một ký hiệu.
  - (NULL L) nhận vào 1 danh sách L. Nếu L rỗng thì trả về kết quả là T, ngược lại thì trả về kết quả là NIL.

Ví du:

>(atom 'a)

= T

>(numberp 4)

= T

```
>(symbolp 'a)
= T
>(listp '(1  2))
= T
>(symbolp NIL)
= T
>(listp NIL)
= T
>(null NIL)
= T
>(null ‘(a  b))
= NIL
>(null 10)
= NIL
```

- **Các hàm logic AND, OR và NOT**
  - (AND E_1 E_2... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm AND định trị các biểu thức E_1 E_2... E_n từ trái sang phải. Nếu gặp một biểu thức là NIL thì dừng và trả về kết quả là NIL. Nếu tất cả các biểu thức đều khác NIL thì trả về giá trị của biểu thức E_n.

Ví dụ:

```
>(AND (>  3  2) (=  3  2) (+  3  2))
= NIL
>(AND (>  3  2) (- 3  2) (+ 3  2))
= 5
```

  - (OR E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm OR định giá các biểu thức E_1 E_2... E_n từ trái sang phải. Nếu gặp một biểu thức khác NIL thì dừng và trả về kết quả là giá trị của biểu thức đó. Nếu tất cả các biểu thức đều là NIL thì trả về kết quả là NIL.

Ví du:

```
>(OR (=  3  2) (+  2  1) (list  1  2))
= 3
>(OR  (=  2  1)  (Cdr ‘(a) )  (listp 3 ))
= NIL
```

  - (NOT E) nhận vào biểu thức E. Nếu E khác NIL thì trả về kết quả là NIL, ngược lại thì trả về kết quả là T.

- **Các hàm điều khiển**
  - (IF E_1 E_2 E_3) nhận vào 3 biểu thức E_1, E_2 và E_3. Nếu E_1 khác NIL thì hàm trả về giá trị của E_2 ngược lại trả về giá trị của E_3
  - (IF E_1 E_2) tương đương (IF E_1 E_2 NIL)
  - Nếu E_2 khác NIL thì (IF E_1 E_2 E_3) tương đương (OR (AND E_1 E_2) E_3)
  - (COND ($ĐK_1$ E_1)
    ($ĐK_2$ E_2)
    .....
    ($ĐK_n$ E_n)
    [(T E_{n+1})]
)

    Nếu $ĐK_1$ khác NIL thì trả về kết quả là giá trị của E_1, ngược lại sẽ xét $ĐK_2$. Nếu $ĐK_2$ khác NIL thì trả về kết quả là giá trị của E_2, ngược lại sẽ xét $ĐK_3$...

    .....

    Nếu $ĐK_n$ khác NIL thì trả về kết quả là giá trị của E_n, ngược lại sẽ trả về NIL hoặc trả về kết quả là giá trị của E_{n+1} (trong trường hợp ta sử dụng (T E_{n+1}))
  - (PROGN E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm định trị các biểu thức E_1, E_2,... E_n từ trái sang phải và trả về kết quả là giá trị của biểu thức E_n.
  - (PROG1 E_1 E_2 ... E_n) nhận vào n biểu thức E_1, E_2,... E_n. Hàm định trị các biểu thức E_1, E_2,... E_n từ trái sang phải và trả về kết quả là giá trị của biểu thức E1.

Hàm do người lập trình định nghĩa

Cú pháp định nghĩa hàm là:

```
(defun <tên hàm> <danh sách các tham số hình thức>
        <biểu thức>
)
```

Ví dụ 1: Định nghĩa hàm lấy bình phương của số a

```
(defun binh_phuong (a)
    (* a  a)
)
```

Sau khi nạp hàm này cho LISP, ta có thể sử dụng như các hàm đã được định nghĩa trước.

```
>(binh_phuong 5)
= 25
>(binh_phuong (+ 5 2))
= 49
```

Ví dụ 2: Định nghĩa hàm DIV chia số a cho số b, lấy phần nguyên.

Trước hết ta có: a DIV b = (a – a MOD b)/b

```
(defun DIV (a  b)
        (/  (-  a  (MOD a  b)) b)
)
```

8.3.4 Đệ quy

Một hàm đệ quy là một hàm có lời gọi chính nó trong biểu thức định nghĩa hàm. Mô tả một đệ quy bao gồm:

- Có ít nhất một trường hợp "dừng" để kết thúc việc gọi đệ quy.
- Lời gọi đệ quy phải bao hàm yếu tố dẫn đến các trường hợp "dừng".

Ví dụ 1: Viết hàm tính n giai thừa

Công thức đệ quy tính n giai thừa là $n! = \begin{cases} 1 \text{ neu } n = 0 \\ n * (n-1)! \end{cases}$

Hàm (giai_thua N) viết bằng ngôn ngữ LISP:

```
(defun giai_thua (n)
        (if (=  n  0) 1 ; trường hợp "dừng"
           (*  n  (giai_thua (1-  n)));  n-1 là yếu tố dẫn đến trường hợp dừng
         ) ; If
)
```

Ví dụ 2: Viết hàm DIV chia a cho b lấy phần nguyên, viết bằng đệ quy.

Công thức đệ quy: $a \text{ DIV } b = \begin{cases} 0 \text{ neu } a < b \\ 1 + (a-b) \text{ DIV } b \end{cases}$

Hàm (DIV a b) viết bằng LISP:

```
(defun DIV (a b)
        (if (<  a  b) 0 ; Trường hợp "dừng"
           (1+    (DIV  (-  a  b) b)); a-b là yếu tố dẫn đến trường hợp dừng
         ) ; If
)
```

Ví dụ 3: Viết hàm (phan_tu i L), nhận vào số nguyên dương i và danh sách L. Hàm trả về phần tử thứ i trong danh sách L hoặc thông báo "không tồn tại".

Công thức đệ quy:

$$\text{Phan tu thu i trong DS L} = \begin{cases} \text{"Khong ton tai" neu DS L rong} \\ \text{Phan tu dau tien cua L neu } i = 1 \\ \text{Phan tu thu } (i-1) \text{ trong DS "duoi" cua L} \end{cases}$$

Hàm (phan_tu i L) viết bằng LISP:

```
(defun phan_tu(i  L)
        (cond
            ((Null  L) "Khong ton tai")
            ((=  i  1) (car  L)); trường hợp dừng thứ hai
            (T (phan_tu (1-   i) (cdr   L)))
```

```
            ) ; cond
    )
```

Trong chương trình trên, (null L) là trường hợp "dừng" thứ nhất; (= i 1) là trường hợp "dừng" thứ hai; (cdr L) là yếu tố dẫn đến trường hợp "dừng" thứ nhất và (1- i) yếu tố dẫn đến trường hợp "dừng" thứ hai.

8.3.5 Các hàm nhập xuất

- **(LOAD <Tên tập tin>)**

Nạp một tập tin vào cho LISP và trả về T nếu việc nạp thành công, ngược lại trả về NIL. Tên tập tin là một chuỗi kí tự có thể bao gồm cả đường dẫn đến nơi lưu trữ tập tin đó. Tên tập tin theo quy tắc của DOS, nghĩa là chỉ có tối đa 8 ký tự trong phần tên và 3 ký tự phần mở rộng và không chứa các ký tự đặc biệt.

Ta có thể sử dụng LOAD để nạp một tập tin chương trình của LISP trước khi gọi thực hiện các hàm đã được định nghĩa trong tập tin đó.

Ví dụ:

```
>(Load "D:\btlisp\bai1.lsp")
```

- **(READ)**

Đọc dữ liệu từ bàn phím cho đến khi gõ phím Enter, trả về kết quả là dữ liệu được nhập từ bàn phím.

- **(PRINT E)**

In ra màn hình giá trị của biểu thức E, xuống dòng và trả về giá trị của E.

- **(PRINC E)**

In ra màn hình giá trị của biểu thức E (không xuống dòng) và trả về giá trị của E.

- **(TERPRI)**

Đưa con trỏ xuống dòng và trả về NIL.

8.3.6 Biến toàn cục và biến cục bộ

Biến toàn cục

Biến toàn cục (global variables) là biến mà phạm vi của nó là tất cả các hàm. Biến toàn cục sẽ tự động giải phóng khi chương trình dịch LISP kết thúc.

- **Hàm (SETQ <tên biến> <biểu thức>)**

Gán trị của <biểu thức> cho <tên biến> và trả về kết quả là giá trị của <biểu thức>.

Ví dụ:

```
>(setq  x (*  2  3))
= 6
> x  ; biến x vẫn còn tồn tại và có giá trị là 6
```

= 6

Biến cục bộ

Biến cục bộ (local variables) là biến mà phạm vi của nó chỉ nằm trong hàm mà nó được tạo ra. Biến cục bộ sẽ tự động giải phóng hàm tạo ra nó kết thúc.

- **(LET ((var1 E1) (var2 E2) ... (vark Ek)) Ek+1 ... En)**

Ta thấy hàm này có 2 phần: phần gán trị cho các biến và phần định trị các biểu thức.

Gán trị của biểu thức E_i cho biến cục bộ var_i tương ứng và thực hiện (PROGN E_{k+1} ... E_n).

Ví dụ:

>(Let ((a 3) (b 5)) (* a b) (+ a b))

= 8

> a ; biến a lúc này đã được giải phóng nên LISP sẽ thông báo lỗi

error: unbound variable - A

Biến cục bộ che biến toàn cục

Trong lập trình hàm, người ta rất hạn chế sử dụng biến, nếu thật sự cần thiết thì nên sử dụng biến cục bộ. Tuy nhiên việc khai báo biến cục bộ trong hàm LET gây khó khăn cho việc viết chương trình hơn là sử dụng biến toàn cục. Để khắc phục tình trạng này, ta sẽ kết hợp cả hai hàm LET và SETQ để sử dụng biến cục bộ che biến toàn cục. Cách làm như sau:

- Trong phần gán trị cho biến của LET ta tạo ra một biến và gán cho nó một giá trị bất kỳ, chẳng hạn số 0.
- Trong phần định trị các biểu thức, ta có thể sử dụng SETQ để gán trị cho biến đã tạo ra ở trên, biến này sẽ là một biến cục bộ chứ không còn là toàn cục nữa.
- Cụ thể chúng ta có thể viết:

```
(LET  ( (var   E1)…..)
            …….
            (SETQ  var  E2)
            ……
)
```

Với cách làm này thì biến var trong hàm SETQ sẽ trở thành biến cục bộ.

Ví dụ: Giả sử ta đã định nghĩa được hàm (ptb2 a b c), giải phương trình bậc hai $ax^2+bx+c = 0$. Bây giờ ta viết hàm (giai_ptb2) cho phép nhập các hệ số a, b, c từ bàn phím và gọi hàm (ptb2 a b c) để thực hiện việc giải phương trình. Có hai phương pháp để viết hàm này.

Phương pháp 1: dùng các biến toàn cục a, b, c

```
(defun giai_ptb2 ()
        (progn
                (print "Chương trình giải phương trình bậc hai")
```

```
                (princ "Nhập hệ số a: ") (setq  a (read))
                (princ "Nhập hệ số b: ") (setq  b (read))
                (princ "Nhập hệ số c: ") (setq  c (read))
                (ptb2  a  b  c)
        )
    )
```

Sau khi thực hiện chương trình này, thì các biến toàn cục a, b và c vẫn còn.

Phương pháp 2: dùng các biến cục bộ d, e, f

```
    (defun giai_ptb2 ()
        (let ((d  0) (e  0) (f  0))
                (print "Chương trình giải phương trình bậc hai")
                (princ "Nhập hệ số a: ") (setq  d (read))
                (princ "Nhập hệ số b: ") (setq  e (read))
                (princ "Nhập hệ số c: ") (setq  f (read))
                (ptb2  d  e  f)
        )
    )
```

Sau khi thực hiện chương trình này, thì các biến cục bộ d, e và f được giải phóng.

8.3.7 Hướng dẫn sử dụng LISP

Sử dụng XLISP

XLISP là một trình thông dịch, chạy dưới hệ điều hành Windows. Chỉ cần chép tập tin thực thi XLISP.EXE có dung lượng 288Kb vào máy tính của bạn là có thể thực hiện được.

Để thực hiện các hàm, chỉ cần gõ trực tiếp hàm đó vào sau dấu chờ lệnh (>) của XLISP. Trong trường hợp không có dấu chờ lệnh, hãy dùng menu Run/Top level hoặc Ctrl-C để làm xuất hiện dấu chờ lệnh.

Việc định nghĩa một hàm cũng có thể gõ trực tiếp vào sau dấu chờ lệnh. Tuy nhiên cách làm này sẽ khó sửa chữa hàm đó và do vậy ta thường định nghĩa các hàm trong một tập tin chương trình, sau đó nạp vào cho XLISP để sử dụng.

Ta có thể lưu trữ lại tình trạng làm việc hiện hành vào trong tập tin .WKS bằng cách dùng menu File/Save workspace và sau đó có thể khôi phục lại bằng cách dùng menu File/Restore workspace.

Soạn thảo tập tin chương trình

Do XLISP không có công cụ để soạn thảo chương trình nên ta có thể sử dụng Notepad để soạn thảo tập tin chương trình.

Trong một tập tin chương trình ta có thể định nghĩa nhiều hàm.

Lưu tập tin chương trình có tên theo quy định của DOS (8.3) với phần mở rộng .LSP và để trong cặp dấu nháy kép.

Nạp hàm tự định nghĩa cho XLISP

Có hai phương pháp để nạp các hàm tự định nghĩa cho XLISP:

- Phương pháp 1: Copy và dán khối
  - Trong Notepad, đánh dấu khối một hàm tự định nghĩa và copy khối (Edit/Copy hoặc Ctrl-C).
  - Trong XLISP, dán khối tại dấu chờ lệnh (Edit/Paste hoặc Ctrl-Ins).
  - Với phương pháp này thì khi viết các hàm, không nên viết một dòng lệnh quá dài.
  - Nếu khối hàm dán vào không có lỗi thì tên hàm sẽ xuất hiện và ta có thể sử dụng được hàm đó.
  - Phương pháp này rất phù hợp với việc kiểm thử từng hàm.
- Phương pháp 2: Mở tập tin chương trình
  - Trong XLISP, sử dụng menu File-Open/Load để mở tập tin chương trình chứa các hàm đã được viết và lưu trữ bởi Notepad. Chúng ta cũng có thể sử dụng hàm (LOAD <tên tập tin>) để mở tập tin chương trình.
  - Nếu việc mở thành công thì có thể gọi thực hiện bất kỳ hàm nào đã có trong tập tin chương trình.
  - Nếu có một hàm viết sai dấu ngoặc thì việc mở tập tin sẽ thất bại và do đó ta không thể dùng bất kỳ hàm nào trong tập tin đó.
  - Phương pháp này thích hợp với việc nạp nhiều hàm đã được kiểm chứng trong một tập tin chương trình để sử dụng.

Một số thông báo lỗi thường gặp

- Unbound function: Hàm không có.
- Bad function: Hàm sai.
- Too many arguments: Thừa tham số.
- Too few arguments: Thiếu tham số.
- Misplaced close paren: Thừa dấu ngoặc đóng/ Thiếu dấu ngoặc mở.
- EOF reached beore expression end: Thừa dấu ngoặc mở/ Thiếu dấu ngoặc đóng.
- Not a number: Đối số của hàm phải là một số.
- Bad argument type: Kiểu của tham số sai.

Ví dụ để suy diễn số nguyên N bất kỳ là một số nguyên tố ta viết:

"N là một số nguyên tố nếu N>0, M là số nguyên tố nào đó, M<N và N không chia hết cho M".

9.3.3 Cấu trúc của một chương trình Prolog

Một chương trình Prolog thường gồm có 3 hoặc 4 đoạn cơ bản: clauses, predicates, domains và goal. Phần goal có thể bỏ đi, nếu ta không thiết kế goal trong chương trình, thì khi thực hiện, hệ thống sẽ yêu cầu ta nhập goal vào.

Phần Domains

Đây là phần định nghĩa kiểu mới dựa vào các kiểu đã biết. Các kiểu được định nghĩa ở đây sẽ được sử dụng cho các đối số trong các vị từ. Nếu các vị từ sử dụng đối số có kiểu cơ bản thì có thể không cần phải định nghĩa lại các kiểu đó. Tuy nhiên để cho chương trình sáng sủa, người ta sẽ định nghĩa lại cả các kiểu cơ bản.

Cú pháp: <**danh sách kiểu mới**> = <**kiểu đã biết**> hoặc <**danh sách kiểu mới**> = <**danh sách kiểu đã biết**>

Trong đó các kiểu mới phân cách nhau bởi dấu phẩy, còn các kiểu đã biết phân cách nhau bởi dấu chấm phẩy.

Ví dụ:

```
Domains
        ten, tac_gia, nha_xb, dia_chi = string
        nam, thang, so_luong = integer
        dien_tich = real
        nam_xb = nxb(thang, nam)
        do_vat = sach(tac_gia, ten, nha_xb, nam_xb);  xe(ten, so_luong); nha(dia_chi,
        dien_tich)
```

Trong ví dụ trên, ta đã định nghĩa các kiểu mới, trong đó các kiểu mới *ten, tac_gia, nha_xb, dia_chi* dựa vào cùng một kiểu đã biết là *string*; các kiểu mới *nam, thang, so_luong* dựa vào cùng một kiểu đã biết là *integer*; kiểu mới *dien_tich* dựa vào kiểu đã biết là *real*; kiểu mới *năm_xb* dựa vào kiểu nxb được xây dựng từ các kiểu đã biết là *thang, nam*; còn kiểu *do_vat* lại dựa vào các kiểu *sach, xe, nha* mà các kiểu này lại dựa vào các kiểu đã biết.

Phần Predicates

Đây là phần bắt buộc phải có. Trong phần này chúng ta cần phải khai báo đầy đủ các vị từ sử dụng trong phần Clauses, ngoại trừ các vị từ mà Turbo Prolog đã xây dựng sẵn.

Cú pháp: <**Tên vị từ**> (<**danh sách các kiểu**>)

Các kiểu là các kiểu cơ bản hoặc là các kiểu đã được định nghĩa trong phần domains và được viết phân cách nhau bơi dấu phẩy.

Ví dụ:

```
Predicates
        so_huu (ten, do_vat)
```

so_nguyen_to(integer)

Trong ví dụ trên ta khai báo hai vị từ. Trong đó vị từ *so_huu (ten, do_vat)* để chỉ một người có tên là *ten* sẽ sở hữu môt *do_vat* nào đó. Còn vị từ *so_nguyen_to(integer)* để xét xem một số *integer* nào đó có phải là số nguyên tố hay không.

Phần Clauses

Đây là phần bắt buộc phải có dùng để mô tả các sự kiện và các luật, sử dụng các vị từ đã khai báo trong phần predicates.

Cú pháp:

<**Tên vị từ**>(<**danh sách các tham số**>) <**kí hiệu**>
<**Tên vị từ 1**>(<**danh sách các tham số 1**>) <**kí hiệu**>
… … …
<**Tên vị từ N**>(<**danh sách các tham số N**>) <**kí hiệu**>

Trong đó: *Tên vị từ* phải là các *tên vị từ* đã được khai báo trong phần predicates. Các *tham số* có thể là các hằng hoặc biến có kiểu tương thích với các kiểu tương ứng đã được khai báo trong các vị từ ở trong phần predicates; các tham số được viết cách nhau bởi dấu phẩy. Các *kí hiệu* bao gồm:

:- (điều kiện nếu).

, (điều kiện và).

; (điều kiện hoặc).

. (kết thúc vị từ)

Ví dụ:

```
Clauses
        so_nguyen_to(2):-   !.
        so_nguyen_to(N):-   N>0,
                            so_nguyen_to(M),
                            M<N,
                            N MOD M <>0.
        so_huu("Nguyen Van A", sach("Do Xuan Loi", "Cau truc DL", "Khoa hoc Ky thuat", nxb(8,1985))).
```

Chú ý: Nếu trong các tham số của một vị từ có biến thì biến này phải xuất hiện ít nhất 2 lần trong vị từ đó hoặc trong các vị từ dùng để suy diễn ra vị từ đó. Nếu chỉ xuất hiện một lần thì bắt buộc phải dùng biến tự do.

Ví dụ: Để diễn tả sự kiện: Tổ hợp chập 0 của N (N bất kỳ) bằng 1, ta không thể viết *Tohop(N,0,1)* vì biến N chỉ xuất hiện đúng một lần trong vị từ này, do đó ta phải viết *Tohop(_,0,1)* .

Phần Goal

Bao gồm các mục tiêu mà ta yêu cầu Turbo Prolog xác định và tìm kết quả. Đây là phần không bắt buộc phải có. Nếu ta viết sẵn trong chương trình thì đó gọi là goal nội; Nếu không, khi chạy chương trình Turbo Prolog sẽ yêu cầu ta nhập goal vào, lúc này gọi là goal ngoại.

Cú pháp phần goal giống như cú pháp phần clauses. Tức là ta đưa vào một hoặc một số các vị từ.

Nếu tất cả các tham số của vị từ là hằng thì kết quả nhận được là Yes (đúng) hoặc No (sai). Nếu trong các tham số của vị từ có biến thì kết quả trả về sẽ là các giá trị của biến.

Ngoài các phần chủ yếu nói trên, ta có thể đưa vào các phần liên quan đến khai báo hằng, các tập tin liên quan hoặc chỉ thị dịch.

Ví dụ:

Constants

Pi = 3.141592653

Một số ví dụ về chương trình prolog

Ví dụ 1: Xét xem một số N có phải là số nguyên tố hay không.

```
domains
        so_nguyen = integer
predicates
        so_nguyen_to(so_nguyen)
Clauses
        so_nguyen_to(2):-   !.
        so_nguyen_to(N):-   N>0,
                            so_nguyen_to(M),
                            M<N,
                            N MOD M <>0.
goal
        so_nguyen_to(13).
```

Ví dụ 2: Giả sử ta có bảng số liệu như sau:

Tên người	giới tính	Đặc điểm	Tiêu chuẩn kết bạn
lan	nữ	đẹp, khoẻ, tốt,	khoẻ, thông minh, đẹp
hồng	nữ	đẹp, thông minh, giàu	khoẻ, thông minh, giàu
thuỷ	nữ	tốt, khoẻ, giàu	đẹp, khoẻ, thông minh
anh	nam	khoẻ, giàu, thông minh	đẹp, thông minh, tốt
bình	nam	đẹp, khoẻ, thông minh	đẹp, khoẻ
hùng	nam	giàu, thông minh, khoẻ	tốt, thông minh, khoẻ

Tiêu chuẩn kết bạn là hai người khác phái, người này hội đủ các tiêu chuẩn của người kia và ngược lại. Hãy viết chương trình để tìm ra các cặp có thể kết bạn với nhau.

```
domains
  ten, g_tinh = symbol
predicates
  gioi_tinh(ten, g_tinh)
  dep(ten)
  tot(ten)
  giau(ten)
  thong_minh(ten)
```

```
  khoe(ten)
  thich(ten,ten)
  ket_ban(ten,ten)
clauses
  gioi_tinh(lan,nu).
  gioi_tinh(hong,nu).
  gioi_tinh(thuy,nu).
  gioi_tinh(anh,nam).
  gioi_tinh(binh,nam).
  gioi_tinh(hung,nam).

  dep(lan).
  dep(hong).
  dep(binh).

  khoe(thuy).
  khoe(lan).
  khoe(binh).
  khoe(anh).
  khoe(hung).

  tot(lan).
  tot(thuy).

  thong_minh(hong).
  thong_minh(anh).
  thong_minh(hung).
  thong_minh(binh).

  giau(hong).
  giau(thuy).
  giau(hung).

  thich(lan,X):-khoe(X), dep(X), thong_minh(X).
  thich(hong,X):-khoe(X), thong_minh(X), giau(X).
  thich(thuy,X):-khoe(X), dep(X), thong_minh(X).
  thich(ann,X):-dep(X), tot(X), thong_minh(X).
  thich(binh,X):-dep(X), khoe(X).
  thich(hung,X):-khoe(X), tot(X), thong_minh(X).

  ket_ban(X,Y):- gioi_tinh(X,M),
                 gioi_tinh(Y,N),
                 M<>N,
                 thich(X,Y),
                 thich(Y,X).
```

9.3.4 Các nguyên tắc của ngôn ngữ Prolog

Việc giải quyết vấn đề trong ngôn ngữ Prolog chủ yếu dựa vào hai nguyên tắc sau: Đồng nhất, quay lui.

Đồng nhất

Một quan hệ có thể đồng nhất với một quan hệ nào đó cùng tên, cùng số lượng tham số, các đại lượng con cũng đồng nhất theo từng cặp.

Một hằng có thể đồng nhất với một hằng.

Một biến có thể đồng nhất với một hằng nào đó và có thể nhận luôn giá trị hằng đó.

Chẳng hạn trong ví dụ 2 nói trên nếu ta sử dụng goal dep(lan) thì có kết quả là Yes. Nếu ta dùng goal dep(X) thì sẽ có 3 kết quả: X=lan, X=hong và X=binh.

Khi ta dùng goal dep(lan) thì dep(lan) sẽ đồng nhất với sự kiện dep(lan) trong phần clauses và do hai vị từ đồng nhất với nhau và hai đối số hằng đồng nhất nhau nên kết quả là Yes.

Khi dùng goal dep(X) thì dep sẽ được đồng nhất với dep và biến X đồng nhất với hằng lan, do đó ta có kết quả X=lan. Tương tự X=hong và X=binh.

Quay lui

Giả sử hệ thống đang chứng minh goal g, trong đó g được mô tả như sau:

$g :- g_1, g_2, \ldots, g_{j-1}, g_j, \ldots, g_n.$

Khi các g_i kiểm chứng từ trái sang phải, đến g_j là sai thì hệ thống sẽ quay lui lại g_{j-1} để tìm lời giải khác.

Chẳng hạn trong ví dụ 2 nói trên, khi ta yêu cầu Goal: thich(lan,X), ta được X=binh.
Vị từ thich(lan,X) sẽ được đồng nhất với thich(lan,X) trong phần clauses, theo đó hệ thống phải chứng minh thich(lan,X):-khoe(X), dep(X), thong_minh(X).

- Trước hết đồng nhất khoe(X) với khoe(thuy) => X=thuy.
- Do dep(thuy) sai nên quay lui đồng nhất khoe(X) với khoe(lan) => X=lan.
- Do dep(lan) đúng nên tiếp tục kiểm tra thong_minh(lan).
- Do thong_minh(lan) sai nên quay lui để đồng nhất khoe(X) với khoe(binh) để có X=binh, sau đó kiểm tra thấy dep(binh) và thong_minh(binh) đều đúng nên X=binh là một nghiệm.

9.3.5 Bộ ký tự, từ khoá

Prolog dùng bộ ký tự sau: các chữ cái và chữ số (A – Z, a – z, 0 – 9); các toán tử (+, -, *, /, <, =, >) và các ký hiệu đặc biệt.

Một số từ khoá:

a. **Trace**: Khi có từ khoá này ở đầu chương trình, thì chương trình được thực hiện từng bước để theo dõi; dùng phím F10 để tiếp tục.

b. **Fail**: Khi ta dùng goal nội, chương trình chỉ cho ta một kết quả (mặc dù có thể còn những kết quả khác), để nhận về tất cả các kết quả khi chạy goal nội, ta dùng toán tử Fail.

c. **!** hay còn gọi là nhát cắt, goal ngoại luôn cho ta mọi kết quả, muốn nhận chỉ một kết quả từ goal ngoại, ta dùng ký hiệu !.

9.3.6 Các kiểu dữ liệu

Trong prolog có kiểu dữ liệu chuẩn và kiểu do người lập trình định nghĩa.

Kiểu dữ liệu chuẩn

Là kiểu dữ liệu do prolog định nghĩa sẵn. Prolog cung cấp các kiểu dữ liệu chuẩn là: char, integer, real, string và symbol.

a. **Char**: Là kiểu ký tự. Hằng ký tự phải nằm giữa hai dấu nháy đơn.

Ví dụ: ‘a’, ‘#’.

b. **Integer:** Là kiểu số nguyên, tập giá trị bao gồm các số nguyên từ -32768 đến 32767.

c. **Real:** Là kiểu số thực, tập giá trị bao gồm các số thực thuộc hai đoạn: đoạn các số âm từ -10307 đến -10-307 và đoạn số dương từ 10-307 đến 10307.

d. **String:** Là kiểu chuỗi ký tự. Hằng chuỗi ký tự phải nằm giữa hai dấu nháy kép.

Ví dụ: “Turbo prolog 2.0”

e. **Symbol:** Là một kiểu sơ cấp, có hình thức giống chuỗi ký tự. Hằng symbol có hai dạng: Dãy các chữ, số và dấu gạch dưới viết liên tiếp, ký tự đầu phải viết thường (chẳng hạn: telephone_number); Dãy các ký tự ở giữa một cặp hai nháy kép (giống như chỗi ký tự)

f. **Một số phép toán của các kiểu**

Phép toán số học

Phép toán	Ý nghĩa	Kiểu của đối số	Kiểu kết quả
+	Cộng hai số	Integer, real	giống kiểu đối số
-	Trừ hai số	Integer, real	giống kiểu đối số
*	Nhân hai số	Integer, real	giống kiểu đối số
/	Chia hai số	Integer, real	giống kiểu đối số
Mod	Phép chia lấy phần dư	Integer	Integer
Div	Phép chia lấy phần nguyên	Integer	Integer

Phép toán quan hệ

Phép toán	Ý nghĩa	Kiểu của đối số	Kết quả
<	Nhỏ hơn	Char, integer, real, string	Yes hoặc No
<=	Nhỏ hơn hay bằng	Char, integer, real, string	Yes hoặc No
=	Bằng	Char, integer, real, string	Yes hoặc No

>	Lớn hơn	Char, integer, real, string	Yes hoặc No
>=	Lớn hơn hay bằng	Char, integer, real, string	Yes hoặc No
<> hay ><	Khác	Char, integer, real, string	Yes hoặc No

Các vị từ như các hàm toán học

Vị từ	Ý nghĩa	Kiểu của đối số	Kiểu kết quả	Ví dụ
Sin(X)	Tính sin của X	real	real	
Tan(X)	Tính tang của X	real	real	
Arctan(X)	Tính arctang của X	real	real	
Exp(X)	Tính e^X	real	real	
Ln(X)	Tính logarit cơ số e của X	real	real	
Log(X)	Tính Logarit cơ số 10 của X	real	real	
SQRT(X)	Tính căn bậc hai của X	real	real	
ROUND(X)	Cho ta số nguyên là số X được làm tròn, dấu là dấu của X	real	integer	round(2.3)=2 round(2.5)=3 round(-2.5)=-2 round(-2.6)=-3
TRUNC(X)	Cho phần nguyên của số X, dấu là dấu của X	real	integer	trunc(2.5)=2 trunc(-2.6)=-2
ABS(X)	Cho ta trị tuyệt đối của X	real	real	
Random(X)	Cho ta số thực X nằm trong khoảng [0, 1)	real	real	
Random(Y, X)	Cho ta số nguyên X nằm trong khoảng [0, Y)	real	integer	

Toán tử NOT(X) : Nếu X là Yes thì NOT(X) là No và ngược lại.

Các kiểu dữ liệu do người lập trình định nghĩa

a. **Kiểu mẩu tin:**

Cú pháp: **<tên kiểu mẩu tin> = tên mẩu tin (danh sách các kiểu phần tử)**

Ví dụ:

```
Domains
        ten, tac_gia, nha_xb, dia_chi = string
        nam, thang, so_luong = integer
        dien_tich = real
        nam_xb = nxb(thang, nam)
        do_vat = sach(tac_gia, ten, nha_xb, nam_xb);  xe(ten, so_luong); nha(dia_chi,
        dien_tich)
predicates
        so_huu(ten,do_vat)
clauses
        so_huu("Nguyen Van A", sach("Do Xuan Loi", "Cau truc DL", "Khoa hoc Ky
thuat", nxb(8,1985))).
```

so_huu("Le thi B", xe("Dream II", 2)).
so_huu("Nguyen Huu C", nha("3/1 Ly Tu Trong, tp Can Tho", 100.5))

b. Kiểu danh sách

Cú pháp: **<tên kiểu danh sách> = <tên kiểu phần tử>***

Ví dụ:

Domains

intlist = integer*

Một danh sách là một dãy các phần tử phân cách nhau bởi dấu phẩy và đặt trong cặp dấu ngoặc vuông.

Ví dụ:

[] % Danh sách rỗng

[1,2,3] % Danh sách gồm ba số nguyên 1, 2 và 3.

Cấu trúc của danh sách bao gồm hai phần: Phần đầu là phần tử đầu tiên của danh sách và phần đuôi là một danh sách của các phần tử còn lại.

Danh sách được viết theo dạng [X|Y] thì X là phần tử đầu và Y là danh sách đuôi. Chẳng hạn trong danh sách [1,2,3] thì đầu là số nguyên 1 và đuôi là danh sách [2,3].

Trong danh sách cũng có thể dùng biến tự do, chẳng hạn ta có thể viết [_|Y] để chỉ một danh sách có đầu là một phần tử nào đó và có đuôi là danh sách Y.

9.3.7 Các hàm xuất nhập chuẩn

Xuất ra màn hình

a. **Write(Arg1, Arg2, … ,Argn)** in ra màn hình giá trị của các đối số.

b. **Writef(Formatstring, Arg1, Arg2, … ,Argn)** in ra màn hình giá trị của các đối số theo định dạng được chỉ định trong Formastring.

Trong đó Formastring là một chuỗi có thể là:

- "%d": In số thập phân bình thường; đối số phải là char hoặc integer.
- "%c": Đối số là một số integer, in ký tự có mã Ascci là đối số đó, chẳng hạn writef("%c",65) được A.
- "%e": In số thực dưới dạng lũy thừa của 10.
- "%x": In số Hexa; đối số phải là char hoặc integer.
- "%s": In một chuỗi hoặc một symbol.

Nhập vào từ bàn phím

a. **Readln(X)**: Nhập một chuỗi ký tự vào biến X.

b. **ReadInt(X)**: Nhập một số nguyên vào biến X.

c. **ReadReal(X)**: Nhập một số thực vào biến X.

d. **ReadChar(X)**: Nhập vào một ký tự vào biến X.

9.3.8 Kỹ thuật đệ quy

Đệ quy là kỹ thuật lập trình được sử dụng trong nhiều ngôn ngữ. Trong Turbo Prolog ta sử dụng đệ quy khi một vị từ được định nghĩa nhờ vào chính vị từ đó.

Như đã nói trong chương lập trình hàm, trong chương trình đệ quy phải có ít nhất một trường hợp dừng và lời gọi đệ quy phải chứa yếu tố dẫn đến trường hợp dừng. Trong Prolog, trường hợp dừng được thể hiện bằng một sự kiện, yếu tố dẫn đến trường hợp dừng thể hiện bằng một biến, liên hệ với biến ban đầu bởi một công thức.

Ví dụ 1: Tính n giai thừa.

```
Predicates
        Facto (integer, integer)
Clauses
        Facto(0,1):- !.
        Facto(N, FactN) :- N > 0, M = N – 1, facto(M, factM), factN = N*factM.
```

Ở ví dụ trên ta đã định nghĩa một vị từ dùng để tính giá trị giai thừa của một số tự nhiên, đối số thứ nhất là số cần tính giai thừa và đối số thứ hai dùng để nhận giá trị trả về.

Trường hợp dừng ở đây được xác đinh bởi sự kiện 0 giai thừa là 1.

Để tính N! ta tính M! với M= N-1. Yếu tố dẫn đến trường hợp dừng là biến M có giá trị bằng N-1.

Ví dụ 2: Xác định một phần tử trong danh sách các symbol

```
domains
        symbol_list = symbol*
predicates
        element1(integer,symbol_list,symbol)
        element  (integer,symbol_list,symbol)
clauses
  % element1 không suy diễn ngược được
        element1(1,[X|_],X).
        element1(N,[_|L],Y):-      M=N-1,
                                   element1(M,L,Y).
  % element có thể suy diễn ngược
        element(1,[X|_],X).
        element(N,[_|L],Y):-       element(M,L,Y),
                                   N=M+1.
```

Sự suy diễn thuận chiều là cho danh sách và vị trí, tìm được phần tử tại vị trí đó, chẳng hạn, nếu ta đưa vào goal element(2,[a,b,c,d],X) ta được X=b.

Sự suy diễn ngược ở đây là cho danh sách và phần tử, tìm được vị trí của phần tử đó, chẳng hạn, nếu ta đưa vào goal element(N,[a,b,c,d], b) ta được N=2.

Ví dụ 3: Sắp xếp một danh sách các số nguyên

```
domains
        list=integer*
predicates
```

```
        insert(integer,list,list)
        sort(list,list)
clauses
        insert(E,[],[E]).
        insert(E,[A|B],[E,A|B]):-    E<=A.
        insert(E,[A|B],[A|C]):-      E>A,insert(E,B,C).

        sort([],[]).
        sort([X|R1],L):-      sort(R1,R),
                              insert(X,R,L).
```

GIÁO TRÌNH

NGÔN NGỮ LẬP TRÌNH C

ThS. Nguyễn Thị Bích Ngân
Dương Thị Mộng Thùy

LỜI MỞ ĐẦU

Ngôn ngữ lập trình C là ngôn ngữ lập trình cơ bản trong hệ thống các ngôn ngữ lập trình. Do đó đây là môn học cơ sở cho sinh viên chuyên chuyên ngành công nghệ thông tin, là kiến thức thiết yếu cho người lập trình. Ở trường Đại học Công nghiệp Thực phẩm TP.HCM, kiến thức ngôn ngữ lập trình cũng đang được giảng dạy cho tất cả sinh viên đại học, cao đẳng, trung cấp chuyên công nghệ thông tin. Ngôn ngữ lập trình còn là một trong những môn thi trong các kỳ thi liên thông từ trung cấp lên cao đẳng hay từ cao đẳng lên đại học.

Trong giáo trình này, chúng tôi sẽ trình bày các khái niệm, qui định và những kỹ thuật lập trình căn bản thể hiện qua ngôn ngữ lập trình C như như sau:

- **Chương 1: Tổng quan**
- **Chương 2: Các kiểu dữ liệu và phép toán**
- **Chương 3: Các lệnh điều khiển**
- **Chương 4: Hàm**
- **Chương 5: Mảng và Con trỏ**
- **Chương 6: Kiểu dữ liệu cấu trúc.**
- **Chương 7: File dữ liệu**

Các vấn đề được trình bày chi tiết với những minh học ví dụ rõ ràng, mỗi ví dụ chương trình có kết quả thực thi kèm theo để minh họa kiểm chứng. Cuối mỗi chương có phần bài tập được sắp xếp từ cơ bản đến nâng cao giúp sinh viên nắm vững từng chương và kiểm tra kiến thức bằng việc giải các bài tập. Chúng tôi mong rằng các sinh viên tự tìm hiểu trước mỗi vấn đề, kết hợp với bài giảng trên lớp của giảng viên và làm bài tập để việc học môn này đạt hiệu quả.

Trong quá trình giảng dạy và biên soạn giáo trình này, chúng tôi đã nhận được nhiều đóng góp quý báu của các đồng nghiệp ở Bộ môn Công nghệ Phần mềm cũng như các đồng nghiệp trong và ngoài Khoa Công nghệ Thông tin. Chúng tôi xin cảm ơn và hy vọng rằng giáo trình này sẽ giúp cho việc giảng dạy và học môn ngôn ngữ lập trình của trường chúng ta ngày càng tốt hơn. Chúng tôi hy vọng nhận được nhiều ý kiến đóng góp để giáo trình ngày càng hoàn thiện.

TPHCM, ngày 22 tháng 02 năm 2012

ThS. Nguyễn Thị Bích Ngân

Dương Thị Mộng Thùy

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN

1.1 Giới thiệu về ngôn ngữ lập trình C

C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống cùng với Assembler và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennish Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennish Ritchie và B.W Kernighan đã cho xuất bản quyển "Ngôn ngữ lập trình C" và được phổ biến rộng rãi đến nay.

Lúc ban đầu, C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các câu lệnh lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của câu lệnh lập trình, C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình, các công ty lập trình sử dụng ngôn ngữ lập trình C một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ lập trình C và chuẩn ANSI C ra đời.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất "mềm dẻo", có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ lập trình C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng mới. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp "rắc rối" là "hơi khó hiểu" do sự "mềm dẻo" của C. Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

1.2 Đặc điểm của ngôn ngữ lập trình C

- *Tính cô đọng (compact):* C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.

- *Tính cấu trúc (structured)*: C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp… Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.

- *Tính tương thích (compatible)*: C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.

- *Tính linh động (flexible)*: C là một ngôn ngữ rất uyển chuyển và cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.

- *Biên dịch (compile)*: C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

Ngôn ngữ lập trình C cũng là một công cụ để truy nhập vào bộ nhớ máy tính, truy cập các chức năng bên trong **DOS** và **BIOS**, lập trình điều khiển cho các linh kiện điện tử khác.

1.3 Cấu trúc chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, định nghĩa kiểu dữ liệu mới, khai báo biến ngoài, các hàm tự tạo, hàm main.

Cấu trúc chương trình C:

Các chỉ thị tiền xử lý

Định nghĩa kiểu dữ liệu

Khai báo các biến ngoài

Khai báo các prototype của hàm tự tạo

Hàm main

Định nghĩa các hàm tự tạo

1.3.1 Các chỉ thị tiền xử lý

Bước tiền xử lý giúp diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor). Các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu (symbol) #.

Có hai chỉ thị quan trọng:

- Chỉ thị gộp vào của các tập tin nguồn khác: #include
- Chỉ thị định nghĩa các ký hiệu: #define

Chỉ thị #include được sử dụng để gộp nội dung của các tập tin cần có, đặc biệt là các hàm trong tập tin thư viện chuẩn.

Cú pháp:	**#include <Tên tập tin thư viện>**

Ví dụ 1.1:

```
#include <stdio.h>
```

Chỉ thị #define được sử dụng trong việc định nghĩa các ký hiệu

Cú pháp:	**#define <Tên kí hiệu> <giá trị tương ứng>**

Ví dụ 1.2:

```
#define NB_COUPS_MAX 100

#define SIZE 25
```

1.3.2 Định nghĩa kiểu dữ liệu

Bước định nghĩa kiểu dữ liệu dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gợi nhớ hay đặt một kiểu dữ liệu riêng dựa trên các kiểu dữ liệu đã có. Đây là phần không bắt buộc định nghĩa trong chương trình.

Cú pháp:	**typedef <Tên kiểu cũ> <Tên kiểu mới>**

***Ví dụ 1.3*:**

```
typedef int SoNguyen; // Kiểu SoNguyen là kiểu int
```

1.3.3 Khai báo các biến ngoài

Bước khai báo biến ngoài dùng để khai báo các biến toàn cục được sử dụng trong cả chương trình. Đây là phần không bắt buộc khai báo trong chương trình.

1.3.4 Khai báo các prototype của hàm tự tạo

Khai báo các prototype là khai báo tên hàm, các tham số, kiểu kết quả trả về,… của hàm tự tạo sẽ cài đặt phía sau, phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm. Đây là phần không bắt buộc khai báo trong chương trình.

***Ví dụ 1.4*:**

```
boolean isPrime(int a); // prototype của hàm isPrime
```

1.3.5 Hàm main

Khi chương trình thực thi thì hàm main được gọi trước tiên. Đây là phần bắt buộc khai báo trong chương trình.

Cú pháp:

<Kiểu dữ liệu trả về> main()
{
[//Các khai báo cục bộ trong hàm main]
[//Các câu lệnh dùng để định nghĩa hàm main]
[return <kết quả trả về>;]
}

***Ví dụ 1.5*:**

```
void main()
{
        printf("Hello");
        getch();
}
```

1.3.6 Định nghĩa các hàm tự tạo

Đây là phần không bắt buộc định nghĩa trong chương trình.

Cú pháp:

<Kiểu dữ liệu trả về> function(các tham số)
{
[//Các khai báo cục bộ trong hàm.]
[//Các câu lệnh dùng để định nghĩa hàm]
[return <kết quả trả về>;]
}

***Ví dụ 1.6*:**

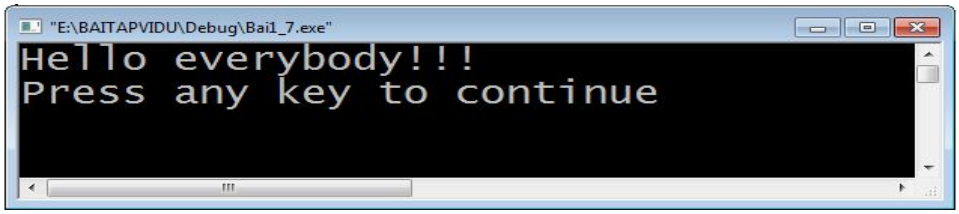
```
int tinhtong(int a, int b)
{
        int t = a+b;
        return t;
}
```

***Ví dụ 1.7:* Chương trình sau sẽ hiển thị ra màn hình dòng chữ : Hello everybody!!!**

#include "stdio.h"

```
void main( )
{
     printf("Hello everybody!!!") ;
}
```

Kết quả thực thi chương trình



Trong đó:

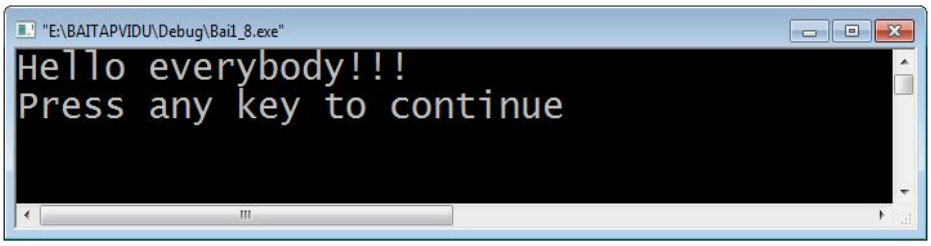
- **main:** hàm chính bắt buộc phải có trong ngôn ngữ lập trình **C**
- **void:** hàm **main** không có giá trị trả về.
- **( ):** chương trình trên không có đối số nào, tức là không có giá trị truyền vào .
- Hai dấu "**{"** và **"}":** qui định thân chương trình, đóng vai trò báo hiệu điểm mở đầu và điểm kết thúc chương trình.

- **printf("Hello everybody!!!");** là lệnh hiển thị dòng chữ "Hello everybody!!!" ra màn hình.

Ví dụ 1.8 : Chương trình hiển thị lên màn hình dòng chữ "Hello everybody!!!" có sử dụng hàm tự tạo.

```
#include "stdio.h"
void Hello();
void main()
{
    Hello();
}
void Hello()
{
    printf("Hello everybody!!!");
}
```

Kết quả thực thi chương trình



Ở ví dụ 1.8 ta thấy cách gọi hàm trong ngôn ngữ lập trình C, hàm main() là hàm chính bắt buộc phải có trong mỗi chương trình. Hàm Hello() được hàm main() gọi đến để thực hiện. Cả ví dụ 1 và ví dụ 2 đều cùng thực hiện việc in ra câu: **Hello everybody!!!**. Nhưng ở đây cho thấy hai cách thể hiện của một câu lệnh trong ngôn ngữ lập trình C.

1.4 Thư viện hàm chuẩn C

Thư viện hàm chuẩn C là tập hợp các hàm đã được xây dựng trước. Mỗi thư viện hàm chứa các hàm theo một công dụng riêng. Tất cả trình biên dịch C đều chứa một thư viện *hàm chuẩn.* Một hàm được viết bởi lập trình viên có thể được đặt trong thư viện và được dùng khi cần thiết. Một số trình biên dịch cho phép thêm hàm vào thư viện chuẩn.

Một số thư viện chuẩn trong C:

- **stdio.h**: Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output). Gồm các hàm in dữ liệu (printf()), nhập giá trị cho biến (scanf()), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhận một dãy ký tự từ bàm phím (gets()), in chuỗi ký tự ra màn hình (puts()), xóa vùng đệm bàn phím (fflush()), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…

- **conio.h** : Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…

- **math.h**: Tập tin định nghĩa các hàm tính toán gồm các hàm abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…

- **alloc.h**: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

- **io.h**: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatênew(), eof(), filelength(), lock(),…

- **graphics.h:** Tập tin định nghĩa các hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …

1.5 Ưu và nhược điểm

1.5.1 Ưu điểm

Ngôn ngữ lập trình C là một ngôn ngữ mạnh, mềm dẻo và có thể truy nhập vào hệ thống, nên thường được sử dụng để viết hệ điều hành, các trình điều khiển thiết bị, đồ họa, có thể xây dựng các phân mềm ngôn ngữ khác , …

Ngôn ngữ lập trình C có cấu trúc module, từ đó ta có thể phân hoạch hay chia nhỏ chương trình để tăng tính hiệu quả, rõ ràng, dễ kiểm tra trong chương trình.

1.5.2 Nhược điểm

Một số kí hiệu của ngôn ngữ lập trìnhC có nhiều ý nghĩa khác nhau. Ví dụ toán tử * là toán tử nhân, cũng là toán tử thay thế, hoặc dùng khai báo con trỏ. Việc sử dụng đúng ý nghĩa của các toán tử phụ thuộc vào ngữ cảnh sử dụng.

Vì C là một ngôn ngữ mềm dẻo, đó là do việc truy nhập tự do vào dữ liệu, trộn lẫn các dữ liệu, …Từ đó, dẫn đến sự lạm dụng và sự bất ổn của chương trình.

Bài tập chương 1

1. Viết chương trình xuất ra câu thông báo: “Chao ban den voi ngon ngu C”.
2. Viết chương trình xuất ra đoạn thông báo:

   “Chao ban!
   Day la chuong trinh C dau tien.
   Vui long nhan phim Enter de ket thuc.”

3. Viết chương trình nhập vào 1 số nguyên, xuất ra màn hình số nguyên vừa nhập.
4. Viết chương trình nhập vào 2 số nguyên, Tính và xuất kết quả tổng, tích 2 số nguyên vừa nhập.
5. Viết chương trình nhập vào 2 số nguyên a, b. Xuất kết quả khi a chia cho b.

CHƯƠNG 2. KIỂU DỮ LIỆU VÀ PHÉP TOÁN

2.1 Danh hiệu

2.1.1 Kí hiệu

Tập kí tự hợp lệ trong ngôn ngữ C bao gồm:

- 52 chữ cái : A,B,C, … ,Z và a,b,c, … ,z
- 10 chữ số : 0,1,2,3,4,5,6,7,8,9
- Các kí hiệu toán học: +, -, *, /, =, <, >, (,)
- Ký tự gạch nối : _ (chú ý phân biệt với dấu trừ " - ")
- Các kí tự đặt biệt như : ., ; : [ ] {} ? ! \ & \ | # $ " ' @ ^…
- Dấu cách (khoảng trắng) dùng để phân cách giữa các từ.

2.1.2 Tên

Tên là một dãy các ký tự liền nhau bắt đầu bằng chữ cái hoặc ký tự gạch dưới theo sau là chữ cái, dấu gạch dưới, chữ số. Một tên không được chứa các kí tự đặc biệt như dấu cách, dấu chấm câu,…

Ví dụ 2.1:

Những tên hợp lệ: x1, chieudai, hoc_sinh, diem_mon_2, _abc, _x_y_z_2,…

Những tên không hợp lệ: 123, 1_xyz, bien#, ma so sinh vien, …

2.1.3 Từ khóa

Các từ sử dụng để dành riêng trong ngôn ngữ lập trình C gọi là từ khoá (keyword). Mỗi một từ khoá có một ý nghĩa riêng của nó. Các từ khóa không được sử dụng làm các biến, hằng, không được được định nghĩa lại các từ khoá. Bảng liệt kê các từ khoá :

auto	break	case	char	continue	default
do	double	else	extern	float	for

gotoif	int	long	register	return	short
sizeof	static	struct	switch	typedef	
union	unsigned	void	while		
_cs	_ds	_es	_ss		
_AH	_AL	_AX	_BH	_BL	_BX
_CH	_CL	_CX	_DH	_DL	_DX
_BP	_DI	_SI	_SP		

2.1.4 Chú thích

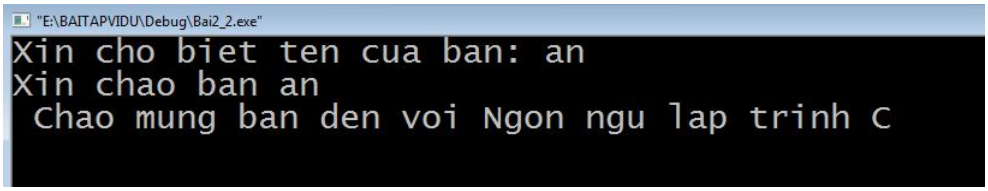
Chú thích là những dòng mô tả diễn tả ý nghĩa câu lệnh đang dùng, giải thích ý nghĩa của một hàm nào đó giúp người lập trình dễ nhớ, dễ hình dung được câu lệnh đang làm. Trình biên dịch không biên dịch các phần ghi chú trong chương trình.

Khi viết chương trình đôi lúc ta cần phải có vài lời ghi chú về một đoạn chương trình nào đó để dễ nhớ và dễ điều chỉnh sau này. Phần nội dung ghi chú này khi biên dịch sẽ được bỏ qua. Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu /* và */.

Ví dụ 2.2:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    /* khai bao bien ten kieu char 50 ky tu */
    char ten[50];
    /*Xuat chuoi ra man hinh*/
    printf("Xin cho biet ten cua ban: ");
    /*Doc vao 1 chuoi la ten cua ban*/
    scanf("%s",ten);
    printf("Xin chao ban %s\n ",ten);
    printf("Chao mung ban den voi NNLT C");
    /*Dung chuong trinh, cho go phim*/
    getch();
    return 0;
 }
```

Kết quả thực thi của chương trình:

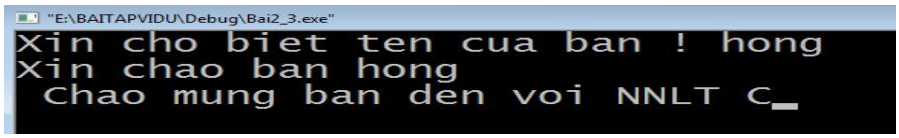


Ngoài ra, nếu chú thích chỉ nằm trên một dòng ta có thể sử dụng kí hiệu //.

Ví dụ 2.3:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
     char ten[50]; //khai báo biến kiểu char 50 ký tự
     //Xuat chuoi ra man hinh
     printf("Xin cho biet ten cua ban !");
     scanf("%s",ten); //Doc vao 1 chuoi ten cua ban.
     printf("Xin chao ban %s\n ",ten);
     printf("Chao mung ban den voi NNLT C");
     //Dung chuong trinh, cho go phim
     getch();
     return 0;
}
```

Kết quả thực thi chương trình:



2.2 Biến

Biến là một khái niệm đại diện cho một giá trị dữ liệu cần lưu trữ tạm thời để tái sử dụng trong các câu lệnh phía sau trong khoảng thời gian chương trình thực thi. Sau khi kết thúc chương trình, biến này sẽ bị hủy. Giá trị này có thể bị thay đổi khi chương trình thực thi. Khi biến được tạo sẽ xuất hiện một vùng nhớ để lưu trữ giá trị của biến.

Một biến có một tên có ý nghĩa đại diện cho một vị trí vùng nhớ. Tên biến giúp chúng ta truy cập vào vùng nhớ mà không cần dùng địa chỉ của vùng nhớ đó.

Hệ điều hành đảm nhiệm việc cấp bộ nhớ còn trống cho những biến này mỗi khi người dùng cần sử dụng. Để tham chiếu đến một giá trị cụ thể trong bộ nhớ, chúng ta chỉ cần dùng tên của biến.

Cú pháp:	**<tên kiểu dữ liệu> <tên biến> [=<giá trị 1>]**

2.3 Các kiểu dữ liệu chuẩn

2.3.1 Kiểu char

Trong ngôn ngữ lập trình C chúng ta có thể xử lý dữ liệu là các chữ viết (kí tự). Các kí tự này là các chữ viết thường dùng như các chữ cái A,B,C….Z, a,b,c,… z; các chữ số 0,1,2,…9; các dấu chấm câu ; , ! …

Kiểu kí tự được biểu diễn trong ngôn ngữ lập trình C với từ khóa **char**. Kiểu char có chiều dài là 1 byte dùng để lưu giữ một kí tự và có miền giá trị trong khoảng –128…127.

Ta khai báo kiểu kí tự như sau:

Cú pháp:	**char <tên biến>**

Ví dụ 2.4: `char ch ; // khai báo ch là kiểu kí tự`

Một hằng kí tự được biểu diễn bằng chữ viết nằm giữa hai dấu phẩy ‘ ’.

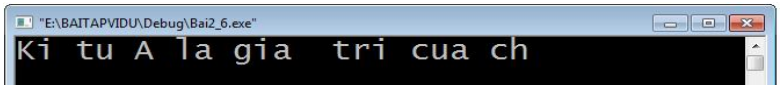
Ví dụ 2.5: `‘a’ , ‘A’ , ‘z ’ , ‘ * ’ , ‘ ! ’ , ’5’…`

Muốn truy xuất giá trị của một biến kí tự ta dùng kí hiệu đại diện % c. Khi đó, giá trị của biến được gọi sẽ hiển thị tại kí tự %c.

Ví dụ 2.6: **Chương trình sau sẽ xuất ra màn hình kí tự của ch, với biến ch được khởi tạo trước.**

```
#include<stdio.h>
void main ()
{
     /* khai báo biến ch có kiểu char */
     char ch ;
     /* khởi tạo cho biến ch có giá trị là ‘A’ */
     ch = ‘A’;
     /* xuất chuỗi kèm theo giá trị biến ch*/
     printf(“Ki tu %c la gia tri cua ch”,ch) ;
}
```

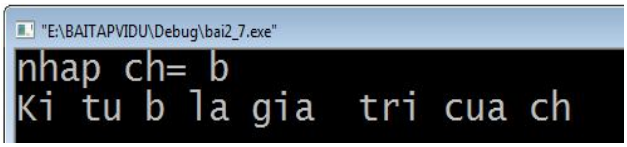
Kết quả thực thi chương trình



Ví dụ 2.7: Chương trình sau nhận một kí tự từ bàn phím và sau đó hiển thị kí tự đó ra màn hình:

```
#include<stdio.h>
void main( )
{
     char ch;
     printf("nhap ch= ");
     scanf("%c",&ch); //đọc kí tự ch từ bàn phím
     printf("Ki tu %c la gia tri cua ch",ch);
}
```

Kết quả thực thi chương trình



Một số kí tự đặc biệt của bảng mã ASCII:

Kí tự	Dãy mã	Giá trị trong bảng mã ASCII		Ý nghĩa
		Hexa-Decimal	Decimal	
BEL	\a	0x07	7	Tiếng chuông
BS	\b	0x08	8	Xóa trái (backspace)
HT	\t	0x09	9	Nhẩy cách ngang (tab)
VT	\v	0x0B	11	Nhẩy cách đứng
LF	\n	0x0A	10	Xuống dòng mới (newline)
FF	\f	0x0C	12	Xuống dòng dưới(form feed)
CR	\r	0x0D	13	Về đầu dòng(carriage return)
“	\”	0x22	34	Dấu “
‘	\’	02x27	39	Dấu ‘
?	\?	0x3F	63	Dấu ?
\	\\	0x5C	92	Dấu \
NULL	\0	0x00	00	Mã NULL

2.3.2 Kiểu int

Trong ngôn ngữ lập trình C, có nhiều loại kiểu số nguyên với các miền giới hạn khác nhau. Kiểu số nguyên cơ bản nhất được định nghĩa với từ khoá int. Tuy nhiên trên máy tính chỉ biểu diễn được một phần nhỏ của tập hợp các số nguyên. Mỗi biến kiểu int chiếm 2 bytes trong bộ nhớ, miền giá trị của kiểu int từ -2^{15} đến $2^{15}-1$.

Khai báo: int tên biến ;

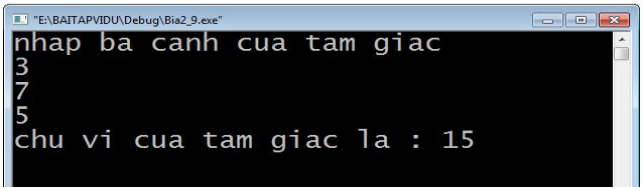
Ví dụ 2.8: `int N; // khai báo biến N là một số kiểu int`

Khi truy xuất giá trị của một biến kiểu số nguyên ta dùng kí hiệu đại diện %d.

Ví dụ 2.9: Chương trình sau nhận giá trị của ba cạnh tam giác, sau đó xuất ra chu vi của tam giác đó:

```
#include<stdio.h>
void main()
{
    int a,b,c ; // ba cạnh của một tam giác
    int  cv ; // chu vi của tam giác
    printf("nhap ba canh cua tam giac ");
    // Nhập vào ba cạnh của tam giác
    scanf("%d%d%d",&a,&b,&c);
    cv = a + b + c ; // tính chu vi của tam giác
    printf("chu vi cua tam giac là : %d", cv);
}
```

Kết quả thực thi chương trình



2.3.3 Kiểu float và double

Một số thực kiểu **float** được biểu diễn bằng **4 bytes**, độ chính xác khoảng 6 chữ số, dãy giá trị trong khoảng 1.2E-38 ÷ 3.4E + 38.

Một số thực kiểu **double** được biểu diễn bằng **8 bytes**, độ chính xác khoảng 15 chữ số, dãy giá trị trong khoảng 2.2E – 308 ÷1.8E + 308.

Một số thực được khai báo như sau:

```
float    x;    //khai báo số thực kiểu float
double  y;   //khai báo số thực kiểu double
```

Ví dụ 2.10 : Xuất ra một số thực có phần thập phân 6 chữ số như sau :

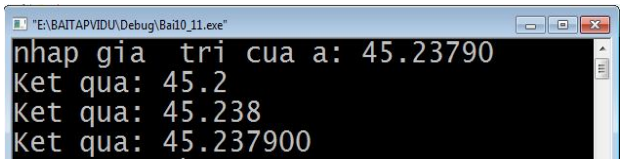
```
// khai báo và khởi tạo biến x = 123.4567
float x = 123.4567;
printf("giá trị của x là %f ", x);
```

Nếu cách hiển thị một số thực là %.nf khi đó giá trị số hiển thị n kí tự cho phần thập phân. Ví dụ như %.5f thì 5 kí tự cho phần thập phân của số hiển thị.

Ví dụ 2.11:

```
#include<stdio.h>
void main()
{
    float a;
    printf("nhap gia tri cua a: ");
    scanf("%f", &a);
    printf("Ket qua: %.1f\n",a);
    printf("Ket qua: %.3f\n",a);
    printf("Ket qua: %.6f\n",a);
}
```

Kết quả thực thi của chương trình:



2.3.4 Các kiểu dữ liệu bổ sung

Các kiểu dữ liệu bổ sung bao gồm **unsigned, signed, short, long**

Các kiểu dữ liệu bổ sung kết hợp với các dữ liệu cơ sở làm thay đổi miền giá trị của chúng.

- **Kết hợp kiểu dữ liệu bổ sung và kiểu dữ liệu chuẩn**

Ta có thể tóm tắt các kiểu chuẩn và kiểu kết hợp qua bảng sau:

Kiểu	Chiều dài	Miền giá trị	Ý nghĩa
unsigned char	8 bits	0…255	Kiểu char không dấu
Char	8 bits	-128…127	Kiểu char
Enum	16 bits	-32768 … 32767	Kiểu enum
unsigned int	16 bits	0 … 65 535	Kiểu số nguyên không dấu
short int	16 bits	-32768 … 32767	Kiểu short int
Int	16 bits	-32768 … 32767	Kiểu số nguyên (int)
unsigned long	32 bits	0 … 4 294 483 647	Kiểu số nguyên (long) không dấu
Long	32 bits	-2 147 483 648 … 2 147 483 648	Kiểu số nguyên (long)
Float	32 bits	$\pm 10^{-38}$…$3.4*10^{38}$	Kiểu số thực (float)
Double	64 bits	$2.2*10^{-308}$ … $1.8*10^{308}$	Kiểu số thực có độ chính xác gấp đôi
Long double	80 bits	$3.4*10^{-4932}$…$3.4*10^{4932}$	Kiểu số thực có độ chính xác cao

- **Mã quy cách định dạng:** sau đây là cách định dạng cho mỗi kiểu dữ liệu khác nhau đối với hàm printf()

Mã quy cách	Ý nghĩa
%**c**	In ra kí tự kiểu char, có thể dùng cho kiểu short và int
%d	Int ra kiểu số nguyên int, có thể dùng cho kiểu char
%u	In ra kiểu số ngyên không dấu, unsigned int, có thể dùng cho kiểu unsigned char, unsigned short
%ld	In ra kiểu long
%lu	In ra kiểu unsigned long
%x , %X	In ra kiểu số viết dưới dạng Hexa ứng với chữ thường và chữ hoa
%o	In ra kiểu số nguyên viết dưới dạng octal (hệ đếm 8)
%f	In ra kiểu số thực dưới dạng bình thường với phần thập phân có 6 chữ số, dùng cho kiểu float, double
%e , %E	In ra kiểu số thực dưới dạng số mũ, với phần định trị có 6 chữ số thập phân, dùng cho kiểu float, double
%g , %G	In ra kiểu %f hoặc %e tùy thuộc kiểu dữ liệu nào ngắn hơn
%s	In ra chuỗi kí tự . Ta phải cung cấp địa chỉ của chuỗi kí tự

- **Các mã quy cách dùng cho hàm scanf():**

Mã quy cách	Ý nghĩa
%**c**	Đọc một kí tự được khai báo là char
%d	Đọc một số nguyên kiểu int
%u	Đọc số nguyên unsigned int
%hd	Đọc số nguyên kiểu short int
%hu	Đọc số nguyên kiểu unsigned int
%ld	Đọc số nguyên kiểu long int
%lu	Đọc số nguyên kiểu unsigned long
%f	Đọc số thực float, có thể nhập theo kiểu viết thông thường hoặc viết theo dạng số mũ
%e	Giống %f
%lf hoặc %lu	Đọc số thực kiểu double
%s	Đọc xâu kí tự không chứa dấu cách, dùng với địa chỉ xâu
%o	Đọc vào số nguyên dưới cơ số 8 (octal)
%x	Đọc vào số nguyên dưới dạng hexa

2.4 Hằng số

Muốn sử dụng hằng, ta cũng phải khai báo trước với từ khóa **const**.

Cú pháp :

const <Tên kiểu dữ liệu> <Tên hằng> = <giá trị hằng số>;

Ví dụ 2.12:

```
const int a= 32767 ;
const float a = 3.14;
const int a= 3, b = 4, C = 5;
```

Chúng ta có thể định nghĩa hằng số theo một kiểu khác với từ khóa define

Cú pháp:

#define <Tên hằng số> <giá trị hằng số>

Ví dụ 2.13:

```
#define PI  3.14
#define E  9.1083e-31
```

Lưu ý: khi định nghĩa hằng bằng define thì không có dấu = và không có dấu chấm phẩy để kết thúc dòng định nghĩa, vì define không phải là một lệnh.

2.5 Biểu thức

Biểu thức là một công thức tính toán để có một giá trị theo đúng quy tắc toán học nào đó. Một biểu thức (expression) bao gồm: toán tử (operator) và toán hạng (operand). Toán tử là phép toán, toán hạng có thể là hằng, là hàm, là biến. Các phần của biểu thức có thể phân thành các số hạng, thừa số, biểu thức đơn giản.

Ví dụ 2.14: `9 + 2 * PI * COS(x)`

Trong ví dụ trên, các toán tử là các phép toán cộng (+), phép nhân (*). Các toán hạng ở đây là các hằng số 9, 2, PI và hàm COS(x).

Các loại biểu thức:

- **Biểu thức số học:** là biểu thức tính ra kết quả là giá trị bằng số (số nguyên, số thực).
- **Biểu thức logic:** là biểu thức mà kết quả là đúng hoặc sai
- **Biểu thức quan hệ:** Một biểu thức chứa các toán tử quan hệ như <, >, <=, >=, ==, != được gọi là biểu thức Boolean đơn giản hay một biểu thức quan hệ. Các toán hạng trong biểu thức quan hệ có thể là các số nguyên, kí tự và chúng không nhất thiết phải tương thích với nhau về kiểu.

2.6 Các phép toán

2.6.1 Toán tử số học

Các toán tử số học thông thường là:

Cộng : +
Trừ : -
Nhân : *
Chia : /
Phép chia lấy phần dư của số nguyên : %

Chú ý:

- Phép toán % không dùng cho kiểu dữ liệu float hay double.

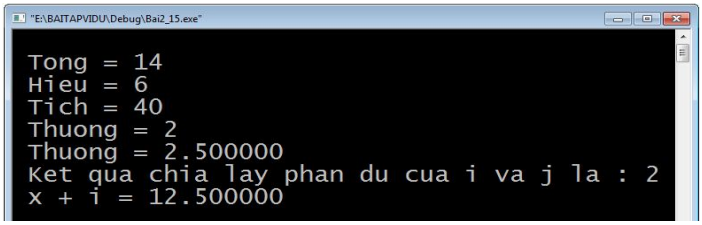
- Phép chia(/) thực hiện theo kiểu của các toán hạng dù là phép chia số nguyên hay số thực.

- Có sự khác nhau giữa i/j và (float)i/j . Theo cách viết (float)i/j thì kết quả sẽ là một số thực.

Ví dụ 2.15: Chương trình sau minh họa cho các phép toán số học:

```
#include<stdio.h>
void main()
{
  int i = 10,j = 4, s, p, r;
  float x;
  s = i + j ;
  printf("\n Tong = %d",s);
  printf("\n Hieu = %d",i-j);
  p = i*j ;
  printf("\nTich = %d",p);
  printf("\nThuong = %d",i/j);
  x=(float)i/j;
  printf("\n Thuong = %f",x);
  r = i % j ;
  printf("\n Phan du la : %d",r);
  printf("\n x + i = %f",x + i );
}
```

Kết quả thực thi của chương trình trên:



2.6.2 Toán tử quan hệ

Các toán tử quan hệ bao gồm :

!= : so sánh sự khác nhau

==: so sánh bằng nhau

>=: so sánh lớn hơn hoặc bằng

<=: so sánh nhỏ hơn hoặc bằng

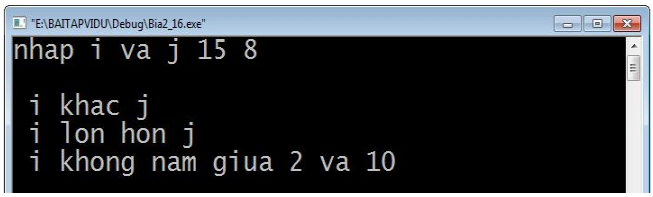
> : so sánh lớn hơn

< : so sánh nhỏ hơn

Ví dụ 2.16: Chương trình sau sẽ minh họa cách sử dụng các toán tử quan hệ:

```
#include<stdio.h>
void main()
{    int i, j;
     printf(“nhap i va j”);
     scanf(“%d%d”,&i,&j);
     if(i == j )  printf(“i bang j”);
     if(i != j )    printf(“i khac j”);
     if(i > j )     printf(“i lon hon j”);
     if(i < j )     printf(“i nho hon j”);
     if( i >= 2 &&i < =10)
          printf(“i nam giua 2 va 10”);
     if(i < 2 | | i > 10)
          printf(“i khong nam giua 2 va 10”);
}
```

Kết quả thực thi chương trình



2.6.3 Toán tử logic

Các phép toán logic gồm:

- && : phép AND logic
- || : phép OR logic
- ! : phép NOT logic

Trong đó:

- Phép && chỉ cho kết quả là đúng chỉ khi hai toán hạng đều đúng.
- Phép || chỉ cho kết quả là sai khi hai toán hạng đều sai.

- Phép ! phủ định lại toán hạng.

2.6.4 Toán tử trên bit

Các toán tử trên bit cho phép xử lý các tín hiệu ở mức bit . Các phép toán này không được dùng cho kiểu float và double.

Các toán tử này bao gồm:

- & : phép AND ở mức nhị phân.
- | : phép OR ở mức nhị phân.
- ^ : phép XOR ở mức nhị phân.
- << : Phép dịch trái(Shift left).
- >> : Phép dịch phải(Shift right).
- ~ : Phép đảo bit.

Cách thực hiện các phép toán trên bit:

1 & 1 = 1	1 | 1 = 1	1 ^ 1 = 0
1 & 0 = 0	1 | 0 = 1	1 ^ 0 = 1
0 & 1 = 0	0 | 1 = 1	0 ^ 1 = 1
0 & 0 = 0	0 | 0 = 0	0 ^ 0 = 0

Các phép dịch trái "<<", dịch phải ">>" gây nên sự lệch trái hay lệch phải nội dụng của một biến.

- x << M nghĩa là dịch sang trái số nguyên x đi M bit, tương đương với $x*2^M$
- x >> M nghĩa là dịch sang phải số nguyên x đi M bit, tương đương với phép chia $x/2^M$ (chia lấy phần nguyên).

Ví dụ 2.17: Ta có thể thay phép tính x*80 bằng cách viết:

x << 6 + x << 2 vì $80 = 2^6 + 2^4$

2.6.5 Toán tử tăng giảm

Trong ngôn ngữ lập trình C, phép tăng giảm 1 có thể viết gọn lại như sau:

- i = i + 1 có thể được viết thành : i ++(tăng sau) hoặc ++ i(tăng trước).
- i = i – 1 có thể được viết thành : i-- (giảm sau) hoặc --i (giảm trước).

Phép ++i thì đầu tiên biến i được tăng 1, sau đó thực hiện phép gán. Còn phép i++ thì phép gán được thực hiện trước, phép tăng 1 sẽ được thực hiện sau.

Ví dụ 2.18: Với i = 3 ; j = 15;

a/ i = ++ j ; i = j kết quả i = 16, j = 16

b/ i = j++ kết quả i = 15, j = 16

c/ j = ++i + 5 kết quả i = 4, j = 9

d/ j = i++ +5 kết quả j = 8, i = 4

2.6.6 Toán tử gán

- **Phép gán đơn giản**

Cú pháp : Tên_một_biến = biểu_thức;

Ví dụ 2.19:

```
i = 3 ; /* I được gán giá trị là 3*/
/* i cộng với 4 được 7, gán 7 vào i*/
i = i + 4 ;
```

Điều này có nghĩa là giá trị của biểu thức bên phải dấu gán = sẽ được đặt vào ô nhớ của biến nằm bên trái dấu gán.

- **Phép gán kép**

Ví dụ 2.20:

```
/* Gán giá trị 5 cho ba biến a, b, c */
a= b = c = 5 ;
/* Gán 5 cho c sau đó c cộng với b, và gán cho a */
a= b + ( c = 5) ;
```

- **Các phép gán mở rộng**

Trong ngôn ngữ lập trình C, phép gán mở rộng được quy định như sau :

x += y ⇔ x = x + y

x -= y ⇔ x = x – y

x *= y ⇔ x = x*y

x /= y ⇔ x = x / y

x %= y ⇔ x = x % y

x >>=y ⇔ x = x >> y

x <<=y ⇔ x = x << y

x &= y ⇔ x = x & y

x |= y ⇔ x = x | y

x ^= y ⇔ x = x ^ y

2.6.7 Toán tử phẩy – biểu thức phẩy

Mỗi câu lệnh trong ngôn ngữ lập trình C được kết thúc bằng dấu chấm phẩy, tuy nhiên trong một biểu thức của ngôn ngữ lập trình C có thể gồm nhiều câu lệnh được cách nhau bởi dấu phẩy.

Ví dụ 2.21:

```
x = a*b, q = x + y, k = q / z;
```

2.6.8 Phép toán biểu thức điều kiện

Cú pháp : <Tên biến> = <Biểu thức điều kiện> ? <biểu thức 1> : <biểu thức 2>

Trong ngôn ngữ lập trình C, toán tử điều kiện (toán tử chấm hỏi " ? ") để so sánh giá trị đúng sai và cho phép có sự chọn lựa thích hợp.

Ví dụ 2.22:

```
m = a > b ? a : b /* m = max(a,b) */
```

Đầu tiên, biểu thức điều kiện a > b được kiểm tra. Nếu biểu thức này có giá trị đúng (True), giá trị của biến a sẽ được gán cho biến m, ngược lại, nếu biểu thức điều kiện a > b là sai (False) thì giá trị biến b sẽ được cho biến m.

Một cách tổng quát, toán tử điều kiện thực hiện các câu lệnh sau : đầu tiên tính biểu thức điều kiện (đứng trước dấu?). Nếu giá trị này khác 0 (tức là TRUE) thì máy sẽ dùng biểu thức thứ nhất, còn nếu bằng 0 (FALSE) thì máy sẽ dùng biểu thức thứ hai.

2.6.9 Độ ưu tiên của toán tử

Ta có thể minh họa độ ưu tiên của toán tử qua một bảng tổng kết sau, với độ ưu tiên được tính từ trên xuống dưới:

Toán tử	Độ ưu tiên
() [] ->	Ưu tiên từ **trái** sang **phải**
- ++ ! ~ sizeof()	Ưu tiên từ **phải** sang **trái**
* / %	Ưu tiên từ **trái** sang **phải**
+ -	Ưu tiên **từ** trái **sang** phải
<< >>	Ưu tiên **từ** trái **sang** phải
< < = > > =	Ưu tiên từ **trái** sang **phải**
== !=	Ưu tiên từ **trái** sang **phải**
&	Ưu tiên từ **trái** sang **phải**
^	Ưu tiên từ **trái** sang **phải**
|	Ưu tiên từ **trái** sang **phải**
&&	Ưu tiên từ **trái** sang **phải**
||	Ưu tiên từ **trái** sang **phải**
? :	Ưu tiên từ **phải** sang **trái**
= += -= *= /= %= ^= | = << = >>	Ưu tiên từ **phải** sang **trái**
,	Ưu tiên từ **trái** sang **phải**

Bài tập chương 2

1. Viết chương trình nhập vào 2 số thực. Tính và xuất kết quả tổng, hiệu, tích, thương của 2 số thực vừa nhập, kết quả lấy 1 số lẻ.
2. Viết chương trình đổi nhiệt đô từ đơn vị Ferarit ra độ C theo công thức:
   C = 5/9 (F-32)
3. Tính tiền khách ở trong tháng.

- Nhập vào ngày đến ở khách sạn, nhập ngày rời khỏi khách sạn.
- Tính tổng số ngày khách đã ở trong tháng.
- Tính tiền khách phải trả, biết rằng đơn giá tuần là 650 và đơn giá ngày là 100.

4. Viết chương trình tính giá trị của biểu thức, trong đó x là số nguyên nhập từ phím.

$F(x) = 5x^2 + 6x + 1$ $G(x) = 2x^4 - 5x^2 + 4x + 1$

5. Viết chương trình tính giá trị của biểu thức, trong đó x là số nguyên nhập từ phím.

$$F(x) = \frac{1+x}{1-x} \qquad G(x) = \frac{1+5x-7x^2}{2+3x^3}$$

6. Viết chương trình tính giá trị của biểu thức, trong đó a,b,c là số nguyên nhập từ phím.

$$F(x) = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad G(x) = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

7. Viết chương trình tính giá trị của biểu thức:

$$f(x) = \frac{3x^2+4x+5}{2x+1} \qquad g(x) = \frac{3x^5+2x+\sqrt{x+1}}{5x^2-3}$$

8. Tình diện tích hình tròn.
9. Nhập vào 2 số nguyên a,b. Tìm số lớn nhất trong 2 số.
10. Nhập vào 5 số nguyên. Tính trung bình cộng 5 số đó.

CHƯƠNG 3. CÁC LỆNH ĐIỀU KHIỂN

3.1 Câu lệnh

3.1.1 Lệnh đơn

Một câu lệnh đơn là một câu lệnh không chứa các câu lệnh khác bên trong nó và kết thúc bằng một dấu chấm phẩy (;)

Ví dụ 3.1:

```
int x=5; //một câu lệnh đơn
x++; //một câu lệnh đơn
printf("Giá trị x là: %d",x); //một câu lệnh đơn
```

3.1.2 Lệnh phức

Một câu lệnh phức là một câu lệnh chứa câu lệnh khác bên trong nó hoặc một khối lệnh gồm nhiều câu lệnh như lệnh điều kiện (lệnh if), lệnh rẽ nhánh (lệnh switch), lệnh lặp (các vòng lặp for, while, do…while).

Một dãy các khai báo cùng với các câu lệnh nằm trong cặp dấu ngoặc móc { và } được gọi là một khối lệnh.

Ví dụ 3.3:

```
{
     char ten[30];
     printf("\n Nhap vao ten cua ban:");
     scanf("%s", ten);
     printf("\n Chao Ban %s",ten);
}
```

Chú ý:

- Nếu một biến được khai báo bên ngoài khối lệnh và không trùng tên với biến bên trong khối lệnh thì nó cũng được sử dụng bên trong khối lệnh.
- Một khối lệnh bên trong có thể sử dụng các biến bên ngoài, các lệnh bên ngoài không thể sử dụng các biến bên trong khối lệnh con.

Ví dụ 3.4:

```
int a=0; /*biến a trong khối lệnh bên ngoài*/
for(int i=0; i<7; i++)
{
     int t = 5;
     /*biến a bên ngoài khối lệnh*/
     a= a+i;
}
printf("gia tri cua a la: %d",a);
printf("gia tri cua t la: %d",t);
/* báo lỗi! không sử dụng được */
```

3.2 Lệnh điều kiện

3.2.1 Lệnh if

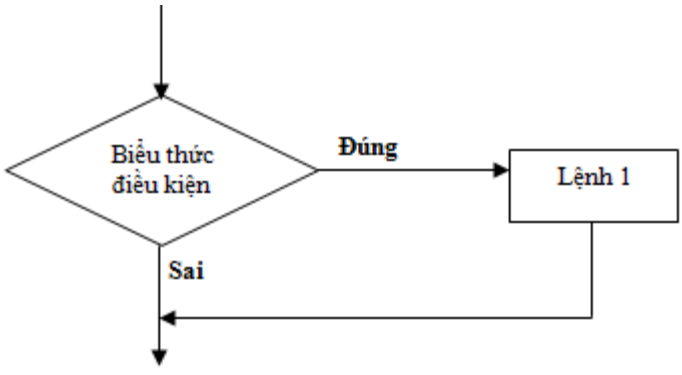
Câu lệnh if cho phép chúng ta thay đổi luồng thực thi của câu lệnh trong chương trình dựa vào điều kiện, một câu lệnh hoặc một khối các câu lệnh sẽ được quyết định thực thi hay không được thực thi.

- **Lệnh if đơn giản:**

Cú pháp:

if (<*Biểu thức điều kiện*>)
<Câu lệnh>

Lưu đồ:



Giải thích:

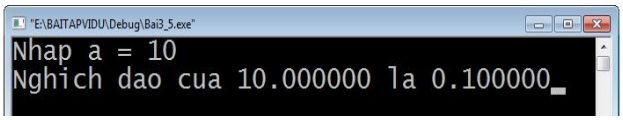
- <Lệnh 1> có thể là một câu lệnh đơn, một khối lệnh hay một câu lệnh phức.
- Kiểm tra Biểu thức điều kiện trước.
- Nếu điều kiện đúng (True) thì thực hiện Lệnh 1 theo sau biểu thức điều kiện.
- Nếu điều kiện sai (False) thì bỏ qua Lệnh 1 (những lệnh và khối lệnh sau đó vẫn được thực hiện bình thường vì nó không phụ thuộc vào điều kiện sau if).

Ví dụ 3.5: Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi a ≠ 0

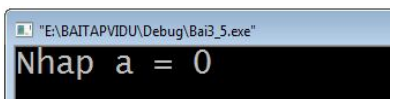
```
#include <stdio.h>
#include <conio.h>
int main ()
{
     float a;
     printf("Nhap a = "); scanf("%f",&a);
     if (a !=0 )
     printf("Nghich dao cua %f la %f",a,1/a);
     getch();
     return 0;
}
```

Nếu nhập vào a ≠0 thì câu lệnh printf("Nghich dao cua %f la %f",a,1/a)được thực hiện, ngược lại câu lệnh này không được thực hiện.

Kết quả thực thi chương trình khi nhập a = 10



Kết quả thực thi chương trình khi a=0

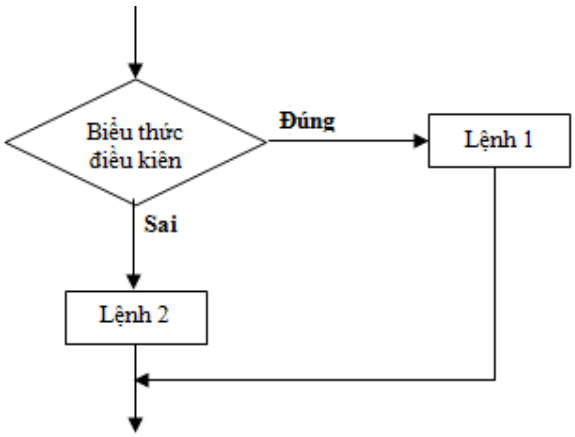


- **Lệnh if – else đơn giản**

Cú pháp:

```
if (<Biểu thức điều kiện>)
        <Lệnh 1>
else
        <Lệnh 2>
```

Lưu đồ :



Giải thích:

+ *Lệnh 1, Lệnh 2* được thể hiện là một câu lệnh đơn, một khối lệnh hay một câu lệnh phức.

+ Đầu tiên *Biểu thức điều kiện* được kiểm tra trước.

+ Nếu điều kiện đúng thì thực hiện Lệnh 1.

+ Nếu điều kiện sai thì thực hiện Lệnh 2.

+ Các lệnh phía sau Lệnh 2 không phụ thuộc vào điều kiện.

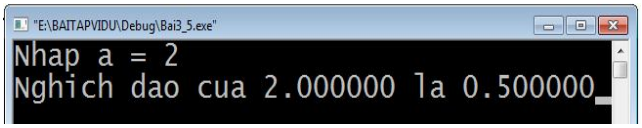
Lệnh if-else đơn giản làm giảm đi độ phức tạp của chương trình, làm cho chương trình dễ hiểu hơn.

Ví dụ 3.6: Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi a ≠0, khi a =0 in ra thông báo "Khong the tim duoc nghich dao cua a"

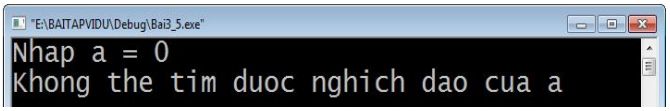
```
#include <stdio.h>
#include <conio.h>
int main ()
{
    float a;
    printf("Nhap a = "); scanf("%f",&a);
    if (a !=0 )
    printf("Nghich dao cua %f la %f",a,1/a);
    else
    printf("Khong the tim duoc nghich dao");
    getch();
    return 0;
}
```

Nếu chúng ta nhập vào a ≠ 0 thì câu lệnh printf("Nghich dao cua %f la %f",a,1/a) được thực hiện, ngược lại câu lệnh printf("Khong the tim duoc nghich dao cua a") được thực hiện.

Kết quả thực thi của chương trình khi nhập a ≠ 0



Kết quả thực thi của chương trình khi nhập a = 0

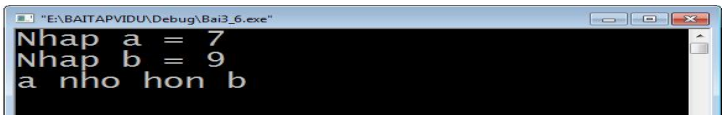


Ví dụ 3.6: Nhập vào 2 số a,b. So sánh số a với số b vừa nhập.

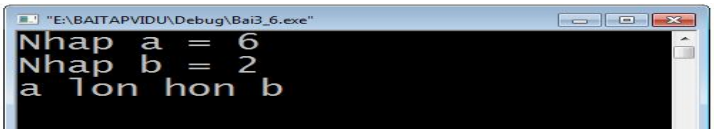
```
#include <stdio.h>
#include <conio.h>
int main ()
{
int a, b;
printf("Nhap a = "); scanf("%d",&a);
printf("Nhap b = "); scanf("%d",&b);
if (a > b )
     printf("a lon hon b");
```

```
        else
             if(a==b)
                  printf("hai so bang nhau");
             else
                  printf("a nho hon b");
        getch();
        return 0;
        }
```

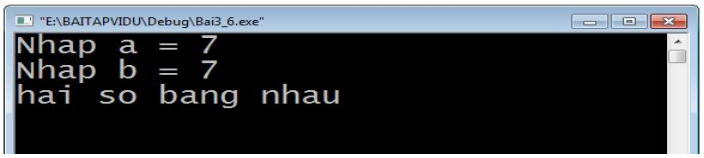
Kết quả thực thi chương trình khi nhập a=7, b=9



Kết quả thực thi chương trình khi nhập a=6, b=2



Kết quả thực thi khi a=7 b=7



3.2.2 Lệnh switch case

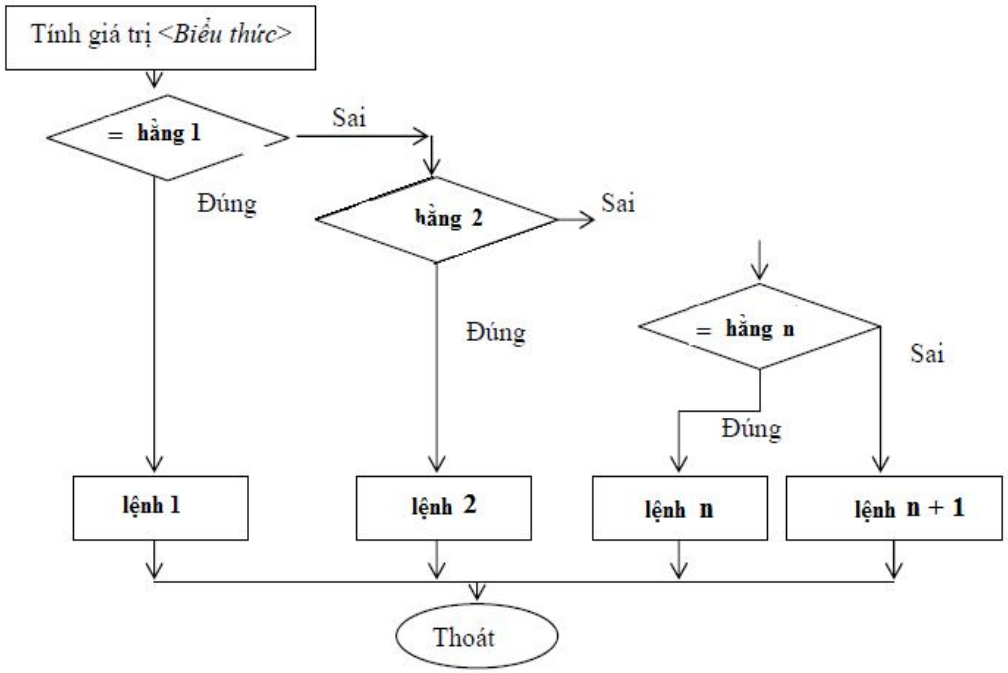
Câu lệnh rẽ nhánh switch-case cho phép lựa chọn một trong các lựa chọn đã đưa ra.

Cú pháp:

```
switch (<biểu thức>)
{
    case <hằng 1> :    lệnh 1 ;
    break;
    case <hằng 2> :    lệnh 2 ;
    break;
    . . . . . .
    case <hằng n> :    lệnh n ;
```

```
        break ;
        default    : lệnh n+1;
    }
```

Lưu đồ :



Giải thích:

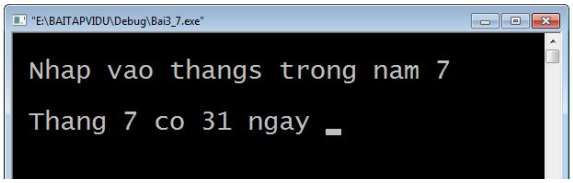
- Tính giá trị của biểu thức trước.
- Nếu giá trị của biểu thức bằng hằng 1 thì thực hiện lệnh 1 rồi thoát.
- Nếu giá trị của biểu thức khác hằng 1 thì so sánh với hằng 2, nếu bằng hằng 2 thì thực hiện lệnh 2 rồi thoát.
- Cứ như thế, so sánh tới hằng n.
- Nếu tất cả các phép so sánh trên đều sai thì thực hiện lệnh n+1 mặc định của trường hợp *default*.

Ví dụ 3.7: Nhập vào giá trị tháng của năm, xuất số ngày trong tháng.

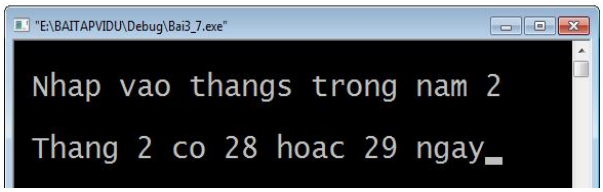
```
#include <stdio.h>
#include<conio.h>
int main ()
```

```
{
   int thang;
   printf("\n Nhap vao thang trong nam ");
   scanf("%d",&thang);
   switch(thang)
   {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
             printf("\n Thang %d co 31 ngay ",thang);
             break;
        case 4:
        case 6:
        case 9:
        case 11:
             printf("\n Thang %d co 30 ngay ",thang);
             break;
        case 2:
             printf ("\n Thang 2 co 28 hoac 29 ngay");
             break;
        default :
             printf("\n Khong co thang %d", thang);
             break;
   }
   getch();
   return 0;
}
```

Kết quả thực thi khi nhập tháng = 7



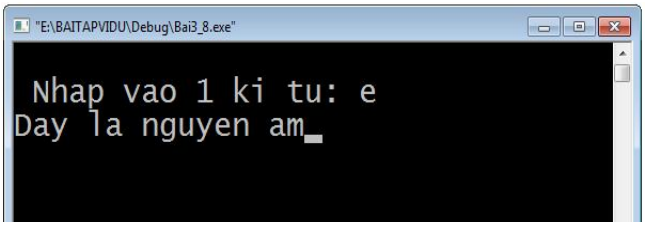
Kết quả thực thi khi nhập tháng = 2



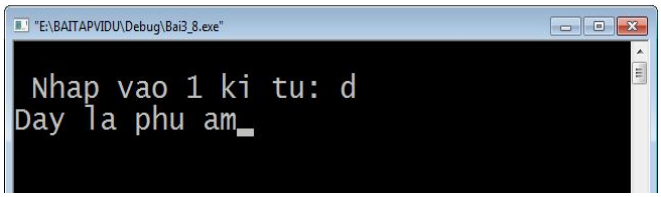
Ví dụ 3.8: Nhập vào 1 kí tự, cho biết kí tự đó là nguyên âm hay phụ âm

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    char ch;
    printf("\n Nhap vao 1 ki tu: ");
    scanf("%c",&ch);
    switch(ch)
    {
         case 'a' :
         case 'o':
         case 'e' :
         case 'u' :
         case 'y' :
         case 'i' : printf("Day la nguyen am") ;
                         break ;
         default :       printf("Day la phu am");
    }
    getch();
    return 1;
}
```

Kết quả thực thi chương trình khi nhập kí tự e:



Kết quả thực thi chương trình khi nhập kí tự d:



Chú ý: Nếu câu lệnh **case N_i** không có câu lệnh **break**, thì máy sẽ tự động thực hiện câu lệnh củaCase N_{i+1}

3.3 Lệnh lặp

Lệnh lặp là một câu lệnh, một đoạn lệnh trong chương trình thực hiện lặp đi lặp lại cho đến khi một điều kiện xác định được thỏa mãn. Có thể nói, một lệnh lặp cho phép lặp lại các câu lệnh nhiều lần.

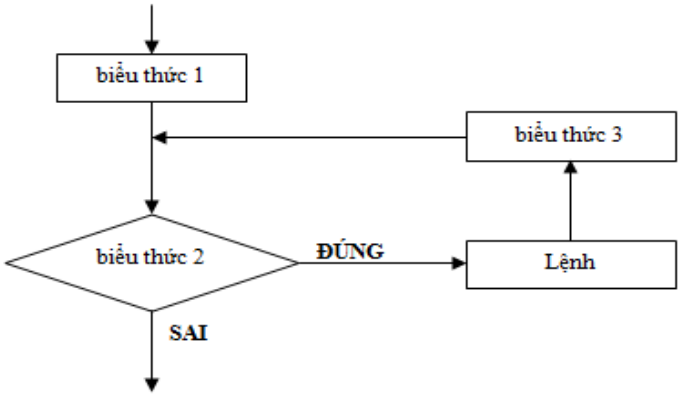
3.3.1 Lệnh for

Lệnh for thực thi việc lặp lại một câu lệnh, một khối lệnh nhiều lần với số lần lặp xác định trước.

Cú pháp :

```
for ( [ <biểu thức 1>]; [ <biểu thức 2>] ; [ <biểu thức 3>])

        <câu lệnh>;
```

Lưu đồ:



Quá trình thực hiện câu lệnh for :

- Bước 1: Xác định giá trị của biểu thức 1

- Bước 2: Xác định giá trị của biểu thức 2
- Bước 3: Nếu biểu thức 2 sai thì sẽ thoát vòng lặp for

  Nếu biểu thức 2 đúng thì máy sẽ thực hiện câu lệnh
- Bước 4: Tính giá trị của biểu thức 3 và quay lại Bước 2

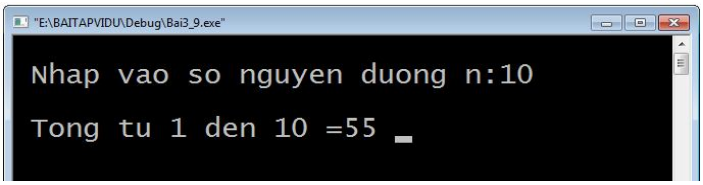
Ta có một số chú ý như sau:

- Biểu thức 1 là biểu thức gán trị khởi động cho biến lặp.
- Biểu thức 2 là biểu thức điều kiện. Nếu biểu thức 2 vắng mặt, điều kiện luôn đúng .
- Biểu thức 3 thông thường là biểu thức thay đổi điều kiện.
- Biểu thức 1, 3 có thể gồm nhiều biểu thức cách nhau bởi dấu phẩy.
- Biểu thức thứ 2 có thể bao gồm nhiều biểu thức, nhưng tính đúng sai của nó được xem là tính đúng sai của biểu thức cuối cùng.

Ví dụ 3.9: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

```
#include <stdio.h>
#include<conio.h>
int main ()
{
     int n,i,tong;
     printf("\n Nhap vao so nguyen duong n:");
     scanf("%d",&n);
     tong=0;
     for (i=1; i<=n; i++)
          tong+=i;
     printf("\n Tong tu 1 den %d =%d ",n,tong);
     getch();
     return 0;
}
```

Kết quả thực thi chương trình:



Đối với câu lệnh for, ta có thể dùng câu lệnh break để thoát khỏi vòng lặp for tại một trường hợp nào đó.

Ví dụ 3.10:

```
void main()
{
    int i, j ;
    printf(" nhập hai số nguyên dương i và j :");
    scanf("%d%d",&i,&j);
    for( ; i > 0 &&j > 0; i- -,j- -)
    {
        if( j == 5 ) break;
          printf(" i = %d, j = %d ", i, j);
    }
}
```

Các vòng for có thể lồng với nhau để thực hiện một câu lệnh nào đó.

Ví dụ 3.11 : Tính bảng cửu chương

```
for( i = 1 ; i <= 9 ; i++)
{
     printf("\n bang cuu chuong thu %d ", i);
     for(j = 1 ; j <= 9 ; j ++)
          printf(" %d x %d = %d ", i, j, i * j);
}
```

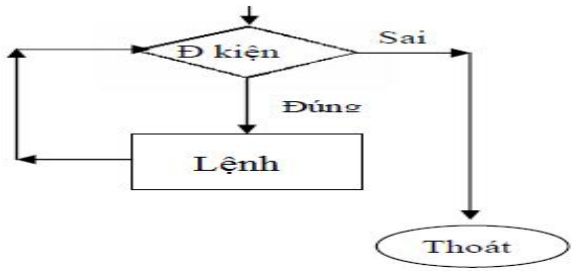
3.3.2 Lệnh while

Lệnh while thực thi việc lặp lại một khối lệnh khi điều kiện kiểm tra là đúng. Điều kiện sẽ được kiểm tra trước khi vào thân vòng lặp do đó nếu có thay đổi giá trị kiểm tra ở trong thân vòng lặp thì khối lệnh vẫn được thực thi cho đến khi kết thúc khối lệnh. Nếu điều kiện kiểm tra là sai (FALSE) ngay từ đầu thì khối lệnh sẽ không được thực hiện dù chỉ là một lần.

Cú pháp :

```
while(<biểu thức điều kiện>)
        < Lệnh >
```

Lưu đồ :



Quá trình thực hiện của vòng lặp while:

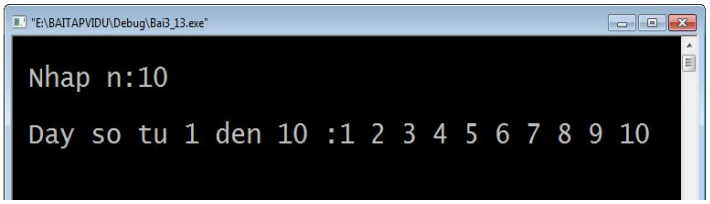
- Bước 1: Tính giá trị của biểu thức điều kiện.
- Bước 2: Nếu biểu thức điều kiện là sai (FALSE), thì máy sẽ thoát ra khỏi vòng lặp. Nếu biểu thức điều kiện là đúng (TRUE) thì máy sẽ thực hiện câu lệnh và quay lại bước 1.

Chú ý: Trong biểu thức điều kiện của vòng while có thể gồm nhiều biểu thức cách nhau bởi dấu phẩy “,” nhưng tính đúng sai của của nó là tính đúng sai của biểu thức sau cùng.

Ví dụ 3.12: In các số nguyên từ 1 đến n, trong đó n nhập từ phím

```
#include <stdio.h>
#include<conio.h>
int main ()
{
   int i,n;
   printf("\n Nhap n:"); scanf("%d", &n);
   printf("\n Day so tu 1 den %d :",n);
   i=1;
   while (i<=n)
       printf("%d ",i++);
   getch();
   return 0;
}
```

Kết quả thực thi chương trình khi nhập n=10



Để tránh xảy ra trường hợp lặp vô hạn, cần chú ý:

- Giá trị của biến được sử dụng trong biểu thức phải được thiết lập trước khi vòng lặp **while** thực hiện. Đây gọi là bước khởi tạo giá trị. Lệnh này chỉ thực hiện một lần trước khi thực hiện vòng lặp.

- Thân vòng lặp phải làm thay đổi giá trị của biến trong biểu thức kiểm tra. Biến này được gọi là biến tăng (**incremented**) nếu giá trị trong thân vòng lặp tăng, và được gọi là biến giảm (**decremented**) nếu giá trị giảm.

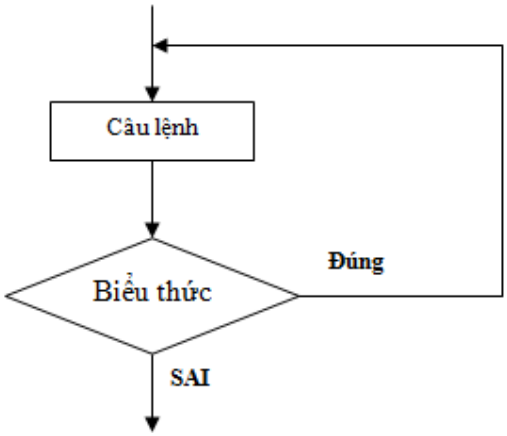
3.3.3 Lệnh do…while

Lệnh do...while thực thi việc lặp lại một khối lệnh nhiều lần. Nó thực hiện khối lệnh ít nhất một lần. Sau đó sẽ kiểm tra điều kiện nếu điều kiện là đúng thì tiếp tục thực thi khối lệnh cần lặp. Nếu điều kiện là sai thì kết thúc vòng lặp.

Cú pháp :

```
do < câu lệnh>

        while(<biểu thức>)
```

Lưu đồ :



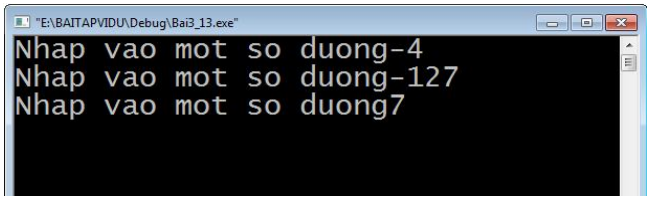
Quy trình thực hiện:

- Bước 1: Câu lệnh được thực hiện trước tiên.
- Bước 2: Tính giá trị của biểu thức, nếu biểu thức là đúng thì quay lại bước, nếu giá trị biểu thức là sai thì ngừng vòng lập

Ví dụ 3.13: Viết chương trình bắc buột nhập vào một số dương, nếu nhập số âm yêu cầu nhập lại.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
     int value;
     do
     {
          printf( "Nhap vao mot so duong");
          scanf("%d",&value);
     } while(value <=0);
     getch();
     return 0;
}
```

Kết quả thực thi chương trình



Người sử dụng được nhắc nhập vào số dương value, điều kiện đứng sau while được kiểm tra. Nếu một giá trị dương được nhập vào, khi đó điều kiện sai và vòng lặp kết thúc. Ngược lại, câu thông báo được hiển thị cho tới khi người sử dụng nhập vào một số dương.

Bài tập chương 3

1. Viết chương trình nhập vào số nguyên, kiểm tra số đã nhập là số âm hay số dương
2. Viết chương trình nhập vào 2 số a, b. Tìm số lớn nhất giữa 2 số.

3. Viết chương trình nhập vào 1 số nguyên, kiểm tra số vừa nhập là số chẵn hay lẽ.
4. Viết chương trình nhập vào 2 số nguyên, so sánh 2 giá trị vừa nhập vào ("Băng nhau, nhỏ hơn, lớn hơn").
5. Viết chương trình nhập vào 2 số a,b. Kiểm tra a có là bội số của a không.
6. Viết chương trình nhập vào đơn giá 1 mặt hàng, và số lượng bán của mặt hàng. Tính tiền khách phải trả, với thông tin như sau:

   Thành tiền: đơn giá * số lượng

   Giảm giá : Nếu thành tiền > 100, thì giảm 3% thành tiền, ngược lại không giảm

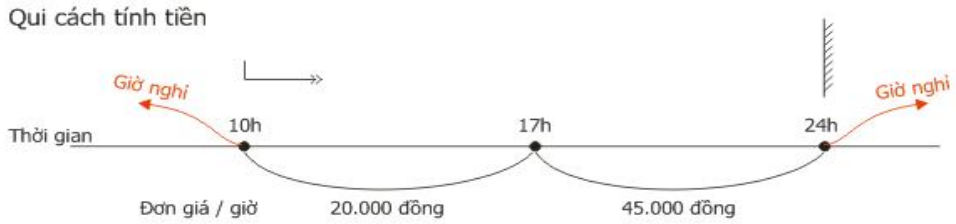
   Tổng tiền phải trả: thành tiền – giảm giá .

7. Viết chương trình hỗ trợ cách giải phương trình bậc 1 (ax + b = 0)
8. Viết chương trình nhập vào 1 kí tự, cho biết kí tự này là nguyên âm hay phụ âm
9. Nhập vào thứ tự của tháng, cho biết tháng đó có bao nhiêu ngày.
10. Nhập vào 1 số bất kỳ (0->9), cho biết cách đọc số vừa nhập
11. Tính chu vi, diện tích của một trong các hình bên dưới (người dùng chọn hình)

    Hình vuông, Hình chữ nhật, Hình tròn
12. Viết chương trình nhập vào 2 số nguyên và lựa chọn phép toán (cộng, trừ, nhân, chia) để tính kết quả.
13. Viết chương trình in trên màn hình các số từ 1->100, các số ngăn cách nhau bởi 1 đoạn khoảng trắng.
14. Viết chương trình tính tổng: 1 + 2 + 3 + 4 + 5 +….+ 20
15. Viết chương trình tính tích: 1*2*3*4*5*….*n, trong đó n nhập từ hím.
16. Viết chương trình tính tổng: 2 + 4 + 6 + 8 +….+ 20.
17. Viết chương trình tính tổng: 1*2 + 2*3+ 3*4 + 4*5 +…..+ n(n+1).
18. Viết chương trình tính tổng: $\frac{1}{1.2.3} + \frac{1}{2.3.4} + \frac{1}{3.4.5} + ... + \frac{1}{n(n+1)(n+2)}$
19. Viết chương trình in bảng cửu chương 1 số
20. Viết chương trình in bảng cửu chương từ 1 -> 9 theo hàng ngang
21. Viết chương trình vẽ hình chữ nhật có kích thước d x r, trong đó d là chiều dài, và r là chiều rộng được nhập từ phím.

```
*   *   *   *   *

*   *   *   *   *

*   *   *   *   *
```

22. Viết chương trình hiển thị tất cả các số lẻ nhỏ hơn n, trong đó n nhập từ phím.
23. Viết chương trình tính tổng các số chẵn nhỏ hơn n, trong đó n nhập từ bàn phím
24. Viết chương trình in ra các số là bội số của 5 nhỏ hơn n, trong đó n nhập từ phím.
25. Viết chương trình nhập vào 2 số nguyên, tìm USCLN, BSCNN.
26. Viết chương trình tính tiền karaoke theo cách sau:



27. Viết chương trình tính tính tiền điện sử dụng trong tháng:

    Từ 1 – 100KW: 5$

    Từ 101 – 150KW: 7$

    Từ 151 – 200KW: 10$

    Từ 201 – 300KW: 15$

    Từ 300KW trở lên: 20$

28. Viết chương trình đếm số lượng số chẵn trong [n,m], trong đó n,m nhập từ phím.
29. Viết chương trình tính tổng các số tự nhiên nhỏ hơn n (sử dụng vòng lặp While)
30. Viết chương trình tìm tổng các số tự nhiên lớn nhất nhỏ hơn 100

CHƯƠNG 4. HÀM

(FUNCTION)

4.1 Khái niệm hàm

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình; người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một câu lệnh nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn. Trong C, chương trình con được gọi là hàm. Hàm trong C có thể trả về kết quả thông qua tên hàm hay có thể không trả về kết quả.

Hàm main() là một hàm đặc biệt của ngôn ngữ lập trình C và là hàm đầu tiên được thực hiện trong chương trình. Khi thực hiện, hàm này sẽ gọi các hàm khác để thực hiện các chức năng riêng rẽ. Các hàm có thể ở trên các tập tin khác nhau và được biên dịch riêng lẻ, sau đó được ghép nối với nhau thành chương trình hoàn chỉnh.

Hàm có hai loại: hàm chuẩn và hàm tự định nghĩa. Trong chương này, ta chú trọng đến cách định nghĩa hàm và cách sử dụng các hàm đó.

Một hàm khi được định nghĩa thì có thể sử dụng bất cứ đâu trong chương trình. Trong C, một chương trình bắt đầu thực thi bằng hàm main.

Ví dụ 4.1: Hàm hàm số lớn giữa 2 số nguyên a, b.

```
int max(int a, int b)
{
     return (a>b) ? a:b;
}
```

Ví dụ 4.2: Chương trình nhập vào 2 số nguyên a,b và in ra màn hình số lớn nhất trong 2 số.

```
#include <stdio.h>
```

```
#include <conio.h>
int max(int a, int b)
{
    return (a>b) ? a:b;
}
int main()
{
    int a, b;
    printf("\n Nhap vao 2 so a, b");
    scanf("%d%d",&a,&b);
    printf("\n So lon la %d",max(a, b));
    getch();
    return 0;
}
```

4.2 Định nghĩa hàm

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình. Định nghĩa một hàm được hiểu là việc chỉ rõ các lệnh cần phải thực hiện mỗi khi hàm đó được gọi đến. Như vậy, có thể nói rằng tất cả các hàm được sử dụng trong một chương trình đều phải có một định nghĩa tương ứng cho nó.

Các hàm này nếu không có sẵn trong thư viện chúng ta phải định nghĩa trước. Nếu các hàm có sẵn sẽ được ngôn ngữ lập trình C tìm và lấy ra từ thư viện.

Cú pháp:

<Tên kiểu kết quả> <Tên hàm> ([<kiểu t số> <tham số >][…])
{
[<Khai báo biến cục bộ và các câu lệnh thực hiện hàm>]
[return <Biểu thức>;]
}

Trong đó:

- *Tên kiểu kết quả:* là kiểu dữ liệu của kết quả trả về, có thể là : int, byte, char, float, void… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp *hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void.*

- *Tên_hàm:* phải đặt hợp lệ và không trùng với một biến nào hoặc một từ khoá nào của C.

- *Kiểu tham số*: là kiểu dữ liệu của tham số.

- *Tham số:* là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. *Tham số này gọi là tham số hình thức, khi gọi hàm chúng ta phải truyền cho nó các tham số thực tế.* Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- *Bên trong thân hàm (phần giới hạn bởi cặp dấu {})* : là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

Khi định nghĩa hàm, ta thường sử dụng câu lệnh return để trả về kết quả thông qua tên hàm. Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

```
return ; /*không trả về giá trị*/

return <biểu thức>; /*Trả về giá trị của biểu thức*/

return (<biểu thức>); /*Trả về giá trị của biểu thức*/
```

Nếu hàm có kết quả trả về, ta bắt buộc phải sử dụng câu lệnh return để trả về kết quả cho hàm.

4.3 Thực thi hàm

Cú pháp: **<Tên hàm> ([Danh sách các tham số])**

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi ta có một lời gọi đến hàm đó. Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tương ứng*.

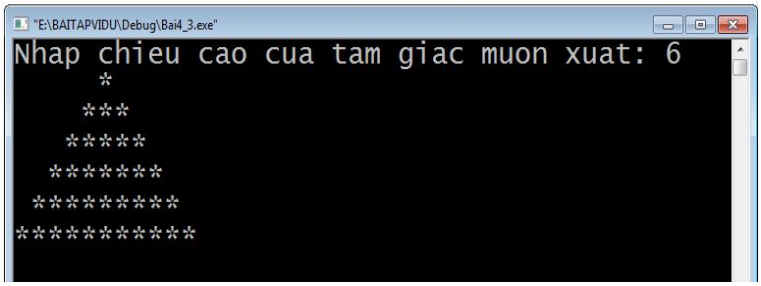
- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.

- Khi gặp lệnh return hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

Ví dụ 4.3: Xuất ra màn hình một tam giác cân hợp bởi các kí tự '*'

```
#include<stdio.h>
#include<conio.h>
void XuatC(char x, int n)
{    int i;
     for(i = 1; i <= n; i++) putchar(x);
}
// Hàm chính
void main()
{    int h, i;
     printf("Nhap chieu cao cua tam giac muon xuat: ");
     scanf("%d", &h);
     for(i = 1; i <= h; i++)
     {    XuatC(' ', h-i); //gọi hàm thực thi
          XuatC('*',2*i-1); //gọi hàm thực thi
          XuatC('\n',1); //gọi hàm thực thi
     }
     getch();
}
```

Kết quả thực thi chương trình

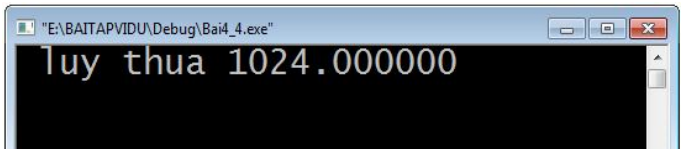


Ở ví dụ trên dòng **void XuatC(char x, int n)** sẽ cho ta biết rằng tên hàm là **XuatC**, các đối số là x, n lần lượt có kiểu là char và int. Từ khóa void cho ta biết rằng hàm này không trả về giá trị nào cả.

Ví dụ 4.4: Chương trình tính luỹ thừa một số

```
#include<stdio.h>
#include<conio.h>
float luythua(float x, int k)
{
    float r =1;
    while(k > 0)
    {
       r *= x;
       k--;
    }
    return r;
}
// Hàm chính
void main()
{
    int k;
    float n, nk ;
    n = 4; k = 5;
    nk = luythua(n, k);
    printf(" luy thua %f", nk);
    getch();
}
```

Kết quả thực thi chương trình



Trong hàm lũy thừa ta có thể thấy rằng x và k là đối số, và khi đó trong hàm main() ta truyền đối số vào là n, k cho hàm luythua(x,k)

Dòng lệnh nk = luythua(n, k); sẽ thực hiện việc lũy thừa và trả về kết quả này cho biến nk.

Trong ngôn ngữ lập trình C có ba loại biến :

- **Biến toàn cục (global)**: Là biến khai báo bên ngoài mọi hàm

- **Biến cục bộ (Local):** Là biến khai báo bên trong một hàm. Biến này chỉ tồn tại khi hàm đang được thực thi. Vùng nhớ của biến cục bộ sẽ bị thu hồi khi hàm thực hiện xong

- **Biến tĩnh(static):** Khai báo biến với chỉ thị **static** bên trong một hàm. Biến này vẫn tồn tại sau khi hàm thực thi xong.

4.4 Truyền tham số

Khi thực hiện một câu lệnh, chúng ta luôn luôn cần dữ liệu để làm tác động. Dữ liệu mà hàm sẽ tác động gọi là tham số (parameter) của hàm. Các tham số này ta xem như là dữ liệu đã có rồi.

Trong C ta dùng kỹ thuật truyền tham số bằng trị để không thay đổi giá trị và truyền tham số bằng địa chỉ hoặc tham chiếu để thay đổi giá trị của tham số. Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị; nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức.

Ví dụ 4.5:

```
void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

Ta thực hiện hàm main() như sau:

```
void main()
{
    int a,b;
    a = 10; b = 20;
    swap(a,b);
}
```

Giá trị của a và b sau khi thực hiện hàm main() là : a = 10, b = 20

Ta có thể viết hàm swap như sau :

Ví dụ 4.6:

```
void swap(int &x,int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
void main()
{
    int a,b;
    a = 10; b = 20;
    swap(a,b);
}
```

Giá trị của a và b sau khi thực hiện hàm main() : a = 20, b = 10

4.5 Kết quả trả về:

Nếu hàm trả về giá trị, giá trị đó phải được trả về thông qua lệnh return.

Ví dụ 4.8: Hàm kiểm tra kí tự có phải là một chữ số hay không.

```
int isdegit(char C)
{
    return ((C >= '0'&&C <= '9')?1:0);
}
```

Nếu một hàm không trả về giá trị thì không cần dùng lệnh return.

Ví dụ 4.9:

```
double mean(int num1,int num2)
{
    return ( num1 + num2)/2;
}
```

4.6 Prototype của hàm

Dạng prototype của một hàm có dạng như sau:

<Tên Kiểu> <Tên Hàm> ([Danh sách các đối số]);

Trong đó kiểu bắt buộc phải có là kiểu trả về của hàm. Kiểu này có thể là một trong những kiểu cơ bản hay kiểu void. Danh sách đối số là khai báo các đối số của hàm, các đối số của hàm được ngăn cách với nhau bằng dấu phẩy.

Ví dụ 4.10:
```
double atof(char s[]);
void func(int i,int j);
double luythua(double n, int so_mu);
```

Prototype của một hàm không những được dùng để khai báo kiểu của kết quả trả về của một hàm mà còn được dùng để kiểm tra các đối số và chuyển kiểu đối số. Việc đưa một prototype của một hàm giúp cho chương trình biên dịch có thể kiểm tra chặt chẽ hơn.

Ví dụ 4.11:

```
double luythua(double n,int pow);
```

và hàm main() được thực hiện như sau:

```
void main()
{
    int num;
    num = 123;
    num = luythua(num,2);
    printf("%6d",num);
}
```

num là một biến có kiểu **int** nhưng hàm lũy thừa vẫn thực hiện đúng nhờ sự chuyển kiểu bắt buộc của **prototype.**

4.7 Các hàm chuẩn

Bên cạnh việc chúng ta có thể tự tạo hàm, tất cả các ngôn ngữ lập trình đều có một số hàm đã được định nghĩa trước. Trong nhiều trường hợp, chúng ta dựa vào các hàm chuẩn để tạo hàm riêng của mình.

Ví dụ 4.12 : avg() – tính trung bình.

sqrt() – tính căn bậc hai

4.8 Thư viện hàm

Thư viện hàm là tập hợp các hàm đã được xây dựng trước. Mỗi thư viện hàm chứa các hàm theo một công dụng riêng. Ví dụ, thư viện hàm toán học chứa các hàm về phép tính toán học như căn bậc hai, trung bình, và lũy thừa, …Ví dụ: math.h

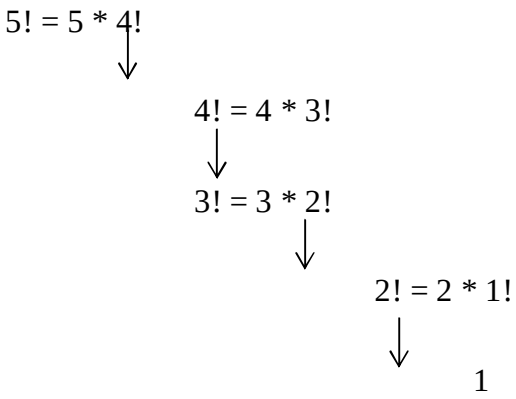
4.9 Sự đệ quy

Đôi khi chúng ta rất khó định nghĩa đối tượng một cách tường minh, nhưng có thể dễ dàng định nghĩa đối tượng này qua chính nó . Kỹ thuật này gọi là đệ quy, đó cũng chính là việc định nghĩa nội dung thông qua chính bản thân của nó nhưng ở mức độ nhỏ hơn.

Ta xét một ví dụ minh họa phương pháp đệ quy qua một bài toán tính giai thừa. Ta biết rằng n! được định nghĩa như sau:

$$\begin{cases} 1! = 1 \\ n! = n*(n-1)\ ! \end{cases}$$

Áp dụng định nghĩa trên ta có thể tính 5! như sau :



Suy ra 5! = 120

Một thuật toán được gọi là đệ quy nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu tới bài toán giống như vậy nhưng có dữ liệu đầu vào nhỏ hơn.

Vì vậy, tư tưởng giải bài toán bằng đệ quy là đưa bài toán hiện tại về một bài toán cùng loại, cùng tính chất nhưng ở cấp độ thấp hơn chẳng hạn: độ lớn dữ liệu nhập nhỏ hơn, giá trị cần tính toán nhỏ hơn,… và quá trình này tiếp tục cho đến khi bài toán được đưa về một cấp độ mà tại đó có thể giải được. Từ kết quả ở cấp độ

này, chúng ta sẽ đi ngược lại để giải bài toán ở cấp độ cao hơn cho tới khi giải được bài toán ban đầu.

Một thuật toán đệ quy gồm hai phần:

- Phần cơ sở

Là các trường hợp không cần thực hiện lại thuật toán cũng có nghĩa là khi làm đến đây sẽ không có việc gọi đệ quy nữa mà ở đây chỉ là một câu lệnh đơn giản dùng để kết thúc phần đệ quy. Nếu thuật toán đệ quy không có phần này thì sẽ dẫn đến việc lặp vô hạn và sẽ xuất hiện lỗi khi thi hành.

- Phần đệ quy

Là phần trong thuật toán có yêu cầu gọi đệ quy, tức là yêu cầu thực hiện lại thuật toán nhưng với cấp độ dữ liệu thấp hơn. Phần đệ quy này được dựa trên cơ sở công thức quy nạp của bài toán.

Ví dụ 4.13: Hàm đệ quy tính giai thừa của số n :

```
float giai_thua(int n)
{
   if (n==0) return 1;
   return n*giai_thua(n -1);
}
```

Bài tập chương 4

1. Viết chương trình tính tổng 2 số tự nhiên (sử dụng chương trình con).
2. Bài 2: Viết chương trình cho phép chọn phép toán (+, - , *, /) để tính kết quả.
3. Bài 3: Viết chương trình tính f(x) =2 x5 + x2 + 3x3
4. Bài 4: Viết chương trình tính f(x) = 11 + 22 + 33 + 44 + 55 + 66
5. Bài 5: Viết chương trình tính f(x) =2 (a+b)5 + (a+b)2 + 3(a+b)3
6. Bài 6: Viết chương trình tính tổng: 1! + 2! + 3! + …. + n!, trong đó n nhập từ phím.
7. Bài 7: Viết chương trình hoán đổi giá trị giữa 2 biến số nguyên.
8. Bài 8: Viết chương trình đếm số nguyên tố nhỏ hơn n (n nhập từ phím) và in các số nguyên tố đó ra màn hình

CHƯƠNG 5. MẢNG VÀ CON TRỎ

(ARRAY & POINTER)

5.1 Mảng 1 chiều

5.1.1 Khái niệm và khai báo mảng 1 chiều

Mảng 1 chiều là một nhóm các phần tử có cùng kích thước, cùng kiểu dữ liệu. Những phần tử này được lưu liên tiếp với nhau trong bộ nhớ. Số phần tử của mảng gọi là kích thước của mảng.

Cú pháp : <Tên kiểu dữ liệu> <Tên mảng> [<số phần tử>];

Trong đó**:**

- Tên kiểu dữ liệu: là kiểu dữ liệu mà mỗi phần tử mảng có dữ liệu thuộc vào.
- Tên mảng: là tên được đặt theo qui tắc đặt tên của danh biểu trong ngôn ngữ lập trình C, còn mang ý nghĩa là tên biến mảng.
- Số phần tử: là 1 hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu.

Ví dụ 5.1:

```
//Khai báo mảng 1 chiều tên a có 20 phần tử kiểu số
    nguyên int.
int a[20];
//Khai báo mảng 1 chiều tên b có 10 phần tử kiểu ký
    tự char.
char    b[10];
```

Mỗi phần tử của mảng 1 chiều được truy nhập giá trị thông qua chỉ số (index) của nó. Chỉ số để xác định phần tử nằm ở vị trí nào trong mảng. Phần tử đầu tiên của mảng có chỉ số là 0, thành phần thứ hai có chỉ số là 1…và tương tự tăng dần cho hết mảng.

Ví dụ 5.2:

```
int  num[5];
```

Chỉ số và giá trị phần tử của mảng 1 chiều num được biểu diễn như sau:

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	num[0]	num[1]	num[2]	num[3]	num[4]

Ở ví dụ 5.2, mảng có 5 phần tử và chỉ số của mảng bắt đầu từ 0 cho nên chỉ số để truy xuất phần tử cuối cùng của mảng là 4. Như vậy, nếu một mảng có n phần tử thì chỉ số cuối cùng của mảng là (n-1).

Chỉ số của mảng có thể là một giá trị cụ thể, giá trị của một biến hay giá trị được tính toán từ một biểu thức đại số.

Ví dụ 5.3 :

```
int i = 3;
int a[20];
a[1]  /* truy cập phần tử thứ 2 của mảng a, vì
    phần tử thứ 1 có chỉ số là 0  */
a[i]  // truy cập phần tử thứ 4 của mảng a
a[i*2 - 1]// truy cập phần tử thứ 6
```

5.1.2 Gán giá trị vào các phần tử của mảng

Thông qua chỉ số phần tử của mảng, chúng ta cũng có thể thay đổi giá trị của các phần tử trong mảng.

Ví dụ 5.4:

Cho mảng num : `int num[5];`

Để gán 10 vào phần tử thứ 3 của mảng num, ta viết : `num[2] = 10;`

Trong lập trình, câu lệnh này có nghĩa là giá trị biểu thức bên phải được gán vào biểu thức bên trái. Do đó, phần tử thứ 3 của mảng num sẽ chứa giá trị 10 sau khi thực hiện câu lệnh trên.

Chỉ số mảng	0	1	2	3	4

Giá trị phần tử trong mảng	num[0]	num[1]	10	num[3]	num[4]

Để chứa các kí tự chữ ‘i’, ‘o’, ‘g’, ‘i’, ‘c’ – chúng ta có thể khai báo mảng ký tự ch như sau:

```
char ch[5];
ch[0] = ‘i’;
ch[1] = ‘o’;
ch[2] = ‘g’;
ch[3] = ‘a’;
ch[4] = ‘c’;
```

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	i	o	g	a	c

5.1.3 Lấy giá trị các phần tử trong mảng

– Khi muốn lấy giá trị một phần tử trong mảng ta tại vị trí ***chỉ số phần tử*** có cú pháp như sau:

<tên mảng>[<chỉ số phần tử>]

Chẳng hạn muốn truy xuất phần tử thứ i trong mảng a, ta ghi là a[i].

– Khi khai báo 1 biến để nhận giá trị của mảng thì biến này phải có cùng kiểu dữ liệu với phần tử của mảng.

Ví dụ 5.5:

Giả sử có mảng 1 chiều num có 5 phần tử là số nguyên như sau:

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	20	30	10	4	6

```
int i;
```

```
i = num[4]; // i nhận giá trị 6
```

Lệnh trên sẽ lấy giá trị của phần tử thứ 5 của mảng, và giá trị này sẽ được chứa trong biến i.

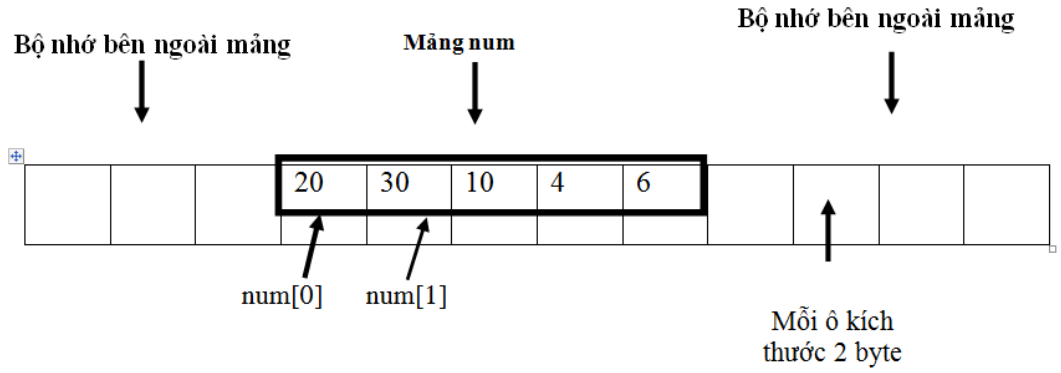
5.1.4 Các phần tử của mảng trong bộ nhớ

Bộ nhớ của máy tính được sắp xếp theo từng byte.

Khi ta khai báo : `int diem;` thì diem là 1 biến có kiểu int. Kích thước lưu trữ của int trong ngôn ngữ lập trình C thường là 2 byte. Vì thế, 2 byte này được cố định trong bộ nhớ và được tham chiếu bằng tên diem.

Như vậy, khi ta khai báo **int num[5];** thì num là mảng 1 chiều có 5 phần tử số nguyên, mỗi số nguyên cần 2 byte. Do đó, bộ nhớ cần cho mảng num là: 5*2= 10 bytes, và những phần tử của mảng này được chứa trong vùng nhớ liên tục.

Xét mảng num có 5 phần tử ở ví dụ 5.5, ta có mô phỏng cách lưu trữ dữ liệu của các phần tử trong bộ nhớ như sau:



5.1.5 Khởi tạo mảng

Trong ngôn ngữ lập trình C, chúng ta có thể khởi tạo giá trị các phần tử mảng ngay khi mảng được khai báo. Việc khởi tạo cho mảng được thực hiện lúc khai báo mảng bằng một loạt giá trị hằng khởi động cho các phần tử của mảng. Các giá trị hằng được đặt giữa một cặp ngoặc nhọn ({}), các phần tử cách nhau bằng dấu phẩy (,).

Ví dụ 5.6: Khởi tạo mảng 1 chiều a chứa số nguyên có 10 phần tử với các giá trị 5, 15, 20, 25, 30 như sau:

```
int a[10] = {5, 15, 20, 25, 30};
```

Trong việc khởi tạo mảng, kích thước của mảng không cần xác định, chương trình C sẽ đếm số phần tử được khởi động và lấy đó làm kích thước. Nếu có xác định kích thước, thì số giá trị được khởi tạo liệt kê phải không được lớn hơn kích thước đã khai báo

Ví dụ 5.7: Ta khai báo

```
double b[ ] = {4.5, 7.5, 8.2, 4.32};
```

thì mảng b sẽ được hiểu là có 4 phần tử kiểu double.

Nếu ta khai báo int a[10] = {2, 5, 6, 1, 0, 6, 7}; thì ta có một mảng gồm 10 phần tử. Trong đó có 7 phần tử đầu tiên được khởi tạo, các phần tử còn lại coi như chưa khởi tạo (bình thường nhận giá trị 0).

Lưu ý:

Trình biên dịch C không báo lỗi khi có sự vượt quá giới hạn của mảng. Chẳng hạn đoạn chương trình sau đây vẫn chấp nhận trong khi có sự vượt qua giới hạn của mảng buf:

```
#include<stdio.h>
void main()
{
    int i;
    char buf[5] = {'a','b','c','d'};
    for(i = 0; i<= 10; i++)
    printf("%c",buf[i]);
}
```

Kết quả thực thi của chương trình:

5.2 Mảng 2 chiều.

5.2.1 Khái niệm

Mảng 2 chiều m dòng n cột được xem như là 1 bảng hình chữ nhật chứa m*n phần tử cùng kiểu dữ liệu (còn gọi là ma trận m*n). Nó sự mở rộng trực tiếp của 1 chiều. Nói cách khác, mảng 2 chiều là mảng 1 chiều mà mỗi phần tử của nó là 1 mảng 1 chiều.

Cú pháp khai báo mảng 2 chiều:

Cú pháp: <tên kiểu dữ liệu> <tên mảng> [<số dòng>] [<số cột>];

Chẳng hạn khai báo mảng 2 chiều a có 5 dòng 3 cột chứa các số nguyên, ta ghi là: int a[5][3];

5.2.2 Chỉ số của mảng

Ta xét bảng điểm 3 môn học của các sinh viên như sau:

	Toán	**Lý**	**Hóa**
Minh	7	8	10
Lan	4	6	8
Nhật	9	10	10
Ngọc	3	5	7

Trong bảng thông tin trên, tiêu đề dòng chứa tên sinh viên và tiêu cột chứa môn học. Chúng ta lưu trữ điểm 3 môn học của mỗi sinh viên. Để đọc từng thông tin riêng biệt, cần xác định vị trí dòng, cột và đọc thông tin tại vị trí đó. Xét từ bảng điểm trên, tìm điểm môn toán của Lan như sau:

Tại dòng thứ hai của bảng chứa các điểm môn học của Lan. Cột thứ hai của dòng này chứa điểm môn Toán. Vì thế, điểm toán của Lan là 4. Do đó, ứng với cấu trúc loại này chúng ta có thể sử dụng một mảng hai chiều để lưu trữ.

Trong mảng 2 chiều, cách xác định chỉ số cũng như mảng một chiều. Đó là chỉ số dòng cột bắt đầu từ 0.

Chúng ta có thể khai báo mảng hai chiều diem để lưu trữ bảng thông tin về điểm các sinh viên như sau: `int diem[4][3];`

	Cột 0	Cột 1	Cột 2
Dòng 0	diem[0][0]=7	diem[0][1]=8	diem[0][2]=10
Dòng 1	diem[1][0]=4	diem[1][1]=6	diem[1][2]=8
Dòng 2	diem[2][0]=9	diem[2][1]=10	diem[2][2]=10
Dòng 3	diem[3][0]=3	diem[3][1]=5	diem[3][2]=7

Trong mảng 2 chiều diem trên, mảng có 4 dòng, 3 cột. Chỉ số dòng của mảng từ 0 đến 3, chỉ số cột của mảng từ 0 đến 2.

5.2.3 Truy xuất phần tử mảng 2 chiều

Mỗi phần tử mảng được truy xuất thông qua cú pháp sau:

<Tên mảng> [<chỉ số dòng phần tử>] [<chỉ số cột phần tử>]

Chẳng hạn muốn truy xuất một phần tử tại dòng thứ i, cột thứ j của mảng a, ta ghi là a[i][j]. Giá trị i, j được tính từ 0 trở đi.

Ví dụ 5.8: Truy xuất phần tử dòng 2 cột 1 của mảng diem như sau:

```
diem[1][0]
```

5.2.4 Khởi tạo mảng 2 chiều

Ta có thể khởi tạo mảng 2 chiều với những giá trị đầu tiên khi mảng này là biến được khai báo.

Ví dụ 5.9:

```
int matrix[5][4] = {  {11, 12, 13, 14},
                      {21, 15, 41, 16},
                      {43, 58, 24, 91},
```

```
                        {32, 15, 25, 16},
                        {56, 23, 45, 47}
                    };
char table[7][4] = {"MON","TUE","WED","THU",
                    "FRI","SAT", "SUN"};
```

5.3 Con trỏ (Pointer)

Chúng ta đã biết các biến được chứa trong bộ nhớ. Mỗi vị trí các biến được chứa trong bộ nhớ thì được gán cho một con số duy nhất gọi là địa chỉ (address). Thông qua địa chỉ, chúng ta có thể biết được biến đó lưu trữ ở đâu trong bộ nhớ. Tương tự như vậy mỗi phần tử của mảng đều có một địa chỉ riêng. Con trỏ là một dạng biến để chứa loại địa chỉ này.

5.3. 1. Khái niệm

Pointer (con trỏ) là một kiểu dữ liệu đặc biệt dùng đê quản lý địa chỉ của các ô nhớ. Một con trỏ quản lý các địa chỉ mà dữ liệu tại các địa chỉ này có kiểu T thì con trỏ đó được gọi là con trỏ kiểu T. Con trỏ kiểu T chỉ được dùng để chứa địa chỉ của biến kiểu T. Nghĩa là con trỏ kiểu int chỉ được dùng để chứa biến kiểu int, con trỏ kiểu char chỉ được dùng chứa biến kiểu char.

Pointer là một phần cơ bản quan trọng của ngôn ngữ lập trình C. Nó là cách duy nhất để thể hiện một số thao tác truy xuất và dữ liệu; nó tạo ra mã code đơn giản, hiệu quả, là một công cụ thực thi mạnh mẽ.

5.3. 2. Khai báo biến con trỏ

Cú pháp khai báo biến con trỏ:

<tên kiểu dữ liệu> *<tên biến con trỏ>

Ví dụ 5.10:

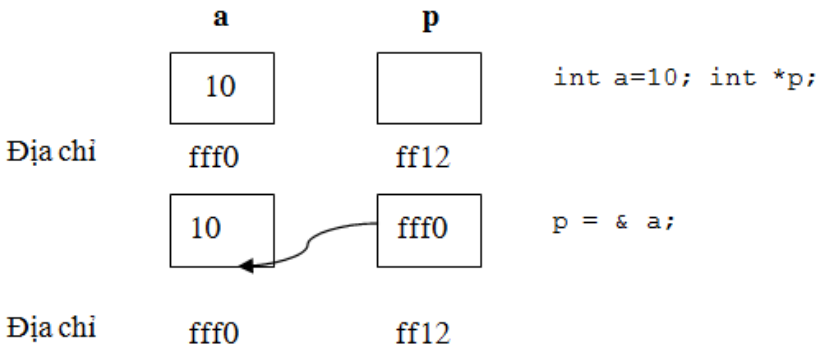
```
// x là biến kiểu int, còn px là con trỏ kiểu int.
int x, *px;
```

px được khai báo là một con trỏ kiểu int, nó chứa địa chỉ của biến kiểu dữ liệu số nguyên. Dấu * không phải là một phần của biến, int * có nghĩa là con trỏ kiểu int.

Đặt tên biến con trỏ giống như tên của các biến khác. Để gán địa chỉ vào con trỏ chúng ta cần phải gán giá trị cho biến như sau:

Ví dụ 5.11:

```
int a = 10;
int *p;
p = &a;// giá trị p chứa địa chỉ của biến a
```



Ví dụ trên được hiểu như sau:

- a là biến kiểu int được khởi tạo bằng 10.
- p là biến con trỏ kiểu int, chứa địa chỉ của kiểu dữ liệu int, lúc này nó không chứa giá trị (hay chứa giá trị NULL).
- Câu lệnh p = &a có nghĩa là "gán địa chỉ của a vào p". Biến con trỏ này bây giờ chứa địa chỉ của biến a.
- Giả sử địa chỉ của biến a và p trong bộ nhớ là fff0 và ff12. Câu lệnh p = &a để gán địa chỉ của a vào p. Dấu '&' viết phía trước biến a được gọi là phép toán địa chỉ (address). Vì thế biến con trỏ này chứa giá trị fff0.

Mặc dù chúng ta khai báo biến con trỏ với dấu '*' ở phía trước, nhưng bộ nhớ chỉ gán cho p chứ không phải *p.

5.3. 3. Toán tử địa chỉ (&) và toán tử nội dung (*)

- **Toán tử địa chỉ (&)**

  Biến được khai báo là **x** thì **&x** là địa chỉ của x.

Kết quả của phép lấy địa chỉ (&) là một con trỏ, do đó có thể dùng để gán cho một biến pointer.

Ví dụ 5.12:

```
a) int *px, num;
// px là một pointer chỉ đến biến kiểu int là num.
px = &num;
//xuất địa chỉ của biến num dạng số hệ 16 (hệ hexa)
printf("%x", &num);
b) int *px, num;
px = &(num +4);    // SAI vì ( num+4) không phải là
                   một biến cụ thể
```

Lưu ý: Chúng ta thấy cú pháp lệnh nhập dữ liệu scanf (lệnh đã được học ở chương 2) trong ngôn ngữ lập trình C luôn có dấu & trước biến cần nhập. Điều này xác định cần đưa dữ liệu vào con trỏ chứa địa chỉ của biến tương ứng.

– **Toán tử nội dung (*)**

Toán tử lấy nội dung của một địa chỉ được kí hiệu là dấu * trước một pointer, dùng để lấy giá trị của biến mà con trỏ đang trỏ đến.

Xét lại ví dụ 5.12, ta có:

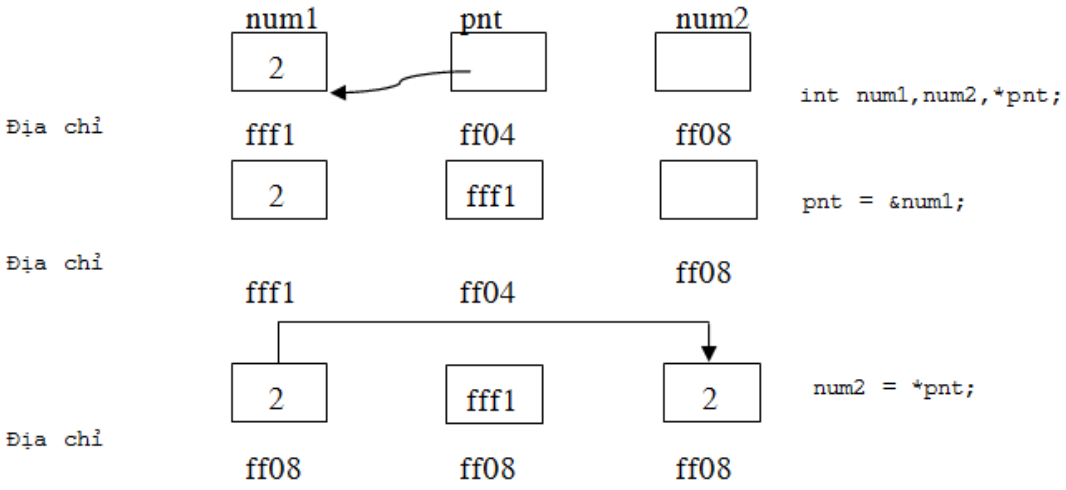
px là một pointer chỉ đến biến num như ví dụ 5.12 a, thì * px là giá trị của biến num.

Ví dụ 5.13:

```
a) //num là biến được khai báo và gán giá trị là 10.
int num = 10 ;
int *px; // px là một con trỏ chỉ đến kiểu int
px= &num ; //px là địa chỉ của biến num.
/*giá trị của *px (tức là num) cộng thêm 3, gán cho
  k. Sau đó *px thực hiện lệnh tăng 1 đơn vị (++)*/
int k = (* px)++  + 3 ;
// Sau câu lệnh trên num = 11, k = 13
b) int num1 = 2, num2, *pnt;
pnt = &num1
num2 = *pnt;
```

Trong ví dụ trên, biến num1 được gán bằng 2. Dòng pnt = &num1 nghĩa là biến con trỏ pnt chứa địa chỉ của biến num1. Phép gán num2 = *pnt, dấu '*' được đặt ở phía trước biến con trỏ, thì giá trị trả về của biến này l giá trị của biến được trỏ tới bởi con trỏ pnt. Do đó, num2 có giá trị là 2.

Ta minh họa qua hình vẽ sau :

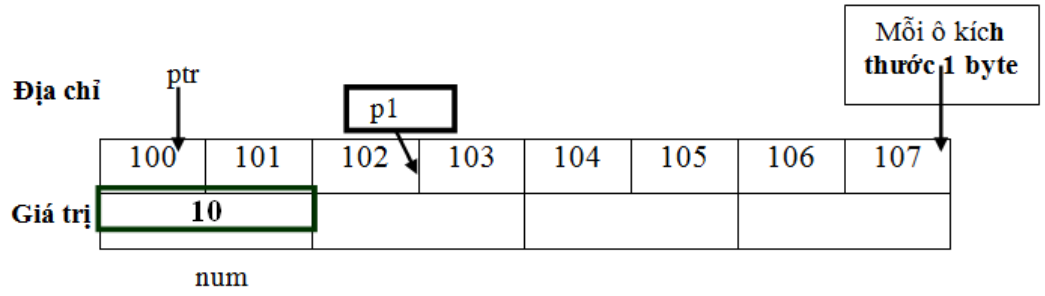


Lưu ý : NULL là hằng khi **pointer** mang ý nghĩa không chứa một địa chỉ nào cả. Ta gọi là **pointer** rỗng.

5.3. 4. Tính toán trên Pointer

Một biến pointer có thể được cộng trừ với một số nguyên (int, long) để cho kết quả là một pointer chỉ đến một vùng nhớ khác.

Ví dụ 5.14:
```
int *ptr, *p1;
int num  = 10;
ptr = &num;
p1 = ptr  + 2;
```



Việc cộng hoặc trừ pointer với một số nguyên n thì pointer sẽ chỉ đến một địa chỉ mới hay nói cách khác là chỉ đến một biến khác nằm cách biến đó n vị trí.

Ví dụ 5.15:

```
int  v[10]; // mảng 10 phần tử lin tiếp .
int  * p ; // Biến pointer chỉ đến một số int .
p= & v[0]; // p là địa chỉ phần tử đầu tiên của mảng
for( i =0; i<10 ; i++)
{
  *p= i * i; // gán cho phần tử mà p đang chỉ đến
  p ++ ;// p được tăng lên để chỉ đến phần tử kế tiếp
}
```

Lưu ý :

- Do cộng một pointer với một giá trị nguyên cho ta một pointer, nên phép trừ hai pointer vẫn được coi là hợp lệ, kết quả cho ta một giá trị int biểu thị khoảng cách (số phần tử) giữa 2 pointer đó.
- Phép cộng 2 pointer là không hợp lệ.
- Không thể nhân, chia , hoặc lấy dư của một pointer với bất kì một số nào.
- Đối với các phép toán khác, pointer được xử lý như một biến bình thường (gán, so sánh …), các toán hạng phải cùng kiểu pointer và cùng kiểu đối tượng của chúng. Mỗi sự chuyển kiểu tự động luôn được cân nhắc và xác nhận từ trình biên dịch.

Địa chỉ của một biến được xem là một pointer hằng và ngôn ngữ lập trình C cho phép thực hiện các phép toán mà pointer chấp nhận trên nó, trừ phép gán lại và phép tăng giảm vì ta không thể gán lại một giá trị hằng bằng một giá trị khác được.

Ví dụ 5.16:

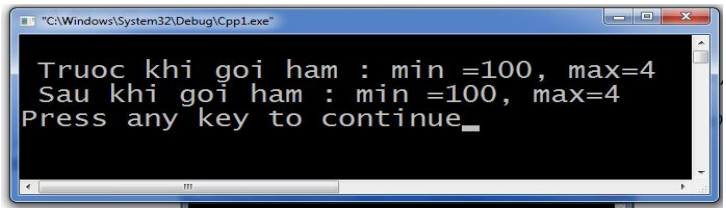
```
int p, *px, num;
px= &num ;//lấy địa chỉ của num gán vào biến: ĐÚNG
px++;  //tăng giảm trên một biến pointer: ĐÚNG
&p = px; //gán lại một địa chỉ hằng: SAI
&p++;  //tăng giảm một địa chỉ hằng:  SAI.
```

5.3. 5. Truyền tham số địa chỉ

Ví dụ 5.17:

```
void swap( int x, int y)
{
    int tmp = x; x=y; y=tmp;
}
void main()
{
    int  min =100, max = 4;
    printf("\n Truoc   khi goi ham : min =%d, max=%d
              ",min, max);
    swap ( min,max);
    printf("\n Sau khi gọi hàm : min =%d, max=%d
              ",min,max);
}
```

Kết quả thực thi chương trình:



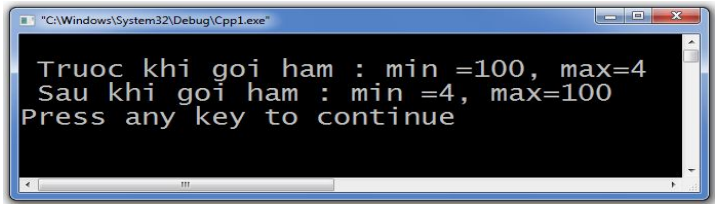
Sau khi gọi hàm ta không đạt được yêu cầu là hoán đổi giá trị min, max vì giá trị của biến không thay đổi khi ra khỏi hàm. Do đó ta phải dùng đến con trỏ.

Ví dụ 5.18:

```
void swap ( int* px, int* py)
{   int  tmp = *px;
    *px=*py;
    *py=tmp;
}
void main()
{
    int  min =100, max = 4;
printf("\n Trước   khi gọi hàm : min =%d, max=%d \n",
    min, max);
    swap ( &min, &max);
```

```
    printf("\n Sau    khi goi hàm : min =%d, max=%d \n",
        min, max);
    }
```

Kết quả thực thi chương trình:



Trong trường hợp này, do các pointer thực sự chỉ đến các biến min, max nên việc hoán đổi đối tượng các pointer này thực sự làm hoán đổi giá trị của 2 biến min, max ở hàm main(). Cách truyền tham số theo địa chỉ vào hàm khi ta muốn hàm đó có thể thay đổi giá trị của tham số mà chúng ta truyền vào.

5.4 Cấp phát và giải phóng vùng nhớ cho biến con trỏ

5.4.1 Cấp phát vùng nhớ cho biến con trỏ

Trước khi sử dụng biến con trỏ, ta nên cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp các hàm:

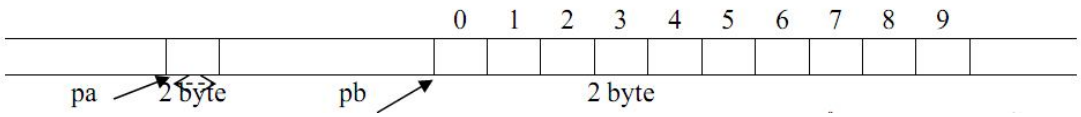
`void *malloc(size_t size):` Cấp phát vùng nhớ có kích thước size.

`void *calloc(size_t nitems, size_t size):` Cấp phát vùng nhớ có kích thước là nitems*size.

Ví dụ 5.18: Giả sử ta có khai báo:

```
int a, *pa, *pb;
pa = (int*)malloc(sizeof(int)); /* Cấp phát vùng nhớ
    có kích thước bằng với kích thước của một số
    nguyên */
pb= (int*)calloc(10, sizeof(int)); /* Cấp phát vùng
    nhớ có thể chứa được 10 số nguyên*/
```

Hình ảnh minh họa trong bộ nhớ như sau:



Lưu ý: Khi sử dụng hàm malloc() hay calloc(), ta phải ép kiểu vì nguyên mẫu các hàm này trả về con trỏ kiểu void.

Cấp phát lại vùng nhớ cho biến con trỏ

Trong quá trình thao tác trên biến con trỏ, nếu ta cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm realloc().

Cú pháp: void *realloc(void *block, size_t size)

Ý nghĩa:

- Cấp phát lại 1 vùng nhớ cho con trỏ block quản lý, vùng nhớ này có kích thước mới là size; khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại.
- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

Ví dụ 5.19: Trong ví dụ trên ta có thể cấp phát lại vùng nhớ do con trỏ pa quản lý như sau:

```
int a, *pa;
/*Cấp phát vùng nhớ có kích thước 2 byte*/
pa=(int*)malloc(sizeof(int));
/* Cấp phát lại vùng nhớ có kích thước 6 byte*/
pa = realloc(pa, 6);
```

Giải phóng vùng nhớ cho biến con trỏ

Một vùng nhớ đã cấp phát cho biến con trỏ, khi không còn sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp: void free(void *block)

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Ví dụ 5.20: Xét lại ví dụ 5.19, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

```
free(pa);
free(pb);
```

5.5 Sự liên hệ giữa cách sử dụng mảng và pointer

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

5.5.1 Khai thác một pointer theo cách của mảng

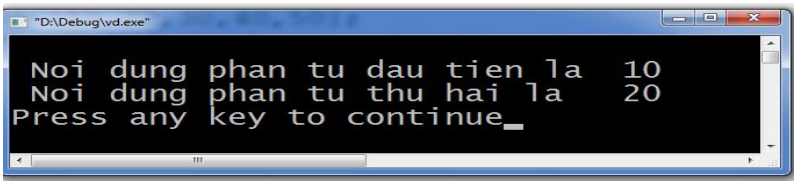
Khi ta khai báo int a[5], ngoài cách tiếp cận thông thường đã học, chúng ta còn có kiểu truy xuất, xử lý mảng bằng pointer như sau:

Ta được khai báo là một mảng thì mỗi sự truy xuất đến riêng tên a được hiểu là một pointer a truy xuất đến địa chỉ của phần tử đầu tiên của mảng a.

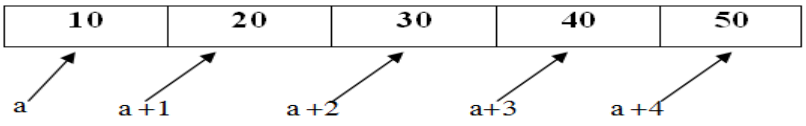
Ví dụ 5.21:

```
void main()
{
int a[10]={10,20,30,40,50};
printf("\n Noi dung phan tu dau tien la  %d",*a);
printf("\n Noi dung phan tu thu hai la   %d",
    *(a+1));
}
```

Chương trình thực thi:



Ta có thể thấy rõ hơn qua hình sau :



Nếu ta có int *pnum;

Muốn pnum trỏ đến phần tử đầu tiên của a ta có thể viết pnum = a; thay cho cách viết pnum =& a[0];

Vì bản thân tên mảng a lại có thể hiểu là một địa chỉ (hoặc một pointer) nên ta có sự tương đương ý nghĩa như sau:

*a	tương đương với	a[0]
*(a+1)	tương đương với	a[1]
…	…	…
*(a+i)	tương đương với	a[i]
a	tương đương với	&a[0]
a+1	tương đương với	&a[1]
…	…	-
a+i	tương đương với	&a[i]

5.5.2 Khai thác một mảng bằng pointer

Ví dụ 5.22:

```
int  num [10];
int *pnum;
//phép gán kể từ lúc này pnum sẽ chỉ về phần tử thứ
1 của mảng num.
pnum =&num[0];
// gán giá trị phần tử thứ 1 của mảng num
(num[0]) cho x.
x = *pnum;
```

Như vậy ta có thể hoàn toàn truy xuất đến mỗi phần tử của một mảng bằng cách sử dụng một pointer chỉ đến đầu mảng.

5.5.3 Những điểm khác nhau quan trọng giữa mảng và con trỏ

– Khi khai báo và định nghĩa mảng, vị trí của mảng đó được cấp rõ ràng và đủ theo kích thước được khai báo. Còn pointer thì vị trí được cấp chỉ là một chỗ cho bản thân của pointer còn vị trí mà pointer chỉ đến thì không được chuẩn bị sẵn.

- Pointer thực sự là một biến, ta có thể tăng giảm và gán lại trên biến pointer đó các địa chỉ khác nhau. Còn tên mảng lại là một địa chỉ hằng chỉ đến vùng mảng cố định, ta có thể sử dụng chứ không thể tăng giảm hoặc gán lại nó.

- Ta có thể lấy địa chỉ của một pointer trỏ đến, địa chỉ của một phần tử mảng chứ không thể lấy địa chỉ của một mảng.

- Pointer có thể thay đổi địa chỉ trỏ tới còn tên mảng thì không thể.

Ví dụ 5.23:

```
float readings[20],totals[20];
float *fptr;
 Các lệnh sau là hợp lệ
fptr = readings;// fptr chỉ tới phần tử đầu mảng readings
fptr = totals;  // fptr chỉ tới phần tử đầu mảng totals
Các lệnh sau là bất hợp lệ
readings = total;
totals = fptr;
```

5.5.4 Hàm có đối số là mảng

Cách khai báo tham số trong hàm khi tham số hàm là mảng.

Ví dụ 5.24: Ta xét ví dụ sau có hàm con là hàm đảo chuỗi:

```
char str[] = "ABCDEF";
void daoChuoi(char*s);
void main()
{
    printf("Trước khi đảo chuỗi : %s \n",str);
    daoChuoi (str);
    printf("Sau khi đảo chuỗi : %s \n",str);
}
void daoChuoi (char*s)
{
…
}
```

Ta có thể thấy rằng, trong hàm main() khi gọi daoChuoi () ta chỉ truyền đối số là tên mảng *str* mà thôi, điều đó có ý nghĩa là ngôn ngữ lập trình C chỉ gửi địa chỉ của mảng, nhưng địa chỉ này chỉ là một bản sao lại và thực chất là một biến pointer chỉ đến phần tử đầu tiên của mảng được gởi. Do đó, ta có thể nói rằng hàm có đối số là mảng không hề nhận được một mảng, nó chỉ nhận một pointer chỉ đến đầu mảng đó mà thôi.

Do đó, ta có thể khai báo đối số của một hàm là pointer hay là một mảng đều như nhau. Và nếu chúng ta khai báo là mảng một chiều, thì không cần xác định kích thước vì C không quan tâm đến điều đó. Ví dụ hàm daoChuoi () có thể khai báo lại như sau:

```
void daoChuoi(char s[])

{

   …

}
```

Hơn nữa, vì đối số này thực sự là biến pointer, nên ta có thể xử lý giống như biến pointer. Ta xét một cách viết khác của hàm strlen() (hàm xác định chiều dài chuỗi) như sau:

```
int strlen(char *s) // hay có thể viết là char s[]
{
  int i;
  for(i = 0; *s ! =’\0’; i++,s++);
       return i;
}
```

Cũng vì nguyên nhân trên, nên chúng ta cũng có thể bị thay đổi nội dung của các mảng khi truyền vào trong một hàm. Ta chỉ có thể tránh được tình trạng đó bằng cách khai báo các mảng đối số của hàm đó là các pointer chỉ đến const. Khi đó nếu hàm này thay đổi giá trị nội dung của mảng này, chúng ta sẽ nhận được thông báo của chương trình biên dịch.

Chẳng hạn ta có thể khai báo: int strlen(const char*s);

Khi đó ta có thể gọi hàm này cho mảng s của chúng ta gửi đi sẽ vẫn không bị thay đổi vì nếu có sự thay đổi chương trình biên dịch sẽ bắt lỗi.

5.5.5 Hàm trả về pointer và mảng

Cả hai trường hợp hàm trả về một mảng hay pointer, ta cũng chỉ định nghĩa như sau:

<Tên kiểu dữ liệu> * <Tên hàm>(<danh sách tham số>)

Trong đó, tên kiểu xác định kiểu của biến mà pointer được trả về. Kiểu này có thể là một kiểu dữ liệu nào đó đã định nghĩa trước.

Điều quan trọng ở đây là mảng được trả về (hoặc biến mà pointer trả về này chỉ đến) phải có "thời gian tồn tại" cho đến lúc ra khỏi hàm này. Vì nếu đối tượng của một pointer không còn tồn tại thì việc trả về bản thân pointer không g còn ý nghĩa gì cả.

Ví dụ 5.25: Viết một hàm nhận một mảng *so* từ bàn phím rồi trả về mảng đó :

```
int*input()
{
        int daySo[10];
        printf("hay nhap vao 10 so :\n");
        for(i = 0; i < 10;i++)
        {
             printf("So thu %d",i);
             scanf("%d",& daySo [i]);
    }
}
```

Trong hàm trên, sẽ không sử dụng được vì có thể địa chỉ của mảng *daySo* vẫn được trả về nhưng đối tượng của địa chỉ đó thì không còn ý nghĩa do "thời gian tồn tại" của mảng *daySo* này chỉ là ở trong hàm input(). Vì vậy, muốn sử dụng hàm này thì mảng *daySo* phải là biến ngoài hàm input, mảng static (mảng tĩnh), hoặc l mảng của hàm khác gửi đến cho hàm input(). Chẳng hạn, ta có thể sửa lại hàm input() như sau:

```
int * input(int* daySo) //hay int daySo[]
{
        printf("hay nhap vao 10 so :\n");
        for(i = 0; i < 10;i++)
        {
             printf("So thu %d",i);
```

```
            scanf("%d",& daySo[i]);
        }

        return (daySo);

    }
```

5.5.6 Mảng các con trỏ hoặc con trỏ của con trỏ (pointer của pointer)

Khái niệm:

Cú pháp:	**<tên kiểu dữ liệu >* <tênmảng> [kích thước];**

Ví dụ 5.26:

```
//khai báo mảng a gồm 10 pointer chỉ đến kiểu char
char   *a[10];
```

Ví dụ 5.27: Ta xét đoạn code sau:

```
// Hàm sắp xếp các phần  tử
void   Sort(int *a[], int n)
{   int i, j, *tmp;
    for(i = 0; i< n –1; i++)
       for(j = i +1; j < n; j++)
            if(*a[i] > *a[j])
            {
                 tmp = a[i];
                 a[i] = a[j];
                 a[j] = tmp;
            }
}
void main()
{
    int d = 10, e = 3, f = 7;
    int a = 12, b = 2, c = 6;
    int * ma[6];
    ma[0] = &a;       ma[3] = &d;
    ma[1] = &b;       ma[4] = &e;
    ma[2] = &c;       ma[5] = &f;
    Sort(ma,6); // sắp xếp lại thứ tự mảng
    sfor(i = 0; i < 6; i++)
```

```
            printf("%d\n",*ma[i]);
    }
```

Hàm Sort sắp xếp lại các địa chỉ trong mảng **a** sao cho các địa chỉ sẽ chỉ đến số bé sẽ được sắp trước các địa chỉ được chỉ đến các số lớn hơn.

Nếu các phần tử trong một mảng các pointer lại được gán địa chỉ của các mảng khác thì ta sẽ được một mảng của các mảng nhưng không giống như mảng 2 chiều. Vì trong mảng 2 chiều các phần tử năm liên tục nhau trong bộ nhớ, nhưng với mảng con trỏ thì các mảng con nằm ở vị trí bất kì. Ta chỉ cần lưu trữ địa chỉ của chúng nên việc sắp xếp lại các địa chỉ của chúng trong mảng các pointer của chúng mà thôi.

Như vậy, qua ví dụ trên ta thấy rằng việc sử dụng mảng các pointer có các ý niệm gần giống như việc sử dụng mảng hai chiều.

Ví dụ 5.27: Nếu chúng ta khai báo

```
int m[10][9];   int *n[10];
```

thì cách viết để truy xuất của các mảng này có thể tương tự nhau, chẳng hạn:

m[6][5] và n[6][5] đều cho ta một kết quả là một số int.

Tổng kết, mảng 2 chiều và mảng các pointer có sự khác nhau cơ bản sau:

- Mảng 2 chiều thực sự là một mảng có khai báo, do đó có chỗ đầy đủ cho tất cả các phần tử của nó.
- Mảng các **pointer** chỉ mới có chỗ cho các **pointer** mà thôi. Vì vậy, ta cần phải xin cấp phát các vùng nhớ để các **pointer** này trỏ đến.
- Như vậy mảng các **pointer** có thể được xem là tốn chỗ hơn là mảng 2 chiều, vì vừa phải lưu trữ các **pointer** và vừa phải có chỗ cho mỗi phần tử sử dụng.
- Mảng **pointer** có các lợi điểm:
  + Việc truy xuất đến các phần tử là truy xuất gián tiếp qua các **pointer**. Vị trí của các mảng con này có thể l bất kì, và chúng có thể là những mảng

đã có bằng cách xin cấp động (**malloc**) hay bằng khai báo mảng bình thường.

+ Các mảng con của nó được chỉ đến bởi các **pointer**, có thể có độ dài tùy ý, hay có thể không có.

+ Đối với các mảng **pointer**, ta có thể hoán chuyển thứ tự của các mảng con được chỉ đến bởi các **pointer** này, bằng cách chỉ hoán chuyển bản thân các **pointer** trong mảng **pointer** là đủ.

Ví dụ 5.28: Viết hàm nhập vào n là số tháng trong năm, sau khi thực thi hàm trả về chuỗi tên tháng tương ứng. Ta dùng mảng các con trỏ ký tự để lưu trữ giá trị chuỗi tên tháng như sau:

```
//n là số của tháng trong năm
char* chuyenTenThang (int n)
{
    static char*tenThang[12]= { "January","February",
        "Match","April","May", June","July","August",
        "September","October","November","December"
                                };
    if(n < 1 || n > 12)    return NULL;
    return (tenThang[n -1]);
}
```

Trong hàm trên, chúng ta đã khởi tạo cho mảng các pointer trỏ đến kiểu *char* là *tenThang* được khai báo *static* bằng các chuỗi hằng. Giá trị trả về của hàm là pointer chỉ đến một chuỗi ứng với giá trị n tương ứng hoặc trả về một pointer NULL nếu tháng không đúng.

Pointer chỉ đến pointer

- Chúng ta có thể khai báo một pointer chỉ đến một pointer trỏ đến một biến có kiểu dữ liệu là kiểu như sau: **kiểu **tenpointer;**

Ví dụ 5.29 :

```
int **pp;
```

Với cách khai báo này, chúng ta có chỗ cho biến pointer của pointer là pp, và được ghi nhận rằng biến của biến pp này chỉ đến là một pointer chỉ đến một biến

int. Chúng ta một lần nữa phải cần nhớ rằng thực sự biến pp cho đến lúc này vẫn chưa có một đối tượng để trỏ đến, và chúng ta phải tự mình gán địa chỉ của một pointer nào đó cho nó.

Ví dụ 5.30:

```
char*monthname[20] =
          {“January”,”February”, ”Match”,”April”,
          “May”,”June”,”July”,”August”,”September”,
          “October”,”November”,”December” };
          char**pp ;
pp = monthname;//tên mảng là địa chỉ của phần tử đầu
     tiên.
```

Lúc đó, *pp sẽ chỉ đến pointer đầu tiên của mảng, pointer này lại chỉ đến chuỗi “January”.

Nếu tăng pp lên pp++ thì sẽ chỉ đến pointer kế tiếp của mảng, pointer này chỉ đên chuỗi “February”. …

5.6 Chuỗi kí tự

5.6.1 Chuỗi kí tự

Trong ngôn ngữ lập trình C không có kiểu dữ liệu chuỗi mà chuỗi trong C là một dãy các kí tự kiểu char. Một chuỗi trong C được đánh dấu kết thúc là ‘\0’ (còn gọi là NULL trong bảng mã Ascii) và có độ dài tùy ý, điều này cũng có nghĩa chuỗi ký tự trong C là một mảng các ký tự char.

Chúng ta có thể gán một chuỗi cho một biến pointer chỉ đến char

Ví dụ 5.30:

```
char str[20]= “ \nHappy New Year”
```

Không thể cộng, trừ, nhân, chia 2 chuỗi kí tự lại bằng phép toán đơn thuần. Tất cả những điều đó phải được làm bằng các hàm riêng lẽ. Ta có thể gán một chuỗi này bằng một chuỗi khác (strcpy), so sánh 2 chuỗi kí tự với nhau theo thứ tự từ điển (strcmp), cộng 2 chuỗi với nhau (strcat),...

Mọi hằng chuỗi đều được ngôn ngữ lập trình C lưu trữ như là một mảng các **char** và kết thúc bằng kí tự '\0'. Hơn nữa, một chuỗi trong chương trình chúng ta chỉ nhận được địa chỉ và chỉ đến đầu mảng lưu trữ. Việc truy xuất đến một hằng chuỗi đều được thực hiện qua một pointer chỉ đến mảng đó.

Ví dụ 5.31: `printf("Happy new year\n");`

thì hàm printf() thực sự cũng chỉ ghi nhận được pointer chỉ đến mảng kí tự này đang lưu trữ đến một chỗ nào đó mà thôi. Vì vậy, chúng ta có thể gán một hằng chuỗi cho một biến pointer chỉ đến char.

Ví dụ 5.32:

```
char*str;
str = "Happy new year \n";
```

Lúc này, ta đã đưa pointer str giữ địa chỉ của chuỗi kí tự này. Ta có thể quy định rằng một mảng kí tự tận cùng bằng kí tự '\0' được gọi là một chuỗi.

5.6.2 Một số hàm thao tác trên chuỗi

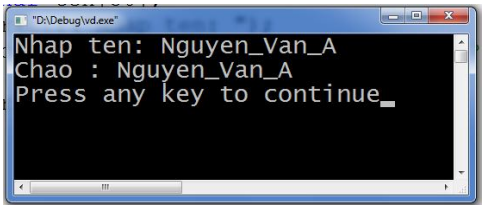
Hàm nhập xuất một chuỗi

Ta có thể dùng hàm scanf() với định dạng %s để nhập chuỗi như ví dụ sau:

Ví dụ 5.33:

```
#include<stdio.h>
void main()
{
    char ten[50];
    printf("Nhap ten: ");
    scanf("%s",ten); /* Không có chỉ thị & vì  ten
chuỗi đã làl một địa chỉ*/
    printf("Chao : %s\n",ten);
}
```

Kết quả thực hiện chương trình:



Lưu ý :

Nếu dùng hàm scanf() để nhập dữ liệu và kết thúc việc nhập dữ liệu bằng phím Enter, thì lúc này phím Enter sẽ cho hai kí tự có m ASCII là 13 và 10 trong vùng đệm. Như vậy nếu dùng hàm scanf () thì kí tự có m ASCII l 10 vẫn còn nằm trong vùng đệm. Nếu ta dùng hàm gets(chuỗi s), kí tự có mà ASCII là 10 được chuyển ngay vào chuỗi s. Tức là hàm gets sẽ lấy tất cả các ký tự trong buffer(vùng đệm) của màn hình vô chuỗi cho nên đôi khi chúng ta sẽ nhận được chuỗi không mong muốn do gets nhận những ký tự dư của các hàm nhập khác. Để tránh điều này ta dùng hàm int flushall(void) để xóa mọi buffer(vùng đệm) hoặc hàm fflush(stdin) để xóa vùng đệm bàn phím trước hàm nhập chuỗi gets(chuỗi s).

Ta viết lại ví dụ trên như sau:

Ví dụ 5.34:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  char ten[30];
  printf(“Nhap ten: ”);
  flushall(); //hoặc dùng hàm fflush(studin);
  gets(ten);
  printf(“Chao :”);
  puts(ten);
  getch();
}
```

Kết quả thực thi của chương trình:

Nhập chuỗi kết thúc bằng phím Enter : **char*gets(char*s);**

Xuất một chuỗi có xuống dòng sau khi xuất:

```
int  put(const char*s);
int printf("%s\n",s);
```

Xuất một chuỗi không xuống dòng sau khi xuất :

```
int printf("%s",s);
int printf(s);
```

Một số hàm cơ bản thường dùng cho mảng chuỗi trong ngôn ngữ lập trình C.

- *Gán một chuỗi này bằng một chuỗi khác*

Thực chất của việc "gán" này là việc gán từng từng phần tử của chuỗi này vào chuỗi kia. Để làm việc này ta dùng hàm strcpy().

Hàm strcpy() sẽ gán chuỗi source vào chuỗi dest từng phần tử cho đến khi copy kí tự kết thúc chuỗi '\0'.

char*strcpy(char*dest, const char*source);

- *So sánh hai chuỗi kí tự với nhau theo thứ tự từ điển*

Việc so sánh được thực hiện bằng cách so sánh từng cặp phần tử của hai chuỗi với nhau. Nếu chúng hoàn toàn giống nhau cho đến kí tự kết thúc thì xem như hai chuỗi là bằng nhau.

```
int strcmp(const char*s1, const char*s2); //phân biệt chữ in và chữ thường

int strcmpi(const char*s1, const char*s2);// không phân biệt chữ in và chữ thường

int stricmp(const char*s1, const char*s2);// không phân biệt chữ in và chữ thường
```

Nếu chuỗi s1 nhỏ hơn chuỗi s2 thì sẽ trả về một số m.

Nếu chuỗi s1 lớn hơn chuỗi s2 thì sẽ trả về một số dương.

Nếu hai chuỗi bằng nhau sẽ trả về số không (0).

- *Cộng hai chuỗi với nhau*

  Hàm chép chuỗi source vào cuối chuỗi dest.

```
char* strcat(char*dest, char*source);
```

- *Tìm một kí tự nào đó trong một chuỗi cho trước*

```
char*strchr(char*s,char ch);
```

- *Tìm độ dài của chuỗi*

```
int strlen(const char*s);
```

Bài tập chương 5

1. Khai báo mảng số nguyên chứa 100 phần tử.
2. Biểu thức **array**[2][4] có nghĩa là gì?
3. Định nghĩa cấu trúc để tính diện tích hình vuông.
4. Xét ví dụ sau phát biểu sau đây:

```
void main()
{    int a=2,b=3,*pnt
     pnt=&a
     b=*pnt
 }
```

   Tính giá trị của b.

5. Xét mảng 2 chiều sau:

10	3	2
40	56	1

30	45	89
28	67	100

Cho biết chỉ số của các phần tử của mảng trong các trường hợp giá trị các phần tử như sau: 10, 56, 89, 28, 100.

Viết chương trình (mã giả) để đảo ngược vị trí các phần tử trong mảng một chiều.

6. Viết chương trình (mã giả) để nhập dữ liệu cho một mảng 2 chiều vuông và xuất ra chuyển vị của nó.

7. Viết chương trình để nhập ma trận 2x3, và trừ tất cả các phần tử cho 2

8. Viết hàm nhập một mảng a gồm n số nguyên.

9. Viết hàm xuất mảng một chiều gồm n phần tử.

10. Nhập mảng a gồm n phần tử sao cho các số chẵn và lẻ xen kẽ nhau.

11. Nhập mảng a gồm n phần tử sao cho mọi phần tử lặp lại không quá 2 lần.

12. Nhập mảng a gồm n phần tử số nguyên sao cho các số dương có thứ tự tăng.

13. Xây dựng hàm nhập mảng a gồm n phần tử thỏa:

- Không chứa số âm.
- Có nhiều nhất là 3 phần tử có giá trị 0.
- Khoảng cách giữa 2 phần tử bất kỳ không quá 4.

14. Xây dựng hàm nhập mảng a gồm n phần tử số nguyên phân biệt thỏa:

- Không chứa số nguyên tố lớn hơn 200.
- Các số không nguyên tố có thứ tự giảm.

15. Viết hàm tìm phần tử lớn nhất trong mảng số nguyên n phần tử.

16. Viết hàm tìm phần tử nhỏ nhất trong mảng số nguyên n phần tử.

17. Viết hàm tính tổng các phần tử của mảng gồm n phần tử.

18. Tính tổng các phần tử của mảng a số nguyên có n phần tử.

19. Tính giá trị trung bình của các phần tử trong mảng số nguyên n phần tử.

20. Sắp xếp mảng a có n phần tử theo thứ tự tăng dần.

21. Tìm kiếm vị trí đầu tiên của x trong mảng a có n phần tử.

22. Xóa phần tử thứ i trong mảng a có n phần tử.

23. Chèn một phần tử x vô vị trí thứ i của mảng a.

24. Viết chương trình nhập một mảng số nguyên. Tính tổng các vị trí chẵn và tổng các vị trí lẻ.

25. Viết chương trình nhập một mảng các số nguyên. Xuất ra số nguyên âm nhỏ nhất và số nguyên dương lớn nhất.

26. Đếm các số không âm trong mảng a có n phần tử.

27. Đếm các số nguyên tố trong mảng a.

28. Đếm số lần xuất hiện của phần tử x trong mảng a.

29. Đếm các số phân biệt trong mảng a.

30. Tìm số nguyên tố lớn nhất trong mảng a.

31. Tìm số có bình phương nhỏ nhất trong mảng a.

32. Tìm số có số lần xuất hiện nhiều nhất trong mảng a.

33. Tạo mảng b chứa tất cả các số dương của mảng a.

34. Tạo mảng b chứa tất cả các phần tử của mảng a sao cho mỗi phần tử chỉ xuất hiện trong b đúng một lần.

35. Xóa tất cả các số nguyên tố trong mảng a.

36. Sắp thứ tự tăng các số dương và giữ cố định các số còn lại.

37. Sắp xếp mảng a sao cho:

    - Các số chẵn ở đầu mảng và có thứ tự tăng.
    - Các số lẻ ở cuối mảng v có thứ tự giảm.

38. Sắp xếp mảng a sao cho:

    - Số dương ở đầu mảng và có thứ tự tăng
    - Số âm ở giữa mảng và có thứ tự giảm
    - Số 0 ở cuối

39. Sắp xếp mảng sao cho các phần tử chẵn tăng, các phần tử còn lại cố định.

40. Sắp thứ tự tăng theo hai tiêu chuẩn:

    - Số lần xuất hiện.
    - Giá trị xuất hiện.

41. Viết chương trình nhập một ma trận nguyên tối đa 20 dòng 20 cột và xuất ra ma trận này.

42. Tìm phần tử lớn nhất của ma trận.

43. Viết chương trình tính tổng dòng thứ i v cột thứ j của ma trận số nguyên m × n.

44. Nhập vào một ma trận các số nguyên và cho biết ma trận này có đối xứng qua đường chéo chính hay không?

45. Viết chương trình nhập vào một ma trận các số nguyên. Sắp xếp ma trận tăng dần theo chiều xoắn ốc.

46. Tìm giá trị lớn nhất trên từng dòng của ma trận.

47. Tìm giá trị lớn nhất trên từng cột của ma trận.

48. Tìm phần tử trong ma trận gần với x nhất.

49. Viết chương trình nhập một ma trận vuông cấp n. Tính tổng tam giác trên, tổng tam giác dưới (kể cả đường chéo).

50. Viết chương trình tìm dòng có tổng lớn nhất trong các dòng và cột có tổng lớn nhất trong các cột của ma trận.

51. Viết chương trình nhập hai ma trận $A_{m \times k}$, $B_{k \times n}$ và tính tích hai ma trận A*B.

52. Viết chương trình nhập vào ma trận các số nguyên, xuất ra ma trận chuyển vị của ma trận đó.

53. Chuyển ma trận a($_{M \times N}$) về mảng một chiều b theo dòng.

54. Chuyển mảng một chiều b sang ma trận a($_{M \times N}$) theo dòng.

55. Chuyển ma trận a($_{M \times N}$) về mảng một chiều b theo cột.

56. Chuyển mảng một chiều b sang ma trận a($_{M \times N}$) theo cột.

57. Tạo ma trận b có cùng kích thước với ma trận a sao cho b[i][j] = tổng các phần tử lớn nhất dòng i và nhỏ nhất cột j.

- Tạo mảng Max chứa các giá trị lớn nhất trên từng dòng.
- Tạo mảng Min chứa các giá trị nhỏ nhất trên từng cột.
- Tạo ma trận mới b có cùng kích thước với a.

58. Tạo ma trận b là ma trận chuyển vị của ma trận a.

59. Tạo ma trận b bằng cách xóa các dòng có chứa số 0 của a.

60. Sắp xếp ma trận a tăng theo cột/ tăng theo dòng.

61. Tìm phần tử lớn nhất dòng i và phần tử nhỏ nhất cột j.

62. Viết hàm xóa các khoảng trắng ở đầu chuỗi, ở cuối chuỗi.

63. Không dòng hàm strlwr() hãy viết chương trình đổi chuỗi ra thành chuỗi thường.

64. Không dòng hàm strupr() hãy viết chương trình đổi chuỗi ra thành chuỗi hoa.

65. Viết hàm đếm số từ trong một chuỗi.

66. Viết chương trình chèn chuỗi S2 vào vị trí thứ i của chuỗi S1.

67. Viết hàm tìm số lần xuất hiện của chuỗi S2 trong chuỗi S1.

68. Viết hàm kiểm tra một chuỗi S có đối xứng hay không.

CHƯƠNG 6. KIỂU DỮ LIỆU CẤU TRÚC

(STRUCT)

6.1 Kiểu struct

6.1.1 Giới thiệu

Đối với một vấn đề cần mô tả, tùy góc nhìn, tùy vào mức độ quan tâm của mỗi người mà có các mô tả khác nhau.

Ví dụ 6.1: Mô tả một nhân viên trong cơ quan:

- **Đối với người kế toán:** một nhân viên được mô tả bằng: mã số, họ, tên lót, tên, ở phòng ban nào, chức vụ gì…

- **Đối với người làm công tác tổ chức:** một nhân viên được mô tả bằng: mã số, họ, tên lót, tên, ngày tháng năm sinh, nơi sinh, địa chỉ, ở phòng ban nào, chức vụ, giới tính, đã có gia đình chưa, lương, …

Ví dụ 6.2: Mô tả một sinh viên.

- **Đối với phòng giáo vụ:** một sinh viên được mô tả bằng: mã số, họ, tên lót, tên, năm sinh, địa chỉ, cha, mẹ, năm nhập học, điểm trúng tuyển, các điểm cho từng môn học…

- **Đối với một giáo viên đang dạy một môn học:** một sinh viên được mô tả bằng: mã số, họ, tên lót, tên, điểm môn học mình dạy…

Cấu trúc (struct) là kiểu dữ liệu phức hợp được xây dựng từ những kiểu dữ liệu khác. Các kiểu dữ liệu bên trong một cấu trúc này có thể là một kiểu cấu trúc khác.

6.1.2 Định nghĩa

- **Kiểu 1**

```
struct     <tên cấu trúc>
{
        <tên kiểu dữ liệu> <tên thành phần 1>;
        <tên kiểu dữ liệu> <tên thành phần 2>;
```

```
    …
};
```

Ví dụ 6.3 : Một nhân viên được mô tả bằng 3 thành phần thông tin, còn gọi là trường (field): mã số (int), họ tên (tối đa 29 ký tự), lương (float) có thể khai báo như sau:

```
struct Tenhanvien// struct khai báo một cấu trúc

{

    int maso;

    char hoten[30]; //chừa 1 ký tự cuối chứa '\0'

    float luong;

}; //để kết thúc một khai báo cấu trúc.
```

Ví dụ 6.4: Một ngày được mô tử bằng 4 thành phần thông tin: ngày (day), tháng (month), năm (year) và thứ của ngày trong tuần (chuỗi ngày).

```
struct  date  // mô tả ngày tháng năm
{
  int day ;
     int month;
  int year;
  char weekdays [4];//Mon, Tue, Wed
};
```

Ví dụ 6.5

Màn hình máy tính sử dụng hệ tọa độ nguyên, một ký tự xuất hiện trên màn hình có thể được mô tả như sau:

```
struct SCR_CHAR    // screen character
{
    char c; //ký tự gì
    short x,y; // được xuất ở điểm nào trên màn hình
  };
```

a. Kiểu 2

```
typedef  struct  <tên cấu trúc>
{
        <tên kiểu dữ liệu> <tên thành phần 1>;
        <tên kiểu dữ liệu> <tên thành phần 2>;
   …
}<tên cấu trúc mới>;
```

Ví dụ 6.6: Kiểu nhân viên có thể được khai báo như sau :

```
typedef  struct Tenhanvien
{
int maso;
        char ten[30];
        float luong;
}TENNV;
```

Sau câu lệnh trên chúng ta có kiểu dữ liệu Nhân viên có 2 tên gọi là **Tenhanvien** và **TENV.**

Nếu ta chỉ khai báo struct, khi cần dùng kiểu dữ liệu đó, chúng ta phải dùng tên kèm theo struct phía trước tên. Dùng typedef để định nghĩa kiểu struct thì khi cần kiểu dữ liệu mới này không cần phải đánh lại chữ struct ở đầu, mà có thể dùng tên gọi mới sau khai báo struct.

Chẳng hạn để khai báo một biến thuộc kiểu Nhân viên, ta có 2 cách ghi:

```
struct Tennhanvien a;
```
hoặc
```
TENNV a;
```

6.1.3 Khai báo

Khai báo một cấu trúc như trên chỉ là khai báo thêm một kiểu dữ liệu mới cho chương trình thay vì dùng các kiểu dữ liệu có sẵn như **int, float**... Để có được một biến cấu trúc ta phải khai báo biến với kiểu là cấu trúc đó.

Cú pháp : <Tên cấu trúc> <tên biến> ;

Ví dụ 6.7:

```
TENNV x; // Khai báo biến x kiểu Nhân viên:

TENNV dsNhanvien[100]; //khai báo mảng 1 chiều
dsNhanvien chứa 100 nhân viên.
```

Nếu định nghĩa cấu trúc **kiểu 1** ta có thể khai báo biến cấu trúc ngay khi định nghĩa cấu trúc (không áp dụng được nếu định nghĩa cấu trúc kiểu 2).

Ví dụ 6.8:

Khai báo biến cấu trúc Ngay kiểu date kết hợp khi khai báo cấu trúc:

```
struct  date
{   int day ;
   int month;
  int year;
  char weekdays [4];
} Ngay
```

Khai báo biến Ngay1 kiểu cấu trúc date

```
struct date Ngay1;
```

Ví dụ 6.9:

```
typedef struct Tenhanvien
{       int maso;
        char ten[30];
        float luong;
}NV;
NV nv1, nv2; //khai báohai biến cấu trúc nv1,nv2 kiểu Tênhanvien
```

6.1.4 Cấu trúc lồng nhau

Cấu trúc lồng nhau là một cấu trúc có thành phần lại là một cấu trúc.

Ví dụ 6.10:

```
typedef struct DATE
{
  int d, m, y; // date, month, year
```

```
}Date;
struct Tênhanvien
{
  int maso;
  char ten[30];
  Date ngaysinh; // file ngày sinh có kiểu struct là
// DATE
  float luong;
}nv1, nv2;  // định nghĩa 2 biến nv1 và nv2.
```

6.1.5 Khởi tạo cấu trúc

Việc khởi tạo biến cấu trúc tương tự như khởi động một mảng. Ta khởi tạo 2 biến cấu trúc nv1 và nv2 như sau:

```
struct Tênhanvien nv1={101, "TRAN HUNG DUNG", 1250000};
hay  Tênhanvien nv2={106, "NGUYEN HOANG ANH THU", 16750000};
```

Khởi tạo biến cấu trúc là ấn định giá trị hằng cho lần lượt các field theo thứ tự đã khai báo, cách nhau bằng dấu phẩy, bao tất cả lại trong cặp {} kết thúc bằng dấu ";"

Khởi tạo một biến cấu trúc Ngay có kiểu date:

```
date Ngay  = { 2,9,1989,"Sat"};
```

6.1.6 Truy xuất các thành phần của một biến cấu trúc

Để truy xuất một thành phần của biến cấu trúc, ngôn ngữ C có các phép toán lấy thành phần như sau:

- Nếu biến cấu trúc là biến thông thường (không phải là con trỏ) thì truy xuất thành cấu trúc bằng dấu chấm **"."**

<Tên biến cấu trúc> . <Tên thành phần>

- Nếu biến cấu trúc là biến con trỏ thì truy xuất thành phần bằng dấu mũi tên " ->" (dấu trừ và dấu lớn).

<Tên biến con trỏ cấu trúc> -> <Tên thành phần>

Ví dụ 6.11:

```
   1) Date Ngay;
Ngay.day = 2; // gán giá trị 2 cho thành phần day của biến
cấu trúc Ngay
   2) struct Tênhanvien
{
int maso;
          char ten[30];
          float luong;
} nv1,nv2;
nv1.maso; // để truy xuất  maso của nhân viên
nv1.ten;  // để truy xuất ten của nhân viên
nv1.luong;     // để truy xuất lương của nhân viên
   3) Date *d;
d->day = 2;// gán giá trị 2 cho thành phần day của biến con
trỏ cấu trúc d
```

6.2 Mảng các struct

Khai báo mảng các cấu trúc hoàn toàn tương tự như khai báo mảng khác, chỉ có điều ở phần kiểu sẽ là tên một cấu trúc.

Cú pháp : struct <tên cấu trúc> <tên mảng> [<kích thước>];

Ví dụ 6.12 :

```
   4) struct date  d[10];
```

Khi đó việc truy xuất đến một phần tử của mảng như d[2] ta sẽ thu được một biến có cấu trúc date và có thể lại tiếp tục truy xuất đến các thành phần của nó.

Ví dụ 6.13:

```
 d[2].month =10;

 strcpy ( d[2]. weekday,"SUN");
```

6.3 Pointer đến một struct

Ta có thể khai báo một biến pointer chỉ đến một cấu trúc để có thể lưu giữ lại địa chỉ của một biến cấu trúc nào đó cần thiết.

Cú pháp: struct <tên cấu trúc> * <tên pointer>;

Ví dụ 6.14:

```
struct date  *p ;
```

Việc sử dụng pointer chỉ đến cấu trúc thường được dùng để gửi cấu trúc đến cho một hàm.

Việc truy xuất đến thành phần của một cấu trúc thông qua một pointer được thực hiện bằng một tóan tử: -> là phép toán lấy thành phần nội dung của pointer (đã được đề cập ở phần 6.1.6).

Ví dụ 6.15:

```
printf("\n Ngay luu tru la : %d ", p -> day);
printf("\n Thang luu tru la : %d ", p -> weekday);
```

6.4 Cấu trúc đệ quy

Người ta thường dùng cấu trúc đệ quy để chỉ các cấu trúc mà thành phần của nó lại có các pointer chỉ đến một biến cấu trúc cùng kiểu.

Ví dụ 6.16:

```
struct node
{
  int num ;
  struct node  *pNext ;
};
```

Hoặc ta có thể có một cấu trúc như sau:

```
struct pnode

{ int key ;

  struct pnode  * left ;

  struct pnode  * right ;

};
```

đây được xem là một cấu trúc đệ quy .

Bài tập chương 6

1. Hãy khai báo một cấu trúc mô tả một điểm trên tọa độ xOy. Sau đó viết chương trình thực hiện các chức năng sau:

- Nhập/xuất điểm
- Kiểm tra điểm có nằm trên trục tung/trục hoành.
- Tạo danh sách chứa các điểm trong Oxy.
- Nhập/xuất danh sách điểm.
- Thêm, xóa điểm trong danh sách.
- Tìm kiếm điểm trong danh sách điểm

2. Hãy khai báo một cấu trúc mô tả hình chữ nhật có các thông tin: điểm góc trên trái, chiều dài và chiều rộng. Sau đó viết chương trình thực hiện các chức năng sau:

- Nhập/xuất thông tin 1 hình chữ nhật
- Kiểm tra hình chữ nhật có phải hình vuông không?
- Tính chu vi, diện tích hình chữ nhật.

3. Hãy khai báo một cấu trúc mô tả hình tròn có các thông tin: điểm O làm tâm và bán kính r. Sau đó viết chương trình thực hiện các chức năng sau:

- Nhập/xuất thông tin 1 hình tròn
- Tính chu vi, diện tích hình chữ nhật.

4. Hãy khai báo một cấu trúc mô tả phân số có các thông tin tử số và mẫu số. Sau đó viết chương trình thực hiện các chức năng sau:

- nhập/ xuất phân số.
- kiểm tra phân số mẫu phải khác 0.
- Tính cộng/trừ/nhân/chia 2 phân số.
- Tối giản phân số.

5. Một lớp học có tối đa 50 học sinh, mỗi học sinh được mô tả bằng các thông tin: mã số(int), họ và tên, phái, điểm học kỳ I, điểm học kỳ II. Hãy viết chương trình quản lý lớp học này với các thao tác sau:

- Nhập danh sách lớp.
- In ra danh sách lớp theo thứ tự mã số (gồm số thứ tự, họ và tên).
- In ra danh sách lớp theo thứ tự của họ và tên.
- Tìm vị trí của một học sinh theo khi nhập họ và tên.
- In ra danh sách các học sinh có điểm trung bình của năm học < 5.0
- Thêm 1 học sinh vào danh sách
- Xóa 1 học sinh trong danh sách.

6. Viết chương trình quản lý nhân sự cho một công ty, mỗi nhân viên trong công ty gồm các thông tin sau: mã số (không có hai người trùng mã số), họ, tên, ngày sinh, nơi sinh, địa chỉ, ngày công tác, lương. Viết chương trình quản lý nhân viên với các thao tác sau:

- Thêm vào một nhân viên.
- Xem danh sách nhân viên.
- Tìm nhân viên theo mã số.
- Tìm một nhân viên theo tên.
- In ra bảng lương của các nhân viên trong công ty theo thứ tự giảm dần.
- Xóa một nhân viên.

CHƯƠNG 7. FILE DỮ LIỆU

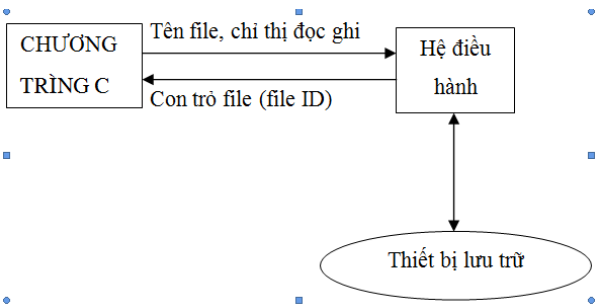
7.1 Giới thiệu về file

7.1.1 Giới thiệu

Như chúng ta đã biết, máy tính là công cụ giúp con người lưu trữ và xử lý thông tin. Hầu hết các chương trình đều cần phải lưu trữ dữ liệu sau khi xử lý chính vì vậy C cung cấp cho chúng ta các kỹ thuật xử lý lưu trữ trên file.

7.1.2 Khái niệm File

- Là một đơn vị lưu trữ logic.
- Được biểu thị bằng một tên.
- Bao gồm một tập hợp dữ liệu do người tạo xác định.
- Được lưu trữ trên thiết bị lưu trữ phụ bằng cách ánh xạ lên đơn vị vật lý của thiết bị.
- Được C hỗ trợ các thao tác truy xuất.
  - Tạo mới.
  - Đọc, ghi phần tử.
  - Xóa.
  - Đổi tên.



Mỗi khi hệ điều hành mở một file, hệ điều hành thao tác với đĩa, truy xuất thông tin cơ bản của file, rồi trả về địa chỉ vùng lưu trữ gọi là handle của file – file ID – để nhận dạng duy nhất cho file này. Chương trình của chúng ta mỗi khi thao tác phải thông qua biến pointer đó. Để lấy pointer của file, chúng ta phải khai báo biến file, ngôn ngữ lập trình C dùng khai báo biến FILE * f, để lấy handle bằng lệnh mở file.

7.1.3 Cách thao tác với file:

Thao tác chuẩn: Người lập trình không cần biết quá trình thực hiện việc thao tác với file như thế nào. Đó là việc của hệ thống.

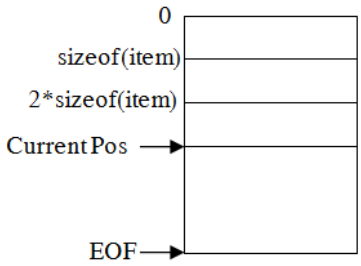
Thao tác mức hệ thống (thao tác thủ công): thao tác file thông qua bộ đệm (buffer - một vùng nhớ). Người lập trình phải tự quản lý các bộ đệm đọc ghi file. Thao tác file này gần giống với cách thao tác file của hệ điều hành MS – DOS. Thông thường chỉ có những người lập trình hệ thống mới sử dụng thao tác file mức hệ thống.

Thao tác với file phải thao tác với phần cứng. Do đó, việc thao tác với file có thể thành công hoặc thất bại.

7.1.4 Tổ chức lưu trữ dữ liệu trên file

Cách nhìn logic

Các phần tử được lưu trữ liên tục(danh sách đặc)



Cách nhìn logic việc lưu trữ dữ liệu trên file

Địa chỉ tuyến tính bắt đầu từ 0 (Zero – base address)

Truy xuất từng n phần tử, từng khối dữ liệu

Cách nhìn vật lý

- Dữ liệu được phân bố trên từng sector(đơn vị lưu trữ vật lý).
- Bao gồm một tập sector xác định.
- Các sector có thể lin tục hay rời nhau.
- Truy xuất từng sector.

Ngôn ngữ C xem file như là một dòng (stream) các byte, với các thiết bị xuất nhập theo từng byte cũng được xem là file, C định nghĩa sẵn các tên cho các thiết bị này và các file này đã được mặc định mở sẵn cho ta truy xuất ngay khi mở máy tính.

Handle	Tên	Thiết bị
0	**stdin**	Standard input – thiết bị nhập chuẩn – Bên phím
1	**stdout**	Standard output – Thiết bị xuất chuẩn – Màn hình
2	**stderr**	Standard error – Thiết bị xuất lỗi chuẩn – Màn hình
3	**stdaux**	Standard auxililary – Thiết bị ngoại vi chuẩn – Cổng nối tiếp
4	**stdprn**	Standard printer – Thiết bị in chuẩn – Máy in song song

Khi thao tác với file, ở mỗi thời điểm chỉ truy xuất được 1 phần tử lưu trữ trong file. Vị trí hiện hành đang thao tác (file position) gọi là chỉ số trong file hay con trỏ file – chính là số thứ tự của phần tử truy xuất hiện hành. Chỉ số thứ tự này bắt đầu từ 0.

7.2 Định nghĩa biến file và các thao tác mở/đóng file

Như chúng ta đã biết, dữ liệu được mã hóa thành dạng nhị phân. Như vậy, với một ký tự lưu trữ ta xem nó như một byte hay là một tập các giá trị số nhị phân.

Khi mở một file, vấn đề quan trọng là chúng ta phải chỉ định cách nhìn của chúng ta về các byte lưu trữ dữ liệu. Nếu chúng ta xem mỗi byte là mã ASCII của

kí tự, ta chỉ định mở file dạng văn bản. Nếu chúng ta xem mỗi byte là một số nhị phân, việc xử lý byte này l số hay chữ sẽ do chương trình giải quyết, thì ta chỉ định mở file dạng nhị phân.

7.2.1 Định nghĩa biến file trong C

Thao tác file chuẩn **FILE * f;**

Thao tác mức hệ thống **int f;**

Dữ liệu trên file là một dãy các byte(8 bit) có giá trị từ 0 đến 255. Số byte của dãy là kích thước thật của file (size on disk).

Có hai loại file

File văn bản thô (text)

- Dữ liệu là một chuỗi kí tự liên tục.
- Phân biệt các kí tự điều khiển.
- Xử lý đặc biệt với các kí tự điều khiển.

File nhị phân (binary)

- Dữ liệu được xem như là một dãy byte liên tục.
- Không phân biệt các kí tự điều khiển.
- Chuyển đổi dữ liệu tùy thuộc vào biến lưu trữ khi đọc/ghi.

7.2.2 Hàm mở, đóng file chuẩn

Mở file:

```
FILE * fopen(const char * filename, const char * mode);
```

Sẽ mở file có tên là filename, dạng mở mode. Nếu mở được thì trả về 1 pointer có trị khác NULL, không mở được trả về trị NULL. Tham số mode là một chuỗi kí tự chỉ định dạng mở.

Các mode mở file thông dụng:

- "**r**" : Mở **file** để đọc (**read**).
- "**w**" : Tạo **file** mới để ghi (**write**). Nếu file này đã tồn tại trên đĩa thì bị ghi đè.
- "**a**" : Mở để ghi vào cuối file nếu file này đã tồn tại, nếu file này chưa có thì sẽ được tạo mới để ghi (**append**).
- "**r+**" : Mở file đã có để cập nhật (cả đọc lẫn ghi).
- "**w+**" :Mở file mới để được cập nhật (cả đọc lẫn ghi). Nếu file này đã có sẽ bị ghi đè
- "**a+**" : Mở để đọc và cập nhật (ghi) vào cuối file. Sẽ tạo mới nếu file này chưa có.

Ghi chú:

- Thêm kí tự "t" để mơ tả mở file dạng text mode (Thí dụ : "rt", "w+t",…).
- Thêm kí tự "b" để mô tả mở file dạng nhị phân (Thí dụ : "wb", "a + b",…).

Đóng file:

int fclose(FILE * f);

Nếu thành công trả giá trị 0, nếu thất bại trả về giá trị EOF (giá trị -1)

Kết thúc file

Sau khi tạo xong một file văn bản, đóng file này, byte mang giá trị 1Ah (26 của hệ 10 – tương đương với khi g một phím Ctrl + Z) sẽ tự động chèn vào cuối file để ấn định hết file.

Nói chung, file được quản lý bằng kích thước của file (số bytes). Khi đã đọc hết số byte có trong file, thì dấu kí hiệu EOF (end of file) được DOS thông báo cho chương trình. Dấu hiệu EOF l tên hằng C khai báo sẵn trong STDIO.H mang giá trị –1.

Như vậy, nếu một file dữ liệu (có cả số) được mở dạng văn bản, nếu trong giữa file mà có giá trị 1Ah thì quá trình đọc sẽ bị ngưng nửa chừng (hàm đọc file sẽ trả về giá trị –1 cho chương trình báo đã kết thc file).

Chỉ định file ở chế độ	Dữ liệu trong chương trình C	Được lưu trữ trên file
Văn bản	**'\n'** **EOF**	**CR** (13), **LF** (10) **1Ah**
Nhị phn	**'\n'** **'\r''\n'** **1Ah** (số)	**LF** **CR LF** **1Ah**

Hàm int eof(int handle); trong IO.H

Trả về 1 : Đã hết file

0 : Chưa hết file

Số nguyên khác : Mô tả lỗi

Sự khác biệt giữa mở file dạng text và dạng nhị phân

Kiểu file văn bản thường hay được dùng trong hệ điều hành UNIX, file thường hay gặp ở DOS. Do vậy, dòng Borland duy trì cả hai dạng để có sự tương hợp với cả hai hệ điều hành.

Dạng nhị phân: Lưu trữ giống như lưu trữ trong bộ nhớ trong (RAM, lưu trữ nhị phân, dữ liệu được xem như các số nhị phân).

Dạng văn bản: Lưu trữ dạng nhị phân là m ASCII của kí tự số (phân loại này nẩy sinh do cách lưu trữ số).

Ví dụ 7.1:

Ghi một vi phần tử lên file TEXT và file BIN:

```
FILE *fTXT, *fBIN;

fTXT = fopen("D:\\z\\TEST.TEXT","wt");

fBIN = fopen("D:\\z\\TEST.BIN","wt");
```

//Lần lượt ghi ‘A’, 26, 10, ‘B’ lên hai file

fput(‘A’,fTXT); fput(26,fTXT); fput(10,fTXT); fput(‘B’,fTXT);

fput(‘A’,fBIN); fput(26,fBIN); fput(10,fBIN); fput(‘B’,fBIN);

fclose(fTXT);

fclose(fBIN);

Kết quả:

File TEST. TEXT chứa chuỗi : 65 26 13 10 66 (5 bytes)

File TEST.BIN chứa chuỗi : 65 26 10 66 (5 bytes)

//Đọc dữ liệu trên file TEST. TEXT

```
fTXT = fopen(“D:\\z\\TEST. TEXT”,”rt”);
while(!feof(fTXT))
{
    char c = fget(fTXT);
    printf(“%c\t”,c);
}
```

Kết quả chỉ xuất ra được kí tự A vì khi gặp kí tự 26 my sẽ hiểu l kết thc file.

//Đọc dữ liệu trên file TEST.BIN

```
fBIN = fopen(“D:\\z\\TEST.BIN”,”rb”);
while(!feof(fBIN))
{
  char c = fget(fBIN);
  printf(“%c\t”,c);
}
```

Kết quả in ra đầy đủ bởi không quan tâm đến các giá trị

//Đọc dữ liệu trên file TEST.BIN

```
fBIN = fopen(“D:\\z\\TEST. TEXT”,”rb”);
while(!feof(fBIN))
```

```
{
  char c = fget(fBIN);
  printf("%c\t",c);
}
```

7.2.3 Thao tác nhập / xuất với file

Thao tác xuất nhập chuẩn trong stdio.h

Ta gọi :

```
FILE * f ;          : handle của file
char c;             : kí tự cần đọc/ghi
char * s;           : chuỗi kí tự cần đọc/ghi
int n;              : số kí tự cần đọc hay là số record
                    cần thao tác
struct { } Rec;     : biến cấu trúc
```

Đơn vị của file	Hàm nhập (đọc từ file ra biến)	Hàm xuất (ghi từ biến lên file)
Ký tự	**C = fgetc(f);** Đọc kí tự hiện hành trong file f ra c	**int fputc(c,f);** Ghi c vào vị trí hiện hành trong f
Chuỗi kí tự	**char * fgets(char*s,int n,f)** Đọc n byte trong f vào s, không đọc được trả về NULL	**fputs(char*s,f)** Thành công : trả về kí tự cuối cùng đã ghi. Có lỗi trả về EOF
Chuỗi định dạng	**fscanf (f,"chuỗi định dạng", &biến1, &biến, &biến,…)**	**fprintf (f,"chuỗi định dạng", biến1, biến, biến,…)**
Struct	**size_t fread (&Rec,sizeof(struct), n, f);** Đọc n record từ f đưa vào biến Rec Đọc được : Trả về số phần tử đọc được Không đọc được : trả về 0	**int fwrite (&Rec,sizeof(struct), n,f);** Ghi n record từ Rec lên file Ghi được trả về số phần tử đã được ghi

Thao tác xuất nhập mức hệ thống : Trong IO.H

int read (int f, void*buf, unsigned len); :Đọc lên byte từ file f đưa vào buffer buf. Giá trị trả về:

- Thành công trả về số nguyên dương cho biết số byte đã đọc được.
- Khi hết file : trả về giá trị 0 vì không được byte nào.

Ghi

int write(int f, void * buf, unsigned len); : Đọc lên byte từ file f đưa vào buffer buf. Giá trị trả về:

- Thành công trả về số nguyên cho biết số byte đã ghi được.
- Thất bại trả về –1

Truy xuất lỗi xuất nhập file

Có thể có nhiều trường hợp có lỗi khi mở file như hết khoảng trống trên đĩa, đĩa mềm dán chống ghi, đĩa hỏng, ổ đĩa hư, … Khi mở file có lỗi, hàm fopen() sẽ trả về giá trị NULL.

Ví dụ 7.2: xuất nội dung file "textfile.txt" ra màn hình. Mở file có kiểm tra, nếu không mở được, thì xuất thông báo, nếu mở được đọc từng kí tự vào biến ch xuất ra màn hình và đóng file lại.

```
# include<stdio.h>
# include<conio.h>
# include<stdlib.h>
void main()
{
  FILE*fptr;
  int  ch;
  if(fptr = fopen("textfile.txt","r") ==NULL
  {    printf("Không thể mở file\n");
       getch();
       exit(0);
  }
  while(ch =fgetc(fptr) != EOF)
  printf("%c",ch);
  fclose(fptr);  // Đóng file
}
```

Để xác định lỗi file hay không ta dùng các hàm:

int ferror(FILE *fptr); - Hàm này trả về giá trị 0 nếu không có lỗi, trả giá trị khác 0 nếu có lỗi.

void perror(const char *string); - sẽ xuất thông báo lỗi mà ta muốn thông báo lên màn hình

Quy trình xử lý File : gồm ba bước:

Bước 1: Mở file

- Xác định chế độ mở chính xác(text/binary)
- Kiểm tra lỗi

Bước 2: Truy xuất xử lý

- Áp dụng hợp lý các hàm truy xuất tùy theo chế độ mở
- Quản lý con trỏ chỉ vị trí
- Kiểm tra lỗi

Bước 3: Đóng file

- Đảm bảo tính toàn vẹn dữ liệu

Ví dụ về thao tác mở file như sau:

```
//Mở chế độ text
FILE *fTXT;
fTXT = fopen("D:\\Z\\TEST.TXT","wt"); // Mở để
    ghi/tạo mới
fTXT = fopen("D:\\Z\\TEST.TXT","rt"); // Mở để đọc
fTXT = fopen("D:\\Z\\TEST.TXT","r+t"); // Mở để
    đọc/ghi
//Mở chế độ binary
FILE *fBIN;
fBIN = fopen("D:\\Z\\TEST.BIN","wb"); // Mở để
    ghi/tạo mới
fBIN = fopen("D:\\Z\\TEST.BIN","rb"); // Mở để đọc
```

```
fBIN = fopen("D:\\Z\\TEST.BIN","r+b"); // Mở để
    đọc/ghi
fBIN = fopen("D:\\Z\\TEST.BIN","ab"); // Mở để ghi
    thêm (append) vào cuối
//Xử lý lỗi
if(fTXT == NULL)
{   printf("Loi mo file = %d. Noi dung =
    %s",errno,strerror(errno));
    return 0;
}
```

Lưu ý : Thao tác đọc /ghi không thể thực hiện liền nhau, cần phải có một thao tác fseek hay rewind ở giữa. Ta xét ví dụ sau:

```
FILE*fBIN;
fBIN = fopen("D:\\z\\TEST.BIN"."r+b");// Mở để update
int x;
while(fread(&x,sizeof(x),1,fBIN))
if(x == 0)
{  x++;
        fseek(fBIN,-sizeof(x),SEEK_CUR);
        fwrite(&x, sizeof(x),1, fBIN);
}
fclose(fBIN);
```

Kết quả: Chỉ update được 1 phần tử

Ví dụ về truy xuất dữ liệu ở chế độ text

```
    n = fscanf(fTXT,"%d,%f,%s",&Item,&fItem,szItem); // trả
về số phần tử đọc được
    if(n < 3 || n ==EOF)
    {
        printf("Không đọc được số phần tử cần thiết. M lỗi
= %d",errno);
        … // Các xử lý đặc biệt
    }
     n = fprintf(fTXT,"%d,%f,%s",&Item,&fItem,szItem);//trả
về số byte ghi được;          if(n                          <
sizeof(Item)+sizeof(fItem)+strlen(szItem) || )
    {
        printf("Ghi không thành công. M lỗi = %d",errno);
```

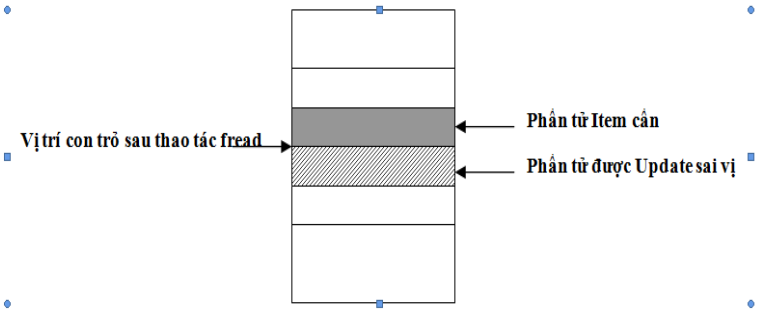
```
        … // Các xử lý đặc biệt
}
```

Ví dụ về truy xuất dữ liệu ở chế độ binary

```
n = fread(&Item, sizeof(Item), nItemRead, fBIN);//trả về
số phần tử đọc được
if(n < nItemRead)
{
    printf("Không đọc được số phần tử cần thiết. M lỗi =
%d",errno);
    … // Các xử lý đặc biệt
}
n = fwrite(&Item, sizeof(Item), nItemWrite, fBIN);//trả
về số byte đọc được
if(n < nItemWrite)
{
    printf("Ghi không thành công. M lỗi = %d",errno);
 … // Các xử lý đặc biệt
}
```

Ví dụ sau sẽ thực hiện việc cập nhật SAI một phần tử khi quản lý bằng con trỏ

```
fread(&Item,sizeof(Item),1,f);
Item ++; // cập nhật phần tử Item
fwrite(&Item,sizeof(Item),1,f);
```



Ví dụ về đóng file sau khi xử lý

Đóng từng file:

```
fclose(fTXT);

fclose(fBIN);
```

Đóng tất cả các file đang mở:

```
fcloseall();
```

Ví dụ sau sẽ cho thấy việc không bảo toàn dữ liệu khi không đóng file

```
char szFileName[] = "D:\\z\\TEST.BIN";
```

Ghi dữ liệu lên file

```
FILE*f = fopen(szFileName,"wb");

for( i = 0; i < n; i++)   fwrite(&i,sizeof(i),1,f);
```

Đọc dữ liệu từ file

```
i = 0;
f = fopen(szFileName,"rb");
while(fread(&i,sizeof(i),1,f)) i++;
printf("%d",i);
fclose(f);
```

Kết quả: (tùy thuộc vào hệ thống)

```
n = 100   → i = 0
n = 500   → i = 256
n = 513   → i = 512
n = 1000 → i = 768
```

Bài tập chương 7

1. Cho một tập tin chứa chi tiết của nhân viên như Mã NV, tên, tuổi, lương cơ bản và trợ cấp. Thực hiện các yêu cầu sau:

- Viết các câu lệnh để tạo tập tin Employee.dat và nhận các giá trị Mã NV, tên, tuổi, lương cơ bản và trợ cấp vào các field.
- Gán chi tiết vào các biến sau đó lần lượt gán vào các field.
- Sau khi tạo tập tin Employee, Hãy liệt k tất cả các mẩu tin của tập tin này.
- Liệt kê các nhân viên có lương bằng 1000.
- Cập nhật lại tiền trợ cấp cho tất cả các nhân viên thành 300 nếu lương cơ bản nhỏ hơn 2000 ngược lại lên 500.
- Xóa các mẩu tin trong tập tin Employee căn cứ vào mã nhân viên nhập vào.
- Tìm các nhân viên trong tập tin Employee theo mã nhân viên, hiển thị thông tin đầy đủ của nhân viên khi tìm thấy. Liệt kê các bộ sưu liệu được sử dụng của chương trình.

2. Viết chương trình tìm độ dài của file (nhớ mở file dạng nhị phân).

3. Viết chương trình đọc một file text và xóa các dòng trống nếu có trong file.

4. Viết chương trình cắt bỏ các dòng thừa, cắt bỏ các khoảng trống thừa, đổi các kí tự đầu mỗi từ ra chữ hoa của một file text.

5. Lập chương trình tạo một tập tin chứa các giá trị ngẫu nhiên. Sắp xếp chúng và lưu trữ sang một tập tin khác.

6. Viết chương trình tính số lần xuất hiện của một kí tự chữ cái trong một tập tin văn bản.

7. Viết chương trình tính số từ có trong một tập tin văn bản.

8. Viết chương trình nối hai tập tin văn bản với nhau thành một tập tin văn bản.

9. Viết chương trình nhập dữ liệu của các nhân viên của một cơ quan và lưu vào file. Sau đó thực hiện các câu lệnh sau: Nhập vào một số thứ tự, sửa dữ liệu của nhân viên theo thứ tự này, lưu lại nhân viên này vào file. Viết các hàm thực hiện các câu lệnh sau:

- Tìm lương thấp nhất của cơ quan.
- Tìm lương trung bình của cơ quan.
- In ra danh sách nhân viên.
- In ra những người có lương cao nhất.
- In ra những người có lương thấp nhất.

10. Viết chương trình quản lý lớp học bằng array. Mỗi học sinh được mô tả bằng: mã số (int), họ(8 kí tự), tên lĩt(24 kí tự), tên(8 kí tự), điểm toán, lý, hóa. Lưu học sinh này vào file HOCSINH.DAT. Chương trình có các chức năng sau:

- Thêm vào một học sinh.
- Xóa một học sinh.
- Xuất danh sách các học sinh (không có điểm).
- Xuất danh sách học sinh có xếp hạng, nếu cùng hạng thì sắp theo tên.

11. Viết chương trình tìm độ dài của file (nhớ mở file dạng nhị phân).

12. Viết chương trình đọc một file text và xóa các dòng trống nếu có trong file.

13. Viết chương trình cắt bỏ các dòng thừa, cắt bỏ các khoảng trống thừa, đổi các kí tự đầu mỗi từ ra chữ hoa của một file text.

14. Lập chương trình tạo một tập tin chứa các giá trị ngẫu nhiên. Sắp xếp chúng và lưu trữ sang một tập tin khác.

15. Viết chương trình tính số lần xuất hiện của một kí tự chữ cái trong một tập tin văn bản.

16. Viết chương trình tính số từ có trong một tập tin văn bản.

17. Viết chương trình nối hai tập tin văn bản với nhau thành một tập tin văn bản.

18. Viết chương trình nhập dữ liệu của các nhân viên của một cơ quan và lưu vào file. Sau đó thực hiện các câu lệnh sau: Nhập vào một số thứ tự, sửa dữ liệu của nhân viên theo thứ tự này, lưu lại nhân viên này vào file. Viết các hàm thực hiện các câu lệnh sau:

- Tìm lương thấp nhất của cơ quan.
- Tìm lương trung bình của cơ quan.
- In ra danh sách nhn viên.
- In ra những người có lương cao nhất.
- In ra những người có lương thấp nhất.

19. Viết chương trình quản lý lớp học bằng array. Mỗi học sinh được mô tả bằng: mã số (int), họ (8 kí tự), tên lót (24 kí tự), tên (8 kí tự), điểm toán, lý, hóa. Lưu học sinh này vào file HOCSINH.DAT. Chương trình có các chức năng sau :

- Thêm vào một học sinh
- Xóa một học sinh
- Xuất danh sách các học sinh (không có điểm)
- Xuất danh sách học sinh có xếp hạng, nếu cùng hạng thì sắp theo tên.

MỘT SỐ HÀM CHUẨN TRONG C

1. File ctype.h : Xử lý kí tự

Các hàm sau sẽ thực hiện việc kiểm tra, trả về là 0 nếu sai, trả về giá trị khác 0 nếu đúng.

int isalpha(int c) : Kiểm tra c có phải là kí tự chữ hay không ? (A … Z, a … z)

int isascii(int c) : Kiểm tra c có phải là kí tự ASCII hay không ?(0 … 127)

int iscntrl(int c) : Kiểm tra c có phải là kí tự điều khiển hay không?

int isdigit(int c) : Kiểm tra c có phải là kí tự số 0 … 9

int isgraph(int c) : Kiểm tra c có phải là kí tự in được, trừ khoảng trống.

int islower(int c) : Kiểm tra c có phải là kí tự thường hay không ?

int isprintf(int c) : Kiểm tra c có phải là kí tự in được

int ispunct(int c) : Kiểm tra c có phải là kí tự điều khiển hay khoảng trống

int isspace(int c) : Kiểm tra c có phải là kí tự khoảng trống?

int isuppper(int c) : Kiểm tra c có phải là kí tự hoa ? (A … Z)

+ **Các hàm toán học**

double acos(double x) : Tính arc cosince(x)

double asin(double x) : Tính arc sine(x)

double atan(double x) : Tính arc tangent(x)

double atan2(double x, double y) : Tính arc tangent(x/y)

double ceil(double x) : Trả về số nguyên(có kiểu double) nhỏ nhất và không nhỏ hơn x

double cos(double x) : Tính cosine(x), x : radian

double **cosh(double x)** **:** Tính Hãyoerbolic cosine(x)

double **exp(double x)** **:** Tính e^x

double **fabs(double x)** **:** Tính | x |

double **floor(double x)** **:** Trả về số nguyên lớn nhất và không lớn hơn x

double **fmod(double x, double y)** **:** Trả về số dư(kiểu double) của phép chia nguyên x/y

double **frexp(double x, int*exponent)** Chia x làm thành phần định trị(mantisa) và luỹ thừa(exponent) của 2 ($x = a*2^{exponent}$) trả về giá trị a

double **ldexp(double x, int exp):** Ngược lại với frexp, trả về $x*2^{exp}$

double **log(double x)** **:** Trả về giá trị log Neper của x

double **log10(double x)** **:** Trả về giá trị log 10 của x

double **modf(double x, double*intptr)** Chia x thnh phần lẻ(fractional – kết quả của hàm) v phần nguyên

double **pow(double x, doubley):** Tính x^y

double **sin(double x)** **:** Tính since(x), x:radian

double **sinh(double x)** **:** Tính Hãyperbolic của x

double **sqrt(double x)** **:** Tính căn bậc 2 của x

double **tan(double x)** **:** Tính tangent của x

double **tanh(double x)** **:** Tính Hãyperbolic tangent của x

+ **Các hàm xuất nhập chuẩn:**

a. Các hàm xuất nhập dữ liệu thông thường

int **getchar(void)** **:** Nhập một kí tự từ bên phím (**stdin**)

int putchar(int c) : Xuất kí tự ra màn hình (**stdout**)

char* gets(char*s) : Nhập một chuỗi kí tự từ bên phím

int puts(const char*s) : Xuất một chuỗi kí tự ra màn hình(có xuống dòng)

int printf(const char * format, [argument, …]) Xuất dữ liệu có định dạng ra màn hình

int scanf(const char * format, [address, …]) Nhập dữ liệu có định dạng ra màn hình

int sprintf(char*buffer,const char*format[argument, …]); Xuất dữ liệu có định dạng sang 1 chuỗi

int sscanf(const char*buffer, const char*format[argument, …]) Đọc một chuỗi

int vprintf(const char * format, va_list arlist);Xuất dữ liệu định dạng dùng danh sách đối số

int vscanf(const char * format, va_list arlist); Nhập dữ liệu định dạng dùng danh sách đối số

int vsprintf(char * buffer,const char*format, va_list arlist);Xuất dữ liệu định dạng ra chuỗi dùng danh sách đối số

int vsscanf(const char * buffer,const char*format, va_list arlist);Nhập dữ liệu định dạng vào chuỗi dùng danh sách đối số

b. Xuất nhập file

void clearerr(FILE*stream) : Xóa thông báolỗi

int fclose(FILE*stream) : Đóng file

int fcloseall(void) : Đóng tất cả các file đang mở

FILE*fdopen(int handle, char*type) : Gán 1 dòng(stream) cho file handle

FILE*fopen(const char*filename, char*type): Mở một file

FILE*freopen(const char*filename,const char * mode,FILE*stream); Mở một file mới và gán cho 1 hfile handle đ mở.

Mở một file với chế độ dùng chung

_fsopen:

#include<stdio.h>

#include<share.h>

FILE*_fsopen(const char*filename, const char*mode,int shflg);

int feof(FILE*stream) : Kiểm tra hết file(Macro)

int ferror(FILE*stream) : Kiểm tra có lỗi hay không

int fflush(FILE*stream) : Ghi buffer của dòng(stream) ln file

int fgetc(FILE*stream) : Lấy kí tự từ file

int fputc(int c, FILE*stream): Ghi kí tự c ln file

int fgetchar(void) : Lấy kí tự từ thiết bị nhập chuẩn(stdin)

int fputchar(int c) : Xuất lí tự ra thiết bị xuất chuẩn(stdout)

int getpos(FILE*stream, fpos_t*pos): Lấy vị trí hiện hnh

int fsetpos(FILE*stream, const fpos_t*pos) Ấn định vị trí file hiện hành

char* fgets(char*s, int n, FILE*stream) : Lấy một chuỗi từ file

int fputs(const char*s, FILE*stream): Ghi một chuỗi ln file

int fileno(FILE*stream) : Lấy file handle

int fflushall(void) : Xố các buffer của dòng nhập

Ghi các buffer của dòng xuất lên file

int fprintf(FILE*stream, const char*format, [, argument,…]) Ghi kết xuất có định dạng lên file

int fscanf(FILE*stream, const char*format, [, address,…]) Đọc dữ liệu có định dạng từ file

size_t fread(void *ptr, size_t size, size n, FILE*stream); :Đọc dữ liệu từ file

size_t fwrite(const void *ptr, size_t size, size n, FILE*stream); :Ghi dữ liệu lên file

int fseek(FILE*stream, long offset, int whence): Nhẩy đến vị trí offset trong file kể từ vị trí whence

long ftell(FILE*stream) **:** Lấy vị trí file hiện hình

int getw(FILE*stream) **:** Đọc một số nguyên từ file

int putw(int w, FILE*stream) **:** Xuất một số nguyên từ file

void perror(const char*s) **:** Xuất một thông báo lỗi hệ thống

int remove(const char*filename) **:** Macro xóa một file

int rename(const char*oldname, const char*newname) Đổi tên một file

void rewind(FILE*stream) **:** Đưa con trỏ về đầu file

int rmtmp(void) **:** Xoá các file tạm đã mở

Gán buffer cho một file:

void setbuf(FILE*stream, char buf)

int setvbuf(FILE*stream, char*buf, int type, size_t size)

FILE*tmpfile(void) **:** Mở một file tạm nhị phân

char* tempnam(char*dir, char *prefix) : Tạo một file có tên duy nhất trong thư mục

int unget(int c, FILE*stream) : Trả kí tự về cho file

int unlink(const char*filename) : Xóa một file

Tạo thông báo:

char*_strerror(const char*s);

char*strerror(int errnum);

+ **Các hàm tiện ích**

a. Đổi số thành chuỗi

Đổi số thực thành chuỗi, lấy ndig số hạng, dec số lẻ, đưa dấu vào biến sign

char*ecvt(double value, int ndig, int *dec, int *sign);

char*fcvt(double value, int ndig, int *dec, int *sign);

char*itoa(int value, char*string, int radix);

char*ltoa(long value, char*string, int radix);

char*utoa(unsigned long value, char*string, int radix);

char*gcvt(double value, int ndec, char*buf);

b. Đổi chuỗi thành số

double atof(const char*s);

long double _atold(const char*s);

int atoi(const char*s);

double strtod(const char *s, charendptr);**

long strtol(const char *s, charendptr, int radix);**

long double strtold(const char *s, charendptr);**

unsigned long strtoul(const char *s, charendptr, int radix);**

c. Xử lý số

Lấy trị tuyệt đối số nguyên, số thực, số phức:

int abs(int x);

complex: double abs(complex x);

double cabs(struct complex z);

long double cabsl(struct _complex z);

double fabs(double x);

long double fabsl(long double @E(x));

long int labs(long int x);

d. Tạo số ngẫu nhiên

void randomize(void); : Khởi động cơ chế lấy số ngẫu nhiên

int rand(void); : Lấy số ngẫu nhiên từ 0 đến RAND_MAX

Lấy số ngẫu nhiên từ 0 đến num – 1

Macro : random(num);

Function: int random(int num);

Lấy số ngẫu nhiên từ 0 đến seed

void srand(unsigned seed);

e. Cấp phát và thu hồi bộ nhớ:

Hàm xin cấp phát một vùng nhớ có size bytes/ nbytes

void * malloc(size_t size);

void far*farmalloc(unsigned long nbytes);

Hàm xin cấp phát một vùng nhớ cho nitems phần tử, mỗi phần tử có size bytes

void * calloc(size_t nitems, size_t size);

void far*farcalloc(unsigned long nitems, unsigned long size);

Hàm xin cấp phát lại vùng nhớ lúc trước đã cấp phát rồi ở địa chỉ oldblock với kích thước mới là size, có copy nội dung cũ sang vùng nhớ mới.

void * realloc(void * oldblock, size_t size);

void far* farrealloc(void far* oldblock, unsigned long nbytes);

Hàm trả về bộ nhớ cho hệ thống

void * free(void*block);

void far* farfree(void far*block);

+ **Xử lý string**

Copy một khối bộ nhớ n bytes từ src sang dest

void *memcácpy(void *dest, const void *src, int c, size_t n);

void *memcpy (void *dest, const void *src, size_t n);

void *memmove(void *dest, const void *src, size_t n);

void far * far _fmemcácpy(void far *dest, const void far *src,

int c, size_t n);

void far * far _fmemcpy (void far *dest, const void far *src,

size_t n);

Tìm kiếm kí tự c trong khối bộ nhớ s(n byte)

void *memchr (const void *s, int c, size_t n);

void far * far _fmemchr(const void far *s, int c, size_t n);

So sánh n byte đầu tiên giữa hai chuỗi s1 và s2, hàm memicmp so sánh nhưng không phân biệt chữ hoa và chữ thường

int memcmp (const void *s1, const void *s2, size_t n);

int memicmp(const void *s1, const void *s2, size_t n);

int far _fmemcmp (const void far *s1, const void far *s2, size_t n);

int far _fmemicmp(const void far *s1, const void far *s2, size_t n);

Cho n byte đầu tiên của s đều là kí tự c

void *memset (void *s, int c, size_t n);

void far * far _fmemset(void far *s, int c, size_t n);

Nối chuỗi src vo chuỗi dest

char *strúcat(char *dest, const char *src);

char far * far _fstrúcat(char far *dest, const char far *src);

Tìm địa chỉ vị trí xuất hiện đầu tiên của kí tự c trong chuỗi s

char *strúchr(const char *s, int c);

char far * far _fstrúchr(const char far *s, int c);

So sánh hai chuỗi, strúcmpi, _fstricmp, stricmp không phân biệt chữ hoa và chữ thường

int strúcmp(const char *s1, const char*s2);

int strúcmpi(const char *s1, const char *s2)

int stricmp(const char *s1, const char *s2);

int far _fstrúcmp(const char far *s1, const char far *s2);

int far _fstricmp(const char far *s1, const char far *s2);

Copy chuỗi src sang chuỗi dest

char *strúcpy(char *dest, const char *src);

char far * _fstrúcpy(char far *dest, const char far *src);

Tìm đoạn trong chuỗi s1 không chứa các kí tự có trong s2

size_t strúcspn(const char *s1, const char *s2);

size_t far _fstrúcspn(const char *s1, const char far *s2);

Tìm đoạn trong chuỗi s1 có chứa các kí tự có trong s2

size_t strspn(const char *s1, const char *s2);

size_t far _fstrspn(const char far *s1, const char far *s2);

Copy một chuỗi sang vị trí khc

char *strdup(const char *s);

char far * far _fstrdup(const char far *s);

Tìm độ dài của chuỗi

size_t strlen(const char *s);

size_t far _fstrlen(const char far *s);

Đổi một chuỗi thành chuỗi chữ thường

char *strlwr(char *s);

char far * far _fstrlwr(char far *s);

Đổi một chuỗi thành chuỗi chữ hoa

char *strupr(char *s);

char far * far _fstrupr(char far *s);

Nối một đoạn của chuỗi src vào chuỗi dest

char *strêncat(char *dest, const char *src, size_t maxlen);

char far * far _fstrêncat(char far *dest, const char far *src,

size_t maxlen);

So snh hai chuỗi

int strúcmp(const char *s1, const char*s2);

int far _fstrúcmp(const char far *s1, const char far *s2);

So sánh hai chuỗi, không phân biệt chuỗi hoa và chuỗi thường

int strúcmpi(const char *s1, const char *s2)

int stricmp(const char *s1, const char *s2);

int far _fstricmp(const char far *s1, const char far *s2);

Copy tối đa maxlen kí tự từ chuỗi src sang chuỗi dest

char *strêncpy(char *dest, const char *src, size_t maxlen);

char far * far _fstrêncpy(char far *dest, const char far *src size_t maxlen);

Cho n byte của chuỗi s l kí tự ch

char *strênset(char *s, int ch, size_t n);

char far * far _fstrênset(char far *s, int ch, size_t n);

Tìm xuất hiện đầu tiên của kí tự bất kỳ của chuỗi s2 trong chuỗi s1

char *strpbrk(const char *s1, const char *s2);

char far *far _fstrpbrk(const char far *s1, const char far *s2);

Tìm xuất hiện cuối cùng của kí tự c trong chuỗi s

char *strrchr(const char *s, int c);

char far * far _fstrrchr(const char far *s, int c);

Đảo chuỗi

char *strrev(char *s);

char far * far _fstrrev(char far *s);

Thiết lập tất cả chuỗi s đều mang kí tự ch

char *strset(char *s, int ch);

char far * far _fstrset(char far *s, int ch);

Tìm xuất hiện đầu tiên của chuỗi s2 trong s1

char *strstr(const char *s1, const char *s2);

char far * far _fstrstr(const char far *s1, const char far *s2);

Tìm từ đầu tiên trong s1 không có mặt trong s2

char *strtok(char *s1, const char *s2);

char far * far _fstrtok(char far *s1, const char far *s2);

TÀI LIỆU THAM KHẢO

[1] Ngôn ngữ lập trình, Khoa CNTT, trường ĐH CNTP TPHCM, 2003

[2] Nguyễn Thanh Thủy (chủ biên), **Nhập môn lập trình ngôn ngữ C**, Nhà xuất bản Khoa học và Kỹ thuật, 2005.

[3] Brian.W.Kernighan, Dennis M.RitChie, **C Programming Language**, 2nd Edition, Prentice Hall Software Series,

[4]. http://publications.gbdirect.co.uk/c_book/ : **The C Book**, Mike Banahan, Declan Brady and Mark Doran (the online version).

[5] Nguyễn Đình Tê, Hoàng Đức Hải, **Giáo trình Lý thuyết và bài tập ngôn ngữ C Tập 1**, Nhà xuất bản Giáo dục, 1999.

[6] Nguyễn Đình Tê, Hoàng Đức Hải, **Giáo trình Lý thuyết và bài tập ngôn ngữ C Tập 2**, Nhà xuất bản Giáo dục, 1999.

[7] Nguyễn Thanh Thủy và Nguyễn Quang Huy, **Bài tập lập trình ngôn ngữ C**, Nhà xuất bản Khoa học và Kỹ thuật, 1999.

[8] Đặng Thành Tín, *Tin Học II*, Nhà xuất bản Giáo dục.

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
Khoa Công nghệ Thông tin

PHẠM HỒNG THÁI

Bài giảng
NGÔN NGỮ LẬP TRÌNH C/C++

Hà Nội – 2003

LỜI NÓI ĐẦU

Ngôn ngữ lập trình (NNLT) C/C++ là một trong những ngôn ngữ lập trình hướng đối tượng mạnh và phổ biến hiện nay do tính mềm dẻo và đa năng của nó. Không chỉ các ứng dụng được viết trên C/C++ mà cả những chương trình hệ thống lớn đều được viết hầu hết trên C/C++. C++ là ngôn ngữ lập trình hướng đối tượng được phát triển trên nền tảng của C, không những khắc phục một số nhược điểm của ngôn ngữ C mà quan trọng hơn, C++ cung cấp cho người sử dụng (NSD) một phương tiện lập trình theo kỹ thuật mới: lập trình hướng đối tượng. Đây là kỹ thuật lập trình được sử dụng hầu hết trong các ngôn ngữ mạnh hiện nay, đặc biệt là các ngôn ngữ hoạt động trong môi truờng Windows như Microsoft Access, Visual Basic, Visual Foxpro …

Hiện nay NNLT C/C++ đã được đưa vào giảng dạy trong hầu hết các trường Đại học, Cao đẳng để thay thế một số NNLT đã cũ như FORTRAN, Pascal … Tập bài giảng này được viết ra với mục đích đó, trang bị kiến thức và kỹ năng thực hành cho sinh viên bắt đầu học vào NNLT C/C++ tại Khoa Công nghệ, Đại học Quốc gia Hà Nội. Để phù hợp với chương trình, tập bài giảng này chỉ đề cập một phần nhỏ đến kỹ thuật lập trình hướng đối tượng trong C++, đó là các kỹ thuật đóng gói dữ liệu, phương thức và định nghĩa mới các toán tử. Tên gọi của tập bài giảng này nói lên điều đó, có nghĩa nội dung của bài giảng thực chất là NNLT C được mở rộng với một số đặc điểm mới của C++. Về kỹ thuật lập trình hướng đối tượng (trong C++) sẽ được trang bị bởi một giáo trình khác. Tuy nhiên để ngắn gọn, trong tập bài giảng này tên gọi C/C++ sẽ được chúng tôi thay bằng C++.

Nội dung tập bài giảng này gồm 8 chương. Phần đầu gồm các chương từ 1 đến 6 chủ yếu trình bày về NNLT C++ trên nền tảng của kỹ thuật lập trình cấu trúc. Các chương còn lại (chương 7 và 8) sẽ trình bày các cấu trúc cơ bản trong C++ đó là kỹ thuật đóng gói (lớp và đối tượng) và định nghĩa phép toán mới cho lớp.

Tuy đã có nhiều cố gắng nhưng do thời gian và trình độ người viết có hạn nên chắc chắn không tránh khỏi sai sót, vì vậy rất mong nhận được sự góp ý của bạn đọc để bài giảng ngày càng một hoàn thiện hơn.

Tác giả.

CHƯƠNG 1

CÁC KHÁI NIỆM CƠ BẢN CỦA C++

Các yếu tố cơ bản
Môi trường làm việc của C++
Các bước để tạo và thực hiện một chương trình
Vào/ra trong C++

I. CÁC YẾU TỐ CƠ BẢN

Một ngôn ngữ lập trình (NNLT) bậc cao cho phép người sử dụng (NSD) biểu hiện ý tưởng của mình để giải quyết một vấn đề, bài toán bằng cách diễn đạt gần với ngôn ngữ thông thường thay vì phải diễn đạt theo ngôn ngữ máy (dãy các kí hiệu 0,1). Hiển nhiên, các ý tưởng NSD muốn trình bày phải được viết theo một cấu trúc chặt chẽ thường được gọi là *thuật toán* hoặc *giải thuật* và theo đúng các qui tắc của ngôn ngữ gọi là *cú pháp* hoặc *văn phạm*. Trong giáo trình này chúng ta bàn đến một ngôn ngữ lập trình như vậy, đó là ngôn ngữ lập trình C++ và làm thế nào để thể hiện các ý tưởng giải quyết vấn đề bằng cách viết thành chương trình trong C++.

Trước hết, trong mục này chúng ta sẽ trình bày về các qui định bắt buộc đơn giản và cơ bản nhất. Thông thường các qui định này sẽ được nhớ dần trong quá trình học ngôn ngữ, tuy nhiên để có một vài khái niệm tương đối hệ thống về NNLT C++ chúng ta trình bày sơ lược các khái niệm cơ bản đó. Người đọc đã từng làm quen với các NNLT khác có thể đọc lướt qua phần này.

1. Bảng ký tự của C++

Hầu hết các ngôn ngữ lập trình hiện nay đều sử dụng các kí tự tiếng Anh, các kí hiệu thông dụng và các con số để thể hiện chương trình. Các kí tự của những ngôn ngữ khác không được sử dụng (ví dụ các chữ cái tiếng Việt). Dưới đây là bảng kí tự được phép dùng để tạo nên những câu lệnh của ngôn ngữ C++.

- Các chữ cái la tinh (viết thường và viết hoa): a .. z và A .. Z. Cùng một chữ cái nhưng viết thường phân biệt với viết hoa. Ví dụ chữ cái 'a' là khác với 'A'.
- Dấu gạch dưới: _
- Các chữ số thập phân: 0, 1, . ., 9.

- Các ký hiệu toán học: +, -, *, /, % , &, ||, !, >, <, = ...
- Các ký hiệu đặc biệt khác: , ;: [ ], {}, #, dấu cách, ...

2. Từ khoá

Một từ khoá là một từ được qui định trước trong NNLT với một ý nghĩa cố định, thường dùng để chỉ các loại dữ liệu hoặc kết hợp thành câu lệnh. NSD có thể tạo ra những từ mới để chỉ các đối tượng của mình nhưng không được phép trùng với từ khoá. Dưới đây chúng tôi liệt kê một vài từ khoá thường gặp, ý nghĩa của các từ này, sẽ được trình bày dần trong các đề mục liên quan.

auto, break, case, char, continue, default, do, double, else, externe, float,

for, goto, if, int, long, register, return, short, sizeof, static, struct, switch,

typedef, union, unsigned, while ...

Một đặc trưng của C++ là các từ khoá luôn luôn được viết bằng chữ thường.

3. Tên gọi

Để phân biệt các đối tượng với nhau chúng cần có một tên gọi. Hầu hết một đối tượng được viết ra trong chương trình thuộc 2 dạng, một dạng đã có sẵn trong ngôn ngữ (ví dụ các từ khoá, tên các hàm chuẩn ...), một số do NSD tạo ra dùng để đặt tên cho hằng, biến, kiểu, hàm ... các tên gọi do NSD tự đặt phải tuân theo một số qui tắc sau:

- Là dãy ký tự liên tiếp (không chứa dấu cách) và phải bắt đầu bằng chữ cái hoặc gạch dưới.
- Phân biệt kí tự in hoa và thường.
- Không được trùng với từ khóa.
- Số lượng chữ cái dùng để phân biệt tên gọi có thể được đặt tuỳ ý.
- Chú ý các tên gọi có sẵn của C++ cũng tuân thủ theo đúng qui tắc trên.

Trong một chương trình nếu NSD đặt tên sai thì trong quá trình xử lý sơ bộ (trước khi chạy chương trình) máy sẽ báo lỗi (gọi là lỗi văn phạm).

Ví dụ 1 :

- Các tên gọi sau đây là đúng (được phép): i, i1, j, tinhoc, tin_hoc, luu_luong
- Các tên gọi sau đây là sai (không được phép): 1i, tin hoc, luu-luong-nuoc
- Các tên gọi sau đây là khác nhau: ha_noi, Ha_noi, HA_Noi, HA_NOI, ...

4. Chú thích trong chương trình

Một chương trình thường được viết một cách ngắn gọn, do vậy thông thường bên cạnh các câu lệnh chính thức của chương trình, NSD còn được phép viết vào chương trình các câu ghi chú, giải thích để làm rõ nghĩa hơn chương trình. Một chú thích có thể ghi chú về nhiệm vụ, mục đích, cách thức của thành phần đang được chú thích như biến, hằng, hàm hoặc công dụng của một đoạn lệnh ... Các chú thích sẽ làm cho chương trình sáng sủa, dễ đọc, dễ hiểu và vì vậy dễ bảo trì, sửa chữa về sau.

Có 2 cách báo cho chương trình biết một đoạn chú thích:

- Nếu chú thích là một đoạn kí tự bất kỳ liên tiếp nhau (trong 1 dòng hoặc trên nhiều dòng) ta đặt đoạn chú thích đó giữa cặp dấu đóng mở chú thích /* (mở) và */ (đóng).
- Nếu chú thích bắt đầu từ một vị trí nào đó cho đến hết dòng, thì ta đặt dấu // ở vị trí đó. Như vậy // sử dụng cho các chú thích chỉ trên 1 dòng.

Như đã nhắc ở trên, vai trò của đoạn chú thích là làm cho chương trình dễ hiểu đối với người đọc, vì vậy đối với máy các đoạn chú thích sẽ được bỏ qua. Lợi dụng đặc điểm này của chú thích đôi khi để tạm thời bỏ qua một đoạn lệnh nào đó trong chương trình (nhưng không xoá hẳn để khỏi phải gõ lại khi cần dùng đến) ta có thể đặt các dấu chú thích bao quanh đoạn lệnh này (ví dụ khi chạy thử chương trình, gỡ lỗi ...), khi cần sử dụng lại ta có thể bỏ các dấu chú thích.

Chú ý: Cặp dấu chú thích /* ... */ không được phép viết lồng nhau, ví dụ dòng chú thích sau là không được phép

```
/* Đây là đoạn chú thích /* chứa đoạn chú thích này */ như đoạn chú thích con */
```

cần phải sửa lại như sau:

- hoặc chỉ giữ lại cặp dấu chú thích ngoài cùng

```
/* Đây là đoạn chú thích chứa đoạn chú thích này như đoạn chú thích con */
```

- hoặc chia thành các đoạn chú thích liên tiếp nhau

```
/* Đây là đoạn chú thích */ /*chứa đoạn chú thích này*/ /*như đoạn chú thích con */
```

II. MÔI TRƯỜNG LÀM VIỆC CỦA C++

1. Khởi động - Thoát khỏi C++

Khởi động C++ cũng như mọi chương trình khác bằng cách nhấp đúp chuột lên biểu tượng của chương trình. Khi chương trình được khởi động sẽ hiện ra giao diện gồm có menu công việc và một khung cửa sổ bên dưới phục vụ cho soạn thảo. Một con

trỏ nhấp nháy trong khung cửa sổ và chúng ta bắt đầu nhập nội dung (văn bản) chương trình vào trong khung cửa sổ soạn thảo này. Mục đích của giáo trình này là trang bị những kiến thức cơ bản của lập trình thông qua NNLT C++ cho các sinh viên mới bắt đầu nên chúng tôi vẫn chọn trình bày giao diện của các trình biên dịch quen thuộc là Turbo C hoặc Borland C. Về các trình biên dịch khác độc giả có thể tự tham khảo trong các tài liệu liên quan.

Để kết thúc làm việc với C++ (soạn thảo, chạy chương trình ...) và quay về môi trường Windows chúng ta ấn **Alt-X**.

2. Giao diện và cửa sổ soạn thảo

a. Mô tả chung

Khi gọi chạy C++ trên màn hình sẽ xuất hiện một menu xổ xuống và một cửa sổ soạn thảo. Trên menu gồm có các nhóm chức năng: **File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help**. Để kích hoạt các nhóm chức năng, có thể ấn **Alt+chữ cái** biểu thị cho menu của chức năng đó (là chữ cái có gạch dưới). Ví dụ để mở nhóm chức năng **File** ấn **Alt+F**, sau đó dịch chuyển hộp sáng đến mục cần chọn rồi ấn Enter. Để thuận tiện cho NSD, một số các chức năng hay dùng còn được gắn với một tổ hợp các phím cho phép người dùng có thể chọn nhanh chức năng này mà không cần thông qua việc mở menu như đã mô tả ở trên. Một số tổ hợp phím cụ thể đó sẽ được trình bày vào cuối phần này. Các bộ chương trình dịch hỗ trợ người lập trình một môi trường tích hợp tức ngoài chức năng soạn thảo, nó còn cung cấp nhiều chức năng, tiện ích khác giúp người lập trình vừa có thể soạn thảo văn bản chương trình vừa gọi chạy chương trình vừa gỡ lỗi …

Các chức năng liên quan đến soạn thảo phần lớn giống với các bộ soạn thảo khác (như WinWord) do vậy chúng tôi chỉ trình bày tóm tắt mà không trình bày chi tiết ở đây.

b. Các chức năng soạn thảo

Giống hầu hết các bộ soạn thảo văn bản, bộ soạn thảo của Turbo C hoặc Borland C cũng sử dụng các phím sau cho quá trình soạn thảo:

- Dịch chuyển con trỏ: các phím mũi tên cho phép dịch chuyển con trỏ sang trái, phải một kí tự hoặc lên trên, xuống dưới 1 dòng. Để dịch chuyển nhanh có các phím như Home (về đầu dòng), End (về cuối dòng), PgUp, PgDn (lên, xuống một trang màn hình). Để dịch chuyển xa hơn có thể kết hợp các phím này cùng phím Control (Ctrl, ^) như ^PgUp: về đầu tệp, ^PgDn: về cuối tệp.
- Chèn, xoá, sửa: Phím Insert cho phép chuyển chế độ soạn thảo giữa chèn và đè. Các phím Delete, Backspace cho phép xoá một kí tự tại vị trí con trỏ và

trước vị trí con trỏ (xoá lùi).

- Các thao tác với khối dòng: Để đánh dấu khối dòng (thực chất là khối kí tự liền nhau bất kỳ) ta đưa con trỏ đến vị trí đầu ấn Ctrl-KB và Ctrl-KK tại vị trí cuối. Cũng có thể thao tác nhanh hơn bằng cách giữ phím Shift và dùng các phím dịch chuyển con trỏ quét từ vị trí đầu đến vị trí cuối, khi đó khối kí tự được đánh dấu sẽ chuyển mầu nền. Một khối được đánh dấu có thể dùng để cắt, dán vào một nơi khác trong văn bản hoặc xoá khỏi văn bản. Để thực hiện thao tác cắt dán, đầu tiên phải đưa khối đã đánh dấu vào bộ nhớ đệm bằng nhóm phím Shift-Delete (cắt), sau đó dịch chuyển con trỏ đến vị trí mới cần hiện nội dung vừa cắt và ấn tổ hợp phím Shift-Insert. Một đoạn văn bản được ghi vào bộ nhớ đệm có thể được dán nhiều lần vào nhiều vị trí khác nhau bằng cách lặp lại tổ hợp phím Shift-Insert tại các vị trí khác nhau trong văn bản. Để xoá một khối dòng đã đánh dấu mà không ghi vào bộ nhớ đệm, dùng tổ hợp phím Ctrl-Delete. Khi một nội dung mới ghi vào bộ nhớ đệm thì nó sẽ xoá (ghi đè) nội dung cũ đã có, do vậy cần cân nhắc để sử dụng phím Ctrl-Delete (xoá và không lưu lại nội dung vừa xoá vào bộ đệm) và Shift-Delete (xoá và lưu lại nội dung vừa xoá) một cách phù hợp.
- Tổ hợp phím Ctrl-A rất thuận lợi khi cần đánh dấu nhanh toàn bộ văn bản.

c. Chức năng tìm kiếm và thay thế

Chức năng này dùng để dịch chuyển nhanh con trỏ văn bản đến từ cần tìm. Để thực hiện tìm kiếm bấm Ctrl-QF, tìm kiếm và thay thế bấm Ctrl-QA. Vào từ hoặc nhóm từ cần tìm vào cửa sổ Find, nhóm thay thế (nếu dùng Ctrl-QA) vào cửa sổ Replace và đánh dấu vào các tuỳ chọn trong cửa sổ bên dưới sau đó ấn Enter. Các tuỳ chọn gồm: không phân biệt chữ hoa/thường, tìm từ độc lập hay đứng trong từ khác, tìm trong toàn văn bản hay chỉ trong phần được đánh dấu, chiều tìm đi đến cuối hay ngược về đầu văn bản, thay thế có hỏi lại hay không hỏi lại … Để dịch chuyển con trỏ đến các vùng khác nhau trong một menu hay cửa sổ chứa các tuỳ chọn ta sử dụng phím Tab.

d. Các chức năng liên quan đến tệp

- Ghi tệp lên đĩa: Chọn menu File\Save hoặc phím F2. Nếu tên tệp chưa có (còn mang tên Noname.cpp) máy sẽ yêu cầu cho tên tệp. Phần mở rộng của tên tệp được mặc định là CPP.
- Soạn thảo tệp mới: Chọn menu File\New. Hiện ra cửa sổ soạn thảo trắng và tên file tạm thời lấy là Noname.cpp.
- Soạn thảo tệp cũ: Chọn menu File\Open hoặc ấn phím F3, nhập tên tệp hoặc dịch chuyển con trỏ trong vùng danh sách tệp bên dưới đến tên tệp cần soạn rồi ấn Enter. Cũng có thể áp dụng cách này để soạn tệp mới khi không nhập

vào tên tệp cụ thể.

- Ghi tệp đang soạn thảo lên đĩa với tên mới: Chọn menu File\Save As và nhập tên tệp mới vào rồi ấn Enter.

e. Chức năng dịch và chạy chương trình

- Ctrl-F9: Khởi động chức năng dịch và chạy toàn bộ chương trình.
- F4: Chạy chương trình từ đầu đến dòng lệnh hiện tại (đang chứa con trỏ)
- F7: Chạy từng lệnh một của hàm main(), kể cả các lệnh con trong hàm.
- F8: Chạy từng lệnh một của hàm main(). Khi đó mỗi lời gọi hàm được xem là một lệnh (không chạy từng lệnh trong các hàm được gọi).

Các chức năng liên quan đến dịch chương trình có thể được chọn thông qua menu Compile (Alt-C).

f. Tóm tắt một số phím nóng hay dùng

- Các phím kích hoạt menu: Alt+chữ cái đại diện cho nhóm menu đó. Ví dụ Alt-F mở menu File để chọn các chức năng cụ thể trong nó như Open (mở file), Save (ghi file lên đĩa), Print (in nội dung văn bản chương trình ra máy in), … Alt-C mở menu Compile để chọn các chức năng dịch chương trình.
- Các phím dịch chuyển con trỏ khi soạn thảo.
- F1: mở cửa sổ trợ giúp. Đây là chức năng quan trọng giúp người lập trình nhớ tên lệnh, cú pháp và cách sử dụng.
- F2: ghi tệp lên đĩa.
- F3: mở tệp cũ ra sửa chữa hoặc soạn thảo tệp mới.
- F4: chạy chương trình đến vị trí con trỏ.
- F5: Thu hẹp/mở rộng cửa sổ soạn thảo.
- F6: Chuyển đổi giữa các cửa sổ soạn thảo.
- F7: Chạy chương trình theo từng lệnh, kể cả các lệnh trong hàm con.
- F8: Chạy chương trình theo từng lệnh trong hàm chính.
- F9: Dịch và liên kết chương trình. Thường dùng chức năng này để tìm lỗi cú pháp của chương trình nguồn trước khi chạy.
- Alt-F7: Chuyển con trỏ về nơi gây lỗi trước đó.
- Alt-F8: Chuyển con trỏ đến lỗi tiếp theo.

- Ctrl-F9: Chạy chương trình.
- Ctrl-Insert: Lưu khối văn bản được đánh dấu vào bộ nhớ đệm.
- Shift-Insert: Dán khối văn bản trong bộ nhớ đệm vào văn bản tại vị trí con trỏ.
- Shift-Delete: Xoá khối văn bản được đánh dấu, lưu nó vào bộ nhớ đệm.
- Ctrl-Delete: Xoá khối văn bản được đánh dấu (không lưu vào bộ nhớ đệm).
- Alt-F5: Chuyển sang cửa sổ xem kết quả của chương trình vừa chạy xong.
- Alt-X: thoát C++ về lại Windows.

3. Cấu trúc một chương trình trong C++

Một chương trình C++ có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trên một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Cụ thể, thông thường một chương trình gồm có các nội dung sau:

- Phần khai báo các tệp nguyên mẫu: khai báo tên các tệp chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà NSD sẽ dùng trong chương trình.
- Phần khai báo các kiểu dữ liệu, các biến, hằng ... do NSD định nghĩa và được dùng chung trong toàn bộ chương trình.
- Danh sách các hàm của chương trình (do NSD viết, bao gồm cả hàm main()). Cấu trúc chi tiết của mỗi hàm sẽ được đề cập đến trong chương 4.

Dưới đây là một đoạn chương trình đơn giản chỉ gồm 1 hàm chính là hàm main(). Nội dung của chương trình dùng in ra màn hình dòng chữ: *Chào các bạn, bây giờ là 2 giờ*.

```
#include <iostream.h>                // khai báo tệp nguyên mẫu để
void main()                          // được sử dụng toán tử in cout <<
{
```

```
    int h = 2,                                  // Khai báo và khởi tạo biến h = 2
    cout << “Chào các bạn, bây giờ là ” << h << " giờ" ;          // in ra màn hình
}
```

Dòng đầu tiên của chương trình là khai báo tệp nguyên mẫu iostream.h. Đây là khai báo bắt buộc vì trong chương trình có sử dụng phương thức chuẩn “cout <<” (in ra màn hình), phương thức này được khai báo và định nghĩa sẵn trong iostream.h.

Không riêng hàm main(), mọi hàm khác đều phải bắt đầu tập hợp các câu lệnh của mình bởi dấu { và kết thúc bởi dấu }. Tập các lệnh bất kỳ bên trong cặp dấu này được gọi là khối lệnh. Khối lệnh là một cú pháp cần thiết trong các câu lệnh có cấu trúc như ta sẽ thấy trong các chương tiếp theo.

III. CÁC BƯỚC ĐỂ TẠO VÀ THỰC HIỆN MỘT CHƯƠNG TRÌNH

1. Qui trình viết và thực hiện chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

1. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output). Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó.Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x1 và x2 của phương trình. Kiểu của a, b, c, x1, x2 là các số thực.
2. Xác định thuật toán giải.
3. Cụ thể hoá các khai báo kiểu và thuật toán thành dãy các lệnh, tức viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.
4. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.
5. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NSD.

2. Soạn thảo tệp chương trình nguồn

Soạn thảo chương trình nguồn là một công việc đơn giản: gõ nội dung của chương trình (đã viết ra giấy) vào trong máy và lưu lại nó lên đĩa. Thông thường khi đã lưu lại chương trình lên đĩa lần sau sẽ không cần phải gõ lại. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp

C++ (Borland C, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của máy. Văn bản chương trình cần được trình bày sáng sủa, rõ ràng. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

3. Dịch chương trình

Sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (ấn tổ hợp phím Alt-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NSD có thể dùng Alt-F8 để chuyển con trỏ đến lỗi tiếp theo hoặc dịch lại. Để chuyển con trỏ về ngược lại lỗi trước đó có thể dùng Alt-F7. Quá trình sửa lỗi – dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp.

Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.Tệp này có thể lưu tạm thời trong bộ nhớ phục vụ cho quá trình chạy chương trình hoặc lưu lại trên đĩa tuỳ theo tuỳ chọn khi dịch của NSD. Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc thông báo chương trình đã được dịch thành công (không còn lỗi). Các lỗi này được gọi là lỗi cú pháp.

Để dịch chương trình ta chọn menu \Compile\Compile hoặc \Compile\Make hoặc nhanh chóng hơn bằng cách ấn tổ hợp phím Alt-F9.

4. Chạy chương trình

Ấn Ctrl-F9 để chạy chương trình, nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NSD kiểm tra. Nếu kết quả chưa được như mong muốn, quay lại văn bản để sửa và lại chạy lại chương trình. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả. Để xem lại kết quả đã hiện ấn Alt-F5. Sau khi xem xong để quay lại cửa sổ soạn thảo ấn phím bất kỳ.

IV. VÀO/RA TRONG C++

Trong phần này chúng ta làm quen một số lệnh đơn giản cho phép NSD nhập dữ liệu vào từ bàn phím hoặc in kết quả ra màn hình. Trong phần sau của giáo trình chúng ta sẽ khảo sát các câu lệnh vào/ra phức tạp hơn

1. Vào dữ liệu từ bàn phím

Để nhập dữ liệu vào cho các biến có tên **biến_1**, **biến_2**, **biến_3** chúng ta sử dụng câu lệnh:

```
cin >> biến_1 ;
cin >> biến_2 ;
cin >> biến_3 ;
```

hoặc:

```
cin >> biến_1 >> biến_2 >> biến_3 ;
```

biến_1, biến_2, biến_3 là các **biến** được sử dụng để lưu trữ các giá trị NSD nhập vào từ bàn phím. Khái niệm biến sẽ được mô tả cụ thể hơn trong chương 2, ở đây biến_1, biến_2, biến_3 được hiểu là các tên gọi để chỉ 3 giá trị khác nhau. Hiển nhiên có thể nhập dữ liệu nhiều hơn 3 biến bằng cách tiếp tục viết tên biến vào bên phải sau dấu >> của câu lệnh.

Khi chạy chương trình nếu gặp các câu lệnh trên chương trình sẽ "tạm dừng" để chờ NSD nhập dữ liệu vào cho các biến. Sau khi NSD nhập xong dữ liệu, chương trình sẽ tiếp tục chạy từ câu lệnh tiếp theo sau của các câu lệnh trên.

Cách thức nhập dữ liệu của NSD phụ thuộc vào loại giá trị của biến cần nhập mà ta gọi là **kiểu,** ví dụ nhập một số có cách thức khác với nhập một chuỗi kí tự. Giả sử cần nhập độ dài hai cạnh của một hình chữ nhật, trong đó cạnh dài được qui ước bằng tên biến cd và chiều rộng được qui ước bởi tên biến cr. Câu lệnh nhập sẽ như sau:

```
cin >> cd >> cr ;
```

Khi máy dừng chờ nhập dữ liệu NSD sẽ gõ giá trị cụ thể của các chiều dài, rộng theo đúng thứ tự trong câu lệnh. Các giá trị này cần cách nhau bởi ít nhất một dấu trắng (ta qui ước gọi dấu trắng là một trong 3 loại dấu được nhập bởi các phím sau: phím spacebar (dấu cách), phím tab (dấu tab) hoặc phím Enter (dấu xuống dòng)). Các giá trị NSD nhập vào cũng được hiển thị trên màn hình để NSD dễ theo dõi.

Ví dụ nếu NSD nhập vào 23 11 ↵ thì chương trình sẽ gán giá trị 23 cho biến cd và 11 cho biến cr.

Chú ý: giả sử NSD nhập 2311 ↵ (không có dấu cách giữa 23 và 11) thì chương trình sẽ xem 2311 là một giá trị và gán cho cd. Máy sẽ tạm dừng chờ NSD nhập tiếp giá trị cho biến cr.

2. In dữ liệu ra màn hình

Để in giá trị của các **biểu thức** ra màn hình ta dùng câu lệnh sau:

```
cout << bt_1 ;
```

```
cout << bt_2 ;
cout << bt_3 ;
```

hoặc:

```
cout << bt_1 << bt_2 << bt_3 ;
```

cũng giống câu lệnh nhập ở đây chúng ta cũng có thể mở rộng lệnh in với nhiều hơn 3 biểu thức. Câu lệnh trên cho phép in giá trị của các biểu thức bt_1, bt_2, bt_3. Các giá trị này có thể là tên của biến hoặc các kết hợp tính toán trên biến.

Ví dụ để in câu "Chiều dài là " và số 23 và tiếp theo là chữ "mét", ta có thể sử dụng 3 lệnh sau đây:

```
cout << "Chiều dài là" ;
cout << 23 ;
cout << "mét";
```

hoặc có thể chỉ bằng 1 lệnh:

```
cout << "Chiều dài là 23 mét" ;
```

Trường hợp chưa biết giá trị cụ thể của chiều dài, chỉ biết hiện tại giá trị này đã được lưu trong biến cd (ví dụ đã được nhập vào là 23 từ bàn phím bởi câu lệnh cin >> cd trước đó) và ta cần biết giá trị này là bao nhiêu thì có thể sử dụng câu lệnh in ra màn hình.

```
cout << "Chiều dài là" << cd << "mét" ;
```

Khi đó trên màn hình sẽ hiện ra dòng chữ: "Chiều dài là 23 mét". Như vậy trong trường hợp này ta phải dùng đến ba lần dấu phép toán << chứ không phải một như câu lệnh trên. Ngoài ra phụ thuộc vào giá trị hiện được lưu trong biến cd, chương trình sẽ in ra số chiều dài thích hợp chứ không chỉ in cố định thành "chiều dài là 23 mét". Ví dụ nếu cd được nhập là 15 thì lệnh trên sẽ in câu "chiều dài là 15 mét".

Một giá trị cần in không chỉ là một biến như cd, cr, ... mà còn có thể là một biểu thức, điều này cho phép ta dễ dàng yêu cầu máy in ra diện tích và chu vi của hình chữ nhật khi đã biết cd và cr bằng các câu lệnh sau:

```
cout << "Diện tích = " << cd * cr ;
cout << "Chu vi = " << 2 * (cd + cr) ;
```

hoặc gộp tất cả thành 1 câu lệnh:

```
cout << Diện tích = " << cd * cr << '\n' << " Chu vi = " << 2 * (cd + cr) ;
```

ở đây có một kí tự đặc biệt: đó là kí tự '\n' kí hiệu cho kí tự xuống dòng, khi gặp kí tự này chương trình sẽ in các phần tiếp theo ở đầu dòng kế tiếp. Do đó kết quả của câu lệnh trên là 2 dòng sau đây trên màn hình:

```
Diện tích = 253
Chu vi = 68
```

ở đây 253 và 68 lần lượt là các giá trị mà máy tính được từ các biểu thức cd * cr, và 2 * (cd + cr) trong câu lệnh in ở trên.

Chú ý: để sử dụng các câu lệnh nhập và in trong phần này, đầu chương trình phải có dòng khai báo #include <iostream.h>.

Thông thường ta hay sử dụng lệnh in để in câu thông báo nhắc NSD nhập dữ liệu trước khi có câu lệnh nhập. Khi đó trên màn hình sẽ hiện dòng thông báo này rồi mới tạm dừng chờ dữ liệu nhập vào từ bàn phím. Nhờ vào thông báo này NSD sẽ biết phải nhập dữ liệu, nhập nội dung gì và như thế nào ... ví dụ:

```
cout << "Hãy nhập chiều dài: "; cin >> cd;
cout << "Và nhập chiều rộng: "; cin >> cr;
```

khi đó máy sẽ in dòng thông báo "Hãy nhập chiều dài: " và chờ sau khi NSD nhập xong 23 ↵, máy sẽ thực hiện câu lệnh tiếp theo tức in dòng thông báo "Và nhập chiều rộng: " và chờ đến khi NSD nhập xong 11 ↵ chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp theo.

Ví dụ 2 : Từ các thảo luận trên ta có thể viết một cách đầy đủ chương trình tính diện tích và chu vi của một hình chữ nhật. Để chương trình có thể tính với các bộ giá trị khác nhau của chiều dài và rộng ta cần lưu giá trị này vào trong các biến (ví dụ cd, cr).

```
#include <iostream.h>      // khai báo tệp nguyên mẫu để dùng được cin, cout
void main()                // đây là hàm chính của chương trình
{
    float cd, cr ;      // khai báo các biến có tên cd, cr để chứa độ dài các cạnh
    cout << "Hãy nhập chiều dài: " ; cin >> cd ;          // nhập dữ liệu
    cout << "Hãy nhập chiều rộng: " ; cin >> cr ;
    cout << "Diện tích = " << cd * cr << '\n' ;           // in kết quả
    cout << "Chu vi = " << 2 * (cd + cr) << '\n';
    return ;
}
```

Chương trình này có thể gõ vào máy và chạy. Khi chạy đến câu lệnh nhập, chương trình dừng để chờ nhận chiều dài và chiều rộng, NSD nhập các giá trị cụ thể, chương trình sẽ tiếp tục thực hiện và in ra kết quả. Thông qua câu lệnh nhập dữ liệu và 2 biến cd, cr NSD có thể yêu cầu chương trình cho kết quả của một hình chữ nhật bất

kỳ chứ không chỉ trong trường hợp hình có chiều dài 23 và chiều rộng 11 như trong ví dụ cụ thể trên.

3. Định dạng thông tin cần in ra màn hình

Một số định dạng đơn giản được chúng tôi trình bày trước ở đây. Các định dạng chi tiết và phức tạp hơn sẽ được trình bày trong các phần sau của giáo trình. Để sử dụng các định dạng này cần khai báo file nguyên mẫu <iomanip.h> ở đầu chương trình bằng chỉ thị #include <iomanip.h>.

- endl: Tương đương với kí tự xuống dòng '\n'.
- setw(n): Bình thường các giá trị được in ra bởi lệnh cout << sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này qui định độ rộng dành để in ra các giá trị là n cột màn hình. Nếu n lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.
- setprecision(n): Chỉ định số chữ số của phần thập phân in ra là n. Số sẽ được làm tròn trước khi in ra.
- setiosflags(ios::showpoint): Phương thức setprecision chỉ có tác dụng trên một dòng in. Để cố định các giá trị đã đặt cho mọi dòng in (cho đến khi đặt lại giá trị mới) ta sử dụng phương thức setiosflags(ios::showpoint).

Ví dụ sau minh hoạ cách sử dụng các phương thức trên.

Ví dụ 3 :

```
#include <iostream.h>                    // để sử dụng cout <<
#include <iomanip.h>                     // để sử dụng các định dạng
#include <conio.h>                       // để sử dụng các hàm clrscr() và getch()
void main()
{
      clrscr();                          // xoá màn hình
      cout << "CHI TIÊU" << endl << "=======" << endl ;
      cout << setiosflags(ios::showpoint) << setprecision(2) ;
      cout << "Sách vở" << setw(20) << 123.456 << endl;
      cout << "Thức ăn" << setw(20) << 2453.6 << endl;
      cout << "Quần áo lạnh" << setw(15) << 3200.0 << endl;
```

```
    getch();                        // tạm dừng (để xem kết quả)
    return ;                        // kết thúc thực hiện hàm main()
}
```

Chương trình này khi chạy sẽ in ra bảng sau:

```
CHI TIÊU
========
Sách vở          123.46
Thức ăn          2453.60
Quần áo lạnh     3200.00
```

Chú ý: toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập kí tự hoặc xâu kí tự, C++ cung cấp các phương thức (hàm) sau:

- **cin.get(c)**: cho phép nhập một kí tự vào biến kí tự c,
- **cin.getline(s,n)**: cho phép nhập tối đa n-1 kí tự vào xâu s.

các hàm trên khi thực hiện sẽ lấy các kí tự còn lại trong bộ nhớ đệm (của lần nhập trước) để gán cho c hoặc s. Do toán tử cin >> x sẽ để lại kí tự xuống dòng trong bộ đệm nên kí tự này sẽ làm trôi các lệnh sau đó như cin.get(c), cin.getline(s,n) (máy không dừng để nhập cho c hoặc s). Vì vậy trước khi sử dụng các phương thức cin.get(c) hoặc cin.getline(s,n) nên sử dụng phương thức cin.ignore(1) để lấy ra kí tự xuống dòng còn sót lại trong bộ đệm. Ví dụ đoạn lệnh sau cho phép nhập một số nguyên x (bằng toán tử >>) và một kí tự c (bằng phương thức cin.get(c)):

```
int x;
char c;
cin >> x; cin.ignore(1);
cin.get(c);
```

4. Vào/ra trong C

Trong phần trên chúng tôi đã trình bày 2 toán tử vào/ra và một số phương thức, hàm nhập và định dạng trong C++. Phần này chúng tôi trình bày các câu lênh nhập xuất theo khuôn dạng cũ trong C. Hiển nhiên các câu lệnh này vẫn dùng được trong chương trình viết bằng C++, tuy nhiên chỉ nên sử dụng hoặc các câu lệnh của C++ hoặc của C, không nên dùng lẫn lộn cả hai vì dễ gây nhầm lẫn. Do đó mục này chỉ có

giá trị tham khảo để bạn đọc có thể hiểu được các câu lệnh vào/ra trong các chương trình viết theo NNLT C cũ.

a. In kết quả ra màn hình

Để in các giá trị bt_1, bt_2, …, bt_n ra màn hình theo một khuôn dạng mong muốn ta có thể sử dụng câu lệnh sau đây:

printf(dòng định dạng, bt_1, bt_2, ..., bt_n) ;

trong đó dòng định dạng là một dãy kí tự đặt trong cặp dấu nháy kép (“”) qui định khuôn dạng cần in của các giá trị bt_1, bt_2, …, bt_n. Các bt_i có thể là các hằng, biến hay các biểu thức tính toán. Câu lệnh trên sẽ in giá trị của các bt_i này theo thứ tự xuất hiện của chúng và theo qui định được cho trong dòng định dạng.

Ví dụ, giả sử x = 4, câu lệnh:

printf(“%d %0.2f”, 3, x + 1) ;

sẽ in các số 3 và 5.00 ra màn hình, trong đó 3 được in dưới dạng số nguyên (được qui định bởi “%d”) và x + 1 (có giá trị là 5) được in dưới dạng số thực với 2 số lẻ thập phân (được qui định bởi “%0.2f”). Cụ thể, các kí tự đi sau kí hiệu % dùng để định dạng việc in gồm có:

d	in số nguyên dưới dạng hệ thập phân
o	in số nguyên dạng hệ 8
x, X	in số nguyên dạng hệ 16
u	in số nguyên dạng không dấu
c	in kí tự
s	in xâu kí tự
e, E	in số thực dạng dấu phẩy động
f	in số thực dạng dấu phẩy tĩnh

– Các kí tự trên phải đi sau dấu %. Các kí tự nằm trong dòng định dạng nếu không đi sau % thì sẽ được in ra màn hình. Muốn in % phải viết 2 lần (tức %%).

Ví dụ câu lệnh: printf(“Tỉ lệ học sinh giỏi: %0.2f %%”, 32.486) ;

sẽ in câu “Tỉ lệ học sinh giỏi: “, tiếp theo sẽ in số 32.486 được làm tròn đến 2 số lẻ thập phân lấp vào vị trí của “%0.2f”, và cuối cùng sẽ in dấu “%” (do có %% trong dòng định dạng). Câu được in ra màn hình sẽ là:

Tỉ lệ học sinh giỏi: 32.49%

Chú ý: Mỗi bt_i cần in phải có một định dạng tương ứng trong dòng định dạng.

Ví dụ câu lệnh trên cũng có thể viết:

```
printf("%s %0.2f" , "Tỉ lệ học sinh giỏi: ", 32.486);
```

trong câu lệnh này có 2 biểu thức cần in. Biểu thức thứ nhất là xâu kí tự "Tỉ lệ học sinh giỏi:" được in với khuôn dạng %s (in xâu kí tự) và biểu thức thứ hai là 32.486 được in với khuôn dạng %0.2f (in số thực với 2 số lẻ phần thập phân).

- Nếu giữa kí tự % và kí tự định dạng có số biểu thị độ rộng cần in thì giá trị in ra sẽ được gióng cột sang lề phải, để trống các dấu cách phía trước. Nếu độ rộng âm (thêm dấu trừ – phía trước) sẽ gióng cột sang lề trái. Nếu không có độ rộng hoặc độ rộng bằng 0 (ví dụ %0.2f) thì độ rộng được tự điều chỉnh đúng bằng độ rộng của giá trị cần in.
- Dấu + trước độ rộng để in giá trị số kèm theo dấu (dương hoặc âm)
- Trước các định dạng số cần thêm kí tự l (ví dụ ld, lf) khi in số nguyên dài long hoặc số thực với độ chính xác gấp đôi double.

Ví dụ 4 :

```
main()
{
    int i = 2, j = 3 ;
    printf("Chương trình tính tổng 2 số nguyên:\ni + j = %d", i+j);
}
```

sẽ in ra:

```
Chương trình tính tổng 2 số nguyên:
i + j = 5.
```

b. Nhập dữ liệu từ bàn phím

scanf(dòng định dạng, biến_1, biến_2, ..., biến_n) ;

Lệnh này cho phép nhập dữ liệu vào cho các biến biến_1, …, biến_n. Trong đó dòng định dạng chứa các định dạng về kiểu biến (nguyên, thực, kí tự …) được viết như trong mô tả câu lệnh printf. Các biến được viết dưới dạng địa chỉ của chúng tức có dấu & trước mỗi tên biến. Ví dụ câu lệnh:

```
scanf("%d %f %ld", &x, &y, &z) ;
```

cho phép nhập giá trị cho các biến x, y, z trong đó x là biến nguyên, y là biến thực và z là biến nguyên dài (long). Câu lệnh:

```
scanf("%2d %f %lf %3s", &i, &x, &d, s);
```

cho phép nhập giá trị cho các biến i, x, d, s, trong đó i là biến nguyên có 2 chữ số, f là biến thực (độ dài tùy ý), d là biến nguyên dài và s là xâu kí tự có 3 kí tự. Giả sử NSD nhập vào dãy dữ liệu: 12345 67abcd ↵ thì các biến trên sẽ được gán các giá trị như sau: i = 12, x = 345, d = 67 và s = "abc". Kí tự d và dấu enter (↵) sẽ được lưu lại trong bộ nhớ và tự động gán cho các biến của lần nhập sau.

Cuối cùng, chương trình trong ví dụ 3 được viết lại với printf() và scanf() như sau:

Ví dụ 5 :

```
#include <stdio.h>              // để sử dụng các hàm printf() và scanf()
#include <conio.h>              // để sử dụng các hàm clrscr() và getch()
void main()
{
    clrscr();                   // xoá màn hình
    printf("CHI TIÊU\n=======\n") ;
    printf("Sách vở %20.2f\n" , 123.456) ;
    printf("Thức ăn %20.2f\n" , 2453.6) ;
    printf("Quần áo lạnh %15.2f\n" , 3200.0) ;
    getch();                    // tạm dừng (để xem kết quả)
    return ;                    // kết thúc thực hiện hàm main()
}
```

BÀI TẬP

1. Những tên gọi nào sau đây là hợp lệ:

– x – 123variabe – tin_hoc – toan tin – so-dem

– RADIUS – one.0 – number# – Radius – nam2000

2. Bạn hãy thử viết một chương trình ngắn nhất có thể được.

3. Tìm các lỗi cú pháp trong chương trình sau:

```
#include (iostream.h)
void main();                    / Giải phương trình bậc 1
{
    cout << 'Day la chương trình: Gptb1.\nXin chao cac ban';
    getch();
}
```

4. Viết chương trình in nội dung một bài thơ nào đó.

5. Viết chương trình in ra 4 dòng, 2 cột gồm các số sau và gióng cột:

– thẳng theo lề trái	0.63	64.1
– thẳng theo lề phải	12.78	-11.678
– thẳng theo dấu chấm thập phân	-124. 6	59.002
	65.7	-1200.654

6. Hãy viết và chạy các chương trình trong các ví dụ 3, 5.

7. Chương trình sau khai báo 5 biến kí tự a, b, c, d, e và một biến số nam. Hãy điền thêm các câu lệnh vào các dòng … để chương trình thực hiện nhiệm vụ sau:

- Nhập giá trị cho biến nam
- Nhập giá trị cho các biến kí tự a, b, c, d, e.
- In ra màn hình dòng chữ được ghép bởi 5 kí tự đã nhập và chữ "năm" sau đó in số đã nhập (nam). Ví dụ nếu 5 chữ cái đã nhập là 'H', 'A', 'N', 'O', 'I' và nam được nhap là 2000, thì màn hình in ra dòng chữ: HANOI năm 2000.
- Nhập chương trình đã sửa vào máy và chạy để kiểm tra kết quả.

```
#include <iostream.h>
#include <conio.h>
main()
{
    int nam;
    char a, b, c, d, e;
    clrscr();
    cin >> nam ;
    ... ;
    cin.get(a); cin.get(b); cin.get(c); ... ; ... ;
```

```
        // in kết quả
        cout << a << ... << ... << ... << ... << " nam " << ... ;
        getch();
}
```

CHƯƠNG 2

KIỂU DỮ LIỆU, BIỂU THỨC VÀ CÂU LỆNH

Kiểu dữ liệu đơn giản
Hằng - khai báo và sử dụng hằng
Biến - khai báo và sử dụng biến
Phép toán, biểu thức và câu lệnh
Thư viện các hàm toán học

I. KIỂU DỮ LIỆU ĐƠN GIẢN

1. Khái niệm về kiểu dữ liệu

Thông thường dữ liệu hay dùng là số và chữ. Tuy nhiên việc phân chia chỉ 2 loai dữ liệu là không đủ. Để dễ dàng hơn cho lập trình, hầu hết các NNLT đều phân chia dữ liệu thành nhiều kiểu khác nhau được gọi là các kiểu cơ bản hay chuẩn. Trên cơ sở kết hợp các kiểu dữ liệu chuẩn, NSD có thể tự đặt ra các kiểu dữ liệu mới để phục vụ cho chương trình giải quyết bài toán của mình. Có nghĩa lúc đó mỗi đối tượng được quản lý trong chương trình sẽ là một tập hợp nhiều thông tin hơn và được tạo thành từ nhiều loại (kiểu) dữ liệu khác nhau. Dưới đây chúng ta sẽ xét đến một số kiểu dữ liệu chuẩn được qui định sẵn bởi C++.

Một biến như đã biết là một số ô nhớ liên tiếp nào đó trong bộ nhớ dùng để lưu trữ dữ liệu (vào, ra hay kết quả trung gian) trong quá trình hoạt động của chương trình. Để quản lý chặt chẽ các biến, NSD cần khai báo cho chương trình biết trước tên biến và kiểu của dữ liệu được chứa trong biến. Việc khai báo này sẽ làm chương trình quản lý các biến dễ dàng hơn như trong việc phân bố bộ nhớ cũng như quản lý các tính toán trên biến theo nguyên tắc: chỉ có các dữ liệu cùng kiểu với nhau mới được phép làm toán với nhau. Do đó, khi đề cập đến một kiểu chuẩn của một NNLT, thông thường chúng ta sẽ xét đến các yếu tố sau:

- tên kiểu: là một từ dành riêng để chỉ định kiểu của dữ liệu.
- số byte trong bộ nhớ để lưu trữ một đơn vị dữ liệu thuộc kiểu này: Thông thường số byte này phụ thuộc vào các trình biên dịch và hệ thống máy khác nhau, ở đây ta chỉ xét đến hệ thống máy PC thông dụng hiện nay.
- Miền giá trị của kiểu: Cho biết một đơn vị dữ liệu thuộc kiểu này sẽ có thể lấy

giá trị trong miền nào, ví dụ nhỏ nhất và lớn nhất là bao nhiêu. Hiển nhiên các giá trị này phụ thuộc vào số byte mà hệ thống máy qui định cho từng kiểu. NSD cần nhớ đến miền giá trị này để khai báo kiểu cho các biến cần sử dụng một cách thích hợp.

Dưới đây là bảng tóm tắt một số kiểu chuẩn đơn giản và các thông số của nó được sử dụng trong C++.

Loại dữ liệu	Tên kiểu	Số ô nhớ	Miền giá trị
Kí tự	char	1 byte	– 128 .. 127
	unsigned char	1 byte	0 .. 255
Số nguyên	int	2 byte	– 32768 .. 32767
	unsigned int	2 byte	0 .. 65535
	short	2 byte	– 32768 .. 32767
	long	4 byte	$-2^{15} .. 2^{15} - 1$
Số thực	float	4 byte	$\pm 10^{-37} . . \pm 10^{+38}$
	double	8 byte	$\pm 10^{-307} . . \pm 10^{+308}$

Bảng 1. Các loại kiểu đơn giản

Trong chương này chúng ta chỉ xét các loại kiểu đơn giản trên đây. Các loại kiểu có cấu trúc do người dùng định nghĩa sẽ được trình bày trong các chương sau.

2. Kiểu ký tự

Một kí tự là một kí hiệu trong bảng mã ASCII. Như đã biết một số kí tự có mặt chữ trên bàn phím (ví dụ các chữ cái, chữ số) trong khi một số kí tự lại không (ví dụ kí tự biểu diễn việc lùi lại một ô trong văn bản, kí tự chỉ việc kết thúc một dòng hay kết thúc một văn bản). Do vậy để biểu diễn một kí tự người ta dùng chính mã ASCII của kí tự đó trong bảng mã ASCII và thường gọi là giá trị của kí tự. Ví dụ phát biểu "Cho kí tự 'A'" là cũng tương đương với phát biểu "Cho kí tự 65" (65 là mã ASCII của kí tự 'A'), hoặc "Xoá kí tự xuống dòng" là cũng tương đương với phát biểu "Xoá kí tự 13" vì 13 là mã ASCII của kí tự xuống dòng.

Như vậy một biến kiểu kí tự có thể được nhận giá trị theo 2 cách tương đương - chữ hoặc giá trị số: ví dụ giả sử `c` là một biến kí tự thì câu lệnh gán `c = 'A'` cũng tương đương với câu lệnh gán `c = 65`. Tuy nhiên để sử dụng giá trị số của một kí tự `c` nào đó ta phải yêu cầu đổi `c` sang giá trị số bằng câu lệnh `int(c)`.

Theo bảng trên ta thấy có 2 loại kí tự là `char` với miền giá trị từ -128 đến 127 và

unsigned char (kí tự không dấu) với miền giá trị từ 0 đến 255. Trường hợp một biến được gán giá trị vượt ra ngoài miền giá trị của kiểu thì giá trị của biến sẽ được tính theo mã bù – (256 – c). Ví dụ nếu gán cho char c giá trị 179 (vượt khỏi miền giá trị đã được qui định của char) thì giá trị thực sự được lưu trong máy sẽ là – (256 – 179) = –77.

Ví dụ 1 :

```
char c, d ;                  // c, d được phép gán giá trị từ -128 đến 127
unsigned e ;                 // e được phép gán giá trị từ 0 đến 255
c = 65 ; d = 179 ;           // d có giá trị ngoài miền cho phép
e = 179; f = 330 ;           // f có giá trị ngoài miền cho phép
cout << c << int(c) ;        // in ra chữ cái 'A' và giá trị số 65
cout << d << int(d) ;        // in ra là kí tự '|' và giá trị số -77
cout << e << int(e)          // in ra là kí tự '|' và giá trị số 179
cout << f << int(f)          // in ra là kí tự 'J' và giá trị số 74
```

Chú ý: Qua ví dụ trên ta thấy một biến nếu được gán giá trị ngoài miền cho phép sẽ dẫn đến kết quả không theo suy nghĩ thông thường. Do vậy nên tuân thủ qui tắc chỉ gán giá trị cho biến thuộc miền giá trị mà kiểu của biến đó qui định. Ví dụ nếu muốn sử dụng biến có giá trị từ 128 .. 255 ta nên khai báo biến dưới dạng kí tự không dấu (unsigned char), còn nếu giá trị vượt quá 255 ta nên chuyển sang kiểu nguyên (int) chẳng hạn.

3. Kiểu số nguyên

Các số nguyên được phân chia thành 4 loại kiểu khác nhau với các miền giá trị tương ứng được cho trong bảng 1. Đó là kiểu số nguyên ngắn (short) tương đương với kiểu số nguyên (int) sử dụng 2 byte và số nguyên dài (long int) sử dụng 4 byte. Kiểu số nguyên thường được chia làm 2 loại có dấu (int) và không dấu (unsigned int hoặc có thể viết gọn hơn là unsigned). Qui tắc mã bù cũng được áp dụng nếu giá trị của biến vượt ra ngoài miền giá trị cho phép, vì vậy cần cân nhắc khi khai báo kiểu cho các biến. Ta thường sử dụng kiểu int cho các số nguyên trong các bài toán với miền giá trị vừa phải (có giá trị tuyệt đối bé hơn 32767), chẳng hạn các biến đếm trong các vòng lặp, ...

4. Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu float hoặc double mà miền giá trị của chúng được cho trong bảng 1. Các giá trị số kiểu double được gọi là số thực với độ chính xác gấp đôi vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu

float để đảm bảo số số lẻ sau một số thực có thể tăng lên đảm bảo tính chính xác cao hơn so với số kiểu float. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu float là đủ dùng.

Như đã nhắc đến trong phần các lệnh vào/ra ở chương 1, liên quan đến việc in ấn số thực ta có một vài cách thiết đặt dạng in theo ý muốn, ví dụ độ rộng tối thiểu để in một số hay số số lẻ thập phân cần in ...

Ví dụ 2 : Chương trình sau đây sẽ in diện tích và chu vi của một hình tròn có bán kính 2cm với 3 số lẻ.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
        float r = 2 ;                    // r là tên biến dùng để chứa bán kính
        cout << "Diện tích = " << setiosflags(ios::showpoint) ;
        cout << setprecision(3) << r * r * 3.1416 ;
        getch() ;
}
```

II. HẰNG - KHAI BÁO VÀ SỬ DỤNG HẰNG

Hằng là một giá trị cố định nào đó ví dụ 3 (hằng nguyên), 'A' (hằng kí tự), 5.0 (hằng thực), "Ha noi" (hằng xâu kí tự). Một giá trị có thể được hiểu dưới nhiều kiểu khác nhau, do vậy khi viết hằng ta cũng cần có dạng viết thích hợp.

1. Hằng nguyên

- kiểu short, int: 3, -7, ...
- kiểu unsigned: 3, 123456, ...
- kiểu long, long int: 3L, -7L, 123456L, ... (viết L vào cuối mỗi giá trị)

Các cách viết trên là thể hiện của số nguyên trong hệ thập phân, ngoài ra chúng còn được viết dưới các hệ đếm khác như hệ cơ số 8 hoặc hệ cơ số 16. Một số nguyên trong cơ số 8 luôn luôn được viết với số 0 ở đầu, tương tự với cơ số 16 phải viết với 0x ở đầu. Ví dụ ta biết 65 trong cơ số 8 là 101 và trong cơ số 16 là 41, do đó 3 cách viết 65, 0101, 0x41 là như nhau, cùng biểu diễn giá trị 65.

2. Hằng thực

Một số thực có thể được khai báo dưới dạng kiểu float hoặc double và các giá trị của nó có thể được viết dưới một trong hai dạng.

a. Dạng dấu phảy tĩnh

Theo cách viết thông thường. Ví dụ: 3.0, -7.0, 3.1416, ...

b. Dạng dấu phảy động

Tổng quát, một số thực x có thể được viết dưới dạng: men hoặc mEn, trong đó m được gọi là phần định trị, n gọi là phần bậc (hay mũ). Số men biểu thị giá trị $x = m \times 10^n$. Ví dụ số $\pi = 3.1416$ có thể được viết:

$\pi = \ldots = 0.031416e2 = 0.31416e1 = 3.1416e0 = 31.416e{-}1 = 314.16e{-}2 = \ldots$

vì $\pi = 0.031416 \times 10^2 = 0.31416 \times 10^1 = 3.1416 \times 10^0 = \ldots$

Như vậy một số x có thể được viết dưới dạng mEn với nhiều giá trị m, n khác nhau, phụ thuộc vào dấu phảy ngăn cách phần nguyên và phần thập phân của số. Do vậy cách viết này được gọi là dạng dấu phảy động.

3. Hằng kí tự

a. Cách viết hằng

Có 2 cách để viết một hằng kí tự. Đối với các kí tự có mặt chữ thể hiện ta thường sử dụng cách viết thông dụng đó là đặt mặt chữ đó giữa 2 dấu nháy đơn như: 'A', '3', ' ' (dấu cách) ... hoặc sử dụng trực tiếp giá trị số của chúng. Ví dụ các giá trị tương ứng của các kí tự trên là 65, 51 và 32. Với một số kí tự không có mặt chữ ta buộc phải dùng giá trị (số) của chúng, như viết 27 thay cho kí tự được nhấn bởi phím Escape, 13 thay cho kí tự được nhấn bởi phím Enter ...

Để biểu diễn kí tự bằng giá trị số ta có thể viết trực tiếp (không dùng cặp dấu nháy đơn) giá trị đó dưới dạng hệ số 10 (như trên) hoặc đặt chúng vào cặp dấu nháy đơn, trường hợp này chỉ dùng cho giá trị viết dưới dạng hệ 8 hoặc hệ 16 theo mẫu sau:

- '\kkk': không quá 3 chữ số trong hệ 8. Ví dụ '\11' biểu diễn kí tự có mã 9.
- '\xkk': không quá 2 chữ số trong hệ 16. Ví dụ '\x1B' biểu diễn kí tự có mã 27.

Tóm lại, một kí tự có thể có nhiều cách viết, chẳng hạn 'A' có giá trị là 65 (hệ 10) hoặc 101 (hệ 8) hoặc 41 (hệ 16), do đó kí tự 'A' có thể viết bởi một trong các dạng sau:

65, 0101, 0x41 hoặc 'A' , '\101' , '\x41'

Tương tự, dấu kết thúc xâu có giá trị 0 nên có thể viết bởi 0 hoặc '\0' hoặc '\x0', trong các cách này cách viết '\0' được dùng thông dụng nhất.

b. Một số hằng thông dụng

Đối với một số hằng kí tự thường dùng nhưng không có mặt chữ tương ứng, hoặc các kí tự được dành riêng với nhiệm vụ khác, khi đó thay vì phải nhớ giá trị của chúng ta có thể viết theo qui ước sau:

'\n'	:	biểu thị kí tự xuống dòng (cũng tương đương với endl)
'\t'	:	kí tự tab
'\a'	:	kí tự chuông (tức thay vì in kí tự, loa sẽ phát ra một tiếng 'bíp')
'\r'	:	xuống dòng
'\f'	:	kéo trang
'\\'	:	dấu \
'\?'	:	dấu chấm hỏi ?
'\''	:	dấu nháy đơn '
'\"'	:	dấu nháy kép "
'\kkk'	:	kí tự có mã là kkk trong hệ 8
'\xkk'	:	kí tự có mã là kk trong hệ 16

Ví dụ:

```
cout << "Hôm nay trời \t nắng \a \a \a \n" ;
```

sẽ in ra màn hình dòng chữ "Hôm nay trời" sau đó bỏ một khoảng cách bằng một tab (khoảng 8 dấu cách) rồi in tiếp chữ "nắng", tiếp theo phát ra 3 tiếng chuông và cuối cùng con trỏ trên màn hình sẽ nhảy xuống đầu dòng mới.

Do dấu cách (phím spacebar) không có mặt chữ, nên trong một số trường hợp để tránh nhầm lẫn chúng tôi qui ước sử dụng kí hiệu <> để biểu diễn dấu cách. Ví dụ trong giáo trình này dấu cách (có giá trị là 32) được viết ' ' (dấu nháy đơn bao một dấu cách) hoặc rõ ràng hơn bằng cách viết theo qui ước <>.

4. Hằng xâu kí tự

Là dãy kí tự bất kỳ đặt giữa cặp dấu nháy kép. Ví dụ: "Lớp K43*", "12A4", "A", "<>", "" là các hằng xâu kí tự, trong đó "" là xâu không chứa kí tự nào, các xâu "<>", "A" chứa 1 kí tự ... Số các kí tự giữa 2 dấu nháy kép được gọi là độ dài của xâu. Ví dụ xâu "" có độ dài 0, xâu "<>" hoặc "A" có độ dài 1 còn xâu "Lớp K43*" có độ dài 8.

Chú ý phân biệt giữa 2 cách viết 'A' và "A", tuy chúng cùng biểu diễn chữ cái A nhưng chương trình sẽ hiểu 'A' là một kí tự còn "A" là một xâu kí tự (do vậy chúng được bố trí khác nhau trong bộ nhớ cũng như cách sử dụng chúng là khác nhau). Tương tự ta không được viết " (2 dấu nháy đơn liền nhau) vì không có khái niệm kí tự

"rỗng". Để chỉ xâu rỗng (không có kí tự nào) ta phải viết "" (2 dấu nháy kép liền nhau).

Tóm lại một giá trị có thể được viết dưới nhiều kiểu dữ liệu khác nhau và do đó cách sử dụng chúng cũng khác nhau. Ví dụ liên quan đến khái niệm 3 đơn vị có thể có các cách viết sau tuy nhiên chúng hoàn toàn khác nhau:

- 3 : số nguyên 3 đơn vị
- 3L : số nguyên dài 3 đơn vị
- 3.0 : số thực 3 đơn vị
- '3' : chữ số 3
- "3" : xâu chứa kí tự duy nhất là 3

5. Khai báo hằng

Một giá trị cố định (hằng) được sử dụng nhiều lần trong chương trình đôi khi sẽ thuận lợi hơn nếu ta đặt cho nó một tên gọi, thao tác này được gọi là khai báo hằng. Ví dụ một chương trình quản lý sinh viên với giả thiết số sinh viên tối đa là 50. Nếu số sinh viên tối đa không thay đổi trong chương trình ta có thể đặt cho nó một tên gọi như SOSV chẳng hạn. Trong suốt chương trình bất kỳ chỗ nào xuất hiện giá trị 50 ta đều có thể thay nó bằng SOSV. Tương tự C++ cũng có những tên hằng được đặt sẵn, được gọi là các hằng chuẩn và NSD có thể sử dụng khi cần thiết. Ví dụ hằng π được đặt sẵn trong C++ với tên gọi M_PI. Việc sử dụng tên hằng thay cho hằng có nhiều điểm thuận lợi như sau:

- Chương trình dễ đọc hơn, vì thay cho các con số ít có ý nghĩa, một tên gọi sẽ làm NSD dễ hình dung vai trò, nội dung của nó. Ví dụ, khi gặp tên gọi SOSV NSD sẽ hình dung được chẳng hạn, "đây là số sinh viên tối đa trong một lớp", trong khi số 50 có thể là số sinh viên mà cũng có thể là tuổi của một sinh viên nào đó.
- Chương trình dễ sửa chữa hơn, ví dụ bây giờ nếu muốn thay đổi chương trình sao cho bài toán quản lý được thực hiện với số sinh viên tối đa là 60, khi đó ta cần tìm và thay thế hàng trăm vị trí xuất hiện của 50 thành 60. Việc thay thế như vậy dễ gây ra lỗi vì có thể không tìm thấy hết các số 50 trong chương trình hoặc thay nhầm số 50 với ý nghĩa khác như tuổi của một sinh viên nào đó chẳng hạn. Nếu trong chương trình sử dụng hằng SOSV, bây giờ việc thay thế trở nên chính xác và dễ dàng hơn bằng thao tác khai báo lại giá trị hằng SOSV bằng 60. Lúc đó trong chương trình bất kỳ nơi nào gặp tên hằng SOSV đều được chương trình hiểu với giá trị 60.

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng ;

hoặc:

const tên_hằng = giá_trị_hằng ;

Ví dụ:

```
#define sosv 50 ;
#define MAX 100 ;
const sosv = 50 ;
```

Như trên đã chú ý một giá trị hằng chưa nói lên kiểu sử dụng của nó vì vậy ta cần khai báo rõ ràng hơn bằng cách thêm tên kiểu trước tên hằng trong khai báo const, các hằng khai báo như vậy được gọi là hằng có kiểu.

Ví dụ:

```
const int sosv = 50 ;
const float nhiet_do_soi = 100.0 ;
```

III. BIẾN - KHAI BÁO VÀ SỬ DỤNG BIẾN

1. Khai báo biến

Biến là các tên gọi để lưu giá trị khi làm việc trong chương trình. Các giá trị được lưu có thể là các giá trị dữ liệu ban đầu, các giá trị trung gian tạm thời trong quá trình tính toán hoặc các giá trị kết quả cuối cùng. Khác với hằng, giá trị của biến có thể thay đổi trong quá trình làm việc bằng các lệnh đọc vào từ bàn phím hoặc gán. Hình ảnh cụ thể của biến là một số ô nhớ trong bộ nhớ được sử dụng để lưu các giá trị của biến.

Mọi biến phải được khai báo trước khi sử dụng. Một khai báo như vậy sẽ báo cho chương trình biết về một biến mới gồm có: tên của biến, kiểu của biến (tức kiểu của giá trị dữ liệu mà biến sẽ lưu giữ). Thông thường với nhiều NNLT tất cả các biến phải được khai báo ngay từ đầu chương trình hay đầu của hàm, tuy nhiên để thuận tiện C++ cho phép khai báo biến ngay bên trong chương trình hoặc hàm, có nghĩa bất kỳ lúc nào NSD thấy cần thiết sử dụng biến mới, họ có quyền khai báo và sử dụng nó từ đó trở đi.

Cú pháp khai báo biến gồm tên kiểu, tên biến và có thể có hay không khởi tạo giá trị ban đầu cho biến. Để khởi tạo hoặc thay đổi giá trị của biến ta dùng lệnh gán (=).

a. Khai báo không khởi tạo

```
tên_kiểu tên_biến_1 ;
tên_kiểu tên_biến_2 ;
```

```
tên_kiểu  tên_biến_3 ;
```

Nhiều biến cùng kiểu có thể được khai báo trên cùng một dòng:

```
tên_kiểu  tên_biến_1, tên_biến_2, tên_biến_3 ;
```

Ví dụ:

```
void main()
{
    int i, j ;                  // khai báo 2 biến i, j có kiểu nguyên
    float x ;                   // khai báo biến thực x
    char c, d[100] ;            // biến kí tự c, xâu d chứa tối đa 100 kí tự
    unsigned int u ;            // biến nguyên không dấu u
    …
}
```

b. Khai báo có khởi tạo

Trong câu lệnh khai báo, các biến có thể được gán ngay giá trị ban đầu bởi phép toán gán (=) theo cú pháp:

tên_kiểu tên_biến_1 = gt_1, tên_biến_2 = gt_2, tên_biến_3 = gt_3 ;

trong đó các giá trị gt_1, gt_2, gt_3 có thể là các hằng, biến hoặc biểu thức.

Ví dụ:

```
const int n = 10 ;
void main()
{
    int i = 2, j , k = n + 5;       // khai báo i và khởi tạo bằng 2, k bằng 15
    float eps = 1.0e-6 ;            // khai báo biến thực epsilon khởi tạo bằng 10^-6
    char c = 'Z';                   // khai báo biến kí tự c và khởi tạo bằng 'A'
    char d[100] = "Tin học";        // khai báo xâu kí tự d chứa dòng chữ "Tin học"
    …
}
```

2. Phạm vi của biến

Như đã biết chương trình là một tập hợp các hàm, các câu lệnh cũng như các khai báo. Phạm vi tác dụng của một biến là nơi mà biến có tác dụng, tức hàm nào, câu lệnh

nào được phép sử dụng biến đó. Một biến xuất hiện trong chương trình có thể được sử dụng bởi hàm này nhưng không được bởi hàm khác hoặc bởi cả hai, điều này phụ thuộc chặt chẽ vào vị trí nơi biến được khai báo. Một nguyên tắc đầu tiên là biến sẽ có tác dụng kể từ vị trí nó được khai báo cho đến hết khối lệnh chứa nó. Chi tiết cụ thể hơn sẽ được trình bày trong chương 4 khi nói về hàm trong C++.

3. Gán giá trị cho biến (phép gán)

Trong các ví dụ trước chúng ta đã sử dụng phép gán dù nó chưa được trình bày, đơn giản một phép gán mang ý nghĩa tạo giá trị mới cho một biến. Khi biến được gán giá trị mới, giá trị cũ sẽ được tự động xoá đi bất kể trước đó nó chứa giá trị nào (hoặc chưa có giá trị, ví dụ chỉ mới vừa khai báo xong). Cú pháp của phép gán như sau:

tên_biến = biểu thức ;

Khi gặp phép gán chương trình sẽ tính toán giá trị của biểu thức sau đó gán giá trị này cho biến. Ví dụ:

```
int n, i = 3;                    // khởi tạo i bằng 3
n = 10;                          // gán cho n giá trị 10
cout << n <<", " << i << endl;   // in ra: 10, 3
i = n / 2;                       // gán lại giá trị của i bằng n/2 = 5
cout << n <<", " << i << endl;   // in ra: 10, 5
```

Trong ví dụ trên n được gán giá trị bằng 10; trong câu lệnh tiếp theo biểu thức n/2 được tính (bằng 5) và sau đó gán kết quả cho biến i, tức i nhận kết quả bằng 5 dù trước đó nó đã có giá trị là 2 (trong trường hợp này việc khởi tạo giá trị 2 cho biến i là không có ý nghĩa).

Một khai báo có khởi tạo cũng tương đương với một khai báo và sau đó thêm lệnh gán cho biến (ví dụ int i = 3 cũng tương đương với 2 câu lệnh int i; i = 3) tuy nhiên về mặt bản chất khởi tạo giá trị cho biến vẫn khác với phép toán gán như ta sẽ thấy trong các phần sau.

4. Một số điểm lưu ý về phép gán

Với ý nghĩa thông thường của phép toán (nghĩa là tính toán và cho lại một giá trị) thì phép toán gán còn một nhiệm vụ nữa là trả lại một giá trị. Giá trị trả lại của phép toán gán chính là giá trị của biểu thức sau dấu bằng. Lợi dụng điều này C++ cho phép chúng ta gán "kép" cho nhiều biến nhận cùng một giá trị bởi cú pháp:

biến_1 = biến_2 = … = biến_n = gt ;

với cách gán này tất cả các biến sẽ nhận cùng giá trị gt. Ví dụ:

```
int i, j, k ;
i = j = k = 1;
```

Biểu thức gán trên có thể được viết lại như (i = (j = (k = 1))), có nghĩa đầu tiên để thực hiện phép toán gán giá trị cho biến i chương trình phải tính biểu thức (j = (k = 1)), tức phải tính k = 1, đây là phép toán gán, gán giá trị 1 cho k và trả lại giá trị 1, giá trị trả lại này sẽ được gán cho j và trả lại giá trị 1 để tiếp tục gán cho i.

Ngoài việc gán kép như trên, phép toán gán còn được phép xuất hiện trong bất kỳ biểu thức nào, điều này cho phép trong một biểu thức có phép toán gán, nó không chỉ tính toán mà còn gán giá trị cho các biến, ví dụ n = 3 + (i = 2) sẽ cho ta i = 2 và n = 5. Việc sử dụng nhiều chức năng của một câu lệnh làm cho chương trình gọn gàng hơn (trong một số trường hợp) nhưng cũng trở nên khó đọc, chẳng hạn câu lệnh trên có thể viết tách thành 2 câu lệnh i = 2; n = 3 + i; sẽ dễ đọc hơn ít nhất đối với các bạn mới bắt đầu tìm hiểu về lập trình.

IV. PHÉP TOÁN, BIỂU THỨC VÀ CÂU LỆNH

1. Phép toán

C++ có rất nhiều phép toán loại 1 ngôi, 2 ngôi và thậm chí cả 3 ngôi. Để hệ thống, chúng tôi tạm phân chia thành các lớp và trình bày chỉ một số trong chúng. Các phép toán còn lại sẽ được tìm hiểu dần trong các phần sau của giáo trình. Các thành phần tên gọi tham gia trong phép toán được gọi là hạng thức hoặc toán hạng, các kí hiệu phép toán được gọi là toán tử. Ví dụ trong phép toán a + b; a, b được gọi là toán hạng và + là toán tử. Phép toán 1 ngôi là phép toán chỉ có một toán hạng, ví dụ –a (đổi dấu số a), &x (lấy địa chỉ của biến x) … Một số kí hiệu phép toán cũng được sử dụng chung cho cả 1 ngôi lẫn 2 ngôi (hiển nhiên với ngữ nghĩa khác nhau), ví dụ kí hiệu – được sử dụng cho phép toán trừ 2 ngôi a – b, hoặc phép & còn được sử dụng cho phép toán lấy hội các bit (a & b) của 2 số nguyên a và b …

*a. Các phép toán số học: +, -, *, /, %*

- Các phép toán + (cộng), – (trừ), * (nhân) được hiểu theo nghĩa thông thường trong số học.
- Phép toán a / b (chia) được thực hiện theo kiểu của các toán hạng, tức nếu cả hai toán hạng là số nguyên thì kết quả của phép chia chỉ lấy phần nguyên, ngược lại nếu 1 trong 2 toán hạng là thực thì kết quả là số thực. Ví dụ:

```
13/5 = 2                              // do 13 và 5 là 2 số nguyên
13.0/5 = 13/5.0 = 13.0/5.0 = 2.6      // do có ít nhất 1 toán hạng là thực
```

– Phép toán a % b (lấy phần dư) trả lại phần dư của phép chia a/b, trong đó a và b là 2 số nguyên. Ví dụ:

```
13%5 = 3                    // phần dư của 13/5
5%13 = 5                    // phần dư của 5/13
```

b. Các phép toán tự tăng, giảm: i++, ++i, i--, --i

– Phép toán ++i và i++ sẽ cùng tăng i lên 1 đơn vị tức tương đương với câu lệnh i = i+1. Tuy nhiên nếu 2 phép toán này nằm trong câu lệnh hoặc biểu thức thì ++i khác với i++. Cụ thể ++i sẽ tăng i, sau đó i mới được tham gia vào tính toán trong biểu thức. Ngược lại i++ sẽ tăng i sau khi biểu thức được tính toán xong (với giá trị i cũ). Điểm khác biệt này được minh hoạ thông qua ví dụ sau, giả sử i = 3, j = 15.

Phép toán	Tương đương	Kết quả
i = ++j ; // tăng trước	j = j + 1 ; i = j ;	i = 16 , j = 16
i = j++ ; // tăng sau	i = j ; j = j + 1 ;	i = 15 , j = 16
j = ++i + 5 ;	i = i + 1 ; j = i + 5 ;	i = 4, j = 9
j = i++ + 5 ;	j = i + 5; i = i + 1;	i = 4, j = 8

Ghi chú: Việc kết hợp phép toán tự tăng giảm vào trong biểu thức hoặc câu lệnh (như ví dụ trong phần sau) sẽ làm chương trình gọn nhưng khó hiểu hơn.

c. Các phép toán so sánh và lôgic

Đây là các phép toán mà giá trị trả lại là đúng hoặc sai. Nếu giá trị của biểu thức là đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của 2 khái niệm "đúng", "sai". Mở rộng hơn C++ quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai".

- Các phép toán so sánh

  == (bằng nhau), != (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Hai toán hạng của các phép toán này phải cùng kiểu. Ví dụ:

```
3 == 3 hoặc 3 == (4 -1)     // nhận giá trị 1 vì đúng
3 == 5                      // = 0 vì sai
3 != 5                      // = 1
3 + (5 < 2)                 // = 3 vì 5<2 bằng 0
```

```
3 + (5 >= 2)                    // = 4 vì 5>=2 bằng 1
```

Chú ý: cần phân biệt phép toán gán (=) và phép toán so sánh (==). Phép gán vừa gán giá trị cho biến vừa trả lại giá trị bất kỳ (là giá trị của toán hạng bên phải), trong khi phép so sánh luôn luôn trả lại giá trị 1 hoặc 0.

- Các phép toán lôgic:

  && (và), || (hoặc), ! (không, phủ định)

Hai toán hạng của loại phép toán này phải có kiểu lôgic tức chỉ nhận một trong hai giá trị "đúng" (được thể hiện bởi các số nguyên khác 0) hoặc "sai" (thể hiện bởi 0). Khi đó giá trị trả lại của phép toán là 1 hoặc 0 và được cho trong bảng sau:

a	b	a && b	a || b	! a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Tóm lại:

– Phép toán "và" đúng khi và chỉ khi hai toán hạng cùng đúng

– Phép toán "hoặc" sai khi và chỉ khi hai toán hạng cùng sai

– Phép toán "không" (hoặc "phủ định") đúng khi và chỉ khi toán hạng của nó sai.

Ví dụ:

```
3 && (4 > 5)            // = 0 vì có hạng thức (4>5) sai
(3 >= 1) && (7)         // = 1 vì cả hai hạng thức cùng đúng
!1                      // = 0
! (4 + 3 < 7)           // = 1 vì (4+3<7) bằng 0
5 || (4 >= 6)           // = 1 vì có một hạng thức (5) đúng
(5 < !0) || (4 >= 6)    // = 0 vì cả hai hạng thức đều sai
```

Chú ý: việc đánh giá biểu thức được tiến hành từ trái sang phải và sẽ dừng khi biết kết quả mà không chờ đánh giá hết biểu thức. Cách đánh giá này sẽ cho những kết quả phụ khác nhau nếu trong biểu thức ta "tranh thủ" đưa thêm vào các phép toán tự tăng giảm. Ví dụ cho i = 2, j = 3, xét 2 biểu thức sau đây:

```
x = (++i < 4 && ++j > 5)        cho kết quả x = 0 , i = 3 , j = 4
```

```
y = (++j > 5 && ++i < 4)          cho kết quả y = 0 , i = 2 , j = 4
```

cách viết hai biểu thức là như nhau (ngoại trừ hoán đổi vị trí 2 toán hạng của phép toán &&). Với giả thiết i = 2 và j = 3 ta thấy cả hai biểu thức trên cùng nhận giá trị 0. Tuy nhiên các giá trị của i và j sau khi thực hiện xong hai biểu thức này sẽ có kết quả khác nhau. Cụ thể với biểu thức đầu vì ++i < 4 là đúng nên chương trình phải tiếp tục tính tiếp ++j > 5 để đánh giá được biểu thức. Do vậy sau khi đánh giá xong cả i và j đều được tăng 1 (i=3, j=4). Trong khi đó với biểu thức sau do ++j > 5 là sai nên chương trình có thể kết luận được toàn bộ biểu thức là sai mà không cần tính tiếp ++i < 4. Có nghĩa chương trình sau khi đánh giá xong ++j > 5 sẽ dừng và vì vậy chỉ có biến j được tăng 1, từ đó ta có i = 2, j = 4 khác với kết quả của biểu thức trên. Ví dụ này một lần nữa nhắc ta chú ý kiểm soát kỹ việc sử dụng các phép toán tự tăng giảm trong biểu thức và trong câu lệnh.

2. Các phép gán

- Phép gán thông thường: Đây là phép gán đã được trình bày trong mục trước.
- Phép gán có điều kiện:

biến = (điều_kiện) ? a: b ;

điều_kiện là một biểu thức logic, a, b là các biểu thức bất kỳ cùng kiểu với kiểu của biến. Phép toán này gán giá trị a cho biến nếu điều kiện đúng và b nếu ngược lại.

Ví dụ:

```
x = (3 + 4 < 7) ? 10: 20      // x = 20 vì 3+4<7 là sai
x = (3 + 4) ? 10: 20          // x = 10 vì 3+4 khác 0, tức điều kiện đúng
x = (a > b) ? a: b            // x = số lớn nhất trong 2 số a, b.
```

- Cách viết gọn của phép gán: Một phép gán dạng x = x @ a ; có thể được viết gọn dưới dạng x @= a trong đó @ là các phép toán số học, xử lý bit ... Ví dụ:

  thay cho viết x = x + 2 có thể viết x += 2;

  hoặc x = x/2 ; x = x*2 có thể được viết lại như x /= 2; x *= 2;

Cách viết gọn này có nhiều thuận lợi khi viết và đọc chương trình nhất là khi tên biến quá dài hoặc đi kèm nhiều chỉ số … thay vì phải viết hai lần tên biến trong câu lệnh thì chỉ phải viết một lần, điều này tránh viết lặp lại tên biến dễ gây ra sai sót. Ví dụ thay vì viết:

```
ngay_quoc_te_lao_dong = ngay_quoc_te_lao_dong + 365;
```

có thể viết gọn hơn bởi:

```
ngay_quoc_te_lao_dong += 365;
```

hoặc thay cho viết :

```
Luong[Nhanvien[3][2*i+1]] = Luong[Nhanvien[3][2*i+1]] * 290 ;
```

có thể được viết lại bởi:

```
Luong[Nhanvien[3][2*i+1]] *= 290;
```

3. Biểu thức

Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng, phép toán và cặp dấu () theo một qui tắc nhất định. Các toán hạng là hằng, biến, hàm. Biểu thức cung cấp một cách thức để tính giá trị mới dựa trên các toán hạng và toán tử trong biểu thức. Ví dụ:

```
(x + y) * 2 - 4 ;       3 - x + sqrt(y) ;  (-b + sqrt(delta)) / (2*a) ;
```

a. Thứ tự ưu tiên của các phép toán

Để tính giá trị của một biểu thức cần có một trật tự tính toán cụ thể và thống nhất. Ví dụ xét biểu thức x = 3 + 4 * 2 + 7

- nếu tính theo đúng trật tự từ trái sang phải, ta có x = ((3+4) * 2) + 7 = 21,
- nếu ưu tiên dấu + được thực hiện trước dấu *, x = (3 + 4) * (2 + 7) = 63,
- nếu ưu tiên dấu * được thực hiện trước dấu +, x = 3 + (4 * 2) + 7 = 18.

Như vậy cùng một biểu thức tính x nhưng cho 3 kết quả khác nhau theo những cách hiểu khác nhau. Vì vậy cần có một cách hiểu thống nhất dựa trên thứ tự ưu tiên của các phép toán, tức những phép toán nào sẽ được ưu tiên tính trước và những phép toán nào được tính sau ...

C++ qui định trật tự tính toán theo các mức độ ưu tiên như sau:

1. Các biểu thức trong cặp dấu ngoặc ()
2. Các phép toán 1 ngôi (tự tăng, giảm, lấy địa chỉ, lấy nội dung con trỏ …)
3. Các phép toán số học.
4. Các phép toán quan hệ, logic.
5. Các phép gán.

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, –, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái. Ví dụ. theo mức ưu tiên đã qui định, biểu thức tính x trong ví dụ trên sẽ được tính như x = 3 + (4 * 2) + 7 = 18.

Phần lớn các trường hợp muốn tính toán theo một trật tự nào đó ta nên sử dụng cụ thể các dấu ngoặc (vì các biểu thức trong dấu ngoặc được tính trước). Ví dụ:

- Để tính $\Delta = b^2 - 4ac$ ta viết delta = b * b – 4 * a * c ;
- Để tính nghiệm phương trình bậc 2: $x = \frac{-b+\sqrt{\Delta}}{2a}$ viết : x = –b + sqrt(delta) / 2*a; là sai vì theo mức độ ưu tiên x sẽ được tính như –b + ((sqrt(delta)/2) * a) (thứ tự tính sẽ là phép toán 1 ngôi đổi dấu –b, đến phép chia, phép nhân và cuối cùng là phép cộng). Để tính chính xác cần phải viết (–b + sqrt(delta)) / (2*a).
- Cho a = 1, b = 2, c = 3. Biểu thức a += b += c cho giá trị c = 3, b = 5, a = 6. Thứ tự tính sẽ là từ phải sang trái, tức câu lệnh trên tương đương với các câu lệnh sau:

```
a = 1 ; b = 2 ; c = 3 ;
b = b + c ;                    // b = 5
a = a + b ;                    // a = 6
```

Để rõ ràng, tốt nhất nên viết biểu thức cần tính trước trong các dấu ngoặc.

b. Phép chuyển đổi kiểu

Khi tính toán một biểu thức phần lớn các phép toán đều yêu cầu các toán hạng phải cùng kiểu. Ví dụ để phép gán thực hiện được thì giá trị của biểu thức phải có **cùng kiểu** với biến. Trong trường hợp kiểu của giá trị biểu thức khác với kiểu của phép gán thì hoặc là chương trình sẽ tự động chuyển kiểu giá trị biểu thức về thành kiểu của biến được gán (nếu được) hoặc sẽ báo lỗi. Do vậy khi cần thiết NSD phải sử dụng các câu lệnh để chuyển kiểu của biểu thức cho phù hợp với kiểu của biến.

- Chuyển kiểu tự động: về mặt nguyên tắc, khi cần thiết các kiểu có giá trị thấp sẽ được chương trình tự động chuyển lên kiểu cao hơn cho phù hợp với phép toán. Cụ thể phép chuyển kiểu có thể được thực hiện theo sơ đồ như sau:

  char $\leftrightarrow$ int $\rightarrow$ long int $\rightarrow$ float $\rightarrow$ double

Ví dụ:

```
int i = 3;
float f ;
f = i + 2;
```

trong ví dụ trên i có kiểu nguyên và vì vậy i+2 cũng có kiểu nguyên trong khi f có kiểu thực. Tuy vậy phép toán gán này là hợp lệ vì chương trình sẽ tự động chuyển kiểu

cuả i+2 (bằng 5) sang kiểu thực (bằng 5.0) rồi mới gán cho f.

– Ép kiểu: trong chuyển kiểu tự động, chương trình chuyển các kiểu từ thấp đến cao, tuy nhiên chiều ngược lại không thể thực hiện được vì nó có thể gây mất dữ liệu. Do đó nếu cần thiết NSD phải ra lệnh cho chương trình. Ví dụ:

```
int i;
float f = 3 ;          // tự động chuyển 3 thành 3.0 và gán cho f
i = f + 2 ;            // sai vì mặc dù f + 2 = 5 nhưng không gán được cho i
```

Trong ví dụ trên để câu lệnh i = f+2 thực hiện được ta phải ép kiểu của biểu thức f+2 về thành kiểu nguyên. Cú pháp tổng quát như sau:

```
(tên_kiểu)biểu_thức          // cú pháp cũ trong C
```

hoặc:

```
tên_kiểu(biểu_thức)          // cú pháp mới trong C++
```

trong đó tên_kiểu là kiểu cần được chuyển sang. Như vậy câu lệnh trên phải được viết lại:

```
i = int(f + 2) ;
```

khi đó f+2 (bằng 5.0) được chuyển thành 5 và gán cho i.

Dưới đây ta sẽ xét một số ví dụ về lợi ích của việc ép kiểu.

- Phép ép kiểu từ một số thực về số nguyên sẽ cắt bỏ tất cả phần thập phân của số thực, chỉ để lại phần nguyên. Như vậy để tính phần nguyên của một số thực x ta chỉ cần ép kiểu của x về thành kiểu nguyên, có nghĩa int(x) là phần nguyên của số thực x bất kỳ. Ví dụ để kiểm tra một số nguyên n có phải là số chính phương, ta cần tính căn bậc hai của n. Nếu căn bậc hai x của n là số nguyên thì n là số chính phương, tức nếu int(x) = x thì x nguyên và n là chính phương, ví dụ:

```
int n = 10 ;
float x = sqrt(n) ;          // hàm sqrt(n) trả lại căn bậc hai của số n
if (int(x) == x) cout << "n chính phương" ;
else cout << "n không chính phương" ;
```

- Để biết mã ASCII của một kí tự ta chỉ cần chuyển kí tự đó sang kiểu nguyên.

```
char c ;
cin >> c ;
cout << "Mã của kí tự vừa nhập là " << int(c) ;
```

Ghi chú: Xét ví dụ sau:

```
int i = 3 , j = 5 ;
float x ;
x = i / j * 10;              // x = 6 ?
cout << x ;
```

trong ví dụ này mặc dù x được khai báo là thực nhưng kết quả in ra sẽ là 0 thay vì 6 như mong muốn. Lý do là vì phép chia giữa 2 số nguyên i và j sẽ cho lại số nguyên, tức i/j = 3/5 = 0. Từ đó x = 0*10 = 0. Để phép chia ra kết quả thực ta cần phải ép kiểu hoặc i hoặc j hoặc cả 2 thành số thực, khi đó phép chia sẽ cho kết quả thực và x được tính đúng giá trị. Cụ thể câu lệnh x = i/j*10 được đổi thành:

```
x = float(i) / j * 10 ;              // đúng
x = i / float(j) * 10      ;         // đúng
x = float(i) / float(j) * 10 ;       // đúng
x = float(i/j) * 10 ;                // sai
```

Phép ép kiểu: x = float(i/j) * 10 ; vẫn cho kết quả sai vì trong dấu ngoặc phép chia i/j vẫn là phép chia nguyên, kết quả x vẫn là 0.

4. Câu lệnh và khối lệnh

Một **câu lệnh** trong C++ được thiết lập từ các từ khoá và các biểu thức … và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ vào/ra hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
cin >> x >> y ;
x = 3 + x ; y = (x = sqrt(x)) + 1 ;
cout << x ;
cout << y ;
```

Các câu lệnh được phép viết trên cùng một hoặc nhiều dòng. Một số câu lệnh được gọi là lệnh có cấu trúc, tức bên trong nó lại chứa dãy lệnh khác. Dãy lệnh này phải được bao giữa cặp dấu ngoặc {} và được gọi là *khối lệnh*. Ví dụ tất cả các lệnh trong một hàm (như hàm main()) luôn luôn là một khối lệnh. Một đặc điểm của khối lệnh là các biến được khai báo trong khối lệnh nào thì chỉ có tác dụng trong khối lệnh đó. Chi tiết hơn về các đặc điểm của lệnh và khối lệnh sẽ được trình bày trong các chương tiếp theo của giáo trình.

V. THƯ VIỆN CÁC HÀM TOÁN HỌC

Trong phần này chúng tôi tóm tắt một số các hàm toán học hay dùng. Các hàm này đều được khai báo trong file nguyên mẫu math.h.

1. Các hàm số học

- abs(x), labs(x), fabs(x) : trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
- pow(x, y) : hàm mũ, trả lại giá trị x lũy thừa y (x^y).
- exp(x) : hàm mũ, trả lại giá trị e mũ x (e^x).
- log(x), log10(x) : trả lại lôgarit cơ số e và lôgarit thập phân của x (lnx, logx) .
- sqrt(x) : trả lại căn bậc 2 của x.
- atof(s_number) : trả lại số thực ứng với số viết dưới dạng xâu kí tự s_number.

2. Các hàm lượng giác

- sin(x), cos(x), tan(x) : trả lại các giá trị sinx, cosx, tgx.

BÀI TẬP

1. Viết câu lệnh khai báo biến để lưu các giá trị sau:

- Tuổi của một người
- Số lượng cây trong thành phố
- Độ dài cạnh một tam giác
- Khoảng cách giữa các hành tinh
- Một chữ số
- Nghiệm x của phương trình bậc 1
- Một chữ cái
- Biệt thức Δ của phương trình bậc 2

2. Viết câu lệnh nhập vào 4 giá trị lần lượt là số thực, nguyên, nguyên dài và kí tự. In ra màn hình các giá trị này để kiểm tra.

3. Viết câu lệnh in ra màn hình các dòng sau (không kể các số thứ tự và dấu: ở đầu mỗi dòng)

1: Bộ Giáo dục và Đào tạo Cộng hoà xã hội chủ nghĩa Việt Nam

2:

3: Sở Giáo dục Hà Nội Độc lập - Tự do - Hạnh phúc

Chú ý: khoảng trống giữa chữ Đào tạo và Cộng hoà (dòng 1) là 2 tab. Dòng 2: để trống.

4. Viết chương trình nhập vào một kí tự. In ra kí tự đó và mã ascii của nó.

5. Viết chương trình nhập vào hai số thực. In ra hai số thực đó với 2 số lẻ và cách nhau 5 cột.

6. Nhập, chạy và giải thích kết quả đạt được của đoạn chương trình sau:

```
#include <iostream.h>
void main()
{
    char c1 = 200; unsigned char c2 = 200 ;
    cout << "c1 = " << c1 << ", c2 = " << c2 << "\n" ;
    cout << "c1+100 = " << c1+100 << ", c2+100 = " << c2+100 ;
}
```

7. Nhập a, b, c. In ra màn hình dòng chữ phương trình có dạng ax^2 + bx + c = 0, trong đó các giá trị a, b, c chỉ in 2 số lẻ (ví dụ với a = 5.141, b = –2, c = 0.8 in ra 5.14 x^2 –2.00 x + 0.80).

8. Viết chương trình tính và in ra giá trị các biểu thức sau với 2 số lẻ:

a. $\sqrt{3+\sqrt{3+\sqrt{3}}}$ b. $\dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2}}}$

9. Nhập a, b, c là các số thực. In ra giá trị của các biểu thức sau với 3 số lẻ:

a. $a^2 - 2b + ab/c$ c. $3a - b^3 - 2\sqrt{c}$

b. $\dfrac{b^2-4ac}{2a}$ d. $\sqrt{a^2/b-4a/bc+1}$

10. In ra tổng, tích, hiệu và thương của 2 số được nhập vào từ bàn phím.

11. In ra trung bình cộng, trung bình nhân của 3 số được nhập vào từ bàn phím.

12. Viết chương trình nhập cạnh, bán kính và in ra diện tích, chu vi của các hình: vuông, chữ nhật, tròn.

13. Nhập a, b, c là độ dài 3 cạnh của tam giác (chú ý đảm bảo tổng 2 cạnh phải lớn

hơn cạnh còn lại). Tính chu vi, diện tích, độ dài 3 đường cao, 3 đường trung tuyến, 3 đường phân giác, bán kính đường tròn nội tiếp, ngoại tiếp lần lượt theo các công thức sau:

C = 2p = a + b + c ; S = $\sqrt{p(p-a)(p-b)(p-c)}$;

$h_a = \frac{2S}{a}$; ma = $\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$; ga = $\frac{2}{b+c}\sqrt{bcp(p-a)}$;

$r = \frac{S}{p}$; $R = \frac{abc}{4S}$;

14. Tính diện tích và thể tích của hình cầu bán kính R theo công thức:

S = $4\pi R^2$; V = RS/3

15. Nhập vào 4 chữ số. In ra tổng của 4 chữ số này và chữ số hàng chục, hàng đơn vị của tổng (ví dụ 4 chữ số 3, 1, 8, 5 có tổng là 17 và chữ số hàng chục là 1 và hàng đơn vị là 7, cần in ra 17, 1, 7).

16. Nhập vào một số nguyên (có 4 chữ số). In ra tổng của 4 chữ số này và chữ số đầu, chữ số cuối (ví dụ số 3185 có tổng các chữ số là 17, đầu và cuối là 3 và 5, kết quả in ra là: 17, 3, 5).

17. Hãy nhập 2 số a và b. Viết chương trình đổi giá trị của a và b theo 2 cách:

- dùng biến phụ t: t = a; a = b; b = t;
- không dùng biến phụ: a = a + b; b = a - b; a = a - b;

In kết quả ra màn hình để kiểm tra.

18. Viết chương trình đoán số của người chơi đang nghĩ, bằng cách yêu cầu người chơi nghĩ một số, sau đó thực hiện một loạt các tính toán trên số đã nghĩ rồi cho biết kết quả. Máy sẽ in ra số mà người chơi đã nghĩ. (ví dụ yêu cầu người chơi lấy số đã nghĩ nhân đôi, trừ 4, bình phương, chia 2 và trừ 7 rồi cho biết kết quả, máy sẽ in ra số người chơi đã nghĩ).

19. Một sinh viên gồm có các thông tin: họ tên, tuổi, điểm toán (hệ số 2), điểm tin (hệ số 1). Hãy nhập các thông tin trên cho 2 sinh viên. In ra bảng điểm gồm các chi tiết nêu trên và điểm trung bình của mỗi sinh viên.

20. Một nhân viên gồm có các thông tin: họ tên, hệ số lương, phần trăm phụ cấp (theo lương) và phần trăm phải đóng BHXH. Hãy nhập các thông tin trên cho 2 nhân viên. In ra bảng lương gồm các chi tiết nêu trên và tổng số tiền cuối cùng mỗi nhân viên được nhận.

CHƯƠNG 3

CẤU TRÚC ĐIỀU KHIỂN VÀ DỮ LIỆU KIỂU MẢNG

Cấu trúc rẽ nhánh
Cấu trúc lặp
Mảng dữ liệu
Mảng hai chiều

I. CẤU TRÚC RẼ NHÁNH

Nói chung việc thực hiện chương trình là hoạt động tuần tự, tức thực hiện từng lệnh một từ câu lệnh bắt đầu của chương trình cho đến câu lệnh cuối cùng. Tuy nhiên, để việc lập trình hiệu quả hơn hầu hết các NNLT bậc cao đều có các câu lệnh rẽ nhánh và các câu lệnh lặp cho phép thực hiện các câu lệnh của chương trình không theo trình tự tuần tự như trong văn bản.

Phần này chúng tôi sẽ trình bày các câu lệnh cho phép rẽ nhánh như vậy. Để thống nhất mỗi câu lệnh được trình bày về cú pháp (tức cách viết câu lệnh), cách sử dụng, đặc điểm, ví dụ minh hoạ và một vài điều cần chú ý khi sử dụng lệnh.

1. Câu lệnh điều kiện if

a. Ý nghĩa

Một câu lệnh **if** cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào một điều kiện được viết trong câu lệnh là đúng hay sai. Nói cách khác câu lệnh **if** cho phép chương trình rẽ nhánh (chỉ thực hiện 1 trong 2 nhánh).

b. Cú pháp

– **if (điều kiện) { khối lệnh 1; } else { khối lệnh 2; }**

– **if (điều kiện) { khối lệnh 1; }**

Trong cú pháp trên câu lệnh if có hai dạng: có else và không có else. điều kiện là một biểu thức lôgic tức nó có giá trị đúng (khác 0) hoặc sai (bằng 0).

Khi chương trình thực hiện câu lệnh if nó sẽ tính biểu thức điều kiện. Nếu điều kiện đúng chương trình sẽ tiếp tục thực hiện các lệnh trong khối lệnh 1, ngược lại nếu

điều kiện sai chương trình sẽ thực hiện khối lệnh 2 (nếu có else) hoặc không làm gì (nếu không có else).

c. Đặc điểm

- Đặc điểm chung của các câu lệnh có cấu trúc là bản thân nó chứa các câu lệnh khác. Điều này cho phép các câu lệnh if có thể lồng nhau.
- Nếu nhiều câu lệnh if (có else và không else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Qui tắc là else sẽ đi với if gần nó nhất mà chưa được ghép cặp với else khác. Ví dụ câu lệnh

```
if (n>0) if (a>b) c = a;
else c = b;
```

là tương đương với

```
if (n>0) { if (a>b) c = a; else c = b;}
```

d. Ví dụ minh hoạ

Ví dụ 1 : Bằng phép toán gán có điều kiện có thể tìm số lớn nhất max trong 2 số a, b như sau: `max = (a > b) ? a: b ;`

hoặc max được tìm bởi dùng câu lệnh if:

```
if (a > b) max = a; else max = b;
```

Ví dụ 2 : Tính năm nhuận. Năm thứ n là nhuận nếu nó chia hết cho 4, nhưng không chia hết cho 100 hoặc chia hết 400. Chú ý: một số nguyên a là chia hết cho b nếu phần dư của phép chia bằng 0, tức a%b == 0.

```
#include <iostream.h>
void main()
{
    int nam;
    cout << “Nam = “ ; cin >> nam ;
    if (nam%4 == 0 && year%100 !=0 || nam%400 == 0)
        cout << nam << "la nam nhuan” ;
    else
        cout << nam << "la nam khong nhuan” ;
}
```

Ví dụ 3 : Giải phương trình bậc 2. Cho phương trình $ax^2 + bx + c = 0$ ($a \neq 0$), tìm x.

```
#include <iostream.h>                    // tệp chứa các phương thức vào/ra
#include <math.h>                        // tệp chứa các hàm toán học
void main()
{
      float a, b, c;                     // khai báo các hệ số
      float delta;
      float x1, x2;                      // 2 nghiem
      cout << “Nhap a, b, c:\n” ; cin >> a >> b >> c ;    // qui ước nhập a ≠ 0
      delta = b*b - 4*a*c ;
      if (delta < 0) cout << “ph. trình vô nghiệm\n” ;
      else if (delta==0) cout<<“ph. trình có nghiệm kép:" << -b/(2*a) << '\n';
      else
      {
            x1 = (-b+sqrt(delta))/(2*a);
            x2 = (-b-sqrt(delta))/(2*a);
            cout << “nghiem 1 = " << x1 << " và nghiem 2 = " << x2 ;
      }
}
```

Chú ý: do C++ quan niệm "đúng" là một giá trị khác 0 bất kỳ và "sai" là giá trị 0 nên thay vì viết if (x != 0) hoặc if (x == 0) ta có thể viết gọn thành if (x) hoặc if (!x) vì nếu (x != 0) đúng thì ta có x ≠ 0 và vì x ≠ 0 nên (x) cũng đúng. Ngược lại nếu (x) đúng thì x ≠ 0, từ đó (x != 0) cũng đúng. Tương tự ta dễ dàng thấy được (x == 0) là tương đương với (!x).

2. Câu lệnh lựa chọn switch

a. Ý nghĩa

Câu lệnh if cho ta khả năng được lựa chọn một trong hai nhánh để thực hiện, do đó nếu sử dụng nhiều lệnh if lồng nhau sẽ cung cấp khả năng được rẽ theo nhiều nhánh. Tuy nhiên trong trường hợp như vậy chương trình sẽ rất khó đọc, do vậy C++ còn cung cấp một câu lệnh cấu trúc khác cho phép chương trình có thể chọn một trong nhiều nhánh để thực hiện, đó là câu lệnh switch.

b. Cú pháp

```
switch (biểu thức điều khiển)
{
    case biểu_thức_1: dãy lệnh 1 ;
    case biểu_thức_2: dãy lệnh 2 ;
    case ……...: ..... ;
    case biểu_thức_n: dãy lệnh n ;
    default: dãy lệnh n+1;
}
```

- biểu thức điều khiển: phải có kiểu nguyên hoặc kí tự,
- các biểu_thức_i: được tạo từ các hằng nguyên hoặc kí tự,
- các dãy lệnh có thể rỗng. Không cần bao dãy lệnh bởi cặp dấu {},
- nhánh default có thể có hoặc không và vị trí của nó có thể nằm bất kỳ trong câu lệnh (giữa các nhánh case), không nhất thiết phải nằm cuối cùng.

c. Cách thực hiện

Để thực hiện câu lệnh **switch** đầu tiên chương trình tính giá trị của biểu thức điều khiển (btđk), sau đó so sánh kết quả của btđk với giá trị của các biểu_thức_i bên dưới lần lượt từ biểu thức đầu tiên (thứ nhất) cho đến biểu thức cuối cùng (thứ n), nếu giá trị của btđk bằng giá trị của biểu thức thứ i đầu tiên nào đó thì chương trình sẽ thực hiện dãy lệnh thứ i và tiếp tục thực hiện tất cả dãy lệnh còn lại (từ dãy lệnh thứ i+1) cho đến hết (gặp dấu ngoặc đóng } của lệnh switch). Nếu quá trình so sánh không gặp biểu thức (nhánh case) nào bằng với giá trị của btđk thì chương trình thực hiện dãy lệnh trong default và tiếp tục cho đến hết (sau default có thể còn những nhánh case khác). Trường hợp câu lệnh switch không có nhánh default và btđk không khớp với bất cứ nhánh case nào thì chương trình không làm gì, coi như đã thực hiện xong lệnh switch.

Nếu muốn lệnh switch chỉ thực hiện nhánh thứ i (khi btđk = biểu_thức_i) mà không phải thực hiện thêm các lệnh còn lại thì cuối dãy lệnh thứ i thông thường ta đặt thêm lệnh **break;** đây là lệnh cho phép thoát ra khỏi một lệnh cấu trúc bất kỳ.

d. Ví dụ minh hoạ

Ví dụ 1 : In số ngày của một tháng bất kỳ nào đó được nhập từ bàn phím.

```
int th;
cout << “Cho biết tháng cần tính: “ ; cin >> th ;
switch (th)
```

```
{
    case 1: case 3: case 5: case 7: case 8: case 10:
    case 12: cout << "tháng này có 31 ngày" ; break ;
    case 2: cout << "tháng này có 28 ngày" ; break;
    case 4: case 6: case 9:
    case 11: cout << "tháng này có 30 ngày" ; break;
    default: cout << "Bạn đã nhập sai tháng, không có tháng này" ;
}
```

Trong chương trình trên giả sử NSD nhập tháng là 5 thì chương trình bắt đầu thực hiện dãy lệnh sau case 5 (không có lệnh nào) sau đó tiếp tục thực hiện các lệnh còn lại, cụ thể là bắt đầu từ dãy lệnh trong case 7, đến case 12 chương trình gặp lệnh in kết quả "tháng này có 31 ngày", sau đó gặp lệnh break nên chương trình thoát ra khỏi câu lệnh switch (đã thực hiện xong). Việc giải thích cũng tương tự cho các trường hợp khác của tháng. Nếu NSD nhập sai tháng (ví dụ tháng nằm ngoài phạm vi 1..12), chương trình thấy th không khớp với bất kỳ nhánh case nào nên sẽ thực hiện câu lệnh trong default, in ra màn hình dòng chữ "Bạn đã nhập sai tháng, không có tháng này" và kết thúc lệnh.

Ví dụ 2 : Nhập 2 số a và b vào từ bàn phím. Nhập kí tự thể hiện một trong bốn phép toán: cộng, trừ, nhân, chia. In ra kết quả thực hiện phép toán đó trên 2 số a, b.

```
void main()
{
    float a, b, c ;                 // các toán hạng a, b và kết quả c
    char dau ;                      // phép toán được cho dưới dạng kí tự
    cout << "Hãy nhập 2 số a, b: " ; cin >> a >> b ;
    cout << "và dấu phép toán: " ; cin >> dau ;
    switch (dau)
    {
        case '+': c = a + b ; break ;
        case '–': c = a - b ; break ;
        case 'x': case '.': case '*': c = a * b ; break ;
        case ':': case '/': c = a / b ; break ;
    }
    cout << setiosflags(ios::showpoint) << setprecision(4) ;      // in 4 số lẻ
```

```
        cout << "Kết quả là: " << c ;
}
```

Trong chương trình trên ta chấp nhận các kí tự x, ., * thể hiện cho phép toán nhân và :, / thể hiện phép toán chia.

3. Câu lệnh nhảy goto

a. Ý nghĩa

Một dạng khác của rẽ nhánh là câu lệnh nhảy **goto** cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình. Nhãn là một tên gọi do NSD tự đặt theo các qui tắt đặt tên gọi. Lệnh goto thường được sử dụng để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này thường được sử dụng rất hạn chế.

b. Cú pháp

Goto <nhãn> ;

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng sau nhãn và dấu hai chấm (:).

c. Ví dụ minh hoạ

Ví dụ 3 : Nhân 2 số nguyên theo phương pháp Ấn độ.

Phương pháp Ấn độ cho phép nhân 2 số nguyên bằng cách chỉ dùng các phép toán nhân đôi, chia đôi và cộng. Các phép nhân đôi và chia đôi thực chất là phép toán dịch bit về bên trái (nhân) hoặc bên phải (chia) 1 bit. Đây là các phép toán cơ sở trong bộ xử lý, do vậy dùng phương pháp này sẽ làm cho việc nhân các số nguyên được thực hiện rất nhanh. Có thể tóm tắt phương pháp như sau: Giả sử cần nhân m với n. Kiểm tra m nếu lẻ thì cộng thêm n vào kq (đầu tiên kq được khởi tạo bằng 0), sau đó lấy m chia 2 và n nhân 2. Quay lại kiểm tra m và thực hiện như trên. Quá trình dừng khi không thể chia đôi m được nữa (m = 0), khi đó kq là kết quả cần tìm (tức kq = m*n). Để dễ hiểu phương pháp này chúng ta tiến hành tính trên ví dụ với các số m, n cụ thể. Giả sử m = 21 và n = 11. Các bước tiến hành được cho trong bảng dưới đây:

Bước	m (chia 2)	n (nhân 2)	kq (khởi tạo kq = 0)
1	21	11	m lẻ, cộng thêm 11 vào kq = 0 + 11 = 11
2	10	22	m chẵn, bỏ qua
3	5	44	m lẻ, cộng thêm 44 vào kq = 11 + 44 = 55

4	2	88	m chẵn, bỏ qua
5	1	176	m lẻ, cộng thêm 176 vào kq = 55 + 176 = 231
6	0		m = 0, dừng cho kết quả kq = 231

Sau đây là chương trình được viết với câu lệnh goto.

```
void main()
{
    long m, n, kq = 0;              // Các số cần nhân và kết quả kq
    cout << "Nhập m và n: " ; cin >> m >> n ;
    lap:                            // đây là nhãn để chương trình quay lại
    if (m%2) kq += n;               // nếu m lẻ thì cộng thêm n vào kq
    m = m >> 1;                     // dịch m sang phải 1 bit tức m = m / 2
    n = n << 1;                     // dịch m sang trái 1 bit tức m = m * 2
    if (m) goto lap;                // quay lại nếu m ≠ 0
    cout << "m nhân n =" << kq ;
}
```

II. CẤU TRÚC LẶP

Một trong những cấu trúc quan trọng của lập trình cấu trúc là các câu lệnh cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình. Chẳng hạn trong ví dụ về bài toán nhân theo phương pháp Ấn độ, để lặp lại một đoạn lệnh chúng ta đã sử dụng câu lệnh goto. Tuy nhiên như đã lưu ý việc dùng nhiều câu lệnh này làm chương trình rất khó đọc. Do vậy cần có những câu lệnh khác trực quan hơn và thực hiện các phép lặp một cách trực tiếp. C++ cung cấp cho chúng ta 3 lệnh lặp như vậy. Về thực chất 3 lệnh này là tương đương (cũng như có thể dùng goto thay cho cả 3 lệnh lặp này), tuy nhiên để chương trình viết được sáng sủa, rõ ràng, C++ đã cung cấp nhiều phương án cho NSD lựa chọn câu lệnh khi viết chương trình phù hợp với tính chất lặp. Mỗi bài toán lặp có một đặc trưng riêng, ví dụ lặp cho đến khi đã đủ số lần định trước thì dừng hoặc lặp cho đến khi một điều kiện nào đó không còn thoả mãn nữa thì dừng … việc sử dụng câu lệnh lặp phù hợp sẽ làm cho chương trình dễ đọc và dễ bảo trì hơn. Đây là ý nghĩa chung của các câu lệnh lặp, do vậy trong các trình bày về câu lệnh tiếp theo sau đây chúng ta sẽ không cần phải trình bày lại ý nghĩa của chúng.

1. Lệnh lặp for

a. Cú pháp

for (dãy biểu thức 1 ; điều kiện lặp ; dãy biểu thức 2) { khối lệnh lặp; }

- Các biểu thức trong các dãy biểu thức 1, 2 cách nhau bởi dấu phảy (,). Có thể có nhiều biểu thức trong các dãy này hoặc dãy biểu thức cũng có thể trống.
- Điều kiện lặp: là biểu thức lôgic (có giá trị đúng, sai).
- Các dãy biểu thức và/hoặc điều kiện có thể trống tuy nhiên vẫn giữ lại các dấu chấm phảy (;) để ngăn cách các thành phần với nhau.

b. Cách thực hiện

Khi gặp câu lệnh **for** trình tự thực hiện của chương trình như sau:

- Thực hiện dãy biểu thức 1 (thông thường là các lệnh khởi tạo cho một số biến),
- Kiểm tra điều kiện lặp, nếu đúng thì thực hiện khối lệnh lặp → thực hiện dãy biểu thức 2 → quay lai kiểm tra điều kiện lặp và lặp lại quá trình trên cho đến bước nào đó việc kiểm tra điều kiện lặp cho kết quả sai thì dừng.

Tóm lại, biểu thức 1 sẽ được thực hiện 1 lần duy nhất ngay từ đầu quá trình lặp sau đó thực hiện các câu lệnh lặp và dãy biểu thức 2 cho đến khi nào không còn thoả điều kiện lặp nữa thì dừng.

c. Ví dụ minh hoạ

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ấn độ

```
void main()
{
    long m, n, kq;                    // Các số cần nhân và kết quả kq
    cout << “Nhập m và n: “ ; cin >> m >> n ;
    for (kq = 0 ; m ; m >>= 1, n <<= 1) if (m%2) kq += n ;
    cout << “m nhân n =” << kq ;
}
```

So sánh ví dụ này với ví dụ dùng goto ta thấy chương trình được viết rất gọn. Để bạn đọc dễ hiểu câu lệnh for, một lần nữa chúng ta nhắc lại cách hoạt động của nó thông qua ví dụ này, trong đó các thành phần được viết trong cú pháp là như sau:

- Dãy biểu thức 1: kq = 0,
- Điều kiện lặp: m. Ở đây điều kiện là đúng nếu $m \neq 0$ và sai nếu m = 0.

– Dãy biểu thức 2: m >>= 1 và n <<= 1. 2 biểu thức này có nghĩa m = m >> 1 (tương đương với m = m / 2) và n = n << 1 (tương đương với n = n * 2).

– Khối lệnh lặp: chỉ có một lệnh duy nhất if (m%2) kq += n ; (nếu phần dư của m chia 2 là khác 0, tức m lẻ thì cộng thêm n vào kq).

Cách thực hiện của chương trình như sau:

- Đầu tiên thực hiện biểu thức 1 tức gán kq = 0. Chú ý rằng nếu kq đã được khởi tạo trước bằng 0 trong khi khai báo (giống như trong ví dụ 6) thì thành phần biểu thức 1 ở đây có thể để trống (nhưng vẫn giữ lại dấu ; để phân biệt với các thành phần khác).
- Kiểm tra điều kiện: giả sử m ≠ 0 (tức điều kiện đúng) for sẽ thực hiện lệnh lặp tức kiểm tra nếu m lẻ thì cộng thêm n vào cho kq.
- Quay lại thực hiện các biểu thức 2 tức chia đôi m và nhân đôi n và vòng lặp được tiếp tục lại bắt đầu bằng việc kiểm tra m …
- Đến một bước lặp nào đó m sẽ bằng 0 (vì bị chia đôi liên tiếp), điều kiện không thoả, vòng lặp dừng và cho ta kết quả là kq.

Ví dụ 2 : Tính tổng của dãy các số từ 1 đến 100.

Chương trình dùng một biến đếm i được khởi tạo từ 1, và một biến kq để chứa tổng. Mỗi bước lặp chương trình cộng i vào kq và sau đó tăng i lên 1 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua 100. Khi i lớn hơn 100 chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int i, kq = 0;
    for (i = 1 ; i <= 100 ; i ++) kq += i ;
    cout << "Tổng = " << kq;
}
```

Ví dụ 3 : In ra màn hình dãy số lẻ bé hơn một số n nào đó được nhập vào từ bàn phím.

Chương trình dùng một biến đếm i được khởi tạo từ 1, mỗi bước lặp chương trình sẽ in i sau đó tăng i lên 2 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt qua n. Khi i lớn hơn n chương trình dừng. Sau đây là văn bản chương trình.

```
void main()
{
    int n, i ;
```

```
    cout << "Hãy nhập n = " ; cin >> n ;
    for (i = 1 ; i < n ; i += 2) cout << i << '\n' ;
}
```

d. Đặc điểm

Thông qua phần giải thích cách hoạt động của câu lệnh for trong ví dụ 7 có thể thấy các thành phần của for có thể để trống, tuy nhiên các dấu chấm phẩy vẫn giữ lại để ngăn cách các thành phần với nhau. Ví dụ câu lệnh for (kq = 0 ; m ; m >>= 1, n <<= 1) if (m%2) kq += n ; trong ví dụ 7 có thể được viết lại như sau:

```
kq = 0;
for ( ; m ; ) { if (m%2) kq += n; m >>= 1; n <<= 1; }
```

Tương tự, câu lệnh for (i = 1 ; i <= 100 ; i ++) kq += i ; trong ví dụ 8 cũng có thể được viết lại như sau:

```
i = 1;
for ( ; i <= 100 ; ) kq += i ++ ;
```

(câu lệnh kq += i++; được thực hiện theo 2 bước: cộng i vào kq và tăng i (tăng sau)).

Trong trường hợp điều kiện trong for cũng để trống chương trình sẽ ngầm định là điều kiện luôn luôn đúng, tức vòng lặp sẽ lặp vô hạn lần (!). Trong trường hợp này để dừng vòng lặp trong khối lệnh cần có câu lệnh kiểm tra dừng và câu lệnh break.

Ví dụ câu lệnh for (i = 1 ; i <= 100 ; i ++) kq += i ; được viết lại như sau:

```
i = 1;
for ( ; ; )
{
    kq += i++;
    if (i > 100) break;
}
```

Tóm lại, việc sử dụng dạng viết nào của for phụ thuộc vào thói quen của NSD, tuy nhiên việc viết đầy đủ các thành phần của for làm cho việc đọc chương trình trở nên dễ dàng hơn.

e. Lệnh for lồng nhau

Trong dãy lệnh lặp có thể chứa cả lệnh for, tức các lệnh for cũng được phép lồng nhau như các câu lệnh có cấu trúc khác.

Ví dụ 4 : Bài toán cổ: vừa gà vừa chó bó lại cho tròn đếm đủ 100 chân. Hỏi có mấy gà

và mấy con chó, biết tổng số con là 36.

Để giải bài toán này ta gọi g là số gà và c là số chó. Theo điều kiện bài toán ta thấy g có thể đi từ 0 (không có con nào) và đến tối đa là 50 (vì chỉ có 100 chân), tương tự c có thể đi từ 0 đến 25. Như vậy ta có thể cho g chạy từ 0 đến 50 và với mỗi giá trị cụ thể của g lại cho c chạy từ 0 đến 25, lần lượt với mỗi cặp (g, c) cụ thể đó ta kiểm tra 2 điều kiện: g + c == 36 ? (số con) và 2g + 4c == 100 ? (số chân). Nếu cả 2 điều kiện đều thoả thì cặp (g, c) cụ thể đó chính là nghiệm cần tìm. Từ đó ta có chương trình với 2 vòng for lồng nhau, một vòng for cho g và một vòng cho c.

```
void main()
{
    int g, c ;
    for (g = 0 ; g <= 50 ; g++)
    for (c = 0 ; c <= 25 ; c++)
    if (g+c == 36 && 2*g+4*c == 100) cout << "gà=" << g << ", chó=" << c ;
}
```

Chương trình trên có thể được giải thích một cách ngắn gọn như sau: Đầu tiên cho g = 0, thực hiện lệnh for bên trong tức lần lượt cho c = 0, 1, …, 25, với c=0 và g=0 kiểm tra điều kiện, nếu thoả thì in kết quả nếu không thì bỏ qua, quay lại tăng c, cho đến khi nào c>25 thì kết thúc vòng lặp trong quay về vòng lặp ngoài tăng g lên 1, lại thực hiện vòng lặp trong với g=1 này (tức lại cho c chạy từ 0 đến 25). Khi g của vòng lặp ngoài vượt quá 50 thì dừng. Từ đó ta thấy số vòng lặp của chương trình là 50 x 25 = 1000 lần lặp.

Chú ý: Có thể giảm bớt số lần lặp bằng nhận xét số gà không thể vượt quá 36 (vì tổng số con là 36). Một vài nhận xét khác cũng có thể làm giảm số vòng lặp, tiết kiệm thời gian chạy của chương trình. Bạn đọc tự nghĩ thêm các phương án giải khác để giảm số vòng lặp đến ít nhất.

Ví dụ 5 : Tìm tất cả các phương án để có 100đ từ các tờ giấy bạc loại 10đ, 20đ và 50đ.

```
main()
{
    int t10, t20, t50;                    // số tờ 10đ, 20đ, 50đ
    sopa = 0;                             // số phương án
    for (t10 = 0 ; t10 <= 10 ; t10++)
    for (t20 = 0 ; t20 <= 5 ; t20++)
    for (t50 = 0 ; t50 <= 2 ; t50++)
```

```
        if (t10*10 + t20*20 + t50*50 == 100)          // nếu thoả thì
        {
              sopa++;                                 // tăng số phương án
              if (t10) cout << t10 << "tờ 10đ " ;     // in số tờ 10đ nếu ≠ 0
              if (t20) cout << "+" << t20 << "tờ 20đ " ;  // thêm số tờ 20đ nếu≠0
              if (t50) cout << "+" << t50 << "tờ 50đ " ;  // thêm số tờ 50đ nếu≠0
              cout << '\n' ;                          // xuống dòng
        }
        cout << "Tong so phuong an = " << sopa ;
  }
```

2. Lệnh lặp while

a. Cú pháp

while (điều kiện) { khối lệnh lặp ; }

b. Thực hiện

Khi gặp lệnh while chương trình thực hiện như sau: đầu tiên chương trình sẽ kiểm tra điều kiện, nếu đúng thì thực hiện khối lệnh lặp, sau đó quay lại kiểm tra điều kiện và tiếp tục. Nếu điều kiện sai thì dừng vòng lặp. Tóm lại có thể mô tả một cách ngắn gọn về câu lệnh while như sau: *lặp lại các lệnh trong khi điều kiện vẫn còn đúng*.

c. Đặc điểm

- Khối lệnh lặp có thể không được thực hiện lần nào nếu điều kiện sai ngay từ đầu.
- Để vòng lặp không lặp vô hạn thì trong khối lệnh thông thường phải có ít nhất một câu lệnh nào đó gây ảnh hưởng đến kết quả của điều kiện, ví dụ làm cho điều kiện đang đúng trở thành sai.
- Nếu điều kiện luôn luôn nhận giá trị đúng (ví dụ biểu thức điều kiện là 1) thì trong khối lệnh lặp phải có câu lệnh kiểm tra dừng và lệnh break.

d. Ví dụ minh hoạ

Ví dụ 1 : Nhân 2 số nguyên theo phương pháp Ấn độ

```
void main()
{
```

```
        long m, n, kq;                  // Các số cần nhân và kết quả kq
        cout << “Nhập m và n: “ ; cin >> m >> n ;
        kq = 0 ;
        while (m)
        {
            if (m%2) kq += n ;
            m >>= 1;
            n <<= 1;
        }
        cout << “m nhân n =” << kq ;
    }
```

Trong chương trình trên câu lệnh while (m) … được đọc là "trong khi m còn khác 0 thực hiện …", ta thấy trong khối lệnh lặp có lệnh m >>= 1, lệnh này sẽ ảnh hưởng đến điều kiện (m), đến lúc nào đó m bằng 0 tức (m) là sai và chương trình sẽ dừng lặp.

Câu lệnh while (m) … cũng có thể được thay bằng while (1) … như sau:

```
    void main()
    {
        long m, n, kq;                  // Các số cần nhân và kết quả kq
        cout << “Nhập m và n: “ ; cin >> m >> n ;
        kq = 0 ;
        while (1) {
            if (m%2) kq += n ;
            m >>= 1;
            n <<= 1;
            if (!m) break ;             // nếu m = 0 thì thoát khỏi vòng lặp
        }
        cout << “m nhân n =” << kq ;
    }
```

Ví dụ 2 : Bài toán cổ: vừa gà vừa chó bó lại cho tròn đếm dủ 100 chân. Hỏi có mấy gà và mấy con chó, biết tổng số con là 36.

```
    void main()
```

```
{
    int g, c ;
    g = 0 ;
    while (g <= 36) {
        c = 0 ;
        while (c <= 50) {
            if (g + c == 36 && 2*g + 4*c == 100) cout << g << c ;
            c++;
        }
        g++;
    }
}
```

Ví dụ 3 : Tìm ước chung lớn nhất (UCLN) của 2 số nguyên m và n.

Áp dụng thuật toán Euclide bằng cách liên tiếp lấy số lớn trừ đi số nhỏ khi nào 2 số bằng nhau thì đó là UCLN. Trong chương trình ta qui ước m là số lớn và n là số nhỏ. Thêm biến phụ r để tính hiệu của 2 số. Sau đó đặt lại m hoặc n bằng r sao cho m > n và lặp lại. Vòng lặp dừng khi m = n.

```
void main()
{
    int m, n, r;
    cout << "Nhập m, n: " ; cin >> m >> n ;
    if (m < n) { int t = m; m = n; n = t; }      // nếu m < n thì đổi vai trò hai số
    while (m != n) {
        r = m - n ;
        if (r > n) m = r; else { m = n ; n = r ; }
    }
    cout << "UCLN = " << m ;
}
```

Ví dụ 4 : Tìm nghiệm xấp xỉ của phương trình $e^x - 1.5 = 0$, trên đoạn [0, 1] với độ chính xác 10^{-6} bằng phương pháp chia đôi.

Để viết chương trình này chúng ta nhắc lại phương pháp chia đôi. Cho hàm f(x) liên tục và đổi dấu trên một đoạn [a, b] nào đó (tức f(a), f(b) trái dấu nhau hay f(a)*f(b)

< 0). Ta đã biết với điều kiện này chắc chắn đồ thị của hàm f(x) sẽ cắt trục hoành tại một điểm x_0 nào đó trong đoạn [a, b], tức x_0 là nghiệm của phương trình f(x) = 0. Tuy nhiên việc tìm chính xác x_0 là khó, vì vậy ta có thể tìm xấp xỉ x' của nó sao cho x' càng gần x_0 càng tốt. Lấy c là điểm giữa của đoạn [a, b], c sẽ chia đoạn [a, b] thành 2 đoạn con [a, c] và [c, b] và do f(a), f(b) trái dấu nên chắc chắn một trong hai đoạn con cũng phải trái dấu, tức nghiệm x_0 sẽ nằm trong đoạn này. Tiếp tục quá trình bằng cách chia đôi đoạn vừa tìm được … cho đến khi ta nhận được một đoạn con (trái dấu, chứa x_0) sao cho độ dài của đoạn con này bé hơn độ xấp xỉ cho trước thì dừng. Khi đó lấy bất kỳ điểm nào trên đoạn con này (ví dụ hai điểm mút hoặc điểm giữa của a và b) thì chắc chắn khoảng cách của nó đến x_0 cũng bé hơn độ xấp xỉ cho trước, tức có thể lấy điểm này làm nghiệm xấp xỉ của phương trình f(x) = 0.

Trong ví dụ này hàm f(x) chính là e^x - 1.5 và độ xấp xỉ là 10^{-6}. Đây là hàm liên tục trên toàn trục số và đổi dấu trên đoạn [0, 1] (vì f(0) = 1 – 1.5 < 0 còn f(1) = e - 1.5 > 0). Sau đây là chương trình.

```
void main()
{
    float a = 0, b = 1, c;              // các điểm mút a, b và điểm giữa c
    float fa, fc;                       // giá trị của f(x) tại các điểm a, c
    while (b-a > 1.0e-6)                // trong khi độ dài đoạn còn lớn hơn ε
    {
        c = (a + b)/2;                          // tìm điểm c giữa đoạn [a,b]
        fa = exp(a) - 1.5; fc = exp(c) - 1.5;   // tính f(a) và f(c)
        if (fa*fc == 0) break;                  // f(c) = 0 tức c là nghiệm
        if (fa*fc > 0) a = c; else b = c;
    }
    cout << "Nghiem xap xi cua phuong trinh = " << c ;
}
```

Trong chương trình trên câu lệnh if (fa*fc > 0) a = c; else b = c; dùng để kiểm tra f(a) và f(c), nếu cùng dấu (f(a)*f(c) > 0) thì hàm f(x) phải trái dấu trên đoạn con [c, b] do đó đặt lại đoạn này là [a, b] (để quay lại vòng lặp) tức đặt a = c và b giữ nguyên, ngược lại nếu hàm f(x) trái dấu trên đoạn con [a, c] thì đặt lại b = c còn a giữ nguyên. Sau đó vòng lặp quay lại kiểm tra độ dài đoạn [a, b] (mới) nếu đã bé hơn độ xấp xỉ thì dừng và lấy c làm nghiệm xấp xỉ, nếu không thì tính lại c và tiếp tục quá trình.

Để tính f(a) và f(c) chương trình đã sử dụng hàm exp(x), đây là hàm cho lại kết quả e^x, để dùng hàm này hoặc các hàm toán học nói chung, cần khai báo file nguyên

mẫu math.h.

3. Lệnh lặp do ... while

a. Cú pháp

do { khối lệnh lặp } while (điều kiện) ;

b. Thực hiện

Đầu tiên chương trình sẽ thực hiện khối lệnh lặp, tiếp theo kiểm tra điều kiện, nếu điều kiện còn đúng thì quay lại thực hiện khối lệnh và quá trình tiếp tục cho đến khi điều kiện trở thành sai thì dừng.

c. Đặc điểm

Các đặc điểm của câu lệnh do … while cũng giống với câu lệnh lặp while trừ điểm khác biệt, đó là khối lệnh trong do … while sẽ được thực hiện ít nhất một lần, trong khi trong câu lệnh while có thể không được thực hiện lần nào (vì lệnh while phải kiểm tra điều kiện trước khi thực hiện khối lệnh, do đó nếu điều kiện sai ngay từ đầu thì lệnh sẽ dừng, khối lệnh không được thực hiện lần nào. Trong khi đó lệnh do … while sẽ thực hiện khối lệnh rồi mới kiểm tra điều kiện lặp để cho phép thực hiện tiếp hoặc dừng).

d. Ví dụ minh hoạ

Ví dụ 1 : Tính xấp xỉ số pi theo công thức Euler $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{n^2}$, với $\frac{1}{n^2} < 10^{-6}$.

```
void main()
{
    int n = 1; float S = 0;
    do S += 1.0/(n*n) while 1.0/(n*n) < 1.0e-6;
    float pi = sqrt(6*S);
    cout << "pi = " << pi ;
}
```

Ví dụ 2 : Kiểm tra một số n có là số nguyên tố.

Để kiểm tra một số n > 3 có phải là số nguyên tố ta lần lượt chia n cho các số i đi

từ 2 đến một nửa của n. Nếu có i sao cho n chia hết cho i thì n là hợp số ngược lại n là số nguyên tố.

```
void main()
{
    int i, n ;                                  // n: số cần kiểm tra
    cout << "Cho biết số cần kiểm tra: " ; cin >> n ;
    i = 2 ;
    do {
        if (n%i == 0) {
            cout << n << "là hợp số" ;
            return ;                            // dừng chương trình
        }
        i++;
    } while (i <= n/2);
    cout << n << "là số nguyên tố" ;
}
```

Ví dụ 3 : Nhập dãy kí tự và thống kê các loại chữ hoa, thường, chữ số và các loại khác còn lại đến khi gặp ENTER thì dừng.

```
void main()
{
    char c;                                     // kí tự dùng cho nhập
    int n1, n2, n3, n4 ;                        // số lượng các loại kí tự
    n1 = n2 = n3 = n4 = 0;
    cout << “Hãy nhập dãy kí tự: \n” ;
    do
    {
        cin >> c;
        if (‘a’ <= c && c <= ‘z’) n1++;         // nếu c là chữ thường thì tăng n1
        else if (‘A’ <= c && c <= ‘Z’) n2++;    // chữ hoa, tăng n2
        else if (‘0’ <= c && c <= ‘9’) n3++;    // chữ số, tăng n3
        else n4++;                              // loại khác, tăng n4
```

```
            cout << n1 << n2 << n3 << n4 ;          // in kết quả
        } while (c != 10) ;                         // còn lặp khi c còn khác kí tự ↵
    }
```

4. Lối ra của vòng lặp: break, continue

a. Lệnh break

Công dụng của lệnh dùng để thoát ra khỏi (chấm dứt) các câu lệnh cấu trúc, chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp sau câu lệnh vừa thoát. Các ví dụ minh hoạ bạn đọc có thể xem lại trong các ví dụ về câu lệnh switch, for, while.

b. Lệnh continue

Lệnh dùng để quay lại đầu vòng lặp mà không chờ thực hiện hết các lệnh trong khối lệnh lặp.

Ví dụ 1 : Giả sử với mỗi i từ 1 đến 100 ta cần thực hiện một loạt các lệnh nào đó trừ những số i là số chính phương. Như vậy để tiết kiệm thời gian, vòng lặp sẽ kiểm tra nếu i là số chính phương thì sẽ quay lại ngay từ đầu để thực hiện với i tiếp theo.

```
int i ;
for (i = 1; i <= 100; i++) {
    if (i là số chính phương) continue;
    {                                       // dãy lệnh khác
        .
        .
        .
    }
}
```

(Để kiểm tra i có là số chính phương chúng ta so sánh căn bậc hai của i với phần nguyên của nó. Nếu hai số này bằng nhau thì i là số chính phương. Cụ thể nếu sqrt(i) = int(sqrt(i)) thì i là số chính phương. Ở đây sqrt(x) là hàm trả lại căn bậc hai của x. Để sử dụng hàm này cần phải khai báo file nguyên mẫu math.h.)

5. So sánh cách dùng các câu lệnh lặp

Thông qua các ví dụ đã trình bày bạn đọc có thể thấy rằng về mặt thực chất để tổ chức một vòng lặp chúng ta có thể chọn một trong các câu lệnh goto, for, while, do … while, có nghĩa về mặt khả năng thực hiện các câu lệnh này là như nhau. Tuy nhiên,

trong một ngữ cảnh cụ thể việc sử dụng câu lệnh phù hợp trong chúng làm cho chương trình sáng sủa, rõ ràng và tăng độ tin cậy lên cao hơn. Theo thói quen lập trình trong một số ngôn ngữ có trước và dựa trên đặc trưng riêng của từng câu lệnh, các lệnh lặp thường được dùng trong các ngữ cảnh cụ thể như sau:

- FOR thường được sử dụng trong những vòng lặp mà số lần lặp được biết trước, nghĩa là vòng lặp thường được tổ chức dưới dạng một (hoặc nhiều) biến đếm chạy từ một giá trị nào đó và đến khi đạt được đến một giá trị khác cho trước thì dừng. Ví dụ dạng thường dùng của câu lệnh for là như sau:
- for (i = gt1 ; i <= gt2 ; i++) … tức i tăng từ gt1 đến gt2 hoặc
- for (i = gt2 ; i >= gt1 ; i--) … tức i giảm từ gt2 xuống gt1
- Ngược lại với FOR, WHILE và DO … WHILE thường dùng trong các vòng lặp mà số lần lặp không biết trước, chúng thường được sử dụng khi việc lặp hay dừng phụ thuộc vào một biểu thức lôgic.
- WHILE được sử dụng khi khả năng thực hiện khối lặp không xảy ra lần nào, tức nếu điều kiện lặp có giá trị sai ngay từ đầu, trong khi đó DO … WHILE được sử dụng khi ta biết chắc chắn khối lệnh lặp phải được thực hiện ít nhất một lần.

III. MẢNG DỮ LIỆU

1. Mảng một chiều

a. Ý nghĩa

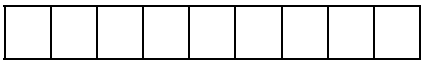
Khi cần lưu trữ một dãy n phần tử dữ liệu chúng ta cần khai báo n biến tương ứng với n tên gọi khác nhau. Điều này sẽ rất khó khăn cho người lập trình để có thể nhớ và quản lý hết được tất cả các biến, đặc biệt khi n lớn. Trong thực tế, hiển nhiên chúng ta gặp rất nhiều dữ liệu có liên quan đến nhau về một mặt nào đó, ví dụ chúng có cùng kiểu và cùng thể hiện một đối tượng: như các toạ độ của một vectơ, các số hạng của một ma trận, các sinh viên của một lớp hoặc các dòng kí tự của một văn bản … Lợi dụng đặc điểm này toàn bộ dữ liệu (cùng kiểu và cùng mô tả một đối tượng) có thể chỉ cần chung một tên gọi để phân biệt với các đối tượng khác, và để phân biệt các dữ liệu trong cùng đối tượng ta sử dụng cách đánh số thứ tự cho chúng, từ đó việc quản lý biến sẽ dễ dàng hơn, chương trình sẽ gọn và có tính hệ thống hơn.

Giả sử ta có 2 vectơ trong không gian ba chiều, mỗi vec tơ cần 3 biến để lưu 3 toạ độ, vì vậy để lưu toạ độ của 2 vectơ chúng ta phải dùng đến 6 biến, ví dụ x1, y1, z1 cho vectơ thứ nhất và x2, y2, z2 cho vectơ thứ hai. Một kiểu dữ liệu mới được gọi là mảng một chiều cho phép ta chỉ cần khai báo 2 biến v1 và v2 để chỉ 2 vectơ, trong đó mỗi v1

hoặc v2 sẽ chứa 3 dữ liệu được đánh số thứ tự từ 0 đến 2, trong đó ta có thể ngầm định thành phần 0 biểu diễn toạ độ x, thành phần 1 biểu diễn toạ độ y và thành phần có số thứ tự 2 sẽ biểu diễn toạ độ z.

Tóm lại, mảng là một dãy các thành phần có cùng kiểu được sắp kề nhau liên tục trong bộ nhớ. Tất cả các thành phần đều có cùng tên là tên của mảng. Để phân biệt các thành phần với nhau, các thành phần sẽ được đánh số thứ tự từ 0 cho đến hết mảng. Khi cần nói đến thành phần cụ thể nào của mảng ta sẽ dùng tên mảng và kèm theo số thứ tự của thành phần đó.

Dưới đây là hình ảnh của một mảng gồm có 9 thành phần, các thành phần được đánh số từ 0 đến 8.



0 1 2 3 4 5 6 7 8

b. Khai báo

<tên kiểu> <tên mảng>[số thành phần] ; // không khởi tạo

<tên kiểu> <tên mảng>[số thành phần] = { dãy giá trị } ; // có khởi tạo

<tên kiểu> <tên mảng>[ ] = { dãy giá trị } ; // có khởi tạo

- Tên kiểu là kiểu dữ liệu của các thành phần, các thành phần này có kiểu giống nhau. Thỉnh thoảng ta cũng gọi các thành phần là phần tử.
- Cách khai báo trên giống như khai báo tên biến bình thường nhưng thêm số thành phần trong mảng giữa cặp dấu ngoặc vuông [] còn được gọi là kích thước của mảng. Mỗi tên mảng là một biến và để phân biệt với các biến thông thường ta còn gọi là biến mảng.
- Một mảng dữ liệu được lưu trong bộ nhớ bởi dãy các ô liên tiếp nhau. Số lượng ô bằng với số thành phần của mảng và độ dài (byte) của mỗi ô đủ để chứa thông tin của mỗi thành phần. Ô đầu tiên được đánh thứ tự bởi 0, ô tiếp theo bởi 1, và tiếp tục cho đến hết. Như vậy nếu mảng có n thành phần thì ô cuối cùng trong mảng sẽ được đánh số là n - 1.
- Dạng khai báo thứ 2 cho phép khởi tạo mảng bởi dãy giá trị trong cặp dấu {}, mỗi giá trị cách nhau bởi dấu phảy (,), các giá trị này sẽ được gán lần lượt cho các phần tử của mảng bắt đầu từ phần tử thứ 0 cho đến hết dãy. Số giá trị có thể bé hơn số phần tử. Các phần tử mảng chưa có giá trị sẽ không được xác định cho đến khi trong chương trình nó được gán một giá trị nào đó.
- Dạng khai báo thứ 3 cho phép vắng mặt số phần tử, trường hợp này số phần tử được xác định bởi số giá trị của dãy khởi tạo. Do đó nếu vắng mặt cả dãy khởi

tạo là không được phép (chẳng hạn khai báo int a[] là sai).

Ví dụ:

- Khai báo biến chứa 2 vectơ a, b trong không gian 3 chiều:

```
float a[3] , b[3] ;
```

- Khai báo 3 phân số a, b, c; trong đó a = 1/3 và b = 3/5:

```
int a[2] = {1, 3} , b[2] = {3, 5} , c[2] ;
```

ở đây ta ngầm qui ước thành phần đầu tiên (số thứ tự 0) là tử và thành phần thứ hai (số thứ tự 1) là mẫu của phân số.

- Khai báo mảng L chứa được tối đa 100 số nguyên dài:

```
long L[100] ;
```

- Khai báo mảng dong (dòng), mỗi dòng chứa được tối đa 80 kí tự:

```
char dong[80] ;
```

- Khai báo dãy Data chứa được 5 số thực độ chính xác gấp đôi:

```
double Data[] = { 0,0,0,0,0 };                    // khởi tạo tạm thời bằng 0
```

c. Cách sử dụng

i. Để chỉ thành phần thứ i (hay chỉ số i) của một mảng ta viết tên mảng kèm theo chỉ số trong cặp ngoặc vuông []. Ví dụ với các phân số trên a[0], b[0], c[0] để chỉ tử số và a[1], b[1], c[1] để chỉ mẫu số của 3 phân số a,b,c.

ii. Tuy mỗi mảng biểu diễn một đối tượng nhưng chúng ta không thể áp dụng các thao tác lên toàn bộ mảng mà phải thực hiện thao tác thông qua từng thành phần của mảng. Ví dụ chúng ta không thể nhập dữ liệu cho mảng a[10] bằng câu lệnh:

```
cin >> a ;                                        // sai
```

mà phải nhập cho từng phần tử từ a[0] đến a[9] của a. Dĩ nhiên trong trường hợp này chúng ta phải cần đến lệnh lặp for:

```
int i ;
for (i = 0 ; i < 10 ; i++) cin >> a[i] ;
```

Tương tự, giả sử chúng ta cần cộng 2 phân số a, b và đặt kết quả vào c.

Không thể viết:

```
c = a + b ;                                       // sai
```

mà cần phải tính từng phần tử của c:

```
c[0] = a[0] * b[1] + a[1] * b[0] ;    // tử số
c[1] = a[1] * b[1] ;                  // mẫu số
```

Để khắc phục nhược điểm này, trong các chương sau C++ cung cấp một kiểu dữ liệu mới gọi là lớp, và cho phép NSD có thể định nghĩa riêng phép cộng cho 2 mảng tuỳ ý, khi đó có thể viết một cách đơn giản và quen thuộc c = a + b để cộng 2 phân số.

d. Ví dụ minh hoạ

Ví dụ 1 : Tìm tổng, tích 2 phân số.

```
void main()
{
    int a[2], b[2], tong[2], tich[2] ;
    cout << "Nhập a. Tử = " ; cin >> a[0] ; cout << "mẫu = " ; cin >> a[1] ;
    cout << "Nhập b. Tử = " ; cin >> b[0] ; cout << "mẫu = " ; cin >> b[1] ;
    tong[0] = a[0]*b[1] + a[1]*b[0] ; tong[1] = a[1] * b[1] ;
    tich[0] = a[0]*b[0]; tich[1] = a[1] * b[1] ;
    cout << "Tổng = " << tong[0] << '/' << tong[1] ;
    cout << "Tích = " << tich[0] << '/' << tich[1] ;
}
```

Ví dụ 2 : Nhập dãy số nguyên, tính: số số hạng dương, âm, bằng không của dãy.

```
void main()
{
    float a[50], i, n, sd, sa, s0;                    // a chứa tối đa 50 số
    cout << "Nhập số phần tử của dãy: " ; cin >> n;       // nhập số phần tử
    for (i=0; i<n; i++) { cout << "a[" << i << "] = " ; cin >> a[i]; }  // nhập dãy số
    sd = sa = s0 = 0 ;
    for (i=1; i<n; i++) {
        if (a[i] > 0 ) sd++;
        if (a[i] < 0 ) sa++;
        if (a[i] == 0 ) s0++;
    }
    cout << "Số số dương = " << sd << " số số âm = " << sa ;
```

```
        cout << "Số số bằng 0 = " << s0 ;
}
```

Ví dụ 3 : Tìm số bé nhất của một dãy số. In ra số này và vị trí của nó trong dãy.

Chương trình sử dụng mảng a để lưu dãy số, n là số phần tử thực sự trong dãy, min lưu số bé nhất tìm được và k là vị trí của min trong dãy. min được khởi tạo bằng giá trị đầu tiên (a[0]), sau đó lần lượt so sánh với các số hạng còn lại, nếu gặp số hạng nhỏ hơn, min sẽ nhận giá trị của số hạng này. Quá trình so sánh tiếp tục cho đến hết dãy. Vì số số hạng của dãy là biết trước (n), nên số lần lặp cũng được biết trước (n-1 lần lặp), do vậy chúng ta sẽ sử dụng câu lệnh for cho ví dụ này.

```
void main()
{
        float a[100], i, n, min, k;                     // a chứa tối đa 100 số
        cout << "Nhập số phần tử của dãy: " ; cin >> n;
        for (i=0; i<n; i++) { cout << "a[" << i << "] = " ; cin >> a[i]; }
        min = a[0]; k = 0;
        for (i=1; i<n; i++) if (a[i] < min ) { min = a[i]; k = i; }
cout << "Số bé nhất là " << min << "tại vị trí " << k;
}
```

Ví dụ 4 : Nhập và sắp xếp tăng dần một dãy số. Thuật toán được tiến hành bằng cách sắp xếp dần từng số hạng bé nhất lên đầu dãy. Giả sử đã sắp được i-1 vị trí, ta sẽ tìm số bé nhất trong dãy còn lại (từ vị trí thứ i đến n-1) và đưa số này lắp vào vị trí thứ i. Để thực hiện, chúng ta so sánh a[i] lần lượt với từng số a[j] trong dãy còn lại (tức j đi từ i+1 đến n), nếu gặp a[j] bé hơn a[i] thì đổi chỗ hai số này với nhau.

```
void main()
{
        float a[100], i, j, n, tam;
        cout << "Cho biết số phần tử n = " ; cin >> n ;
        for (i=0; i<n; i++) {cout<<"a[" <<i<< "] = "; cin >> a[i] ;}    // nhập dữ liệu
        for (i=0; i<n; i++) {
                for (j=i+1; j<n; j++)
                if (a[i] > a[j]) { tam = a[i]; a[i] = a[j]; a[j] = tam; }          // đổi chỗ
        }
```

```
    for (i=0; i<n; i++) cout << a[i] ;                    // in kết quả
    getch();
}
```

2. Xâu kí tự

Một xâu kí tự là một dãy bất kỳ các kí tự (kể cả dấu cách) do vậy nó có thể được lưu bằng mảng kí tự. Tuy nhiên để máy có thể nhận biết được mảng kí tự này là một xâu, cần thiết phải có kí tự kết thúc xâu, theo qui ước là kí tự có mã 0 (tức '\0') tại vị trí nào đó trong mảng. Khi đó xâu là dãy kí tự bắt đầu từ phần tử đầu tiên (thứ 0) đến kí tự kết thúc xâu đầu tiên (không kể các kí tự còn lại trong mảng).

0	1	2	3	4	5	6	7
H	E	L	L	O	\0		
H	E	L	\0	L	O	\0	
\0	H	E	L	L	O	\0	

Hình vẽ trên minh hoạ 3 xâu, mỗi xâu được chứa trong mảng kí tự có độ dài tối đa là 8. Nội dung xâu thứ nhất là "Hello" có độ dài thực tế là 5 kí tự, chiếm 6 ô trong mảng (thêm ô chứa kí tự kết thúc '\0'). Xâu thứ hai có nội dung "Hel" với độ dài 3 (chiếm 4 ô) và xâu cuối cùng biểu thị một xâu rỗng (chiếm 1 ô). Chú ý mảng kí tự được khai báo với độ dài 8 tuy nhiên các xâu có thể chỉ chiếm một số kí tự nào đó trong mảng này và tối đa là 7 kí tự.

a. Khai báo

```
char <tên xâu>[độ dài] ;                    // không khởi tạo
char <tên xâu>[độ dài] = xâu kí tự ;        // có khởi tạo
char <tên xâu>[] = xâu kí tự ;              // có khởi tạo
```

- Độ dài mảng là số kí tự tối đa có thể có trong xâu. Độ dài thực sự của xâu chỉ tính từ đầu mảng đến dấu kết thúc xâu (không kể dấu kết thúc xâu '\0').
- Do một xâu phải có dấu kết thúc xâu nên trong khai báo độ dài của mảng cần phải khai báo thừa ra một phần tử. Thực chất độ dài tối đa của xâu = độ dài mảng - 1. Ví dụ nếu muốn khai báo mảng s chứa được xâu có độ dài tối đa 80 kí tự, ta cần phải khai báo `char s[81]`.
- Cách khai báo thứ hai có kèm theo khởi tạo xâu, đó là dãy kí tự đặt giữa cặp dấu nháy kép. Ví dụ:

```
char hoten[26] ;                    // xâu họ tên chứa tối đa 25 kí tự
```

```
char monhoc[31] = "NNLT C++" ;
```

xâu môn học chứa tối đa 30 kí tự, được khởi tạo với nội dung "NNLT C++" với độ dài thực sự là 10 kí tự (chiếm 11 ô đầu tiên trong mảng monhoc[31]).

– Cách khai báo thứ 3 tự chương trình sẽ quyết định độ dài của mảng bởi xâu khởi tạo (bằng độ dài xâu + 1). Ví dụ:

```
char thang[] = "Mười hai" ;          // độ dài mảng = 9
```

b. Cách sử dụng

Tương tự như các mảng dữ liệu khác, xâu kí tự có những đặc trưng như mảng, tuy nhiên chúng cũng có những điểm khác biệt. Dưới đây là các điểm giống và khác nhau đó.

- Truy cập một kí tự trong xâu: cú pháp giống như mảng. Ví dụ:

```
char s[50] = "I\'m a student" ;      // chú ý kí tự ' phải được viết là \'
cout << s[0] ;                       // in kí tự đầu tiên, tức kí tự 'I'
s[1] = 'a' ;                         // đặt lại kí tự thứ 2 là 'a'
```

- Không được thực hiện các phép toán trực tiếp trên xâu như:

```
char s[20] = "Hello", t[20] ;        // khai báo hai xâu s và t
t = "Hello" ;                        // sai, chỉ gán được khi khai báo
t = s ;                              // sai, không gán được toàn bộ mảng
if (s < t) …                         // sai, không so sánh được hai mảng
…
```

- Toán tử nhập dữ liệu >> vẫn dùng được nhưng có nhiều hạn chế. Ví dụ

```
char s[60] ;
cin >> s ;
cout << s ;
```

nếu xâu nhập vào là "Tin học hoá" chẳng hạn thì toán tử >> chỉ nhập "Tin" cho s (bỏ tất cả các kí tự đứng sau dấu trắng), vì vậy khi in ra trên màn hình chỉ có từ "Tin".

Vì các phép toán không dùng được trực tiếp trên xâu nên các chương trình dịch đã viết sẵn các hàm thư viện được khai báo trong file nguyên mẫu string.h. Các hàm này giải quyết được hầu hết các công việc cần thao tác trên xâu. Nó cung cấp cho NSD phương tiện để thao tác trên xâu như gán, so sánh, sao chép, tính độ dài xâu, nhập, in, … Để sử dụng được các hàm này đầu chương trình cần có khai báo string.h. Phần lớn các hàm này sẽ được giới thiệu trong phần tiếp sau.

c. Phương thức nhập xâu (#include <iostream.h>)

Do toán tử nhập >> có hạn chế đối với xâu kí tự nên C++ đưa ra hàm riêng (còn gọi là phương thức) **cin.getline(s,n)** để nhập xâu kí tự. Hàm có 2 đối với s là xâu cần nhập nội dung và n-1 là số kí tự tối đa của xâu. Giống phương thức nhập kí tự cin.get(c), khi gặp hàm cin.getline(s,n) chương trình sẽ nhìn vào bộ đệm bàn phím lấy ra n-1 kí tự (nếu đủ hoặc lấy tất cả kí tự còn lại, trừ kí tự enter) và gán cho s. Nếu tại thời điểm đó bộ đệm đang rỗng, chương trình sẽ tạm dừng chờ NSD nhập dữ liệu (dãy kí tự) vào từ bàn phím. NSD có thể nhập vào dãy với độ dài bất kỳ cho đến khi nhấn Enter, chương trình sẽ lấy ra n-1 kí tự đầu tiên gán cho s, phần còn lại vẫn được lưu trong bộ đệm (kể cả kí tự Enter) để dùng cho lần nhập sau. Hiển nhiên, sau khi gán các kí tự cho s, chương trình sẽ tự động đặt kí tự kết thúc xâu vào ô tiếp theo của xâu s.

Ví dụ 1 : Xét đoạn lệnh sau

```
char s[10] ;
cin.getline(s, 10) ;
cout << s << endl ;
cin.getline(s, 10) ;
cout << s << endl ;
```

giả sử ta nhập vào bàn phím dòng kí tự: 1234567890abcd ↵. Khi đó lệnh cin.getline(s,10) đầu tiên sẽ gán xâu "123456789" (9 kí tự) cho s, phần còn lại vẫn lưu trong bộ đệm bàn phím. Tiếp theo s được in ra màn hình. Đến lệnh cin.getline(s,10) thứ hai NSD không phải nhập thêm dữ liệu, chương trình tự động lấy nốt số dữ liệu còn lại (vì chưa đủ 9 kí tự) "0abcd" để gán cho s. Sau đó in ra màn hình. Như vậy trên màn hình sẽ xuất hiện hai dòng:

```
123456789
0abcd
```

Ví dụ 2 : Nhập một ngày tháng dạng Mỹ (mm/dd/yy), đổi sang ngày tháng dạng Việt Nam rồi in ra màn hình.

```
#include <iostream.h>
main()
{
    char US[9], VN[9] = " / / " ;          // khởi tạo trước hai dấu /
    cin.getline(US, 9) ;              // nhập ngày tháng, ví dụ "05/01/99"
    VN[0] = US[3]; VN[1] = US[4] ;        // ngày
    VN[3] = US[0]; VN[4] = US[1] ;        // tháng
```

```
        VN[6] = US[6]; VN[7] = US[7] ;          // năm
        cout << VN << endl ;
}
```

d. Một số hàm xử lí xâu (#include <string.h>)

- **strcpy(s, t) ;**

Gán nội dung của xâu t cho xâu s (thay cho phép gán = không được dùng). Hàm sẽ sao chép toàn bộ nội dung của xâu t (kể cả kí tự kết thúc xâu) vào cho xâu s. Để sử dụng hàm này cần đảm bảo độ dài của mảng s ít nhất cũng bằng độ dài của mảng t. Trong trường hợp ngược lại kí tự kết thúc xâu sẽ không được ghi vào s và điều này có thể gây treo máy khi chạy chương trình.

Ví dụ:

```
char s[10], t[10] ;
t = "Face" ;                        // không được dùng
s = t ;                             // không được dùng
strcpy(t, "Face") ;                 // được, gán "Face" cho t
strcpy(s, t) ;                      // được, sao chép t sang s
cout << s << " to " << t ;          // in ra: Face to Face
```

- **strncpy(s, t, n) ;**

Sao chép n kí tự của t vào s. Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn kí tự kết thúc xâu cho s. Do vậy NSD phải thêm câu lệnh đặt kí tự '\0' vào cuối xâu s sau khi sao chép xong.

Ví dụ:

```
char s[10], t[10] = "Steven";
strncpy(s, t, 5) ;                  // copy 5 kí tự "Steve" vào s
s[5] = '\0' ;                       // đặt dấu kết thúc xâu
// in câu: Steve is young brother of Steven
cout << s << " is young brother of " << t ;
```

Một sử dụng có ích của hàm này là copy một xâu con bất kỳ của t và đặt vào s. Ví dụ cần copy xâu con dài 2 kí tự bắt đầu từ kí tự thứ 3 của xâu t và đặt vào s, ta viết **strncpy(s, t+3, 2)**. Ngoài ra xâu con được copy có thể được đặt vào vị trí bất kỳ của s (không nhất thiết phải từ đầu xâu s) chẳng hạn đặt vào từ vị trí thứ 5, ta viết: **strncpy(s+5, t+3, 2)**. Câu lệnh này có nghĩa: lấy 2 kí tự thứ 3 và thứ 4 của xâu t đặt vào 2 ô thứ 5 và thứ 6 của xâu s. Trên cơ sở này chúng ta có thể viết các đoạn chương

trình ngắn để thay thế một đoạn con bất kỳ nào đó trong s bởi một đoạn con bất kỳ (có độ dài tương đương) trong t. Ví dụ các dòng lệnh chuyển đổi ngày tháng trong ví dụ trước có thể viết lại bằng cách dùng hàm strncpy như sau:

```
strncpy(VN+0, US+3, 2) ;          // ngày
strncpy(VN+3, US+0, 2) ;          // tháng
strncpy(VN+6, US+6, 2);           // năm
```

- **strcat(s, t);**

Nối một bản sao của t vào sau s (thay cho phép +). Hiển nhiên hàm sẽ loại bỏ kí tự kết thúc xâu s trước khi nối thêm t. Việc nối sẽ đảm bảo lấy cả kí tự kết thúc của xâu t vào cho s (nếu s đủ chỗ) vì vậy NSD không cần thêm kí tự này vào cuối xâu. Tuy nhiên, hàm không kiểm tra xem liệu độ dài của s có đủ chỗ để nối thêm nội dung, việc kiểm tra này phải do NSD đảm nhiệm. Ví dụ:

```
char a[100] = "Mẫn", b[4] = "tôi";
strcat(a, " và ");
strcat(a, b);
cout << a                                  // Mẫn và tôi

char s[100] , t[100] = "Steve" ;
strncpy(s, t, 3); s[3] = '\0';             // s = "Ste"
strcat(s, "p");                            // s = "Step"
cout << t << " goes "<< s << " by " <<s    // Steve goes Step by Step
```

- **strncat(s, t, n);**

Nối bản sao n kí tự đầu tiên của xâu t vào sau xâu s. Hàm tự động đặt thêm dấu kết thúc xâu vào s sau khi nối xong (tương phản với strncpy()). Cũng giống strcat hàm đòi hỏi độ dài của s phải đủ chứa kết quả. Tương tự, có thể sử dụng cách viết **strncat(s, t+k, n)** để nối n kí tự từ vị trí thứ k của xâu t cho s.

Ví dụ:

```
char s[20] = "Nhà " ;
char t[] = "vua chúa"
strncat(s, t, 3) ;                 // s = "Nhà vua"
hoặc:
strncat(s, t+4, 4) ;               // s = "Nhà chúa"
```

- **strcmp(s, t);**

Hàm so sánh 2 xâu s và t (thay cho các phép toán so sánh). Giá trị trả lại là hiệu 2 kí tự khác nhau đầu tiên của s và t. Từ đó, nếu s1 < s2 thì hàm trả lại giá trị âm, bằng 0 nếu s1==s2, và dương nếu s1 > s2. Trong trường hợp chỉ quan tâm đến so sánh bằng, nếu hàm trả lại giá trị 0 là 2 xâu bằng nhau và nếu giá trị trả lại khác 0 là 2 xâu khác nhau.

Ví dụ:

```
if (strcmp(s,t)) cout << "s khác t"; else cout << "s bằng t" ;
```

- **strncmp(s, t) ;**

Giống hàm strcmp(s, t) nhưng chỉ so sánh tối đa n kí tự đầu tiên của hai xâu.

Ví dụ:

```
char s[] = "Hà Nội" , t[] = "Hà nội" ;
cout << strcmp(s,t) ;                  // -32 (vì 'N' = 78, 'n' = 110)
cout << strncmp(s, t, 3) ;             // 0 (vì 3 kí tự đầu của s và t là như nhau)
```

- **strcmpi(s, t) ;**

Như strcmp(s, t) nhưng không phân biệt chữ hoa, thường.

Ví dụ:

```
char s[] = "Hà Nội" , t[] = "hà nội" ;
cout << strcmpi(s, t) ;                // 0 (vì s = t)
```

- **strupr(s);**

Hàm đổi xâu s thành in hoa, và cũng trả lại xâu in hoa đó.

Ví dụ:

```
char s[10] = "Ha noi" ;
cout << strupr(s) ;                    // HA NOI
cout << s ;                            // HA NOI (s cũng thành in hoa)
```

- **strlwr(s);**

Hàm đổi xâu s thành in thường, kết quả trả lại là xâu s.

Ví dụ:

```
char s[10] = "Ha Noi" ;
cout << strlwr(s) ;                    // ha noi
cout << s ;                            // ha noi (s cũng thành in thường)
```

- **strlen(s) ;**

Hàm trả giá trị là độ dài của xâu s.

Ví dụ:

```
char s[10] = "Ha Noi" ;
cout << strlen(s) ;                    // 5
```

Sau đây là một số ví dụ sử dụng tổng hợp các hàm trên.

Ví dụ 1 : Thống kê số chữ 'a' xuất hiện trong xâu s.

```
main()
{
    const int MAX = 100;
    char s[MAX+1];
    int sokitu = 0;
    cin.getline(s, MAX+1);
    for (int i=0; i < strlen(s); i++)   if (s[i] = 'a ') sokitu++;
    cout << "Số kí tự = " << sokitu << endl ;
}
```

Ví dụ 2 : Tính độ dài xâu bằng cách đếm từng kí tự (tương đương với hàm strlen())

```
main()
{
    char s[100];                       // độ dài tối đa là 99 kí tự
    cin.getline(s, 100);               // nhập xâu s
    for (int i=0 ; s[i] != '\0' ; i++) ;   // chạy từ đầu đến cuối xâu
    cout << "Độ dài xâu = " << i ;
}
```

Ví dụ 3 : Sao chép xâu s sang xâu t (tương đương với hàm strcpy(t,s))

```
void main()
{
    char s[100], t[100];
    cin.getline(s, 100);               // nhập xâu s
    int i=0;
```

```
        while ((t[i] = s[i]) != '\0') i++;          // copy cả dấu kết thúc xâu '\0'
        cout << t << endl ;
}
```

Ví dụ 4 : Cắt dấu cách 2 đầu của xâu s. Chương trình sử dụng biến i chạy từ đầu xâu đến vị trí đầu tiên có kí tự khác dấu trắng. Từ vị trí này sao chép từng kí tự còn lại của xâu về đầu xâu bằng cách sử dụng thêm biến j để làm chỉ số cho xâu mới. Kết thúc sao chép j sẽ ở vị trí cuối xâu (mới). Cho j chạy ngược về đầu xâu cho đến khi gặp kí tự đầu tiên khác dấu trắng. Đặt dấu kết thúc xâu tại đây.

```
main()
{
        char s[100];
        cin.getline(s, 100);                        // nhập xâu s
        int i, j ;
        i = j = 0;
        while (s[i++] == ' '); i-- ;                // bỏ qua các dấu cách đầu tiên
        while (s[i] != '\0') s[j++] = s[i++] ;      // sao chép phần còn lại vào s
        while (s[--j] == ' ') ;                     // bỏ qua các dấu cách cuối
        s[j+1] = '\0' ;                             // đặt dấu kết thúc xâu
        cout << s ;
}
```

Ví dụ 5 : Chạy dòng chữ quảng cáo vòng tròn từ phải sang trái giữa màn hình.

Giả sử hiện 30 kí tự của xâu quảng cáo. Ta sử dụng vòng lặp. Cắt 30 kí tự đầu tiên của xâu cho vào biến hien, hiện biến này ra màn hình. Bước lặp tiếp theo cắt ra 30 kí tự của xâu nhưng dịch sang phải 1 kí tự cho vào biến hien và hiện ra màn hình. Quá trình tiếp tục, mỗi bước lặp ta dịch chuyển nội dung cần hiện ra màn hình 1 kí tự, do hiệu ứng của mắt ta thấy dòng chữ sẽ chạy từ biên phải về biên trái của màn hình. Để quá trình chạy theo vòng tròn (khi hiện đến kí tự cuối của xâu sẽ hiện quay lại từ kí tự đầu của xâu) chương trình sử dụng biến i đánh dấu điểm đầu của xâu con cần cắt cho vào hien, khi i bằng độ dài của xâu chương trình đặt lại i = 0 (cắt lại từ đầu xâu). Ngoài ra, để phần cuối xâu nối với phần đầu (tạo thành vòng tròn) ngay từ đầu chương trình, xâu quảng cáo sẽ được nối thành gấp đôi.

Vòng lặp tiếp tục đến khi nào NSD ấn phím bất kỳ (chương trình nhận biết điều này nhờ vào hàm kbhit() thuộc file nguyên mẫu conio.h) thì dừng. Để dòng chữ chạy

không quá nhanh chương trình sử dụng hàm trễ delay(n) (thuộc dos.h, tạm dừng trong n phần nghìn giây) với n được điều chỉnh thích hợp theo tốc độ của máy. Hàm gotoxy(x, y) (thuộc conio.h) trong chương trình đặt con trỏ màn hình tại vị trí cột x dòng y để đảm bảo dòng chữ luôn luôn hiện ra tại đúng một vị trí trên màn hình.

```
#include <iostream.h>
#include <conio.h>
#include <dos.h>
main()
{
    char qc[100] = "Quảng cáo miễn phí: Không có tiền thì không có kem. ";
    int dd = strlen(qc);
    char tam[100] ; strcpy(tam, qc) ;
    strcat(qc, tam) ;                       // nhân đôi dòng quảng cáo
    clrscr();                               // xoá màn hình
    char hien[31] ;                         // chứa xâu dài 30 kí tự để hiện
    i = 0;
    while (!kbhit()) {                      // trong khi chưa ấn phím bất kỳ
        strncpy(hien, s+i, 30);
        hien[30] = '\0';                    // copy 30 kí tự từ qc[i] sang hien
        gotoxy(20,10); cout << hien ;       // in hien tại dòng 10 cot 20
        delay(100);                         // tạm dừng 1/10 giây
        i++; if (i==dd) i = 0;              // tăng i
    }
}
```

Ví dụ 6 : Nhập mật khẩu (không quá 10 kí tự). In ra "đúng" nếu là "HaNoi2000", "sai" nếu ngược lại. Chương trình cho phép nhập tối đa 3 lần. Nhập riêng rẽ từng kí tự (bằng hàm getch()) cho mật khẩu. Hàm getch() không hiện kí tự NSD gõ vào, thay vào đó chương trình chỉ hiện kí tự 'X' để che giấu mật khẩu. Sau khi NSD đã gõ xong (9 kí tự) hoặc đã Enter, chương trình so sánh xâu vừa nhập với "HaNoi2000", nếu đúng chương trình tiếp tuc, nếu sai tăng số lần nhập (cho phép không quá 3 lần).

```
#include <iostream.h>
#include <conio.h>
```

```
#include <string.h>
void main()
{
        char pw[11]; int solan = 0;                 // Cho phep nhap 3 lan
        do {
                clrscr(); gotoxy(30,12) ;
                int i = 0;
                while ((pw[i]=getch()) != 13 && ++i < 10) cout << 'X' ; // 13 = Enter
                pw[i] = '\0' ;
                cout << endl ;
                if (!strcmp(pw, "HaNoi2000")) { cout << "Mời vào" ; break; }
                else { cout << "Sai mật khẩu. Nhập lại") ; solan++ ; }
        } while (solan < 3);
}
```

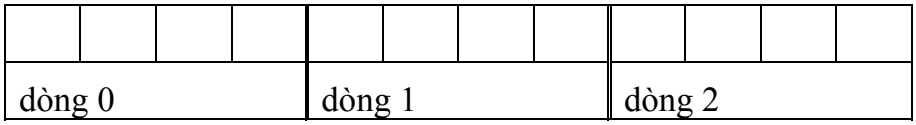
IV. MẢNG HAI CHIỀU

Để thuận tiện trong việc biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu có nhiều chỉ tiêu, C++ đưa ra kiểu dữ liệu mảng nhiều chiều. Tuy nhiên, việc sử dụng mảng nhiều chiều rất khó lập trình vì vậy trong mục này chúng ta chỉ bàn đến mảng hai chiều. Đối với mảng một chiều m thành phần, nếu mỗi thành phần của nó lại là mảng một chiều n phần tử thì ta gọi mảng là hai chiều với số phần tử (hay kích thước) mỗi chiều là m và n. Ma trận là một minh hoạ cho hình ảnh của mảng hai chiều, nó gồm m dòng và n cột, tức chứa m x n phần tử, và hiển nhiên các phần tử này có cùng kiểu. Tuy nhiên, về mặt bản chất mảng hai chiều không phải là một tập hợp với m x n phần tử cùng kiểu mà là tập hợp với m thành phần, trong đó mỗi thành phần là một mảng một chiều với n phần tử. Điểm nhấn mạnh này sẽ được giải thích cụ thể hơn trong các phần trình bày về con trỏ của chương sau.

	0	1	2	3
0				
1				
2				

Hình trên minh hoạ hình thức một mảng hai chiều với 3 dòng, 4 cột. Thực chất

trong bộ nhớ tất cả 12 phần tử của mảng được sắp liên tiếp theo từng dòng của mảng như minh hoạ trong hình dưới đây.



a. Khai báo

<kiểu thành phần > <tên mảng>[m][n] ;

- m, n là số hàng, số cột của mảng.
- kiểu thành phần là kiểu của m x n phần tử trong mảng.
- Trong khai báo cũng có thể được khởi tạo bằng dãy các dòng giá trị, các dòng cách nhau bởi dấu phẩy, mỗi dòng được bao bởi cặp ngoặc {} và toàn bộ giá trị khởi tạo nằm trong cặp dấu {}.

b. Sử dụng

- Tương tự mảng một chiều các chiều trong mảng cũng được đánh số từ 0.
- Không sử dụng các thao tác trên toàn bộ mảng mà phải thực hiện thông qua từng phần tử của mảng.
- Để truy nhập phần tử của mảng ta sử dụng tên mảng kèm theo 2 chỉ số chỉ vị trí hàng và cột của phần tử. Các chỉ số này có thể là các biểu thức thực, khi đó C++ sẽ tự chuyển kiểu sang nguyên.

Ví dụ:

- Khai báo 2 ma trận 4 hàng 5 cột A, B chứa các số nguyên:

```
int A[3][4], B[3][4] ;
```

- Khai báo có khởi tạo:

```
int A[3][4] = { {1,2,3,4}, {3,2,1,4}, {0,1,1,0} };
```

với khởi tạo này ta có ma trận:

1	2	3	4
3	2	1	4
0	1	1	0

trong đó: A[0][0] = 1, A[0][1] = 2, A[1][0] = 3, A[2][3] = 0 …

– Trong khai báo có thể vắng số hàng (không được vắng số cột), số hàng này được xác định thông qua khởi tạo.

```
float A[][3] = { {1,2,3}, {0,1,0} } ;
```

trong khai báo này chương trình tự động xác định số hàng là 2.

– Phép khai báo và khởi tạo sau đây là cũng hợp lệ:

```
float A[][3] = { {1,2}, {0} } ;
```

chương trình cũng xác định số hàng là 2 và số cột (bắt buộc phải khai báo) là 3 mặc dù trong khởi tạo không thể xác định được số cột. Các phần tử chưa khởi tạo sẽ chưa được xác định cho đến khi nào nó được nhập hoặc gán giá trị cụ thể. Trong ví dụ trên các phần tử A[0][2], A[1][1] và A[1][2] là chưa được xác định.

c. Ví dụ minh hoạ

Ví dụ 1 : Nhập, in và tìm phần tử lớn nhất của một ma trận.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main()
{
    float a[10][10] ;
    int m, n ;                          // số hàng, cột của ma trận
    int i, j ;                          // các chỉ số trong vòng lặp
    int amax, imax, jmax ;              // số lớn nhất và chỉ số của nó
    clrscr(); cout << "Nhập số hàng và cột: " ; cin >> m >> n ;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++)
    {
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
    }
    amax = a[0][0]; imax = 0; jmax = 0;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++)
    if (amax < a[i][j])
```

```
        {
                amax = a[i][j]; imax = i; jmax = j;
        }
        cout << "Ma trận đã nhập\n" ;
        cout << setiosflags(ios::showpoint) << setprecision(1) ;
        for (i=0; i<m; i++)
        for (j=0; j<n; j++)
        {
                if (j==0) cout << endl;
                cout << setw(6) << a[i][j] ;
        }
        cout << "Số lớn nhất là " << setw(6) << amax << endl;
        cout << "tại vị trí (" << imax << "," << jmax << ")" ;
        getch();
}
```

Ghi chú: Khi làm việc với mảng (1 chiều, 2 chiều) do thói quen chúng ta thường tính chỉ số từ 1 (thay vì 0), do vậy trong mảng ta có thể bỏ qua hàng 0, cột 0 bằng cách khai báo số hàng và cột tăng lên 1 so với số hàng, cột thực tế của mảng và từ đó có thể làm việc từ hàng 1, cột 1 trở đi.

Ví dụ 2 : Nhân 2 ma trận. Cho 2 ma trận A (m x n) và B (n x p). Tính ma trận C = A x B, trong đó C có kích thước là m x p. Ta lập vòng lặp tính từng phần tử của C. Giá trị của phần tử C tại hàng i, cột j chính là tích vô hướng của hàng i ma trận A với cột j ma trận B. Để tránh nhầm lẫn ta qui ước bỏ các hàng, cột 0 của các ma trận A, B, C (tức các chỉ số được tính từ 1 trở đi).

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
main()
{
        float A[10][10], B[10], C[10][10] ;
        int m, n, p ;                           // số hàng, cột của ma trận
        int i, j, k ;                           // các chỉ số trong vòng lặp
```

```
        clrscr();
        cout << "Nhập số hàng và cột của 2 ma trận: " ; cin >> m >> n >> p;
        // Nhập ma trận A
        for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
        {
                cout << "A[" << i << "," << j << "] = " ; cin >> A[i][j] ;
        }
        // Nhập ma trận B
        for (i=1; i<=n; i++)
        for (j=1; j<=p; j++)
        {
                cout << "B[" << i << "," << j << "] = " ; cin >> B[i][j] ;
        }
        // Tính ma trận C = A x B
        for (i=1; i<=m; i++)
        for (j=1; j<=p; j++)
        {
                C[i][j] = 0; for (k=1; k<=n; k++) C[i][j] += A[i][k]*B[k][j] ;
        }
        // In kết quả
        cout << "Ma trận kết quả\n" ;
        cout << setiosflags(ios::showpoint) << setprecision(2) ;
        for (i=1; i<m; i++)
        for (j=1; j<n; j++)
        {
                if (j==1) cout << endl;
                cout << setw(6) << a[i][j] ;
        }
        getch();
}
```

BÀI TẬP

Lệnh rẽ nhánh

1. Nhập một kí tự. Cho biết kí tự đó có phải là chữ cái hay không.

2. Nhập vào một số nguyên. Trả lời số nguyên đó: âm hay dương, chẵn hay lẻ ?

3. Cho n = x = y và bằng: a. 1 b. 2 c. 3 d. 4

Hãy cho biết giá trị của x, y sau khi chạy xong câu lệnh:

```
if (n % 2 == 0) if (x > 3) x = 0;
else y = 0;
```

4. Tính giá trị hàm

a. $f(x) = \begin{cases} 3x+\sqrt{x} & , x>0 \\ e^x+4 & , x\le 0 \end{cases}$

b. $f(x) = \begin{cases} \sqrt{x^2+1} & , x\ge 1 \\ 3x+5 & , -1<x<1 \\ x^2+2x-1 & , x\le -1 \end{cases}$

5. Viết chương trình giải hệ phương trình bậc nhất 2 ẩn: $\begin{cases} ax+by=c \\ dx+ey=f \end{cases}$

6. Nhập 2 số a, b. In ra max, min của 2 số đó. Mở rộng với 3 số, 4 số ?

7. Nhập 3 số a, b, c. Hãy cho biết 3 số trên có thể là độ dài 3 cạnh của một tam giác ? Nếu là một tam giác thì đó là tam giác gì: vuông, đều, cân, vuông cân hay tam giác thường ?

8. Nhập vào một số, in ra thứ tương ứng với số đó (qui ước 2 là thứ hai, …, 8 là chủ nhật).

9. Nhập 2 số biểu thị tháng và năm. In ra số ngày của tháng năm đó (có kiểm tra năm nhuận).

10. Lấy ngày tháng hiện tại làm chuẩn. Hãy nhập một ngày bất kỳ trong tháng. Cho biết thứ của ngày vừa nhập ?

Lệnh lặp

11. Giá trị của i bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

```
for (i = 0; i < 100; i++);
```

12. Giá trị của x bằng bao nhiêu sau khi thực hiện cấu trúc for sau:

```
for (x = 2; i < 10; x+=3) ;
```

13. Bạn bổ sung gì vào lệnh for sau:

```
for ( ; nam < 1997 ; ) ;
```

để khi kết thúc nam có giá trị 2000.

14. Bao nhiêu kí tự 'X' được in ra màn hình khi thực hiện đoạn chương trình sau:

```
for (x = 0; x < 10; x ++) for (y = 5; y > 0; y --) cout << 'X';
```

15. Nhập vào tuổi cha và tuổi con hiện nay sao cho tuổi cha lớn hơn 2 lần tuổi con. Tìm xem bao nhiêu năm nữa tuổi cha sẽ bằng đúng 2 lần tuổi con (ví dụ 30 và 12, sau 6 năm nữa tuổi cha là 36 gấp đôi tuổi con là 18).

16. Nhập số nguyên dương N. Tính:

a. $S_1 = \dfrac{1+2+3+...+N}{N}$

b. $S_2 = \sqrt{1^2+2^2+3^2+...+N^2}$

17. Nhập số nguyên dương n. Tính:

a. $S_1 = \sqrt{3+\sqrt{3+\sqrt{3+...+\sqrt{3}}}}$ n dấu căn

b. $S_2 = \dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2+...\dfrac{1}{2}}}}$ n dấu chia

18. Nhập số tự nhiên n. In ra màn hình biểu diễn của n ở dạng nhị phân.

19. In ra màn hình các số có 2 chữ số sao cho tích của 2 chữ số này bằng 2 lần tổng của 2 chữ số đó (ví dụ số 36 có tích 3*6 = 18 gấp 2 lần tổng của nó là 3 + 6 = 9).

20. Số *hoàn chỉnh* là số bằng tổng mọi ước của nó (không kể chính nó). Ví dụ 6 = 1 + 2 + 3 là một số hoàn chỉnh. Hãy in ra màn hình tất cả các số hoàn chỉnh < 1000.

21. Các số *sinh đôi* là các số nguyên tố mà khoảng cách giữa chúng là 2. Hãy in tất cả cặp số sinh đôi < 1000.

22. Nhập dãy kí tự đến khi gặp kí tự '.' thì dừng. Thống kê số chữ cái viết hoa, viết thường, số chữ số và tổng số các kí tự khác đã nhập. Loại kí tự nào nhiều nhất ?

23. Tìm số nguyên dương n lớn nhất thoả mãn điều kiện:

a. $1+\frac{1}{3}+\frac{1}{5}+\ldots+\frac{1}{2n-1} < 2.101999$.

b. $e^n - 1999 log_{10} n < 2000$.

24. Cho ε = 1e–6. Tính gần đúng các số sau:

a. Số pi theo công thức Euler: $\frac{\pi^2}{6}=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2}$ dừng lặp khi $\frac{1}{n^2}<10^{-6}$.

b. e^x theo công thức: $e^x = 1+\frac{x^1}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}$ dừng lặp khi $\left|\frac{x^n}{n!}\right| < 10^{-6}$.

c. $sin\, x = x-\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+(-1)^n \frac{x^{2n+1}}{(2n+1)!}$, dừng lặp khi $\left|\frac{x^{2n+1}}{(2n+1)!}\right| < 10^{-6}$.

d. $\sqrt{a}\ (a>0)$ theo công thức: $s_n = \begin{cases} a & n=0 \\ (s_{n-1}^2+a)/2s_{n-1} & n>0 \end{cases}$, dừng khi $|s_n - s_{n-1}|<10^{-6}$.

25. In ra mã của phím bất kỳ được nhấn. Chương trình lặp cho đến khi nhấn ESC để thoát.

26. Bằng phương pháp chia đôi, hãy tìm nghiệm xấp xỉ (độ chính xác 10^{-6}) của các phương trình sau:

a. $e^x - 1.5 = 0$, trên đoạn [0, 1].

b. $x2^x - 1 = 0$, trên đoạn [0, 1].

c. $a_0x^n + a_1x^{n-1} + \ldots + a_n = 0$, trên đoạn [a, b]. Các số thực a_i, a, b được nhập từ bàn phím sao cho f(a) và f(b) trái dấu.

Mảng

27. Nhập vào dãy n số thực. Tính tổng dãy, trung bình dãy, tổng các số âm, dương và tổng các số ở vị trí chẵn, vị trí lẻ trong dãy. Tìm phần tử gần số trung bình nhất

của dãy.

28. Tìm và chỉ ra vị trí xuất hiện đầu tiên của phần tử x trong dãy.

29. Nhập vào dãy n số. Hãy in ra số lớn nhất, bé nhất của dãy.

30. Nhập vào dãy số. In ra dãy đã được sắp xếp tăng dần, giảm dần.

31. Cho dãy đã được sắp tăng dần. Chèn thêm vào dãy phần tử x sao cho dãy vẫn sắp xếp tăng dần.

32. Hãy nhập vào 16 số nguyên. In ra thành 4 dòng, 4 cột.

33. Nhập ma trận A và in ra ma trận đối xứng của nó.

34. Cho một ma trận nguyên kích thước m*n. Tính:

- Tổng tất cả các phần tử của ma trận.
- Tổng tất cả các phần tử dương của ma trận.
- Tổng tất cả các phần tử âm của ma trận.
- Tổng tất cả các phần tử chẵn của ma trận.
- Tổng tất cả các phần tử lẻ của ma trận.

35. Cho một ma trận thực kích thước m*n. Tìm:

- Số nhỏ nhất, lớn nhất (kèm chỉ số) của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của từng hàng của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của từng cột của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của đường chéo chính của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của đường chéo phụ của ma trận.

36. Nhập 2 ma trận vuông cấp n A và B. Tính A + B, A – B, A * B và A^2 - B^2.

Xâu kí tự

37. Hãy nhập một xâu kí tự. In ra màn hình đảo ngược của xâu đó.

38. Nhập xâu. Thống kê số các chữ số '0', số chữ số '1', …, số chữ số '9' trong xâu.

39. In ra vị trí kí tự trắng đầu tiên từ bên trái (phải) một xâu kí tự.

40. Nhập xâu. In ra tất các các vị trí của chữ 'a' trong xâu và tổng số lần xuất hiện của nó.

41. Nhập xâu. Tính số từ có trong xâu. In mỗi dòng một từ.

42. Nhập xâu họ tên, in ra họ, tên dưới dạng viết hoa.

43. Thay kí tự x trong xâu s bởi kí tự y (s, x, y được đọc vào từ bàn phím)

44. Xoá mọi kí tự x có trong xâu s (s, x được đọc vào từ bàn phím). (Gợi ý: nên xoá ngược từ cuối xâu về đầu xâu).

45. Nhập xâu. Không phân biệt viết hoa hay viết thường, hãy in ra các kí tự có mặt trong xâu và số lần xuất hiện của nó (ví dụ xâu "Trach – Van – Doanh" có chữ a xuất hiện 3 lần, c(1), d(1), h(2), n(2), o(1), r(1), t(1), –(2), space(4)).

CHƯƠNG 4

HÀM VÀ CHƯƠNG TRÌNH

Con trỏ và số học địa chỉ
Hàm
Đệ qui
Tổ chức chương trình

I. CON TRỎ VÀ SỐ HỌC ĐỊA CHỈ

Trước khi bàn về hàm và chương trình, trong phần này chúng ta sẽ nói về một loại biến mới gọi là con trỏ, ý nghĩa, công dụng và sử dụng nó như thế nào. Biến con trỏ là một đặc trưng mạnh của C++, nó cho phép chúng ta thâm nhập trực tiếp vào bộ nhớ để xử lý các bài toán khó bằng chỉ vài câu lệnh đơn giản của chương trình. Điều này cũng góp phần làm cho C++ trở thành ngôn ngữ gần gũi với các ngôn ngữ cấp thấp như hợp ngữ. Tuy nhiên, vì tính đơn giản, ngắn gọn nên việc sử dụng con trỏ đòi hỏi tính cẩn thận cao và giàu kinh nghiệm của người lập trình.

1. Địa chỉ, phép toán &

Mọi chương trình trước khi chạy đều phải bố trí các biến do NSD khai báo vào đâu đó trong bộ nhớ. Để tạo điều kiện truy nhập dễ dàng trở lại các biến này, bộ nhớ được đánh số, mỗi byte sẽ được ứng với một số nguyên, được gọi là địa chỉ của byte đó từ 0 đến hết bộ nhớ. Từ đó, mỗi biến (với tên biến) được gắn với một số nguyên là địa chỉ của byte đầu tiên mà biến đó được phân phối. Số lượng các byte phân phối cho biến là khác nhau (nhưng đặt liền nhau từ thấp đến cao) tuỳ thuộc kiểu dữ liệu của biến (và tuỳ thuộc vào quan niệm của từng NNLT), tuy nhiên chỉ cần biết tên biến hoặc địa chỉ của biến ta có thể đọc/viết dữ liệu vào/ra các biến đó. Từ đó ngoài việc thông qua tên biến chúng ta còn có thể thông qua địa chỉ của chúng để truy nhập vào nội dung. Tóm lại biến, ô nhớ và địa chỉ có quan hệ khăng khít với nhau. C++ cung cấp một toán tử một ngôi & để lấy địa chỉ của các biến (ngoại trừ biến mảng và xâu kí tự). Nếu x là một biến thì &x là địa chỉ của x. Từ đó câu lệnh sau cho ta biết x được bố trí ở đâu trong bộ nhớ:

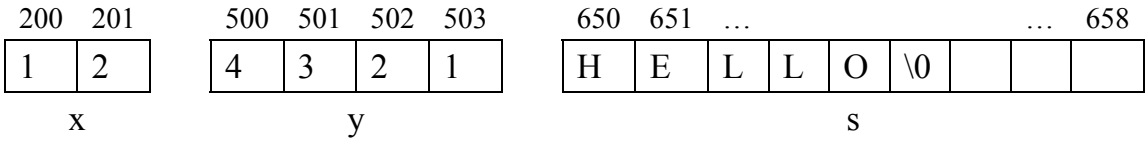
```
int x ;
cout << &x ;          // địa chỉ sẽ được hiện dưới dạng cơ số 16. Ví dụ 0xfff4
```

Đối với biến kiểu mảng, thì tên mảng chính là địa chỉ của mảng, do đó không cần dùng đến toán tử &. Ví dụ địa chỉ của mảng a chính là a (không phải &a). Mặt khác địa chỉ của mảng a cũng chính là địa chỉ của byte đầu tiên mà mảng a chiếm và nó cũng chính là địa chỉ của phần tử đầu tiên của mảng a. Do vậy địa chỉ của mảng a là địa chỉ của phần tử a[0] tức &a[0]. Tóm lại, địa chỉ của mảng a là a hoặc &a[0].

Tóm lại, cần nhớ:

```
int x;                  // khai báo biến nguyên x
long y;                 // khai báo biến nguyên dài y
cout << &x << &y;       // in địa chỉ các biến x, y
char s[9];              // khai báo mảng kí tự s
cout << a;              // in địa chỉ mảng s
cout << &a[0];          // in địa chỉ mảng s (tức địa chỉ s[0])
cout << &a[2];          // in địa chỉ kí tự s[2]
```

Hình vẽ sau đây minh hoạ một vài biến và địa chỉ của nó trong bộ nhớ.



Biến x chiếm 2 byte nhớ, có địa chỉ là 200, biến y có địa chỉ là 500 và chiếm 4 byte nhớ. Xâu s chiếm 9 byte nhớ tại địa chỉ 650. Các byte nhớ của một biến là liền nhau.

Các phép toán liên quan đến địa chỉ được gọi là số học địa chỉ. Tuy nhiên, chúng ta vẫn không được phép thao tác trực tiếp trên các địa chỉ như đặt biến vào địa chỉ này hay khác (công việc này do chương trình dịch đảm nhiệm), hay việc cộng, trừ hai địa chỉ với nhau là vô nghĩa … Các thao tác được phép trên địa chỉ vẫn phải thông qua các biến trung gian chứa địa chỉ, được gọi là biến con trỏ.

2. Con trỏ

a. Ý nghĩa

- Con trỏ là một biến chứa địa chỉ của biến khác. Nếu p là con trỏ chứa địa chỉ của biến x ta gọi p trỏ tới x và x được trỏ bởi p. Thông qua con trỏ ta có thể làm việc được với nội dung của những ô nhớ mà p trỏ đến.
- Để con trỏ p trỏ tới x ta phải gán địa chỉ của x cho p.

– Để làm việc với địa chỉ của các biến cần phải thông qua các biến con trỏ trỏ đến biến đó.

b. Khai báo biến con trỏ

<kiểu được trỏ> <*tên biến> ;

Địa chỉ của một biến là địa chỉ byte nhớ đầu tiên của biến đó. Vì vậy để lấy được nội dung của biến, con trỏ phải biết được số byte của biến, tức kiểu của biến mà con trỏ sẽ trỏ tới. Kiểu này cũng được gọi là kiểu của con trỏ. Như vậy khai báo biến con trỏ cũng giống như khai báo một biến thường ngoại trừ cần thêm dấu * trước tên biến (hoặc sau tên kiểu). Ví dụ:

```
int *p ;          // khai báo biến p là biến con trỏ trỏ đến kiểu dữ liệu nguyên.
float *q, *r ;    // hai con trỏ thực q và r.
```

*c. Sử dụng con trỏ, phép toán **

- Để con trỏ p trỏ đến biến x ta phải dùng phép gán p = địa chỉ của x.
  - Nếu x không phải là mảng ta viết: p = &x.
  - Nếu x là mảng ta viết: p = x hoặc p = &x[0].
- Không gán p cho một hằng địa chỉ cụ thể. Ví dụ viết p = 200 là sai.
- Phép toán * cho phép lấy nội dung nơi p trỏ đến, ví dụ để gán nội dung nơi p trỏ đến cho biến f ta viết f = *p.
- & và * là 2 phép toán ngược nhau. Cụ thể nếu p = &x thì x = *p. Từ đó nếu p trỏ đến x thì bất kỳ nơi nào xuất hiện x đều có thể thay được bởi *p và ngược lại.

Ví dụ 1 :

```
int i, j ;                  // khai báo 2 biến nguyên i, j
int *p, *q ;                // khai báo 2 con trỏ nguyên p, q
p = &i;                     // cho p trỏ tới i
q = &j;                     // cho q trỏ tới j
cout << &i ;                // hỏi địa chỉ biến i
cout << q ;                 // hỏi địa chỉ biến j (thông qua q)
i = 2;                      // gán i bằng 2
*q = 5;                     // gán j bằng 5 (thông qua q)
i++ ; cout << i ;           // tăng i và hỏi i, i = 3
```

```
(*q)++ ; cout << j ;          // tăng j (thông qua q) và hỏi j, j = 6
(*p) = (*q) * 2 + 1;          // gán lại i (thông qua p)
cout << i ;                   // 13
```

Qua ví dụ trên ta thấy mọi thao tác với i là tương đương với *p, với j là tương đương với *q và ngược lại.

3. Các phép toán với con trỏ

Trên đây ta đã trình bày về 2 phép toán một ngôi liên quan đến địa chỉ và con trỏ là & và *. Phần này chúng ta tiếp tục xét với các phép toán khác làm việc với con trỏ.

a. Phép toán gán

- Gán con trỏ với địa chỉ một biến: p = &x ;
- Gán con trỏ với con trỏ khác: p = q ; (sau phép toán gán này p, q chứa cùng một địa chỉ, cùng trỏ đến một nơi).

Ví dụ 2 :

```
int i = 10 ;                  // khai báo và khởi tạo biến i = 10
int *p, *q, *r ;              // khai báo 3 con trỏ nguyên p, q, r
p = q = r = &i ;              // cùng trỏ tới i
*p = q**q + 2**r + 1 ;        // i = 10*10 + 2*10 + 1
cout << i ;                   // 121
```

b. Phép toán tăng giảm địa chỉ

p ± n: con trỏ trỏ đến thành phần thứ n sau (trước) p.

Một đơn vị tăng giảm của con trỏ bằng kích thước của biến được trỏ. Ví dụ giả sử p là con trỏ nguyên (2 byte) đang trỏ đến địa chỉ 200 thì p+1 là con trỏ trỏ đến địa chỉ 202. Tương tự, p + 5 là con trỏ trỏ đến địa chỉ 210. p – 3 chứa địa chỉ 194.

194	195	196	197	198	199	200	201	202
p – 3						p		p + 1

Như vậy, phép toán tăng, giảm con trỏ cho phép làm việc thuận lợi trên mảng. Nếu con trỏ đang trỏ đến mảng (tức đang chứa địa chỉ đầu tiên của mảng), việc tăng con trỏ lên 1 đơn vị sẽ dịch chuyển con trỏ trỏ đến phần tử thứ hai, … Từ đó ta có thể cho con trỏ chạy từ đầu đến cuối mảng bằng cách tăng con trỏ lên từng đơn vị như trong câu lệnh for dưới đây.

Ví dụ 3 :

```
int a[100] = { 1, 2, 3, 4, 5, 6, 7 }, *p, *q;
p = a; cout << *p ;                    // cho p trỏ đến mảng a, *p = a[0] = 1
p += 5; cout << *p ;                                  // *p = a[5] = 6 ;
q = p - 4 ; cout << *q ;                              // q = a[1] = 2 ;
for (int i=0; i<100; i++) cout << *(p+i) ;            // in toàn bộ mảng a
```

c. Phép toán tự tăng giảm

p++, p--, ++p, --p: tương tự p+1 và p-1, có chú ý đến tăng (giảm) trước, sau.

Ví dụ 4 : Ví dụ sau minh hoạ kết quả kết hợp phép tự tăng giảm với lấy giá trị nơi con trỏ trỏ đến. a là một mảng gồm 2 số, p là con trỏ trỏ đến mảng a. Các lệnh dưới đây được qui ước là độc lập với nhau (tức lệnh sau không bị ảnh hưởng bởi lệnh trước, đối với mỗi lệnh p luôn luôn trỏ đến phần tử đầu (a[0]) của a.

```
int a[2] = {3, 7}, *p = a;
(*p)++ ;        // tăng (sau) giá trị nơi p trỏ ≡ tăng a[0] thành 4
++(*p) ;        // tăng (trước) giá trị nơi p trỏ ≡ tăng a[0] thành 4
*(p++) ;        // lấy giá trị nơi p trỏ (3) và tăng trỏ p (tăng sau), p → a[1]
*(++p) ;        // tăng trỏ p (tăng trước), p → a[1] và lấy giá trị nơi p trỏ (7)
```

Chú ý:

- Phân biệt p+1 và p++ (hoặc ++p):
- p+1 được xem như một con trỏ khác với p. p+1 trỏ đến phần tử sau p.
- p++ là con trỏ p nhưng trỏ đến phần tử khác. p++ trỏ đến phần tử đứng sau phần tử p trỏ đến ban đầu.
- Phân biệt *(p++) và *(++p):

Các phép toán tự tăng giảm cũng là một ngôi, mức ưu tiên của chúng là cao hơn các phép toán hai ngôi khác và cao hơn phép lấy giá trị (*). Cụ thể:

```
*p++        ≡    *(p++)
*++p        ≡    *(++p)
++*p        ≡    ++(*p)
```

Cũng giống các biến nguyên việc kết hợp các phép toán này với nhau rất dễ gây nhầm lẫn, do vậy cần sử dụng cặp dấu ngoặc để qui định trình tự tính toán.

d. Hiệu của 2 con trỏ

Phép toán này chỉ thực hiện được khi p và q là 2 con trỏ cùng trỏ đến các phần tử của một dãy dữ liệu nào đó trong bộ nhớ (ví dụ cùng trỏ đến 1 mảng dữ liệu). Khi đó hiệu p - q là số thành phần giữa p và q (chú ý p - q không phải là hiệu của 2 địa chỉ mà là số thành phần giữa p và q).

Ví dụ: giả sử p và q là 2 con trỏ nguyên, p có địa chỉ 200 và q có địa chỉ 208. Khi đó p - q = –4 và q - p = 4 (4 là số thành phần nguyên từ địa chỉ 200 đến 208).

e. Phép toán so sánh

Các phép toán so sánh cũng được áp dụng đối với con trỏ, thực chất là so sánh giữa địa chỉ của hai nơi được trỏ bởi các con trỏ này. Thông thường các phép so sánh <, <=, >, >= chỉ áp dụng cho hai con trỏ trỏ đến phần tử của cùng một mảng dữ liệu nào đó. Thực chất của phép so sánh này chính là so sánh chỉ số của 2 phần tử được trỏ bởi 2 con trỏ đó.

Ví dụ 5 :

```
float a[100], *p, *q ;
p = a ;                                  // p trỏ đến mảng (tức p trỏ đến a[0])
q = &a[3] ;                              // q trỏ đến phần tử thứ 3 (a[3]) của mảng
cout << (p < q) ;                        // 1
cout << (p + 3 == q) ;                   // 1
cout << (p > q - 1) ;                    // 0
cout << (p >= q - 2) ;                   // 0
for (p=a ; p < a+100; p++) cout << *p ;      // in toàn bộ mảng a
```

4. Cấp phát động, toán tử cấp phát, thu hồi new, delete

Khi tiến hành chạy chương trình, chương trình dịch sẽ bố trí các ô nhớ cụ thể cho các biến được khai báo trong chương trình. Vị trí cũng như số lượng các ô nhớ này tồn tại và cố định trong suốt thời gian chạy chương trình, chúng xem như đã bị chiếm dụng và sẽ không được sử dụng vào mục đích khác và chỉ được giải phóng sau khi chấm dứt chương trình. Việc phân bổ bộ nhớ như vậy được gọi là cấp phát tĩnh (vì được cấp sẵn trước khi chạy chương trình và không thể thay đổi tăng, giảm kích thước hoặc vị trí trong suốt quá trình chạy chương trình). Ví dụ nếu ta khai báo một mảng nguyên chứa 1000 số thì trong bộ nhớ sẽ có một vùng nhớ liên tục 2000 bytes để chứa dữ liệu của mảng này. Khi đó dù trong chương trình ta chỉ nhập vào mảng và làm việc với một vài số thì phần mảng rỗi còn lại vẫn không được sử dụng vào việc khác. Đây là hạn chế thứ nhất của kiểu mảng. Ở một hướng khác, một lần nào đó chạy chương trình ta lại

cần làm việc với hơn 1000 số nguyên. Khi đó vùng nhớ mà chương trình dịch đã dành cho mảng là không đủ để sử dụng. Đây chính là hạn chế thứ hai của mảng được khai báo trước.

Khắc phục các hạn chế trên của kiểu mảng, bây giờ chúng ta sẽ không khai báo (bố trí) trước mảng dữ liệu với kích thước cố định như vậy. Kích thước cụ thể sẽ được cấp phát trong quá trình chạy chương trình theo đúng yêu cầu của NSD. Nhờ vậy chúng ta có đủ số ô nhớ để làm việc mà vẫn tiết kiệm được bộ nhớ, và khi không dùng nữa ta có thể thu hồi (còn gọi là giải phóng) số ô nhớ này để chương trình sử dụng vào việc khác. Hai công việc cấp phát và thu hồi này được thực hiện thông qua các toán tử new, delete và con trỏ p. Thông qua p ta có thể làm việc với bất kỳ địa chỉ nào của vùng được cấp phát. Cách thức bố trí bộ nhớ như thế này được gọi là cấp phát động. Sau đây là cú pháp của câu lệnh new.

```
p = new <kiểu> ;                    // cấp phát 1 phần tử
p = new <kiểu>[n] ;                 // cấp phát n phần tử
```

Ví dụ:

```
int *p ;
p = new int       ;          // cấp phát vùng nhớ chứa được 1 số nguyên
p = float int[100] ;         // cấp phát vùng nhớ chứa được 100 số thực
```

Khi gặp toán tử new, chương trình sẽ tìm trong bộ nhớ một lượng ô nhớ còn rỗi và liên tục với số lượng đủ theo yêu cầu và cho p trỏ đến địa chỉ (byte đầu tiên) của vùng nhớ này. Nếu không có vùng nhớ với số lượng như vậy thì việc cấp phát là thất bại và p = NULL (NULL là một địa chỉ rỗng, không xác định). Do vậy ta có thể kiểm tra việc cấp phát có thành công hay không thông qua kiểm tra con trỏ p bằng hay khác NULL. Ví dụ:

```
float *p ;
int n ;
cout << "Số lượng cần cấp phát = "; cin >> n;
p = new double[n];
if (p == NULL) {
      cout << "Không đủ bộ nhớ" ;
      exit(0) ;
}
```

Ghi chú: lệnh exit(0) cho phép thoát khỏi chương trình, để sử dụng lệnh này cần khai báo file tiêu đề <process.h>.

Để giải phóng bộ nhớ đã cấp phát cho một biến (khi không cần sử dụng nữa) ta sử dụng câu lệnh delete.

delete p ; // p là con trỏ được sử dụng trong new

và để giải phóng toàn bộ mảng được cấp pháp thông qua con trỏ p ta dùng câu lệnh:

delete[] p ; // p là con trỏ trỏ đến mảng

Dưới đây là ví dụ sử dụng tổng hợp các phép toán trên con trỏ.

Ví dụ 1 : Nhập dãy số (không dùng mảng). Sắp xếp và in ra màn hình.

Trong ví dụ này chương trình xin cấp phát bộ nhớ đủ chứa n số nguyên và được trỏ bởi con trỏ head. Khi đó địa chỉ của số nguyên đầu tiên và cuối cùng sẽ là head và head+n-1. p và q là 2 con trỏ chạy trên dãy số này, so sánh và đổi nội dung của các số này với nhau để sắp thành dãy tăng dần và cuối cùng in kết quả.

```
main()
{
    int *head, *p, *q, n, tam;          // head trỏ đến (đánh dấu) đầu dãy
    cout << "Cho biết số số hạng của dãy: "); cin >> n ;
    head = new int[n] ;                 // cấp phát bộ nhớ chứa n số nguyên
    for (p=head; p<head+n; p++)         // nhập dãy
    {
        cout << "So thu " << p-head+1 << ": " ; cin >> *p ;
    }
    for (p=head; p<head+n-1; p++)                       // sắp xếp
    for (q=p+1; q<head+n; q++)
        if (*q < *p) { tam = *p; *p = *q; *q = tam; }   // đổi chỗ
    for (p=head; p<head+n; p++) cout << *p ;            // in kết quả
}
```

5. Con trỏ và mảng, xâu kí tự

a. Con trỏ và mảng 1 chiều

Việc cho con trỏ trỏ đến mảng cũng tương tự trỏ đến các biến khác, tức gán địa chỉ của mảng (chính là tên mảng) cho con trỏ. Chú ý rằng địa chỉ của mảng cũng là địa chỉ của thành phần thứ 0 nên a+i sẽ là địa chỉ thành phần thứ i của mảng. Tương tự, nếu p trỏ đến mảng a thì p+i là địa chỉ thành phần thứ i của mảng a và do đó *(p+i) =

a[i] = *(a+i).

Chú ý khi viết *(p+1) = *(a+1) ta thấy vai trò của p và a trong biểu thức này là như nhau, cùng truy cập đến giá trị của phần tử a[1]. Tuy nhiên khi viết *(p++) thì lại khác với *(a++), cụ thể viết p++ là hợp lệ còn a++ là không được phép. Lý do là tuy p và a cùng thể hiện địa chỉ của mảng a nhưng p thực sự là một biến, nó có thể thay đổi được giá trị còn a là một hằng, giá trị không được phép thay đổi. Ví dụ viết x = 3 và sau đó có thể tăng x bởi x++ nhưng không thể viết x = 3++.

Ví dụ 1 : In toàn bộ mảng thông qua con trỏ.

```
int a[5] = {1,2,3,4,5}, *p, i;
1:  p = a; for (i=1; i<=5; i++) cout << *(p+i);          // p không thay đổi
```

hoặc:

```
2: for (p=a; p<=a+4; p++) cout << *p ;                  // thay đổi p
```

Trong phương án 1, con trỏ p không thay đổi trong suốt quá trình làm việc của lệnh for, để truy nhập đến phần tử thứ i của mảng a ta sử dụng cú pháp *(p+i).

Đối với phương án 2 con trỏ sẽ dịch chuyển dọc theo mảng a bắt đầu từ địa chỉ a (phần tử đầu tiên) đến phần tử cuối cùng. Tại bước thứ i, p sẽ trỏ vào phần tử a[i], do đó ta chỉ cần in giá trị *p. Để kiểm tra khi nào p đạt đến phần tử cuối cùng, ta có thể so sánh p với địa chỉ cuối mảng chính là địa chỉ đầu mảng cộng thêm số phần tử trong a và trừ 1 (tức a+4 trong ví dụ trên).

b. Con trỏ và xâu kí tự

Một con trỏ kí tự có thể xem như một biến xâu kí tự, trong đó xâu chính là tất cả các kí tự kể từ byte con trỏ trỏ đến cho đến byte '\0' gặp đầu tiên. Vì vậy ta có thể khai báo các xâu dưới dạng con trỏ kí tự như sau.

```
char *s ;
char *s = "Hello" ;
```

Các hàm trên xâu vẫn được sử dụng như khi ta khai báo nó dưới dạng mảng kí tự. Ngoài ra khác với mảng kí tự, ta được phép sử dụng phép gán cho 2 xâu dưới dạng con trỏ, ví dụ:

```
char *s, *t = "Tin học" ; s = t;            // thay cho hàm strcpy(s, t) ;
```

Thực chất phép gán trên chỉ là gán 2 con trỏ với nhau, nó cho phép s bây giờ cũng được trỏ đến nơi mà t trỏ (tức dãy kí tự "Tin học" đã bố trí sẵn trong bộ nhớ)

Khi khai báo xâu dạng con trỏ nó vẫn chưa có bộ nhớ cụ thể, vì vậy thông thường kèm theo khai báo ta cần phải xin cấp phát bộ nhớ cho xâu với độ dài cần thiết. Ví dụ:

```
char *s = new char[30], *t ;
```

```
strcpy(s, "Hello") ;          // trong trường hợp này không cần cấp phát bộ
t = s ;                       // nhớ cho t vì t và s cùng sử dụng chung vùng nhớ
```

nhưng:

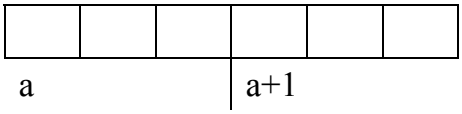
```
char *s = new char[30], *t ;
strcpy(s, "Hello") ;
t = new char[30];             // trong trường hợp này phải cấp bộ nhớ cho t vì
strcpy(t, s) ;                // có chỗ để strcpy sao chép sang nội dung của s.
```

c. Con trỏ và mảng hai chiều

Để dễ hiểu việc sử dụng con trỏ trỏ đến mảng hai chiều, chúng ta nhắc lại về mảng 2 chiều thông qua ví dụ. Giả sử ta có khai báo:

```
float a[2][3], *p;
```

khi đó a được bố trí trong bộ nhớ như là một dãy 6 phần tử float như sau



tuy nhiên a không được xem là mảng 1 chiều với 6 phần tử mà được quan niệm như mảng một chiều gồm 2 phần tử, mỗi phần tử là 1 bộ 3 số thực. Do đó địa chỉ của mảng a chính là địa chỉ của phần tử đầu tiên a[0][0], và a+1 không phải là địa chỉ của phần tử tiếp theo a[0][1] mà là địa chỉ của phần tử a[1][0]. Nói cách khác a+1 cũng là tăng địa chỉ của a lên một thành phần, nhưng 1 thành phần ở đây được hiểu là toàn bộ một dòng của mảng.

Mặt khác, việc lấy địa chỉ của từng phần tử (float) trong a thường là không chính xác. Ví dụ: viết &a[i][j] (địa chỉ của phần tử dòng i cột j) là được đối với mảng nguyên nhưng lại không đúng đối với mảng thực.

Từ các thảo luận trên, phép gán p = a là dễ gây nhầm lẫn vì p là con trỏ float còn a là địa chỉ mảng (1 chiều). Do vậy trước khi gán ta cần ép kiểu của a về kiểu float. Tóm lại cách gán địa chỉ của a cho con trỏ p được thực hiện như sau:

Cách sai:

```
p = a ;                       // sai vì khác kiểu
```

Các cách đúng:

```
p = (float*)a;                // ép kiểu của a về con trỏ float (cũng là kiểu của p)
p = a[0];                     // gán với địa chỉ của mảng a[0]
p = &a[0][0];                 // gán với địa chỉ số thực đầu tiên trong a
```

trong đó cách dùng p = (float*)a; là trực quan và đúng trong mọi trường hợp nên được dùng thông dụng hơn cả.

Sau khi gán a cho p (p là con trỏ thực), việc tăng giảm p chính là dịch chuyển con trỏ trên từng phần tử (thực) của a. Tức:

```
p      trỏ tới a[0][0]
p+1    trỏ tới a[0][1]
p+2    trỏ tới a[0][2]
p+3    trỏ tới a[1][0]
p+4    trỏ tới a[1][1]
p+5    trỏ tới a[1][2]
```

Tổng quát, đối với mảng m x n phần tử:

```
p + i*n + j trỏ tới a[i][j]      hoặc      a[i][j] = *(p + i*n + j)
```

Từ đó để truy nhập đến phần tử a[i][j] thông qua con trỏ p ta nên sử dụng cách viết sau:

```
p = (float*)a;
cin >> *(p+i*n+j) ;          // nhập cho a[i][j]
cout << *(p+i*n+j);          // in a[i][j]
```

Ví dụ sau đây cho phép nhập và in một mảng 2 chiều m*n (m dòng, n cột) thông qua con trỏ p. Nhập liên tiếp m*n số vào mảng và in thành ma trận m dòng, n cột.

```
main()
{
    clrscr();
    float a[m][n], *p;
    int i, j;
    p = (float*) a;
    for (i=0; i<m*n; i++) cin >> *(p+i);          // nhập như dãy mxn phần tử
    *(p+2*n+3) = 100; *(p+4*n) = 100;             // gán a[2,3] = a[4][0] = 100
    for (i=0; i<m; i++)                           // in lại dưới dạng ma trận
    {
        for (j=0; j<n; j++) cout << *(p+i*n+j);
        cout << endl;
```

```
        }
        getch();
}
```

Chú ý: việc lấy địa chỉ phần tử a[i][j] của mảng thực a là không chính xác. Tức: viết p = &a[i][j] có thể dẫn đến kết quả sai.

6. Mảng con trỏ

a. Khái niệm chung

Thực chất một con trỏ cũng là một biến thông thường có tên gọi (ví dụ p, q, …), do đó cũng giống như biến, nhiều biến cùng kiểu có thể tổ chức thành một mảng với tên gọi chung, ở đây cũng vậy nhiều con trỏ cùng kiểu cũng được tổ chức thành mảng. Như vậy mỗi phần tử của mảng con trỏ là một con trỏ trỏ đến một mảng nào đó. Nói cách khác một mảng con trỏ cho phép quản lý nhiều mảng dữ liệu cùng kiểu. Cách khai báo:

<kiểu> *a[size];

Ví dụ:

```
int *a[10];
```

khai báo một mảng chứa 10 con trỏ. Mỗi con trỏ a[i] chứa địa chỉ của một mảng nguyên nào đó.

b. Mảng xâu kí tự

Là trường hợp riêng của mảng con trỏ nói chung, trong đó kiểu cụ thể là char. Mỗi thành phần mảng là một con trỏ trỏ đến một xâu kí tự, có nghĩa các thao tác tiến hành trên *a[i] như đối với một xâu kí tự.

Ví dụ 1 : Nhập vào và in ra một bài thơ.

```
main()
{
        clrscr();
        char *dong[100];                    // khai báo 100 con trỏ kí tự (100 dòng)
        int i, n;
        cout << "so dong = "; cin >> n ;  // nhập số dòng thực sự
        cin.ignore();                       // loại dấu ↵ trong lệnh cin ở trên
        for (i=0; i<n; i++)
```

```
    {
        dong[i] = new char[80];      // cấp bộ nhớ cho dòng i
        cin.getline(dong[i],80);     // nhập dòng i
    }
    for (i=0; i<n; i++) cout << dong[i] << endl;   // in kết quả
    getch();
}
```

II. HÀM

Hàm là một chương trình con trong chương trình lớn. Hàm nhận (hoặc không) các đối số và trả lại (hoặc không) một giá trị cho chương trình gọi nó. Trong trường hợp không trả lại giá trị, hàm hoạt động như một thủ tục trong các NNLT khác. Một chương trình là tập các hàm, trong đó có một hàm chính với tên gọi main(), khi chạy chương trình, hàm main() sẽ được chạy đầu tiên và gọi đến hàm khác. Kết thúc hàm main() cũng là kết thúc chương trình.

Hàm giúp cho việc phân đoạn chương trình thành những môđun riêng rẽ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, dễ cho việc kiểm tra và bảo trì chương trình. Hàm có một số đặc trưng:

- Nằm trong hoặc ngoài văn bản có chương trình gọi đến hàm. Trong một văn bản có thể chứa nhiều hàm,
- Được gọi từ chương trình chính (main), từ hàm khác hoặc từ chính nó (đệ quy),
- Không lồng nhau.
- Có 3 cách truyền giá trị: Truyền theo tham trị, tham biến và tham trỏ.

1. Khai báo và định nghĩa hàm

a. Khai báo

Một hàm thường làm chức năng: tính toán trên các tham đối và cho lại giá trị kết quả, hoặc chỉ đơn thuần thực hiện một chức năng nào đó, không trả lại kết quả tính toán. Thông thường kiểu của giá trị trả lại được gọi là kiểu của hàm. Các hàm thường được khai báo ở đầu chương trình. Các hàm viết sẵn được khai báo trong các file nguyên mẫu *.h. Do đó, để sử dụng được các hàm này, cần có chỉ thị #include <*.h> ở ngay đầu chương trình, trong đó *.h là tên file cụ thể có chứa khai báo của các hàm

được sử dụng (ví dụ để sử dụng các hàm toán học ta cần khai báo file nguyên mẫu math.h). Đối với các hàm do NSD tự viết, cũng cần phải khai báo. Khai báo một hàm như sau:

<kiểu giá trị trả lại> <tên hàm>(d/s kiểu đối) ;

trong đó, kiểu giá trị trả lại còn gọi là kiểu hàm và có thể nhận kiểu bất kỳ chuẩn của C++ và cả kiểu của NSD tự tạo. Đặc biệt nếu hàm không trả lại giá trị thì kiểu của giá trị trả lại được khai báo là **void**. Nếu kiểu giá trị trả lại được bỏ qua thì chương trình ngầm định hàm có kiểu là **int** (phân biệt với void !).

Ví dụ 1 :

```
int bp(int);              // Khai báo hàm bp, có đối kiểu int và kiểu hàm là int
int rand100();            // Không đối, kiểu hàm (giá trị trả lại) là int
void alltrim(char[]) ;    // đối là xâu kí tự, hàm không trả lại giá trị (không kiểu).
cong(int, int);           // Hai đối kiểu int, kiểu hàm là int (ngầm định).
```

Thông thường để chương trình được rõ ràng chúng ta nên tránh lạm dụng các ngầm định. Ví dụ trong khai báo cong(int, int); nên khai báo rõ cả kiểu hàm (trong trường hợp này kiểu hàm ngầm định là int) như sau : int cong(int, int);

b. Định nghĩa hàm

Cấu trúc một hàm bất kỳ được bố trí cũng giống như hàm main() trong các phần trước. Cụ thể:

- Hàm có trả về giá trị

```
<kiểu hàm> <tên hàm>(danh sách tham đối hình thức)
{
    khai báo cục bộ của hàm ;     // chỉ dùng riêng cho hàm này
    dãy lệnh của hàm ;
    return (biểu thức trả về);    // có thể nằm đâu đó trong dãy lệnh.
}
```

– Danh sách tham đối hình thức còn được gọi ngắn gọn là danh sách đối gồm dãy các đối cách nhau bởi dấu phẩy, đối có thể là một biến thường, biến tham chiếu hoặc biến con trỏ, hai loại biến sau ta sẽ trình bày trong các phần tới. Mỗi đối được khai báo giống như khai báo biến, tức là cặp gồm <kiểu đối> <tên đối>.

– Với hàm có trả lại giá trị cần có câu lệnh **return** kèm theo sau là một biểu thức. Kiểu của giá trị biểu thức này chính là kiểu của hàm đã được khai báo ở

phần tên hàm. Câu lênh return có thể nằm ở vị trí bất kỳ trong phần câu lệnh, tuỳ thuộc mục đích của hàm. Khi gặp câu lệnh return chương trình tức khắc thoát khỏi hàm và trả lại giá trị của biểu thức sau return như giá trị của hàm.

Ví dụ 2 : Ví dụ sau định nghĩa hàm tính luỹ thừa n (với n nguyên) của một số thực bất kỳ. Hàm này có hai đầu vào (đối thực x và số mũ nguyên n) và đầu ra (giá trị trả lại) kiểu thực với độ chính xác gấp đôi là x^n.

```
double luythua(float x, int n)
{
    int i ;                     // biến chỉ số
    double kq = 1 ;             // để lưu kết quả
    for (i=1; i<=n; i++) kết quả *= x ;
    return kq;
}
```

- Hàm không trả về giá trị

Nếu hàm không trả lại giá trị (tức kiểu hàm là void), khi đó có thể có hoặc không có câu lệnh return, nếu có thì đằng sau return sẽ không có biểu thức giá trị trả lại.

Ví dụ 3 : Hàm xoá màn hình 100 lần, hàm chỉ làm công việc cẩn thận xoá màn hình nhiều lần để màn hình thật sạch, nên không có giá trị gì để trả lại.

```
void xmh()
{
    int i;
    for (i=1; i<=100; i++) clrscr();
    return ;
}
```

Hàm main() thông thường có hoặc không có giá trị trả về cho hệ điều hành khi chương trình chạy xong, vì vậy ta thường khai báo kiểu hàm là int main() hoặc void main() và câu lệnh cuối cùng trong hàm thường là return 1 hoặc return. Trường hợp bỏ qua từ khoá void nhưng trong thân hàm không có câu lệnh return (giống phần lớn ví dụ trong giáo trình này) chương trình sẽ ngầm hiểu hàm main() trả lại một giá trị nguyên nhưng vì không có nên khi dịch chương trình ta sẽ gặp lời cảnh báo "Cần có giá trị trả lại cho hàm" (một lời cảnh báo không phải là lỗi, chương trình vẫn chạy bình thường). Để tránh bị quấy rầy về những lời cảnh báo "không mời" này chúng ta có thể đặt thêm câu lệnh return 0; (nếu không khai báo void main()) hoặc khai báo kiểu hàm là void main() và đặt câu lệnh return vào cuối hàm.

c. Chú ý về khai báo và định nghĩa hàm

- Danh sách đối trong khai báo hàm có thể chứa hoặc không chứa tên đối, thông thường ta chỉ khai báo kiểu đối chứ không cần khai báo tên đối, trong khi ở dòng đầu tiên của định nghĩa hàm phải có tên đối đầy đủ.
- Cuối khai báo hàm phải có dấu chấm phẩy (;), trong khi cuối dòng đầu tiên của định nghĩa hàm không có dấu chấm phẩy.
- Hàm có thể không có đối (danh sách đối rỗng), tuy nhiên cặp dấu ngoặc sau tên hàm vẫn phải được viết. Ví dụ clrscr(), lamtho(), vietgiaotrinh(), …
- Một hàm có thể không cần phải khai báo nếu nó được định nghĩa trước khi có hàm nào đó gọi đến nó. Ví dụ có thể viết hàm main() trước (trong văn bản chương trình), rồi sau đó mới viết đến các hàm "con". Do trong hàm main() chắc chắn sẽ gọi đến hàm con này nên danh sách của chúng phải được khai báo trước hàm main(). Trường hợp ngược lại nếu các hàm con được viết (định nghĩa) trước thì không cần phải khai báo chúng nữa (vì trong định nghĩa đã hàm ý khai báo). Nguyên tắc này áp dụng cho hai hàm A, B bất kỳ chứ không riêng cho hàm main(), nghĩa là nếu B gọi đến A thì trước đó A phải được định nghĩa hoặc ít nhất cũng có dòng khai báo về A.

2. Lời gọi và sử dụng hàm

Lời gọi hàm được phép xuất hiện trong bất kỳ biểu thức, câu lệnh của hàm khác … Nếu lời gọi hàm lại nằm trong chính bản thân hàm đó thì ta gọi là đệ quy. Để gọi hàm ta chỉ cần viết tên hàm và danh sách các giá trị cụ thể truyền cho các đối đặt trong cặp dấu ngoặc tròn ().

tên hàm(danh sách tham đối thực sự) ;

- Danh sách tham đối thực sự còn gọi là danh sách giá trị gồm các giá trị cụ thể để gán lần lượt cho các đối hình thức của hàm. Khi hàm được gọi thực hiện thì tất cả những vị trí xuất hiện của đối hình thức sẽ được gán cho giá trị cụ thể của đối thực sự tương ứng trong danh sách, sau đó hàm tiến hành thực hiện các câu lệnh của hàm (để tính kết quả).
- Danh sách tham đối thực sự truyền cho tham đối hình thức có số lượng bằng với số lượng đối trong hàm và được truyền cho đối theo thứ tự tương ứng. Các tham đối thực sự có thể là các hằng, các biến hoặc biểu thức. Biến trong giá trị có thể trùng với tên đối. Ví dụ ta có hàm in n lần kí tự c với tên hàm inkitu(int n, char c); và lời gọi hàm inkitu(12, 'A'); thì n và c là các đối hình thức, 12 và 'A' là các đối thực sự hoặc giá trị. Các đối hình thức n và c sẽ lần lượt được gán bằng các giá trị tương ứng là 12 và 'A' trước khi tiến hành các câu lệnh trong phần thân hàm. Giả sử hàm in kí tự được khai báo lại thành inkitu(char

c, int n); thì lời gọi hàm cũng phải được thay lại thành inkitu('A', 12).

- Các giá trị tương ứng được truyền cho đối phải có kiểu cùng với kiểu đối (hoặc C++ có thể tự động chuyển kiểu được về kiểu của đối).
- Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện dòng lệnh đầu tiên trong hàm được gọi. Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

Ví dụ 4 : Giả sử ta cần tính giá trị của biểu thức $2x^3 - 5x^2 - 4x + 1$, thay cho việc tính trực tiếp x^3 và x^2, ta có thể gọi hàm luythua() trong ví dụ trên để tính các giá trị này bằng cách gọi nó trong hàm main() như sau:

```
#include <iostream.h>
#include <iomanip.h>
double luythua(float x, int n)                  // trả lại giá trị x^n
{
      int i ;                                   // biến chỉ số
      double kq = 1 ;                           // để lưu kết quả
      for (i=1; i<=n; i++) kết quả *= x ;
      return kq;
}

void xmh(int n)                                 // xoá màn hình n lần
{
      int i;
      for (i=1; i<=n; i++) clrscr();
      return ;
}

main()                                          // tính giá trị 2x^3 - 5x^2 - 4x + 1
{
      float x ;                                 // tên biến có thể trùng với đối của hàm
      double f ;                                // để lưu kết quả
      cout << "x = " ; cin >> x
      f = 2*luythua(x,3) - 5*luythua(x,2) - 4*x + 1;
      xmh(100);                                 // xoá thật sạch màn hình 100 lần
```

```
        cout << setprecision(2) << f << endl ;
}
```

Qua ví dụ này ta thấy lợi ích của lập trình cấu trúc, chương trình trở nên gọn hơn, chẳng hạn hàm luythua() chỉ được viết một lần nhưng có thể sử dụng nó nhiều lần (2 lần trong ví dụ này) chỉ bằng một câu lệnh gọi đơn giản cho mỗi lần sử dụng thay vì phải viết lại nhiều lần đoạn lệnh tính luỹ thừa.

3. Hàm với đối mặc định

Mục này và mục sau chúng ta bàn đến một vài mở rộng thiết thực của C++ đối với C có liên quan đến hàm, đó là hàm với đối mặc định và cách tạo, sử dụng các hàm có chung tên gọi. Một mở rộng quan trọng khác là cách truyền đối theo tham chiếu sẽ được bàn chung trong mục truyền tham đối thực sự cho hàm.

Trong phần trước chúng ta đã khẳng định số lượng tham đối thực sự phải bằng số lượng tham đối hình thức khi gọi hàm. Tuy nhiên, trong thực tế rất nhiều lần hàm được gọi với các giá trị của một số tham đối hình thức được lặp đi lặp lại. Trong trường hợp như vậy lúc nào cũng phải viết một danh sách dài các tham đối thực sự giống nhau cho mỗi lần gọi là một công việc không mấy thú vị. Từ thực tế đó C++ đưa ra một cú pháp mới về hàm sao cho một danh sách tham đối thực sự trong lời gọi không nhất thiết phải viết đầy đủ nếu một số trong chúng đã có sẵn những giá trị định trước. Cú pháp này được gọi là hàm với tham đối mặc định và được khai báo với cú pháp như sau:

<kiểu hàm> <tên hàm>(đ1, …, đn, đmđ1 = gt1, …, đmđm = gtm) ;

- Các đối đ1, …, đn và đối mặc định đmđ1, …, đmđm đều được khai báo như cũ nghĩa là gồm có kiểu đối và tên đối.
- Riêng các đối mặc định đmđ1, …, đmđm có gán thêm các giá trị mặc định gt1, …, gtm. Một lời gọi bất kỳ khi gọi đến hàm này đều phải có đầy đủ các tham đối thực sự ứng với các đ1, …, đm nhưng có thể có hoặc không các tham đối thực sự ứng với các đối mặc định đmđ1, …, đmđm. Nếu tham đối nào không có tham đối thực sự thì nó sẽ được tự động gán giá trị mặc định đã khai báo.

Ví dụ 5 :

- Xét hàm xmh(int n = 100), trong đó n mặc định là 100, nghĩa là nếu gọi xmh(99) thì màn hình được xoá 99 lần, còn nếu gọi xmh(100) hoặc gọn hơn xmh() thì chương trình sẽ xoá màn hình 100 lần.
- Tương tự, xét hàm int luythua(float x, int n = 2); Hàm này có một tham đối mặc định là số mũ n, nếu lời gọi hàm bỏ qua số mũ này thì chương trình hiểu là tính bình phương của x (n = 2). Ví dụ lời gọi luythua(4, 3) được hiểu là 4^3

còn luythua(4) được hiểu là 4^2.

– Hàm tính tổng 4 số nguyên: int tong(int m, int n, int i = 0; int j = 0); khi đó có thể tính tổng của 5, 2, 3, 7 bằng lời gọi hàm **tong(5,2,3,7)** hoặc có thể chỉ tính tổng 3 số 4, 2, 1 bằng lời gọi **tong(4,2,1)** hoặc cũng có thể gọi **tong(6,4)** chỉ để tính tổng của 2 số 6 và 4.

Chú ý: Các đối ngầm định phải được khai báo liên tục và xuất hiện cuối cùng trong danh sách đối. Ví dụ:

```
int tong(int x, int y=2, int z, int t=1);     // sai vì các đối mặc định không liên tục
void xoa(int x=0, int y)                      // sai vì đối mặc định không ở cuối
```

4. Khai báo hàm trùng tên

Hàm trùng tên hay còn gọi là hàm chồng (đè). Đây là một kỹ thuật cho phép sử dụng cùng một tên gọi cho các hàm "giống nhau" (cùng mục đích) nhưng xử lý trên các kiểu dữ liệu khác nhau hoặc trên số lượng dữ liệu khác nhau. Ví dụ hàm sau tìm số lớn nhất trong 2 số nguyên:

```
int max(int a, int b) { return (a > b) ? a: b ; }
```

Nếu đặt c = max(3, 5) ta sẽ có c = 5. Tuy nhiên cũng tương tự như vậy nếu đặt c = max(3.0, 5.0) chương trình sẽ bị lỗi vì các giá trị (float) không phù hợp về kiểu (int) của đối trong hàm max. Trong trường hợp như vậy chúng ta phải viết hàm mới để tính max của 2 số thực. Mục đích, cách làm việc của hàm này hoàn toàn giống hàm trước, tuy nhiên trong C và các NNLT cổ điển khác chúng ta buộc phải sử dụng một tên mới cho hàm "mới" này. Ví dụ:

```
float fmax(float a, float b) { return (a > b) ? a: b ; }
```

Tương tự để tuận tiện ta sẽ viết thêm các hàm

```
char cmax(char a, char b) { return (a > b) ? a: b ; }
long lmax(long a, long b) { return (a > b) ? a: b ; }
double dmax(double a, double b) { return (a > b) ? a: b ; }
```

Tóm lại ta sẽ có 5 hàm: max, cmax, fmax, lmax, dmax, việc sử dụng tên như vậy sẽ gây bất lợi khi cần gọi hàm. C++ cho phép ta có thể khai báo và định nghĩa cả 5 hàm trên với cùng 1 tên gọi ví dụ là max chẳng hạn. Khi đó ta có 5 hàm:

```
1:  int max(int a, int b) { return (a > b) ? a: b ; }
2:  float max(float a, float b) { return (a > b) ? a: b ; }
3:  char max(char a, char b) { return (a > b) ? a: b ; }
4:  long max(long a, long b) { return (a > b) ? a: b ; }
```

```
5:    double max(double a, double b) { return (a > b) ? a: b ; }
```

Và lời gọi hàm bất kỳ dạng nào như max(3,5), max(3.0,5), max('O', 'K') đều được đáp ứng. Chúng ta có thể đặt ra vấn đề: với cả 5 hàm cùng tên như vậy, chương trình gọi đến hàm nào. Vấn đề được giải quyết dễ dàng vì chương trình sẽ dựa vào kiểu của các đối khi gọi để quyết định chạy hàm nào. Ví dụ lời gọi max(3,5) có 2 đối đều là kiểu nguyên nên chương trình sẽ gọi hàm 1, lời gọi max(3.0,5) hướng đến hàm số 2 và tương tự chương trình sẽ chạy hàm số 3 khi gặp lời gọi max('O','K'). Như vậy một đặc điểm của các hàm trùng tên đó là trong danh sách đối của chúng phải có ít nhất một cặp đối nào đó khác kiểu nhau. Một đặc trưng khác để phân biệt thông qua các đối đó là số lượng đối trong các hàm phải khác nhau (nếu kiểu của chúng là giống nhau).

Ví dụ việc vẽ các hình: thẳng, tam giác, vuông, chữ nhật trên màn hình là giống nhau, chúng chỉ phụ thuộc vào số lượng các điểm nối và toạ độ của chúng. Do vậy ta có thể khai báo và định nghĩa 4 hàm vẽ nói trên với cùng chung tên gọi. Chẳng hạn:

```
void ve(Diem A, Diem B) ;                    // vẽ đường thẳng AB
void ve(Diem A, Diem B, Diem C) ;            // vẽ tam giác ABC
void ve(Diem A, Diem B, Diem C, Diem D) ;    // vẽ tứ giác ABCD
```

trong ví dụ trên ta giả thiết Diem là một kiểu dữ liệu lưu toạ độ của các điểm trên màn hình. Hàm ve(Diem A, Diem B, Diem C, Diem D) sẽ vẽ hình vuông, chữ nhật, thoi, bình hành hay hình thang phụ thuộc vào toạ độ của 4 điểm ABCD, nói chung nó được sử dụng để vẽ một tứ giác bất kỳ.

Tóm lại nhiều hàm có thể được định nghĩa chồng (với cùng tên gọi giống nhau) nếu chúng thoả các điều kiện sau:

- Số lượng các tham đối trong hàm là khác nhau, hoặc
- Kiểu của tham đối trong hàm là khác nhau.

Kỹ thuật chồng tên này còn áp dụng cả cho các toán tử. Trong phần lập trình hướng đối tượng, ta sẽ thấy NSD được phép định nghĩa các toán tử mới nhưng vẫn lấy tên cũ như +, -, *, / …

5. Biến, đối tham chiếu

Một biến có thể được gán cho một bí danh mới, và khi đó chỗ nào xuất hiện biến thì cũng tương đương như dùng bí danh và ngược lại. Một bí danh như vậy được gọi là một biến tham chiếu, ý nghĩa thực tế của nó là cho phép "tham chiếu" tới một biến khác cùng kiểu của nó, tức sử dụng biến khác nhưng bằng tên của biến tham chiếu.

Giống khai báo biến bình thường, tuy nhiên trước tên biến ta thêm dấu và (&). Có thể tạm phân biến thành 3 loại: biến thường với tên thường, biến con trỏ với dấu * trước tên và biến tham chiếu với dấu &.

<kiểu biến> &<tên biến tham chiếu> = <tên biến được tham chiếu>;

Cú pháp khai báo này cho phép ta tạo ra một biến tham chiếu mới và cho nó tham chiếu đến biến được tham chiếu (cùng kiểu và phải được khai báo từ trước). Khi đó biến tham chiếu còn được gọi là bí danh của biến được tham chiếu. Chú ý không có cú pháp khai báo chỉ tên biến tham chiếu mà không kèm theo khởi tạo.

Ví dụ:

```
int hung, dung ;          // khai báo các biến nguyên hung, dung
int &ti = hung;           // khai báo biến tham chiếu ti, teo tham chieu đến
int &teo = dung;          // hung dung. ti, teo là bí danh của hung, dung
```

Từ vị trí này trở đi việc sử dụng các tên hung, ti hoặc dung, teo là như nhau.

Ví dụ:

```
hung = 2 ;
ti ++;                    // tương đương hung ++;
cout << hung << ti ;      // 3 3
teo = ti + hung ;         // tương đương dung = hung + hung
dung ++ ;                 // tương đương teo ++
cout << dung << teo ;           // 7 7
```

Vậy sử dụng thêm biến tham chiếu để làm gì ?

Cách tổ chức bên trong của một biến tham chiếu khác với biến thường ở chỗ nội dung của nó là địa chỉ của biến mà nó đại diện (giống biến con trỏ), ví dụ câu lệnh

```
cout << teo ;             // 7
```

in ra giá trị 7 nhưng thực chất đây không phải là nội dung của biến teo, nội dung của teo là địa chỉ của dung, khi cần in teo, chương trình sẽ tham chiếu đến dung và in ra nội dung của dung (7). Các hoạt động khác trên teo cũng vậy (ví dụ teo++), thực chất là tăng một đơn vị nội dung của dung (chứ không phải của teo). Từ cách tổ chức của biến tham chiếu ta thấy chúng giống con trỏ nhưng thuận lợi hơn ở chỗ khi truy cập đên giá trị của biến được tham chiếu (dung) ta chỉ cần ghi tên biến tham chiếu (teo) chứ không cần thêm toán tử (*) ở trước như trường hợp dùng con trỏ. Điểm khác biệt này có ích khi được sử dụng để truyền đối cho các hàm với mục đích làm thay đổi nội dung của biến ngoài. Tư tưởng này được trình bày rõ ràng hơn trong mục 6 của chương.

Chú ý:

- Biến tham chiếu phải được khởi tạo khi khai báo.

- Tuy giống con trỏ nhưng không dùng được các phép toán con trỏ cho biến tham chiếu. Nói chung chỉ nên dùng trong truyền đối cho hàm.

6. Các cách truyền tham đối

Có 3 cách truyền tham đối thực sự cho các tham đối hình thức trong lời gọi hàm. Trong đó cách ta đã dùng cho đến thời điểm hiện nay được gọi là truyền theo tham trị, tức các đối hình thức sẽ nhận các giá trị cụ thể từ lời gọi hàm và tiến hành tính toán rồi trả lại giá trị. Để dễ hiểu các cách truyền đối chúng ta sẽ xem qua cách thức chương trình thực hiện với các đối khi thực hiện hàm.

a. Truyền theo tham trị

Ta xét lại ví dụ hàm luythua(float x, int n) tính x^n. Giả sử trong chương trình chính ta có các biến a, b, f đang chứa các giá trị a = 2, b = 3, và f chưa có giá trị. Để tính a^b và gán giá trị tính được cho f, ta có thể gọi f = luythua(a,b). Khi gặp lời gọi này, chương trình sẽ tổ chức như sau:

- Tạo 2 biến mới (tức 2 ô nhớ trong bộ nhớ) có tên x và n. Gán nội dung các ô nhớ này bằng các giá trị trong lời gọi, tức gán 2 (a) cho x và 3 (b) cho n.
- Tới phần khai báo (của hàm), chương trình tạo thêm các ô nhớ mang tên kq và i.
- Tiến hành tính toán (gán lại kết quả cho kq).
- Cuối cùng lấy kết quả trong kq gán cho ô nhớ f (là ô nhớ có sẵn đã được khai báo trước, nằm bên ngoài hàm).
- Kết thúc hàm quay về chương trình gọi. Do hàm luythua đã hoàn thành xong việc tính toán nên các ô nhớ được tạo ra trong khi thực hiện hàm (x, n, kq, i) sẽ được xoá khỏi bộ nhớ. Kết quả tính toán được lưu giữ trong ô nhớ f (không bị xoá vì không liên quan gì đến hàm).

Trên đây là truyền đối theo cách thông thường. Vấn đề đặt ra là giả sử ngoài việc tính f, ta còn muốn thay đổi các giá trị của các ô nhớ a, b (khi truyền nó cho hàm) thì có thể thực hiện được không ? Để giải quyết bài toán này ta cần theo một kỹ thuật khác, nhờ vào vai trò của biến con trỏ và tham chiếu.

b. Truyền theo dẫn trỏ

Xét ví dụ tráo đổi giá trị của 2 biến. Đây là một yêu cầu nhỏ nhưng được gặp nhiều lần trong chương trình, ví dụ để sắp xếp một danh sách. Do vậy cần viết một hàm để thực hiện yêu cầu trên. Hàm không trả kết quả. Do các biến cần trao đổi là chưa được biết trước tại thời điểm viết hàm, nên ta phải đưa chúng vào hàm như các tham đối, tức hàm có hai tham đối x, y đại diện cho các biến sẽ thay đổi giá trị sau này.

Từ một vài nhận xét trên, theo thông thường hàm tráo đổi sẽ được viết như sau:

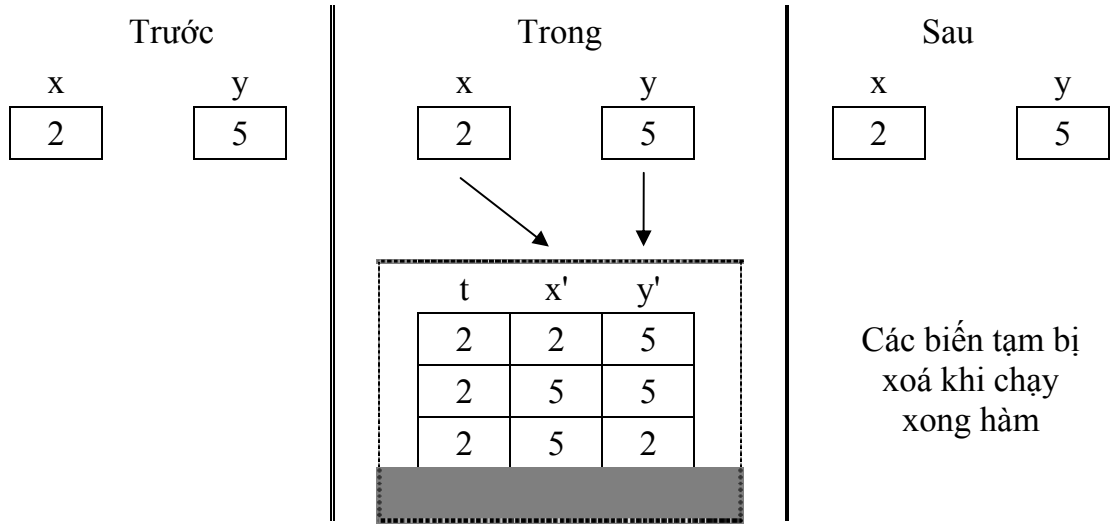
```
void swap(int x, int y)
{
    int t ;  t = x ; x = y ; y = t ;
}
```

Giả sử trong chương trình chính ta có 2 biến x, y chứa các giá trị lần lượt là 2, 5. Ta cần đổi nội dung 2 biến này sao cho x = 5 còn y = 2 bằng cách gọi đến hàm swap(x, y).

```
main()
{
    int x = 2; int y = 5;
    swap(x, y) ;
    cout << x << y ;                    // 2, 5 (x, y vẫn không đổi)
}
```

Thực sự sau khi chạy xong chương trình ta thấy giá trị của x và y vẫn không thay đổi !?.

Như đã giải thích trong mục trên (gọi hàm luythua), việc đầu tiên khi chương trình thực hiện một hàm là tạo ra các biến mới (các ô nhớ mới, độc lập với các ô nhớ x, y đã có sẵn) tương ứng với các tham đối, trong trường hợp này cũng có tên là x, y và gán nội dung của x, y (ngoài hàm) cho x, y (mới). Và việc cuối cùng của chương trình sau khi thực hiện xong hàm là xoá các biến mới này. Do vậy nội dung của các biến mới thực sự là có thay đổi, nhưng không ảnh hưởng gì đến các biến x, y cũ. Hình vẽ dưới đây minh hoạ cách làm việc của hàm swap, trước, trong và sau khi gọi hàm.



Như vậy hàm swap cần được viết lại sao cho việc thay đổi giá trị không thực hiện trên các biến tạm mà phải thực sự thực hiện trên các biến ngoài. Muốn vậy thay vì truyền giá trị của các biến ngoài cho đối, bây giờ ta sẽ truyền địa chỉ của nó cho đối, và các thay đổi sẽ phải thực hiện trên nội dung của các địa chỉ này. Đó chính là lý do ta phải sử dụng con trỏ để làm tham đối thay cho biến thường. Cụ thể hàm swap được viết lại như sau:

```
void swap(int *p, int *q)
{
    int t;              // khai báo biến tạm t
    t = *p ;            // đặt giá trị của t bằng nội dung nơi p trỏ tới
    *p = *q ;           // thay nội dung nơi p trỏ bằng nội dung nơi q trỏ
    *q = t ;            // thay nội dung nơi q trỏ tới bằng nội dung của t
}
```

Với cách tổ chức hàm như vậy rõ ràng nếu ta cho p trỏ tới biến x và q trỏ tới biến y thì hàm swap sẽ thực sự làm thay đổi nội dung của x, y chứ không phải của p, q.

Từ đó lời gọi hàm sẽ là swap(&x, &y) (tức truyền địa chỉ của x cho p, p trỏ tới x và tương tự q trỏ tới y).

Như vậy có thể tóm tắt 3 đặc trưng để viết một hàm làm thay đổi giá trị biến ngoài như sau:

- Đối của hàm phải là con trỏ (ví dụ int *p)
- Các thao tác liên quan đến đối này (trong thân hàm) phải thực hiện tại nơi nó trỏ đến (ví dụ *p = …)
- Lời gọi hàm phải chuyển địa chỉ cho p (ví dụ &x).

Ngoài hàm swap đã trình bày, ở đây ta đưa thêm ví dụ để thấy sự cần thiết phải có hàm cho phép thay đổi biến ngoài. Ví dụ hàm giải phương trình bậc 2 rất hay gặp trong các bài toán khoa học kỹ thuật. Tức cho trước 3 số a, b, c như 3 hệ số của phương trình, cần tìm 2 nghiệm x1, x2 của nó. Không thể lấy giá trị trả lại của hàm để làm nghiệm vì giá trị trả lại chỉ có 1 trong khi ta cần đến 2 nghiệm. Do vậy ta cần khai báo 2 biến "ngoài" trong chương trình để chứa các nghiệm, và hàm phải làm thay đổi 2 biến này (tức chứa giá trị nghiệm giải được). Như vậy hàm được viết cần phải có 5 đối, trong đó 3 đối a, b, c đại diện cho các hệ số, không thay đổi và 2 biến x1, x2 đại diện cho nghiệm, 2 đối này phải được khai báo dạng con trỏ. Ngoài ra, phương trình có thể vô nghiệm, 1 nghiệm hoặc 2 nghiệm do vậy hàm sẽ trả lại giá trị là số nghiệm của phương trình, trong trường hợp 1 nghiệm (nghiệm kép), giá trị nghiệm sẽ được cho vào x1.

Ví dụ 6 : Dưới đây là một dạng đơn giản của hàm giải phương trình bậc 2.

```
int gptb2(float a, float b, float c, float *p, float *q)
{
    float d ;                                   // để chứa Δ
    d = (b*b) - 4*a*c ;
    if (d < 0) return 0 ;
    else if (d == 0) { *p = -b/(2*a) ; return 1 ; }
    else {
        *p = (-b + sqrt(d))/(2*a) ;
        *q = (-b - sqrt(d))/(2*a) ;
        return 2 ;
    }
}
```

Một ví dụ của lời gọi hàm trong chương trình chính như sau:

```
main()
{
    float a, b, c ;                             // các hệ số
    float x1, x2 ;                              // các nghiệm
    cout << "Nhập hệ số: " ;
    cin >> a >> b >> c;
    switch (gptb2(a, b, c, &x1, &x2))
    {
        case 0: cout << "Phương trình vô nghiệm" ; break;
        case 1: cout << "Phương trình có nghiệm kép x = " << x1 ; break ;
        case 2: cout << "Phương trình có 2 nghiệm phân biệt:" << endl ;
        cout << "x1 = " << x1 << " và x2 = " << x2 << endl ; break;
    }
}
```

Trên đây chúng ta đã trình bày cách xây dựng các hàm cho phép thay đổi giá trị của biến ngoài. Một đặc trưng dễ nhận thấy là cách viết hàm tương đối phức tạp. Do vậy C++ đã phát triển một cách viết khác dựa trên đối tham chiếu và việc truyền đối cho hàm được gọi là truyền theo tham chiếu.

c. Truyền theo tham chiếu

Một hàm viết dưới dạng đối tham chiếu sẽ đơn giản hơn rất nhiều so với đối con trỏ và giống với cách viết bình thường (truyền theo tham trị), trong đó chỉ có một khác biệt đó là các đối khai báo dưới dạng tham chiếu.

Để so sánh 2 cách sử dụng ta nhắc lại các điểm khi viết hàm theo con trỏ phải chú ý đến, đó là:

- Đối của hàm phải là con trỏ (ví dụ int *p)
- Các thao tác liên quan đến đối này trong thân hàm phải thực hiện tại nơi nó trỏ đến (ví dụ *p = …)
- Lời gọi hàm phải chuyển địa chỉ cho p (ví dụ &x).

Hãy so sánh với đối tham chiếu, cụ thể:

- Đối của hàm phải là tham chiếu (ví dụ int &p)
- Các thao tác liên quan đến đối này phải thực hiện tại nơi nó trỏ đến, tức địa chỉ cần thao tác. Vì một thao tác trên biến tham chiếu thực chất là thao tác trên biến được nó tham chiếu nên trong hàm chỉ cần viết p trong mọi thao tác (thay vì *p như trong con trỏ)
- Lời gọi hàm phải chuyển địa chỉ cho p. Vì bản thân p khi tham chiếu đến biến nào thì sẽ chứa địa chỉ của biến đó, do đó lời gọi hàm chỉ cần ghi tên biến, ví dụ x (thay vì &x như đối với dẫn trỏ).

Tóm lại, đối với hàm viết theo tham chiếu chỉ thay đổi ở đối (là các tham chiếu) còn lại mọi nơi khác đều viết đơn giản như cách viết truyền theo tham trị.

Ví dụ 7 : Đổi giá trị 2 biến

```
void swap(int &x, int &y)
{
    int t = x; x = y; y = t;
}
```

và lời gọi hàm cũng đơn giản như trong truyền đối theo tham trị. Ví dụ:

```
int a = 5, b = 3;
swap(a, b);
cout << a << b;
```

Bảng dưới đây minh hoạ tóm tắt 3 cách viết hàm thông qua ví dụ đổi biến ở trên.

	Tham trị	Tham chiếu	Dẫn trỏ
Khai báo đối	void swap(int x, int y)	void swap(int &x, int &y)	void swap(int *x, int *y)
Câu lệnh	t = x; x = y; y = t;	t = x; x = y; y = t;	t = *x; *x = *y; *y = t;
Lời gọi	swap(a, b);	swap(a, b);	swap(&a, &b);
Tác dụng	a, b không thay đổi	a, b có thay đổi	a, b có thay đổi

7. Hàm và mảng dữ liệu

a. Truyền mảng 1 chiều cho hàm

Thông thường chúng ta hay xây dựng các hàm làm việc trên mảng như vectơ hay ma trận các phần tử. Khi đó tham đối thực sự của hàm sẽ là các mảng dữ liệu này. Trong trường hợp này ta có 2 cách khai báo đối. Cách thứ nhất đối được khai báo bình thường như khai báo biến mảng nhưng không cần có số phần tử kèm theo, ví dụ:

- int x[];
- float x[];

Cách thứ hai khai báo đối như một con trỏ kiểu phần tử mảng, ví dụ:

- int *p;
- float *p

Trong lời gọi hàm tên mảng a sẽ được viết vào danh sách tham đối thực sự, vì a là địa chỉ của phần tử đầu tiên của mảng a, nên khi hàm được gọi địa chỉ này sẽ gán cho con trỏ p. Vì vậy giá trị của phần tử thứ i của a có thể được truy cập bởi x[i] (theo khai báo 1) hoặc *(p+i) (theo khai báo 2) và nó cũng có thể được thay đổi thực sự (do đây cũng là cách truyền theo dẫn trỏ).

Sau đây là ví dụ đơn giản, nhập và in vectơ, minh hoạ cho cả 2 kiểu khai báo đối.

Ví dụ 8 : Hàm nhập và in giá trị 1 vectơ

```
void nhap(int x[], int n)                    // n: số phần tử
{
      int i;
      for (i=0; i<n; i++) cin >> x[i];       // hoặc cin >> *(x+i)
}
void in(int *p, int n)
{
```

```
    int i;
    for (i=0; i<n; i++) cout << *(p+i);
}
main()
{
    int a[10] ;                    // mảng a chứa tối đa 10 phần tử
    nhap(a,7);                     // vào 7 phần tử đầu tiên cho a
    in(a,3);                       // ra 3 phần tử đầu tiên của a
}
```

b. Truyền mảng 2 chiều cho hàm

Đối với mảng 2 chiều khai báo đối cũng như lời gọi là phức tạp hơn nhiều so với mảng 1 chiều. Ta có hai cách khai báo đối như sau:

– Khai báo theo đúng bản chất của mảng 2 chiều float x[m][n] do C++ qui định, tức x là mảng 1 chiều m phần tử, mỗi phần tử của nó có kiểu float[n]. Từ đó, đối được khai báo như một mảng hình thức 1 chiều (không cần số phần tử - ở đây là số dòng) của kiểu float[n]. Tức có thể khai báo như sau:

```
float x[][n] ;      // mảng với số phần tử không định trước, mỗi phần tử là n số
float (*x)[n] ;     // một con trỏ, có kiểu là mảng n số (float[n])
```

Để truy nhập đến đến phần tử thứ i, j ta vẫn sử dụng cú pháp x[i][j]. Tên của mảng a được viết bình thường trong lời gọi hàm. Nói chung theo cách khai báo này việc truy nhập là đơn giản nhưng phương pháp cũng có hạn chế đó là số cột của mảng truyền cho hàm phải cố định bằng n.

– Xem mảng float x[m][n] thực sự là mảng một chiều float x[m*n] và sử dụng cách khai báo như trong mảng một chiều, đó là sử dụng con trỏ float *p để truy cập được đến từng phần tử của mảng. Cách này có hạn chế trong lời gọi: địa chỉ truyền cho hàm không phải là mảng a mà cần phải ép kiểu về (float*) (để phù hợp với p). Với cách này gọi k là thứ tự của phần tử a[i][j] trong mảng một chiều (m*n), ta có quan hệ giữa k, i, j như sau:

  - k = *(p + i*n + j)
  - i = k/n
  - j = k%n

trong đó n là số cột của mảng truyền cho hàm. Điều này có nghĩa để truy cập đến a[i][j] ta có thể viết *(p+i*n+j), ngược lại biết chỉ số k có thể tính được dòng i, cột j

của phần tử này. Ưu điểm của cách khai báo này là ta có thể truyền mảng với kích thước bất kỳ (số cột không cần định trước) cho hàm.

Sau đây là các ví dụ minh hoạ cho 2 cách khai báo trên.

Ví dụ 9 : Tính tổng các số hạng trong ma trận

```
float tong(float x[][10], int m, int n)      // hoặc float tong(float (*x)[10], int m, int n)
{                                            // m: số dòng, n: số cột
    float t = 0;
    int i, j ;
    for (i=0; i<m; i++)
    for (j=0; j<n; j++) t += x[i][j] ;
    return t;
}

main()
{
    float a[8][10], b[5][7] ;
    int i, j, ma, na, mb, nb;
    cout << "nhập số dòng, số cột ma trận a: " ; cin >> ma >> na;
    for (i=0; i<ma; i++)                     // nhập ma trận a
    for (j=0; j<na; j++)
    { cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ; }
    cout << "nhập số dòng, số cột ma trận b: " ; cin >> mb >> nb;
    for (i=0; i<mb; i++)                     // nhập ma trận b
    for (j=0; j<nb; j++)
    { cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ; }
    cout << tong(a, ma, na);                 // in tổng các số trong ma trận
    cout << tong(b, mb, nb);                 // sai vì số cột của b khác 10
}
```

Ví dụ 10 : Tìm phần tử bé nhất của ma trận

```
void minmt(float *x, int m, int n)           // m: số dòng, n: số cột
{
```

```
        float min = *x;                         // gán phần tử đầu tiên cho min
        int k, kmin;
        for (k=1; k<m*n; k++)
        if (min > *(x+k)) { min = *(x+k) ; kmin = k; }
        cout << "Giá trị min la: " << min << " tại dòng " << k/n << " cột " << k%n;
}

main()
{
        float a[8][10], b[5][7] ;
        int i, j ;
        for (i=0; i<8; i++)                     // nhập ma trận a
        for (j=0; j<10; j++)
        { cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ; }
        for (i=0; i<5; i++)                     // nhập ma trận b
        for (j=0; j<7; j++)
        { cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ; }
        minmt((float*)a, 8, 10) ;       // in giá trị và vị trí số bé nhất trong a
        minmt((float*)b, 5, 7) ;        // in giá trị và vị trí số bé nhất trong b
}
```

Ví dụ 11 : Cộng 2 ma trận và in kết quả.

```
void inmt(float *x, int m, int n)
{
        int i, j;
        for (i=0; i<m; i++)
        {
                for (j=0; j<n; j++) cout << *(x+i*n+j);
                cout << endl;
        }
}

void cong(float *x, float *y, int m, int n)
```

```
{
    float *t = new float[m*n];          // t là ma trận kết quả (xem như dãy số)
    int k, i, j ;
    for (k = 0; k < m*n; k++) *(t+k) = *(x+k) + *(y+k) ;
    inmt((float*)t, m, n);
}

main()
{
    float a[8][10], b[5][7] ;
    int i, j, m, n;
    cout << "nhập số dòng, số cột ma trận: " ; cin >> m >> n;
    for (i=0; i<m; i++)                          // nhập ma trận a, b
    for (j=0; j<n; j++)
    {
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
        cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ;
    }
    cong((float*)a, (float*)b, m, n);            // cộng và in kết quả a+b
}
```

Xu hướng chung là chúng ta xem mảng (1 hoặc 2 chiều) như là một dãy liên tiếp các số trong bộ nhớ, tức một ma trận là một đối con trỏ trỏ đến thành phần của mảng. Đối với mảng 2 chiều m*n khi truyền đối địa chỉ của ma trận cần phải ép kiểu về kiểu con trỏ. Ngoài ra bước chạy k của con trỏ (từ 0 đến m*n-1) tương ứng với các toạ độ của phần tử a[i][j] trong mảng như sau:

- k = *(p + i*n + j)
- i = k/n
- j = k%n

từ đó, chúng ta có thể viết các hàm mà không cần phải băn khoăn gì về kích thước của ma trận sẽ truyền cho hàm.

c. Giá trị trả lại của hàm là một mảng

Không có cách nào để giá trị trả lại của một hàm là mảng. Tuy nhiên thực sự mỗi

mảng cũng chính là một con trỏ, vì vậy việc hàm trả lại một con trỏ trỏ đến dãy dữ liệu kết quả là tương đương với việc trả lại mảng. Ngoài ra còn một cách dễ dùng hơn đối với mảng 2 chiều là mảng kết quả được trả lại vào trong tham đối của hàm (giống như nghiệm của phương trình bậc 2 được trả lại vào trong các tham đối). Ở đây chúng ta sẽ lần lượt xét 2 cách làm việc này.

1. Giá trị trả lại là con trỏ trỏ đến mảng kết quả. Trước hết chúng ta xét ví dụ nhỏ sau đây:

```
int* tragiatri1()                    // giá trị trả lại là con trỏ trỏ đến dãy số nguyên
{
      int kq[3] = { 1, 2, 3 };       // tạo mảng kết quả với 3 giá trị 1, 2, 3
      return kq ;                    // trả lại địa chỉ cho con trỏ kết quả hàm
}

int* tragiatri2()                    // giá trị trả lại là con trỏ trỏ đến dãy số nguyên
{
      int *kq = new int[4];          // cấp phát 3 ô nhớ nguyên
      *kq = *(kq+1) = *(kq+2) = 0 ;  // tạo mảng kết quả với 3 giá trị 1, 2, 3
      return kq ;                    // trả lại địa chỉ cho con trỏ kết quả hàm
}

main()
{
      int *a, i;
      a = tragiatri1();
      for (i=0; i<3; i++) cout *(a+i);     // không phải là 1, 2, 3
      a = tragiatri2();
      for (i=0; i<3; i++) cout *(a+i);     // 1, 2, 3
}
```

Qua ví dụ trên ta thấy hai hàm trả giá trị đều tạo bên trong nó một mảng 3 số nguyên và trả lại địa chỉ mảng này cho con trỏ kết quả hàm. Tuy nhiên, chỉ có tragiatri2() là cho lại kết quả đúng. Tại sao ? Xét mảng kq được khai báo và khởi tạo trong tragiatri1(), đây là một mảng cục bộ (được tạo bên trong hàm) như sau này chúng ta sẽ thấy, các loại biến "tạm thời" này (và cả các tham đối) chỉ tồn tại trong quá trình hàm hoạt động. Khi hàm kết thúc các biến này sẽ mất đi. Do vậy tuy hàm đã trả lại địa

chỉ của kq trước khi nó kết thúc, thế nhưng sau khi hàm thực hiện xong, toàn bộ kq sẽ được xoá khỏi bộ nhớ và vì vậy con trỏ kết quả hàm đã trỏ đến vùng nhớ không còn các giá trị như kq đã có. Từ điều này việc sử dụng hàm trả lại con trỏ là phải hết sức cẩn thận. Muốn trả lại con trỏ cho hàm thì con trỏ này phải trỏ đến dãy dữ liệu nào sao cho nó không mất đi sau khi hàm kết thúc, hay nói khác hơn đó phải là những dãy dữ liệu được khởi tạo bên ngoài hàm hoặc có thể sử dụng theo phương pháp trong hàm tragiatri2(). Trong tragiatri2() một mảng kết quả 3 số cũng được tạo ra nhưng bằng cách xin cấp phát vùng nhớ. Vùng nhớ được cấp phát này sẽ vẫn còn tồn tại sau khi hàm kết thúc (nó chỉ bị xoá đi khi sử dụng toán tử delete). Do vậy hoạt động của tragiatri2() là chính xác.

Tóm lại, ví dụ trên cho thấy nếu muốn trả lại giá trị con trỏ thì vùng dữ liệu mà nó trỏ đến phải được cấp phát một cách tường minh (bằng toán tử new), chứ không để chương trình tự động cấp phát và tự động thu hồi. Ví dụ sau minh hoạ hàm cộng 2 vectơ và trả lại vectơ kết quả (thực chất là con trỏ trỏ đến vùng nhớ đặt kết quả)

```
int* congvt(int *x, int *y, int n)                  // n số phần tử của vectơ
{
      int* z = new int[n];                          // xin cấp phát bộ nhớ
      for (int i=0; i<n; i++) z[i] = x[i] + y[i];
      return c;
}

main()
{
      int i, n, a[10], b[10], c[10] ;
      cout << "n = " ; cin >> n;                    // nhập số phần tử
      for (i=0; i<n; i++) cin >> a[i] ;             // nhập vectơ a
      for (i=0; i<n; i++) cin >> b[i] ;             // nhập vectơ b
      c = congvt(a, b, n);
      for (i=0; i<n; i++) cout << c[i] ;            // in kết quả
}
```

Chú ý: a[i], b[i], c[i] còn được viết dưới dạng tương đương *(a+i), *(b+i), *(c+i).

2. Trong cách này, mảng cần trả lại được khai báo như một tham đối trong danh sách đối của hàm. Tham đối này là một con trỏ nên hiển nhiên khi truyền mảng đã khai báo sẵn (để chứa kết quả) từ ngoài vào cho hàm thì mảng sẽ thực sự nhận được nội dung kết quả (tức có thay đổi trước và sau khi gọi hàm

- xem mục truyền tham đối thực sự theo dẫn trỏ). Ở đây ta xét 2 ví dụ: bài toán cộng 2 vectơ trong ví dụ trước và nhân 2 ma trận.

Ví dụ 12 : Cộng 2 vectơ, vectơ kết quả trả lại trong tham đối của hàm. So với ví dụ trước giá trị trả lại là void (không trả lại giá trị) còn danh sách đối có thêm con trỏ z để chứa kết quả.

```
void congvt(int *x, int *y, int *z, int n)          // z lưu kết quả
{
    for (int i=0; i<n; i++) z[i] = x[i] + y[i];
}

main()
{
    int i, n, a[10], b[10], c[10] ;
    cout << "n = " ; cin >> n;                      // nhập số phần tử
    for (i=0; i<n; i++) cin >> a[i] ;              // nhập vectơ a
    for (i=0; i<n; i++) cin >> b[i] ;              // nhập vectơ b
    congvt(a, b, c, n);
    for (i=0; i<n; i++) cout << c[i] ;             // in kết quả
}
```

Ví dụ 13 : Nhân 2 ma trận kích thước m*n và n*p. Hai ma trận đầu vào và ma trận kết quả (kích thước m*p) đều được khai báo dưới dạng con trỏ và là đối của hàm nhanmt(). Nhắc lại, trong lời gọi hàm địa chỉ của 3 mảng cần được ép kiểu về (int*) để phù hợp với các con trỏ tham đối.

```
void nhanmt(int *x, int *y, int *z, int m, int n, int p)    // z lưu kết quả
{
    int i, j, k ;
    for (i=0; i<m; i++)
    for (j=0; j<p; j++)
    {
        *(z+i*p+j) = 0;                            // tức z[i][j] = 0
        for (k=0; k<n; k++)
        *(z+i*p+j) += *(x+i*n+k)**(y+k*p+j) ;      // tức z[i][j] += x[i][k]*y[k][j]
```

```
    }
}

main()
{
    int a[10][10], b[10][10], c[10][10] ;          // khai báo 3 mảng a, b, c
    int m, n, p ;                                  // kích thước các mảng
    cout << "m, n, p = " ; cin >> m >> n >> p ;    // nhập số phần tử
    for (i=0; i<m; i++)                            // nhập ma trận a
    for (j=0; j<n; j++)
        cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
    for (i=0; i<n; i++)                            // nhập ma trận b
    for (j=0; j<p; j++)
        cout << "b[" << i << "," << j << "] = " ; cin >> b[i][j] ;
    nhanmt((int*)a, (int*)b, (int*)c, m, n, p);    // gọi hàm
    for (i=0; i<m; i++)                            // in kết quả
    {
        for (j=0; j<p; j++) cout << c[i][j] ;
        cout << endl;
    }
}
```

d. Đối và giá trị trả lại là xâu kí tự

Giống các trường hợp đã xét với mảng 1 chiều, đối của các hàm xâu kí tự có thể khai báo dưới 2 dạng: mảng kí tự hoặc con trỏ kí tự. Giá trị trả lại luôn luôn là con trỏ kí tự. Ngoài ra hàm cũng có thể trả lại giá trị vào trong các đối con trỏ trong danh sách đối.

Ví dụ sau đây dùng để tách họ, tên của một xâu họ và tên. Ví dụ gồm 3 hàm. Hàm họ trả lại xâu họ (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng mảng. Hàm tên trả lại xâu tên (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng con trỏ kí tự. Thực chất đối họ và tên trong hai hàm họ, tên có thể được khai báo theo cùng cách thức, ở đây chương trình muốn minh hoạ các cách khai báo đối khác nhau (đã đề cập đến trong phần đối mảng 1 chiều). Hàm thứ ba cũng trả lại họ, tên nhưng cho vào trong danh sách tham đối, do vậy hàm không trả lại giá trị (void). Để đơn giản ta qui ước xâu

họ và tên không chứa các dấu cách đầu và cuối xâu, trong đó họ là dãy kí tự từ đầu cho đến khi gặp dấu cách đầu tiên và tên là dãy kí tự từ sau dấu cách cuối cùng đến kí tự cuối xâu.

```
char* ho(char hoten[])                          // hàm trả lại họ
{
    char* kq = new char[10];                    // cấp bộ nhớ để chứa họ
    int i=0;
    while (hoten[i] != '\40') i++;              // i dừng tại dấu cách đầu tiên
    strncpy(kq, hoten, i) ;                     // copy i kí tự của hoten vào kq
    return kq;
}

char* ten(char* hoten)                          // hàm trả lại tên
{
    char* kq = new char[10];                    // cấp bộ nhớ để chứa tên
    int i=strlen(hoten);
    while (hoten[i] != '\40') i--;              // i dừng tại dấu cách cuối cùng
    strncpy(kq, hoten+i+1, strlen(hoten)-i-1) ; // copy tên vào kq
    return kq;
}

void tachht(char* hoten, char* ho, char* ten)
{
    int i=0;
    while (hoten[i] != '\40') i++;              // i dừng tại dấu cách đầu tiên
    strncpy(ho, hoten, i) ;                     // copy i kí tự của hoten vào ho
    i=strlen(hoten);
    while (hoten[i] != '\40') i--;              // i dừng tại dấu cách cuối cùng
    strncpy(ten, hoten+i+1, strlen(hoten)-i-1) ; // copy tên vào ten
}

main()
{
```

```
    char ht[30], *h, *t ;                          // các biến họ tên, họ, tên
    cout << "Họ và tên = " ; cin.getline(ht,30) ; // nhập họ tên
    h = ho(ht); t = ten(ht);
    cout << "Họ = " << h << ", tên = " << t << endl;
    tachht(ht, h, t);
    cout << "Họ = " << h << ", tên = " << t << endl;
}
```

e. Đối là hằng con trỏ

Theo phần truyền đối cho hàm ta đã biết để thay đổi biến ngoài đối tương ứng phải được khai báo dưới dạng con trỏ. Tuy nhiên, trong nhiều trường hợp các biến ngoài không có nhu cầu thay đổi nhưng đối tương ứng với nó vẫn phải khai báo dưới dạng con trỏ (ví dụ đối là mảng hoặc xâu kí tự). Điều này có khả năng do nhầm lẫn, các biến ngoài này sẽ bị thay đổi ngoài ý muốn. Trong trường hợp như vậy để cẩn thận, các đối con trỏ nếu không muốn thay đổi (chỉ lấy giá trị) cần được khai báo như là một hằng con trỏ bằng cách thêm trước khai báo kiểu của chúng từ khoá const. Từ khoá này khẳng định biến tuy là con trỏ nhưng nó là một hằng không thay đổi được giá trị. Nếu trong thân hàm ta cố tình thay đổi chúng thì chương trình sẽ báo lỗi. Ví dụ đối hoten trong cả 3 hàm ở trên có thể được khai báo dạng const char* hoten.

Ví dụ 14 : Đối là hằng con trỏ. In hoa một xâu kí tự

```
void inhoa(const char* s)
{
    char *t;
    strcpy(t, s);
    cout << s << strupr(t);                    // không dùng được strupr(s)
}
main()
{
    char *s = "abcde" ;
    inhoa(s);                                  // abcdeABCDE
}
```

8. Con trỏ hàm

Một hàm (tập hợp các lệnh) cũng giống như dữ liệu: có tên gọi , có địa chỉ lưu

trong bộ nhớ và có thể truy nhập đến hàm thông qua tên gọi hoặc địa chỉ của nó. Để truy nhập (gọi hàm) thông qua địa chỉ chúng ta phải khai báo một con trỏ chứa địa chỉ này và sau đó gọi hàm bằng cách gọi tên con trỏ.

a. Khai báo

<kiểu giá trị> (*tên biến hàm)(d/s tham đối);

<kiểu giá trị> (*tên biến hàm)(d/s tham đối) = <tên hàm>;

Ta thấy cách khai báo con trỏ hàm cũng tương tự khai báo con trỏ biến (chỉ cần đặt dấu * trước tên), ngoài ra còn phải bao *tên hàm giữa cặp dấu ngoặc (). Ví dụ:

```
– float (*f)(int);          // khai báo con trỏ hàm có tên là f trỏ đến hàm
                            // có một tham đối kiểu int và cho giá trị kiểu float.
– void (*f)(float, int);    // con trỏ trỏ đến hàm với cặp đối (float, int).
```

hoặc phức tạp hơn:

```
– char* (*m[10])(int, char) // khai báo một mảng 10 con trỏ hàm trỏ đến
                            // các hàm có cặp tham đối (int, char), giá trị trả
                            // lại của các hàm này là xâu kí tự.
```

Chú ý: phân biệt giữa 2 khai báo: float (*f)(int) và float *f(int). Cách khai báo trước là khai báo con trỏ hàm có tên là f. Cách khai báo sau có thể viết lại thành float* f(int) là khai báo hàm f với giá trị trả lại là một con trỏ float.

b. Khởi tạo

Một con trỏ hàm cũng giống như các con trỏ, được phép khởi tạo trong khi khai báo hoặc gán với một địa chỉ hàm cụ thể sau khi khai báo. Cũng giống như kiểu dữ liệu mảng, tên hàm chính là một hằng địa chỉ trỏ đến bản thân nó. Do vậy cú pháp của khởi tạo cũng như phép gán là như sau:

biến con trỏ hàm = tên hàm;

trong đó f và tên hàm được trỏ phải giống nhau về kiểu trả lại và danh sách đối. Nói cách khác với mục đích sử dụng con trỏ f trỏ đến hàm (lớp hàm) nào đó thì f phải được khai báo với kiểu trả lại và danh sách đối giống như hàm đó. Ví dụ:

```
float luythua(float, int);  // khai báo hàm luỹ thừa
float (*f)(float, int);     // khai báo con trỏ f tương thích với hàm luythua
f = luythua;                // cho f trỏ đến hàm luỹ thừa
```

c. Sử dụng con trỏ hàm

Để sử dụng con trỏ hàm ta phải gán nó với tên hàm cụ thể và sau đó bất kỳ nơi nào được phép xuất hiện tên hàm thì ta đều có thể thay nó bằng tên con trỏ. Ví dụ như các thao tác gọi hàm, đưa hàm vào làm tham đối hình thức cho một hàm khác … Sau đây là các ví dụ minh hoạ.

Ví dụ 15 : Dùng tên con trỏ để gọi hàm

```
float bphuong(float x)                    // hàm trả lại x2
{
      return x*x;
}
void main()
{
      float (*f)(float);
      f = bphuong;
      cout << "Bình phương của 3.5 là " << f(3.5) ;
}
```

Ví dụ 16 : Dùng hàm làm tham đối. Tham đối của hàm ngoài các kiểu dữ liệu đã biết còn có thể là một hàm. Điều này có tác dụng rất lớn trong các bài toán tính toán trên những đối tượng là hàm toán học như tìm nghiệm, tính tích phân của hàm trên một đoạn ... Hàm đóng vai trò tham đối sẽ được khai báo dưới dạng con trỏ hàm. Ví dụ sau đây trình bày hàm tìm nghiệm xấp xỉ của một hàm liên tục và đổi dấu trên đoạn [a, b]. Để hàm tìm nghiệm này sử dụng được trên nhiều hàm toán học khác nhau, trong hàm sẽ chứa một biến con trỏ hàm và hai cận a, b, cụ thể bằng khai báo **float timnghiem(float (*f)(float), float a, float b).** Trong lời gọi hàm f sẽ được thay thế bằng tên hàm cụ thể cần tìm nghiệm.

```
#define EPS 1.0e-6
float timnghiem(float (*f)(float), float a, float b);
float emu(float);
float loga(float);
void main()
{
      clrscr();
      cout << "Nghiệm của e mũ x - 2 trên đoạn [0,1] = ";
      cout << timnghiem(emu,0,1));
```

```
        cout << "Nghiệm của loga(x) - 1 trên đoạn [2,3] = ";
        cout << timnghiem(loga,2,3));
        getch();
}

float timnghiem(float (*f)(float), float a, float b)
{
        float c = (a+b)/2;
        while (fabs(a-b)>EPS && f(c)!=0)
        {
                if (f(a)*f(c)>0) a = c ; else b = c;
                c = (a+b)/2;
        }
        return c;
}

float emux(float x)     { return (exp(x)-2); }
float logx(float x)     { return (log(x)-1); }
```

d. Mảng con trỏ hàm

Tương tự như biến bình thường các con trỏ hàm giống nhau có thể được gộp lại vào trong một mảng, trong khai báo ta chỉ cần thêm [n] vào sau tên mảng với n là số lượng tối đa các con trỏ. Ví dụ sau minh hoạ cách sử dụng này. Trong ví dụ chúng ta xây dựng 4 hàm cộng, trừ, nhân, chia 2 số thực. Các hàm này giống nhau về kiểu, số lượng đối, … Chúng ta có thể sử dụng 4 con trỏ hàm riêng biệt để trỏ đến các hàm này hoặc cũng có thể dùng mảng 4 con trỏ để trỏ đến các hàm này. Chương trình sẽ in ra kết quả cộng, trừ, nhân, chia của 2 số nhập vào từ bàn phím.

Ví dụ 17 :

```
void cong(int a, int b){ cout << a << " + " << b << " = " << a+b ; }
void tru(int a, int b)   { cout << a << " - " << b << " = " << a-b ; }
void nhan(int a, int b){ cout << a << " x " << b << " = " << a*b ; }
void chia(int a, int b)  { cout << a << ": " << b << " = " << a/b ; }
main()
{
```

```
    clrscr();
    void (*f[4])(int, int) = {cong, tru, nhan, chia}; // khai báo, khởi tạo 4 con trỏ
    int m, n;
    cout "Nhập m, n " ; cin >> m >> n ;
    for (int i=0; i<4; i++) f[i](m,n);
    getch();
}
```

III. ĐỆ QUI

1. Khái niệm đệ qui

Một hàm gọi đến hàm khác là bình thường, nhưng nếu hàm lại gọi đến chính nó thì ta gọi hàm là đệ qui. Khi thực hiện một hàm đệ qui, hàm sẽ phải chạy rất nhiều lần, trong mỗi lần chạy chương trình sẽ tạo nên một tập biến cục bộ mới trên ngăn xếp (các đối, các biến riêng khai báo trong hàm) độc lập với lần chạy trước đó, từ đó dễ gây tràn ngăn xếp. Vì vậy đối với những bài toán có thể giải được bằng phương pháp lặp thì không nên dùng đệ qui.

Để minh hoạ ta hãy xét hàm tính n giai thừa. Để tính n! ta có thể dùng phương pháp lặp như sau:

```
main()
{
    int n; doule kq = 1;
    cout << "n = " ; cin >> n;
    for (int i=1; i<=n; i++) kq *= i;
    cout << n << "! = " << kq;
}
```

Mặt khác, n! giai thừa cũng được tính thông qua (n-1)! bởi công thức truy hồi

```
n! = 1          nếu n = 0
n! = (n-1)!n    nếu n > 0
```

do đó ta có thể xây dựng hàm đệ qui tính n! như sau:

```
double gt(int n)
{
```

```
        if (n==0) return 1;
        else return gt(n-1)*n;
}

main()
{
        int n;
        cout << "n = " ; cin >> n;
        cout << gt(n);
}
```

Trong hàm main() giả sử ta nhập 3 cho n, khi đó để thực hiện câu lệnh cout << gt(3) để in 3! đầu tiên chương trình sẽ gọi chạy hàm gt(3). Do 3 ≠ 0 nên hàm gt(3) sẽ trả lại giá trị gt(2)*3, tức lại gọi hàm gt với tham đối thực sự ở bước này là n = 2. Tương tự gt(2) = gt(1)*2 và gt(1) = gt(0)*1. Khi thực hiện gt(0) ta có đối n = 0 nên hàm trả lại giá trị 1, từ đó gt(1) = 1*1 = 1 và suy ngược trở lại ta có gt(2) = gt(1)*2 = 1*2 = 2, gt(3) = gt(2)*3 = 2*3 = 6, chương trình in ra kết quả 6.

Từ ví dụ trên ta thấy hàm đệ qui có đặc điểm:

- Chương trình viết rất gọn,
- Việc thực hiện gọi đi gọi lại hàm rất nhiều lần phụ thuộc vào độ lớn của đầu vào. Chẳng hạn trong ví dụ trên hàm được gọi n lần, mỗi lần như vậy chương trình sẽ mất thời gian để lưu giữ các thông tin của hàm gọi trước khi chuyển điều khiển đến thực hiện hàm được gọi. Mặt khác các thông tin này được lưu trữ nhiều lần trong ngăn xếp sẽ dẫn đến tràn ngăn xếp nếu n lớn.

Tuy nhiên, đệ qui là cách viết rất gọn, dễ viết và đọc chương trình, mặt khác có nhiều bài toán hầu như tìm một thuật toán lặp cho nó là rất khó trong khi viết theo thuật toán đệ qui thì lại rất dễ dàng.

2. Lớp các bài toán giải được bằng đệ qui

Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:

- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp *suy biến* hay *cơ sở*, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó,
- Đối với trường hợp *tổng quát*, bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham đối khác có kích thước nhỏ hơn tham đối ban đầu. Và sau một số bước hữu hạn biến đổi cùng dạng, bài toán đưa được về trường hợp

suy biến.

Như vậy trong trường hợp tính n! nếu n = 0 hàm cho ngay giá trị 1 mà không cần phải gọi lại chính nó, đây chính là trường hợp suy biến. Trường hợp n > 0 hàm sẽ gọi lại chính nó nhưng với n giảm 1 đơn vị. Việc gọi này được lặp lại cho đến khi n = 0.

Một lớp rất rộng của bài toán dạng này là các bài toán có thể định nghĩa được dưới dạng đệ qui như các bài toán lặp với số bước hữu hạn biết trước, các bài toán UCLN, tháp Hà Nội, ...

3. Cấu trúc chung của hàm đệ qui

Dạng thức chung của một chương trình đệ qui thường như sau:

```
if (trường hợp suy biến)
{
    trình bày cách giải          // giả định đã có cách giải
}
else                             // trường hợp tổng quát
{
    gọi lại hàm với tham đối "bé" hơn
}
```

4. Các ví dụ

Ví dụ 1 : Tìm UCLN của 2 số a, b. Bài toán có thể được định nghĩa dưới dạng đệ qui như sau:

- nếu a = b thì UCLN = a
- nếu a > b thì UCLN(a, b) = UCLN(a-b, b)
- nếu a < b thì UCLN(a, b) = UCLN(a, b-a)

Từ đó ta có chương trình đệ qui để tính UCLN của a và b như sau.

```
int UCLN(int a, int b)                  // qui uoc a, b > 0
{
    if (a < b) UCLN(a, b-a);
    if (a == b) return a;
    if (a > b) UCLN(a-b, b);
}
```

Ví dụ 2 : Tính số hạng thứ n của dãy Fibonaci là dãy f(n) được định nghĩa:

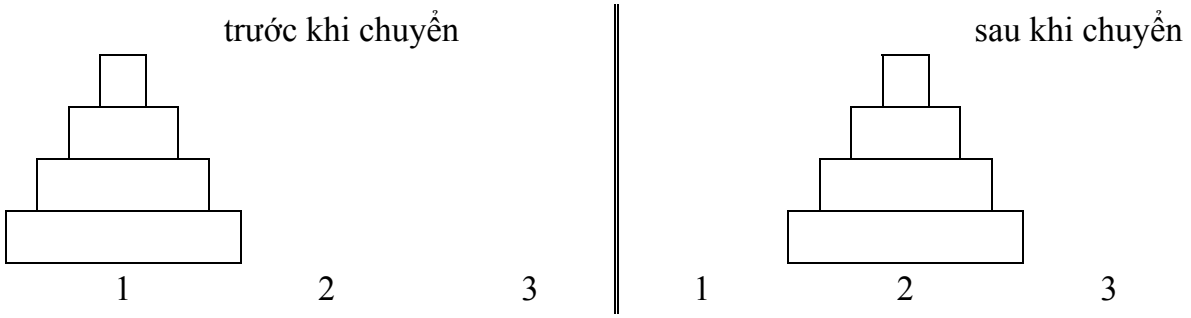
- f(0) = f(1) = 1
- f(n) = f(n-1) + f(n-2) với $\forall n \geq 2$.

```
long Fib(int n)
{
  long kq;
  if (n==0 || n==1) kq = 1; else kq = Fib(n-1) + Fib(n-2);
  return kq;
}
```

Ví dụ 3 : Chuyển tháp là bài toán cổ nổi tiếng, nội dung như sau: Cho một tháp n tầng, đang xếp tại vị trí 1. Yêu cầu bài toán là hãy chuyển toàn bộ tháp sang vị trí 2 (cho phép sử dụng vị trí trung gian 3) theo các điều kiện sau đây

- mỗi lần chỉ được chuyển một tầng trên cùng của tháp,
- tại bất kỳ thời điểm tại cả 3 vị trí các tầng tháp lớn hơn phải nằm dưới các tầng tháp nhỏ hơn.

Bài toán chuyển tháp được minh hoạ bởi hình vẽ dưới đây.



Bài toán có thể được đặt ra tổng quát hơn như sau: chuyển tháp từ vị trí di đến vị trí den, trong đó di, den là các tham số có thể lấy giá trị là 1, 2, 3 thể hiện cho 3 vị trí. Đối với 2 vị trí di và den, dễ thấy vị trí trung gian (vị trí còn lại) sẽ là vị trí 6-di-den (vì di+den+tg = 1+2+3 = 6). Từ đó để chuyển tháp từ vị trí di đến vị trí den, ta có thể xây dựng một cách chuyển đệ qui như sau:

- chuyển 1 tầng từ di sang tg,
- chuyển n-1 tầng còn lại từ di sang den,
- chuyển trả tầng tại vị trí tg về lại vị trí den

hiển nhiên nếu số tầng là 1 thì ta chỉ phải thực hiện một phép chuyển từ di sang den.

Mỗi lần chuyển 1 tầng từ vị trí i đến j ta kí hiệu i → j. Chương trình sẽ nhập vào input là số tầng và in ra các bước chuyển theo kí hiệu trên.

Từ đó ta có thể xây dựng hàm đệ qui sau đây ;

```
void chuyen(int n, int di, int den)              // n: số tầng, di, den: vị trí đi, đến
{
    if (n==1) cout << di << " → " << den << endl;
    else {
        cout << di << "→" << 6-di-den << endl; // 1 tầng từ di qua trung gian
        chuyen(n-1, di, den) ;                    // n-1 tầng từ di qua den
        cout << 6-di-den << "→" den << endl;      // 1 tầng từ tg về lại den
    }
}

main()
{
    int sotang ;
    cout << "Số tầng = " ; cin >> sotang;
    chuyen(sotang, 1, 2);
}
```

Ví dụ nếu số tầng bằng 3 thì chương trình in ra kết quả là dãy các phép chuyển sau đây:

$$1 \to 2\ ,\ 1 \to 3\ ,\ 2 \to 3\ ,\ 1 \to 2\ ,\ 3 \to 1\ ,\ 3 \to 2\ ,\ 1 \to 2.$$

có thể tính được số lần chuyển là $2^n - 1$ với n là số tầng.

IV. TỔ CHỨC CHƯƠNG TRÌNH

1. Các loại biến và phạm vi

a. Biến cục bộ

Là các biến được khai báo trong thân của hàm và chỉ có tác dụng trong hàm này, kể cả các biến khai báo trong hàm main() cũng chỉ có tác dụng riêng trong hàm main(). Từ đó, tên biến trong các hàm là được phép trùng nhau. Các biến của hàm nào sẽ chỉ

tồn tại trong thời gian hàm đó hoạt động. Khi bắt đầu hoạt động các biến này được tự động sinh ra và đến khi hàm kết thúc các biến này sẽ mất đi. Tóm lại, một hàm được xem như một đơn vị độc lập, khép kín.

Tham đối của các hàm cũng được xem như biến cục bộ.

Ví dụ 1 : Dưới đây ta nhắc lại một chương trình nhỏ gồm 3 hàm: luỹ thừa, xoá màn hình và main(). Mục đích để minh hoạ biến cục bộ.

```
float luythua(float x, int n)                    // hàm trả giá trị x^n
{
      int i ;
      float kq = 1;
      for (i=1; i<=n; i++) kq *= x;
      return kq;
}

void xmh(int n)                                  // xoá màn hình n lần
{
      int i;
      for (i=1; i<=n; i++) clrscr();
}

main()
{
      float x; int n;
      cout << "Nhập x và n: "; cin >> x >> n;
      xmh(5);                                    // xoá màn hình 5 lần
      cout << luythua(x, n);                     // in x^n
}
```

Qua ví dụ trên ta thấy các biến i, đối n được khai báo trong hai hàm: luythua() và xmh(). kq được khai báo trong luythua và main(), ngoài ra các biến x và n trùng với đối của hàm luythua(). Tuy nhiên, tất cả khai báo trên đều hợp lệ và đều được xem như khác nhau. Có thể giải thích như sau:

– Tất cả các biến trên đều cục bộ trong hàm nó được khai báo.

- x và n trong main() có thời gian hoạt động dài nhất: trong suốt quá trình chạy chương trình. Chúng chỉ mất đi khi chương trình chấm dứt. Đối x và n trong luythua() chỉ tạm thời được tạo ra khi hàm luythua() được gọi đến và độc lập với x, n trong main(), nói cách khác tại thời điểm đó trong bộ nhớ có hai biến x và hai biến n. Khi hàm luythua chay xong biến x và n của nó tự động biến mất.
- Tương tự 2 đối n, 2 biến i trong luythua() và xoá màn hình cũng độc lập với nhau, chúng chỉ được tạo và tồn tại trong thời gian hàm của chúng được gọi và hoạt động.

b. Biến ngoài

Là các biến được khai báo bên ngoài của tất cả các hàm. Vị trí khai báo của chúng có thể từ đầu văn bản chương trình hoặc tại một một vị trí bất kỳ nào đó giữa văn bản chương trình. Thời gian tồn tại của chúng là từ lúc chương trình bắt đầu chạy đến khi kết thúc chương trình giống như các biến trong hàm main(). Tuy nhiên về phạm vi tác dụng của chúng là bắt đầu từ điểm khai báo chúng đến hết chương trình, tức tất cả các hàm khai báo sau này đều có thể sử dụng và thay đổi giá trị của chúng. Như vậy các biến ngoài được khai báo từ đầu chương trình sẽ có tác dụng lên toàn bộ chương trình. Tất cả các hàm đều sử dụng được các biến này nếu trong hàm đó không có biến khai báo trùng tên. Một hàm nếu có biến trùng tên với biến ngoài thì biến ngoài bị che đối với hàm này. Có nghĩa nếu i được khai báo như một biến ngoài và ngoài ra trong một hàm nào đó cũng có biến i thì như vậy có 2 biến i độc lập với nhau và khi hàm truy nhập đến i thì có nghĩa là i của hàm chứ không phải i của biến ngoài.

Dưới đây là ví dụ minh hoạ cho các giải thích trên.

Ví dụ 2 : Chúng ta xét lại các hàm luythua() và xmh(). Chú ý rằng trong cả hai hàm này đều có biến i, vì vậy chúng ta có thể khai báo i như một biến ngoài (để dùng chung cho luythua() và xmh()), ngoài ra x, n cũng có thể được khai báo như biến ngoài. Cụ thể:

```
#include <iostream.h>
#include <iomanip.h>
float x; int n; int i ;
float luythua(float x, int n)
{
      float kq = 1;
      for (i=1; i<=n; i++) kq *= x;
}
```

```
void xmh()
{
        for (i=1; i<=n; i++) clrscr();
}

main()
{
        cout << "Nhập x và n: "; cin >> x >> n;
        xmh(5);                                    // xoá màn hình 5 lần
        cout << luythua(x, n);                     // in x^n
}
```

Trong ví dụ này ta thấy các biến x, n, i đều là các biến ngoài. Khi ta muốn sử dụng biến ngoài ví dụ i, thì biến i sẽ không được khai báo trong hàm sử dụng nó. Chẳng hạn, luythua() và xmh() đều sử dụng i cho vòng lặp for của mình và nó không được khai báo lại trong 2 hàm này. Các đối x và n trong luythua() là độc lập với biến ngoài x và n. Trong luythua() khi sử dụng đến x và n (ví dụ câu lệnh kq *= x) thì đây là x của hàm chứ không phải biến ngoài, trong khi trong main() không có khai báo về x và n nên ví dụ câu lệnh cout << luythua(x, n); là sử dụng x, n của biến ngoài.

Nói chung trong 2 ví dụ trên chương trình đều chạy tốt và như nhau. Tuy nhiên, việc khai báo khác nhau như vậy có ảnh hưởng hoặc gây nhầm lẫn gì cho người lập trình ? Liệu chúng ta có nên tự đặt ra một nguyên tắc nào đó trong khai báo biến ngoài và biến cục bộ để tránh những nhầm lẫn có thể xảy ra. Chúng ta hãy xét tiếp cũng ví dụ trên nhưng thay đổi một số khai báo và tính 2^3 (có thể bỏ bớt biến n) như sau:

```
#include <iostream.h>
#include <iomanip.h>
float x; int i ;                                   // không dùng n
float luythua(float x, int n)
{
        float kq = 1;
        for (i=1; i<=n; i++) kq *= x;
}

void xmh()
{
```

```
        for (i=1; i<=n; i++) clrscr();
}

main()
{
        x = 2;
        i = 3;
        xmh(5);                                  // xoá màn hình 5 lần
        cout << luythua(x, i);                   // in x^i, kết quả x = 2^3 = 8 ?
}
```

Nhìn vào hàm main() ta thấy giá trị 2^3 được tính bằng cách đặt x = 2, i = 3 và gọi hàm luythua(x,i). Kết quả ta mong muốn sẽ là giá trị 8 hiện ra màn hình, tuy nhiên không đúng như vậy. Trước khi in kết quả này ra màn hình hàm xmh() đã được gọi đến để xoá màn hình. Hàm này sử dụng một biến ngoài i để làm biến đếm cho mình trong vòng lặp for và sau khi ra khỏi for (cũng là kết thúc xmh()) i nhận giá trị 6. Biến i ngoài này lại được sử dụng trong lời gọi luythua(x,i) của hàm main(), tức tại thời điểm này x = 2 và i = 6, kết quả in ra màn hình sẽ là $2^6 = 64$ thay vì 8 như mong muốn.

Tóm lại "điểm yếu" dẫn đến sai sót của chương trình trên là ở chỗ lập trình viên đã "tranh thủ" sử dụng biến i cho 2 hàm xmh() và main() (bằng cách khai báo nó như biến ngoài) nhưng lại với mục đích khác nhau. Do vậy sau khi chạy xong hàm xmh() i bị thay đổi khác với giá trị i được khởi tạo lúc ban đầu. Để khắc phục lỗi trong chương trình trên ta cần khai báo lại biến i: hoặc trong main() khai báo thêm i (nó sẽ che biến i ngoài), hoặc trong cả hai xmh() và main() đều có biến i (cục bộ trong từng hàm).

Từ đó, ta nên đề ra một vài nguyên tắc lập trình sao cho nó có thể tránh được những lỗi không đáng có như vậy:

- **nếu một biến chỉ sử dụng vì mục đích riêng của một hàm thì nên khai báo biến đó như biến cục bộ trong hàm**. Ví dụ các biến đếm của vòng lặp, thông thường chúng chỉ được sử dụng thậm chí chỉ riêng trong vòng lặp chứ cũng chưa phải cho toàn bộ cả hàm, vì vậy không nên khai báo chúng như biến ngoài. Những biến cục bộ này sau khi hàm kết thúc chúng cũng sẽ kết thúc, không gây ảnh hưởng đến bất kỳ hàm nào khác. Một đặc điểm có lợi nữa cho khai báo cục bộ là chúng tạo cho hàm tính cách hoàn chỉnh, độc lập với mọi hàm khác, chương trình khác. Ví dụ hàm xmh() có thể mang qua chạy ở chương trình khác mà không phải sửa chữa gì nếu i đã được khai báo bên trong hàm. Trong khi ở ví dụ này hàm xmh() vẫn hoạt động được nhưng trong chương trình khác nếu không có i như một biến ngoài (để xmh() sử dụng) thì hàm sẽ gây lỗi.

- với các biến mang tính chất **sử dụng chung** rõ nét (đặc biệt với những biến kích thước lớn) mà nhiều hàm cùng sử dụng chúng với mục đích giống nhau thì nên khai báo chúng như biến ngoài. Điều này tiết kiệm được thời gian cho người lập trình vì không phải khai báo chúng nhiều lần trong nhiều hàm, tiết kiệm bộ nhớ vì không phải tạo chúng tạm thời mỗi khi chạy các hàm, tiết kiệm được thời gian chạy chương trình vì không phải tổ chức bộ nhớ để lưu trữ và giải phóng chúng. Ví dụ trong chương trình quản lý sinh viên (chương 6), biến sinh viên được dùng chung và thống nhất trong hầu hết các hàm (xem, xoá, sửa, bổ sung, thống kê …) nên có thể khai báo chúng như biến ngoài, điều này cũng tăng tính thống nhất của chương trình (mọi biến sinh viên là như nhau cho mọi hàm con của chương trình).

Tóm lại, nguyên tắc tổng quát nhất là cố gắng tạo hàm một cách độc lập, khép kín, không chịu ảnh hưởng của các hàm khác và không gây ảnh hưởng đến hoạt động của các hàm khác đến mức có thể.

2. Biến với mục đích đặc biệt

a. Biến hằng và từ khoá const

Để sử dụng hằng có thể khai báo thêm từ khoá const trước khai báo biến. Phạm vi và miền tác dụng cũng như biến, có nghĩa biến hằng cũng có thể ở dạng cục bộ hoặc toàn thể. Biến hằng luôn luôn được khởi tạo trước.

Có thể khai báo từ khoá const trước các tham đối hình thức để không cho phép thay đổi giá trị của các biến ngoài (đặc biệt đối với với mảng và xâu kí tự, vì bản thân các biến này được xem như con trỏ do đó hàm có thể thay đổi được giá trị của các biến ngoài truyền cho hàm này).

Ví dụ sau thể hiện hằng cũng có thể được khai báo ở các phạm vi khác nhau.

```
const int MAX = 30;                          // toàn thể
void vidu(const int *p)                      // cục bộ
{
      const MAX = 10;                        // cục bộ
…
}
void main()
{
      const MAX = 5;                         // cục bộ
…
```

```
}
…
```

Trong Turbo C, BorlandC và các chương trình dịch khác có nhiều hằng số khai báo sẵn trong tệp values.h như MAXINT, M_PI hoặc các hằng đồ hoạ trong graphics.h như WHITE, RED, …

b. Biến tĩnh và từ khoá static

Được khai báo bằng từ khoá static. Là biến cục bộ nhưng vẫn giữ giá trị sau khi ra khỏi hàm. Phạm vi tác dụng như biến cục bộ, nghĩa là nó chỉ được sử dụng trong hàm khai báo nó. Tuy nhiên thời gian tác dụng được xem như biến toàn thể, tức sau khi hàm thực hiện xong biến vẫn còn tồn tại và vẫn lưu lại giá trị sau khi ra khỏi hàm. Giá trị này này được tiếp tục sử dụng khi hàm được gọi lại, tức biến static chỉ được khởi đầu một lần trong lần chạy hàm đầu tiên. Nếu không khởi tạo, C++ tự động gán giá trị 0 (ngầm định = 0). Ví dụ:

```
int i = 1;
void bp()
{
        static int lanthu = 0;
        lanthu++;
        i = 2 * i;
        cout << "Hàm chạy lần thứ " << lanthu << ", i = " << i ;
…
}

main()
{
        ham();                          // Hàm chạy lần thứ 1, i = 1
        ham();                          // Hàm chạy lần thứ 2, i = 2
        ham();                          // Hàm chạy lần thứ 3, i = 4
…
}
```

c. Biến thanh ghi và từ khoá register

Để tăng tốc độ tính toán C++ cho phép một số biến được đặt trực tiếp vào thanh ghi thay vì ở bộ nhớ. Khai báo bằng từ khoá **register** đứng trước khai báo biến. Tuy

nhiên khai báo này chỉ có tác dụng đối với các biến có kích thước nhỏ như biến char, int.

Ví dụ: register char c; register int dem;

d. Biến ngoài và từ khoá extern

Như đã biết một chương trình có thể được đặt trên nhiều file văn bản khác nhau. Một biến không thể được khai báo nhiều lần với cùng phạm vi hoạt động. Do vậy nếu một hàm sử dụng biến được khai báo trong file văn bản khác thì biến này phải được khai báo với từ khoá extern. Từ khoá này cho phép chương trình dịch tìm và liên kết biến này từ bên ngoài file đang chứa biến. Chúng ta hãy xét ví dụ gây lỗi sau đây và tìm phương án khắc phục chúng.

```
void in();
void main()
{
        int i = 1;
        in();
}

void in()
{
        cout << i ;
}
```

- Lỗi (cú pháp) vì i là biến cục bộ trong main(), trong in() không nhận biết i, nếu trong hoặc trước in() khai báo thêm i thì lỗi ngữ nghĩa (tức chương trình in giá trị i khác không theo ý muốn của lập trình viên).
- Giả thiết khai báo lại như sau:

```
void in();
void main() { ... }          // Bỏ khai báo i trong main()
int i;                       // Đưa khai báo i ra trước in() và sau main()
void in() { ... }
```

cách khai báo này cũng gây lỗi vì main() không nhận biết i. Cuối cùng để main() có thể nhận biết i thì i phải được khai báo dưới dạng biến extern. Thông thường trong trường hợp này cách khắc phục hay nhất là khai báo trước main() để bỏ các extern (không cần thiết).

Giả thiết 2 chương trình trên nằm trong 2 tệp khác nhau. Để liên kết (link) biến i giữa 2 chương trình cần định nghĩa tổng thể i trong một và khai báo extern trong chương trình kia.

```
/* program1.cpp*/
void in();
int i;
void main()
{
        i = 1;
        in();
}
/* program2.cpp */
void in()
{
        extern i;
        cout << i ;
}
```

Hàm in() nằm trong tệp văn bản program2.cpp, được dùng để in giá trị của biến i khai báo trong program1.cpp, tạm gọi là tệp gốc (hai tệp này khi dịch sẽ được liên kết với nhau). Từ đó trong tệp gốc, i phải được khai báo là biến ngoài, và bất kỳ hàm ở tệp khác muốn sử dụng biến i này đều phải có câu lệnh khai báo extern int i (nếu không có từ khoá extern thì biến i lại được xem là biến cục bộ, khác với biến i trong tệp gốc).

Để liên kết các tệp nguồn có thể tạo một dự án (project) thông qua menu PROJECT (Alt-P). Các phím nóng cho phép mở dự án, thêm bớt tệp vào danh sách tệp của dự án … được hướng dẫn ở dòng cuối của cửa sổ dự án.

3. Các chỉ thị tiền xử lý

Như đã biết trước khi chạy chương trình (bắt đầu từ văn bản chương trình tức chương trình nguồn) C++ sẽ dịch chương trình ra tệp mã máy còn gọi là chương trình đích. Thao tác dịch chương trình nói chung gồm có 2 phần: xử lý sơ bộ chương trình và dịch. Phần xử lý sơ bộ được gọi là tiền xử lý, trong đó có các công việc liên quan đến các chỉ thị được đặt ở đầu tệp chương trình nguồn như #include, #define …

a. Chỉ thị bao hàm tệp #include

Cho phép ghép nội dung các tệp đã có khác vào chương trình trước khi dịch. Các tệp cần ghép thêm vào chương trình thường là các tệp chứa khai báo nguyên mẫu của các hằng, biến, hàm … có sẵn trong C hoặc các hàm do lập trình viên tự viết. Có hai dạng viết chỉ thị này.

1: #include <tệp>

2: #include "đường dẫn\tệp"

Dạng khai báo 1 cho phép C++ ngầm định tìm tệp tại thư mục định sẵn (khai báo thông qua menu Options\Directories) thường là thư mục TC\INCLUDE và tệp là các tệp nguyên mẫu của thư viện C++.

Dạng khai báo 2 cho phép tìm tệp theo đường dẫn, nếu không có đường dẫn sẽ tìm trong thư mục hiện tại. Tệp thường là các tệp (thư viện) được tạo bởi lập trình viên và được đặt trong cùng thư mục chứa chương trình. Cú pháp này cho phép lập trình viên chia một chương trình thành nhiều môđun đặt trên một số tệp khác nhau để dễ quản lý. Nó đặc biệt hữu ích khi lập trình viên muốn tạo các thư viện riêng cho mình.

b. Chỉ thị macro #define

#define tên_macro xaukitu

Trước khi dịch bộ tiền xử lý sẽ tìm trong chương trình và thay thế bất kỳ vị trí xuất hiện nào của tên_macro bởi xâu kí tự. Ta thường sử dụng macro để định nghĩa các hằng hoặc thay cụm từ này bằng cụm từ khác dễ nhớ hơn, ví dụ:

```
#define then                    // thay then bằng dấu cách
#define begin {                 // thay begin bằng dấu {
#define end }                   // thay end bằng dấu }
#define MAX 100                 // thay MAX bằng 100
#define TRUE 1                  // thay TRUE bằng 1
```

từ đó trong chương trình ta có thể viết những đoạn lệnh như:

```
if (i < MAX) then
begin
    Ok = TRUE;
    cout << i ;
end
```

trước khi dịch bộ tiền xử lý sẽ chuyển đoạn chương trình trên thành

```
if (i < 100)
{
```

```
        Ok = 1;
        cout << i ;
    }
```

theo đúng cú pháp của C++ và rồi mới tiến hành dịch.

Ngoài việc chỉ thị #define cho phép thay tên_macro bởi một xâu kí tự bất kỳ, nó còn cũng được phép viết dưới dạng có đối. Ví dụ, để tìm số lớn nhất của 2 số, thay vì ta phải viết nhiều hàm max (mỗi hàm ứng với một kiểu số khác nhau), bây giờ ta chỉ cần thay chúng bởi một macro có đối đơn giản như sau:

#define max(A,B) ((A) > (B) ? (A): (B))

khi đó trong chương trình nếu có dòng x = max(a, b) thì nó sẽ được thay bởi: x = ((a) > (b) ? (a): (b))

Chú ý:

- Tên macro phải được viết liền với dấu ngoặc của danh sách đối. Ví dụ không viết max (A,B).
- #define bp(x) (x*x) viết sai vì bp(5) đúng nhưng bp(a+b) sẽ thành (a+b*a+b) (tức a+b+ab).
- Cũng tương tự viết #define max(A,B) (A > B ? A: B) là sai (?) vì vậy luôn luôn bao các đối bởi dấu ngoặc.
- #define bp(x) ((x)*(x)) viết đúng nhưng nếu giả sử lập trình viên muốn tính bình phương của 2 bằng đoạn lệnh sau:

```
int i = 1;
cout << bp(++i);                    // 6
```

thì kết quả in ra sẽ là 6 thay vì kết quả đúng là 4. Lí do là ở chỗ chương trình dịch sẽ thay bp(++i) bởi ((++i)*(++i)), và với i = 1 chương trình sẽ thực hiện như 2*3 = 6. Do vậy cần cẩn thận khi sử dụng các phép toán tự tăng giảm trong các macro có đối. Nói chung, nên hạn chế việc sử dụng các macro phức tạp, vì nó có thể gây nên những hiệu ứng phụ khó kiểm soát.

c. Các chỉ thị biên dịch có điều kiện #if, #ifdef, #ifndef

- Chỉ thị:

  #if dãy lệnh … #endif

  #if dãy lệnh … #else dãy lệnh … #endif,

Các chỉ thị này giống như câu lệnh if, mục đích của nó là báo cho chương trình dịch biết đoạn lệnh giữa #if (điều kiện) và #endif chỉ được dịch nếu điều kiện đúng. Ví

dụ:

```
const int M = 1;
void main() {
int i = 5;
#if M==1
        cout << i ;
#endif
}
```

hoặc:

```
const int M = 10;
void main() {
int i = 5;
#if M > 8
        cout << i+i ;
#else
        cout << i*i ;
#endif
}
```

- **Chỉ thị #ifdef và #ifndef**

Chỉ thị này báo cho chương trình dịch biết đoạn lệnh có được dịch hay không khi một tên gọi đã được định nghĩa hay chưa. #ifdef được hiểu là nếu tên đã được định nghĩa thì dịch, còn #ifndef được hiểu là nếu tên chưa được định nghĩa thì dịch. Để định nghĩa một tên gọi ta dùng chỉ thị #define tên.

Chỉ thị này đặc biệt có ích khi chèn các tệp thư viện vào để sử dụng. Một tệp thư viện có thể được chèn nhiều lần trong văn bản do vậy nó có thể sẽ được dịch nhiều lần, điều này sẽ gây ra lỗi vì các biến được khai báo nhiều lần. Để tránh việc này, ta cần sử dụng chỉ thị trên như ví dụ minh hoạ sau: Giả sử ta đã viết sẵn 2 tệp thư viện là mylib.h và mathfunc.h, trong đó mylib.h chứa hàm max(a,b) tìm số lớn nhất giữa 2 số, mathfunc.h chứa hàm max(a,b,c) tìm số lớn nhất giữa 3 số thông qua sử dụng hàm max(a,b). Do vậy mathfunc.h phải có chỉ thị #include mylib.h để sử dụng được hàm max(a,b).

– Thư viện 1. tên tệp: MYLIB.H

```
int max(int a, int b)
```

```
{
  return (a>b? a: b);
}
```

– Thư viện 2. tên tệp: MATHFUNC.H

```
#include "mylib.h"
int max(int a, int b)
{
  return (a>b? a: b);
}
```

Hàm main của chúng ta nhập 3 số, in ra max của từng cặp số và max của cả 3 số. Chương trình cần phải sử dụng cả 2 thư viện.

```
#include "mylib.h"
#include "mathfunc.h"
main()
{
      int a, b, c;
      cout << "a, b, c = " ; cin >> a >> b >> c;
      cout << max(a,b) << max(b,c) << max(a,c) << max(a,b,c) ;
}
```

Trước khi dịch chương trình, bộ tiền xử lý sẽ chèn các thư viện vào trong tệp chính (chứa main()) trong đó mylib.h được chèn vào 2 lần (một lần của tệp chính và một lần của mathfunc.h), do vậy khi dịch chương trình, C++ sẽ báo lỗi (do hàm int max(inta, int b) được khai báo hai lần). Để khắc phục tình trạng này trong mylib.h ta thêm chỉ thị mới như sau:

```
// tệp mylib.h
#ifndef _MYLIB_                    // nếu chưa định nghĩa tên gọi
_MYLIB_
#define _MYLIB_                    // thì định nghĩa nó
int max(int a, int b)              // và các hàm khác
{
      return (a>b? a: b);
}
```

```
#endif
```

Như vậy khi chương trình dịch xử lý mylib.h lần đầu do _MYLIB_ chưa định nghĩa nên máy sẽ định nghĩa từ này, và dịch đoạn chương trình tiếp theo cho đến #endif. Lần thứ hai khi gặp lại đoạn lệnh này do _MYLIB_ đã được định nghĩa nên chương trình bỏ qua đoạn lệnh này không dịch.

Để cẩn thận trong cả mathfunc.h ta cũng sử dụng cú pháp này, vì có thể trong một chương trình khác mathfunc.h lại được sử dụng nhiều lần.

BÀI TẬP

Con trỏ

1. Hãy khai báo biến kí tự ch và con trỏ kiểu kí tự pc trỏ vào biến ch. Viết ra các cách gán giá trị 'A' cho biến ch.

2. Cho mảng nguyên cost. Viết ra các cách gán giá trị 100 cho phần tử thứ 3 của mảng.

3. Cho p, q là các con trỏ cùng trỏ đến kí tự c. Đặt *p = *q + 1. Có thể khẳng định: *q = *p - 1 ?

4. Cho p, q là các con trỏ trỏ đến biến nguyên x = 5. Đặt *p = *q + 1; Hỏi *q ?

5. Cho p, q, r, s là các con trỏ trỏ đến biến nguyên x = 10. Đặt *q = *p + 1; *r = *q + 1; *s = *r + 1. Hỏi giá trị của biến x ?

6. Chọn câu đúng nhất trong các câu sau:

A: Địa chỉ của một biến là số thứ tự của byte đầu tiên máy dành cho biến đó.

B: Địa chỉ của một biến là một số nguyên.

C: Số học địa chỉ là các phép toán làm việc trên các số nguyên biểu diễn địa chỉ của biến

D: a và b đúng

7. Chọn câu sai trong các câu sau:

A: Các con trỏ có thể phân biệt nhau bởi kiểu của biến mà nó trỏ đến.

B: Hai con trỏ trỏ đến các kiểu khác nhau sẽ có kích thước khác nhau.

C: Một con trỏ kiểu void có thể được gán bởi con trỏ có kiểu bất kỳ (cần ép kiểu).

D: Hai con trỏ cùng trỏ đến kiểu cấu trúc có thể gán cho nhau.

8. Cho con trỏ p trỏ đến biến x kiểu float. Có thể khẳng định ?

A: p là một biến và *p cũng là một biến

B: p là một biến và *p là một giá trị hằng

C: Để sử dụng được p cần phải khai báo float *p; và gán *p = x;

D: Cũng có thể khai báo void *p; và gán (float)p = &x;

9. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu x = *px thì y = *py B: Nếu x = y + z thì *px = y + z

C: Nếu *px = y + z thì *px = *py + z D: a, b, c đúng

10. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu *px = x thì *py = y B: Nếu *px = *py - z thì *px = y - z

C: Nếu *px = y - z thì x = y - z D: a, b, c đúng

11. Không dùng mảng, hãy nhập một dãy số nguyên và in ngược dãy ra màn hình.

12. Không dùng mảng, hãy nhập một dãy số nguyên và chỉ ra vị trí của số bé nhất, lớn nhất.

13. Không dùng mảng, hãy nhập một dãy số nguyên và in ra dãy đã được sắp xếp.

14. Không dùng mảng, hãy nhập một dãy kí tự. Thay mỗi kí tự ‘a’ trong dãy thành kí tự ‘b’ và in kết quả ra màn hình.

Con trỏ và xâu kí tự

15. Giả sử p là một con trỏ kiểu kí tự trỏ đến xâu "Tin học". Chọn câu đúng nhất trong các câu sau:

A: cout << p sẽ in ra dòng "Tin học" B: cout << p sẽ in ra dòng "Tin học"

C: cout << p sẽ in ra chữ cái 'T' D: b và c đúng

16. Xét chương trình (không kể các khai báo file nguyên mẫu):

```
char st[] = "tin học";
main() {
    char *p; p = new char[10];
    for (int i=0; st[i] != '\0'; i++) p[i] = st[i];
}
```

Chương trình trên chưa hoàn chỉnh vì:

A: Sử dụng sai cú pháp toán tử new

B: Sử dụng sai cú pháp p[i] (đúng ra là *(p+i))

C: Xâu p chưa có kết thúc

D: Cả a, b, c, đều sai

17. Để tính độ dài xâu một sinh viên viết đoạn chương trình sau:

```
char *st;
main()
{
        int len = 0; gets(st); while (st++ != '\0') len++; printf("%d",len);
}
```

Hãy chọn câu đúng nhất:

A: Chương trình trên là hoàn chỉnh B: Cần thay len++ bởi ++len

C: Cần thay st++ bởi *st++ D: Cần thay st++ != '\0' bởi st++ == '\0'

18. Cho xâu kí tự (dạng con trỏ) s. Hãy in ngược xâu ra màn hình.

19. Cho xâu kí tự (dạng con trỏ) s. Hãy copy từ s sang xâu t một đoạn bắt đầu tại vị trí m với độ dài n.

20. Cho xâu kí tự (dạng con trỏ) s. Hãy thống kê tần xuất xuất hiện của các kí tự có trong s. In ra màn hình theo thứ tự giảm dần của các tần xuất (tần xuất là tỉ lệ % số lần xuất hiện của x trên tổng số kí tự trong s).

Hàm

21. Chọn câu sai trong các câu sau đây:

A: Hàm không trả lại giá trị thì không cần khai báo kiểu giá trị của hàm.

B: Các biến được khai báo trong hàm là cục bộ, tự xoá khi hàm thực hiện xong

C: Hàm không trả lại giá trị sẽ có kiểu giá trị ngầm định là void.

D: Hàm là đơn vị độc lập, không được khai báo hàm lồng nhau.

22. Chọn câu đúng nhất trong các câu sau đây:

A: Hàm phải được kết thúc với 1 câu lệnh return

B: Phải có ít nhất 1 câu lệnh return cho hàm

C: Các câu lệnh return được phép nằm ở vị trí bất kỳ trong thân hàm

D: Không cần khai báo kiểu giá trị trả lại của hàm nếu hàm không có lệnh return

23. Chọn câu sai trong các câu sau đây:

A: Số tham số thực sự phải bằng số tham số hình thức trong lời gọi hàm

B: Các biến cục bộ trong thân hàm được chương trình dịch cấp phát bộ nhớ

C: Các tham số hình thức sẽ được cấp phát bộ nhớ tạm thời khi hàm được gọi

D: Kiểu của tham số thực sự phải bằng kiểu của tham số hình thức tương ứng với nó trong lời gọi hàm

24. Để thay đổi giá trị của tham biến, các đối của hàm cần khai báo dưới dạng:

A: biến bình thường và tham đối được truyền theo giá trị

B: biến con trỏ và tham đối được truyền theo giá trị

C: biến bình thường và tham đối được truyền theo địa chỉ

D: biến tham chiếu và tham đối được truyền theo giá trị

25. Viết hàm tìm UCLN của 2 số. áp dụng hàm này (AD:) để tìm UCLN của 4 số nhập từ bàn phím.

26. Viết hàm kiểm tra một số nguyên n có là số nguyên tố. AD: In ra các số nguyên tố bé hơn 1000.

27. Viết hàm kiểm tra một số nguyên n có là số nguyên tố. AD: In các cặp số sinh đôi < 1000. (Các số "sinh đôi" là các số nguyên tố mà khoảng cách giữa chúng là 2).

28. Viết hàm kiểm tra một năm có là năm nhuận. AD: In ra các năm nhuận từ năm 1000 đến 2000.

29. Viết hàm xoá dấu cách đầu tiên trong một xâu. AD: Xoá tất cả dấu cách trong xâu.

30. Viết hàm thay 2 dấu cách bởi 1 dấu cách. AD: Cắt các dấu cách giữa 2 từ của một xâu về còn 1 dấu cách.

31. Viết hàm tráo đổi giá trị của 2 số. AD: sắp xếp dãy số.

32. Viết hàm giải phương trình bậc 2. Dùng chương trình con này tìm nghiệm của một phương trình chính phương bậc 4.

33. Số hoàn chỉnh là số bằng tổng mọi ước của nó (Ví dụ 6 = 1 + 2 + 3). Hãy in ra mọi số hoàn chỉnh từ 1 đến 100.

34. Tính tổng của dãy phân số. In ra màn hình kết quả dưới dạng phân số tối giản.

35. Nhập số tự nhiên chẵn n > 2. Hãy kiểm tra số này có thể biểu diễn được dưới dạng tổng của 2 số nguyên tố hay không ?.

36. In tổng của n số nguyên tố đầu tiên.

37. Tính phần diện tích giới hạn bởi hình tròn bán kính R và hình vuông ngoại tiếp của nó.

38. Chuẩn hoá một xâu (cắt kí tự trắng 2 đầu, cắt bớt các dấu trắng (chỉ để lại 1) giữa các từ, viết hoa đầu từ).

39. Viết chương trình nhập số nguyên lớn (không quá một tỷ), hãy đọc giá trị của nó bằng cách in ra xâu kí tự tương ứng với giá trị của nó. Ví dụ 1094507 là "Một triệu, (không trăm) chín tư nghìn, năm trăm linh bảy đơn vị".

40. Viết chương trình sắp xếp theo tên một mảng họ tên nào đó.

41. Tìm tất cả số tự nhiên có 4 chữ số mà trong mỗi số không có 2 chữ số nào giống nhau.

42. Nhập số tự nhiên n. Hãy biểu diễn n dưới dạng tích của các thừa số nguyên tố.

Đệ qui

43. Nhập số nguyên dương N. Viết hàm đệ qui tính:

a. $S_1 = \dfrac{1+2+3+...+N}{N}$

b. $S_2 = \sqrt{1^2+2^2+3^2+...+N^2}$

1. Nhập số nguyên dương n. Viết hàm đệ qui tính:

a. $S_1 = \sqrt{3+\sqrt{3+\sqrt{3+...+\sqrt{3}}}}$ n dấu căn

b. $S_2 = \dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2+...\dfrac{1}{2}}}}$ n dấu chia

44. Viết hàm đệ qui tính n!. áp dụng chương trình con này tính tổ hợp chập k theo công thức truy hồi: C(n, k) = n!/(k! (n-k)!)

45. Viết hàm đệ qui tính số fibonaci thứ n. Dùng chương trình con này tính f(2) + f(4) + f(6) + f(8).

46. Viết dưới dạng đệ qui các hàm

a. daoxau b. UCLN c. Fibonaci d. Tháp Hà Nội

47. Viết macro tráo đổi nội dung 2 biến. AD: Sắp xếp dãy số.

CHƯƠNG 5

DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

Kiểu cấu trúc
Cấu trúc tự trỏ và danh sách liên kết
Kiểu hợp
Kiểu liệt kê

Để lưu trữ các giá trị gồm nhiều thành phần dữ liệu giống nhau ta có kiểu biến mảng. Thực tế rất nhiều dữ liệu là tập các kiểu dữ liệu khác nhau tập hợp lại, để quản lý dữ liệu kiểu này C++ đưa ra kiểu dữ liệu cấu trúc. Một ví dụ của dữ liệu kiểu cấu trúc là một bảng lý lịch trong đó mỗi nhân sự được lưu trong một bảng gồm nhiều kiểu dữ liệu khác nhau như họ tên, tuổi, giới tính, mức lương …

I. KIỂU CẤU TRÚC

1. Khai báo, khởi tạo

Để tạo ra một kiểu cấu trúc NSD cần phải khai báo tên của kiểu (là một tên gọi do NSD tự đặt), tên cùng với các thành phần dữ liệu có trong kiểu cấu trúc này. Một kiểu cấu trúc được khai báo theo mẫu sau:

```
struct <tên kiểu>
{
    các thành phần ;
} <danh sách biến>;
```

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).
- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

– Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

struct <tên cấu trúc> <danh sách biến> ; // kiểu cũ trong C

hoặc

<tên cấu trúc> <danh sách biến> ; // trong C++

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> biến = { giá trị khởi tạo } ;

Ví dụ:

– Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số.

```
struct Phanso
{
        int tu ;
        int mau ;
} ;
```

hoặc:

```
struct Phanso { int tu, mau ; }
```

– Kiểu ngày tháng gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Ngaythang {
        int ng ;
        int th ;
        int nam ;
} holiday = { 1,5,2000 } ;
```

một biến holiday cũng được khai báo kèm cùng kiểu này và được khởi tạo bởi bộ số 1. 5. 2000. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 1, th = 5 và nam = 2000.

– Kiểu Lop dùng chứa thông tin về một lớp học gồm tên lớp và sĩ số sinh viên. Các biến kiểu Lop được khai báo là daihoc và caodang, trong đó daihoc được khởi tạo bởi bộ giá trị {"K41T", 60} với ý nghĩa tên lớp đại học là K41T và sĩ số là 60 sinh viên.

```
struct Lop {
        char tenlop[10],
```

```
        int soluong;
    } ;
    struct Lop daihoc = {"K41T", 60}, caodang ;
```

hoặc:

```
    Lop daihoc = {"K41T", 60}, caodang ;
```

– Kiểu Sinhvien gồm có các trường hoten để lưu trữ họ và tên sinh viên, ns lưu trữ ngày sinh, gt lưu trữ giới tính dưới dạng số (qui ước 1: nam, 2: nữ) và cuối cùng trường diem lưu trữ điểm thi của sinh viên. Các trường trên đều có kiểu khác nhau.

```
    struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
    } x, *p, K41T[60];
    Sinhvien y = {"NVA", {1,1,1980}, 1} ;
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng K41T với 60 phần tử kiểu Sinhvien. Một biến y được khai báo thêm và kèm theo khởi tạo giá trị {"NVA", {1,1,1980}, 1}, tức họ tên của sinh viên y là "NVA", ngày sinh là 1/1/1980, giới tính nam và điểm thi để trống. Đây là kiểu khởi tạo thiếu giá trị, giống như khởi tạo mảng, các giá trị để trống phải nằm ở cuối bộ giá trị khởi tạo (tức các thành phần bỏ khởi tạo không được nằm xen kẽ giữa những thành phần được khởi tạo).Ví dụ này còn minh hoạ cho các cấu trúc lồng nhau, cụ thể trong kiểu cấu trúc Sinhvien có một thành phần cũng kiểu cấu trúc là thành phần ns.

2. Truy nhập các thành phần kiểu cấu trúc

Để truy nhập vào các thành phần kiểu cấu trúc ta sử dụng cú pháp: tên biến.tên thành phần hoặc tên biến → tên thành phần đối với biến con trỏ cấu trúc. Cụ thể:

– Đối với biến thường: **tên biến.tên thành phần**

Ví dụ:

```
    struct Lop {
        char tenlop[10];
        int siso;
```

```
} ;
Lop daihoc = "K41T", caodang ;
caodang.tenlop = daihoc.tenlop ;      // gán tên lớp cđẳng bởi tên lớp đhọc
caodang.siso++;                       // tăng sĩ số lớp caodang lên 1
```

– Đối với biến con trỏ: **tên biến → tên thành phần**

Ví dụ:

```
struct Sinhvien {
      char hoten[25] ;
      Ngaythang ns;
      int gt;
      float diem ;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1} ;
y.diem = 5.5 ;                        // gán điểm thi cho sinh viên y
p = new Sinhvien ;                    // cấp bộ nhớ chứa 1 sinh viên
strcpy(p→hoten, y.hoten) ;            // gán họ tên của y cho sv trỏ bởi p
cout << p→hoten << y.hoten;           // in hoten của y và con trỏ p
```

– Đối với biến mảng: truy nhập thành phần mảng rồi đến thành phần cấu trúc.

Ví dụ:

```
strcpy(K41T[1].hoten, p→hoten) ;      // gán họ tên cho sv đầu tiên của lớp
K41T[1].diem = 7.0 ;                  // gán điểm cho sv đầu tiên
```

– Đối với cấu trúc lồng nhau. Truy nhập thành phần ngoài rồi đến thành phần của cấu trúc bên trong, sử dụng các phép toán . hoặc → (các phép toán lấy thành phần) một cách thích hợp.

```
x.ngaysinh.ng = y.ngaysinh.ng ;       // gán ngày,
x.ngaysinh.th = y.ngaysinh.th ;       // tháng,
x.ngaysinh.nam = y.ngaysinh.nam ;     // năm sinh của y cho x.
```

3. Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết

câu lệnh vào/ra từng cho từng thành phần. Nhận xét này được minh họa trong ví dụ sau:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem
cout << "Thông tin về sinh viên x là:" << endl ;
cout << "Họ và tên: " << x.hoten << endl;
cout << "Sinh ngày: " << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam ;
cout << "Giới tính: " << (x.gt == 1) ? "Nam": "Nữ ;
cout << x.diem
```

Tuy nhiên, khác với biến mảng, **đối với cấu trúc chúng ta có thể gán giá trị của 2 biến cho nhau**. Phép gán này cũng tương đương với việc gán từng thành phần của cấu trúc. Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y, *p ;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
```

```
cin >> x.diem

y = x ;       // Đối với biến mảng, phép gán này là không thực hiện được
p = new Sinhvien[1] ; *p = x ;

cout << "Thông tin về sinh viên y là:" << endl ;
cout << "Họ và tên: " << y.hoten << endl;
cout << "Sinh ngày: " << y.ns.ng << "/" << y.ns.th << "/" << y.ns.nam ;
cout << "Giới tính: " << (y.gt = 1) ? "Nam": "Nữ ;
cout << y.diem
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc. Cách gán này chỉ thực hiện được khi khởi tạo. Ví dụ:

```
Sinhvien x = { "NVA", {1,1,1980}, 1, 7.0}, y ;          // được
y = { "NVA", {1,1,1980}, 1, 7.0};                       // không được
y = x;                                                  // được
```

4. Các ví dụ minh hoạ

Dưới đây chúng ta đưa ra một vài ví dụ minh hoạ cho việc sử dụng kiểu cấu trúc.

Ví dụ 1 : Cộng, trừ, nhân chia hai phân số được cho dưới dạng cấu trúc.

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
      int tu ;
      int mau ;
} a, b, c ;

void main()
{
      clrscr();
      cout << "Nhập phân số a:" << endl ;          // nhập a
      cout << "Tử:"; cin >> a.tu;
      cout << "Mẫu:"; cin >> a.mau;
```

```
        cout << "Nhập phân số b:" << endl ;              // nhập b
        cout << "Tử:"; cin >> b.tu;
        cout << "Mẫu:"; cin >> b.mau;
        c.tu = a.tu*b.mau + a.mau*b.tu;                 // tính và in a+b
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau - a.mau*b.tu;                 // tính và in a-b
        c.mau = a.mau*b.mau;
        cout << "a - b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.tu;                               // tính và in axb
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau;                              // tính và in a/b
        c.mau = a.mau*b.tu;
        cout << "a + b = " << c.tu << "/" << c.mau;
        getch();
}
```

Ví dụ 2 : Nhập mảng K41T. Tính tuổi trung bình của sinh viên nam, nữ. Hiện danh sách của sinh viên có điểm thi cao nhất.

```
#include <iostream.h>
#include <conio.h>
void main()
{
        struct Sinhvien {
                char hoten[25] ;
                Ngaythang ns;
                int gt;
                float diem ;
        } x, K41T[60];
        int i, n;
```

```
        // nhập dữ liệu
        cout << "Cho biết số sinh viên: "; cin >> n;
        for (i=1, i<=n, i++)
        {
                cout << "Nhap sinh vien thu " << i);
                cout << "Ho ten: " ; cin.getline(x.hoten);
                cout << "Ngày sinh: " ; cin >> x.ns.ng >> x.ns.th >>x.ns.nam ;
                cout << "Giới tính: " ; cin >> x.gt ;
                cout << "Điểm: " ; cin >> x.diem ;
                cin.ignore();
                K41T[i] = x ;
        }
}

// Tính điểm trung bình
float tbnam = 0, tbnu = 0;
int sonam = 0, sonu = 0 ;
for (i=1; i<=n; i++)
        if (K41T[i].gt == 1) { sonam++ ; tbnam += K41T[1].diem ; }
else { sonu++ ; tbnu += K41T[1].diem ; }
cout << "Điểm trung bình của sinh viên nam là " << tbnam/sonam ;
cout << "Điểm trung bình của sinh viên nữ là " << tbnu/sonu ;

// In danh sách sinh viên có điểm cao nhất
float diemmax = 0;
for (i=1; i<=n; i++)                                    // Tìm điểm cao nhất
if (diemmax < K41T[i].diem) diemmax = K41T[i].diem ;

for (i=1; i<=n; i++)                                    // In danh sách
{
        if (K41T[i].diem < diemmax) continue ;
```

```
            x = K41T[1] ;
            cout << x.hoten << '\t' ;
            cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << '\t' ;
            cout << (x.gt == 1) ? "Nam": "Nữ" << '\t' ;
            cout << x.diem << endl;
        }
    }
```

5. Hàm với cấu trúc

a. Con trỏ và địa chỉ cấu trúc

Một con trỏ cấu trúc cũng giống như con trỏ trỏ đến các kiểu dữ liệu khác, có nghĩa nó chứa địa chỉ của một biến cấu trúc hoặc một vùng nhớ có kiểu cấu trúc nào đó. Một con trỏ cấu trúc được khởi tạo bởi:

- Gán địa chỉ của một biến cấu trúc, một thành phần của mảng, tương tự nếu địa chỉ của mảng (cũng là địa chỉ của phần tử đầu tiên của mảng) gán cho con trỏ thì ta cũng gọi là con trỏ mảng cấu trúc. Ví dụ:

```
struct Sinhvien {
  char hoten[25] ;
  Ngaythang ns;
  int gt;
  float diem ;
} x, y, *p, lop[60];

p = &x ;                    // cho con trỏ p trỏ tới biến cấu trúc x
p→diem = 5.0;               // gán giá trị 5.0 cho điểm của biến x
p = &lop[10] ;              // cho p trỏ tới sinh viên thứ 10 của lớp
cout << p→hoten;            // hiện họ tên của sinh viên này
*p = y ;                    // gán lại sinh viên thứ 10 là y
(*p).gt = 2;                // sửa lại giới tính của sinh viên thứ 10 là nữ
```

Chú ý: thông qua ví dụ này ta còn thấy một cách khác nữa để truy nhập các thành phần của x được trỏ bởi con trỏ p. Khi đó *p là tương đương với x, do vậy ta dùng lại cú

pháp sử dụng toán tử . sau *p để lấy thành phần như (*p).hoten, (*p).diem, …

- Con trỏ được khởi tạo do xin cấp phát bộ nhớ. Ví dụ:

```
Sinhvien *p, *q ;
p = new Sinhvien[1];
q = new Sinhvien[60];
```

trong ví dụ này *p có thể thay cho một biến kiểu sinh viên (tương đương biến x ở trên) còn q có thể được dùng để quản lý một danh sách có tối đa là 60 sinh viên (tương đương biến lop[60], ví dụ khi đó *(p+10) là sinh viên thứ 10 trong danh sách).

- Đối với con trỏ p trỏ đến mảng a, chúng ta có thể sử dụng một số cách sau để truy nhập đến các trường của các thành phần trong mảng, ví dụ để truy cập hoten của thành phần thứ i của mảng a ta có thể viết:
  - p[i].hoten
  - (p+i)→hoten
  - *(p+i).hoten

Nói chung các cách viết trên đều dễ nhớ do suy từ kiểu mảng và con trỏ mảng. Cụ thể trong đó p[i] là thành phần thứ i của mảng a, tức a[i]. (p+i) là con trỏ trỏ đến thành phần thứ i và *(p+i) chính là a[i]. Ví dụ sau gán giá trị cho thành phần thứ 10 của mảng sinh viên lop, sau đó in ra màn hình các thông tin này. Ví dụ dùng để minh hoạ các cách truy nhập trường dữ liệu của thành phần trong mảng lop.

Ví dụ 3 :

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60] ;

strcpy(lop[10].hoten, "NVA");
lop[10].gt = 1; lop[10].diem = 9.0 ;
Sinhvien *p ;                    // khai báo thêm biến con trỏ Sinh viên
p = &lop ;                       // cho con trỏ p trỏ tới mảng lop
```

```
cout << p[10].hoten ;          // in họ tên sinh viên thứ 10
cout << (p+10) → gt ;          // in giới tính của sinh viên thứ 10
cout << (*(p+10)).diem ;       // in điểm của sinh viên thứ 10
```

Chú ý: Độ ưu tiên của toán tử lấy thành phần (dấu chấm) là cao hơn các toán tử lấy địa chỉ (&) và lấy giá trị (*) nên cần phải viết các dấu ngoặc đúng cách.

b. Địa chỉ của các thành phần của cấu trúc

Các thành phần của một cấu trúc cũng giống như các biến, do vậy cách lấy địa chỉ của các thành phần này cũng tương tự như đối với biến bình thường. Chẳng hạn địa chỉ của thành phần giới tính của biến cấu trúc x là &x.gt (lưu ý độ ưu tiên của . cao hơn &, nên &x.gt là cũng tương đương với &(x.gt)), địa chỉ của trường hoten của thành phần thứ 10 trong mảng lớp là lop[10].hoten (hoten là xâu kí tự), tương tự địa chỉ của thành phần diem của biến được trỏ bởi p là &(p→diem).

Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60], *p, x = { "NVA", {1,1,1980}, 1, 9.0) };
        lop[10] = x; p = &lop[10] ;     // p trỏ đến sinh viên thứ 10 trong lop
        char *ht; int *gt; float *d;    // các con trỏ kiểu thành phần
        ht = x.ht ;                     // cho ht trỏ đến thành phần hoten của x
        gt = &(lop[10].gt) ;            // gt trỏ đến gt của sinh viên thứ 10
        d = &(p→diem) ;                 // p trỏ đến diem của sv p đang trỏ
        cout << ht ;                    // in họ tên sinh viên x
        cout << *gt ;                   // in giới tính của sinh viên thứ 10
        cout << *d ;                    // in điểm của sinh viên p đang trỏ.
```

c. Đối của hàm là cấu trúc

Một cấu trúc có thể được sử dụng để làm đối của hàm dưới các dạng sau đây:

– Là một biến cấu trúc, khi đó tham đối thực sự là một cấu trúc.

- Là một con trỏ cấu trúc, tham đối thực sự là địa chỉ của một cấu trúc.
- Là một tham chiếu cấu trúc, tham đối thực sự là một cấu trúc.
- Là một mảng cấu trúc hình thức hoặc con trỏ mảng, tham đối thực sự là tên mảng cấu trúc.

Ví dụ 4 : Ví dụ sau đây cho phép tính chính xác khoảng cách của 2 ngày tháng bất kỳ, từ đó có thể suy ra thứ của một ngày tháng bất kỳ. Đối của các hàm là một biến cấu trúc.

- Khai báo

```
struct DATE {                                   // Kiểu ngày tháng
  int ngay ;
  int thang;
  int nam ;
};
// Số ngày của mỗi tháng
int n[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
```

- Hàm tính năm nhuận hay không nhuận, trả lại 1 nếu năm nhuận, ngược lại trả 0.

```
int Nhuan(int nm)
{
  return (nam%4==0 && nam%100!=0 || nam%400==0)? 1: 0;
}
```

- Hàm trả lại số ngày của một tháng bất kỳ. Nếu năm nhuận và là tháng hai số ngày của tháng hai (28) được cộng thêm 1.

```
int Ngayct(int thang, int nam)
{
  return n[thang] + ((thang==2) ? Nhuan(nam): 0);
}
```

- Hàm trả lại số ngày tính từ ngày 1 tháng 1 năm 1 bằng cách cộng dồn số ngày của từng năm từ năm 1 đến năm hiện tại, tiếp theo cộng dồn số ngày từng tháng của năm hiện tại cho đến tháng hiện tại và cuối cùng cộng thêm số ngày hiện tại.

```
long Tongngay(DATE d)
{
  long i, kq = 0;
  for (i=1; i<d.nam; i++) kq += 365 + Nhuan(i);
  for (i=1; i<d.thang; i++) kq += Ngayct(i,d.nam);
  kq += d.ngay;
  return kq;
}
```

- Hàm trả lại khoảng cách giữa 2 ngày bất kỳ.

```
long Khoangcach(DATE d1, DATE d2)
{
  return Tongngay(d1)-Tongngay(d2);
}
```

- Hàm trả lại thứ của một ngày bất kỳ. Qui ước 1 là chủ nhật, 2 là thứ hai, … Để tính thứ hàm dựa trên một ngày chuẩn nào đó (ở đây là ngày 1/1/2000, được biết là thứ bảy). Từ ngày chuẩn này nếu cộng hoặc trừ 7 sẽ cho ra ngày mới cũng là thứ bảy. Từ đó, tính khoảng cách giữa ngày cần tính thứ và ngày chuẩn. Tìm phần dư của phép chia khoảng cách này với 7, nếu phần dư là 0 thì thứ là bảy, phần dư là 1 thì thứ là chủ nhật …

```
int Thu(DATE d)
{
  DATE curdate = {1,1,2000};                    // ngày 1/1/2000 là thứ bảy
  long kc = Khoangcach(d, curdate);
  int du = kc % 7; if (du < 0) du += 7;
  return du;
}
```

- Hàm dịch một số dư sang thứ

```
char* Dich(int t)
{
  char* kq = new char[10];
  switch (t) {
  case 0: strcpy(kq, "thứ bảy"); break;
        case 1: strcpy(kq, "chủ nhật"); break;
  case 2: strcpy(kq, "thứ hai"); break;
        case 3: strcpy(kq, "thứ ba"); break;
  case 4: strcpy(kq, "thứ tư"); break;
  case 5: strcpy(kq, "thứ năm"); break;
        case 6: strcpy(kq, "thứ sáu"); break;
  }
  return kq;
}
```

- Hàm main()

```
void main()
{
  DATE d;
  cout << "Nhap ngay thang nam: " ;
  cin >> d.ngay >> d.thang >> d.nam ;
  cout << "Ngày " << d.ngay << "/" << d.thang << "/" << d.nam ;
  cout << " là " << Dich(Thu(d));
}
```

Ví dụ 5 : Chương trình đơn giản về quản lý sinh viên.

- Khai báo.

```
struct Sinhvien {                    // cấu trúc sinh viên
  char hoten[25] ;
  Ngaythang ns;
  int gt;
```

```
    float diem ;
} ;
Sinhvien lop[3];                    // lớp chứa tối đa 3 sinh viên
```

- Hàm in thông tin về sinh viên sử dụng biến cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void in(Sinhvien x)
{
    cout << x.hoten << "\t" ;
    cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << "\t" ;
    cout << x.gt << "\t";
    cout << x.diem << endl;
}
```

- Hàm nhập thông tin về sinh viên sử dụng con trỏ sinh viên làm đối. Trong lời gọi sử dụng địa chỉ của một cấu trúc để truyền cho hàm.

```
void nhap(Sinhvien *p)
{
    cin.ignore();
    cout << "Họ tên: "; cin.getline(p→hoten, 25) ;
    cout << "Ngày sinh: ";
    cin >> (p→ns).ng >> (p→ns).th >> (p→ns).nam ;
    cout << "Giới tính: "; cin >> (p→gt) ;
    cout << "Điểm: "; cin >> (p→diem) ;
}
```

- Hàm sửa thông tin về sinh viên sử dụng tham chiếu cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void sua(Sinhvien &r)
{
    int chon;
    do {
        cout << "1: Sửa họ tên" << endl ;
        cout << "2: Sửa ngày sinh" << endl ;
```

```
            cout << "3: Sửa giới tính" << endl ;
            cout << "4: Sửa điểm" << endl ;
            cout << "0: Thôi" << endl ;
            cout << "Sửa (0/1/2/3/4) ? ; cin >> chon ; cin.ignore();
            switch (chon) {
                case 1: cin.getline(r.hoten, 25) ; break;
                case 2: cin >> r.ns.ng >> r.ns.th >> r.ns.nam ; break;
                case 3: cin >> r.gt ; break;
                case 4: cin >> r.diem ; break;
            }
        } while (chon) ;
    }
```

- Hàm nhapds nhập thông tin của mọi sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void nhapds(Sinhvien *a)
{
    int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
    for (int i=1; i<=sosv; i++) nhap(&a[i]) ;
}
```

- Hàm suads cho phép sửa thông tin của sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void suads(Sinhvien *a)
{
    int chon;
    cout << "Chọn sinh viên cần sửa: " ; cin >> chon ; cin.ignore();
    sua(a[chon]) ;
}
```

- Hàm inds hiện thông tin của mọi sinh viên trong mảng, sử dụng hằng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void hien(const Sinhvien *a)
{
        int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
        for (int i=1; i<=sosv; i++) in(a[i]) ;
}
```

- Hàm main() gọi chạy các hàm trên để nhập, in, sửa danh sách sinh viên.

```
void main()
{
    nhapds(lop) ;
    inds(lop);
    suads(lop);
    inds(lop);
}
```

d. Giá trị hàm là cấu trúc

Cũng tương tự như các kiểu dữ liệu cơ bản, giá trị trả lại của một hàm cũng có thể là các cấu trúc dưới các dạng sau:

- là một biến cấu trúc.
- là một con trỏ cấu trúc.
- là một tham chiếu cấu trúc.

Sau đây là các ví dụ minh hoạ giá trị cấu trúc của hàm.

Ví dụ 6 : Đối và giá trị của hàm là cấu trúc: Cộng, trừ hai số phức.

– Khai báo kiểu số phức

```
struct Sophuc                        // Khai báo kiểu số phức dùng chung
{
        float thuc;
        float ao;
};
```

- Hàm cộng 2 số phức, trả lại một số phức

```
Sophuc Cong(Sophuc x, Sophuc y)
{
```

```
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm trừ 2 số phức, trả lại một số phức

```
Sophuc Tru(Sophuc x, Sophuc y)
{
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm in một số phức dạng (r + im)

```
void In(Sophuc x)
{
        cout << "(" << x.thuc << "," << x.ao << ")" << endl ;
}
```

- Hàm chính

```
void main()
{
        Sophuc x, y, z ;
        cout << "x = " ; cin >> x.thuc >> x.ao ;
        cout << "y = " ; cin >> y.thuc >> y.ao ;
        cout << "x + y = " ; In(Cong(x,y)) ;
        cout << "x - y = " ; In(Tru(x,y)) ;
}
```

Ví dụ 7 : Chương trình nhập và in thông tin về một lớp cùng sinh viên có điểm cao nhất lớp.

- Khai báo kiểu dữ liệu Sinh viên và biến mảng lop.

```
struct Sinhvien {
        char *hoten ;
        float diem ;
} lop[4] ;
```

- Hàm nhập sinh viên, giá trị trả lại là một con trỏ trỏ đến dữ liệu vừa nhập.

```
Sinhvien* nhap()
{
        Sinhvien* kq = new Sinhvien[1];          // nhớ cấp phát vùng nhớ
        kq->hoten = new char[15];                // cho cả con trỏ hoten
        cout << "Họ tên: "; cin.getline(kq->hoten,30);
        cout << "Điểm: "; cin >> kq->diem; cin.ignore();
        return kq;                               // trả lại con trỏ kq
}
```

- Hàm tìm sinh viên có điểm cao nhất, giá trị trả lại là một tham chiếu đến sinh viên tìm được.

```
Sinhvien& svmax()
{
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;   // bỏ thành phần thứ 0
        float maxdiem = 0;
        int kmax;                                    // chỉ số sv có điểm max
        for (int i=1; i<sosv; i++)
        if (maxdiem < lop[i].diem)
        {
                maxdiem = lop[i].diem ;
                kmax = i;
        }
        return lop[kmax];                            // trả lại sv có điểm max
```

```
}
```

- Hàm in thông tin của một sinh viên x

```
void in(Sinhvien x)
{
        cout << x.hoten << "\t";
        cout << x.diem << endl;
}
```

- Hàm chính

```
void main()
{
        clrscr();
        int i;
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;    // bỏ thành phần thứ 0
        for (i=1; i<=sosv; i++) lop[i] = *nhap();       // nhập danh sách lớp
        for (i=1; i<=sosv; i++) in(lop[i]);             // in danh sách lớp
        Sinhvien &b = svmax();          // khai báo tham chiếu b và cho
                                        // tham chiếu đến sv có điểm max
        in(b);                          // in sinh viên có điểm max
        getch();
}
```

6. Cấu trúc với thành phần kiểu bit

a. Trường bit

Thông thường các trường trong một cấu trúc thường sử dụng ít nhất là 2 byte tức 16 bit. Trong nhiều trường hợp một số trường có thể chỉ cần đến số bit ít hơn, ví dụ trường gioitinh thông thường chỉ cần đến 1 bit để lưu trữ. Những trường hợp như vậy ta có thể khai báo kiểu bit cho các trường này để tiết kiệm bộ nhớ. Tuy nhiên, cách khai báo này ít được sử dụng trừ khi cần thiết phải truy nhập đến mức bit của dữ liệu trong các chương trình liên quan đến hệ thống.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau,

trong đó $0 \le n < 15$. Ví dụ do độ lớn của ngày không vượt quá 31, tháng không vượt quá 12 nên 2 trường này trong cấu trúc ngày tháng có thể khai báo tiết kiệm hơn bằng 5 và 4 bit như sau:

```
struct Date {
        int ng: 5;
        int th: 4;
        int nam;
} ;
```

b. Đặc điểm

Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

- Các bit được bố trí liên tục trên dãy các byte.
- Kiểu trường bit phải là int (signed hoặc unsigned).
- Độ dài mỗi trường bit không quá 16 bit.
- Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu {
        int: 8;
        int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

- Không cho phép lấy địa chỉ của thành phần kiểu bit.
- Không thể xây dựng được mảng kiểu bit.
- Không được trả về từ hàm một thành phần kiểu bit. Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ;                    // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ; return tam ;
```

- Tiết kiệm bộ nhớ
- Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu hợp).

7. Câu lệnh typedef

Để thuận tiện trong sử dụng, thông thường các kiểu được NSD tạo mới sẽ được gán cho một tên kiểu bằng câu lệnh typedef như sau:

typedef <kiểu> <tên_kiểu> ;

Ví dụ: Để tạo kiểu mới có tên Bool và chỉ chứa giá trị nguyên (thực chất chỉ cần 2 giá trị 0, 1), ta có thể khai báo:

```
typedef int Bool;
```

khai báo này cho phép xem Bool như kiểu số nguyên.

hoặc có thể đặt tên cho kiểu ngày tháng là Date với khai báo sau:

```
typedef struct Date {
    int ng;
    int th;
    int nam;
};
```

khi đó ta có thể sử dụng các tên kiểu này trong các khai báo (ví dụ tên kiểu của đối, của giá trị hàm trả lại …).

8. Hàm sizeof()

Hàm trả lại kích thước của một biến hoặc kiểu. Ví dụ:

```
Bool a, b;
Date x, y, z[50];
cout << sizeof(a) << sizeof(b) << sizeof(Bool) ;                    // in 2 2 2
cout << "Số phần tử của z = " << sizeof(z) / sizeof(Date)           // in 50
```

II. CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT

Thông thường khi thiết kế chương trình chúng ta chưa biết được số lượng dữ liệu cần dùng là bao nhiêu để khai báo số biến cho phù hợp. Đặc biệt là biến dữ liệu kiểu mảng. Số lượng các thành phần của biến mảng cần phải khai báo trước và chương trình dịch sẽ bố trí vùng nhớ cố định cho các biến này. Do buộc phải khai báo trước số lượng thành phần nên kiểu mảng thường dẫn đến hoặc là lãng phí bộ nhớ (khi chương trình không dùng hết) hoặc là không đủ để chứa dữ liệu (khi chương trình cần chứa dữ liệu với số lượng thành phần lớn hơn).

Để khắc phục tình trạng này C++ cho phép cấp phát bộ nhớ động, nghĩa là số

lượng các thành phần không cần phải khai báo trước. Bằng toán tử new chúng ta có thể xin cấp phát vùng nhớ theo nhu cầu mỗi khi chạy chương trình. Tuy nhiên, cách làm này dẫn đến việc dữ liệu của một danh sách sẽ không còn nằm liên tục trong bộ nhớ như đối với biến mảng. Mỗi lần xin cấp phát bởi toán tử new, chương trình sẽ tìm một vùng nhớ đang rỗi bất kỳ để cấp phát cho biến và như vậy dữ liệu sẽ nằm rải rác trong bộ nhớ. Từ đó, để dễ dàng quản lý trật tự của một danh sách các dữ liệu, mỗi thành phần của danh sách cần phải chứa địa chỉ của thành phần tiếp theo hoặc trước nó (điều này là không cần thiết đối với biến mảng vì các thành phần của chúng sắp xếp liên tục, kề nhau). Từ đó, mỗi thành phần của danh sách sẽ là một cấu trúc, ngoài các thành phần chứa thông tin của bản thân, chúng còn phải có thêm một hoặc nhiều con trỏ để trỏ đến các thành phần tiếp theo hay đứng trước. Các cấu trúc như vậy được gọi là cấu trúc tự trỏ vì các thành phần con trỏ trong cấu trúc này sẽ trỏ đến các vùng dữ liệu có kiểu chính là kiểu của chúng.

1. Cấu trúc tự trỏ

Một cấu trúc có chứa ít nhất một thành phần con trỏ có kiểu của chính cấu trúc đang định nghĩa được gọi là cấu trúc tự trỏ. Có thể khai báo cấu trúc tự trỏ bởi một trong những cách sau:

Cách 1:

```
typedef struct <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc
struct <tên cấu trúc>
{
      các thành phần chứa thông tin … ;
      <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sv Sinhvien ;                    // lưu ý phải có từ khoá struct
struct Sv
{
      char hoten[30] ;        // thành phần chứa thông tin
      float diem ;            // thành phần chứa thông tin
      Sinhvien *tiep ;        // thành phần con trỏ chứa địa chỉ tiếp theo
} ;
```

Cách 2:

```
struct <tên cấu trúc>
{
    các thành phần chứa thông tin … ;
    <tên cấu trúc> *con trỏ ;
} ;
typedef <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc tự trỏ
```

Ví dụ:

```
struct Sv
{
    char hoten[30] ;            // thành phần chứa thông tin
    float diem ;                // thành phần chứa thông tin
    Sv *tiep ;                  // thành phần con trỏ chứa địa chỉ tiếp theo
} ;
typedef Sv Sinhvien ;
```

Cách 3:

```
typedef struct <tên kiểu>       // định nghĩa tên cấu trúc tự trỏ
{
    các thành phần chứa thông tin … ;
    <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sinhvien
{
    char hoten[30] ;            // thành phần chứa thông tin
    float diem ;                // thành phần chứa thông tin
    Sinhvien *tiep ;            // con trỏ chứa địa chỉ thành phần tiếp theo
} ;
```

Cách 4:

```
struct <tên kiểu>
{
        các thành phần chứa thông tin … ;
        <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
struct Sinhvien
{
        char hoten[30] ;            // thành phần chứa thông tin
        float diem ;                // thành phần chứa thông tin
        Sinhvien *tiep ;            // con trỏ chứa địa chỉ của phần tử tiếp.
} ;
```

Trong các cách trên ta thấy 2 cách khai báo cuối cùng là đơn giản nhất. C++ quan niệm các tên gọi đứng sau các từ khoá struct, union, enum là các tên kiểu (dù không có từ khoá typedef), do vậy có thể sử dụng các tên này để khai báo.

2. Khái niệm danh sách liên kết

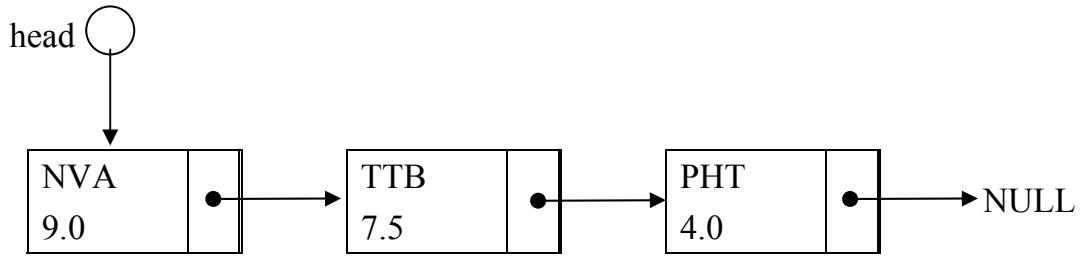
Danh sách liên kết là một cấu trúc dữ liệu cho phép thể hiện và quản lý danh sách bằng các cấu trúc liên kết với nhau thông qua các con trỏ trong cấu trúc. Có nhiều dạng danh sách liên kết phụ thuộc vào các kết nối, ví dụ:

- Danh sách liên kết đơn, mỗi cấu trúc chứa một con trỏ trỏ đến cấu trúc tiếp theo hoặc trước đó. Đối với danh sách con trỏ trỏ về trước, cấu trúc đầu tiên của danh sách sẽ trỏ về hằng con trỏ NULL, cấu trúc cuối cùng được đánh dấu bởi con trỏ last là con trỏ trỏ vào cấu trúc này. Đối với danh sách con trỏ trỏ về cấu trúc tiếp theo, cấu trúc đầu sẽ được đánh dấu bằng con trỏ head và cấu trúc cuối cùng chưa con trỏ NULL.
- Danh sách liên kết kép gồm 2 con trỏ, một trỏ đến cấu trúc trước và một trỏ đến cấu trúc sau, 2 đầu của danh sách được đánh dấu bởi các con trỏ head, last.
- Danh sách liên kết vòng gồm 1 con trỏ trỏ về sau (hoặc trước), hai đầu của danh sách được nối với nhau tạo thành vòng tròn. Chỉ cần một con trỏ head để đánh dấu đầu danh sách.

Do trong cấu trúc có chứa các con trỏ trỏ đến cấu trúc tiếp theo và/hoặc cấu trúc đứng trước nên từ một cấu trúc này chúng ta có thể truy cập đến một cấu trúc khác

(trước và/hoặc sau nó). Kết hợp với các con trỏ đánh dấu 2 đầu danh sách (head, last) chúng ta sẽ dễ dàng làm việc với bất kỳ phần tử nào của danh sách. Có thể kể một số công việc thường thực hiện trên một danh sách như: bổ sung phần tử vào cuối danh sách, chèn thêm một phần tử mới, xoá một phần tử của danh sách, tìm kiếm, sắp xếp danh sách, in danh sách …

Hình vẽ bên dưới minh hoạ một danh sách liên kết đơn quản lý sinh viên, thông tin chứa trong mỗi phần tử của danh sách gồm có họ tên sinh viên, điểm. Ngoài ra mỗi phần tử còn chứa con trỏ tiep để nối với phần tử tiếp theo của nó. Phần tử cuối cùng nối với cấu trúc rỗng (NULL).



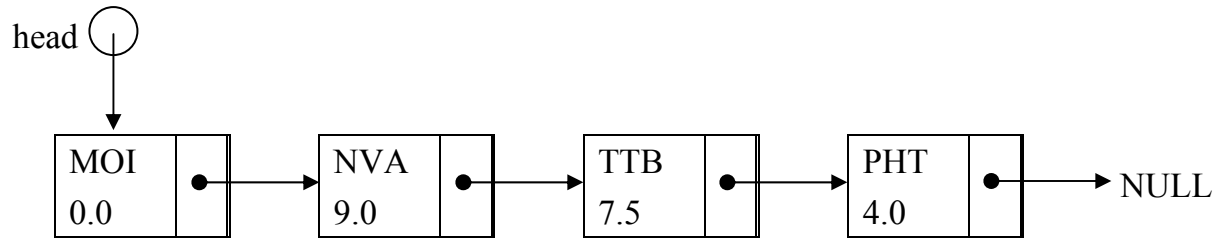
3. Các phép toán trên danh sách liên kết

Dưới đây chúng ta mô tả tóm tắt cách thức thực hiện một số thao tác trên danh sách liên kết đơn.

a. Tạo phần tử mới

Để tạo phần tử mới thông thường chúng ta thực hiện theo các bước sau đây:

- dùng toán tử new xin cấp phát một vùng nhớ đủ chứa một phần tử của danh sách.
- nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ tiep được đặt bằng NULL.
- gắn phần tử vừa tạo được vào danh sách. Có hai cách:
  - hoặc gắn vào đầu danh sách, khi đó vị trí của con trỏ head (chỉ vào đầu danh sách) được điều chỉnh lại để chỉ vào phần tử mới.
  - hoặc gắn vào cuối danh sách bằng cách cho con trỏ tiep của phần tử cuối danh sách (đang trỏ vào NULL) trỏ vào phần tử mới. Nếu danh sách có con trỏ last để chỉ vào cuối danh sách thì last được điều chỉnh để trỏ vào phần tử mới. Nếu danh sách không có con trỏ last thì để tìm được phần tử cuối chương trình phải duyệt từ đầu, bắt đầu từ con trỏ head cho đến khi gặp phần tử trỏ vào NULL, đó là phần tử cuối của danh sách.

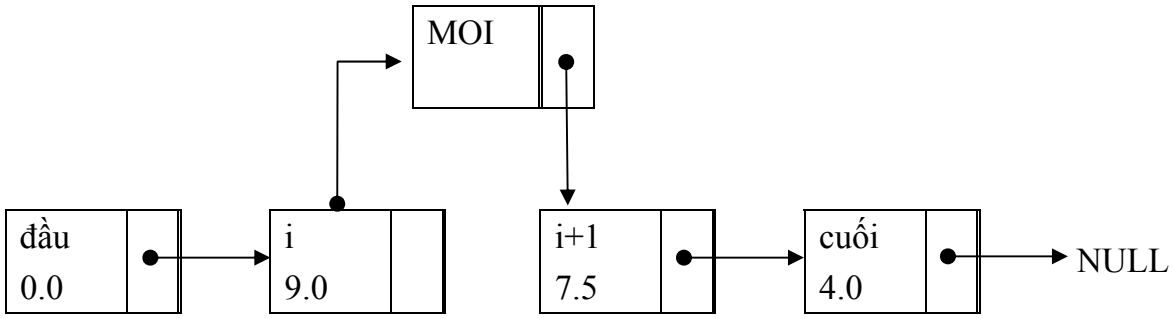


Gắn phần tử mới vào đầu danh sách

b. Chèn phần tử mới vào giữa

Giả sử phần tử mới được chèn vào giữa phần tử thứ i và i+1. Để chèn ta nối phần tử thứ i vào phần tử mới và phần tử mới nối vào phần tử thứ i+1. Thuật toán sẽ như sau:

- Cho con trỏ p chạy đến phần tử thứ i.
- Cho con trỏ tiep của phần tử mới trỏ vào phần tử thứ i+1 (tuc p->tiep).
- Cho con trỏ tiep của phần tử thứ i (hiện được trỏ bởi p) thay vì trỏ vào phần tử thứ i+1 bây giờ sẽ trỏ vào phần tử mới.



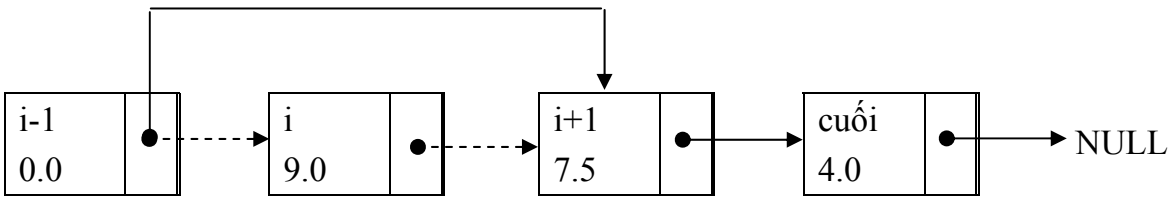
Chèn phần tử mới vào giữa phần tử i và i+1

a. Xoá phần tử thứ i khỏi danh sách

Việc xoá một phần tử ra khỏi danh sách rất đơn giản bởi chỉ việc thay đổi các con trỏ. Cụ thể giả sử cần xoá phần tử thứ i ta chỉ cần cho con trỏ tiep của phần tử thứ i-1 trỏ ("vòng qua" phần tử thứ i) vào phần tử thứ i+1. Như vậy bây giờ khi chạy trên danh sách đến phần tử thứ i-1, phần tử tiếp theo là phần tử thứ i+1 chứ không còn là phần tử thư i. Nói cách khác phần tử thứ i không được nối bởi bất kỳ phần tử nào nên nó sẽ

không thuộc danh sách. Có thể thực hiện các bước như sau:

- Cho con trỏ p chạy đến phần tử thứ i-1.
- Đặt phần tử thứ i vào biến x.
- Cho con trỏ tiep của phần tử thứ i-1 trỏ vào phần tử thứ i+1 bằng cách đặt tiep = x.tiep.
- Giải phóng bộ nhớ được trỏ bởi x bằng câu lệnh delete x.



Xóa phần tử thứ i

c. Duyệt danh sách

Duyệt là thao tác đi qua từng phần tử của danh sách, tại mỗi phần tử chương trình thực hiện một công việc gì đó trên phần tử mà ta gọi là thăm phần tử đó. Một phép thăm có thể đơn giản là hiện nội dung thông tin của phần tử đó ra màn hình chẳng hạn. Để duyệt danh sách ta chỉ cần cho một con trỏ p chạy từ đầu đến cuối danh sách đến khi phần tử cuối có con trỏ tiep = NULL thì dừng. Câu lệnh cho con trỏ p chuyển đến phần tử tiếp theo của nó là:

p = p → tiep ;

d. Tìm kiếm

Cho một danh sách trong đó mỗi phần tử của danh sách đều chứa một trường gọi là trường khoá, thường là các trường có kiểu cơ sở hoặc kết hợp của một số trường như vậy. Bài toán đặt ra là tìm trên danh sách phần tử có giá trị của trường khoá bằng với một giá trị cho trước. Tiến trình thực hiện nhiệm vụ thựcchất cũng là bài toán duyệt, trong đó thao tác "thăm" chính là so sánh trường khoá của phần tử với giá trị cho trước, nếu trùng nhau ta in kết quả và dừng, Nếu đã duyệt hết mà không có phần tử nào có trường khoá trùng với giá trị cho trước thì xem danh sách không chứa giá trị này.

Ngoài các thao tác trên, nói chung còn nhiều các thao tác quen thuộc khác tuy nhiên chúng ta không trình bày ở đây vì nó không thuộc phạm vi của giáo trình này.

Dưới đây là một ví dụ minh hoạ cho cấc cấu trúc tự trỏ, danh sách liên kết và một vài thao tác trên danh sách liên kết thông qua bài toán quản lý sinh viên.

- Khai báo

```
struct DATE
{
        int day, month, year;                   // ngày, tháng, năm
};
struct Sinhvien {                               // cấu trúc tự trỏ
        char hoten[31];
        DATE ns;
        float diem;
        Sinhvien *tiep ;
};
Sinhvien *dau = NULL, *cuoi = NULL;     // Các con trỏ tới đầu và cuối ds
Sinhvien *cur = NULL;                   // Con trỏ tới sv hiện tại
int sosv = 0;                           // Số sv của danh sách
```

- Tạo sinh viên mới và nhập thông tin, trả lại con trỏ trỏ đến sinh viên mới.

```
Sinhvien* Nhap1sv()                     // Tạo 1 khối dữ liệu cho sv mới
{
        Sinhvien *kq = new Sinhvien[1] ;        // Cấp phát bộ nhớ cho kq
        cout << "\nSinh vien thu ", sosv+1 ;
        cout << "Ho ten = " ; cin.getline(kq->hoten);
        cout << "Ns = " ; cin >> kq->ns.day >> kq->ns.month >> kq->ns.year;
        cout << "Diem = " ; cin >> kq->diem ; cin.ignore() ;
        kq->tiep = NULL;
        return kq ;
}
```

- Bổ sung sinh viên mới vào cuối danh sách.

```
void Bosung()                           // Bổ sung sv mới vào cuối ds
{
        cur = Nhap1sv();
```

```
        if (sosv == 0) {dau = cuoi = cur;}
        else { cuoi->tiep = cur; cuoi = cur; }
        sosv++;
    }
```

- Chèn sv mới vào trước sinh viên thứ n.

```
    void Chentruoc(int n)                    // Chèn sv mới vào trước sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0) { dau = cuoi = cur; sosv++; return; }
        if (sosv==1 || n==1) {cur->tiep = dau; dau = cur; sosv++; return;}
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
        cur -> tiep = sau;
        sosv ++;
    }
```

- Chèn sv mới vào sau sinh viên thứ n.

```
    void Chensau(int n)                      // Chèn sv mới vào sau sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0 || sosv<n) { dau = cuoi = cur; sosv++; return; }
        Sinhvien *truoc, *sau;
        truoc = dau; sau = dau -> tiep;
        for (int i=1; i<n; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
```

```
        cur -> tiep = sau;
        sosv ++;
    }
```

- Xoá sinh viên thứ n.

```
    void Xoa(int n)                                    // Xoá sinh viên thứ n
    {
        if (sosv==1&&n==1) { delete dau ; dau = cuoi = NULL; sosv--; return; }
        if (n==1) { cur = dau; dau = cur->tiep; delete cur; sosv--; return; }
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        cur = truoc->tiep; sau = cur->tiep; truoc->tiep = sau;
        delete cur ;
        sosv --;
    }
```

- Tạo danh sách sinh viên.

```
    void Taods()                                       // Tạo danh sách
    {
        int tiep = 1;
        while (tiep) {
            Bosung();
            cout << "Tiep (0/1) ? " ; cin >> tiep ;
        }
    }
```

- In danh sách sinh viên.

```
    void Inds()                                        // In danh sách
    {
```

```
        cur = dau;        int i=1;
        while (cur != NULL) {
            cout << "\nSinh vien thu " << i << " ----------\n") ;
            cout << "Hoten:" << cur->hoten ;
            cout << "Ngay sinh: "
            cout << cur -> ns.day << "/" ;
            cout << cur -> ns.month << "/" ;
            cout << cur -> ns.year ;
            cout << "Diem: " << cur->diem ;
            cur = cur->tiep; i++;
        }
    }
```

- Hàm chính.

```
    void main()
    {
        clrscr();
        Taods();
        Inds();
        getch();
    }
```

III. KIỂU HỢP

1. Khai báo

Giống như cấu trúc, kiểu hợp cũng có nhiều thành phần nhưng các thành phần của chúng sử dụng chung nhau một vùng nhớ. Do vậy kích thước của một kiểu hợp là độ dài của trường lớn nhất và việc thay đổi một thành phần sẽ ảnh hưởng đến tất cả các thành phần còn lại.

```
union <tên kiểu> {
    Danh sách các thành phần;
};
```

2. Truy cập

Cú pháp truy cập đến các thành phần của hợp cũng tương tự như kiểu cấu trúc, tức cũng sử dụng toán tử lấy thành phần (dấu chấm . hoặc → cho biến con trỏ kiểu hợp).

Dưới đây là một ví dụ minh hoạ việc sử dụng khai báo kiểu hợp để tách byte thấp, byte cao của một số nguyên.

Ví dụ 1 :

```
void main()
{
    union songuyen {
        int n;
        unsigned char c[2];
    } x;
    cout << "Nhập số nguyên: " ; cin >> x.n ;
    cout << "Byte thấp của x = " << x.c[0] << endl ;
    cout << "Byte cao của x = " << x.c[1] << endl;
}
```

Ví dụ 2 : Kết hợp cùng kiểu nhóm bit trong cấu trúc, chúng ta có thể tìm được các bit của một số như chương trình sau. Trong chương trình ta sử dụng một biến u có kiểu hợp. Trong kiểu hợp này có 2 thành phần là 2 cấu trúc lần lượt có tên s và f.

```
union {
    struct { unsigned a, b ; } s;
    struct {
        unsigned n1: 1;
        unsigned: 15;
        unsigned n2: 1;
        unsigned: 7;
        unsigned n3: 8;
    } t ;
} u;
```

với khai báo trên đây khi nhập u.s thì nó cũng ảnh hưởng đến u.t, cụ thể

- u.t.n1 là bit đầu tiên (0) của thành phần u.s.a
- u.t.n2 là bit 0 của thành phần u.s.b
- u.t.n3 là byte cao của u.s.b

IV. KIỂU LIỆT KÊ

Có thể gán các giá trị nguyên liên tiếp (tính từ 0) cho các tên gọi cụ thể bằng kiểu liệt kê theo khai báo sau đây:

enum tên_kiểu { d/s tên các giá trị };

Ví dụ:

```
enum Bool {false, true};
```

khai báo kiểu mới đặt tên Bool chỉ nhận 1 trong 2 giá trị đặt tên false và true, trong đó false ứng với giá trị 0 và true ứng với giá trị 1. Cách khai báo kiểu enum trên cũng tương đương với dãy các macro sau:

```
#define false 0
#define true 1
```

Với kiểu Bool ta có thể khai báo một số biến như sau:

```
Bool Ok, found;
```

hai biến Ok và found sẽ chỉ nhận 1 trong 2 giá trị false (thay cho 0) hoặc true (thay cho 1). Có nghĩa có thể gán:

```
Ok = true;
```

hoặc: `found = false;`

Tuy nhiên không thể gán các giá trị nguyên trực tiếp cho các biến enum mà phải thông qua ép kiểu. Ví dụ:

```
Ok = 0;              // sai
Ok = Bool(0) ;       // đúng
hoặc Ok = false ;    // đúng
```

BÀI TẬP

1. Có thể truy nhập thành phần của cấu trúc thông qua con trỏ như sau (với p là con trỏ cấu trúc và a là thành phần của cấu trúc):

A: (*p).a B: *p→a C: a và b sai D: a và b đúng

2. Cho khai báo struct T {int x; float y;} t, *p, a[10]; Câu lệnh nào trong các câu sau là không hợp lệ:

(1) p = &t; (2) p = &t.x; (3) p = a;

(4) p = &a (5) p = &a[5]; (6) p = &a[5].y;

A: 1, 2 và 3 B: 4, 5 và 6 C: 1, 3 và 5 D: 2, 4 và 6

3. Cho các khai báo sau:

```
struct ngay {int ng, th, nam;} vaotruong, ratruong;
typedef struct {char hoten[25]; ngay ngaysinh;} sinhvien;
```

Hãy chọn câu đúng nhất

A: Không được phép gán: ratruong = vaotruong;

B: sinhvien là tên cấu trúc, vaotruong, ratruong là biến cấu trúc

C: Có thể viết: vaotruong.ng, ratruong.th, sinhvien.vaotruong.nam để truy nhập đến các thành phần tương ứng.

D: a, b, c đúng

4. Trong các khởi tạo giá trị cho các cấu trúc sau, khởi tạo nào đúng:

```
struct S1 {
        int ngay, thang, nam;
} s1 = {2,3};
struct S2 {
        char hoten[10];
        struct S1 ngaysinh;
} s2 = {"Ly Ly",1,2,3};
struct S3 {
        struct S2 sinhvien;
        float diem;
} s3 = {{{"Cốc cốc", {4,5,6}}, 7};
```

A: S1 và S2 đúng B: S2 và S3 đúng C: S3 và S1 đúng D: Cả 3 cùng đúng

5. Đối với kiểu cấu trúc, cách gán nào dưới đây là không được phép:

A: Gán hai biến cho nhau.

B: Gán hai phần tử mảng (kiểu cấu trúc) cho nhau

C: Gán một phần tử mảng (kiểu cấu trúc) cho một biến và ngược lại

D: Gán hai mảng cấu trúc cùng số phần tử cho nhau

6. Cho đoạn chương trình sau:

```
struct {
    int to ;
    float soluong;
} x[10];
for (int i = 0; i < 10; i++) cin >> x[i].to >> x[i].soluong ;
```

Chọn câu đúng nhất trong các câu sau:

A: Đoạn chương trình trên có lỗi cú pháp

B: Không được phép sử dụng toán tử lấy địa chỉ đối với các thành phần to và soluong

C: Lấy địa chỉ thành phần soluong dẫn đến chương trình hoạt động không đúng đắn

D: Cả a, b, c đều sai

7. Chọn câu đúng nhất trong các câu sau:

A: Các thành phần của kiểu hợp (union) được cấp phát một vùng nhớ chung

B: Kích thước của kiểu hợp bằng kích thước của thành phần lớn nhất

C: Một biến kiểu hợp có thể được tổ chức để cho phép thay đổi được kiểu dữ liệu của biến trong qua trình chạy chương trình

D: a, b, c đúng

8. Cho khai báo:

```
union {
    unsigned x;
    unsigned char y[2];
} z = {0xabcd};
```

Chọn câu đúng nhất trong các câu sau:

A: Khai báo trên là sai vì thiếu tên kiểu

B: Khởi tạo biến z là sai vì chỉ có một giá trị (0xabcd)

C: z.y[0] = 0xab

D: z.y[1] = 0xab

9. Cho kiểu hợp:

```
union U {
    char x[1];
    int y[2]; float z[3];
} u;
```

Chọn câu đúng nhất trong các câu sau:

A: sizeof(U) = 1+2+3 = 6

B: sizeof(U) = max(sizeof(char), sizeof(int), sizeof(float))

C: sizeof(u) = max(sizeof(u.x), sizeof(u.y), sizeof(u.z))

D: b và c đúng

10. Cho khai báo:

```
union {
    unsigned x;
    struct {
        unsigned char a, b;
    } y;
} z = {0xabcd};
```

Giá trị của z.y.a và z.y.b tương ứng:

A: 0xab, 0xcd B: 0xcd, 0xab C: 0xabcd, 0 D: 0, 0xabcd

11. Cho khai báo:

```
union {
    struct {
        unsigned char a, b;
    } y;
    unsigned x;
```

```
} z = {{1,2}};
```

Giá trị của z.x bằng:

A: 513 B: 258

C: Không xác định vì khởi tạo sai D: Khởi tạo đúng nhưng z.x chưa có giá trị

12. Xét đoạn lệnh:

```
union U {
    int x; char y;
} u;
u.x = 0; u.y = 200;
```

Tìm giá trị của u.x + u.y ?

A: 122 B: 144 C: 200 D: 400

13. Cho số phức dưới dạng cấu trúc gồm 2 thành phần là thực và ảo. Viết chương trình nhập 2 số phức và in ra tổng, tích, hiệu, thương của chúng.

14. Cho phân số dưới dạng cấu trúc gồm 2 thành phần là tử và mẫu. Viết chương trình nhập 2 phân số, in ra tổng, tích, hiệu, thương của chúng dưới dạng tối giản.

15. Tính số ngày đã qua kể từ đầu năm cho đến ngày hiện tại. Qui ước ngày được khai báo dưới dạng cấu trúc và để đơn giản một năm bất kỳ được tính 365 ngày và tháng bất kỳ có 30 ngày.

16. Nhập một ngày tháng năm dưới dạng cấu trúc. Tính chính xác (kể cả năm nhuận) số ngày đã qua kể từ ngày 1/1/1 cho đến ngày đó.

17. Tính khoảng cách giữa 2 ngày tháng bất kỳ.

18. Hiện thứ của một ngày bất kỳ nào đó, biết rằng ngày 1/1/1 là thứ hai.

19. Hiện thứ của một ngày bất kỳ nào đó, lấy ngày thứ hiện tại để làm chuẩn.

20. Viết chương trình nhập một mảng sinh viên, thông tin về mỗi sinh viên gồm họ tên và ngày sinh (kiểu cấu trúc). Sắp xếp mảng theo tuổi và in ra màn hình

21. Để biểu diễn số phức có thể sử dụng định nghĩa sau:

```
typedef struct {  float re, im; } sophuc;
```

Cần bổ sung thêm trường nào vào cấu trúc để có thể lập được một danh sách liên kết các số phức.

22. Để tạo danh sách liên kết, theo bạn sinh viên nào dưới đây khai báo đúng cấu trúc tự trỏ sẽ được dùng:

Sinh viên 1: struct SV {char ht[25]; int tuoi; struct SV *tiep;};

Sinh viên 2: typedef struct SV node; struct SV {char ht[25]; int tuoi; node *tiep;};

Sinh viên 3: typedef struct SV {char ht[25]; int tuoi; struct SV *tiep;} node;

A: Sinh viên 1 B: Sinh viên 2 C: Sinh viên 2 và 3 D: Sinh viên 1, 2 và 3

23. Lập danh sách liên kết chứa bảng chữ cái A, B, C … Hãy đảo phần đầu từ A .. M xuống cuối thành N, O, … Z, A, …M.

24. Viết chương trình tìm người cuối cùng trong trò chơi: 30 người xếp vòng tròn. Đếm vòng tròn (bắt đầu từ người số 1) cứ đến người thứ 7 thì người này bị loại ra khỏi vòng. Hỏi người còn lại cuối cùng ?

25. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên. Viết chương trình sắp xếp danh sách theo thứ tự giảm dần.

26. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên được sắp giảm dần. Viết chương trình cho phép chèn thêm một phần tử vào danh sách sao cho danh sách vẫn được sắp giảm dần.

27. Tạo danh sách liên kết các số thực x_1, x_2, ..., xn. Gọi m là trung bình cộng: $m = \frac{x_1 + x_2 + ... + x_n}{n}$.

Hãy in lần lượt ra màn hình các giá trị: m, x_1 - m, x_2 - m, ..., xn - m.

28. Sử dụng kiểu union để in ra byte thấp, byte cao của một số nguyên.

CHƯƠNG 6

ĐỒ HOẠ VÀ ÂM THANH

Đồ hoạ
Âm thanh

I. ĐỒ HOẠ

1. Khái niệm đồ hoạ

a. Điểm ảnh và độ phân giải

Màn hình ở chế độ đồ hoạ là tập hợp các điểm (pixel-picture elements) ảnh. Số điểm ảnh và cách bố trí theo chiều ngang, dọc của màn hình được gọi là độ phân giải (resolution). Vì vậy độ phân giải thường được đặc trưng bởi một cặp số chỉ định số điểm ảnh theo chiều ngang và chiều dọc của màn hình. Ví dụ màn hình VGA ở mode 2 có độ phân giải là 640x480, tức trên mỗi dòng ngang của màn hình có thể vẽ được 640 điểm ảnh và trên mỗi cột dọc vẽ được 480 điểm ảnh. Các cột và dòng được đánh số từ 0, theo chiều từ trái sang phải (đối với cột) và từ trên xuống dưới (đối với dòng). Một điểm ảnh hay còn gọi là pixel là giao điểm của một cột và một dòng nào đó trên màn hình và vị trí của nó được thể hiện bởi cặp toạ độ (x,y) với x biểu diễn cho cột và y biểu diễn cho dòng. Ví dụ với màn hình trên điểm ảnh "đầu tiên" nằm ở góc trên bên trái của màn hình có toạ độ (0,0) và điểm "cuối cùng" ở góc dưới bên phải có toạ độ (639,479). Điểm có toạ độ (150,200) là giao điểm của cột thứ 150 và dòng 200.

b. Trình điều khiển đồ hoạ

Màn hình đồ hoạ có nhiều loại khác nhau. Mỗi loại màn hình cần có trình điều khiển tương ứng. C cung cấp các trình điều khiển màn hình trong thư mục BGI đặt dưới thư mục gốc của C (TC hoặc BC) gồm có:

Tên trình điều khiển	**Kiểu màn hình đồ hoạ**
ATT.BGI	ATT & T6300 (400 dòng)
CGA.BGI	IBMCGA, MCGA và các máy tương thích
EGAVGA.BGI	IBM EGA, VGA và các máy tương thích

HERC.BGI	Hercules mono và các máy tương thích
IBM8514.BGI	IBM 8514 và các máy tương thích
PC3270.BGI	IBM 3270 PC

Ngoài các trình điều khiển trong thư mục BGI còn chứa các file font chữ có đuôi CHR gồm:

GOTH.CHR

LITT.CHR

SANS.CHR

TRIP.CHR

c. Mốt (mode) đồ hoạ

Mỗi màn hình đồ hoạ có thể hoạt động dưới nhiều mốt khác nhau. Độ phân giải của màn hình phụ thuộc vào từng mốt. Ví dụ màn hình VGA có thể hoạt động dưới các mốt 0 (VGALO: độ phân giải thấp 640x200), 1 (VGAMED: độ phân giải trung bình 640x350), 2 (VGAHI: độ phân giải cao 640x480).

2. Vào/ra chế độ đồ hoạ

Trong C++ các hàm liên quan đến đồ hoạ được khai báo trong tệp <graphics.h>

a. Khởi động chế độ đồ hoạ

void initgraph(int *graphdriver, int *graphmode, char *drivepath)

- drivepath: đường dẫn của thư mục chứa các trình điều khiển đồ hoạ. Nếu rỗng sẽ tìm trong thư mục hiện tại.
- graphdriver, graphmode: Chỉ định trình quản lý và mốt màn hình cần sử dụng. Trong đó graphdriver có thể nhận 1 trong các giá trị sau:

DETECT	0
CGA	1
EGA	3
EGA64	4
EGAMONO	5
VGA	9
.....	..

Hiển nhiên việc chọn giá trị của graphdriver phải tương ứng với màn hình thực tế. Trong trường hợp ta không biết chủng loại thực tế của màn hình có thể sử dụng giá trị DETECT (hoặc 0) là giá trị chỉ định cho chương trình tự tìm hiểu về màn hình và gọi trình điều khiển tương ứng. Trong trường hợp này graphmode sẽ được gán giá trị tự động với mode có độ phân giải cao nhất có thể. Về graphmode có thể nhận các giá trị sau:

CGAC0	0	320 x 200	
CGAC1	1	320 x 200	
CGAC2	2	320 x 200	
CGAC3	3	320 x 200	
CGAHI	4	640 x 200	2 color
EGALO	0	640 x 200	16 color
EGAHI	1	640 x 350	16 color
EGA64LO	0	640 x 200	16 color
EGA64HI	1	640 x 350	4 color
VGALO	0	640 x 200	16 color
VGAMED	0	640 x 350	16 color
VGAHI	0	640 x 480	16 color

Trong quá trình sử dụng để xoá màn hình đồ hoạ ta dùng hàm **cleardevice();**

b. Kết thúc chế độ đồ hoạ

Để kết thúc chế độ đồ hoạ về lại chế độ văn bản ta sử dụng hàm **closegraph();**

c. Lỗi đồ hoạ

- Sau mỗi thao tác đồ hoạ, hàm **graphresult()** sẽ cho giá trị 0 nếu không có lỗi, hoặc các giá trị âm (-1 .. -18) tương ứng với lỗi. Hàm **grapherrormsg(n)** trả lại nội dung lỗi và mã lỗi.

Mã lỗi	Hằng lỗi (graphresult())	Nội dung lỗi (grapherrormsg())
0	grOk	No error
-1	grNoInitGraph	(BGI) Không có BGI
-2	grNotDetected	Graphics hardware not detected

-3	grFileNotFound	Device driver file not found

.....

Ví dụ 1 :

Ví dụ sau đây khởi tạo chế độ đồ hoạ với graphdriver = 0 (DETECT) và thông báo lỗi nếu không thành công hoặc thông báo chế độ đồ hoạ cũng như mode màn hình. Để biết độ phân giải của màn hình có thể dùng các hàm **getmaxx()** (số cột) và **getmaxy()** (số dòng)

```
void main()
{
    int gd = DETECT, gm, maloi;
    initgraph(&gd, &gm, "C:\\BC\\BGI");
    maloi = graphresult();
    if (maloi != grOk)
    {
        cout << "Lỗi: " << grapherrormsg(maloi)) << endl;
        cout << "An phím bất kỳ để dừng "; getch();
        exit(1);
    } else {
        cout << "Chế độ màn hình = " << gd << endl;
        cout << "Mode màn hình = " << gm << endl;
        cout << "Độ phân giải: " << getmaxx() << "," << getmaxy() << endl;
        getch();
    }
    closegraph();
}
```

Các phần tiếp theo sau đây sẽ cung cấp các câu lệnh để vẽ trong chế độ đồ họa.

3. Vẽ điểm, đường, khối, màu sắc

a. Màu sắc

- **getmaxcolor()**: Trả lại số hiệu (hằng) tương ứng với màu tối đa của màn hình hiện tại. Do các hằng màu được tính từ 0 nên số màu sẽ bằng hằng trả lại cộng

thêm 1.

- **setbkcolor(màu):** Đặt màu nền. Có tác dụng với văn bản và các nét vẽ.
- **setcolor(màu):** Đặt màu vẽ. Có tác dụng với văn bản và các nét vẽ.
- **getbkcolor():** Trả lại màu nền hiện tại.
- **getcolor():** Trả lại màu vẽ hiện tại (từ 0 đến getmaxcolor()).

Ví dụ: In toạ độ tại vị trí hiện tại của con trỏ màn hình. Trong ví dụ này chúng ta sử dụng câu lệnh sprintf(xâu s, "dòng đk", các biểu thức cần in), câu lệnh này sẽ thay việc in các biểu thức ra màn hình thành in ra xâu s (tức tạo xâu s từ các biểu thức). Ngoài ra hàm outtextxy(x, y, s) sẽ in xâu s tại vị trí (x,y).

```
void intoado(dx, dy, color)
{
    int oldcolor;
    oldcolor = getcolor();
    setcolor(color);
    char td[10];
    sprintf(td, "(%d, %d)", getx(), gety());
    outtextxy(getx()+dx, gety()+dy, td);
    setcolor(oldcolor);
}
```

b. Vẽ điểm

- **putpixel(x, y, c)**: Vẽ điểm (x, y) với màu c.
- **getpixel (x, y)**: Trả lại màu tại điểm (x, y).

Ví dụ 2 : Vẽ bầu trời sao

```
void sky()
{
    int maxx, maxy, maxc;
    int i, xarr[3001], yarr[3001];
    maxx = getmaxx();
    maxy = getmaxy();
    maxc = getmaxcolor();
```

```
        randomize();
        for (i=1;i<3001;i++) {xarr[i]=random(maxx);yarr[i]=random(maxy);}
        while (!kbhit()) {
            for (i=1;i<3001;i++)
            {
                putpixel(xarr[i], yarr[i], random(maxc));delay(1);
            }
        for (i=1;i<3001;i++)
        if (getpitxel(xarr[i], yarr[i]) == random(maxc))
            putpitxel(xarr[i], yarr[i], 0);
        }
    }
```

c. Vẽ đường thẳng và gấp khúc

- **line(x1, y1, x2, y2):** Vẽ đường thẳng từ (x1, y1) đến (x2, y2). Con trỏ màn hình vẫn đứng tại vị trí cũ.
- **lineto(x, y):** Vẽ đường thẳng từ vị trí hiện tại của con trỏ đến vị trí (x, y). con trỏ chuyển về (x, y).
- **linerel(dx, dy):** Gọi (x, y) là vị trí hiện tại của con trỏ, lệnh này sẽ vẽ đường thẳng nối (x, y) với điểm mới có tọa độ (x+dx, y+dy). Tức lệnh này cũng tương đương với lệnh lineto(getx()+dx, gety()+dy), trong đó getx() và gety() là hai hàm trả lại vị trí x, y hiện tại của con trỏ. Lệnh linerel sau khi thực hiện xong sẽ đặt con trỏ tại vị trí cuối của đường thẳng vừa vẽ.

Ví dụ 3 : Vẽ hình bao thư bằng 1 nét.

```
void baothu()
{
    setbkcolor(1);
    setcolor(YELLOW);
    moveto(100, 100);
    lineto(300, 100); lineto(300, 200); lineto(100, 200); lineto(100, 100);
    lineto(200, 50); lineto(300, 100);
}
```

- **rectangle(x1, y1, x2, y2)**: Vẽ hình khung chữ nhật với góc trên bên trái có tọa độ (x1, y1) và góc dưới bên phải có tọa độ (x2, y2).
- **bar(x1, y1, x2, y2):** Vẽ hình chữ nhật đặc. Màu khung được đặt bởi *setcolor* và màu nền lẫn mẫu tô nền được đặt bởi lệnh *setlinestyle*. Mẫu nền ngầm định là đặc và màu là getmaxcolor.
- **bar3d(x1, y1, x2, y2, c, top):** Vẽ hình trụ chữ nhật với đáy là (x1, y1, x2, y2) và độ cao c, nếu top = 1 hình sẽ có nắp và nếu top = 0 hình không có nắp.

Ví dụ : Vẽ các hình khối chữ nhật với mầu nền và mẫu tô khác nhau.

```
void main()
{
        int gdriver = DETECT, gmode;
        initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
        int midx = getmaxx() / 2;
        int midy = getmaxy() / 2;
        for (int i=SOLID_FILL; i<USER_FILL; i++)
        {
                setfillstyle(i, i);
                bar3d(midx-50, midy-50, midx+50, midy+50, 100, 0);
                getch();
        }
        closegraph();
}
```

Ghi chú: để xoá điểm hoặc đường ta vẽ lại điểm hoặc đường đó bằng màu nền hiện tại. Để biết màu nền hiện tại ta sử dụng hàm getbkcolor().

d. Các thuộc tính về đường (kiểu đường, độ rộng)

- **setlinestyle(style, pattern, width)**: đặt các thuộc tính về đường vẽ, trong đó style là kiểu đường, pattern là mẫu tô và width là độ đậm của đường vẽ. Các thuộc tính này được giải thích bên dưới.
- **getlinesettings(struct linesettingstype *info)**: Lấy các thuộc tính về đường vẽ hiện tại cho vào biến được trỏ bởi info.
- Kiểu của biến chứa các thuộc tính đường vẽ:

```
struct linesettingstype {
    int linetsyle;
    int upattern;
    int thickness;
}
```

- Các hằng số qui định các kiểu đường (style):

```
style:    SOLID_LINE = 0
          DOTTED_LINE = 1
          CENTER_LINE = 2
          DASHED_LINE = 3
          USERBIT_LINE = 4,        // Kiểu đường do NSD định nghĩa
```

- pattern: Do NSD định nghĩa theo 2 byte cho một đường. Chỉ có tác dụng khi style = 4.
- Các hằng số qui định độ đậm (độ dày) của đường (width):

```
          NORM_WIDTH = 1
          THICK_WIDTH = 3
```

Ví dụ 4 :

```
void netve()
{
    char *lname[] = {"Duong lien net", "Duong cham cham",
    "Duong trung tam", "Duong dut net", "Duong do NSD dinh nghia" };
    int style, midx, midy, mauNSD;
    midx = getmaxx() / 2; midy = getmaxy() / 2;
    // Mẫu đường được định nghĩa bởi NSD "0000000000000001"
    mauNSD = 1;
    for (style=SOLID_LINE; style<=USERBIT_LINE; style++) {
        setlinestyle(style, mauNSD, 1);
        line(0, 0, midx-10, midy);
        rectangle(0, 0, getmaxx(), getmaxy());
        outtextxy(midx, midy, lname[style]);
```

```
            line(midx, midy+10, midx+8*strlen(lname[style]), midy+10);
            getch();
            cleardevice();
        }
    }
```

e. Các thuộc tính về hình (mẫu tô, màu tô)

- setfillstyle(mẫu tô, màu tô): Đặt mẫu tô, màu tô
- setfillpattern(mẫu tô, màu tô): Định nghĩa mẫu tô.
- getfillsettings(struct fillsettingstype *info): Lấy mẫu tô hiện tại

```
struct fillsettingstype {
        int pattern;
        int color;
};
```

- **getfillpattern(mẫu tô)**: Trả lại mẫu tô hiện do NSD định nghĩa. Là một con trỏ trỏ đến mảng 8 kí tự. Sau đây là một số mẫu tô và các hằng tương ứng

EMPTY_FILL	0
SOLID_FILL	1
LINE_FILL	2
LTSLASH_FILL	3
SLASH_FILL	4
BKSLASH_FILL	5
LTBKSLASH_FILL	6
HATCH_FILL	7
XHATCH_FILL	8
INTERLEAVE_FILL	9
WIDE_DOT_FILL	10
CLOSE_DOT_FILL	11
USER_FILL	12

Ví dụ 5 : Đặt mẫu tô.

```
char caro[8] = {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};
```

```
maxx = getmaxx();
maxy = getmaxy();
setfillpattern(caro, getmaxcolor());
// Tô màn hình theo mẫu
bar(0, 0, maxx, maxy);
getch();
```

f. Vẽ đa giác

- **drawpoly(số đỉnh, vị trí đỉnh)**: Vẽ đường đa giác theo setlinestyle;
- **fillpoly(số đỉnh, vị trí đỉnh)**: Vẽ hình đa giác đặc theo setfillstyle;

  Vị trí đỉnh là con trỏ trỏ đến dãy các toạ độ, thông thường dùng mảng.

  Để vẽ đa giác đóng phải đưa ra n+1 toạ độ trong đó toạ độ n = toạ độ 0.

Ví dụ 6 :

```
int poly[10];
poly[0] = 20; poly[1] = maxy / 2;              // đỉnh thứ nhất
poly[2] = maxx - 20; poly[3] = 20;             // đỉnh thứ hai
poly[4] = maxx - 50; poly[5] = maxy - 20;      // đỉnh thứ ba
poly[6] = maxx / 2; poly[7] = maxy / 2;        // đỉnh thứ tư
poly[8] = poly[0]; poly[9] = poly[1];
// vẽ đa giác
drawpoly(5, poly);
```

g. Vẽ đường cong

- **arc(x, y, góc đầu, góc cuối, bán kính):** Vẽ cung tròn có tâm (x, y) với các góc và bán kính tương ứng.
- **circle(x, y, bán kính):** Vẽ đường tròn có tâm tại (x, y).
- **pieslice(x, y, góc đầu, góc cuối, bán kính):** Vẽ hình quạt tròn đặc với mẫu hiện tại;
- **ellipse(x, y, góc đầu, góc cuối, bkx, bky):** Vẽ cung elip với tâm, các góc và các bán kính theo hoàng độ và tung độ tương ứng.
- **fillellipse(x, y, bkx, bky):** Vẽ hình elip đặc.
- sector(x, y, góc đầu, góc cuối, bkx, bky): Vẽ hình quạt elip.

Chú ý: Nếu góc đầu = 0 và góc cuối = 360 cung, lệnh trên sẽ vẽ đường tròn hoặc elip.

Ví dụ 7 : Vẽ đường tròn và elip.

```
arc(200, 200, 45, 135, 100) ;          // cung tròn
arc(200, 200, 0, 360, 100) ;           // đường tròn
circle(200, 200, 100) ;                // đường tròn
ellipse(200, 200, 45, 135, 100, 80) ;  // cung elip
ellipse(200, 200, 0, 360, 100, 80) ;   // đường elip;
setfillstyle(EMPTY_FILL, getmaxcolor());
pieslice(200, 200, 45, 135, 100) ;     // đường quạt tròn
fillellipse(200, 200, 0, 360, 100, 80) ;  // đường elip
setfillstyle(SOLID_FILL, getmaxcolor());
pieslice(200, 200, 45, 135, 100);      // hình quạt tròn;
circle(200, 200, 100);                 // hình tròn;
fillellipse(200, 200, 0, 360, 100, 80);  //  hình elip;
sector(200, 200, 45, 135, 100, 80);    // hình quạt elip
```

h. Tô mầu

- **floodfill(x, y, c)**: Tô màu một hình kín chứa điểm x, y và màu viền c. Mầu dùng để tô được đặt bởi hàm setfillstyle(kiểu tô, màu tô). Ví dụ:

```
void fill()
{
    rectangle(100, 100, 180, 140);          // Vẽ hình chữ nhật
    setfillstyle(1, BLUE);                  // Mẫu tô đặc, màu xanh
    floodfill(120, 120, 15);                // Tô hình chữ nhật đã vẽ
    int tg[8] = {150, 120, 180, 280, 350, 180, 150, 120};
    drawpoly(4, tg);
    setfillstyle(2, RED);
    floodfill(180, 200, 15);
    circle(380, 210, 100);
    setfillstyle(3, GREEN);
    floodfill(380, 210, 15);
```

```
}

void fill2()                          // Vẽ và tô màu dãy đường tròn liên tiếp
{
      int i, x = 0, y = 0, r = 0;
      for (i=1;i<10;i++) {
            r = 10*i;
            y = x += r;
            circle(x, y, r);
            setfillstyle(i, i);
            floodfill(x, y, 15);
      }
}
```

4. Viết văn bản trong màn hình đồ họa

a. Viết văn bản

outtext(s) ;

outtextxy(x, y, s) ;

Câu lệnh trên cho phép viết xâu kí tự tại vị trí con trỏ trên màn hình đồ họa. Câu lệnh tiếp theo cho phép viết s ra tại vị trí (x, y). Vị trí con trỏ sau khi thực hiện outtext(s) sẽ đặt tại vị trí cuối của xâu được in trong khi vị trí con trỏ sau khi thực hiện lệnh outtextxy(x, y, s) là không thay đổi. Ví dụ sau in ra màn hình đồ họa dòng chữ "Đây là chương trình minh họa lệnh outtext(s)" tại vị trí (100, 20):

```
moveto(100, 20) ;                     // chuyen con tro den cot 100, dong 20
outtext("Đây là chương trình minh họa lệnh outtext(s)") ;  hoặc
outtext("Đây là chương trình ") ;
outtext("minh họa lệnh ") ;
outtext("outtext(s)") ;
```

hoặc dòng văn bản trên cũng có thể được in bởi lệnh outtextxy(x, y, s);

```
outtextxy(100, 20, "Đây là chương trình minh họa lệnh outtextxy(x, y, s)");
```

b. Điều chỉnh font, hướng và cỡ chữ

settextstyle(Font, Hướng, Cỡ chữ);

a. Font : Gồm các loại font tương ứng với các hằng sau đây:

DEFAULT_FONT	0
SMALL_FONT	1
TRIPLEX_FONT	2
SANS_SERIF_FONT	3
GOTHIC_FONT	4

- Hướng : hướng viết theo kiểu nằm ngang hay thẳng đứng, tương ứng với các hằng:

HOIRIZ_DIR	0
VERT_DIR	1

- Cỡ chữ : Gồm các cỡ chữ đánh số tăng dần từ 1. Cỡ chữ ngầm định là 1.

Ví dụ sau lần lượt in tại tâm màn hình tên của các font với các cỡ chữ lớn dần, theo hướng nằm ngang.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main()
{
    char *fname[] = {"ngầm định", "Triplex", "Small", "Sans Serif", "Gothic" };
    int gdriver = DETECT, gmode;
    int font, midx, midy;
    int size = 1;
    initgraph(&gdriver, &gmode, "C:\\Borlandc\\BGI");
    midx = getmaxx() / 2; midy = getmaxy() / 2;
    for (font = DEFAULT_FONT; font <= GOTHIC_FONT; font++)
    {
        cleardevice();
        size = font;
        settextstyle(font, HORIZ_DIR, size);
        outtextxy(midx, midy, fname[font]);
```

```
        getch();
    }
    closegraph();
}
```

c. Điều chỉnh cách viết

Theo mỗi hướng (nằm ngang hay thẳng đứng) có 3 cách viết tương ứng với các hằng số sau:

1. Theo hướng nằm ngang:

   LEFT_TEXT = 0 : Viết từ trái sang phải.

   CENTER_TEXT = 1 : Viết từ vị trí con trỏ sang hai bên.

   RIGHT_TEXE = 2 : Viết từ phải sang trái.

2. Theo hướng thẳng đứng:

   BOTTOM_TEXT = 0 : Viết từ dưới lên.

   CENTER_TEXT = 1 : Viết từ vị trí con trỏ lên trên và xuống dưới.

   TOP_TEXT = 2. Viết từ trên xuống.

Để chỉ định một trong các cách viết trên ta dùng lệnh

settextjustify(Theo hướng ngang, Theo hướng dọc);

5. Chuyển động

Nguyên tắc: xóa hình ở vị trí cũ rồi vẽ lại hình đó tại vị trí mới theo hướng chuyển động. Để xoá, ta vẽ lại hình ngay tại vị trí cũ nhưng với mầu vẽ trùng với màu nền (do đó hình vẽ bị chìm vào nền giống như đã bị xóa). Để biết màu nền hiện tại có thể dùng hàm setcolor(getbkcolor()). Tóm lại có thể đưa ra sơ đồ như sau:

– vẽ lại hình với màu nền tại vị trí cũ // xóa hình

– delay // tạm dừng

– vẽ lại hình (với màu của hình) tại vị trí mới // hình chuyển đến vị trí khác

Các bước trên nếu được lặp đi lặp lại ta sẽ thấy hình chuyển động từ vị trí này đến vị trí khác.

Đối với các hình vẽ phức tạp, để xóa nhanh ta có thể vẽ lại hình trong chế độ XOR_PUT như được trình bày trong phần sau.

Chúng ta hãy xem qua một số hàm phức tạp hơn để vẽ hình.

- **setviewport(x1, y1, x2, y2, clip)**: Tạo một cửa sổ mới trong chế độ đồ hoạ. Khi đó tọa độ của các điểm sẽ được tính lại theo cửa sổ mới này. Cụ thể điểm (x1, y1) của màn hình bây giờ sẽ lại được tính với tọa độ mới là (0,0). Nếu clip = 0 sẽ cho phép các hình vẽ được mở rộng khỏi khung cửa sổ, nếu clip = 1 các phần của hình vẽ nằm ngoài khung cửa sổ sẽ bị cắt.
- **getviewsettings(struct viewporttype *vp)**: Lấy toạ độ cửa sổ hiện tại vào biến con trỏ vp. Kiểu của cuả sổ là một cấu trúc như sau:

  struct viewporttype {int left, top, right, bottom, clip;};
- **imagesize(x1, y1, x2, y2)**: Cho lại kích thước (byte) của một ảnh bitmap trong khung chữ nhật được xác định bởi các tọa độ (x1, y1, x2, y2).
- **getimage(x1, y1, x2, y2, *pict)**: Lưu ảnh từ màn hình vào vùng bộ nhớ được trỏ bởi con trỏ pict.
- **putimage(x1, y1, *pict, op)**: Ghi ra màn hình ảnh đã được lưu tại vị trí con trỏ pict. op là chế độ qui định việc hiện ảnh lên màn hình, màu của các điểm sẽ được qui định thông qua màu của ảnh được lưu trong pict và màu hiện tại của điểm trên màn hình. Hai màu này sẽ "trộn" theo các phép toán qui định bởi op dưới đây để cho ra màu vẽ của ảnh:

COPY_PUT = 0	Săn cầu thủ
XOR_PUT = 1	Hoặc loại trừ (giống nhau thì bằng 0). Để xóa ảnh ta có thể vẽ lại chúng với chế độ này.
OR_PUT = 2	Hoặc
AND_PUT = 3	Và
NOT_PUT = 4	Not

Ví dụ 8 : Vẽ bánh xe xoay

```
void bx(int x, int y, int r, float phi, int xoa)        // xoá ảnh nếu xoa = 1
{
    int i, x1, x2, y1, y2;
    if (xoa) setcolor(BLACK);                           // đặt màu vẽ bằng màu nền
    circle(x, y, r);                                    // vẽ vành bánh xe
    for (i=0; i<6; i++) {
        x1 = x+int(r*cos(phi)); y1 = y-int(r*sin(phi));
        x2 = x-int(r*cos(phi)); y2 = y+int(r*sin(phi));
        line(x1, y1, x2, y2);                           // vẽ các nan hoa
```

```
            phi = phi + pi/3;                    // lệch nhau 60^0
        }
        setcolor(WHITE);
    }

    void xoay()
    {
        int i, x, y, r;
        static float phi = 0;
        x = midx; y = midy; r = 100;
        while (!kbhit()) {
            bx(x, y, r, phi, 0);                 // vẽ bánh xe
            delay(100);                          // tạm dừng
            bx(x, y, r, phi, 1);                 // xóa bánh xe
            phi = phi-pi/72;                     // xoay đi một góc phi
        }
    }
```

Ví dụ 8 : Vẽ bánh xe lăn trên đường nằm ngang

```
    void lan()
    {
        int i, x, y, r;
        float phi=0;
        x = 0; y = maxy-110; r = 60;
        setlinestyle(SOLID_LINE, 1, 3);
        line(0, maxy-50, maxx, maxy-50);
        setlinestyle(SOLID_LINE, 1, 1);
        while (x-r<=maxx) {
            bx(x, y, r, phi, 0);
            delay(20); bx(x, y, r, phi, 1); x += 1;    phi = phi-pi/72;
        }
```

```
}
```

6. Vẽ đồ thị của các hàm toán học

Để vẽ đồ thị của một hàm toán học, ta vẽ từng điểm một của đồ thị. Mỗi điểm được xác định bởi cặp tọa độ (x, y) trên màn hình. Do vậy cần tính các điểm này theo tọa độ trên màn hình. Các bước cần làm gồm có:

- Xác định hệ trục tọa độ. Thông thường ta sẽ lấy tâm màn hình làm tâm hệ trục bằng việc xác định lại cửa sổ màn hình bởi câu lệnh:

```
viewport(midx, midy, maxx, maxy, 0);
```

trong đó midx, midy là tọa độ tâm màn hình, maxx, maxy là tọa độ góc dưới bên phải của màn hình. Câu lệnh trên tạo một cửa sổ là phần tư bên phải, phía dưới của màn hình. Tham trị cuối (1) cho phép các hình vẽ sẽ được vẽ ra ngoài khung cửa sổ này. Như vậy tâm màn hình sẽ biến thành tâm của hệ trục tọa độ. Tọa độ của tâm màn hình bây giờ được tính là (0,0).

- Xác định tỉ lệ: Cần xác định một đơn vị của x và y của hàm cần vẽ sẽ tương ứng với bao nhiêu điểm trên trục x và y của màn hình. Do số điểm theo chiều rộng và chiều cao của màn hình khác nhau và do giá trị của hàm (y) có thể rất lớn so với giá trị của đối (x) (ví dụ hàm $y = x^4$) hoặc rất bé (ví dụ hàm y = sinx) nên các tỉ lệ này theo x và y có thể khác nhau để hình vẽ trên màn hình được cân đối. Việc xác định các tỉ lệ này phụ thuộc vào kinh nghiệm và thường được điều chỉnh sau khi chạy thử chương trình.
- Vẽ hệ trục : Có thể vẽ hệ trục tọa độ hay không. Hàm sau cho phép vẽ các trục tọa độ với tâm nằm giữa màn hình.

```
void vetruc()                                   // Ve truc toa do
{
        line(0, midy, maxx, midy);              // truc hoanh
        line(maxx-7, midy-3, maxx, midy);       // mui ten
        line(maxx-7, midy+3, maxx, midy);
        line(midx, 0, midx, maxy);              // truc tung
        line(midx-3, 7, midx, 0);               // mui ten
        line(midx+3, 7, midx, 0);
        outtextxy(midx+6, midy+6, "(0, 0)");    // in toa do (0,0)
}
```

Các ví dụ sau sẽ vẽ đồ thị của một số hàm quen thuộc.

```
void Sinx()                                  // Do thi ham Sinx
{
    int tileX = 20, tileY = 60;              // Tỉ lệ theo X và Y
    int x, y, i;
    setviewport(midx, midy, maxx, maxy, 0);
    for (i = -400; i<=400; i++) {
        x = 2*pi*i*tileX/200;
        y = sin(2*pi*i/200)*tileY;
        putpixel(x, y, 1);
    }
    setviewport(0, 0, maxx, maxy, 0);
}

void Sinoverx()                              // Ham Sinx/x
{
    float t;
    float tileX = 50/pi;
    float tileY = 80;
    int x, y;
    for (x = 30; x < maxx-30; x++) {
        t = ((x==midx)? 1 : (x-midx))/tileX;
        y = midy - int(sin(t)/t*tileY);
        putpixel(x, y, 2);
    }
}
```

Ve do thi theo tham so (x = x(t), y = y(t))

```
void Hypocycloide()                          // Ham x = cos^3t, y = sin^3t
{                                            // t ∈ [0, 2π]
```

```
        float t;
        int i, x, y;
        for (i = 0; i<1000; i++) {
                t = (pi/500)*i;
                x = int(120*pow(cos(t), 3)) + midx;
                y = int(120*pow(sin(t), 3)) + midy;
                putpixel(x, y, 3);
        }
}
void Trocoide()                          // Ham (2t-3sint, 2-3cost)
{                                        // t ∈ [-9, 9]
        float t;
        int i, x, y;
        for (i = -1000; i<=1000; i++) {
                t = 0.01*i;
                x = int(15*(2*t-3*sin(t))) + midx;
                y = -int(10*(2-3*cos(t))) + midy;
                putpixel(x, y, 4);
        }
}
void So3()                               // x = sintcos²t + sint
{                                        // y = sin²tcost, t ∈ [0, 2π]
        float t;
        int i, x, y;
        for (i = 0; i<=1000; i++) {
                t = (pi/500)*i;
                x = int(150*(sin(t)*(1+cos(t)*cos(t)))) + midx;
                y = int(200*sin(t)*sin(t)*cos(t)) + midy;
                putpixel(x, y, 5);
        }
}
```

Ve do thi theo toa do cuc r = φ(θ)

```
void Archimede()                          // Ham r = θ, θ ∈ [0, 40]
{
    int i, x, y;
    float r, t;
    for (i = 0;i<=2500;i++) {
        t = 0.02*i;
        x = int(3*t*cos(t))+midx;
        y = -int(3*t*sin(t))+midy;
        putpixel(x, y, 6);
    }
}

void Hoahong()                            // Ham r = sin2θ, θ ∈ [0, 2π]
{
    int i, x, y;
    float r, t;
    for (i = 0;i<=2000;i++) {
        t = (pi/500)*i;
        x = int(150*(sin(2*t)*cos(t)))+midx;
        y = int(150*sin(2*t)*sin(t))+midy;
        putpixel(x, y, 7);
    }
}
```

Chương trình dưới đây cho phép vẽ hai mặt trong không gian 3 chiều được cho bởi hai hàm f = sinx.siny và g = $\frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$

```
typedef struct TOADO {
    int OX, OY, UX, UY, UZ;          // truc hoanh, tung va don vi cac truc
    double Xx, Xy;  // goc (OX, ox), (OY, oy)
```

```
};
TOADO gr3 = { 320, 20, 20, 20, 20, 0.8*pi, 0.2*pi };

void Vetruc()                                   // Ve truc Ox, Oy
{
        setviewport(0, 0, maxx, maxy, 0);
        settextstyle(DEFAULT_FONT, HORIZ_HUONG, 0);
        setcolor(WHITE);
        line(0, midy, maxx, midy);
        line(maxx-7, midy-3, maxx, midy); line(maxx-7, midy+3, maxx, midy);
        line(midx, 0, midx, maxy);
        line(midx-3, 7, midx, 0); line(midx+3, 7, midx, 0);
        outtextxy(midx+6, midy+6, "(0, 0)");
        outtextxy(maxx-18, midy+6, "x"); outtextxy(midx+8, 6, "y");
        setbkcolor(CYAN); setcolor(RED);
        settextstyle(TRIPLEX_FONT, HORIZ_HUONG, 2);
        outtextxy(10, 0, "DO THI KHONG GIAN 3 CHIEU");
}

int X(double x, double y, double z)     // doi toa do xyz sang truc X
{
        return gr3.OX + x*gr3.UX*cos(gr3.Xx) + y*gr3.UY*cos(gr3.Xy);
}
int Y(double x, double y, double z)     // doi toa do xyz sang truc Y
{
        return gr3.OY + x*gr3.UX*sin(gr3.Xx) + y*gr3.UY*sin(gr3.Xy) – z*gr3.UZ;
}
double f(double x, double y)            // Ham f(x, y) can ve
{
        return 4*sin(x)*sin(y);
}
```

```
double g(double x, double y)            // Ham g(x, y) can ve
{
        return 5*sin(sqrt(x*x+y*y))/sqrt(x*x+y*y);
}
void Vehamf()
{
        double x, y, z;
        double xa = -6.28, xb = 6.28;
        double ya = -6.28, yb = 6.28;
        double xp = 0.2, yp = 0.2;
        int mat[8];
        settextstyle(TRIPLEX_FONT, HORIZ_HUONG, 1);
        outtextxy(10, 20, "Ham z = sinx.siny");
        setviewport(0, midy, maxx, maxy, 0);
        for (x = xa; x <= xb; x+=xp)                    // ve mat an
        for (y = ya; y <= yb; y+=yp)
        {
                if (kbhit()) return;
                z = f(x, y);                            // diem A
                mat[0] = X(x, y, z); mat[1] = Y(x, y, z);
                z = f(x, y+yp);                         // diem B
                mat[2] = X(x, y+yp, z); mat[3] = Y(x, y+yp, z);
                z = f(x+xp, y+yp);                      // diem C
                mat[4] = X(x+xp, y+yp, z); mat[5] = Y(x+xp, y+yp, z);
                z = f(x+xp, y);                         // diem D
                mat[6] = X(x+xp, y, z); mat[7] = Y(x+xp, y, z);
                if ((mat[3]-mat[1]) * (mat[6]-mat[0]) –
                        (mat[7]-mat[1]) * (mat[2]-mat[0]) < 0)
                setfillstyle(1, YELLOW); else setfillstyle(1, GREEN);
                fillpoly(4, mat);
                delay(10);
```

```
        }
        getch();
}

void Vehamg()
{
        double x, y, z;
        double xa = -10, xb = 10;
        double ya = -10, yb = 10;
        double xp = 0.1, yp = 0.1;
        settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);
        outtextxy(10, 20, "Ham z = sin(sqrt(x*x+y*y))");
        outtextxy(100, 30, "------------");
        outtextxy(115, 40, "sqrt(x*x+y*y)");
        setviewport(0, midy, maxx, maxy, 0);
        setcolor(BLUE);
        for (x = xa; x <= xb; x+=xp)
        for (y = ya; y <= yb; y+=yp)
        {
                if (kbhit()) return;
                z = g(x, y); lineto(X(x, y, z), Y(x, y, z));
                delay(10);
        }
        getch();
}

void main()
{
        Initgraph(); Vetruc(); Vehamf();
        cleardevice(); Vetruc(); Vehamg();
        closegraph();
}
```

II. ÂM THANH

Âm thanh được đặc trưng bởi cao độ (tần số) và trường độ (độ dài). Việc tạo ra một chuỗi âm (bài hát chẳng hạn), là sự kết hợp lặp đi lặp lại của các hàm sau với các tham số n và t được chọn thích hợp.

– sound(n): phát âm ra loa máy tính với tần số n.

– delay(t): kéo dài trong t miligiây.

– nosound(): tắt âm thanh đã phát.

Ví dụ 1 : Tiếng còi báo động

```
void coi(int cao; int thap)
{
    do {
        sound(cao); delay(400); sound(thap); delay(400);
    } while (!kbhit())
    nosound();
}
```

Ví dụ 2 : Tiếng bóng nảy

```
void bong(int cao; int thap)
{
    int sodem = 20;
    while (sodem > 1) {
        sound(thap-2*sodem); delay(sodem*500/20);
        nosound(); delay(100);
        sound(cao); delay(sodem*500/15); nosound(); delay(150);
        sodem --;
    }
}
```

Ví dụ 3 : Tiếng bom

```
void bong(int cao; int thap; int t)
{
    int sodem = thap;
    while (sodem <= cao) {
        sound(sodem); delay(t/sodem*75);
        sodem += 10;
    }
    for (sodem =1 to 3) {
        nosound();
        sound(40); delay(500); nosound(); delay(100);
    }
    sound(40); delay(3000); nosound();
}
```

Để tạo âm phát của một nốt nhạc có tần số (cao độ) n và dài trong t miligiây cần viết hàm :

```
void not(unsigned n, float t);
{
    sound(n);
    delay(t);
    nosound();
}
```

Ví dụ 4 : Chơi bài hát Tiến quân ca trên nền cờ đỏ sao vàng.

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
// cao độ của các nốt nhạc

#define  do1    66    #define  dod1   70    #define  re1    73
#define  red1   78    #define  mi1    82    #define  fa1    86
```

```
#define  fad1   91    #define  sol1   96    #define  sold1  102
#define  la1    108   #define  lad1   115   #define  si1    122
#define  do2    130   #define  dod2   139   #define  re2    148
#define  re2    148   #define  red2   156   #define  mi2    164
#define  fa2    176   #define  fad2   188   #define  sol2   196
#define  sold2  209   #define  la2    230   #define  lad2   233
#define  si2    247   #define  do3    264   #define  dod3   281
#define  re3    297   #define  red3   313   #define  mi3    330
#define  fa3    352   #define  fad3   374   #define  sol3   396
#define  sold3  415   #define  la3    440   #define  lad3   468
#define  si3    495   #define  do4    528   #define  dod4   565
#define  re4    594   #define  red4   625   #define  mi4    660
#define  fa4    704   #define  fad4   748   #define  sol4   792
#define  sold4  836   #define  la4    880   #define  lad4   935
#define  si4    990   #define  lang   30000

void not(unsigned caodo, float truongdo)
{ sound(caodo); delay(truongdo); nosound(); }

void main()
{
      int gdriver = DETECT, gmode;
      initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
      int star[20] = {320, 150, 285, 225, 200, 225, 270, 270, 240, 350,
      320, 300, 390, 350, 360, 270, 430, 225, 345, 225};
      setbkcolor(RED); setcolor(YELLOW);       // Vẽ lá cờ đỏ sao vàng
      setfillstyle(SOLID_FILL, YELLOW);
      fillpoly(10, star);

      // Trường độ các nốt nhạc
      float d = 300;                  // đen
      float tr = 2*d;                 // trắng
```

```
        float tro = 4*d;            // tròn
        float md = d/2;             // móc đen
        float mk = d/4;             // móc kép
        float m3 = d/8;             // móc 3
        float m4 = d/16;            // móc 4
        float dc = 3*d/2;           // đen chấm
        float trc = 3*d;            // trắng chấm
        float troc = 6*d;           // tròn chấm

        // Choi bai TQC
        not(re2, d); not(mi2, d); not(re2, d); not(sol2, tr); not(sol2, tro);
        not(la2, d); not(sol2, d); not(si2, tr); not(si2, tro); not(la2, d);
        not(sol2, d); not(mi2, tr); not(sol2, tr); not(sol2, d); not(mi2, tro);
        not(re2, d); not(si2, d); not(re2, tro); not(sol2, d); not(la2, d);
        not(si2, tr); not(si2, tr); not(si2, tr); not(la2, d); not(sol2, d);
        not(re3, tro); not(si2, d); not(sol2, d); not(la2, tr); not(la2, tr);
        not(si2, tr); not(fad2, d); not(re2, d); not(sol2, tro); not(si2, d);
        not(do3, d); not(re3, tr); not(re3, tr); not(mi3, tro); not(re3, d);
        not(si2, tro); not(si2, tr); not(la2, d); not(sol2, tr); not(re2, tr);
        not(fad2, tr); not(fad2, d); not(la2, d); not(sol2, tr); not(si2, d);
        not(do3, d); not(re3, tr); not(re3, tr); not(mi3, tro); not(re3, d);
        not(si2, tro); not(si2, tr); not(la2, d); not(sol2, tr); not(sol2, tr);
        not(re2, tro); not(re3, tro); not(si2, tr); not(sol2, tr); not(mi3, tro);
        not(re3, tr); not(si2, d); not(la2, d); not(re2, d); not(la2, tr);
        not(la2, tro); not(si2, tr); not(sol2, tro);
        closegraph();
}
```

BÀI TẬP

1. Vẽ hai hình chữ nhật, lần lượt cho mất từng hình, rồi hiện lại cả hai.

2. Biểu diễn dãy 5 giá trị (được nhập từ bàn phím) bằng biểu đồ bar.

3. Biểu diễn dãy 5 giá trị (được nhập từ bàn phím) bằng biểu đồ hình quạt.

4. Vẽ một bàn cờ quốc tế với các ô đen trắng.

5. Viết chương trình vẽ đồ thị hàm số y = 100*sin(x/4.8) trong khoảng x $\in$ [0, 60] với giá trị mỗi bước $\Delta x = 0,1$. Yêu cầu :

- nền màn hình mầu đen.
- trục tọa độ màu xanh lá cây
- đồ thị màu trắng.

6. Viết chương trình vẽ tam giác với các tọa độ đỉnh lần lượt là A(300, 20), B(100, 220), C(500, 220) và đường tròn ngoại tiếp của nó. Yêu cầu :

- nền màn hình mầu đen.
- các cạnh tam giác màu xanh lá cây
- đường tròn ngoại tiếp màu đỏ tươi.

7. Viết chương trình vẽ hình chữ nhật có tọa độ đỉnh góc trên bên trái là (100,150), chiều ngang 120, chiều dọc 90 và đường tròn ngoại tiếp nó. Yêu cầu :

- nền màn hình mầu đen.
- các cạnh hình chữ nhật màu xanh da trời.
- đường tròn ngoại tiếp màu đỏ tươi.

8. Vẽ tam giác nội tiếp trong hình tròn, hình tròn nội tiếp trong elip. Tô các màu khác nhau cho các miền tạo bởi các đường trên.

9. Vẽ một đài phát sóng. Các vòng sóng phát từ đỉnh của tháp ở tâm màn hình lan tỏa ra chung quanh. Quá trình lặp đến khi ấn phím bất kỳ thì dừng.

10. Vẽ hai hình người đi vào từ 2 phía màn hình với tốc độ khác nhau. Gặp nhau hai hình người xoay lại và đi ngược về 2 phía màn hình. Chương trình dừng khi cả hai đã đi khuất vào hai phía của màn hình.

CHƯƠNG 7

LỚP VÀ ĐỐI TƯỢNG

Lập trình có cấu trúc và lập trình hướng đối tượng
Lớp và đối tượng
Đối của phương thức - Con trỏ this
Hàm tạo (contructor)
Hàm hủy (destructor)
Các hàm trực tuyến (inline)

I. LẬP TRÌNH CÓ CẤU TRÚC VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1. Phương pháp lập trình cấu trúc

- Lập trình cấu trúc là tổ chức chương trình thành các chương trình con. Trong một số ngôn ngữ như PASCAL có 2 kiểu chương trình con là thủ tục và hàm, còn trong C++ chỉ có một loại chương trình con là hàm.
- Hàm là một đơn vị chương trình độc lập dùng để thực hiện một phần việc nào đó như: Nhập số liệu, in kết quả hay thực hiện một số công việc tính toán. Hàm cần có đối và các biến, mảng cục bộ dùng riêng cho hàm.
- Việc trao đổi dữ liệu giữa các hàm thực hiện thông qua các đối và các biến toàn cục.
- Một chương trình cấu trúc gồm các cấu trúc dữ liệu (như biến, mảng, bản ghi) và các hàm, thủ tục.
- Nhiệm vụ chính của việc tổ chức thiết kế chương trình cấu trúc là tổ chức chương trình thành các hàm, thủ tục.

Ví dụ, ta xét yêu cầu sau: Viết chương trình nhập toạ độ (x,y) của một dãy điểm, sau đó tìm một cặp điểm cách xa nhau nhất.

Trên tư tưởng của lập trình cấu trúc có thể tổ chức chương trình như sau:

- Sử dụng 2 mảng thực toàn bộ x và y để chứa toạ độ dãy điểm.
- Xây dựng 2 hàm:

Hàm nhapsl dùng để nhập toạ độ n điểm, hàm này có một đối là biến nguyên n và được khai báo như sau:

```
void nhapsl(int n);
```

Hàm do_dai dùng để tính độ dài đoạn thẳng đi qua 2 điểm có chỉ số là i và j,

nó được khai báo như sau:

```
float do_dai(int i, int j);
```

Chương trình C của ví dụ trên được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float x[100],y[100];
float do_dai(int i, int j)
{
    return sqrt(pow(x[i]-x[j],2)+pow(y[i]-y[j],2));
}
void nhapsl(int n)
{
    int i;
    for (i=1; i<=n; ++i)
    {
        printf("\n Nhap toa do x, y cua diem thu %d : ",i);
    scanf("%f%f",&x[i],&y[i]);
    }
}
void main()
{
    int n, i, j, imax,jmax;
    float d, dmax;
    printf("\n So diem N= ");
    scanf("%d", &n);
    nhapsl(n);
    dmax=do_dai(1,2);imax=1; jmax=2;
    for(i=1; i<=n-1; ++i)
    for (j=i+1; j<=n; ++j)
    {
        d=do_dai(i,j);
        if (d>dmax)
```

```
            {
                dmax=d;
                imax=i; jmax=j;
            }
        }
        printf("\nDoan thang lon nhat co do dai bang: %0.2f",dmax);
        printf("\n Di qua 2 diem co chi so la %d va %d",imax,jmax);
        getch();
}
```

2. Phương pháp lập trình hướng đối tượng

Là lập trình có cấu trúc + trừu tượng hóa dữ liệu. Có nghĩa chương trình tổ chức dưới dạng cấu trúc. Tuy nhiên việc thiết kế chương trình sẽ xoay quanh dữ liệu, lấy dữ liệu làm trung tâm. Nghĩa là trả lời câu hỏi: Chương trình làm việc với những đối tượng dữ liệu nào, trên các đối tượng dữ liệu này cần thao tác, thực hiện những gì. Từ đó gắn với mỗi đối tượng dữ liệu một số thao tác thực hiên cố định riêng của đối tượng dữ liệu đó, điều này sẽ qui định chặt chẽ hơn những thao tác nào được thực hiện trên đối tượng dữ liệu nào. Khác với lập trình cấu trúc thuần túy, trong đó dữ liệu được khai báo riêng rẽ, tách rời với thao tác xử lý, do đó việc xử lý dữ liệu thường không thống nhất khi chương trình được xây dựng từ nhiều lập trình viên khác nhau.

Từ đó lập trình hướng đối tượng được xây dựng dựa trên đặc trưng chính là khái niệm đóng gói. Đóng gói là khái niệm trung tâm của phương pháp lập trình hướng đối tượng, trong đó dữ liệu và các thao tác xử lý nó sẽ được qui định trước và "đóng" thành một "gói" thống nhất, riêng biệt với các dữ liệu khác tạo thành kiểu dữ liệu với tên gọi là các lớp. Như vậy một lớp không chỉ chứa dữ liệu bình thường như các kiểu dữ liệu khác mà còn chứa các thao tác để xử lý dữ liệu này. Các thao tác được khai báo trong gói dữ liệu nào chỉ xử lý dữ liệu trong gói đó và ngược lại dữ liệu trong một gói chỉ bị tác động, xử lý bởi thao tác đã khai báo trong gói đó. Điều này tạo tính tập trung cao khi lập trình, mọi đối tượng trong một lớp sẽ chứa cùng loại dữ liệu được chỉ định và cùng được xử lý bởi các thao tác như nhau. Mọi lập trình viên khi làm việc với dữ liệu trong một gói đều sử dụng các thao tác như nhau để xử lý dữ liệu trong gói đó. C++ cung cấp cách thức để tạo một cấu trúc dữ liệu mới thể hiện các gói nói trên, cấu trúc dữ liệu này được gọi là **lớp**.

Để minh hoạ các khái niệm vừa nêu về kiêu dữ liệu lớp ta trở lại xét bài toán tìm độ dài lớn nhất đi qua 2 điểm. Trong bài toán này ta gặp một thực thể là dãy điểm. Các thành phần dữ liệu của lớp dãy điểm gồm:

- Biến nguyên n là số điểm của dãy
- Con trỏ x kiểu thực trỏ đến vùng nhớ chứa dãy hoành độ

- Con trỏ y kiểu thực trỏ đến vùng nhớ chứa dãy tung độ

Các phương thức cần đưa vào theo yêu cầu bài toán gồm:

- Nhập toạ độ một điểm
- Tính độ dài đoạn thẳng đi qua 2 điểm

Dưới đây là chương trình viết theo thiết kế hướng đối tượng.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <alloc.h>
class daydiem
{
        int n;
        float *x,*y;
public:
        float do_dai(int i, int j)
        {
                return sqrt(pow(x[i]-x[j],2)+pow(y[i]-y[j],2));
        }
        void nhapsl(void);
};
void daydiem::nhapsl(void)
{
        int i;
        printf("\n So diem N= ");
        scanf("%d",&n);
        x = (float*)malloc((n+1)*sizeof(float));
        y = (float*)malloc((n+1)*sizeof(float));
        for (i=1; i<=n; ++i)
        {
                printf("\n Nhap toa do x, y cua diem thu %d : ",i);
                scanf("%f%f",&x[i],&y[i]);
        }
}
```

```
void main()
{
	clrscr();
	daydiem p;
	p.nhapsl();
	int n,i,j,imax,jmax;
	float d,dmax;
	n = p.n;
	dmax=p.do_dai(1,2);imax=1; jmax=2;
	for (i=1;i<=n-1;++i)
	for (j=i+1;j<=n;++j)
	{
		d=p.do_dai(i,j);
		if (d>dmax)
		{
			dmax=d;
			imax=i; jmax=j;
		}
	}
	printf("\n Doan thang lon nhat co do dai bang: %0.2f",dmax);
	printf("\n Di qua 2 diem co chi so la %d va %d" , imax,jmax);
	getch();
}
```

II. LỚP VÀ ĐỐI TƯỢNG

Trong lập trình hướng đối tượng, lớp (class) là một khái niệm rất quan trọng, nó cho phép giải quyết các vấn đề phức tạp của việc lập trình. Một lớp đơn (được định nghĩa như struct, union, hoặc class) bao gồm các hàm và dữ liệu có liên quan. Các hàm này là các hàm thành phần (member functon) hay còn là phương thức (method), thể hiện tác động của lớp có thể được thực hiện trên dữ liệu của chính lớp đó (data member).

Cũng giống như cấu trúc, lớp có thể xem như một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và lớp được dùng để khai báo các biến, mảng đối tượng (như thể dùng kiểu int để khai báo các biến mảng nguyên).

Như vậy từ một lớp có thể tạo ra (bằng cách khai báo) nhiều đối tượng (biến,

mảng) khác nhau. Mỗi đối tượng có vùng nhớ riêng của mình và vì vậy ta cũng có thể quan niệm lớp chính là tập hợp các đối tượng cùng kiểu.

1. Khai báo lớp

Để khai báo một lớp, ta sử dụng từ khoá class như sau:

```
class tên_lớp
{
    // Khai báo các thành phần dữ liệu (thuộc tính)
    // Khai báo các phương thức (hàm)
};
```

Chú ý: Việc khai báo một lớp không chiếm giữ bộ nhớ, chỉcác đối tượng của lớp mới thực sự chiếm giữ bộ nhớ.

Thuộc tính của lớp có thể là các biến, mảng, con trỏ có kiểu chuẩn (int, float, char, char*, long,...) hoặc kiểu ngoài chuẩn đã định nghĩa trước (cấu trúc, hợp, lớp,...). Thuộc tính của lớp không thể có kiểu của chính lớp đó, nhưng có thể là con trỏ của lớp này, ví dụ:

```
class A
{
    A x;          //Không cho phép, vì x có kiểu lớp A
    A* p ;        //Cho phép , vì p là con trỏ kiểu lớp A
} ;
```

2. Khai báo các thành phần của lớp (thuộc tính và phương thức)

a. Các từ khóa private và public

Khi khai báo các thành phần dữ liệu và phương thức có thể dùng các từ khoá private và public để quy định phạm vi sử dụng của các thành phần này. Trong đó từ khóa private qui định các thành phần (được khai báo với từ khóa này) chỉ được sử dụng bên trong lớp (trong thân các phương thức của lớp). Các hàm bên ngoài lớp (không phải là phương thức của lớp) không được phép sử dụng các thành phần này. Đặc trưng này thể hiện tính che giấu thông tin trong nội bộ của lớp, để đến được các thông tin này cần phải thông qua chính các thành phần hàm của lớp đó. Do vậy thông tin có tính toàn vẹn cao và việc xử lý thông tin (dữ liệu) này mang tính thống nhất hơn và hầu như dữ liệu trong các lớp đều được khai báo với từ khóa này. Ngược lại với private, các thành phần được khai báo với từ khóa public được phép sử dụng ở cả bên trong và bên ngoài lớp, điều này cho phép trong chương trình có thể sử dụng các hàm này để truy nhập đến dữ liệu của lớp. Hiển nhiên nếu các thành phần dữ liệu đã khai báo là privte thì các thành phần hàm phải có ít nhất một vài hàm được khai báo dạng public để chương trình có thể truy cập được, nếu không

toàn bộ lớp sẽ bị đóng kín và điều này không giúp gì cho chương trình. Do vậy cách khai báo lớp tương đối phổ biến là các thành phần dữ liệu được ở dạng private và thành phần hàm dưới dạng public. Nếu không quy định cụ thể (không dùng các từ khoá private và public) thì C++ hiểu đó là private.

b. Các thành phần dữ liệu (thuộc tính)

Được khai báo như khai báo các thành phần trong kiểu cấu trúc hay hợp. Bình thường các thành phần này được khai báo là private để bảo đảm tính giấu kín, bảo vệ an toàn dữ liệu của lớp không cho phép các hàm bên ngoài xâm nhập vào các dữ liệu này.

c. Các phương thức (hàm thành viên)

Thường khai báo là public để chúng có thể được gọi tới (sử dụng) từ các hàm khác trong chương trình.

Các phương thức có thể được khai báo và định nghĩa bên trong lớp hoặc chỉ khai báo bên trong còn định nghĩa cụ thể của phương thức có thể được viết bên ngoài. Thông thường, các phương thức ngắn được viết (định nghĩa) bên trong lớp, còn các phương thức dài thì viết bên ngoài lớp.

Một phương thức bất kỳ của một lớp, có thể sử dụng bất kỳ thành phần (thuộc tính và phương thức) nào của lớp đó và bất kỳ hàm nào khác trong chương trình (vì phạm vi sử dụng của hàm là toàn chương trình).

Giá trị trả về của phương thức có thể có kiểu bất kỳ (chuẩn và ngoài chuẩn)

Ví dụ sau sẽ minh hoạ các điều nói trên. Chúng ta sẽ định nghĩa lớp để mô tả và xử lý các điểm trên màn hình đồ hoạ. Lớp được đặt tên là DIEM.

- Các thuộc tính của lớp gồm:

```
int x ;          // Hoành độ (cột)
int y ;          // Tung độ (hàng)
int m ;          // Mầu
```

- Các phương thức:

  Nhập dữ liệu một điểm

  Hiển thị một điểm

  Ẩn một điểm

Lớp điểm được xây dựng như sau:

```
#include <iostream.h>
#include <graphics.h>
class DIEM
{
```

```
private:
        int x, y, m ;
public:
        void nhapsl() ;
        void hien() ;
        void an() { putpixel(x, y, getbkcolor());}
};

void DIEM::nhapsl()
{
        cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem: ";
        cin >> x >> y ;
        cout << "\n Nhap ma mau cua diem: ";
        cin >> m ;
}

void DIEM::hien()
{
        int mau_ht ;
        mau_ht = getcolor();
        putpixel(x, y, m);
        setcolor(mau_ht);
}
```

Qua ví dụ trên có thể rút ra một số chú ý sau:

+ Trong cả 3 phương thức (dù viết trong hay viết ngoài định nghĩa lớp) đều được phép truy nhập đến các thuộc tính x, y và m của lớp.
+ Các phương thức viết bên trong định nghĩa lớp (như phương thức an()) được viết như một hàm thông thường.
+ Khi xây dựng các phương thức bên ngoài lớp, cần dùng thêm tên lớp và toán tử phạm vi :: đặt ngay trước tên phương phức để quy định rõ đây là phương thức của lớp nào.

3. Biến, mảng và con trỏ đối tượng

Như đã nói ở trên, một lớp (sau khi định nghĩa) có thể xem như một kiểu đối tượng và có thể dùng để khai báo các biến, mảng đối tượng. Cách khai báo biến, mảng đối tượng cũng giống như khai báo biến, mảng các kiểu khác (như int, float,

cấu trúc, hợp,...), theo mẫu sau:

Tên_lớp danh sách đối ;

Tên_lớp danh sách mảng ;

Ví dụ sử dụng DIEM ở trên, ta có thể khai báo các biến, mảng DIEM như sau:

```
DIEM d1, d2, d3 ;      // Khai báo 3 biến đối tượng d1, d2, d3
DIEM d[20] ;           // Khai báo mảng đối tượng d gồm 20 phần tử
```

Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để chứa các thuộc tính của nó. Chú ý rằng sẽ không có vùng nhớ riêng để chứa các phương thức cho mỗi đối tượng, các phương thức sẽ được sử dụng chung cho tất cả các đối tượng cùng lớp. Như vậy về bộ nhớ được cấp phát thì đối tượng giống cấu trúc.

Trong trường hợp này:

```
sizeof(d1) = sizeof(d2) = sizeof(d3) = 3*sizeof(int) = 6
sizeof(d) = 20*6 = 120
```

a. Thuộc tính của đối tượng

Trong ví dụ trên, mỗi đối tượng d1, d2, d3 và mỗi phần tử d[i] đều có 3 thuộc tính là x, y, m. Chú ý là mỗi thuộc tính đều thuộc về một đối tượng, vì vậy không thể viết tên thuộc tính một cách riêng rẽ mà bao giờ cũng phải có tên đối tượng đi kèm, giống như cách viết trong cấu trúc của C. Nói cách khác, cách viết thuộc tính của đối tượng như sau:

tên_đối_tượng.Tên_thuộc_tính

Với các đối tượng d1, d2, d3 và mảng d, có thể viết như sau:

```
d1.x;          // Thuộc tính x của đối tượng d1
d2.x;          // Thuộc tính x của đối tượng d2
d3.y;          // Thuộc tính y của đối tượng d3
d[2].m;        // Thuộc tính m của phần tử d[2]
d1.x = 100;    // Gán 100 cho d1.x
d2.y =d1.x;    // Gán d1.x cho d2.y
```

b. Sử dụng các phương thức

Cũng giống như hàm, một phương thức được sử dụng thông qua lời gọi. Tuy nhiên trong lời gọi phương thức bao giờ cũng phải có tên đối tượng để chỉ rõ phương thức thực hiện trên các thuộc tính của đối tượng nào.

Ví dụ lời gọi sau sẽ thực hiện nhập số liệu vào các thành phần d1.x, d1.y và d1.m: **d1.nhapsl();** Câu lệnh sau sẽ thực hiện nhập số liệu vào các thành phần d[3].x, d[3].y và d[3].m: **d[3].nhapsl() ;**

Chúng ta sẽ minh họa các điều nói trên bằng một chương trình đơn giản sử

dụng lớp DIEM để nhập 3 điểm, hiện rồi ẩn các điểm vừa nhập. Trong chương trình đưa vào hàm kd_do_hoa() dùng để khởi động hệ đồ hoạ.

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>

class DIEM
{
private:
        int x, y, m ;
public:
        void nhapsl();
        void an() { putpixel(x,y,getbkcolor());}
        void hien();
};

void DIEM::nhapsl()
{
        cout << "\n Nhap hoanh do (cot) va tung do (hang) cua DIEM: " ;
        cin>> x >> y ;
        cout << " \n Nhap ma tran cua diem: " ;
        cin >> m ;
}

void DIEM::hien()
{
        int mau_ht;
        mau_ht = getcolor() ;
        putpixel(x,y,m);
        setcolor(mau_ht);
}

void kd_do_hoa()
{
        int mh, mode ;
```

```
    mh=mode=0;
    initgraph(&mh, &mode, "C:\\TC\BGI");
}

void main()
{
    DIEMd1, d2, d3 ;
    d1.nhapsl(); d2.nhapsl(); d3.nhapsl();
    kd_do_hoa();
    setbkcolor(BLACK);
    d1.hien(); d2.hien(); d3.hien();
    getch();
    d1.an(); d2.an(); d3.an();
    getch();
    closegraph();
}
```

c. Con trỏ đối tượng

Con trỏ đối tượng dùng để chứa địa chỉ của biến, mảng đối tượng. Nó được khai báo như sau:

```
Tên_lớp *con trỏ;
```

Ví dụ dùng lớp DIEM có thể khai báo:

```
DIEM *p1, *p2, *p3 ;        // Khai báo 3 con trỏ p1, p2, p3
DIEM d1, d2 ;               // Khai báo 2 đối tượng d1, d2
DIEM d[20] ;                // Khai báo mảng đối tượng
```

và có thể thực hiện các câu lệnh:

```
p1= &d2 ;                   // p1 chứa địa chỉ của d2 , hay p1 trỏ tới d2
p2 = d ;                    // p2 trỏ tới đầu mảng d
p3 = new DIEM               // Tạo một đt và chứa địa chỉ của nó vào p3
```

Để sử dụng thuộc tính của đối tượng thông qua con trỏ, ta viết như sau:

Tên_con_trỏ → Tên_thuộc_tính

Chú ý: Nếu con trỏ chứa địa chỉ đầu của mảng, có thể dùng con trỏ như tên mảng.

Như vậy sau khi thực hiện các câu lệnh trên thì:

p1 → x và d2.x là như nhau

p2[i].y và d[i].y là như nhau

Từ đó ta có quy tắc sử dụng thuộc tính: Để sử dụng một thuộc tính của đối tượng ta phải dùng phép . hoặc phép →. Trong chương trình, không cho phép viết tên thuộc tính một cách đơn độc mà phải đi kèm tên đối tượng hoặc tên con trỏ theo các mẫu sau:

Tên_đối_tượng.Tên_thuộc_tính

Tên_con_trỏ → Tên_thuộc_tính

Tên_mảng_đối_ tượng[chỉ_số].Tên_thuộc_tính

Tên_con_trỏ[chỉ_số].Tên_thuộc_tính

Chương trình dưới đây cũng sử dụng lớp DIEM để nhập một dãy điểm, hiển thị và ẩn các điểm vừa nhập. Chương trình dùng một con trỏ kiểu DIEM và dùng toán tử new để tạo ta một dãy đối tượng

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
class DIEM
{
private:
      int x, y, m ;
public:
      void nhapsl();
      void an() { putpixel(x,y,getbkcolor());}
      void hien();
};

void DIEM::nhapsl()
{
      cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
      cin >> x >> y ;
      cout << " \n Nhap ma mau cua DIEM: " ;
      cin >> m ;
}

void DIEM::hien()
{
```

```
        int mau_ht;
        mau_ht = getcolor() ;
        putpixel(x,y,m);
        setcolor(mau_ht);
}

void kd_do_hoa()
{
        int mh, mode ;
        mh=mode=0;
        initgraph(&mh, &mode, "C:\\TC\BGI");
}

void main()
{
        DIEM *p;
        int i, n;
        cout << "So diem: " ;
        cin >> n;
        p = new DIEM[n+1];
        for (i=1;i<=n;++i)
        p[i].nhapsl();
        kd_do_hoa();
        for (i=1;i<=n;++i) p[i].hien();
        getch();
        for (i=1;i<=n;++i) p[i].an();
        getch();
        closegraph();
}
```

III. ĐỐI CỦA PHƯƠNG THỨC, CON TRỎ THIS

1. Con trỏ this là đối thứ nhất của phương thức

Chúng ta hãy xem lại phương thức nhapsl của lớp DIEM

```
void DIEM::nhapsl()
```

```
{
    cout <<"\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
    cin >> x >> y ;
    cout <<" \n Nhap ma mau cua diem: " ;
    cin >> m ;
}
```

Trong phương thức này chúng ta sử dụng tên các thuộc tính x, y và m một cách đơn độc. Điều này có vẻ như mâu thuẫn với quy tắc sử dụng thuộc tính nêu trong mục trước. Thực tế C++ đã ngầm định sử dụng một con trỏ đặc biệt với tên gọi this trong các phương thức trên. Các thuộc tính viết trong phương thức được hiểu là thuộc một đối tượng do con trỏ this trỏ tới. Do đó, nếu tường minh hơn, phương thức nhapsl() có thể được viết dưới dạng tương đương như sau:

```
void DIEM::nhapsl()
{
    cout << "\n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
    cin >> this → x >> this → y ;
    cout << "\n Nhap ma mau cua diem: " ;
    cin >>this → m;
}
```

Như vậy có thể kết luận rằng: Phương thức bao giờ cũng có ít nhất một đối là con trỏ this và nó luôn luôn là đối đầu tiên của phương thức.

2. Tham số ứng với đối con trỏ this

Xét một lời gọi tới phương thức nhapsl() :

```
DIEM d1;
d1.nhapsl() ;
```

Trong trường hợp này tham số truyền cho con trỏ this chính là địa chỉ của d1:

```
this = &d1
```

Do đó:

this → x chính là d1.x

this → y chính là d1.y

this → m chính là d1.m

Như vậy câu lệnh:**d1.nhapsl() ;**sẽ nhập dữ liệu cho các thuộc tính của đối tượng d1. Từ đó có thể rút ra kết luận sau:

Tham số truyền cho đối con trỏ this chính là địa chỉ của đối tượng đi kèm với

phương thức trong lời gọi phương thức.

3. Các đối khác của phương thức

Ngoài đối đặc biệt this (đối này không xuất hiện một cách tường minh), phương thức còn có các đối khác được khai báo thư trong các hàm. Đối của phương thức có thể cókiểu bất kỳ (chuẩn và ngoài chuẩn).

Ví dụ để xây dựng phương thức vẽ đường thẳng qua 2 điểm ta cần đưa vào 3 đối: Hai đối là 2 biến kiểu DIEM, đối thứ ba kiểu nguyên xác định mã mầu. Vì đã có đối ngầm định this là đối thứ nhất, nên chỉ cần khai báo thêm 2 đối. Phương thức có thể viết như sau:

```
void DIEM::doan_thang(DIEM d2, int mau)
{
    int mau_ht;
    mau_ht = getcolor();
    setcolor(mau);
    line(this → x, this → y,d2.x,d2.y);
    setcolor(mau_ht);
}
```

Chương trình sau minh hoạ các phương thức có nhiều đối. Ta vẫn dùng lớp DIEM nhưng có một số thay đổi:

- Bỏ thuộc tính m (mầu)
- Bỏ các phương thức ***hien*** và ***an***
- Đưa vào 4 phương thức mới:

  ve_doan_thang (Vẽ đoạn thẳng qua 2 điểm)

  ve_tam_giac (Vẽ tam giác qua 3 điểm)

  do_dai (Tính độ dài của đoạn thẳng qua 2 điểm)

  chu_vi (Tính chu vi tam giác qua 3 điểm)

Chương trình còn minh hoạ:

- Việc phương thức này sử dụng phương thức khác (phương thức ve_tam_giac sử dụng phương thức ve_doan_thang, phương thức chu_vi sử dụng phương thức do_dai)
- Sử dụng con trỏ this trong thân các phương thức ve_tam_giac và chu_vi

Nội dung chương trình là nhập 3 điểm, vẽ tam giác có đỉnh là 3 điểm vừa nhập sau đó tính chu vi tam giác.

```
#include <conio.h>
```

```
#include <iostream.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
class DIEM
{
private:
	int x, y ;
public:
	void nhapsl();
	void ve_doan_thang(DIEM d2, int mau) ;
	void ve_tam_giac(DIEM d2, DIEM d3,int mau) ;
	double do_dai(DIEM d2)
	{
		DIEM d1 = *this ;
		return sqrt(pow(d1.x - d2.x,2) + pow(d1.y - d2.y,2) ) ;
	}
	double chu_vi(DIEM d2, DIEM d3);
};

void DIEM::nhapsl()
{
	cout <<" \n Nhap hoanh do (cot) va tung do (hang) cua diem:" ;
	cin >> x >> y;
}

void kd_do_hoa()
{
	int mh, mode ;
	mh=mode=0;
	initgraph(&mh, &mode, "");
}

void DIEM::ve_doan_thang(DIEM d2, int mau)
{
```

```
        setcolor(mau);
        line(this → x,this → y,d2.x,d2.y);
}
void DIEM:: ve_tam_giac(DIEM d2, DIEM d3,int mau)
{
        (*this).ve_doan_thang(d2,mau);
        d2.ve_doan_thang(d3,mau);
        d3.ve_doan_thang(*this,mau);
}
double DIEM:: chu_vi(DIEM d2, DIEM d3)
{
        double s;
        s=(*this).do_dai(d2)+ d2.do_dai(d3) + d3.do_dai(*this) ;
        return s;
}
void main()
{
        DIEM d1, d2, d3;
        char tb_cv[20] ;
        d1.nhapsl();
        d2.nhapsl();
        d3.nhapsl();
        kd_do_hoa();
        d1.ve_tam_giac(d2,d3,15);
        double s = d1.chu_vi(d2,d3);
        sprintf(tb_cv, "chu_vi = %0.2f", s);
        outtextxy(10,10,tb_cv);
        getch();
        closegraph();
}
```

Một số nhận xét về đối của phương thức và lời gọi phương thức:

+ Quan sát nguyên mẫu phương thức:

```
void ve_doan_thang(DIEM d2, int mau) ;
```

sẽ thấy phương thức có 3 đối:

Đối thứ nhất là một đối tượng DIEM do this trỏ tới

Đối thứ hai là đối tượng DIEM d2

Đối thứ ba là biến nguyên mẫu

Nội dung phương thức là vẽ một đoạn thẳng đi qua các điểm *this và d2 theo mã mầu mau. Xem thân của phương sẽ thấy được nội dung này:

```
void DIEM::ve_doan_thang(DIEM d2, int mau) ;
{
    setcolor(mau);
    line(this → x,this → y,d2.x,d2.y);
}
```

Tuy nhiên trong trường hợp này, vai trò của this không cao lắm, vì nó được đưa vào chỉ cốt làm rõ đối thứ nhất. Trong thân phương thức có thể bỏ từ khóa this vẫn được.

+ Vai trò của this trở nên quan trọng trong phương thức ve_tam_giac:

```
voidve_tam_giac(DIEM d2, DIEM d3, int mau)
```

Phương thức này có 4 đối là:

this : trỏ tới một đối tượng kiểu DIEM

d2 : một đối tượng kiểu DIEM

d3 : một đối tượng kiểu DIEM

mau : một biến nguyên

Nội dung phương thức là vẽ 3 cạnh:

cạnh 1 đi qua *this và d2

cạnh 2 đi qua d2 và d3

cạnh 3 đi qua d3 và *this

Các cạnh trên đuợc vẽ nhờ sử dụng phương thức ve_doan_thang:

Vẽ cạnh 1 dùng lệnh: (*this).ve_doan_thang(d2,mau) ;

Vẽ cạnh 2 dùng lệnh: d2.ve_doan_thang(d3,mau);

Vẽ cạnh 3 dùng lệnh: d3.ve_doan_thang(*this,mau);

Trong trường này rõ ràng vai trò của this rất quan trọng. Nếu không dùng nó thì công việc trở nên khó khăn, dài dòng và khó hiểu hơn. Chúng ta hãy so sánh 2 phương án:

Phương án dùng this trong phương thức ve_tam_giac:

```
void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    (*this).ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau); d3.ve_doan_thang(*this,mau);
}
```

phương án không dùng this trong phương thức ve_tam_giac:

```
void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    DIEM d1;
    d1.x = x; d1.y = y;
    d1.ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau);
    d3.ve_doan_thang(d1,mau);
}
```

IV. HÀM TẠO (CONSTRUCTOR)

1. Hàm tạo (hàm thiết lập)

Hàm tạo cũng là một phương thức của lớp (nhưng là hàm đặc biệt) dùng để tạo dựng một đối tượng mới. Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng sau đó sẽ gọi đến hàm tạo. Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc khác nhằm chuẩn bị cho đối tượng mới.

a. Cách viết hàm tạo

i. Điểm khác của hàm tạo và các phương thức thông thường:

Khi viết hàm tạo cần để ý 3 sự khác biệt của hàm tạo so với các phương thức khác như sau:

- Tên của hàm tạo: Tên của hàm tạo bắt buộc phải trùng với tên của lớp.
- Không khai báo kiểu cho hàm tạo.
- Hàm tạo không có kết quả trả về.

ii. Sự giống nhau của hàm tạo và các phương thức thông thường

Ngoài 3 điểm khác biệt trên, hàm tạo được viết như các phương thức khác:

- Hàm tạo có thể được xây dựng bên trong hoặc bên ngoài định nghĩa lớp.

- Hàm tạo có thể có đối hoặc không có đối.
- Trong một lớp có thể có nhiều hàm tạo (cùng tên nhưng khác bộ đối).

Ví dụ sau định nghĩa lớp DIEM_DH (Điểm đồ họa) có 3 thuộc tính:

```
int x;          // hoành độ (cột) của điểm
int y;          // tung độ (hàng) của điểm
int m;          // mầu của điểm
```

và đưa vào 2 hàm tạo để khởi gán cho các thuộc tính của lớp:

```
// Hàm tạo không đối: Dùng các giá trị cố định để khởi gán cho x, y, m
DIEM_DH() ;
// Hàm tạo có đối: Dùng các đối x1, y1, m1 để khởi gán cho x, y, m
DIEM_DH(int x1, int y1, int m1 = 15) ;      // Đối m1 có giá trị mặc định 15
class DIEM_DH                               // (mầu trắng)
{
private:
    int x, y, m ;
public:
    // Hàm tạo không đối: khởi gán cho x = 0, y = 0, m = 1
    // Hàm này viết bên trong định nghĩa lớp
    DIEM_DH()
    {
        x = y = 0;
        m = 1;
    }
    // Hàm tạo này xây dựng bên ngoài định nghĩa lớp
    DIEM_DH(int x1, int y1, int m1 = 15) ;
    // Các phương thức khác
};

// Xây dựng hàm tạo bên ngoài định nghĩa lớp
DIEM_DH:: DIEM_DH(int x1, int y1, int m1) ;
{
    x = x1; y = y1; m = m1;
}
```

b. Dùng hàm tạo trong khai báo

+ Khi đã xây dựng các hàm tạo, ta có thể dùng chúng trong khai báo để tạo ra một đối tượng đồng thời khởi gán cho các thuộc tính của đối tượng được tạo. Dựa vào các tham số trong khai báo mà trình biên dịch sẽ biết cần gọi đến hàm tạo nào.
+ Khi khai báo một biến đối tượng có thể sử dụng các tham số để khởi gán cho các thuộc tính của biến đối tượng.
+ Khi khai báo mảng đối tượng không cho phép dùng các tham số để khởi gán.
+ Câu lệnh khai báo một biến đối tượng sẽ gọi tới hàm tạo 1 lần.
+ Câu lệnh khai báo một mảng n đối tượng sẽ gọi tới hàm tạo n lần.

Ví dụ:

```
DIEM_DH d;                      // Gọi tới hàm tạo không đối.
// Kết quả d.x = 0, d.y = 0, d.m = 1
DIEM_DH u(300, 100, 5);         // Gọi tới hàm tạo có đối.
// Kết quả u.x = 300, u.y = 100, d.m = 5
DIEM_DH v(400, 200);            // Gọi tới hàm tạo có đối.
// Kết quả v.x = 400, v.y = 200, d.m = 15
DIEM_DH p[20] ;                 // Gọi tới hàm tạo không đối 20 lần
```

Chú ý: Với các hàm có đối kiểu lớp, thì đối chỉ xem là các tham số hình thức, vì vậy khai báo đối (trong dòng đầu của hàm) sẽ không tạo ra đối tượng mới và do đó không gọi tới các hàm tạo.

c. Dùng hàm tạo trong cấp phát bộ nhớ

+ Khi cấp phát bộ nhớ cho một đối tượng có thể dùng các tham số để khởi gán cho các thuộc tính của đối tượng, ví dụ

```
DIEM_DH *q = new DIEM_DH(40, 20, 4);        // Gọi tới hàm tạo có đối
// Kết quả q → x = 40, q → y = 20, q → m = 4
DIEM_DH *r = new DIEM_DH ;                  //Gọi tới hàm tạo không đối
// Kết quả r → x = 0, r → y = 0, r → m = 1
```

+ Khi cấp phát bộ nhớ cho một dãy đối tượng không cho phép dùng tham số để khởi gán, ví dụ:

```
int n = 30;
DIEM_DH *s = new DIEM_DH[n] ;     // Gọi tới hàm tạo không đối 30 lần.
```

d. Dùng hàm tạo để biểu diễn các đối tượng hằng

+ Như đã biết, sau khi định nghĩa lớp DIEM_DH thì có thể xem lớp này như một kiểu dữ liệu như int, double, char, ...

Với kiểu int chúng ta có các hằng int, như 253.

Với kiểu double chúng ta có các hằng double, như 75.42

Khái niệm hằng kiểu int, hằng kiểu double có thể mở rộng cho hằng kiểu DIEM_DH

+ Để biểu diễn một hằng đối tượng (hay còn gọi: Đối tượng hằng) chúng ta phải dùng tới hàm tạo. Mẫu viết như sau:

Tên_lớp(danh sách tham số) ;

Ví dụ đối với lớp DIEM_DH nói trên, có thể viết như sau:

```
DIEM_DH(234, l 23, 4)      // Biểu thị một đối tượng kiểu DIEM_DH
                           // có các thuộc tính x = 234, y = 123, m = 4
```

Chú ý: Có thể sử dụng một hằng đối tượng như một đối tượng. Nói cách khác, có thể dùng hằng đối tượng để thực hiện một phương thức, ví dụ nếu viết:

```
DIEM_DH(234, l 23, 4).in();
```

thì có nghĩa là thực hiện phương thức in() đối với hằng đối tượng.

e. Ví dụ minh họa

Chương trình sau đây minh họa cách xây dựng hàm tạo và cách sử dùng hàm tạo trong khai báo, trong cấp phát bộ nhớ và trong việc biểu diễn các hằng đối tượng.

```
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Hàm bạn dùng để in đối tượng DIEM_DH
    friend void in(DIEM_DH d)
    {
        cout <<"\n " << d.x << " "<< d.y<<" " << d.m ;
    }
    // Phương thức dùng để in đối tượng DIEM_DH
```

```
        void in()
        {
                cout <<"\n " << x << " "<< y<<" " << m ;
        }
        // Hàm tạo không đối
        DIEM_DH()
        {
                x = y = 0;
                m = 1;
        }
        // Hàm tạo có đối, đối m1 có giá trí mặc định là 15 (mầu trắng)
        DIEM_DH(int x1, int y1, int m1 = 15);
};

// Xây dựng hàm tạo
DIEM_DH::DIEM_DH(int x1, int y1, int m1)
{
        x = x1; y = y1; m = m1;
}

void main()
{
        DIEM_DH d1;                     // Gọi tới hàm tạo không đối
        DIEM_DH d2(200, 200, 10);       // Gọi tới hàm tạo có đối
        DIEM_DH*d;
        d = new DIEM_DH(300, 300);      // Gọi tới hàm tạo có đối
        clrscr();
        in(d1);                         //Gọi hàm bạn in()
        d2.in();                        //Gọi phương thức in()
        in(*d);                         // Gọi hàm bạn in()
        DIEM_DH(2, 2, 2).in();          // Gọi phương thức in()
        DIEM_DH t[3];                   // 3 lần gọi hàm tạo không đối
        DIEM_DH*q;                      // Gọi hàm tạo không đối
        int n;
```

```
        cout << "\n N = "; cin >> n;
        q = new DIEM_DH[n+1];          // (n+1) lần gọi hàm tạo không đối
        for (int i = 0; i< = n; ++i)
            q[i] = DIEM_DH(300+i, 200+i, 8); //(n+1) lần gọi hàm tạo có đối
        for (i = 0; i< = n; ++i)
            q[i].in();                          // Gọi phương thức in()
        for (i = 0; i< = n; ++i)
            DIEM_DH(300+i, 200+i, 8).in();      // Gọi phương thức in()
        getch();
}
```

2. Lớp không có hàm tạo và hàm tạo mặc định

a. Nếu lớp không có hàm tạo

Chương trình dịch sẽ cung cấp một hàm tạo mặc định không đối (default), hàm này thực chất không làm gì cả. Như vậy một đối tượng tạo ra chỉ được cấp phát bộ nhớ, còn các thuộc tính của nó chưa được xác định. Chúng ta có thể kiểm chứng điều này, bằng cách chạy chương trình sau:

```
// Hàm tạo mặc định
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức
    void in() { cout <<"\n " << x << " "<< y<<" " << m ; }
};

void main()
{
    DIEM_DH d;
    d.in();
    DIEM_DH *p;
```

```
    p = new DIEM_DH[10];
    clrscr();
    d.in();
    for (int i = 0; i<10; ++i) (p+i)->in();
    getch();
}
```

b. Nếu trong lớp đã có ít nhất một hàm tạo

Khi đó hàm tạo mặc định sẽ không được phát sinh nữa và mọi câu lệnh xây dựng đối tượng mới đều sẽ gọi đến một hàm tạo của lớp. Nếu không tìm thấy hàm tạo cần gọi thì chương trình dịch sẽ báo lỗi. Điều này thường xẩy ra khi chúng ta không xây dựng hàm tạo không đối, nhưng lại sử dụng các khai báo không tham số như ví dụ sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức dùng để in đối tượng DIEM_DH
    void in()
    {
        cout <<"\n" << x << " "<< y<<" " << m ;
    }
    //Hàm tạo có đối
    DIEM_DH::DIEM_DH(int x1, int y1 , int m1)
    {
        x = x1; y = y1 ; m = m1;
    }
};

void main()
{
    DIEM_DH d1(200, 200, 10); // Gọi tới hàm tạo có đối
```

```
    DIEM_DH d2; // Gọi tới hàm tạo không đối
    d2 = DIEM_DH(300, 300, 8); // Gọi tới hàm tạo có đối
    d1.in();
    d2.in();
    getch();
}
```

Trong các câu lệnh trên, chỉ có câu lệnh thứ 2 trong hàm main() là bị báo lỗi. Câu lệnh này sẽ gọi tới hàm tạo không đối, mà hàm này chưa được xây dựng.

Giải pháp: có thể chọn một trong 2 giải pháp sau:

- Xây dựng thêm hàm tạo không đối.
- Gán giá trị mặc định cho tất cả các đối x1, y1 và m1 của hàm tạo đã xây dựng ở trên.

Theo phương án 2, chương trình có thể sửa như sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x, y, m;
public:
    // Phương thức dùng để in đối tượng DIEM_DH
    void in() { cout <<"\n " << x << " "<< y<<" " << m ; }
    // Hàm tạo có đối , tất cả các đối đều có giá trị mặc định
    DIEM_DH::DIEM_DH(int x1 = 0, int y1 = 0, int m1 = 15)
    {
        x = x1; y = y1; m = m1;
    }
};

void main()
{
    // Gọi tới hàm tạo, không dùng tham số mặc định
    DIEM_DH d1(200, 200, 10);
    // Gọi tới hàm tạo, dùng 3 tham số mặc định
```

```
    DIEM_DH d2;
    // Gọi tới hàm tạo, dùng 1 tham số mặc định
    d2 = DIEM_DH(300, 300);
    d1.in();
    d2.in();
    getch();
}
```

3. Hàm tạo sao chép (Copy Constructor)

a. Hàm tạo sao chép mặc định

Giả sử đã định nghĩa một lớp nào đó, ví dụ lớp PS (phân số). Khi đó:

+ Ta có thể dùng câu lệnh khai báo hoặc cấp phát bộ nhớ để tạo các đối tượng mới, ví dụ:

```
PS p1, p2 ;
PS *p = new PS ;
```

+ Ta cũng có thể dùng lệnh khai báo để tạo một đối tượng mới từ một đối tượng đã tồn tại, ví dụ:

```
PS u;
PS v(u) ; // Tạo v theo u
```

ý nghĩa của câu lệnh này như sau:

– Nếu trong lớp PS chưa xây dựng hàm tạo sao chép, thì câu lệnh này sẽ gọi tới một hàm tạo sao chép mặc định (của C++). Hàm này sẽ sao chép nội dung từng bit của u vào các bit tương ứng của v. Như vậy các vùng nhớ của u và v sẽ có nội dung như nhau. Rõ ràng trong đa số các trường hợp, nếu lớp không có các thuộc tính kiểu con trỏ hay tham chiếu, thì việc dùng các hàm tạo sao chép mặc định (để tạo ra một đối tượng mới có nội dung như một đối tượng cho trước) là đủ và không cần xây dựng một hàm tạo sao chép mới.

– Nếu trong lớp PS đã có hàm tạo sao chép (cách viết sẽ nói sau) thì câu lệnh: PS v(u); sẽ tạo ra đối tượng mới v, sau đó gọi tới hàm tạo sao chép để khởi gán v theo u.

Ví dụ sau sẽ minh họa cách dùng hàm tạo sao chép mặc định:

Trong chương trình đưa vào lớp PS (phân số):

+ Các thuộc tính gồm: t (tử số) và m (mẫu).

+ Trong lớp không có phương thức nào cả mà chỉ có 2 hàm bạn là các hàm toán tử nhập (>>) và xuất (<<).

+ Nội dung chương trình là: Dùng lệnh khai báo để tạo một đối tượng u (kiểu PS) có nội dung như đối tượng đã có d.

```
// Ham tao sao chep mac dinh
#include <conio.h>
#include <iostream.h>
class PS
{
private:
	int t, m ;
public:
	friend ostream& operator<< (ostream&os, const PS &p)
	{
		os << " = " << p.t << "/" << p.m;
		return os;
	}
	friend istream& operator>> (istream& is, PS &p)
	{
		cout << "\n Nhap tu va mau: " ;
		is >> p.t >> p.m ;
		return is;
	}
} ;

void main()
{
	PS d;
	cout << "\n Nhap PS d "; cin >> d;
	cout << "\n PS d " << d;
	PS u(d);
	cout << "\n PS u "<< u;
	getch();
}
```

b. Cách xây dựng hàm tạo sao chép

+ Hàm tạo sao chép sử dụng một đối kiểu tham chiếu đối tượng để khởi gán

cho đối tượng mới. Hàm tạo sao chép được viết theo mẫu:

```
Tên_lớp (const Tên_lớp & dt)
{
        // Các câu lệnh dùng các thuộc tính của đối tượng dt
        // để khởi gán cho các thuộc tính của đối tượng mới
}
```

+ Ví dụ có thể xây dựng hàm tạo sao chép cho lớp PS như sau:

```
class PS
{
private:
        int t, m ;
public:
        PS (const PS &p)
        {
                this->t = p.t ;
                this->m = p.m ;
        }
        ...
} ;
```

c. Khi nào cần xây dựng hàm tạo sao chép

+ Nhận xét: Hàm tạo sao chép trong ví dụ trên không khác gì hàm tạo sao chép mặc định.
+ Khi lớp không có các thuộc tính kiểu con trỏ hoặc tham chiếu, thì dùng hàm tạo sao chép mặc định là đủ.
+ Khi lớp có các thuộc tính con trỏ hoặc tham chiếu, thì hàm tạo sao chép mặc định chưa đáp ứng được yêu cầu.

Ví dụ:

```
class DT
{
private:
        int n; // Bac da thuc
        double *a; // Tro toi vung nho chua cac he so da thuc a0, a1, ...
public:
```

```
        DT() { this->n0; this->a = NULL; }
        DT(int n1)
        {
            this->n = n1;
            this->a = new double[n1+1];
        }
        friend ostream& operator << (ostream& os, const DT &d);
        friend istream& operator>> (istream& is, DT &d);
        ...
    } ;
```

Bây giờ chúng ta hãy theo dõi xem việc dùng hàm tạo mặc định trong đoạn chương trình sau sẽ dẫn đến sai lầm như thế nào:

```
        DT d ; // Tạo đối tượng d kiểu DT
        cin >> d ;
```

/* Nhập đối tượng d, gồm: nhập một số nguyên dương và gán cho d.n, cấp phát vùng nhớ cho d.a, nhập các hệ số của đa thức và chứa vào vùng nhớ được cấp phát */

```
        DT u(d);
```

/* Dùng hàm tạo mặc định để xây dựng đối tượng u theo d. Kết quả: u.n = d.n và u.a = d.a. Như vậy 2 con trỏ u.a và d.a cùng trỏ đến một vùng nhớ */

Nhận xét: Mục đích là tạo ra một đối tượng u giống như d, nhưng độc lập với d. Nghĩa là khi d thay đổi thì u không bị ảnh hưởng gì. Thế nhưng mục tiêu này không đạt được, vì u và d có chung một vùng nhớ chứa hệ số của đa thức, nên khi sửa đổi các hệ số của đa thức trong d thì các hệ số của đa thức trong u cũng thay đổi theo. Còn một trường hợp nữa cũng dẫn đến lỗi là khi một trong 2 đối tượng u và d bị giải phóng (thu hồi vùng nhớ chứa đa thức) thì đối tượng còn lại cũng sẽ không còn vùng nhớ nữa.

Ví dụ sau sẽ minh họa nhận xét trên: Khi d thay đổi thì u cũng thay đổi và ngược lại khi u thay đổi thì d cũng thay đổi theo.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
```

```
        double *a; // Tro tơi vung nho chua cac he so da thuc a0, a1 , ...
public:
        DT() { this->n = 0; this->a = NULL; }
        DT(int n1)
        {
                this->n = n1 ;
                this->a = new double[n1+1];
        }
        friend ostream& operator<< (ostream& os, const DT &d);
        friend istream& operator>> (istream& is, DT &d);
} ;

ostream& operator<< (ostream& os, const DT &d)
{
        os << " Cac he so (tu ao): ";
        for (int i = 0 ; i< = d.n ; ++i)
        os << d.a[i] <<" " ;
        return os;
}

istream& operator >> (istream& is, DT &d)
{
        if (d.a! = NULL) delete d.a;
        cout << " \n Bac da thuc: " ;
        cin >> d.n;
        d.a = new double[d.n+1];
        cout << "Nhap cac he so da thuc:\n" ;
        for (int i = 0 ; i< = d.n ; ++i)
        {
                cout << "He so bac "<< i << " = " ;
                is >> d.a[i] ;
        }
        return is;
}
```

```
void main()
{
        DT d;
        clrscr();
        cout <<"\n Nhap da thuc d " ; cin >> d;
        DT u(d);
        cout << "\n Da thuc d "<< d ;
        cout << "\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc d " ; cin >> d;
        cout << "\nDa thuc d " << d;
        cout <<"\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc u " ; cin >> u;
        cout << "\n Da thuc d "<< d ;
        cout << "\n Da thuc u " << u ;
        getch();
}
```

d. Ví dụ về hàm tạo sao chép

Trong chương trình trên đã chỉ rõ: Hàm tạo sao chép mặc định là chưa thoả mãn đối với lớp DT. Vì vậy cần viết hàm tạo sao chép để xây dựng đối tượng mới (ví dụ u) từ một đối tượng đang tồn tại (ví dụ d) theo các yêu cầu sau:

+ Gán d.n cho u.n
+ Cấp phát một vùng nhớ cho u.a để có thể chứa được (d.n + 1) hệ số.
+ Gán các hệ số chứa trong vùng nhớ của d.a sang vùng nhớ của u.a

Như vậy chúng ta sẽ tạo được đối tượng u có nội dung ban đầu giống như d, nhưng độc lập với d.

Để đáp ứng các yêu cầu nêu trên, hàm tạo sao chép cần được xây dựng như sau:

```
DT::DT(const DT &d)
{
        this → n = d.n ;
        this → a = new double[d.n+1];
        for (int i = 0; i< = d.n; ++i)
        this → a[i] = d.a[i];
}
```

Chương trình sau sẽ minh họa điều này: Sự thay đổi của d không làm ảnh hưởng đến u và ngược lại sự thay đổi của u không làm ảnh hưởng đến d.

```
// Viết hàm tạo sao chép cho lớp DT
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
    double *a; // Tro toi vung nho chua cac he so da thuc a0, a1 , ...
public:
    DT() { this → n = 0; this → a = NULL; }
    DT(int n1)
    {
        this → n = n1 ;
        this → a = new double[n1+1];
    }
    DT(const DT &d);
    friend ostream& operator<< (ostream& os, const DT&d);
    friend istream& operator>> (istream& is, DT&d);
};

DT::DT(const DT&d)
{
    this → n = d.n;
    this → a = new double[d.n+1];
    for (int i = 0; i< = d.n; ++i)
    this → a[i] = d.a[i];
}

ostream& operator<< (ostream& os, const DT &d)
{
    os << " Cac he so (tu ao): " ;
```

```
        for (int i = 0 ; i< = d.n ; ++i) os << d.a[ i] <<" " ;
        return os;
}

istream& operator>> (istream& is, DT &d)
{
        if (d.a! = NULL) delete d.a;
        cout << "\n Bac da thuc: " ;
        cin >> d.n;
        d.a = new double[d.n+1];
        cout << "Nhap cac he so da thuc:\n" ;
        for (int i = 0 ; i< = d.n ; ++i)
        {
                cout << "He so bac " << i << " = " ;
                is >> d.a[i] ;
        }
        return is;
}

void main()
{
        DT d;
        clrscr();
        cout <<"\n Nhap da thuc d " ; cin >> d;
        DT u(d);
        cout <<"\n Da thuc d " << d ;
        cout << "\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc d " ; cin >> d;
        cout << "\n Da thuc d "<< d ;
        cout <<"\n Da thuc u " << u ;
        cout <<"\n Nhap da thuc u " ; cin >> u;
        cout << "\n Da thuc d " << d ;
        cout <<"\n Da thuc u " << u ;
        getch();
```

```
}
```

V. HÀM HỦY (DESTRUCTOR)

Hàm hủy là một hàm thành viên của lớp (phương thức) có chức năng ngược với hàm tạo. Hàm hủy được gọi trước khi giải phóng (xoá bỏ) một đối tượng để thực hiện một số công việc có tính "dọn dẹp" trước khi đối tượng được hủy bỏ, ví dụ như giải phóng một vùng nhớ mà đối tượng đang quản lý, xoá đối tượng khỏi màn hình nếu như nó đang hiển thị, ...

Việc hủy bỏ một đối tượng thường xẩy ra trong 2 trường hợp sau:

+ Trong các toán tử và các hàm giải phóng bộ nhớ, như delete, free, ...
+ Giải phóng các biến, mảng cục bộ khi thoát khỏi hàm, phương thức.

1. Hàm hủy mặc định

Nếu trong lớp không định nghĩa hàm hủy, thì một hàm hủy mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm hủy mặc định là đủ, và không cần đưa vào một hàm hủy mới.

2. Quy tắc viết hàm hủy

Mỗi lớp chỉ có một hàm hủy viết theo các quy tắc sau:

+ Kiểu của hàm: Hàm hủy cũng giống như hàm tạo là hàm không có kiểu, không có giá trị trả về.
+ Tên hàm: Tên của hàm hủy gồm một dấu ngã (đứng trước) và tên lớp:

  ~Tên_lớp
+ Đối: Hàm hủy không có đối

Ví dụ có thể xây dựng hàm hủy cho lớp DT (đa thức) như sau:

```
class DT
{
private:
    int n; // Bac da thua
    double *a; // Tro toi vung nho chua cac he so da thuc a0, a1 , ...
public:
    ~DT()
    {
        this → n = 0;
        delete this → a;
```

```
        }
    ...
};
```

3. Vai trò của hàm hủy trong lớp DT

Trong phần trước định nghĩa lớp DT (đa thức) khá đầy đủ gồm:

+ Các hàm tạo
+ Các toán tử nhập >>, xuất <<
+ Các hàm toán tử thực hiện các phép tính +, -, *, /

Tuy nhiên vẫn còn thiếu hàm hủy để giải phóng vùng nhớ mà đối tượng kiểu DT (cần hủy) đang quản lý.

Chúng ta hãy phân tích các khiếm khuyết của chương trình này:

+ Khi chương trình gọi tới một phương thức toán tử để thực hiện các phép tính cộng, trừ, nhân đa thức, thì một đối tượng trung gian được tạo ra. Một vùng nhớ được cấp phát và giao cho nó (đối tượng trung gian) quản lý.
+ Khi thực hiện xong phép tính sẽ ra khỏi phương thức. Đối tượng trung gian bị xoá, tuy nhiên chỉ vùng nhớ của các thuộc tính của đối tượng này được giải phóng. Còn vùng nhớ (chứa các hệ số của đa thức) mà đối tượng trung gian đang quản lý thì không hề bị giải phóng. Như vậy số vùng nhớ bị chiếm dụng vô ích sẽ tăng lên.

Nhược điểm trên dễ dàng khắc phục bằng cách đưa vào lớp DT hàm hủy trong mục 3 ở trên.

4. Ví dụ

Phần này chúng tôi trình bày một ví dụ tương đối hoàn chỉnh về lớp các hình tròn trong chế độ đồ họa. Chương trình gồm:

i. Lớp HT (hình tròn) với các thuộc tính:

```
int r;                  // Bán kính
int m;                  // Mầu hình tròn
int xhien, yhien;       // Vị trí hiển thị hình tròn trên màn hình
char *pht;              // Con trỏ trỏ tới vùng nhớ chứa ảnh hình tròn
int hienmh;             // Trạng thái hiện (hienmh = 1), ẩn (hienmh = 0)
```

ii. Các phương thức

+ Hàm tạo không đối thực hiện việc gán giá trị bằng 0 cho các thuộc tính của lớp. HT();
+ Hàm tạo có đối. HT(int n, int m1 = 15);

Thực hiện các việc:

– Gán r1 cho r, m1 cho m

– Cấp phát bộ nhớ cho pht

– Vẽ hình tròn và lưu ảnh hình tròn vào vùng nhớ của pht

+ Hàm hủy: ~HT();

Thực hiện các việc:

– Xoá hình tròn khỏi màn hình (nếu đang hiển thị)

– Giải phóng bộ nhớ đã cấp cho pht

+ Phương thức: void hien(int x, int y);

Có nhiệm vụ hiển thị hình tròn tại (x, y)

+ Phương thức : void an()

Có nhiệm vụ làm ẩn hình tròn

iii. Các hàm độc lập:

```
void ktdh();                    // Khởi tạo đồ họa
void ve_bau_troi();             // Vẽ bầu trời sao
void ht_di_dong_xuong();        // Vẽ một cặp 2 hình tròn di chuyển xuống
void ht_di_dong_len();          // Vẽ một cặp 2 hình tròn di chuyển lên trên
```

Nội dung chương trình là tạo ra các chuyển động xuống và lên của các hình tròn.

```
// Lop do hoa
// Ham huy
// Trong ham huy co the goi PT khac
#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
void ktdh();                    // Khởi tạo đồ họa
void ve_bau_troi();             // Vẽ bầu trời sao
void ht_di_dong_xuong();        // Vẽ một cặp 2 hình tròn di chuyển xuống
void ht_di_dong_len();          // Vẽ một cặp 2 hình tròn di chuyển lên trên
```

```
int xmax, ymax;
class HT
{
private:
        int r, m ;
        int xhien, yhien;
        char *pht;
        int hienmh;
public:
        HT();
        HT(int n, int m1 = 15);
        ~HT();
        void hien(int x, int y);
        void an();
};

HT:: HT()
{
        r = m = hienmh = 0;
        xhien = yhien = 0;
        pht = NULL;
}

HT::HT(int n, int m1)
{
        r = n; m = m1; hienmh = 0;
        xhien = yhien = 0;
        if (r<0) r = 0;
        if (r = = 0) pht = NULL;
        else
        {
                int size; char *pmh;
                size = imagesize(0, 0, r+r, r+r);
                pmh = new char[size];
```

```
            getimage(0, 0, r+r, r+r, pmh);
            setcolor(m);
            circle(r, r, r );
            setfillstyle(1, m);
            floodfill(r, r, m);
            pht = new char[size];
            getimage(0, 0, r+r, r+r, pht);
            putimage(0, 0, pmh, COPY_PUT);
            delete pmh;
            pmh = NULL;
        }
    }

    void HT::hien(int x, int y)
    {
        if (pmh! = NULL && !hienmh) // Chua hien
        {
            hienmh = 1;
            xhien = x; yhien = y;
            putimage(x, y, pht, XOR_PUT);
        }
    }

    void HT::an()
    {
        if (hienmh) // Dang hien
        {
            hienmh = 0;
            putimage(xhien, yhien, pht, XOR_PUT);
        }
    }

    HT::~HT()
    {
        an();
```

```
        if (pht! = NULL)
        {
                delete pht;
                pht = NULL;
        }
}

void ktdh()
{
        int mh = 0, mode = 0;
        initgraph(&mh, &mode, " ");
        xmax = getmaxx();
        ymax = getmaxy();
}

void ve_bau_troi()
{
        for (int i = 0; i<2000; ++i)
        putpixel(random(xmax), random(ymax), 1+random( 15));
}

void ht_di_dong_xuong()
{
        HT h(50, 4);
        HT u(60, 15);
        h.hien(0, 0);
        u.hien(40, 0);
        for (int x = 0; x< = 340; x+ = 10)
        {
                h.an();
                u.an();
                h.hien(x, x);
                delay(200);
                u.hien(x+40, x);
                delay(200);
```

```
        }
}

void ht_di_dong_len()
{
        HT h(50, 4);
        HT u(60, 15);
        h.hien(340, 340);
        u.hien(380, 340);
        for (int x = 340; x> = 0; x- = 10)
        {
                h.an();
                u.an();
                u.hien(x, x);
                delay(200);
                u.hien(x+40, x);
                delay(200);
        }
};

void main()
{
        ktdh();
        ve_bau_troi();
        ht_di_dong_xuong();
        ht_di_dong_len();
        getch();
        closegraph();
}
```

Các nhận xét:

+ Trong thân hàm hủy gọi tới phương thức an().
+ Điều gì xẩy ra khi bỏ đi hàm hủy:
  - Khi gọi hàm ht_di_dong_xuong() thì có 2 đối tượng kiểu HT được tạo ra. Trong thân hàm sử dụng các đối tượng này để vẽ các hình tròn di

chuyển xuống. Khi thoát khỏi hàm thì 2 đối tượng (tạo ra ở trên) được giải phóng. Vùng nhớ của các thuộc tính của chúng bị thu hồi, nhưng vùng nhớ cấp phát cho thuộc tính pht chưa được giải phóng và ảnh của 2 hình tròn (ở phía dưới màn hình) vẫn không được cất đi.

- Điều tương tự xẩy ra sau khi ra khỏi hàm ht_di_dong_len(): vùng nhớ cấp phát cho thuộc tính pht chưa được giải phóng và ảnh của 2 hình tròn (ở phía trên màn hình) vẫn không được thu dọn.

VI. CÁC HÀM TRỰC TUYẾN (INLINE)

Một số mở rộng của C++ đối với C đã được trình bày trong các chương trước như biến tham chiếu, định nghĩa chồng hàm, hàm với đối mặc định … Phần này ta xem một đặc trưng khác của C++ được gọi là hàm trực tuyến (inline).

1. Ưu nhược điểm của hàm

Việc tổ chức chương trình thành các hàm có 2 ưu điểm rõ rệt:

Thứ nhất là chia chương trình thành các đơn vị độc lập, làm cho chương trình được tổ chức một cách khoa học dễ kiểm soát, dễ phát hiện lỗi, dễ phát triển và mở rộng.

Thứ hai là giảm được kích thước chương trình, vì mỗi đoạn chương trình thực hiện nhiệm vụ của hàm được thay bằng một lời gọi hàm.

Tuy nhiên hàm cũng có nhược điểm là làm chậm tốc độ chương trình do phải thực hiện một số thao tác có tính thủ tục mỗi khi gọi hàm như: cấp phát vùng nhớ cho các đốivà biến cục bộ, truyền dữ liệu của các tham số cho các đối, giải phóng vùng nhớ trước khi thoát khỏi hàm.

Các hàm trực tuyến trong C++ có khả năng khắc phục được các nhược điểm nói trên.

2. Các hàm trực tuyến

Để biến một hàm thành trực tuyến ta viết thêm từ khoá **inline** vào trước khai báo nguyên mẫu hàm. Nếu không dùng nguyên mẫu thì viết từ khoá này trước dòng đầu tiên của định nghĩa hàm.

Ví dụ 1 :

```
inline float f(int n, float x);
float f(int n, float x)
{
      // Các câu lệnh trong thân hàm
}
```

hoặc

```
inline float f(int n, float x)
{
    // Các câu lệnh trong thân hàm
}
```

Chú ý: Trong mọi trường hợp, từ khoá inline phải xuất hiện trước các lời gọi hàm thì trình biên dịch mới biết cần xử lý hàm theo kiểu inline.

Ví dụ hàm f trong chương trình sau sẽ không phải là hàm trực tuyến vì từ khoá inline viết sau lời gọi hàm:

```
#include <conio.h>
#include <iostream.h>
void main()
{
    int s ;
    s = f(5,6);
    cout << s ;
    getch();
}
inline int f(int a, int b)
{
    return a*b;
}
```

Chú ý: Trong C^{++}, nếu hàm được xây dựng sau lời gọi hàm thì bắt buộc phải khai báo nguyên mẫu hàm trước lời gọi. Trong ví dụ trên, trình biên dịch C^{++} sẽ bắt lỗi vì thiếu khai báo nguyên ngẫu hàm f .

3. Cách biên dịch và dùng hàm trực tuyến

Chương trình dịch xử lý các hàm inline như các macro (được định nghĩa trong lệnh #define), nghĩa là nó sẽ thay mỗi lời gọi hàm bằng một đoạn chương trình thực hiện nhiệm vụ của hàm. Cách này làm cho chương trình dài ra, nhưng tốc độ chương trình tăng lên do không phải thực hiện các thao tác có tính thủ tục khi gọi hàm.

Phương án dùng hàm trực tuyến rút ngắn được thời gian chạy máy nhưng lại làm tăng khối lượng bộ nhớ chương trình (nhất là đối với các hàm trực tuyến có nhiều câu lệnh). Vì vậy chỉ nên dùng phương án trực tuyến đối với các hàm nhỏ.

4. Sự hạn chế của trình biên dịch

Không phải khi gặp từ khoá inline là trình biên dịch nhất thiết phải xử lý hàm theo kiểu trực tuyến.

Có một số hàm mà các trình biên dịch thường không xử lý theo cách inline như các hàm chứa biến static, hàm chứa các lệnh chu trình hoặc lệnh goto hoặc lệnh switch, hàm đệ quy. Trong trường hợp này từ khoá inline lẽ dĩ nhiên bị bỏ qua.

Thậm chí từ khoá inline vẫn bị bỏ qua ngay cả đối với các hàm không có những hạn chế nêu trên nếu như trình biên dịch thấy cần thiết (ví dụ đã có quá nhiều hàm inline làm cho bộ nhớ chương trình quá lớn)

Ví dụ 2 : Chương trình sau sử dụng hàm inline tính chu vi và diện tích của hình chữ nhật:

Cách 1: Không khai báo nguyên mẫu. Khi đó hàm dtcvhcn phải đặt trước hàm main.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv)
{
	dt=a*b;
	cv=2*(a+b);
}
void main()
{
	int a[20],b[20],cv[20],dt[20],n;
	cout << "\n So hinh chu nhat: " ;
	cin >> n;
	for (int i=1; i<=n; ++i)
	{
		cout <<"\n Nhap 2 canh cua hinh chu nhat thu " << i << ": ";
		cin >> a[i] >> b[i];
		dtcvhcn(a[i],b[i],dt[i], cv[i]);
	}
	clrscr();
	for (i=1; i<=n; ++i)
	{
		cout << "\n Hinh chu nhat thu "<< i << " : ";
```

```
            cout << "\n Do dai 2 canh= " << a[i] << " va " << b[i] ;
            cout <<"\n Dien tich= " << dt[i] ;
            cout << "\n Chu vi= " << cv[i] ;
        }
        getch();
    }
```

Cách 2:Sử dụng khai báo nguyên mẫu. Khi đó từ khoá inline đặt trước nguyên mẫu.

Chú ý: Không được đặt inline trước định nghĩa hàm. Trong chương trình dưới đây nếu đặt inline trước định nghĩa hàm thì hậu quả như sau: Chương trình vẫn dịch thông, nhưng khi chạy thì chương trình bị quẩn và không thoát đi được.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv);
void main()
{
    int a[20],b[20],cv[20],dt[20],n;
    cout << "\n So hinh chu nhat: " ;
    cin >> n;
    for (int i=1; i<=n; ++i)
    {
        cout <<"\n Nhap 2 canh cua hinh chu nhat thu " << i << ": ";
        cin >> a[i] >> b[i];
        dtcvhcn(a[i],b[i],dt[i], cv[i]);
    }
    clrscr();
    for (i=1; i<=n; ++i)
    {
        cout << "\n Hinh chu nhat thu "<< i << " : ";
        cout << "\n Do dai 2 canh= " << a[i] << " va " << b[i] ;
        cout <<"\n Dien tich= " << dt[i] ;
        cout << "\n Chu vi= " << cv[i] ;
    }
    getch();
```

```
}
void dtcvhcn(int a, int b, int&dt, int &cv)
{
	dt=a*b;
	cv=2*(a+b);
}
```

CHƯƠNG 8

HÀM BẠN, ĐỊNH NGHĨA PHÉP TOÁN CHO LỚP

Hàm bạn
Định nghĩa phép toán cho lớp

I. HÀM BẠN (FRIEND FUNCTION)

1. Hàm bạn

Để một hàm trở thành bạn của một lớp, có 2 cách viết:

Cách 1: Dùng từ khóa friend để khai báo hàm trong lớp và xây dựng hàm bên ngoài như các hàm thông thường (không dùng từ khóa friend). Mẫu viết như sau:

```
class A
{
private:
        // Khai báo các thuộc tính
public:
        ...
        // Khai báo các hàm bạn của lớp A
        friend void f1(...);
        friend double f2(...);
        friend A f3(...) ;
        ...
} ;
// Xây dựng các hàm f1, f2, f3
void f1(...)
{
        ...
}
double f2(...)
{
```

```
        ...
}
A f3(...)
{
        ...
}
```

Cách 2: Dùng từ khóa friend để xây dựng hàm trong định nghĩa lớp. Mẫu viết như sau:

```
class A
{
private:
        // Khai báo các thuộc tính
public:
        // Xây dựng các hàm bạn của lớp A
        void f1(...)
        {
                ...
        }
        double f2(...)
        {
                ...
        }
        A f3(...)
        {
                ...
        }
        ...
} ;
```

2. Tính chất của hàm bạn

Trong thân hàm bạn của một lớp có thể truy nhập tới các thuộc tính của các đối tượng thuộc lớp này. Đây là sự khác nhau duy nhất giữa hàm bạn và hàm thông thường.

Chú ý rằng hàm bạn không phải là phương thức của lớp. Phương thức có một

đối ẩn (ứng với con trỏ this) và lời gọi của phương thức phải gắn với một đối tượng nào đó (địa chỉ đối tượng này được truyền cho con trỏ this). Lời gọi của hàm bạn giống như lời gọi của hàm thông thường.

Ví dụ sau sẽ so sánh phương thức, hàm bạn và hàm thông thường.

Xét lớp SP (số phức), hãy so sánh 3 phương án để thực hiện việc cộng 2 số phức:

Phương án 1: Dùng phương thức

```
class SP
{
private:
    double a; // phần thực
    double b; // Phần ảo
public:
    SP cong(SP u2)
    {
        SP u:
        u.a = this → a + u2.a ;
        u.b = this → b + u2.b ;
        return u;
    }
};
```

Cách dùng:

```
SP u, u1, u2;
u = u1.cong(u2);
```

Phương án 2: Dùng hàm bạn

```
class SP
{
private:
    double a; // Phần thực
    double b; // Phần ảo
public:
    friend SP cong(SP u1 , SP u2)
```

```
    {
        SP u:
        u.a = u1.a + u2.a ;
        u.b = u1.b + u2.b ;
        return u;
    }
};
```

Cách dùng

```
SP u, u1, u2;
u = cong(u1, u2);
```

Phương án 3: Dùng hàm thông thường

```
class SP
{
private:
    double a; // phần thực
    double b; // Phần ảo
public:
    ...
};

SP cong(SP u1, SP u2)
{
    SP u:
    u.a = u1.a + u2.a ;
    u.b = u1.b + u2.b ;
    return u;
}
```

Phương án này không được chấp nhận, trình biên dịch sẽ báo lỗi trong thân hàm không được quyền truy xuất đến các thuộc tính riêng (private) a, b của các đối tượng u, u1 và u2 thuộc lớp SP.

3. Hàm bạn của nhiều lớp

Khi một hàm là bạn của nhiều lớp, thì nó có quyền truy nhập tới tất cả các thuộc tính của các đối tượng trong các lớp này.

Để làm cho hàm f trở thành bạn của các lớp A, B và C ta sử dụng mẫu viết như sau:

```
class A;          // Khai báo trước lớp A
class B;          // Khai báo trước lớp B
class C;          // Khai báo trước lớp C
// Định nghĩa lớp A
class A
{
      // Khai báo f là bạn của A
      friend void f(...) ;
} ;
// Định nghĩa lớp B
class B
{
      // Khai báo f là bạn của B
      friend void f(...) ;
} ;
// Định nghĩa lớp C
class C
{
      // Khai báo f là bạn của C
      friend void f(...) ;
} ;
// Xây dựng hàm f
void f(...)
{
      ...
}
```

Chương trình sau đây minh họa cách dùng hàm bạn (bạn của một lớp và bạn của nhiều lớp). Chương trình đưa vào 2 lớp VT (véc tơ), MT (ma trận) và 3 hàm bạn để thực hiện các thao tác trên 2 lớp này:

```
// Hàm bạn với lớp VT dùng để in một véc tơ
      friend void in(const VT &x);
// Hàm bạn với lớp MT dùng để in một ma trận
      friend void in(const MT &a);
```

```
// Hàm bạn với cả 2 lớp MT và VT dùng để nhân ma trận với véc tơ
    friend VT tich(const MT &a, const VT &x);
```

Nội dung chương trình là nhập một ma trận vuông cấp n và một véc tơ cấp n, sau đó thực hiện phép nhân ma trận với véc tơ vừa nhập.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
class VT;
class MT;
class VT
{
private:
    int n;
    double x[20]; // Toa do cua diem
public:
    void nhapsl();
    friend void in(const VT &x);
    friend VT tich(const MT &a, const VT &x) ;
};

class MT
{
private:
    int n;
    double a[20][20];
public:
    friend VT tich(const MT &a, const VT &x);
    friend void in(const MT &a);
    void nhapsl();
} ;

void VT::nhapsl()
{
    cout << "\n Cap vec to = ";
```

```
        cin >> n ;
        for (int i = 1; i< = n ; ++i)
        {
                cout << "\n Phan tu thu " << i <<" = " ;
                cin >> x[i];
        }
}

void MT::nhapsl()
{
        cout <<"\n Cap ma tran = ";
        cin >> n ;
        for (int i = 1; i< = n ; ++i)
        for (int j = 1; j< = n; ++j)
        {
                cout << "\n Phan tu thu: "<<i<< " hang "<< i << " cot " << j << " = ";
                cin >> a[i][j];
        }
}

VT tich(const MT &a, const VT &x)
{
        VT y;
        int n = a.n;
        if (n! = x.n)
        return x;
        y.n = n;
        for (int i = 1; i< = n; ++i)
        {
                y.x[i] = 0;
                for (int j = 1; j< = n; ++j)
                y.x[i] = a.a[i][j]*x.x[j];
        }
        return y;
```

```
}
void in(const VT &x)
{
	cout << "\n";
	for (int i = 1; i< = x.n; ++i)
	cout << x.x[i] << " ";
}
void in(const MT &a)
{
	for (int i = 1; i< = a.n; ++i)
	{
		cout << "\n " ;
		for (int j = 1; j< = a.n; ++j)
		cout << a.a[i][j] << " ";
	}
}
void main()
{
	MT a; VT x, y;
	clrscr();
	a.nhapsl();
	x.nhapsl();
	y = tich(a, x);
	clrscr();
	cout << "\n Ma tran A:";
	in(a);
	cout << "\n Vec to x: " ;
	in(x);
	cout << "\n Vec to y = Ax: " ;
	in(y);
	getch();
}
```

II. ĐỊNH NGHĨA PHÉP TOÁN CHO LỚP

Đối với mỗi lớp ta có thể sử dụng lại các kí hiệu phép toán thông dụng (+, -, *, …) để định nghĩa cho các phép toán của lớp. Sau khi được định nghĩa các kí hiệu này sẽ được dùng như các phép toán của lớp theo cách viết thông thường. Cách định nghĩa này được gọi là phép chồng toán tử (như khái niệm chồng hàm trong các chương trước).

1. Tên hàm toán tử

Gồm từ khoá operator và tên phép toán.

Ví dụ:

operator+(định nghĩa chồng phép +)

operator- (định nghĩa chồng phép -)

2. Các đối của hàm toán tử

– Với các phép toán có 2 toán hạng thì hàm toán tử cần có 2 đối. Đối thứ nhất ứng với toán hạng thứ nhất, đối thứ hai ứng với toán hạng thứ hai. Do vậy, với các phép toán không giao hoán (phép -) thì thứ tự đối là rất quan trọng.

Ví dụ: Các hàm toán tử cộng, trừ phân số được khai báo như sau:

```
struct PS
{
    int a;      //Tử số
    int b;      // Mẫu số
};
PS operator+(PS p1, PS p2);          // p1 + p2
PS operator-(PS p1 , PS p2);         // p1 - p2
PS operator*(PS p1, PS p2);          // p1 *p2
PS operator/(PS p1, PS p2);          // p1/p2
```

– Với các phép toán có một toán hạng, thì hàm toán tử có một đối. Ví dụ hàm toán tử đổi dấu ma trận (đổi dấu tất cả các phần tử của ma trận) được khai báo như sau:

```
struct MT
{
    double a[20][20] ;    // Mảng chứa các phần tử ma trận
    int m ;               // Số hàng ma trận
```

```
    int n ;                    // Số cột ma trận
};
MT operator-(MT x) ;
```

3. Thân của hàm toán tử

Viết như thân của hàm thông thường. Ví dụ hàm đổi dấu ma trận có thể được định nghĩa như sau:

```
struct MT
{
    double a[20][20] ;         // Mảng chứa các phần tử ma trận
    int m ;                    // Số hàng ma trận
    int n ;                    // Số cột ma trận
};
MT operator-(MT x)
{
    MT y;
    for (int i=1 ;i<= y.m ; ++i)
    for (int j =1 ;j<= y.n ; ++j)y.a[i][j] =- x.a[i][j];
    return y;
}
```

a. Cách dùng hàm toán tử

Có 2 cách dùng:

Cách 1: Dùng như một hàm thông thường bằng cách viết lời gọi

Ví dụ:

```
PS p, q, u, v ;
u = operator+(p, q) ;          // u = p + q
v = operator-(p, q) ;          // v= p - q
```

Cách 2: Dùng như phép toán của C++

Ví dụ:

```
PS p, q, u, v ;
u = p + q ;                    // u = p + q
v = p - q ;                    //v = p - q
```

Chú ý: Khi dùng các hàm toán tử như phép toán của C++ ta có thể kết hợp nhiều

phép toán để viết các công thức phức tạp. Cũng cho phép dùng dấu ngoặc tròn để quy định thứ tự thực hiện các phép tính. Thứ tự ưu tiên của các phép tính vẫn tuân theo các quy tắc ban đầu của C++. Chẳng hạn các phép * và / có thứ tự ưu tiên cao hơn so với các phép + và -

b. Các ví dụ về định nghĩa chồng toán tử

Ví dụ 1 : Trong ví dụ này ngoài việc sử dụng các hàm toán tử để thực hiện 4 phép tính trên phân số, còn định nghĩa chồng các phép toán << và >> để xuất và nhập phân số.

Hàm operator<< có 2 đối kiểu ostream& và PS (Phân số). Hàm trả về giá trị kiểu ostream& và được khai báo như sau:

```
ostream& operator<< (ostream& os, PS p);
```

Tượng tự hàm operator>> được khai báo như sau:

```
istream& operator>> (istream& is,PS &p);
```

Dưới đây sẽ chỉ ra cách xây dựng và sử dụng các hàm toán tử.

Chúng ta cũng sẽ thấy việc sử dụng các hàm toán tử rất tự nhiên, ngắn gọn và tiện lợi.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
typedef struct
{
      int a,b;
} PS;
ostream& operator<< (ostream& os, PS p);
istream& operator>> (istream& is,PS &p);
int uscln(int x, int y);
PS rutgon(PS p);
PS operator+(PS p1, PS p2);
PS operator-(PS p1, PS p2);
PS operator*(PS p1, PS p2);
PS operator/(PS p1, PS p2);
ostream& operator<< (ostream& os, PS p)
{
      os << p.a << '/' << p.b ;
```

```
        return os;
}
istream& operator>> (istream& is,PS &p)
{
        cout << "\n Nhap tu va mau: " ;
        is >> p.a >> p.b ;
        return is;
}
int uscln(int x, int y)
{
        x=abs(x);y=abs(y);
        if (x*y==0) return 1;
        while (x!=y)
        {
                if (x>y) x-=y;
                else y-=x;
        }
        return x;
}
PS rutgon(PS p)
{
        PS q;
        int x;
        x=uscln(p.a,p.b);
        q.a = p.a / x ;
        q.b = p.b/ x ;
        return q;
}
PS operator+(PS p1, PS p2)
{
        PS q;
        q.a = p1.a*p2.b + p2.a*p1.b;
        q.b = p1 .b * p2.b ;
```

```
        return rutgon(q);
}
PS operator-(PS p1, PS p2)
{
        PS q;
        q.a = p1.a*p2.b - p2.a*p1 .b;
        q.b = p1.b * p2.b ;
        return rutgon(q);
}
PS operator*(PS p1, PS p2)
{
        PS q;
        q.a = p1.a * p2.a ;
        q.b = p1.b * p2.b ;
        return rutgon(q);
}
PS operator/(PS p1 , PS p2)
{
        PS q;
        q.a = p1.a * p2.b ;
        q.b = p1.b * p2.a ;
        return rutgon(q);
}
void main()
{
        PS p, q, z, u, v ;
        PS s;
        cout <<"\nNhap cac PS p, q, z, u, v: " ;
        cin >> p >> q >> z >> u >> v ;
        s = (p - q*z) / (u + v) ;
        cout << "\n Phan so s = " << s;
        getch();
}
```

Ví dụ 2 : Chương trình đưa vào các hàm toán tử:

operator- có một đối dùng để đảo dấu một đa thức

operator+ có 2 đối dùng để cộng 2 đa thức

operator- có 2 đối dùng để trừ 2 đa thức

operator* có 2 đối dùng để nhân 2 đa thức

operator^có 2 đối dùng để tính giá đa thức tại x

operator<< có 2 đối dùng để in đa thức

operator>> có 2 đối dùng để nhập đa thức

Chương trình sẽ nhập 4 đa thức: p, q, r, s. Sau đó tính đa thức: f = -(p+q)*(r-s)
Cuối cùng tính giá trị f(x), với x là một số thực nhập từ bàn phím.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
struct DT
{
	double a[20];// Mang chua cac he so da thuc a0, a1,...
	int n ;// Bac da thuc
};
ostream& operator<< (ostream& os, DT d);
istream& operator>> (istream& is, DT &d);
DT operator-(const DT& d);
DT operator+(DT d1, DT d2);
DT operator-(DT d1, DT d2);
DT operator*(DT d1, DT d2);
double operator^(DT d, double x);// Tinh gia tri da thuc
ostream& operator<< (ostream& os, DT d)
{
	os << " Cac he so (tu ao): " ;
	for (int i=0 ;i<= d.n ;++i)
	os << d.a[i] <<" " ;
```

```
        return os;
}
istream& operator>> (istream& is, DT &d)
{
        cout << " Bac da thuc: " ;
        cin >> d.n;
        cout << "Nhap cac he so da thuc:" ;
        for (int i=0 ;i<=d.n ;++i)
        {
                cout << "\n He so bac " << i <<" = " ;
                is >> d.a[i] ;
        }
        return is;
}
DT operator-(const DT& d)
{
        DT p;
        p.n = d.n;
        for (int i=0 ;i<=d.n ;++i)
        p.a[i] = -d.a[i];
        return p;
}
DT operator+(DT d1, DT d2)
{
        DT d;
        int k,i;
        k = d1.n > d2.n ? d1.n : d2.n ;
        for (i=0;i<=k ;++i)
                if (i<=d1.n && i<=d2.n) d.a[i] = d1.a[i] + d2.a[i];
                else  if (i<=d1.n) d.a[i] = d1.a[i];
                        else d.a[i] = d2.a[i];
        i = k;
        while (i>0 && d.a[i]==0.0) --i;
```

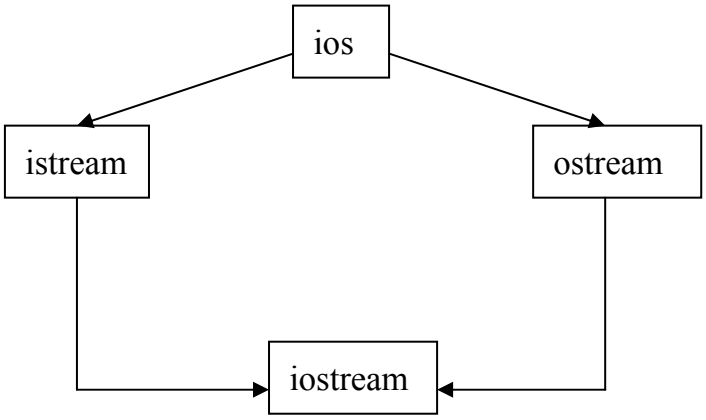
```
        d.n=i;
        return d ;
}
DT operator-(DT d1, DT d2)
{
        return (d1 + (-d2));
}
DT operator*(DT d1 , DT d2)
{
        DT d;
        int k, i, j;
        k = d.n = d1.n + d2.n ;
        for (i=0;i<=k;++i) d.a[i] = 0;
        for (i=0 ;i<= d1 .n ;++i)
        for (j=0 ;j<= d2.n ;++j)
        d.a[i+j] += d1 .a[i]*d2.a[j];
        return d;
}
double operator^(DT d, double x)
{
        double s=0.0 , t=1.0;
        for (int i=0 ;i<= d.n ;++i)
        {
                s += d.a[i]*t;
                t *= x;
        }
        return s;
}
void main()
{
        DT p,q,r,s,f;
        double x,g;
        clrscr();
```

```
        cout <<"\n Nhap da thuc P " ;cin >> p;
        cout <<"\n Nhap da thuc Q " ;cin >> q;
        cout <<"\n Nhap da thuc R " ;cin >> r;
        cout <<"\n Nhap da thuc S " ;cin >> s;
        cout << "\n Nhap so thuc x: " ;cin >> x;
        f = -(p+q)*(r-s);
        g = f^x;
        cout << "\n Da thuc f "<< f ;
        cout << "\n x = " << x;
        cout << "\n f(x) = " << g;
        getch();
}
```

CHƯƠNG 9
CÁC DÒNG NHẬP/XUẤT VÀ FILE

Nhập/xuất với cin/cout
Định dạng
In ra máy in
Làm việc với File
Nhập/xuất nhị phân

Trong C++ có sẵn một số lớp chuẩn chứa dữ liệu và các phương thức phục vụ cho các thao tác nhập/xuất dữ liệu của NSD, thường được gọi chung là ***stream*** (dòng). Trong số các lớp này, lớp có tên **ios** là lớp cơ sở, chứa các thuộc tính để định dạng việc nhập/xuất và kiểm tra lỗi. Mở rộng (kế thừa) lớp này có các lớp **istream, ostream** cung cấp thêm các toán tử nhập/xuất như >>, << và các hàm get, getline, read, ignore, put, write, flush … Một lớp rộng hơn có tên **iostream** là tổng hợp của 2 lớp trên. Bốn lớp nhập/xuất cơ bản này được khai báo trong các file tiêu đề có tên tương ứng (với đuôi *.h). Sơ đồ thừa kế của 4 lớp trên được thể hiện qua hình vẽ dưới đây.



Đối tượng của các lớp trên được gọi là các *dòng* dữ liệu. Một số đối tượng thuộc lớp iostream đã được khai báo sẵn (*chuẩn*) và được gắn với những thiết bị nhập/xuất cố định như các đối tượng **cin, cout, cerr, clog** gắn với bàn phím (cin) và màn hình (cout, cerr, clog). Điều này có nghĩa các toán tử >>, << và các hàm kể trên khi làm việc với các đối tượng này sẽ cho phép NSD nhập dữ liệu thông qua bàn phím hoặc xuất kết quả thông qua màn hình.

Để nhập/xuất thông qua các thiết bị khác (như máy in, file trên đĩa …), C++

cung cấp thêm các lớp **ifstream, ofstream, fstream** cho phép NSD khai báo các đối tượng mới gắn với thiết bị và từ đó nhập/xuất thông qua các thiết bị này.

Trong chương này, chúng ta sẽ xét các đối tượng chuẩn **cin, cout** và một số toán tử, hàm nhập xuất đặc trưng của lớp **iostream** cũng như cách tạo và sử dụng các đối tượng thuộc các lớp **ifstream, ofstream, fstream** để làm việc với các thiết bị như máy in và file trên đĩa.

I. NHẬP/XUẤT VỚI CIN/COUT

Như đã nhắc ở trên, **cin** là dòng dữ liệu nhập (đối tượng) thuộc lớp *istream.* Các thao tác trên đối tượng này gồm có các toán tử và hàm phục vụ nhập dữ liệu vào cho biến từ bàn phím.

1. Toán tử nhập >>

Toán tử này cho phép nhập dữ liệu từ một dòng Input_stream nào đó vào cho một danh sách các biến. Cú pháp chung như sau:

Input_stream >> biến1 >> biến2 >> …

trong đó Input_stream là đối tượng thuộc lớp istream. Trường hợp Input_stream là cin, câu lệnh nhập sẽ được viết:

cin >> biến1 >> biến2 >> …

câu lệnh này cho phép nhập dữ liệu từ bàn phím cho các biến. Các biến này có thể thuộc các kiểu chuẩn như : kiểu nguyên, thực, ký tự, xâu kí tự. Chú ý 2 đặc điểm quan trọng của câu lệnh trên.

- Lệnh sẽ bỏ qua không gán các dấu trắng (dấu cách <>, dấu Tab, dấu xuống dòng ↵) vào cho các biến (kể cả biến xâu kí tự).
- Khi NSD nhập vào dãy byte nhiều hơn cần thiết để gán cho các biến thì số byte còn lại và kể cả dấu xuống dòng ↵ sẽ nằm lại trong cin. Các byte này sẽ tự động gán cho các biến trong lần nhập sau mà không chờ NSD gõ thêm dữ liệu vào từ bàn phím. Do vậy câu lệnh

```
cin >> a >> b >> c;
```

cũng có thể được viết thành

```
cin >> a;
cin >> b;
cin >> c;
```

và chỉ cần nhập dữ liệu vào từ bàn phím một lần chung cho cả 3 lệnh (mỗi dữ liệu nhập cho mỗi biến phải cách nhau ít nhất một dấu trắng)

Ví dụ 1 : Nhập dữ liệu cho các biến

```
int a;
float b;
char c;
char *s;
cin >> a >> b >> c >> s;
```

giả sử NSD nhập vào dãy dữ liệu : <><>12<>34.517ABC<>12E<>D ↵

khi đó các biến sẽ được nhận những giá trị cụ thể sau:

a = 12

b = 34.517

c = 'A'

s = "BC"

trong cin sẽ còn lại dãy dữ liệu : <>12E<>D ↵.

Nếu trong đoạn chương trình tiếp theo có câu lệnh cin >> s; thì s sẽ được tự động gán giá trị "12E" mà không cần NSD nhập thêm dữ liệu vào cho cin.

Qua ví dụ trên một lần nữa ta nhắc lại đặc điểm của toán tử nhập >> là các biến chỉ lấy dữ liệu vừa đủ cho kiểu của biến (ví dụ biến c chỉ lấy một kí tự 'A', b lấy giá trị 34.517) hoặc cho đến khi gặp dấu trắng đầu tiên (ví dụ a lấy giá trị 12, s lấy giá trị "BC" dù trong cin vẫn còn dữ liệu). Từ đó ta thấy toán tử >> là không phù hợp khi nhập dữ liệu cho các xâu kí tự có chứa dấu cách. C++ giải quyết trường hợp này bằng một số hàm (phương thức) nhập khác thay cho toán tử >>.

2. Các hàm nhập kí tự và xâu kí tự

a. Nhập kí tự

- **cin.get()** : Hàm trả lại một kí tự (kể cả dấu cách, dấu ↵).. Ví dụ:

  ```
  char ch;
  ch = cin.get();
  ```

  - nếu nhập AB↵, ch nhận giá trị 'A', trong cin còn B↵.
  - nếu nhập A↵, ch nhận giá trị 'A', trong cin còn ↵.
  - nếu nhập ↵, ch nhận giá trị '↵', trong cin rỗng.

- **cin.get(ch)** : Hàm nhập kí tự cho ch và trả lại một tham chiếu tới cin. Do hàm trả lại tham chiếu tới cin nên có thể viết các phương thức nhập này liên tiếp trên một đối tượng cin. Ví dụ:

  ```
  char c, d;
  cin.get(c).get(d);
  ```

nếu nhập AB↵ thì c nhận giá trị 'A' và d nhận giá trị 'B'. Trong cin còn 'C↵'.

b. Nhập xâu kí tự

- **cin.get(s, n, fchar)** : Hàm nhập cho s dãy kí tự từ cin. Dãy được tính từ kí tự đầu tiên trong cin cho đến khi đã đủ n – 1 kí tự hoặc gặp kí tự kết thúc fchar. Kí tự kết thúc này được ngầm định là dấu xuống dòng nếu bị bỏ qua trong danh sách đối. Tức có thể viết câu lệnh trên dưới dạng **cin.get(s, n)** khi đó xâu s sẽ nhận dãy kí tự nhập cho đến khi đủ n-1 kí tự hoặc đến khi NSD kết thúc nhập (bằng dấu ↵).

Chú ý :

– Lệnh sẽ tự động gán dấu kết thúc xâu ('\0') vào cho xâu s sau khi nhập xong.

– Các lệnh có thể viết nối nhau, ví dụ: **cin.get(s1, n1).get(s2,n2);**

– Kí tự kết thúc fchar (hoặc ↵) vẫn nằm lại trong cin. Điều này có thể làm trôi các lệnh get() tiếp theo. Ví dụ:

```
struct Sinhvien {
        char *ht;                   // họ tên
        char *qq;                   // quê quán
};
void main()
{
        int i;
        for (i=1; i<=3; i++) {
                cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
                cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
        }
        …
}
```

Trong đoạn lệnh trên sau khi nhập họ tên của sinh viên thứ 1, do kí tự ↵ vẫn nằm trong bộ đệm nên khi nhập quê quán chương trình sẽ lấy kí tự ↵ này gán cho qq, do đó quê quán của sinh viên sẽ là xâu rỗng.

Để khắc phục tình trạng này chúng ta có thể sử dụng một trong các câu lệnh nhập kí tự để "nhấc" dấu enter còn "rơi vãi" ra khỏi bộ đệm. Có thể sử dụng các câu lệnh sau :

```
cin.get();                  // đọc một kí tự trong bộ đệm
cin.ignore(n);              //đọc n kí tự trong bộ đệm (với n=1)
```

như vậy để đoạn chương trình trên hoạt động tốt ta có thể tổ chức lại như sau:

```
void main()
{
    int i;
    for (i=1; i<=3; i++) {
        cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
    cin.get();                    // nhấc 1 kí tự (enter)
    cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
    cin.get()                     // hoặc cin.ignore(1);
}
…
}
```

- **cin.getline(s, n, fchar):** Phương thức này hoạt động hoàn toàn tương tự phương thức **cin.get(s, n, fchar),** tuy nhiên nó có thể khắc phục "lỗi enter" của câu lệnh trên. Cụ thể hàm sau khi gán nội dung nhập cho biến s sẽ xóa kí tự enter khỏi bộ đệm và do vậy NSD không cần phải sử dụng thêm các câu lệnh phụ trợ (cin.get(), cin.ignore(1)) để loại enter ra khỏi bộ đệm.
- **cin.ignore(n):** Phương thức này của đối tượng cin dùng để đọc và loại bỏ n kí tự còn trong bộ đệm (dòng nhập cin).

Chú ý: Toán tử nhập >> cũng giống các phương thức nhập kí tự và xâu kí tự ở chỗ cũng để lại kí tự enter trong cin. Do vậy, chúng ta nên sử dụng các phương thức cin.get(), cin.ignore(n) để loại bỏ kí tự enter trước khi thực hiện lệnh nhập kí tự và xâu kí tự khác.

Tương tự dòng nhập cin, **cout** là dòng dữ liệu xuất thuộc lớp *ostream.* Điều này có nghĩa dữ liệu làm việc với các thao tác xuất (in) sẽ đưa kết quả ra cout mà đã được mặc định là màn hình. Do đó ta có thể sử dụng toán tử xuất << và các phương thức xuất trong các lớp ios (lớp cơ sở) và ostream.

3. Toán tử xuất <<

Toán tử này cho phép xuất giá trị của dãy các biểu thức đến một dòng Output_stream nào đó với cú pháp chung như sau:

Output_stream << bt_1 << bt_2 << …

ở đây Output_stream là đối tượng thuộc lớp ostream. Trường hợp Output_stream là cout, câu lệnh xuất sẽ được viết:

cout << bt_1 << bt_2 << …

câu lệnh này cho phép in kết quả của các biểu thức ra màn hình. Kiểu dữ liệu của

các biểu thức có thể là số nguyên, thực, kí tự hoặc xâu kí tự.

II. ĐỊNH DẠNG

Các giá trị in ra màn hình có thể được trình bày dưới nhiều dạng khác nhau thông qua các công cụ định dạng như các phương thức, các cờ và các bộ phận khác được khai báo sẵn trong các lớp ios và ostream.

1. Các phương thức định dạng

a. Chỉ định độ rộng cần in

cout.width(n) ;

Số cột trên màn hình để in một giá trị được ngầm định bằng với độ rộng thực (số chữ số, chữ cái và kí tự khác trong giá tị được in). Để đặt lại độ rộng màn hình dành cho giá trị cần in (thông thường lớn hơn độ rộng thực) ta có thể sử dụng phương thức trên.

Phương thức này cho phép các giá trị in ra màn hình với độ rộng n. Nếu n bé hơn độ rộng thực sự của giá trị thì máy sẽ in giá trị với số cột màn hình bằng với độ rộng thực. Nếu n lớn hơn độ rộng thực, máy sẽ in giá trị căn theo lề phải, và để trống các cột thừa phía trước giá trị được in. Phương thức này chỉ có tác dụng với giá trị cần in ngay sau nó. Ví dụ:

```
int a = 12; b = 345;          // độ rộng thực của a là 2, của b là 3
cout << a;                    // chiếm 2 cột màn hình
cout.width(7);                // đặt độ rộng giá trị in tiếp theo là 7
cout  << b;                   // b in trong 7 cột với 4 dấu cách đứng trước
```

Kết quả in ra sẽ là: 12<><><><>345

b. Chỉ định kí tự chèn vào khoảng trống trước giá trị cần in

cout.fill(ch) ;

Kí tự độn ngầm định là dấu cách, có nghĩa khi độ rộng của giá trị cần in bé hơn độ rộng chỉ định thì máy sẽ độn các dấu cách vào trước giá trị cần in cho đủ với độ rộng chỉ định. Có thể yêu cầu độn một kí tự ch bất kỳ thay cho dấu cách bằng phương thức trên. Ví dụ trong dãy lệnh trên, nếu ta thêm dòng lệnh cout.fill('*') trước khi in b chẳng hạn thì kết quả in ra sẽ là: 12****345.

Phương thức này có tác dụng với mọi câu lệnh in sau nó cho đến khi gặp một chỉ định mới.

c. Chỉ định độ chính xác (số số lẻ thập phân) cần in

cout.precision(n) ;

Phương thức này yêu cầu các số thực in ra sau đó sẽ có n chữ số lẻ. Các số

thực trước khi in ra sẽ được làm tròn đến chữ số lẻ thứ n. Chỉ định này có tác dụng cho đến khi gặp một chỉ định mới. Ví dụ:

```
int a = 12.3; b = 345.678;          // độ rộng thực của a là 4, của b là 7
cout << a;                          // chiếm 4 cột màn hình
cout.width(10);                     // đặt độ rộng giá trị in tiếp theo là 10
cout.precision(2);                  // đặt độ chính xác đến 2 số lẻ
cout  << b;                  // b in trong 10 cột với 4 dấu cách đứng trước
```

Kết quả in ra sẽ là: 12.3<><><><>345.68

2. Các cờ định dạng

Một số các qui định về định dạng thường được gắn liền với các "cờ". Thông thường nếu định dạng này được sử dụng trong suốt quá trình chạy chương trình hoặc trong một khoảng thời gian dài trước khi gỡ bỏ thì ta "bật" các cờ tương ứng với nó. Các cờ được bật sẽ có tác dụng cho đến khi cờ với định dạng khác được bật. Các cờ được cho trong file tiêu đề iostream.h.

Để bật/tắt các cờ ta sử dụng các phương thức sau:

```
cout.setf(danh sách cờ);      // Bật các cờ trong danh sách
cout.unsetf(danh sách cờ);    // Tắt các cờ trong danh sách
```

Các cờ trong danh sách được viết cách nhau bởi phép toán hợp bit (|). Ví dụ lệnh cout.setf(ios::left | ios::scientific) sẽ bật các cờ ios::left và ios::scientific. Phương thức cout.unsetf(ios::right | ios::fixed) sẽ tắt các cờ ios::right | ios::fixed.

Dưới đây là danh sách các cờ cho trong iostream.h.

a. Nhóm căn lề

- **ios::left** : nếu bật thì giá trị in nằm bên trái vùng in ra (kí tự độn nằm sau).
- **ios::right** : giá trị in nằm bên phái vùng in ra (kí tự độn nằm trước), đây là trường hợp ngầm định nếu ta không sử dụng cờ cụ thể.
- **ios::internal** : giống cờ ios::right tuy nhiên dấu của giá trị in ra sẽ được in đầu tiên, sau đó mới đến kí tự độn và giá trị số.

Ví dụ:

```
int a = 12.3; b = –345.678;     // độ rộng thực của a là 4, của b là 8
cout << a;                      // chiếm 4 cột màn hình
cout.width(10);                 // đặt độ rộng giá trị in tiếp theo là 10
cout.fill('*') ;                // dấu * làm kí tự độn
cout.precision(2);              // đặt độ chính xác đến 2 số lẻ
cout.setf(ios::left) ;          // bật cờ ios::left
```

```
cout << b;                    // kết qủa: 12.3–345.68***
cout.setf(ios::right) ;       // bật cờ ios::right
cout << b;                    // kết qủa: 12.3***–345.68
cout.setf(ios::internal) ;    // bật cờ ios::internal
cout << b;                    // kết qủa: 12.3–***345.68
```

b. Nhóm định dạng số nguyên

- **ios::dec** : in số nguyên dưới dạng thập phân (ngầm định)
- **ios::oct** : in số nguyên dưới dạng cơ số 8
- **ios::hex** : in số nguyên dưới dạng cơ số 16

c. Nhóm định dạng số thực

- **ios::fixed** : in số thực dạng dấu phảy tĩnh (ngầm định)
- **ios::scientific** : in số thực dạng dấu phảy động
- **ios::showpoint** : in đủ n chữ số lẻ của phần thập phân, nếu tắt (ngầm định) thì không in các số 0 cuối của phần thập phân.

Ví dụ: giả sử độ chính xác được đặt với 3 số lẻ (bởi câu lệnh cout.precision(3))

- nếu fixed bật + showpoint bật :

  123.2500 được in thành 123.250
  123.2599 được in thành 123.260
  123.2 được in thành 123.200

- nếu fixed bật + showpoint tắt :

  123.2500 được in thành 123.25
  123.2599 được in thành 123.26
  123.2 được in thành 123.2

- nếu scientific bật + showpoint bật :

  12.3 được in thành 1.230e+01
  2.32599 được in thành 2.326e+00
  324 được in thành 3.240e+02

- nếu scientific bật + showpoint tắt :

  12.3 được in thành 1.23e+01
  2.32599 được in thành 2.326e+00
  324 được in thành 3.24e+02

d. Nhóm định dạng hiển thị

- **ios::showpos** : nếu tắt (ngầm định) thì không in dấu cộng (+) trước số dương. Nếu bật trước mỗi số dương sẽ in thêm dấu cộng.
- **ios::showbase** : nếu bật sẽ in số 0 trước các số nguyên hệ 8 và in 0x trước số hệ 16. Nếu tắt (ngầm định) sẽ không in 0 và 0x.
- **ios::uppercase** : nếu bật thì các kí tự biểu diễn số trong hệ 16 (A..F) sẽ viết hoa, nếu tắt (ngầm định) sẽ viết thường.

3. Các bộ và hàm định dạng

iostream.h cũng cung cấp một số bộ và hàm định dạng cho phép sử dụng tiện lợi hơn so với các cờ và các phương thức vì nó có thể được viết liên tiếp trên dòng lệnh xuất.

a. Các bộ định dạng

```
dec                 // tương tự ios::dec
oct                 // tương tự ios::dec
hex                 // tương tự ios::hex
endl                // xuất kí tự xuống dòng ('\n')
flush               // đẩy toàn bộ dữ liệu ra dòng xuất
```

Ví dụ :

```
cout.setf(ios::showbase) ;          // cho phép in các kí tự biểu thị cơ số
cout.setf(ios::uppercase) ;         // dưới dạng chữ viết hoa
int a = 171; int b = 32 ;
cout << hex << a << endl << b ;     // in 0xAB và 0x20
```

b. Các hàm định dạng (#include <iomanip.h>)

```
setw(n)             // tương tự cout.width(n)
setprecision(n)     // tương tự cout.precision(n)
setfill(c)          // tương tự cout.fill(c)
setiosflags(l)      // tương tự cout.setf(l)
resetiosflags(l)    // tương tự cout.unsetf(l)
```

III. IN RA MÁY IN

Như trong phần đầu chương đã trình bày, để làm việc với các thiết bị khác với màn hình và đĩa … chúng ta cần tạo ra các đối tượng (thuộc các lớp ifstream, ofstream và fstream) tức các dòng tin bằng các hàm tạo của lớp và gắn chúng với

thiết bị bằng câu lệnh:

ofstream Tên_dòng(thiết bị) ;

Ví dụ để tạo một đối tượng mang tên Mayin và gắn với máy in, chúng ta dùng lệnh:

ofstream Mayin(4) ;

trong đó 4 là số hiệu của máy in.

Khi đó mọi câu lệnh dùng toán tử xuất << và cho ra Mayin sẽ đưa dữ liệu cần in vào một bộ đệm mặc định trong bộ nhớ. Nếu bộ đệm đầy, một số thông tin đưa vào trước sẽ tự động chuyển ra máy in. Để chủ động đưa tất cả dữ liệu còn lại trong bộ đệm ra máy in chúng ta cần sử dụng bộ định dạng flush (Mayin << flush << …) hoặc phương thức flush (Mayin.flush();). Ví dụ:

Sau khi đã khai báo một đối tượng mang tên Mayin bằng câu lệnh như trên Để in chu vi và diện tích hình chữ nhật có cạnh cd và cr ta có thể viết:

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl;
Mayin.flush();
```

hoặc :

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl << flush;
```

khi chương trình kết thúc mọi dữ liệu còn lại trong các đối tượng sẽ được tự động chuyển ra thiết bị gắn với nó. Ví dụ máy in sẽ in tất cả mọi dữ liệu còn sót lại trong Mayin khi chương trình kết thúc.

IV. LÀM VIỆC VỚI FILE

Làm việc với một file trên đĩa cũng được quan niệm như làm việc với các thiết bị khác của máy tính (ví dụ như làm việc với máy in với đối tượng Mayin trong phần trên hoặc làm việc với màn hình với đối tượng chuẩn cout). Các đối tượng này được khai báo thuộc lớp ifstream hay ofstream tùy thuộc ta muốn sử dụng file để đọc hay ghi.

Như vậy, để sử dụng một file dữ liệu đầu tiên chúng ta cần tạo đối tượng và gắn cho file này. Để tạo đối tượng có thể sử dụng các hàm tạo có sẵn trong hai lớp ifstream và ofstream. Đối tượng sẽ được gắn với tên file cụ thể trên đĩa ngay trong quá trình tạo đối tượng (tạo đối tượng với tham số là tên file) hoặc cũng có thể được gắn với tên file sau này bằng câu lệnh mở file. Sau khi đã gắn một đối tượng với file trên đĩa, có thể sử dụng đối tượng như đối với Mayin hoặc cin, cout. Điều này có nghĩa trong các câu lệnh in ra màn hình chỉ cần thay từ khóa cout bởi tên đối tượng mọi dữ liệu cần in trong câu lệnh sẽ được ghi lên file mà đối tượng đại diện. Cũng tương tự nếu thay cin bởi tên đối tượng, dữ liệu sẽ được đọc vào từ file thay cho từ

bàn phím. Để tạo đối tượng dùng cho việc ghi ta khai báo chúng với lớp *ofstream* còn để dùng cho việc đọc ta khai báo chúng với lớp *ifstream*.

1. Tạo đối tượng gắn với file

Mỗi lớp ifstream và ofstream cung cấp 4 phương thức để tạo file. Ở đây chúng tôi chỉ trình bày 2 cách (2 phương thức) hay dùng.

+ Cách 1: **<Lớp> đối_tượng;**

  đối_tượng.open(tên_file, chế_độ);

Lớp là một trong hai lớp ifstream và ofstream. Đối tượng là tên do NSD tự đặt. Chế độ là cách thức làm việc với file (xem dưới). Cách này cho phép tạo trước một đối tượng chưa gắn với file cụ thể nào. Sau đó dùng tiếp phương thức open để đồng thời mở file và gắn với đối tượng vừa tạo.

Ví dụ:

```
ifstream f;              // tạo đối tượng có tên f để đọc hoặc
ofstream f;              // tạo đối tượng có tên f để ghi
f.open("Baitap");        // mở file Baitap và gắn với f
```

+ Cách 2: **<Lớp> đối_tượng(tên_file, chế_độ)**

Cách này cho phép đồng thời mở file cụ thể và gắn file với tên đối tượng trong câu lệnh.

Ví dụ:

```
ifstream f("Baitap");    // mở file Baitap gắn với đối tượng f để
ofstream f("Baitap);     // đọc hoặc ghi.
```

Sau khi mở file và gắn với đối tượng f, mọi thao tác trên f cũng chính là làm việc với file Baitap.

Trong các câu lệnh trên có các chế độ để qui định cách thức làm việc của file. Các chế độ này gồm có:

- ios::binary : quan niệm file theo kiểu nhị phân. Ngầm định là kiểu văn bản.
- ios::in : file để đọc (ngầm định với đối tượng trong ifstream).
- ios::out : file để ghi (ngầm định với đối tượng trong ofstream), nếu file đã có trên đĩa thì nội dung của nó sẽ bị ghi đè (bị xóa).ios::app : bổ sung vào cuối file
- ios::trunc : xóa nội dung file đã có
- ios::ate : chuyển con trỏ đến cuối file
- ios::nocreate : không làm gì nếu file chưa có

- ios::replace : không làm gì nếu file đã có

có thể chỉ định cùng lúc nhiều chế độ bằng cách ghi chúng liên tiếp nhau với toán tử hợp bit |. Ví dụ để mở file bài tập như một file nhị phân và ghi tiếp theo vào cuối file ta dùng câu lệnh:

```
ofstream f("Baitap", ios::binary | ios::app);
```

2. Đóng file và giải phóng đối tượng

Để đóng file được đại diện bởi f, sử dụng phương thức close như sau:

đối_tượng.close();

Sau khi đóng file (và giải phóng mối liên kết giữa đối tượng và file) có thể dùng đối tượng để gắn và làm việc với file khác bằng phương thức open như trên.

Ví dụ 2 : Đọc một dãy số từ bàn phím và ghi lên file. File được xem như file văn bản (ngầm định), các số được ghi cách nhau 1 dấu cách.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
void main()
{
    ofstream f;                    // khai báo (tạo) đối tượng f
    int x;
    f.open("DAYSO");               // mở file DAYSO và gắn với f
    for (int i = 1; i<=10; i++) {
        cin >> x;
        f << x << ' ';
    }
    f.close();
}
```

Ví dụ 3 : Chương trình sau nhập danh sách sinh viên, ghi vào file 1, đọc ra mảng, sắp xếp theo tuổi và in ra file 2. Dòng đầu tiên trong file ghi số sinh viên, các dòng tiếp theo ghi thông tin của sinh viên gồm họ tên với độ rộng 24 kí tự, tuổi với độ rộng 4 kí tự và điểm với độ rộng 8 kí tự.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
struct Sv {
	char *hoten;
	int tuoi;
	double diem;
};
class Sinhvien {
	int sosv ;
	Sv *sv;
public:
	Sinhvien() {
		sosv = 0;
		sv = NULL;
	}
	void nhap();
	void sapxep();
	void ghifile(char *fname);
};

void Sinhvien::nhap()
{
	cout << "\nSố sinh viên: "; cin >> sosv;
	int n = sosv;
	sv = new Sinhvien[n+1];                    // Bỏ phần tử thứ 0
	for (int i = 1; i <= n; i++) {
		cout << "\nNhập sinh viên thứ: " << i << endl;
		cout << "\nHọ tên: "; cin.ignore(); cin.getline(sv[i].hoten);
		cout << "\nTuổi: "; cin >> sv[i].tuoi;
		cout << "\nĐiểm: "; cin >> sv[i].diem;
	}
}
```

```
void Sinhvien::ghi(char fname)
{
	ofstream f(fname) ;
	f << sosv;
	f << setprecision(1) << setiosflags(ios::showpoint) ;
	for (int i=1; i<=sosv; i++) {
		f << endl << setw(24) << sv[i].hoten << setw(4) << tuoi;
		f << setw(8) << sv[i].diem;
	}
	f.close();
}

void Sinhvien::doc(char fname)
{
	ifstream f(fname) ;
	f >> sosv;
	for (int i=1; i<=sosv; i++) {
		f.getline(sv[i].hoten, 25);
		f >> sv[i].tuoi >> sv[i].diem;
	}
	f.close();
}

void Sinhvien::sapxep()
{
	int n = sosv;
	for (int i = 1; i < n; i++) {
	for (int j = j+1; j <= n; j++) {
	if (sv[i].tuoi > sv[j].tuoi) {
		Sinhvien t = sv[i]; sv[i] = sv[j]; sv[j] = t;
	}
}

void main() {
	clrscr();
```

```
    Sinhvien x ;
    x.nhap(); x.ghi("DSSV1");
    x.doc("DSSV1"); x.sapxep(); x.ghi("DSSV2");
    cout << "Đã xong";
    getch();
}
```

3. Kiểm tra sự tồn tại của file, kiểm tra hết file

Việc mở một file chưa có để đọc sẽ gây nên lỗi và làm dừng chương trình. Khi xảy ra lỗi mở file, giá trị trả lại của phương thức bad là một số khác 0. Do vậy có thể sử dụng phương thức này để kiểm tra một file đã có trên đĩa hay chưa. Ví dụ:

```
ifstream f("Bai tap");
if (f.bad()) {
    cout << "file Baitap chưa có";
    exit(1);
}
```

Khi đọc hoặc ghi, con trỏ file sẽ chuyển dần về cuối file. Khi con trỏ ở cuối file, phương thức eof() sẽ trả lại giá trị khác không. Do đó có thể sử dụng phương thức này để kiểm tra đã hết file hay chưa.

Chương trình sau cho phép tính độ dài của file Baitap. File cần được mở theo kiểu nhị phân.

```
 #include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
    clrscr();
    long dodai = 0;
    char ch;
    ifstream f("Baitap", ios::in | ios::binary) ;
    if (f.bad()) {
        cout << "File Baitap không có";
        exit(1);
    }
```

```
        while (!f.eof()) {
            f.get(ch));
            dodai++;
        }
        cout << "Độ dài của file = " << dodai;
        getch();
}
```

4. Đọc ghi đồng thời trên file

Để đọc ghi đồng thời, file phải được gắn với đối tượng của lớp fstream là lớp thừa kế của 2 lớp ifstream và ofstream. Khi đó chế độ phải được bao gồm chỉ định ios::in | ios::out. Ví dụ:

```
fstream f("Data", ios::in | ios::out) ;
```

hoặc

```
fstream f ;
f.open("Data", ios::in | ios::out) ;
```

5. Di chuyển con trỏ file

Các phương thức sau cho phép làm việc trên đối tượng của dòng xuất (ofstream).

- **đối_tượng.seekp(n)** ; Di chuyển con trỏ đến byte thứ n (các byte được tính từ 0)
- **đối_tượng.seekp(n, vị trí xuất phát) ;** Di chuyển đi n byte (có thể âm hoặc dương) từ vị trí xuất phát. Vị trí xuất phát gồm:
  - ios::beg : từ đầu file
  - ios::end : từ cuối file
  - ios::cur : từ vị trí hiện tại của con trỏ.
- **đối_tượng.tellp(n) ;** Cho biết vị trí hiện tại của con trỏ.

Để làm việc với dòng nhập tên các phương thức trên được thay tương ứng bởi các tên : **seekg** và **tellg**. Đối với các dòng nhập lẫn xuất có thể sử dụng được cả 6 phương thức trên.

Ví dụ sau tính độ dài tệp đơn giản hơn ví dụ ở trên.

```
fstream f("Baitap");
f.seekg(0, ios::end);
cout << "Độ dài bằng = " << f.tellg();
```

Ví dụ 4 : Chương trình nhập và in danh sách sinh viên trên ghi/đọc đồng thời.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main() {
        int stt ;
        char *hoten, *fname, traloi;
        int tuoi;
        float diem;
        fstream f;
        cout << "Nhập tên file: "; cin >> fname;
        f.open(fname, ios::in | ios::out | ios::noreplace) ;
        if (f.bad()) {
                cout << "Tệp đã có. Ghi đè (C/K)?" ;
                cin.get(traloi) ;
                if (toupper(traloi) == 'C') {
                        f.close() ;
                        f.open(fname, ios::in | ios::out | ios::trunc) ;
                } else exit(1);
        }
        stt = 0;
        f << setprecision(1) << setiosflags(ios::showpoint) ;
        // nhập danh sách
        while (1) {
                stt++;
                cout << "\nNhập sinh viên thứ " << stt ;
                cout << "\nHọ tên: "; cin.ignore() ; cin.getline(hoten, 25);
                if (hoten[0] = 0) break;
                cout << "\nTuổi: "; cin >> tuoi;
```

```
            cout << "\nĐiểm: "; cin >> diem;
            f << setw(24) << hoten << endl;
            f << setw(4) << tuoi << set(8) << diem ;
        }
        // in danh sách
        f.seekg(0) ;                        // quay về đầu danh sách
        stt = 0;
        clrscr();
        cout << "Danh sách sinh viên đã nhập\n" ;
        cout << setprecision(1) << setiosflags(ios::showpoint) ;
        while (1) {
            f.getline(hoten,25);
            if (f.eof()) break;
            stt++;
            f >> tuoi >> diem;
            f.ignore();
            cout << "\nSinh viên thứ " << stt ;
            cout << "\nHọ tên: " << hoten;
            cout << "\nTuổi: " << setw(4) << tuoi;
            cout << "\nĐiểm: " << setw(8) << diem;
        }
        f.close();
        getch();
    }
```

V. NHẬP/XUẤT NHỊ PHÂN

1. Khái niệm về 2 loại file: văn bản và nhị phân

a. File văn bản

Trong file văn bản mỗi byte được xem là một kí tự. Tuy nhiên nếu 2 byte 10 (LF), 13 (CR) đi liền nhau thì được xem là một kí tự và nó là kí tự xuống dòng. Như vậy file văn bản là một tập hợp các dòng kí tự với kí tự xuống dòng có mã là 10. Kí tự có mã 26 được xem là kí tự kết thúc file.

b. File nhị phân

Thông tin lưu trong file được xem như dãy byte bình thường. Mã kết thúc file được chọn là -1, được định nghĩa là EOF trong stdio.h. Các thao tác trên file nhị phân thường đọc ghi từng byte một, không quan tâm ý nghĩa của byte.

Một số các thao tác nhập/xuất sẽ có hiệu quả khác nhau khi mở file dưới các dạng khác nhau.

Ví dụ 1 : giả sử ch = 10, khi đó f << ch sẽ ghi 2 byte 10,13 lên file văn bản f, trong khi đó lệnh này chỉ khi 1 byte 10 lên file nhị phân.

Ngược lại, nếu f la file văn bản thì f.getc(ch) sẽ trả về chỉ 1 byte 10 khi đọc được 2 byte 10, 13 liên tiếp nhau.

Một file luôn ngầm định dưới dạng văn bản, do vậy để chỉ định file là nhị phân ta cần sử dụng cờ ios::binary.

2. Đọc, ghi kí tự

- **put(c);** // ghi kí tự ra file
- **get(c);** // đọc kí tự từ file

Ví dụ 2 : Sao chép file 1 sang file 2. Cần sao chép và ghi từng byte một do vậy để chính xác ta sẽ mở các file dưới dạng nhị phân.

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
        clrscr();
        fstream fnguon("DATA1", ios::in | ios::binary);
        fstream fdich("DATA2", ios::out | ios::binary);
        char ch;
        while (!fnguon.eof()) {
                fnguon.get(ch);
                fdich.put(ch);
        }
        fnguon.close();
        fdich.close();
}
```

3. Đọc, ghi dãy kí tự

- **write(char *buf, int n);** // ghi n kí tự trong buf ra dòng xuất
- **read(char *buf, int n);** // nhập n kí tự từ buf vào dòng nhập
- **gcount();** // cho biết số kí tự read đọc được

Ví dụ 3 : Chương trình sao chép file ở trên có thể sử dụng các phương thức mới này như sau:

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
        clrscr();
        fstream fnguon("DATA1", ios::in | ios::binary);
        fstream fdich("DATA2", ios::out | ios::binary);
        char buf[2000] ;
        int n = 2000;
        while (n) {
                fnguon.read(buf, 2000);
                n = fnguon.gcount();
                fdich.write(buf, n);
        }
        fnguon.close();
        fdich.close();
}
```

4. Đọc ghi đồng thời

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <conio.h>
#include <string.h>
#include <ctype.h>

struct Sv {
	char *hoten;
	int tuoi;
	double diem;
};
class Sinhvien {
	int sosv;
	Sv x;
	char fname[30];
	static int size;
public:
	Sinhvien(char *fn);
	void tao();
	void bosung();
	void xemsua();
};

int Sinhvien::size = sizeof(Sv);
Sinhvien::Sinhvien(char *fn)
{
	strcpy(fname, fn) ;
	fstream f;
	f.open(fname, ios::in | ios::ate | ios::binary);
	if (!f.good) sosv = 0;
	else {
		sosv = f.tellg() / size;
	}
}

void Sinhvien::tao()
```

```
{
        fstream f;
        f.open(fname, ios::out | ios::noreplace | ios::binary);
        if (!f.good()) {
                cout << "danh sach da co. Co tao lai (C/K) ?";
                char traloi = getch();
                if (toupper(traloi) == 'C') return;
                else {
                        f.close() ;
                        f.open(fname, ios::out | ios::trunc | ios::binary);
                }
        }
        sosv = 0
        while (1) {
                cout << "\nSinh viên thứ: " << sosv+1;
                cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
                if (x.hoten[0] == 0) break;
                cout << "\nTuổi: "; cin >> x.tuoi;
                cout << "\nĐiểm: "; cin >> x.diem;
                f.write((char*)(&x), size);
                sosv++;
        }
        f.close();
}

void Sinhvien::bosung()
{
        fstream f;
        f.open(fname, ios::out | ios::app | ios::binary);
        if (!f.good()) {
                cout << "danh sach chua co. Tao moi (C/K) ?";
                char traloi = getch();
                if (toupper(traloi) == 'C') return;
                else {
```

```
                f.close() ;
                f.open(fname, ios::out | ios::binary);
            }
        }
        int stt = 0
        while (1) {
            cout << "\nBổ sung sinh viên thứ: " << stt+1;
            cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
            if (x.hoten[0] == 0) break;
            cout << "\nTuổi: "; cin >> x.tuoi;
            cout << "\nĐiểm: "; cin >> x.diem;
            f.write((char*)(&x), size);
            stt++;
        }
        sosv += stt;
        f.close();
}

void Sinhvien::xemsua()
{
        fstream f;
        int ch;
        f.open(fname, ios::out | ios::app | ios::binary);
        if (!f.good()) {
            cout << "danh sach chua co";
            getch(); return;
        }
        cout << "\nDanh sách sinh viên" << endl;
        int stt ;
        while (1) {
            cout << "\nCần xem (sua) sinh viên thứ (0: dừng): " ;
            cin >> stt;
            if (stt < 1 || stt > sosv) break;
            f.seekg((stt-1) * size, ios::beg);
```

```
            f.read((char*)(&x), size);
            cout << "\nHọ tên: " << x.hoten;
            cout << "\nTuổi: " << x.tuoi;
            cout << "\nĐiểm: " << x.diem;
            cout << "Có sửa không (C/K) ?";
            cin >> traloi;
            if (toupper(traloi) == 'C') {
                f.seekg(-size, ios::cur);
                cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
                cout << "\nTuổi: "; cin >> x.tuoi;
                cout << "\nĐiểm: "; cin >> x.diem;
                f.write((char*)(&x), size);
            }
        }
        f.close();
}

void main()
{
        int chon;
        Sinhvien SV("DSSV") ;
        while (1) {
            clrscr();
            cout << "\n1: Tạo danh sách sinh viên";
            cout << "\n2: Bổ sung danh sách";
            cout << "\n3: Xem – sửa danh sách";
            cout << "\n0: Kết thúc";
            chon = getch();
            chon = chon – 48;
            clrscr();
            if (chon == 1) SV.tao();
            else if (chon == 2) SV.bosung();
            else if (chon == 3) SV.xemsua();
            else break;
```

```
        }
    }
```

BÀI TẬP

1. Viết chương trình đếm số dòng của một file văn bản.

2. Viết chương trình đọc in từng kí tự của file văn bản ra màn hình, mỗi màn hình 20 dòng.

3. Viết chương trình tìm xâu dài nhất trong một file văn bản.

4. Viết chương trình ghép một file văn bản thứ hai vào file văn bản thứ nhất, trong đó tất cả chữ cái của file văn bản thứ nhất phải đổi thành chữ in hoa.

5. Viết chương trình in nội dung file ra màn hình và cho biết tổng số chữ cái, tổng số chữ số đã xuất hiện trong file.

6. Cho 2 file số thực (đã được sắp tăng dần). In ra màn hình dãy số xếp tăng dần của cả 2 file. (Cần tạo cả 2 file dữ liệu này bằng Editor của C++).

7. Viết hàm nhập 10 số thực từ bàn phím vào file INPUT.DAT. Viết hàm đọc các số thực từ file trên và in tổng bình phương của chúng ra màn hình.

8. Viết hàm nhập 10 số nguyên từ bàn phím vào file văn bản tên INPUT.DAT. Viết hàm đọc các số nguyên từ file trên và ghi những số chẵn vào file EVEN.DAT còn các số lẻ vào file ODD.DAT.

9. Nhập bằng chương trình 2 ma trận số nguyên vào 2 file văn bản. Hãy tạo file văn bản thứ 3 chứa nội dung của ma trận tích của 2 ma trận trên.

10. Tổ chức quản lý file sinh viên (Họ tên, ngày sinh, giới tính, điểm) với các chức năng : Nhập, xem, xóa, sửa, tính điểm trung chung.

11. Thông tin về một nhân viên trong cơ quan bao gồm : họ và tên, nghề nghiệp, số điện thoại, địa chỉ nhà riêng. Viết hàm nhập từ bàn phím thông tin của 7 nhân viên và ghi vào file INPUT.DAT. Viết hàm tìm trong file INPUT.DAT và in ra thông tin của 1 nhân viên theo số điện thoại được nhập từ bàn phím.

MỤC LỤC

Chương 3. CẤU TRÚC ĐIỀU KHIỂN VÀ DỮ LIỆU KIỂU MẢNG

Chương 4. HÀM VÀ CHƯƠNG TRÌNH

Chương 5. DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

TÀI LIỆU THAM KHẢO

1. B.W. Kerninghan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978. Ngô Trung Việt (dịch). *Ngôn ngữ lập trình C*. Viện Tin học, Hà Nội 1990.
2. Peter Norton. *Advanced C Programming*. Nguyễn Việt Hải (dịch). *Lập trình C nâng cao*. Nhà xuất bản Giao thông vận tải. Hà Nội, 1995.
3. Phạm Văn Ất. *Kỹ thuật lập trình C. Cơ sở và nâng cao*. Nhà xuất bản Khoa học và kỹ thuật. Hà Nội, 1996.
4. Phạm Văn Ất. *C++ và lập trình hướng đối tượng*. Nhà xuất bản Khoa học và kỹ thuật. Hà Nội, 2000.
5. Scott Robert Ladd. Nguyễn Hùng (dịch). *C++ Kỹ thuật và ứng dụng*. Công ty cổ phần tư vấn và dịch vụ KHKT - SCITEC, 1992.
6. Jan Skansholm. C++ From the Beginning. Addison-Wesley, 1997.

Bài 2. Ngôn ngữ lập trình C++

I. Giới thiệu

- Ngôn ngữ lập trình C++ là ngôn ngữ được phát triển dựa trên ngôn ngữ lập trình C.
- Do đó về cơ bản, cú pháp của C++ giống với cú pháp của C. Tuy nhiên nó có một số mở rộng sau đây:
  - Nhập, xuất dữ liệu (cout, cin)
  - Hàm có đối mặc định, hàm có đối tham chiếu
  - Nạp chồng hàm (hay tải bội hàm – overload function)
  - Hàm mẫu
  - Lớp (có khả năng xây dựng các chương trình HĐT)

II. Nhập xuất dữ liệu

- Nhập dữ liêu kiểu số

  cin>>Tênbiến1>>Tênbiến2>>…>>Tênbiếnn;

  Ví dụ:

```
float x,y;
int m, n;
cin>>x>>y;
cin>>m;
cin>>n;
```

- Nhập dữ liệu kiểu xâu ký tự

cin.ignore(1);

cin.get(Tênbiến, n); //n là số ký tự tối đa cần gán cho biến

Ví dụ:

```
char ht[30];
char w[10];
cin.ignore(1);
cin.get(ht, 30);
cin.ignore(1);
cin.get(w, 5);
```

- Xuất dữ liệu

cout<<Bthức1<<Bthức2<<…<Bthứcn;

Ví dụ:

```
#include”iostream.h”

    float x, y=10;
    cout<<“Nhap x=“;
    cin>>x;
    cout<<“x+y=“<<x+y;
    cout<<“x-y=“<<x-y;
```

III. Hàm

- Khi xây dựng các hàm ngoài các kiểu hàm như trong C thì C++ còn cho phép xây dựng các kiểu hàm sau đây:
  - Đối tham chiếu
  - Đối mặc định
  - Nạp chồng hàm (overload function)
  - Hàm mẫu (template)

- **Hàm có đối tham chiếu**

  - Khai báo hàm:

    DataType Func_Name(DataType & Arg_Nam,..);

  - Sử dụng hàm: Các đối thực sự tương ứng với đối tham chiếu phải là các biến cùng kiểu.

  - Sự hoạt động của hàm như hàm có đối con trỏ

Ví dụ: Xây dựn hàm hoán đổi giá trị của hai biến

```
void hoandoi(float &a, float &b)
{
        float tg;
        tg = a;
        a = b;
        b = tg;
}
```

```
void main(){
float x, y;
cout<<“ Nhap x, y: ”;
cin>>x>>y;
cout<< “x = “<< x <<“ y = “<<y;
hoandoi(x,y);
cout<< “x = “<< x <<“ y = “<<y;
getch(); }
```

```
#include <iostream.h>
#include <conio.h>
void duplicate (int& a, int& b, int& c)
 { a = 2; b = 2; c = 2; }
 int main (){
        int x=1, y=3, z=7;
       duplicate (x, y, z);
       cout << "x=" << x << ", y=" << y << ", z=" << z;
       return 0;
}
```

- **Hàm có đối mặc định**

- Khai báo hàm

DataType Func_Name(DataType Arg_Nam1, DataType Arg_Nam2 = value2, ...);

- Sử dụng hàm: Có thể không truyền đối thực sự cho đối mặc định
- Nếu truyền thì hàm nhận giá trị của đối thực sự, nếu không truyền hàm nhận giá trị mặc định

```
Func_Name(Arg1, Arg2);
Func_Name(Arg1);
```

Ví dụ:

```
#include <iostream.h>
#include <conio.h>
int divide (int a, int b=2)
{ int r; r=a/b;
 return (r); }
```

```
int main () {
      cout << divide (12);
      cout <<"\n"<< divide (12, 4);
      return 0;
}
```

Kết quả

```
6
3
```

• Nạp chồng hàm

- Nạp chồng hàm là khả năng cho phép định nghĩa lại một hàm đã có. Tức là trong một chương trình cho phép nhiều hàm trùng tên nhau.

- **Một số lưu ý khi nạp chồng hàm**

Các hàm phải có ít nhất một trong các đặc điểm sau:

- Khác nhau về số lượng đối
- Khác nhau về kiểu của đối

Ví dụ: Xây dựng hàm nhân, chia hai số có cùng tên hàm

```
#include <iostream.h>
#include <conio.h>
int operate (int a, int b)
{ return (a*b); }
float operate (float a, int b)
{ return (a/b); }
int main ()
{ int x=5,y=2;
  float n=5.0,m=2.0;
  cout << operate (x,y);
  cout << "\n" << operate (n,m);
  return 0;
}
```

- **Ví du:**

  - Hàm nhập một dãy số

    void Nhapday(float *, int);

    void Nhapday(int *, int);

  - Hàm tìm uscln của hai số nguyên

  int uscln(int,int);

  long uscln(long, long);

  long uscln(long, int);

Hàm mẫu (template)

- Hàm mẫu là hàm được xây dựng như là một mẫu để thực hiện một chức năng nào đó mà kiểu của các đối vào chưa được xác định.
- **Khai báo**

```
template<class DataType,…>
DataType  Func_Name(DataType Arg_Name,…){
    các câu lệnh;
};
```

Trong đó DataType là một tên kiểu bất kỳ do người lập trình đặt

Ví dụ 1

```
#include <iostream.h>
#include <conio.h>
template <class T>
T GetMax (T a, T b)
{ T result;
    result = (a>b)? a : b;
    return (result);
 }
```

```
int main () {
int i=5, j=6, k;
long l=10, m=5, n;
k = GetMax(i,j);
n = GetMax(l,m);
cout << k << endl;
cout << n << endl;
return 0;
}
```

Ví dụ 2

Xây dựng hàm nhập, in một dãy số có kiểu bất kỳ

```
template<class T>
void Nhapday(T *a, int n, char ch){
    for(int i=0; i<n; i++){
        cout<<ch<<"["<<i<<"]=";
        cin>>a[i]; }
 }

template<class D>
void Inday(D *a, int n){
   for(int i=0; i<n; i++){
        cout<<a[i] <<"  ";
}
```

```
void main(){
        int m,n;
        float a[100];
        long  b[100];
        cout<<"Nhap m,n:";
        cin>>m>>n;
        Nhapday(a,m,'a');
        Nhapday(b,n,'b');
        Inday(a,n);
        Inday(b,n);
}
```

Hết

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGÔN NGỮ LẬP TRÌNH C++

(Dùng cho sinh viên hệ đào tạo đại học từ xa)

Lưu hành nội bộ

HÀ NỘI - 2006

NGÔN NGỮ LẬP TRÌNH C++

PGS.TS. Trần Đình Quế
KS. Nguyễn Mạnh Hùng

Lập trình nâng cao với C++

Lập trình hướng đối tượng với C++

GIỚI THIỆU

C++ là ngôn ngữ lập trình hướng đối tượng được mở rộng từ ngôn ngữ C. Do vậy, C++ có ưu điểm là kế thừa được các điểm mạnh truyền thống của ngôn ngữ C như uyển chuyển, tương thích với các thiết bị phần cứng. Hiện nay, C++ là một ngôn ngữ lập trình phổ biến, được giảng dạy tại các trường đại học trong nước và trên thế giới và đặc biệt được sử dụng rộng rãi cho nhu cầu phát triển của công nghiệp phần mềm hiện nay. Tài liệu này không những nhằm giới thiệu cho sinh viên ngôn ngữ lập trình C++, mà còn mong muốn qua đó sinh viên có thể hiểu được tư tưởng của phương pháp lập trình hướng đối tượng nói chung. Nội dung của tài liệu bao gồm hai phần chính:

- Phần thứ nhất là lập trình nâng cao với C++, bao gồm lập trình C++ với con trỏ và mảng, các kiểu dữ liệu có cấu trúc cùng các thao tác vào ra trên tệp.
- Phần thứ hai là lập trình hướng đối tượng với C++, bao gồm các định nghĩa và các thao tác trên lớp đối tượng, tính kế thừa và tương ứng bội trong C++, cách sử dụng một số lớp cơ bản trong thư viện C++.

Nội dung tài liệu được tổ chức thành 7 chương:

Chương 1: Giới thiệu tổng quan về các phương pháp lập trình

Trình bày các phương pháp lập trình tuyến tính, lập trình cấu trúc và đặc biệt, làm quen với các khái niệm trong lập trình hướng đối tượng.

Chương 2: Con trỏ và mảng

Trình bày cách khai báo và sử dụng các kiểu con trỏ và mảng trong ngôn ngữ C++.

Chương 3: Kiểu dữ liệu có cấu trúc

Trình bày cách biểu diễn và cài đặt một số kiểu cấu trúc dữ liệu trừu tượng trong C++. Sau đó, trình bày cách áp dụng các kiểu dữ liệu này trong các ứng dụng cụ thể.

Chương 4: Vào ra trên tệp

Trình bày các thao tác đọc, ghi dữ liệu trên các tệp tin khác nhau: tệp tin văn bản và tệp tin nhị phân. Trình bày các cách truy nhập tệp tin trực tiếp.

Chương 5: Lớp đối tượng

Trình bày các khái niệm mở đầu cho lập trình hướng đối tượng trong C++, bao gồm cách khai báo và sử dụng lớp, các thuộc tính của lớp; cách khởi tạo và huỷ bỏ đối tượng, các quy tắc truy nhập đến các thành phần của lớp.

Chương 6: Tính kế thừa và tương ứng bội

Trình bày cách thức kế thừa giữa các lớp trong C++, các nguyên tắc truy nhập trong kế thừa, định nghĩa nạp chồng các phương thức và tính đa hình trong lập trình hướng đối tương với C++.

Chương 7: Một số lớp quan trọng

Trình bày cách sử dụng một số lớp có sẵn trong thư viện chuẩn của C++, bao gồm các lớp: lớp tập hợp, lớp chuỗi, lớp ngăn xếp, lớp hàng đợi và lớp danh sách liên kết.

Để đọc được cuốn sách này, sinh viên phải quen biết các khái niệm cơ bản về lập trình, có một số kỹ năng lập trình với ngôn ngữ C hoặc C++. Cuốn sách này cũng có thể dùng tài liệu tham khảo cho những sinh viên muốn tìm hiểu các kỹ thuật lập trình nâng cao và lập trình hướng đối tượng

với C++. Cuốn sách này có kèm theo một đĩa chương trình chứa toàn bộ các chương trình được lấy làm minh hoạ và các bài tập trong cuốn sách.

Mặc dù các tác giả đã có nhiều cố gắng trong việc biên soạn tài liệu này, song không thể tránh khỏi những thiếu sót. Rất mong nhận được những ý kiến đóng góp quý báu từ các sinh viên và các bạn đồng nghiệp.

CHƯƠNG 1

GIỚI THIỆU VỀ CÁC PHƯƠNG PHÁP LẬP TRÌNH

Nội dung của chương này tập trung trình bày các phương pháp lập trình:

- Phương pháp lập trình tuyến tính
- Phương pháp lập trình hướng cấu trúc
- Phương pháp lập trình hướng đối tượng.

1.1 LẬP TRÌNH TUYẾN TÍNH

Đặc trưng cơ bản của lập trình tuyến tính là tư duy theo lối tuần tự. Chương trình sẽ được thực hiện theo thứ tự từ đầu đến cuối, lệnh này kế tiếp lệnh kia cho đến khi kết thúc chương trình.

Đặc trưng

Lập trình tuyến tính có hai đặc trưng:

- **Đơn giản**: chương trình được tiến hành đơn giản theo lối tuần tự, không phức tạp.
- **Đơn luồng**: chỉ có một luồng công việc duy nhất, và các công việc được thực hiện tuần tự trong luồng đó.

Tính chất

- **Ưu điểm**: Do tính đơn giản, lập trình tuyến tính được ứng dụng cho các chương trình đơn giản và có ưu điểm dễ hiểu.
- **Nhược điểm**: Với các ứng dụng phức tạp, người ta không thể dùng lập trình tuyến tính để giải quyết.

Ngày nay, lập trình tuyến tính chỉ tồn tại trong phạm vi các modul nhỏ nhất của các phương pháp lập trình khác. Ví dụ trong một chương trình con của lập trình cấu trúc, các lệnh cũng được thực hiện theo tuần tự từ đầu đến cuối chương trình con.

1.2 LẬP TRÌNH HƯỚNG CẤU TRÚC

1.2.1 Đặc trưng của lập trình hướng cấu trúc

Trong lập trình hướng cấu trúc, chương trình chính được chia nhỏ thành các chương trình con và mỗi chương trình con thực hiện một công việc xác định. Chương trình chính sẽ gọi đến chương trình con theo một giải thuật, hoặc một cấu trúc được xác định trong chương trình chính.

Các ngôn ngữ lập trình cấu trúc phổ biến là Pascal, C và C++. Riêng C++ ngoài việc có đặc trưng của lập trình cấu trúc do kế thừa từ C, còn có đặc trưng của lập trình hướng đối tượng. Cho nên C++ còn được gọi là ngôn ngữ lập trình nửa cấu trúc, nửa hướng đối tượng.

Đặc trưng

Đặc trưng cơ bản nhất của lập trình cấu trúc thể hiện ở mối quan hệ:

Chương trình = Cấu trúc dữ liệu + Giải thuật

Trong đó:

- **Cấu trúc dữ liệu** là cách tổ chức dữ liệu cho việc xử lý bởi một hay nhiều chương trình nào đó.
- **Giải thuật** là một quy trình để thực hiện một công việc xác định

Trong chương trình, giải thuật có quan hệ phụ thuộc vào cấu trúc dữ liệu:

- Một cấu trúc dữ liệu chỉ phù hợp với một số hạn chế các giải thuật.
- Nếu thay đổi cấu trúc dữ liệu thì phải thay đổi giải thuật cho phù hợp.
- Một giải thuật thường phải đi kèm với một cấu trúc dữ liệu nhất định.

Tính chất

- Mỗi chương trình con có thể được gọi thực hiện nhiều lần trong một chương trình chính.
- Các chương trình con có thể được gọi đến để thực hiện theo một thứ tự bất kì, tuỳ thuộc vào giải thuật trong chương trình chính mà không phụ thuộc vào thứ tự khai báo của các chương trình con.
- Các ngôn ngữ lập trình cấu trúc cung cấp một số cấu trúc lệnh điều khiển chương trình.

Ưu điểm

- Chương trình sáng sủa, dễ hiểu, dễ theo dõi.
- Tư duy giải thuật rõ ràng.

Nhược điểm

- Lập trình cấu trúc không hỗ trợ mạnh việc sử dụng lại mã nguồn: Giải thuật luôn phụ thuộc chặt chẽ vào cấu trúc dữ liệu, do đó, khi thay đổi cấu trúc dữ liệu, phải thay đổi giải thuật, nghĩa là phải viết lại chương trình.
- Không phù hợp với các phần mềm lớn: tư duy cấu trúc với các giải thuật chỉ phù hợp với các bài toán nhỏ, nằm trong phạm vi một modul của chương trình. Với dự án phần mềm lớn, lập trình cấu trúc tỏ ra không hiệu quả trong việc giải quyết mối quan hệ vĩ mô giữa các modul của phần mềm.

Vấn đề

Vấn đề cơ bản của lập trình cấu trúc là bằng cách nào để phân chia chương trình chính thành các chương trình con cho phù hợp với yêu cầu, chức năng và mục đích của mỗi bài toán. Thông thường, để phân rã bài toán trong lập trình cấu trúc, người ta sử dụng phương pháp thiết kế trên xuống (top-down).

1.2.2 Phương pháp thiết kế trên xuống (top-down)

Phương pháp thiết kế top-down tiếp cận bài toán theo hướng từ trên xuống dưới, từ tổng qúat đến chi tiết. Theo đó, một bài toán được chia thành các bài toán con nhỏ hơn. Mỗi bài toán con lại được chia nhỏ tiếp, nếu có thể, thành các bài toán con nhỏ hơn nữa. Quá trình này còn được gọi là quá trình làm mịn dần. Quá trình này sẽ dừng lại khi các bài toán con không cần chia nhỏ thêm

nữa. Nghĩa là khi mỗi bài toán con đều có thể giải quyết bằng một chương trình con với một giải thuật đơn giản.

Ví dụ, sử dụng phương pháp top-down để giải quyết bài toán xây một căn nhà mới. Chúng ta có thể phân rã bài toán theo các bước như sau:

- Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như *làm móng*, đổ cột, đổ trần, xây tường, lợp mái.
- Ở mức thứ hai, phân rã các công việc ở mức thứ nhất như việc *làm móng* nhà có thể phân rã tiếp thành các công việc *đào móng*, gia cố nền, làm khung sắt, đổ bê tong; công việc đổ cột được phần rã thành …
- Ở mức thứ ba, phân rã các công việc của mức thứ hai như việc *đào móng* có thể phân chia tiếp thành các công việc như đo đạc, cắm mốc, chăng dây, đào và kiểm tra móng. Việc gia cố nền được phân rã thành …

Quá trình phân rã có thể dừng ở mức này, bởi vì các công việc con thu được như đo đạc, cắm mốc, chăng dây, đào… có thể thực hiện được ngay, không cần chia nhỏ thêm nữa.

Lưu ý:

- Cùng sử dụng phương pháp top-down với cùng một bài toán, nhưng có thể cho ra nhiều kết quả khác nhau. Nguyên nhân là do sự khác nhau trong tiêu chí để phân rã một bài toán thành các bài toán con.

Ví dụ, vẫn áp dụng phương pháp top-down để giải quyết bài toán xây nhà, nhưng nếu sử dụng một cách khác để phân chia bài toán, ta có thể thu được kết quả khác biệt so với phương pháp ban đầu:

- Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như *làm phần gỗ*, làm phần sắt, làm phần bê tông và làm phần gạch.
- Ở mức thứ hai, phân rã các công việc ở mức thứ nhất là *làm phần gỗ* có thể chia thành các công việc như xẻ gỗ, gia công gỗ, tạo khung, lắp vào nhà. Việc làm sắt có thể chia nhỏ thành…

Rõ ràng, với cách làm mịn thế này, ta sẽ thu được một kết quả khác hẳn với cách thức đã thực hiện ở phần trên.

1.3 LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1.3.1 Lập trình hướng đối tượng

Trong lập trình hướng đối tượng:

- Người ta coi các thực thể trong chương trình là các đối tượng và sau đó trừu tượng hoá đối tượng thành lớp đối tượng.
- Dữ liệu được tổ chức thành các thuộc tính của lớp. Người ta ngăn chặn việc thay đổi tuỳ tiện dữ liệu trong chương trình bằng các cách giới hạn truy nhập như chỉ cho phép truy nhập dữ liệu thông qua đối tượng, thông qua các phương thức mà đối tượng được cung cấp…
- Quan hệ giữa các đối tượng là quan hệ ngang hàng hoặc quan hệ kế thừa: Nếu lớp B kế thừa từ lớp A thì A được gọi là *lớp cơ sở* và B được gọi là *lớp dẫn xuất*.

Ngôn ngữ lập trình hướng đối tượng phổ biến hiện nay là Java, C++, C#...Mặc dù C++ cũng có những đặc trưng cơ bản của lập trình hướng đối tượng nhưng vẫn không phải là ngôn ngữ lập trình thuần hướng đối tượng.

Đặc trưng

Lập trình hướng đối tượng có hai đặc trưng cơ bản:

- **Đóng gói dữ liệu**: dữ liệu luôn được tổ chức thành các thuộc tính của lớp đối tượng. Việc truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp.
- **Sử dụng lại mã nguồn**: việc sử dụng lại mã nguồn được thể hiện thông qua cơ chế kế thừa. Cơ chế này cho phép các lớp đối tượng có thể kế thừa từ các lớp đối tượng khác. Khi đó, trong các lớp dẫn xuất, có thể sử dụng các phương thức (mã nguồn) của các lớp cơ sở mà không cần phải định nghĩa lại.

Ưu điểm

Lập trình hướng đối tượng có một số ưu điểm nổi bật:

- Không còn nguy cơ dữ liệu bị thay đổi tự do trong chương trình. Vì dữ liệu đã được đóng gói vào các đối tượng. Nếu muốn truy nhập vào dữ liệu phải thông qua các phương thức được cho phép của đối tượng.
- Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi mã nguồn của các đối tượng khác, mà chỉ cần thay đổi một số thành phần của đối tượng dẫn xuất. Điều này hạn chế sự ảnh hưởng xấu của việc thay đổi dữ liệu đến các đối tượng khác trong chương trình.
- Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên, chi phí thời gian. Vì nguyên tắc kế thừa cho phép các lớp dẫn xuất sử dụng các phương thức từ lớp cơ sở như những phương thức của chính nó, mà không cần thiết phải định nghĩa lại.
- Phù hợp với các dự án phần mềm lớn, phức tạp.

1.3.2 Một số khái niệm cơ bản

Trong mục này, chúng ta sẽ làm quen với một số khái niệm cơ bản trong lập trình hướng đối tượng. Bao gồm:

- Khái niệm đối tượng (object)
- Khái niệm đóng gói dữ liệu (encapsulation)
- Khái niệm kế thừa (inheritance)
- Khái niệm đa hình (polymorphism)

Đối tượng (Object)

Trong lập trình hướng đối tượng, đối tượng được coi là đơn vị cơ bản nhỏ nhất. Các dữ diệu và cách xử lí chỉ là thành phần của đối tượng mà không được coi là thực thể. Một đối tượng chứa các dữ liệu của riêng nó, đồng thời có các phương thức (hành động) thao tác trên các dữ liệu đó:

Đối tượng = dữ liệu + phương thức

Lớp (Class)

Khi có nhiều đối tượng giống nhau về mặt dữ liệu và phương thức, chúng được nhóm lại với nhau và gọi chung là lớp:

- Lớp là sự trừu tượng hoá của đối tượng
- Đối tượng là một thể hiện của lớp.

Đóng gói dữ liệu (Encapsulation)

- Các dữ liệu được đóng gói vào trong đối tượng. Mỗi dữ liệu có một phạm vi truy nhập riêng.
- Không thể truy nhập đến dữ liệu một cách tự do như lập trình cấu trúc
- Muốn truy nhập đến các dữ liệu đã được bảo vệ, phải thông qua các đối tượng, nghĩa là phải sử dụng các phương thức mà đối tượng cung cấp mới có thể truy nhập đến dữ liệu của đối tượng đó.

Tuy nhiên, vì C++ chỉ là ngôn ngữ lập trình nửa đối tượng, cho nên C++ vẫn cho phép định nghĩa các biến dữ liệu và các hàm tự do, đây là kết quả kế thừa từ ngôn ngữ C, một ngôn ngữ lập trình thuần cấu trúc.

Kế thừa (Inheritance)

Tính kế thừa của lập trình hướng đối tượng cho phép một lớp có thể kế thừa từ một số lớp đã tồn tại. Khi đó, lớp mới có thể sử dụng dữ liệu và phương thức của các lớp cơ sở như là của mình. Ngoài ra, lớp dẫn xuất còn có thể bổ sung thêm một số dữ liệu và phương thức. Ưu điểm của kế thừa là khi thay đổi dữ liệu của một lớp, chỉ cần thay đổi các phương thức trong phạm vi lớp cơ sở mà không cần thay đổi trong các lớp dẫn xuất.

Đa hình (Polymorphsim)

Đa hình là khái niệm luôn đi kèm với kế thừa. Do tính kế thừa, một lớp có thể sử dụng lại các phương thức của lớp khác. Tuy nhiên, nếu cần thiết, lớp dẫn xuất cũng có thể định nghĩa lại một số phương thức của lớp cơ sở. Đó là sự nạp chồng phương thức trong kế thừa. Nhờ sự nạp chồng phương thức này, ta chỉ cần gọi tên phương thức bị nạp chồng từ đối tượng mà không cần quan tâm đó là đối tượng của lớp nào. Chương trình sẽ tự động kiểm tra xem đối tượng là thuộc kiểu lớp cơ sở hay thuộc lớp dẫn xuất, sau đó sẽ gọi phương thức tương ứng với lớp đó. Đó là tính đa hình.

1.3.3 Lập trình hướng đối tượng trong C++

Vì C++ là một ngôn ngữ lập trình được mở rộng từ một ngôn ngữ lập trình cấu trúc C nên C++ được xem là ngôn ngữ lập trình nửa hướng đối tượng, nửa hướng cấu trúc.

Những đặc trưng hướng đối tượng của C++

- Cho phép định nghĩa lớp đối tượng.
- Cho phép đóng gói dữ liệu vào các lớp đối tượng. Cho phép định nghĩa phạm vi truy nhập dữ liệu của lớp bằng các từ khoá phạm vi: **`public, protected, private`**.

- Cho phép kế thừa lớp với các kiểu kế thừa khác nhau tuỳ vào từ khoá dẫn xuất.
- Cho phép lớp dẫn xuất sử dụng các phương thức của lớp cơ sở (trong phạm vi quy định).
- Cho phép định nghĩa chồng phương thức trong lớp dẫn xuất.

Những hạn chế hướng đối tượng của C++

Những hạn chế này là do C++ được phát triển từ một ngôn ngữ lập trình thuần cấu trúc C.

- Cho phép định nghĩa và sử dụng các biến dữ liệu tự do.
- Cho phép định nghĩa và sử dụng các hàm tự do.
- Ngay cả khi dữ liệu được đóng gói vào lớp, dữ liệu vẫn có thể truy nhập trực tiếp như dữ liệu tự do bởi các hàm bạn, lớp bạn (friend) trong C++.

TỔNG KẾT CHƯƠNG 1

Chương 1 đã trình bày tổng quan về các phương pháp lập trình hiện nay. Nội dung tập trung vào ba phương pháp lập trình có liên quan trực tiếp đến ngôn ngữ lập trình C++:

- Lập trình tuyến tính
- Lập trình hướng cấu trúc
- Lập trình hướng đối tượng.

C++ là ngôn ngữ lập trình được mở rộng từ ngôn ngữ lập trình cấu trúc C. Do đó, C++ vừa có những đặc trưng của lập trình cấu trúc, vừa có những đặc trưng của lập trình hướng đối tượng.

CHƯƠNG 2

CON TRỎ VÀ MẢNG

Nội dung của chương này tập trung trình bày các vấn đề cơ bản liên quan đến các thao tác trên kiểu dữ liệu con trỏ và mảng trong C++:

- Khái niệm con trỏ, cách khai báo và sử dụng con trỏ.
- Mối quan hệ giữa con trỏ và mảng
- Con trỏ hàm
- Cấp phát bộ nhớ cho con trỏ

2.1 KHÁI NIỆM CON TRỎ

2.1.1 Khai báo con trỏ

Con trỏ là một biến đặc biệt chứa địa chỉ của một biến khác. Con trỏ có cùng kiểu dữ liệu với kiểu dữ liệu của biến mà nó trỏ tới. Cú pháp khai báo một con trỏ như sau:

```
<Kiểu dữ liệu> *<Tên con trỏ>;
```

Trong đó:

- **Kiểu dữ liệu**: Có thể là các kiểu dữ liệu cơ bản của C++, hoặc là kiểu dữ liệu có cấu trúc, hoặc là kiểu đối tượng do người dùng tự định nghĩa.
- **Tên con trỏ**: Tuân theo qui tắc đặt tên biến của C++:
  - Chỉ được bắt đầu bằng một kí tự (chữ), hoặc dấu gạch dưới "_".
  - Bắt đầu từ kí tự thứ hai, có thể có kiểu kí tự số.
  - Không có dấu trống (space bar) trong tên biến.
  - Có phân biệt chữ hoa và chữ thường.
  - Không giới hạn độ dài tên biến.

Ví dụ, để khai báo một biến con trỏ có kiểu là int và tên là pointerInt, ta viết như sau:

```
int *pointerInt;
```

Lưu ý

- Có thể viết dấu con trỏ "*" ngay sau kiểu dữ liệu, nghĩa là hai cách khai báo sau là tương đương:

```
int *pointerInt;
int* pointerInt;
```

- Các cách khai báo con trỏ như sau là sai cú pháp:

```
*int pointerInt;  // Khai báo sai con trỏ
int pointerInt*;  // Khai báo sai con trỏ
```

2.1.2 Sử dụng con trỏ

Con trỏ được sử dụng theo hai cách:

- Dùng con trỏ để lưu địa chỉ của biến để thao tác
- Lấy giá trị của biến do con trỏ trỏ đến để thao tác

Dùng con trỏ để lưu địa chỉ của biến

Bản thân con trỏ sẽ được trỏ vào địa chỉ của một biến có cùng kiểu dữ liệu với nó. Cú pháp của phép gán như sau:

```
<Tên con trỏ> = &<tên biến>;
```

Lưu ý

- Trong phép toán này, tên con trỏ không có dấu "*".

Ví dụ:

```
int x, *px;
px = &x;
```

sẽ cho con trỏ px có kiểu int trỏ vào địa chỉ của biến x có kiểu nguyên. Phép toán `&<Tên biến>` sẽ cho địa chỉ của biến tương ứng.

Lấy giá trị của biến do con trỏ trỏ đến

Phép lấy giá trị của biến do con trỏ trỏ đến được thực hiện bằng cách gọi tên:

```
*<Tên con trỏ>;
```

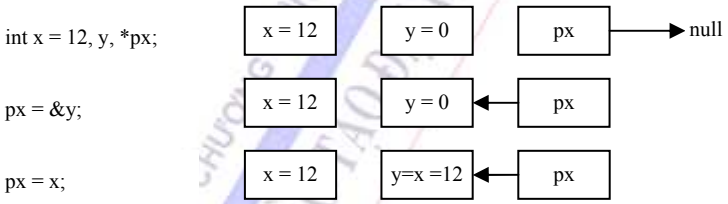
Lưu ý

- Trong phép toán này, phải có dấu con trỏ "*". Nếu không có dấu con trỏ, sẽ trở thành phép lấy địa chỉ của biến do con trỏ trỏ tới.

Ví dụ:

```
int x = 12, y, *px;
px = &y;
*px = x;
```

Quá trình diễn ra như sau:



con trỏ px vẫn trỏ tới địa chỉ biến y và giá trị của biến y sẽ là 12.

Phép gán giữa các con trỏ

Các con trỏ cùng kiểu có thể gán cho nhau thông qua phép gán và lấy địa chỉ con trỏ:

```
<Tên con trỏ 1> = <Tên con trỏ 2>;
```

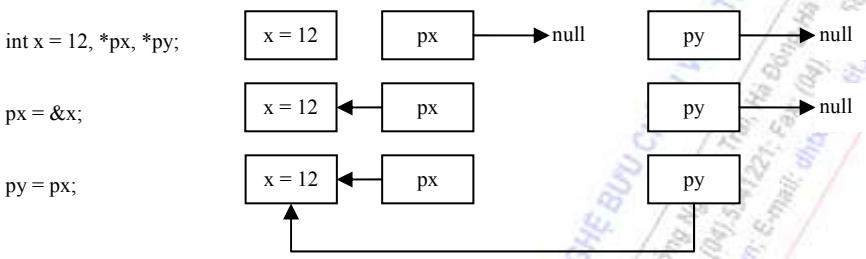
Lưu ý

- Trong phép gán giữa các con trỏ, bắt buộc phải dùng phép lấy **địa chỉ của biến** do con trỏ trỏ tới (không có dấu "*" trong tên con trỏ) mà không được dùng phép lấy **giá trị của biến** do con trỏ trỏ tới.

- Hai con trỏ phải cùng kiểu. Trong trường hợp hai con trỏ khác kiểu, phải sử dụng các phương thức ép kiểu tương tự như trong phép gán các biến thông thường có kiểu khác nhau.

Ví dụ:

```
int x = 12, *px, *py;
px = &x;
py = px;
```



con trỏ py cũng trỏ vào địa chỉ của biến x như con trỏ px. Khi đó *py cũng có giá trị 12 giống như *px và là giá trị của biến x.

Chương trình 2.1 minh hoạ việc dùng con trỏ giữa các biến của một chương trình C++.

Chương trình 2.1

```
#include <stdio.h>
#include <conio.h>
void main(void){
      int x = 12, *px, *py;
      cout << "x = " << x << endl;

      px = &x;          // Con trỏ px trỏ tới địa chỉ của x
      cout << "px = &x, *px = " << *px << endl;

      *px = *px + 20;   // Nội dung của px là 32
      cout << "*px = *px+20, x = " << x << endl;

      py = px;          // Cho py trỏ tới chỗ mà px trỏ: địa chỉ của x
      *py += 15;        // Nội dung của py là 47
      cout << "py = px, *py +=15, x =  " << x << endl;
}
```

Trong chương trình 2.1, ban đầu biến x có giá trị 12. Sau đó, con trỏ px trỏ vào địa chỉ của biến x nên con trỏ px cũng có giá trị 12. Tiếp theo, ta tăng giá trị của con trỏ px thêm 20, giá trị của con trỏ px là 32. Vì px đang trỏ đến địa chỉ của x nên x cũng có giá trị là 32. Sau đó, ta cho con trỏ py trỏ đến vị trí mà px đang trỏ tới (địa chỉ của biến x) nên py cũng có giá trị 32. Cuối cùng, ta tăng giá trị của con trỏ py thêm 15, py sẽ có giá trị 37. Vì py cũng đang trỏ đến địa chỉ của x nên x cũng có giá trị 37. Do đó, ví dụ 2.1 sẽ in ra kết quả như sau:

```
x = 12
px = &x, *px = 12
*px = *px + 20, x = 32
py = px, *py += 15, x = 37
```

2.2 CON TRỎ VÀ MẢNG

2.2.1 Con trỏ và mảng một chiều

Mảng một chiều

Trong C++, tên một mảng được coi là một kiểu con trỏ hằng, được định vị tại một vùng nhớ xác định và địa chỉ của tên mảng trùng với địa chỉ của phần tử đầu tiên của mảng.

Ví dụ khai báo:

```
int A[5];
```

thì địa chỉ của mảng A (cũng viết là A) sẽ trùng với địa chỉ phần tử đầu tiên của mảng A (là &A[0]) nghĩa là:

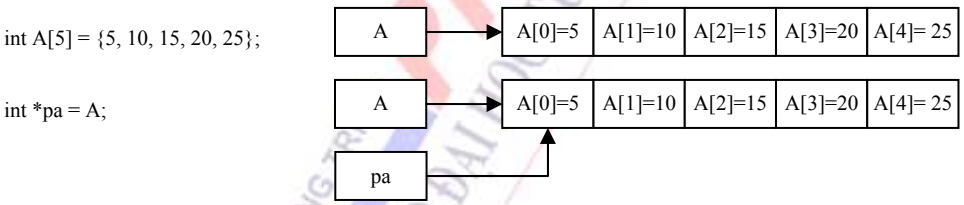
```
A = &A[0];
```

Quan hệ giữa con trỏ và mảng

Vì tên của mảng được coi như một con trỏ hằng, nên nó có thể được gán cho một con trỏ có cùng kiểu.

Ví dụ khai báo:

```
int A[5] = {5, 10, 15, 20, 25};
int *pa = A;
```



thì con trỏ pa sẽ trỏ đến mảng A, tức là trỏ đến địa chỉ của phần tử A[0], cho nên hai khai báo sau là tương đương:

```
pa = A;
pa = &A[0];
```

Với khai báo này, thì địa chỉ trỏ tới của con trỏ pa là địa chỉ của phần tử A[0] và giá trị của con trỏ pa là giá trị của phần tử A[0], tức là *pa = 5;

Phép toán trên con trỏ và mảng

Khi một con trỏ trỏ đến mảng, thì các phép toán tăng hay giảm trên con trỏ sẽ tương ứng với phép dịch chuyển trên mảng.

Ví dụ khai báo:

```
int A[5] = {5, 10, 15, 20, 25};
```

```
int *pa = &A[2];
```

thì con trỏ pa sẽ trỏ đến địa chỉ của phần tử A[2] và giá trị của pa là: `*pa = A[2] = 15`.

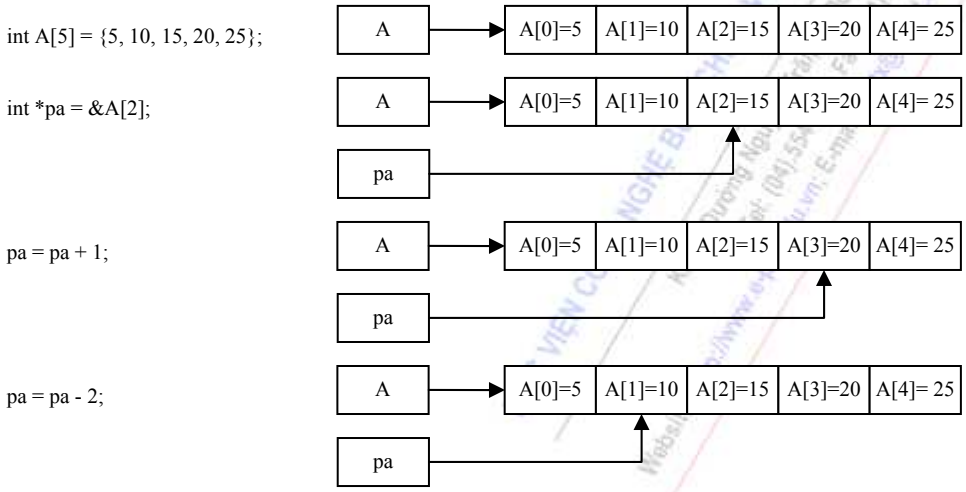
Khi đó, phép toán:

```
pa = pa + 1;
```

sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử tiếp theo của mảng A, đó là địa chỉ của A[3]. Sau đó, phép toán:

```
pa = pa - 2;
```

sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử A[1].



Lưu ý:

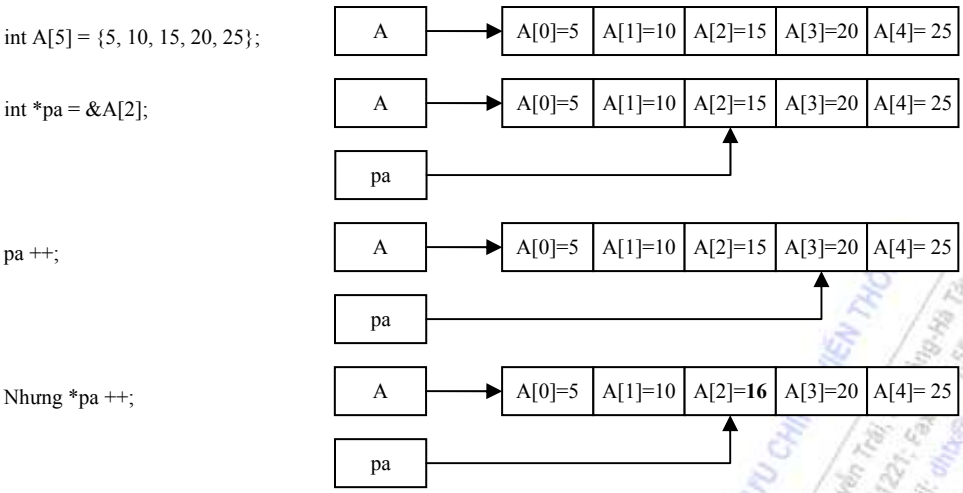
- Hai phép toán pa++ và *pa++ có tác dụng hoàn toàn khác nhau trên mảng, pa++ là thao tác trên con trỏ, tức là trên bộ nhớ, nó sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử tiếp theo của mảng. *pa++ là phép toán trên giá trị, nó tăng giá trị hiện tại của phần tử mảng lên một đơn vị.

  Ví dụ:

  ```
  int A[5] = {5, 10, 15, 20, 25};
  int *pa = &A[2];
  ```

  thì `pa++` là tương đương với `pa = &A[3] và *pa = 20`.

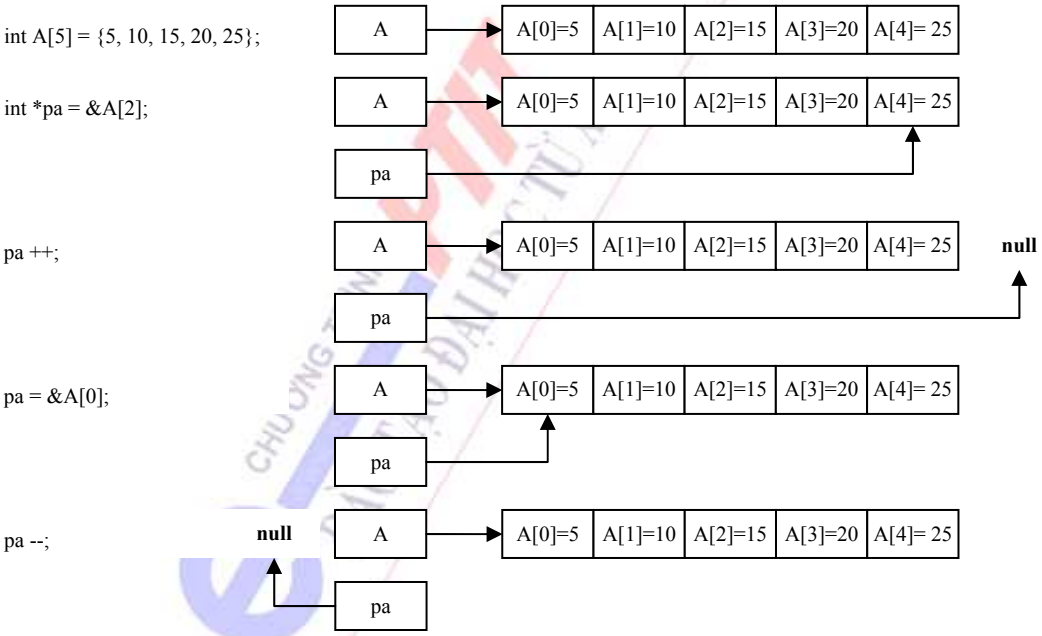
  nhưng `*pa++` lại tương đương với `pa = &A[2] và *pa = 15+1 = 16, A[2] = 16`.



- Trong trường hợp:

```
int A[5] = {5, 10, 15, 20, 25};
int *pa = &A[4];
```

thì phép toán pa++ sẽ đưa con trỏ pa trỏ đến một địa chỉ không xác định. Lí do là A[4] là phần tử cuối của mảng A, nên pa++ sẽ trỏ đến địa chỉ ngay sau địa chỉ của A[4], địa chỉ này nằm ngoài vùng chỉ số của mảng A nên không xác định. Tương tự với trường hợp pa=&A[0], phép toán pa-- cũng đưa pa trỏ đến một địa chỉ không xác định.



- Vì mảng A là con trỏ hằng, cho nên không thể thực hiện các phép toán trên A mà chỉ có thể thực hiện trên các con trỏ trỏ đến A: các phép toán pa++ hoặc pa--là hợp lệ, nhưng các phép toán A++ hoặc A--là không hợp lệ.

Chương trình 2.2a minh hoạ việc cài đặt một thủ tục sắp xếp các phần tử của một mảng theo cách thông thường.

Chương trình 2.2a

```
void SortArray(int A[], int n){
      int temp;
      for(int i=0; i<n-1; i++)
      for(int j=i+1; j<n; j++)
            if(A[i] > A[j]){
                  temp = A[i];
                  A[i] = A[j];
                  A[j] = temp;
            }
}
```

Chương trình 2.2b cài đặt một thủ tục tương tự bằng con trỏ. Hai thủ tục này có chức năng hoàn toàn giống nhau.

Chương trình 2.2b

```
void SortArray(int *A, int n){
      int temp;
      for(int i=0; i<n-1; i++)
      for(int j=i+1; j<n; j++)
            if(*(A+i) > *(A+j)){
                  temp = *(A+i);
                  *(A+i) = *(A+j);
                  *(A+j) = temp;
            }
}
```

Trong chương trình 2.2b, thay vì dùng một mảng, ta dùng một con trỏ để trỏ đến mảng cần sắp xếp. Khi đó, ta có thể dùng các thao tác trên con trỏ thay vì các thao tác trên các phần tử mảng.

2.2.2 Con trỏ và mảng nhiều chiều

Con trỏ và mảng nhiều chiều

Một câu hỏi đặt ra là nếu một ma trận một chiều thì tương đương với một con trỏ, vậy một mảng nhiều chiều thì tương đương với con trỏ như thế nào?

Xét ví dụ:

```
      int A[3][3] = {
                  {5, 10, 15},
                  {20, 25, 30},
                  {35, 40, 45}
            };
```

Khi đó, địa chỉ của ma trận A chính là địa chỉ của hàng đầu tiên của ma trận A, và cũng là địa chỉ của phần tử đầu tiên của hàng đầu tiên của ma trận A:

- Địa chỉ của ma trận A: `A = A[0] = *(A+0) = &A[0][0];`
- Địa chỉ của hàng thứ nhất: `A[1] = *(A+1) = &A[1][0];`
- Địa chỉ của hàng thứ i: `A[i] = *(A+i) = &A[i][0];`
- Địa chỉ phần tử `&A[i][j] = (*(A+i)) + j;`
- Giá trị phần tử `A[i][j] = *((*(A+i)) + j);`

Như vậy, một mảng hai chiều có thể thay thế bằng một mảng một chiều các con trỏ cùng kiểu:

```
int A[3][3];
```

có thể thay thế bằng:

```
int (*A)[3];
```

Con trỏ trỏ tới con trỏ

Vì một mảng hai chiều int A[3][3] có thể thay thế bằng một mảng các con trỏ int (*A)[3]. Hơn nữa, một mảng int A[3] lại có thể thay thế bằng một con trỏ int *A. Do vậy, một mảng hai chiều có thể thay thế bằng một mảng các con trỏ, hoặc một con trỏ trỏ đến con trỏ. Nghĩa là các cách viết sau là tương đương:

```
int A[3][3];
int (*A)[3];
int **A;
```

2.3 CON TRỎ HÀM

Mặc dù hàm không phải là một biến cụ thể nên không có một địa chỉ xác định. Nhưng trong khi chạy, mỗi một hàm trong C++ cũng có một vùng nhớ xác định, do vậy, C++ cho phép dùng con trỏ để trỏ đến hàm. Con trỏ hàm được dùng để truyền tham số có dạng hàm.

Khai báo con trỏ hàm

Con trỏ hàm được khai báo tương tự như khai báo nguyên mẫu hàm thông thường trong C++, ngoại trừ việc có thêm kí hiệu con trỏ "*" trước tên hàm. Cú pháp khai báo con trỏ hàm như sau:

```
<Kiểu dữ liệu trả về> (*<Tên hàm>)([<Các tham số>]);
```

Trong đó:

- **Kiểu dữ liệu trả về**: là các kiểu dữ liệu thông thường của C++ hoặc kiểu do người dùng tự định nghĩa.
- **Tên hàm**: tên do người dùng tự định nghĩa, tuân thủ theo quy tắc đặt tên biến trong C++.
- **Các tham số**: có thể có hoặc không (phần trong dấu "[]" là tuỳ chọn). Nếu có nhiều tham số, mỗi tham số được phân cách nhau bởi dấu phẩy.

Ví dụ khai báo:

```
int (*Calcul)(int a, int b);
```

là khai báo một con trỏ hàm, tên là Calcul, có kiểu int và có hai tham số cũng là kiểu int.

Lưu ý:

- Dấu "()" bao bọc tên hàm là cần thiết để chỉ ra rằng ta đang khai báo một con trỏ hàm. Nếu không có dấu ngoặc đơn này, trình biên dịch sẽ hiểu rằng ta đang khai báo một hàm thông thường và có giá trị trả về là một con trỏ.

  Ví dụ, hai khai báo sau là khác nhau hoàn toàn:

```
// Khai báo một con trỏ hàm
int (*Calcul)(int a, int b);
// Khai báo một hàm trả về kiểu con trỏ
int *Calcul(int a, int b);
```

Sử dụng con trỏ hàm

Con trỏ hàm được dùng khi cần gọi một hàm như là tham số của một hàm khác. Khi đó, một hàm được gọi phải có khuôn mẫu giống với con trỏ hàm đã được khai báo.

Ví dụ, với khai báo:

```
int (*Calcul)(int a, int b);
```

thì có thể gọi các hàm có hai tham số kiểu int và trả về cũng kiểu int như sau:

```
int add(int a, int b);
int sub(int a, int b);
```

nhưng không được gọi các hàm khác kiểu tham số hoặc kiểu trả về như sau:

```
int add(float a, int b);
int add(int a);
char* sub(char* a, char* b);
```

Chương trình 2.3 minh hoạ việc khai báo và sử dụng con trỏ hàm.

Chương trình 2.3

```
#include <ctype.h>
#include <string>

// Hàm có sử dụng con trỏ hàm như tham số
void Display(char[] str, int (*Xtype)(int c)){
      int index = 0;
      while(str[index] != '\0'){
            cout << (*Xtype)(str[index]); // Sử dụng con trỏ hàm
            index ++;
      }
      return;
}

// Hàm main, dùng lời gọi hàm đến con trỏ hàm
void main(){
      char input[500];
      cout << "Enter the string: ";
      cin >> input;
```

```
      char reply;
      cout << "Display the string in uppercase or lowercase (u,l): ";
      cin >> reply;
      if(reply == 'l')  // Hiển thị theo dạng lowercase
            Display(str, tolower);
      else              // Hiển thị theo dạng uppercase
            Display(str, toupper);
      return;
}
```

Chương trình 2.3 khai báo hàm Display() có sử dụng con trỏ hàm có khuôn mẫu

```
int (*Xtype)(int c);
```

Trong hàm main, con trỏ hàm này được gọi bởi hai thể hiện là các hàm tolower() và hàm toupper(). Hai hàm này được khai báo trong thư viện ctype.h với mẫu như sau:

```
int tolower(int c);
int toupper(int c);
```

Hai khuôn mẫu này phù hợp với con trỏ hàm Xtype trong hàm Display() nên lời gọi hàm Display() trong hàm main là hợp lệ.

2.4 CẤP PHÁT BỘ NHỚ ĐỘNG

Xét hai trường hợp sau đây:

- Trường hợp 1, khai báo một con trỏ và gán giá trị cho nó:

```
int *pa = 12;
```

- Trường hợp 2, khai báo con trỏ đến phần tử cuối cùng của mảng rồi tăng thêm một đơn vị cho nó:

```
int A[5] = {5, 10, 15, 20, 25};
int *pa = &A[4];
pa++;
```

Trong cả hai trường hợp, ta đều không biết thực sự con trỏ pa đang trỏ đến địa chỉ nào trong bộ nhớ: trường hợp 1 chỉ ra rằng con trỏ pa đang trỏ tới một địa chỉ không xác định, nhưng lại chứa giá trị là 12 do được gán vào. Trường hợp 2, con trỏ pa đã trỏ đến địa chỉ ngay sau địa chỉ phần tử cuối cùng của mảng A, đó cũng là một địa chỉ không xác định. Các địa chỉ không xác định này là các địa chỉ nằm ở vùng nhớ tự do còn thừa của bộ nhớ. Vùng nhớ này có thể bị chiếm dụng bởi bất kì một chương trình nào đang chạy.

Do đó, rất có thể các chương trình khác sẽ chiếm mất các địa chỉ mà con trỏ pa đang trỏ tới. Khi đó, nếu các chương trình thay đổi giá trị của địa chỉ đó, giá trị pa cũng bị thay đổi theo mà ta không thể kiểm soát được. Để tránh các rủi ro có thể gặp phải, C++ yêu cầu phải cấp phát bộ nhớ một cách tường minh cho con trỏ trước khi sử dụng chúng.

2.4.1 Cấp phát bộ nhớ động cho biến

Cấp phát bộ nhớ động

Thao tác cấp phát bộ nhớ cho con trỏ thực chất là gán cho con trỏ một địa chỉ xác định và đưa địa chỉ đó vào vùng đã bị chiếm dụng, các chương trình khác không thể sử dụng địa chỉ đó. Cú pháp cấp phát bộ nhớ cho con trỏ như sau:

```
<tên con trỏ> = new <kiểu con trỏ>;
```

Ví dụ, khai báo:

```
int *pa;
pa = new int;
```

sẽ cấp phát bộ nhớ hợp lệ cho con trỏ pa.

Lưu ý:

- Ta có thể vừa cấp phát bộ nhớ, vừa khởi tạo giá trị cho con trỏ theo cú pháp sau:

  ```
  int *pa;
  pa = new int(12);
  ```

  sẽ cấp phát cho con trỏ pa một địa chỉ xác định, đồng thời gán giá trị của con trỏ *pa = 12.

Giải phóng bộ nhớ động

Địa chỉ của con trỏ sau khi được cấp phát bởi thao tác **new** sẽ trở thành vùng nhớ đã bị chiếm dụng, các chương trình khác không thể sử dụng vùng nhớ đó ngay cả khi ta không dùng con trỏ nữa. Để tiết kiệm bộ nhớ, ta phải huỷ bỏ vùng nhớ của con trỏ ngay sau khi không dùng đến con trỏ nữa. Cú pháp huỷ bỏ vùng nhớ của con trỏ như sau:

```
delete <tên con trỏ>;
```

Ví dụ:

```
int *pa = new int(12);  // Khai báo con trỏ pa, cấp phát bộ nhớ
                        // và gán giá trị ban đầu cho pa là 12.
delete pa;              // Giải phóng vùng nhớ vừa cấp cho pa.
```

Lưu ý:

- Một con trỏ, sau khi bị giải phóng địa chỉ, vẫn có thể được cấp phát một vùng nhớ mới hoặc trỏ đến một địa chỉ mới:

  ```
  int *pa = new int(12);  // Khai báo con trỏ pa, cấp phát bộ nhớ
                          // và gán giá trị ban đầu cho pa là 12.
  delete pa;              // Giải phóng vùng nhớ vừa cấp cho pa.
  int A[5] = {5, 10, 15, 20, 25};
  pa = A;                 // Cho pa trỏ đến địa chỉ của mảng A
  ```

- Nếu có nhiều con trỏ cùng trỏ vào một địa chỉ, thì chỉ cần giải phóng bộ nhớ của một con trỏ, tất cả các con trỏ còn lại cũng bị giải phóng bộ nhớ:

  ```
  int *pa = new int(12);  // *pa = 12
  int *pb = pa;           // pb trỏ đến cùng địa chỉ pa.
  *pb += 5;               // *pa = *pb = 17
  delete pa;              // Giải phóng cả pa lẫn pb
  ```

- Một con trỏ sau khi cấp phát bộ nhớ động bằng thao tác **new**, cần phải phóng bộ nhớ trước khi trỏ đến một địa chỉ mới hoặc cấp phát bộ nhớ mới:

```
int *pa = new int(12);  // pa được cấp bộ nhớ và *pa = 12
*pa = new int(15);      // pa trỏ đến địa chỉ khác và *pa = 15.
                        // địa chỉ cũ của pa vẫn bị coi là bận
```

2.4.2 Cấp phát bộ nhớ cho mảng động một chiều

Cấp phát bộ nhớ cho mảng động một chiều

Mảng một chiều được coi là tương ứng với một con trỏ cùng kiểu. Tuy nhiên, cú pháp cấp phát bộ nhớ cho mảng động một chiều là khác với cú pháp cấp phát bộ nhớ cho con trỏ thông thường:

```
<Tên con trỏ> = new <Kiểu con trỏ>[<Độ dài mảng>];
```

Trong đó:

- **Tên con trỏ**: tên do người dùng đặt, tuân thủ theo quy tắc đặt tên biến của C++.
- **Kiểu con trỏ**: Kiểu dữ liệu cơ bản của C++ hoặc là kiểu do người dùng tự định nghĩa.
- **Độ dài mảng**: số lượng các phần tử cần cấp phát bộ nhớ của mảng.

Ví dụ:

```
int *A = new int[5];
```

sẽ khai báo một mảng A có 5 phần tử kiểu int được cấp phát bộ nhớ động.

Lưu ý:

- Khi cấp phát bộ nhớ cho con trỏ có khởi tạo thông thường, ta dùng dấu "()", khi cấp phát bộ nhớ cho mảng, ta dùng dấu "[]". Hai lệnh cấp phát sau là hoàn toàn khác nhau:

```
// Cấp phát bộ nhớ và khởi tạo cho một con trỏ int
int *A = new int(5);
// Cấp phát bộ nhớ cho một mảng 5 phần tử kiểu int
int *A = new int[5];
```

Giải phóng bộ nhớ của mảng động một chiều

Để giải phóng vùng nhớ đã được cấp phát cho một mảng động, ta dùng cú pháp sau:

```
delete [] <tên con trỏ>;
```

Ví dụ:

```
// Cấp phát bộ nhớ cho một mảng có 5 phần tử kiểu int
int *A = new int[5];
// Giải phóng vùng nhớ do mảng A đang chiếm giữ.
delete [] A;
```

Chương trình 2.4 minh hoạ hai thủ tục khởi tạo và giải phóng một mảng động một chiều.

Chương trình 2.4

```
void InitArray(int *A, int length){
      A = new int[length];
      for(int i=0; i<length; i++)
```

```
            A[i] = 0;
      return;
}

void DeleteArray(int *A){
      delete [] A;
      return;
}
```

2.4.3 Cấp phát bộ nhớ cho mảng động nhiều chiều

Cấp phát bộ nhớ cho mảng động nhiều chiều

Một mảng hai chiều là một con trỏ đến một con trỏ. Do vậy, ta phải cấp phát bộ nhớ theo từng chiều theo cú pháp cấp phát bộ nhớ cho mảng động một chiều.

Ví dụ:

```
int **A;
const int length = 10;
A = new int*[length];    // Cấp phát bộ nhớ cho số dòng của ma trận A
for(int i=0; i<length; i++)
      // Cấp phát bộ nhớ cho các phần tử của mỗi dòng
      A[i] = new int[length];
```

sẽ cấp phát bộ nhớ cho một mảng động hai chiều, tương đương với một ma trận có kích thước 10*10.

Lưu ý:

- Trong lệnh cấp phát `A = new int*[length]`, cần phải có dấu "*" để chỉ ra rằng cần cấp phát bộ nhớ cho một mảng các phần tử có kiểu là `con trỏ int (int*)`, khác với kiểu int bình thường.

Giải phóng bộ nhớ của mảng động nhiều chiều

Ngược lại với khi cấp phát, ta phải giải phóng lần lượt bộ nhớ cho con trỏ tương ứng với cột và hàng của mảng động.

Ví dụ:

```
int **A;
…;                          // cấp phát bộ nhớ
…
for(int i=0; i<length; i++)
      delete [] A[i];   // Giải phóng bộ nhớ cho mỗi dòng
delete [] A;            // Giải phóng bộ nhớ cho mảng các dòng
```

sẽ giải phóng bộ nhớ cho một mảng động hai chiều.

Chương trình 2.5 minh hoạ việc dùng mảng động hai chiều để tính tổng của hai ma trận.

Chương trình 2.5

```
#include<stdio.h>
#include<conio.h>

/* Khai báo nguyên mẫu hàm */
void InitArray(int **A, int row, int colum);
void AddArray(int **A, int **B, int row, int colum);
void DisplayArray(int **A, int row, int colum);
void DeleteArray(int **A, int row);

void InitArray(int **A, int row, int colum){
      A = new int*[row];
      for(int i=0; i<row; i++){
            A[i] = new int[colum];
            for(int j=0; j<colum; j++){
                  cout << "Phan tu [" << i << "," << j << "] = ";
                  cin >> A[i][j];
            }
      return;
}

void AddArray(int **A, int **B, int row, int colum){
      for(int i=0; i<row; i++)
      for(int j=0; j<colum; j++)
            A[i][j] += B[i][j];
      return;
}

void DisplayArray(int **A, int row, int colum){
      for(int i=0; i<row; i++){
            for(int j=0; j<colum; j++)
                  cout << A[i][j] << " ";
            cout << endl; // Xuống dòng
      return;
}

void DeleteArray(int **A, int row){
      for(int i=0; i<row; i++)
            delete [] A[i];
      delete [] A;
      return;
}
```

```
void main(){
      clrscr();
      int **A, **B, row, colum;
      cout << "So dong: ";
      cin >> row;
      cout << "So cot: ";
      cin >> colum;

      /* Khởi tạo các ma trận */
      cout << "Khoi tao mang A:" << endl;
      InitArray(A, row, colum);

      cout << "Khoi tao mang B:" << endl;
      InitArray(B, row, colum);

      // Cộng hai ma trận
      AddArray(A, B, row, colum);

      // Hiển thị ma trận kết quả
      cout << "Tong hai mang A va mang B:" << endl;
      DisplayArray(A, row, colum);

      // Giải phóng bộ nhớ
      DeleteArray(A, row);
      DeleteArray(B, row);
      return;
}
```

TỔNG KẾT CHƯƠNG 2

Nội dung chương 2 đã trình bày các vấn đề liên quan đến việc khai báo và sử dụng con trỏ và mảng trong ngôn ngữ C++:

- Con trỏ là một kiểu biến đặc biệt, nó trỏ đến địa chỉ của một biến khác. Có hai cách truy nhập đến con trỏ là truy nhập đến địa chỉ hoặc truy nhập đến giá trị của địa chỉ mà con trỏ trỏ đến.
- Con trỏ có thể tham gia vào các phép toán như các biến thông thường bằng phép lấy giá trị.
- Một con trỏ có sự tương ứng với một mảng một chiều có cùng kiểu.
- Một ma trận hai chiều có thể thay thế bằng một mảng các con trỏ hoặc một con trỏ trỏ đến con trỏ.
- Một con trỏ có thể trỏ đến một hàm, khi đó, nó được dùng để gọi một hàm như là một tham số cho hàm khác.

- Một con trỏ cần phải trỏ vào một địa chỉ xác định hoặc phải được cấp phát bộ nhớ qua phép toán **new** và giải phóng bộ nhớ sau khi dùng bằng thao tác **delete**.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

1. Trong các khai báo con trỏ sau, những khai báo nào là đúng:

   a. `int A*;`

   b. `*int A;`

   c. `int* A, B;`

   d. `int* A, *B;`

   e. `int *A, *B;`

2. Với khai báo:

```
int a = 12;
int *pa;
```

   Các phép gán nào sau đây là hợp lệ:

   a. `pa = &a;`

   b. `pa = a;`

   c. `*pa = &a;`

   d. `*pa = a;`

3. Với khai báo:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A+2;
```

   Khi đó, *pa = ?

   a. 10

   b. 20

   c. 30

   d. 40

   e. 50

4. Với đoạn chương trình:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A;
*pa += 2;
```

   Khi đó, *pa = ?

   a. 10

   b. 12

   c. 30

   d. 32

5. Với đoạn chương trình:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A;
```

```
pa += 2;
```

Khi đó, *pa = ?

a. 10

b. 12

c. 30

d. 32

6. Với đoạn chương trình:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A;
pa += 2;
```

Khi đó, pa = ?

a. `&A[0]`

b. `A[2]`

c. `&A[2]`

d. Không xác định

7. Với đoạn chương trình:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A;
pa -= 2;
```

Khi đó, pa = ?

a. `&A[0]`

b. `&A[2]`

c. `&A[4]`

d. Không xác định

8. Với đoạn chương trình:

```
int A[3][3] = {
                {10, 20, 30},
                {40, 50, 60},
                {70, 80, 90}
                };
int *pa;
```

Khi đó, để có được kết quả *pa = 50, các lệnh nào sau đây là đúng?

a. `pa = A + 4;`

b. `pa = (*(A+1)) + 1;`

c. `pa = &A[1][1];`

d. `pa = *((*(A+1)) + 1);`

9. Giả sử ta khai báo một hàm có sử dụng con trỏ hàm với khuôn mẫu như sau:

```
int Calcul(int a, int b, int (*Xcalcul)(int x, int y)){}
```

Và ta có cài đặt một số hàm như sau:

```
int add(int a, int b);
void cal(int a, int b);
int squere(int a);
```

Khi đó, lời gọi hàm nào sau đây là đúng:

a. `Calcul(5, 10, add);`

b. `Calcul(5, 10, add(2, 3));`

c. `Calcul(5, 10, cal);`

d. `Calcul(5, 10, squere);`

10. Ta muốn cấp phát bộ nhớ cho một con trỏ kiểu int và khởi đầu giá trị cho nó là 20. Lệnh nào sau đây là đúng:

a. `int *pa = 20;`

b. `int *pa = new int{20};`

c. `int *pa = new int(20);`

d. `int *pa = new int[20];`

11. Ta muốn cấp phát bộ nhớ cho một mảng động kiểu int có chiều dài là 20. Lệnh nào sau đây là đúng:

a. `int *pa = 20;`

b. `int *pa = new int{20};`

c. `int *pa = new int(20);`

d. `int *pa = new int[20];`

12. Xét đoạn chương trình sau:

```
int A[5] = {10, 20, 30, 40, 50};
int *pa = A;
pa = new int(2);
```

Khi đó, *pa = ?

a. 10

b. 30

c. 2

d. Không xác định

13. Xét đoạn chương trình sau:

```
1>    int A[5] = {10, 20, 30, 40, 50};
2>    int *pa = A;
3>    pa += 15;
4>    delete pa;
```

Đoạn chương trình trên có lỗi ở dòng nào?

a. 1

b. 2

c. 3

d. 4

14. Viết chương trình thực hiện các phép toán cộng, trừ, nhân, chia trên đa thức. Các đa thức được biểu diễn bằng mảng động một chiều. Bậc của đa thức và các hệ số tương ứng được nhập từ bàn phím.
15. Viết chương trình thực hiện các phép toán cộng, trừ, nhân hai ma trận kích thước m*n. Các ma trận được biểu diễn bằng mảng động hai chiều. Giá trị kích cỡ ma trận (m, n) và giá trị các phần tử của ma trận được nhập từ bàn phím.

CHƯƠNG 3

KIỂU DỮ LIỆU CẤU TRÚC

Nội dung chương này tập trung trình bày các vấn đề liên quan đến kiểu dữ liệu có cấu trúc trong C++:

- Định nghĩa một cấu trúc
- Sử dụng một cấu trúc bằng các phép toán cơ bản trên cấu trúc
- Con trỏ cấu trúc, khai báo và sử dụng con trỏ cấu trúc
- Mảng các cấu trúc, khai báo và sử dụng mảng các cấu trúc
- Một số kiểu dữ liệu trừu tượng khác như ngăn xếp, hàng đợi, danh sách liên kết.

3.1 ĐỊNH NGHĨA CẤU TRÚC

Kiểu dữ liệu có cấu trúc được dùng khi ta cần nhóm một số biến dữ liệu luôn đi kèm với nhau. Khi đó, việc xử lí trên một nhóm các biến được thực hiện như trên các biến cơ bản thông thường.

3.1.1 Khai báo cấu trúc

Trong C++, một cấu trúc do người dùng tự định nghĩa được khai báo thông qua từ khoá **struct**:

```
struct <Tên cấu trúc>{
      <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
      <Kiểu dữ liệu 2> <Tên thuộc tính 2>;
      …
      <Kiểu dữ liệu n> <Tên thuộc tính n>;
};
```

Trong đó:

- **struct**: là tên từ khoá để khai báo một cấu trúc, bắt buộc phải có khi định nghĩa cấu trúc.
- **Tên cấu trúc**: là tên do người dùng tự định nghĩa, tuân thủ theo quy tắc đặt tên biến trong C++. Tên này sẽ trở thành tên của kiểu dữ liệu có cấu trúc tương ứng.
- **Thuộc tính**: mỗi thuộc tính của cấu trúc được khai báo như khai báo một biến thuộc kiểu dữ liệu thông thường, gồm có kiểu dữ liệu và tên biến tương ứng. Mỗi khai báo thuộc tính phải kết thúc bằng dấu chấm phẩy ";" như một câu lệnh C++ thông thường.

Ví dụ, để quản lí nhân viên của một công ty, khi xử lí thông tin về mỗi nhân viên, ta luôn phải xử lí các thông tin liên quan như:

- Tên
- Tuổi
- Chức vụ
- Lương

Do đó, ta sẽ dùng cấu trúc để lưu giữ thông tin về mỗi nhân viên bằng cách định nghĩa một cấu trúc có tên là Employeee với các thuộc tính như sau:

```
struct Employeee{
      char name[20];     // Tên nhân viên
      int age;           // Tuổi nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
};
```

Lưu ý:

- Cấu trúc chỉ cần định nghĩa một lần trong chương trình và có thể được khai báo biến cấu trúc nhiều lần. Khi cấu trúc đã được định nghĩa, việc khai báo biến ở lần khác trong chương trình được thực hiện như khai báo biến thông thường:

  ```
  <Tên cấu trúc> <tên biến 1>, <tên biến 2>;
  ```

  Ví dụ, sau khi đã định nghĩa cấu trúc Employeee, muốn có biến myEmployeee, ta khai báo như sau:

  ```
  Employee myEmployeee;
  ```

3.1.2 Cấu trúc lồng nhau

Các cấu trúc có thể được định nghĩa lồng nhau khi một thuộc tính của một cấu trúc cũng cần có kiểu là một cấu trúc khác. Khi đó, việc định nghĩa cấu trúc cha được thực hiện như một cấu trúc bình thường, với khai báo về thuộc tính đó là một cấu trúc con:

```
struct <Tên cấu trúc cha>{
      <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
      // Có kiểu cấu trúc
      <Kiểu cấu trúc con> <Tên thuộc tính 2>;
      …
      <Kiểu dữ liệu n> <Tên thuộc tính n>;
};
```

Ví dụ, với kiểu cấu trúc Employee, ta không quan tâm đến tuổi nhân viên nữa, mà quan tâm đến ngày sinh của nhân viên. Vì ngày sinh cần có các thông tin luôn đi với nhau là ngày sinh, tháng sinh, năm sinh. Do đó, ta định nghĩa một kiểu cấu trúc con cho kiểu ngày sinh:

```
struct Date{
      int day;
      int month;
      int year;
};
```

khi đó, cấu trúc Employee trở thành:

```
struct Employee{
      char name[20];          // Tên nhân viên
      Date birthDay;          // Ngày sinh của nhân viên
      char role[20];          // Chức vụ của nhân viên
      float salary;           // Lương của nhân viên
};
```

Lưu ý:

- Trong định nghĩa các cấu trúc lồng nhau, cấu trúc con phải được định nghĩa trước cấu trúc cha để đảm bảo các kiểu dữ liệu của các thuộc tính của cấu trúc cha là tường minh tại thời điểm nó được định nghĩa.

3.1.3 Định nghĩa cấu trúc với từ khoá typedef

Để tránh phải dùng từ khoá **struct** mỗi khi khai báo biến cấu trúc, ta có thể dùng từ khóa **typedef** khi định nghĩa cấu trúc:

```
typedef struct {
      <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
      <Kiểu dữ liệu 2> <Tên thuộc tính 2>;
      …
      <Kiểu dữ liệu n> <Tên thuộc tính n>;
} <Tên kiểu dữ liệu cấu trúc>;
```

Trong đó:

- **Tên kiểu dữ liệu cấu trúc**: là tên kiểu dữ liệu của cấu trúc vừa định nghĩa. Tên này sẽ được dùng như một kiểu dữ liệu thông thường khi khai báo biến cấu trúc.

Ví dụ, muốn có kiểu dữ liệu có cấu trúc nhân viên, có tên là Employee, ta dùng từ khoá **typedef** để định nghĩa cấu trúc như sau:

```
typedef struct {
      char name[20];     // Tên nhân viên
      int age;           // Tuổi nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;
```

Khi đó, muốn có hai biến là myEmployee1 và myEmployee2 có kiểu cấu trúc Employee, ta chỉ cần khai báo như sau mà không cần từ khoá **struct**:

```
Employee myEmployee1, myEmployee2;
```

Trong ví dụ khai báo lồng cấu trúc Employee, dùng từ khoá **typedef** cho kiểu Date:

```
typedef struct {
      int day;
      int month;
      int year;
} Date;
```

cấu trúc Employee trở thành:

```
typedef struct {
      char name[20];     // Tên nhân viên
      Date birthDay;     // Ngày sinh của nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;
```

Lưu ý:

- Khi không dùng từ khoá **typedef**, tên cấu trúc (nằm sau từ khoá struct) được dùng để khai báo biến. Trong khi đó, khi có từ khoá **typedef**, tên kiểu dữ liệu cấu trúc (dòng cuối cùng trong định nghĩa) mới được dùng để khai báo biến.
- Khi dùng từ khoá **typedef** thì không thể khai báo biến đồng thời với định nghĩa cấu trúc.

3.2 THAO TÁC TRÊN CẤU TRÚC

Các thao tác trên cấu trúc bao gồm:

- Khai báo và khởi tạo giá trị ban đầu cho biến cấu trúc
- Truy nhập đến các thuộc tính của cấu trúc

3.2.1 Khởi tạo giá trị ban đầu cho cấu trúc

Khởi tạo biến có cấu trúc đơn

Biến cấu trúc được khai báo theo các cách sau:

```
<Tên kiểu dữ liệu cấu trúc> <tên biến>;
```

Ngoài ra, ta có thể khởi tạo các giá trị cho các thuộc tính của cấu trúc ngay khi khai báo bằng các cú pháp sau:

```
<Tên kiểu dữ liệu cấu trúc> <tên biến> = {
      <giá trị thuộc tính 1>,
      <giá trị thuộc tính 2>,
      …
      <giá trị thuộc tính n>
};
```

Trong đó:

- **Giá trị thuộc tính**: là giá trị khởi đầu cho mỗi thuộc tính, có kiểu phù hợp với kiểu dữ liệu của thuộc tính. Mỗi giá trị của thuộc tính được phân cách bằng dấu phẩy “,”.

Ví dụ, với định nghĩa cấu trúc:

```
typedef struct {
      char name[20];     // Tên nhân viên
      int age;           // Tuổi nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;
```

thì có thể khai báo và khởi tạo cho một biến như sau:

```
Employee myEmployee1 = {
      “Nguyen Van A”,
      27,
      “Nhan vien”,
      300f
```

```
};
```

Khởi tạo các biến có cấu trúc lồng nhau

Trong trường hợp các cấu trúc lồng nhau, phép khởi tạo cũng thực hiện như thông thường với phép khởi tạo cho tất cả các cấu trúc con.

Ví dụ với khai báo cấu trúc như sau:

```
typedef struct {
      int day;
      int month;
      int year;
} Date;
```

và:

```
typedef struct {
      char name[20];     // Tên nhân viên
      Date birthDay;     // Ngày sinh của nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;
```

Thì khai báo và khởi tạo một biến có kiểu `Employee` có thể thực hiện như sau:

```
Employee myEmployee1 = {
      "Nguyen Van A",
      {15, 05, 1980},   // Khởi tạo cấu trúc con
      "Nhan vien",
      300f
};
```

3.2.2 Truy nhập đến thuộc tính của cấu trúc

Việc truy nhập đến thuộc tính của cấu trúc được thực hiện bằng cú pháp:

```
<Tên biến cấu trúc>.<tên thuộc tính>
```

Ví dụ, với một biến cấu trúc kiểu Employee đơn:

```
Employee myEmployee1 = {
      "Nguyen Van A",
      27,
      "Nhan vien",
      300f
};
```

ta có thể truy xuất như sau:

```
cout << myEmployee1.name;      // hiển thị ra "Nguyen Van A"
myEmployee1.age += 1;          // Tăng số tuổi lên 1
```

Đối với kiểu cấu trúc lồng nhau, phép truy nhập đến thuộc tính được thực hiện lần lượt từ cấu trúc cha đến cấu trúc con.

Ví dụ, với một biến cấu trúc kiểu Employee lồng nhau:

```
Employee myEmployee1 = {
      "Nguyen Van A",
      {15, 05, 1980},
      "Nhan vien",
      300f
};
```

ta có thể truy xuất như sau:

```
cout << myEmployee1.name;           // hiển thị ra "Nguyen Van A"
myEmployee1.birthDay.day = 16;      // Sửa lại ngày sinh thành 16
myEmployee1.birthDay.month = 07;    // Sửa lại tháng sinh thành 07
```

Chương trình 3.1a minh hoạ việc tạo lập và sử dụng cấu trúc Employee đơn, không dùng từ khoá **typedef**.

Chương trình 3.1a

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

struct Employee{
      char name[20];    // Tên nhân viên
      int age;          // Tuổi nhân viên
      char role[20];    // Chức vụ của nhân viên
      float salary;     // Lương của nhân viên
};

/* Khai báo khuôn mẫu hàm */
void Display(Employee myEmployee);

void Display(Employee myEmployee){
      cout << "Name: " << myEmployee.name << endl;
      cout << "Age: " << myEmployee.age << endl;
      cout << "Role: " << myEmployee.role << endl;
      cout << "Salary: " << myEmployee.salary << endl;
      return;
}

void main(){
      clrscr();
      // Hiển thị giá trị mặc định
      Employee myEmployee =
```

```
            {"Nguyen Van A", 27, "Nhan vien", 300f};
      cout << "Thông tin mặc định:" << endl;
      Display(myEmployee);

      // Thay đổi giá trị cho các thuộc tính
      cout << "Name: ";
      cin >> myEmployee.name;
      cout << "Age: ";
      cin >> myEmployee.age;
      cout << "Role: ";
      cin >> myEmployee.role;
      cout << "Salary: ";
      cin >> myEmployee.salary;

      cout << "Thông tin sau khi thay đổi:" << endl;
      Display(myEmployee);
      return;
}
```

Chương trình 3.1b minh hoạ việc tạo lập và sử dụng cấu trúc Employee lồng nhau, có dùng từ khoá **typedef**.

Chương trình 3.1b

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

typedef struct {
      int day;
      int month;
      int year;
} Date;

typedef struct {
      char name[20];     // Tên nhân viên
      Date birthDay;     // Ngày sinh của nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;

/* Khai báo khuôn mẫu hàm */
void Display(Employee myEmployee);
```

```
void Display(Employee myEmployee){
      cout << "Name: " << myEmployee.name << endl;
      cout << "Birth day: " << myEmployee.birthDay.day << "/"
            << myEmployee.birthDay.month << "/"
            << myEmployee.birthDay.year << endl;
      cout << "Role: " << myEmployee.role << endl;
      cout << "Salary: " << myEmployee.salary << endl;
      return;
}

void main(){
      clrscr();
      // Hiển thị giá trị mặc định
      Employee myEmployee =
            {"Nguyen Van A", {15, 5, 1980}, "Nhan vien", 300f};
      cout << "Thông tin mặc định:" << endl;
      Display(myEmployee);

      // Thay đổi giá trị cho các thuộc tính
      cout << "Name: ";
      cin >> myEmployee.name;
      cout << "Day of birth: ";
      cin >> myEmployee.birthDay.day;
      cout << "Month of birth: ";
      cin >> myEmployee.birthDay.month;
      cout << "Year of birth: ";
      cin >> myEmployee.birthDay.year;
      cout << "Role: ";
      cin >> myEmployee.role;
      cout << "Salary: ";
      cin >> myEmployee.salary;

      cout << "Thông tin sau khi thay đổi:" << endl;
      Display(myEmployee);
      return;
}
```

3.3 CON TRỎ CẤU TRÚC VÀ MẢNG CẤU TRÚC

3.3.1 Con trỏ cấu trúc

Con trỏ cấu trúc là một con trỏ trỏ đến địa chỉ của một biến có kiểu cấu trúc. Cách khai báo và sử dụng con trỏ cấu trúc được thực hiện như con trỏ thông thường.

Khai báo con trỏ cấu trúc

Con trỏ cấu trúc được khai báo theo cú pháp:

```
<Tên kiểu cấu trúc> *<Tên biến>;
```

Ví dụ, với kiểu khai báo cấu trúc:

```
typedef struct {
      int day;
      int month;
      int year;
} Date;
```

và:

```
typedef struct {
      char name[20];    // Tên nhân viên
      Date birthDay;    // Ngày sinh của nhân viên
      char role[20];    // Chức vụ của nhân viên
      float salary;     // Lương của nhân viên
} Employee;
```

thì ta có thể khai báo một con trỏ cấu trúc như sau:

```
Employee *ptrEmployee;
```

Lưu ý:

- Cũng như khai báo con trỏ thông thường, dấu con trỏ "*" có thể nằm ngay trước tên biến hoặc nằm ngay sau tên kiểu cấu trúc.

Cũng giống con trỏ thông thường, con trỏ cấu trúc được sử dụng khi:

- Cho nó trỏ đến địa chỉ của một biến cấu trúc
- Cấp phát cho nó một vùng nhớ xác định.

Gán địa chỉ cho con trỏ cấu trúc

Một con trỏ cấu trúc có thể trỏ đến địa chỉ của một biến cấu trúc có cùng kiểu thông qua phép gán:

```
<Tên biến con trỏ> = &<Tên biến thường>;
```

Ví dụ, khai báo và phép gán:

```
Employee *ptrEmployee, myEmployee;
ptrEmployee = &myEmployee;
```

sẽ đưa con trỏ ptrEmployee trỏ đến địa chỉ của biến cấu trúc myEmployee.

Cấp phát bộ nhớ động cho con trỏ cấu trúc

Trong trường hợp ta muốn tạo ra một con trỏ cấu trúc mới, không trỏ vào một biến cấu trúc có sẵn nào, để sử dụng con trỏ mới này, ta phải cấp phát vùng nhớ cho nó. Cú pháp cấp phát vùng nhớ cho con trỏ cấu trúc:

```
<Tên biến con trỏ> = new <Kiểu cấu trúc>;
```

Ví dụ, cấu trúc Employee được khai báo bằng từ khoá **typedef**, ta có thể cấp phát vùng nhớ cho con trỏ cấu trúc như sau:

```
Employee *ptrEmployee;
ptrEmployee = new Employee;
```

hoặc cấp phát ngay khi khai báo:

```
Employee *ptrEmployee = new Employee;
```

Sau khi cấp phát vùng nhớ cho con trỏ bằng thao tác **new**, khi con trỏ không được dùng nữa, hoặc cần trỏ sang một địa chỉ khác, ta phải giải phóng vùng nhớ vừa được cấp phát cho con trỏ bằng thao tác:

```
delete <Tên biến con trỏ>;
```

Ví dụ:

```
Employee *ptrEmployee = new Employee;
…
// Thực hiện các thao tác trên con trỏ
…
delete ptrEmployee;
```

Lưu ý:

- Thao tác **delete** chỉ được thực hiện đối với con trỏ mà trước đó, nó được cấp phát bộ nhớ động thông qua thao tác **new**:

  ```
  Employee *ptrEmployee = new Employee;
  delete ptrEmployee;     //đúng
  ```

  mà không thể thực hiện với con trỏ chỉ trỏ đến địa chỉ của một biến cấu trúc khác:

  ```
  Employee *ptrEmployee, myEmployee;
  ptrEmployee = &myEmployee;
  delete ptrEmployee;     //lỗi
  ```

Truy nhập thuộc tính của con trỏ cấu trúc

Thuộc tính của con trỏ cấu trúc có thể được truy nhập thông qua hai cách:

Cách 1:

```
<Tên biến con trỏ> -> <Tên thuộc tính>;
```

Cách 2:

```
(*<Tên biến con trỏ>).<Tên thuộc tính>;
```

Ví dụ, thuộc tính tên nhân viên của cấu trúc Employee có thể được truy nhập thông qua hai cách:

```
Employee *ptrEmployee = new Employee;
cin >> ptrEmployee -> name;
```

hoặc:

```
      cin >> (*ptrEmployee).name;
```

Lưu ý:

- Trong cách truy nhập thứ hai, phải có dấu ngoặc đơn "()" quanh tên con trỏ vì phép toán truy nhập thuộc tính "." có độ ưu tiên cao hơn phép toán lấy giá trị con trỏ "*".
- Thông thường, ta dùng cách thứ nhất cho đơn giản và thuận tiện.

Chương trình 3.2 cài đặt việc khởi tạo và hiển thị nội dung của một con trỏ cấu trúc.

Chương trình 3.2

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

typedef struct {
      int day;
      int month;
      int year;
} Date;

typedef struct {
      char name[20];     // Tên nhân viên
      Date birthDay;     // Ngày sinh của nhân viên
      char role[20];     // Chức vụ của nhân viên
      float salary;      // Lương của nhân viên
} Employee;

/* Khai báo khuôn mẫu hàm */
void InitStruct(Employee *myEmployee);
void Display(Employee *myEmployee);

void InitStruct(Employee *myEmployee){
      myEmployee = new Employee;
      cout << "Name: ";
      cin >> myEmployee->name;
      cout << "Day of birth: ";
      cin >> myEmployee->birthDay.day;
      cout << "Month of birth: ";
      cin >> myEmployee->birthDay.month;
      cout << "Year of birth: ";
      cin >> myEmployee->birthDay.year;
      cout << "Role: ";
      cin >> myEmployee->role;
      cout << "Salary: ";
```

```
      cin >> myEmployee->salary;
}

void Display(Employee myEmployee){
      cout << "Name: " << myEmployee->name << endl;
      cout << "Birth day: " << myEmployee->birthDay.day << "/"
            << myEmployee->birthDay.month << "/"
            << myEmployee->birthDay.year << endl;
      cout << "Role: " << myEmployee->role << endl;
      cout << "Salary: " << myEmployee->salary << endl;
      return;
}

void main(){
      clrscr();
      Employee *myEmployee;
      InitStruct(myEmployee);
      Display(myEmployee);
      return;
}
```

3.3.2 Mảng cấu trúc

Khi cần xử lí nhiều đối tượng có dùng kiểu dữ liệu cấu trúc, ta có thể sử dụng mảng các cấu trúc. Vì một mảng một chiều là tương đương với một con trỏ có cùng kiểu. Do đó, có thể khai báo mảng theo hai cách: Khai báo mảng tĩnh như thông thường hoặc khai báo mảng động thông qua con trỏ.

Khai báo mảng tĩnh các cấu trúc

Khai báo mảng tĩnh các cấu trúc theo cú pháp:

```
<Tên kiểu cấu trúc> <Tên biến mảng>[<Số phần tử mảng>];
```

Ví dụ:

```
Employee employees[10];
```

là khai báo một mảng tên là employees gồm 10 phần tử có kiểu là cấu trúc Employee.

Khai báo mảng động các cấu trúc

Khai báo một mảng động các cấu trúc hoàn toàn tương tự khai báo một con trỏ cấu trúc cùng kiểu:

```
<Tên kiểu cấu trúc> *<Tên biến>;
```

Ví dụ, khai báo:

```
Employee *employees;
```

vừa có thể coi là khai báo một con trỏ thông thường có cấu trúc Employee, vừa có thể coi là khai báo một mảng động các cấu trúc có kiểu cấu trúc Employee.

Tuy nhiên, cách cấp phát bộ nhớ động cho mảng các cấu trúc khác với một con trỏ. Đây là cách để chương trình nhận biết ta đang dùng một con trỏ cấu trúc hay một mảng động có cấu trúc. Cú pháp cấp phát bộ nhớ cho mảng động như sau:

```
<Tên biến mảng> = new <Kiểu cấu trúc>[<Số lượng phần tử>];
```

Ví dụ, khai báo:

```
Employee *employees = new Employee[10];
```

sẽ cấp phát bộ nhớ cho một mảng động employees có 10 phần tử kiểu cấu trúc Employee.

Truy nhập đến phần tử của mảng cấu trúc

Việc truy nhập đến các phần tử của mảng cấu trúc được thực hiện như truy cập đến phần tử của mảng thông thường. Ví dụ muốn truy nhập đến thuộc tính tên nhân viên phần tử nhân viên thứ i trong mảng cấu trúc, ta viết như sau:

```
Employee *employees = new Employee[10];
employees[i].name;
```

Chương trình 3.3 cài đặt việc khởi tạo một mảng các nhân viên của một phòng trong một công ty. Sau đó, chương trình sẽ tìm và in ra thông tin về nhân viên có lương cao nhất và nhân viên có lương thấp nhất trong phòng.

Chương trình 3.3

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

typedef struct {
      int day;
      int month;
      int year;
} Date;

typedef struct {
      char name[20];      // Tên nhân viên
      Date birthDay;      // Ngày sinh của nhân viên
      char role[20];      // Chức vụ của nhân viên
      float salary;       // Lương của nhân viên
} Employee;

/* Khai báo khuôn mẫu hàm */
void InitArray(Employee *myEmployee, int length);
Employee searchSalaryMax(Employee *myEmployee, int length);
Employee searchSalaryMin(Employee *myEmployee, int length);
void Display(Employee myEmployee);
```

```
void InitArray(Employee *myEmployee, int length){
      myEmployee = new Employee[length];
      for(int i=0; i<length; i++){
            cout << "Nhan vien thu " << i << endl;
            cout << "Name: ";
            cin >> myEmployee[i].name;
            cout << "Day of birth: ";
            cin >> myEmployee[i].birthDay.day;
            cout << "Month of birth: ";
            cin >> myEmployee[i].birthDay.month;
            cout << "Year of birth: ";
            cin >> myEmployee[i].birthDay.year;
            cout << "Role: ";
            cin >> myEmployee[i].role;
            cout << "Salary: ";
            cin >> myEmployee[i].salary;
      }
      return;
}

Employee searchSalaryMax(Employee *myEmployee, int length){
      int index = 0;
      int maxSalary = myEmployee[0].salary;
      for(int i=1; i<length; i++)
            if(myEmployee[i].salary > maxSalary){
                  maxSalary = myEmployee[i].salary;
                  index = i;
            }
      return myEmployee[index];
}

Employee searchSalaryMin(Employee *myEmployee, int length){
      int index = 0;
      int minSalary = myEmployee[0].salary;
      for(int i=1; i<length; i++)
            if(myEmployee[i].salary < minSalary){
                  minSalary = myEmployee[i].salary;
                  index = i;
            }
      return myEmployee[index];
}

void Display(Employee myEmployee){
```

```
        cout << "Name: " << myEmployee.name << endl;
        cout << "Birth day: " << myEmployee.birthDay.day << "/"
             << myEmployee.birthDay.month << "/"
             << myEmployee.birthDay.year << endl;
        cout << "Role: " << myEmployee.role << endl;
        cout << "Salary: " << myEmployee.salary << endl;
        return;
}

void main(){
        clrscr();
        Employee *myEmployee, tmpEmployee;
        int length = 0;
        cout << "So luong nhan vien: ";
        cin >> length;

        // Khởi tạo danh sách nhân viên
        InitArray(myEmployee);

        // Nhân viên có lương cao nhất
        tmpEmployee = searchSalaryMax(myEmployee, length);
        Display(tmpEmployee);

        // Nhân viên có lương thấp nhất
        tmpEmployee = searchSalaryMin(myEmployee, length);
        Display(tmpEmployee);

        // Giải phóng vùng nhớ
        delete [] myEmployee;
        return;
}
```

3.4 MỘT SỐ KIỂU DỮ LIỆU TRỪU TƯỢNG

Nội dung phần này tập trung trình bày việc cài đặt một số cấu trúc dữ liệu trừu tượng, bao gồm:

- Ngăn xếp (stack)
- Hàng đợi (queue)
- Danh sách liên kết (list)

3.4.1 Ngăn xếp

Ngăn xếp (stack) là một kiểu danh sách cho phép thêm và bớt các phần tử ở một đầu danh sách, gọi là đỉnh của ngăn xếp. Ngăn xếp hoạt động theo nguyên lí: phần tử nào được đưa vào sau, sẽ được lấy ra trước.

Định nghĩa cấu trúc ngăn xếp

Vì ta chỉ cần quan tâm đến hai thuộc tính của ngăn xếp là:

- Danh sách các phần tử của ngăn xếp
- Vị trí đỉnh của ngăn xếp

nên ta có thể định nghĩa cấu trúc ngăn xếp như sau (các phần tử của ngăn xếp có kiểu int):

```
typedef SIZE 100;
typedef struct {
      int top;          // Vị trí của đỉnh
      int nodes[SIZE];  // Danh sách các phần tử
} Stack;
```

Tuy nhiên, định nghĩa này tồn tại một vấn đề, đó là kích thước (SIZE) của danh sách chứa các phần tử là tĩnh. Do đó:

- Nếu ta chọn SIZE lớn, nhưng khi gặp ứng dụng chỉ cần một số ít phần tử cho ngăn xếp thì rất tốn bộ nhớ.
- Nếu ta khai báo SIZE nhỏ, thì khi gặp bài toán cần ngăn xếp có nhiều phần tử, ta sẽ không thêm được các phần tử mới vào, chương trình sẽ có lỗi.

Để khắc phục hạn chế này, ta có thể sử dụng bộ nhớ động (mảng động thông qua con trỏ) để lưu danh sách các phần tử của ngăn xếp. Khi đó, định nghĩa cấu trúc ngăn xếp sẽ có dạng như sau:

```
typedef struct {
      int top;          // Vị trí của đỉnh
      int *nodes;       // Danh sách các phần tử
} Stack;
```

Ta sẽ sử dụng định nghĩa này trong các chương trình ứng dụng ngăn xếp.

Các thao tác trên ngăn xếp

Đối với các thao tác trên ngăn xếp, ta quan tâm đến hai thao tác cơ bản:

- Thêm một phần tử mới vào đỉnh ngăn xếp, gọi là **push**.
- Lấy ra một phần tử từ đỉnh ngăn xếp, gọi là **pop**.

Khi thêm một phần tử mới vào ngăn xếp, ta làm các bước như sau:

1. Số phần tử trong ngăn xếp cũ là (top+1). Do đó, ta cấp phát một vùng nhớ mới để lưu được (top+1+1) = (top+2) phần tử.
2. Sao chép (top+1) phần tử cũ sang vùng mới. Nếu danh sách ban đầu rỗng (top = -1) thì không cần thực hiện bước này.
3. Thêm phần tử mới vào cuối vùng nhớ mới
4. Giải phóng vùng nhớ của danh sách cũ

5. Cho danh sách nodes trỏ vào vùng nhớ mới.

Chương trình 3.4a cài đặt thủ tục thêm một phần tử mới vào ngăn xếp.

Chương trình 3.4a

```
void push(Stack *stack, int node){
      int *tmpNodes = new int[stack->top + 2];// Cấp phát vùng nhớ mới
      stack->top ++;                          // Tăng chỉ số của node đỉnh
      for(int i=0; i<stack->top; i++)     // Sao chép sang vùng nhớ mới
            tmpNodes[i] = stack->nodes[i];
      tmpNodes[stack->top] = node;          // Thêm node mới vào đỉnh
      delete [] stack->nodes;               // Giải phóng vùng nhớ cũ
      stack->nodes = tmpNodes;              // Trỏ vào vùng nhớ mới
      return;
}
```

Khi lấy ra một phần tử của ngăn xếp, ta làm các bước như sau:

- Kiểm tra xem ngăn xếp có rỗng (top = -1) hay không. Nếu không rỗng thì thực hiện các bước tiếp theo.
- Lấy phần tử ở đỉnh ngăn xếp ra
- Cấp phát một vùng nhớ mới có (top+1) -1 = top phần tử
- Sao chép top phần tử từ danh sách cũ sang vùng nhớ mới (trừ phần tử ở đỉnh).
- Giải phóng vùng nhớ cũ
- Cho con trỏ danh sách trỏ vào vùng nhớ mới.
- Trả về giá trị phần tử ở đỉnh đã lấy ra.

Chương trình 3.4b cài đặt thủ tục lấy một phần tử từ ngăn xếp.

Chương trình 3.4b

```
int pop(Stack *stack){
      if(stack->top < 0){                   // Kiểm tra ngăn xếp rỗng
            cout << "Stack is empty!" << endl;
            return 0;
      }
      int result = stack->nodes[stack->top];// Lưu giữ giá trị đỉnh
      int *tmpNodes = new int[stack->top];// Cấp phát vùng nhớ mới
      for(int i=0; i<stack->top; i++)     // Sao chép sang vùng nhớ mới
            tmpNodes[i] = stack->nodes[i];
      stack->top --;                        // Giảm chỉ số của node đỉnh
      delete [] stack->nodes;               // Giải phóng vùng nhớ cũ
      stack->nodes = tmpNodes;              // Trỏ vào vùng nhớ mới
      return result;                        // Trả về giá trị node đỉnh
```

```
}
```

Áp dụng

Ngăn xếp được sử dụng trong các ứng dụng thoả mãn nguyên tắc: cái nào đặt vào trước sẽ được lấy ra sau. Chương trình 3.4c minh hoạ việc dùng ngăn xếp để đảo ngược một xâu kí tự được nhập vào từ bàn phím.

Chương trình 3.4c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct {
      int top;                        // Vị trí node đỉnh
      int *nodes;                     // Danh sách phần tử
} Stack;

/* Khai báo nguyên mẫu hàm */
void init(Stack *stack);
void push(Stack *stack, int node);
int pop(Stack *stack);
void release(Stack *stack);

void init(Stack *stack){
      stack = new Stack;              // Cấp phát vùng nhớ cho con trỏ
      stack->top = -1;                // Khởi tạo ngăn xếp rỗng
}

void push(Stack *stack, int node){
      int *tmpNodes = new int[stack->top + 2];// Cấp phát vùng nhớ mới
      stack->top ++;                          // Tăng chỉ số của node đỉnh
      for(int i=0; i<stack->top; i++)         // Sao chép sang vùng nhớ mới
            tmpNodes[i] = stack->nodes[i];
      tmpNodes[stack->top] = node;            // Thêm node mới vào đỉnh
      delete [] stack->nodes;                 // Giải phóng vùng nhớ cũ
      stack->nodes = tmpNodes;                // Trỏ vào vùng nhớ mới
      return;
}

int pop(Stack *stack){
      if(stack->top < 0){                     // Kiểm tra ngăn xếp rỗng
```

```
            cout << "Stack is empty!" << endl;
            return 0;
      }
      int result = stack->nodes[stack->top];// Lưu giữ giá trị đỉnh
      int *tmpNodes = new int[stack->top];// Cấp phát vùng nhớ mới
      for(int i=0; i<stack->top; i++)     // Sao chép sang vùng nhớ mới
            tmpNodes[i] = stack->nodes[i];
      stack->top --;                      // Giảm chỉ số của node đỉnh
      delete [] stack->nodes;             // Giải phóng vùng nhớ cũ
      stack->nodes = tmpNodes;            // Trỏ vào vùng nhớ mới
      return result;                      // Trả về giá trị node đỉnh
}

void release(Stack *stack){
      delete [] stack->nodes;             // Giải phóng vùng danh sách
      delete stack;                       // Giải phóng con trỏ
      return;
}

void main(){
      clrscr();
      Stack *stack;
      init(stack);                        // Khởi tạo ngăn xếp
      char strIn[250];
      // Nhập chuỗi kí tự từ bàn phím
      cout << "Nhap chuoi: ";
      cin >> strIn;
      for(int i=0; i<strlen(strIn); i++)  // Đặt vào ngăn xếp
            push(stack, strIn[i]);
      while(stack->top > -1)              // Lấy ra từ ngăn xếp
            cout << pop(stack);
      release(stack);                     // Giải phóng bộ nhớ
      return;
}
```

3.4.2 Hàng đợi

Hàng đợi (queue) cũng là một cấu trúc tuyến tính các phần tử. Trong đó, các phần tử luôn được thêm vào ở một đầu, gọi là đầu cuối hàng đợi, và việc lấy ra các phần tử luôn được thực hiện ở đầu còn lại, gọi là đầu mặt của hàng đợi. Hàng đợi hoạt động theo nguyên lí: phần tử nào được đưa vào trước, sẽ được lấy ra trước.

Định nghĩa cấu trúc hàng đợi

Hàng đợi có các thuộc tính:

- Một danh sách các phần tử có mặt trong hàng đợi.
- Chỉ số của phần tử đứng đầu của danh sách (front).
- Chỉ số phần tử cuối của danh sách (rear).

Nếu dùng cấu trúc tĩnh để định nghĩa, hàng đợi có cấu trúc như sau:

```
typedef SIZE 100;
typedef struct {
      int front, rear;  // Vị trí của đỉnh đầu, đỉnh cuối
      int nodes[SIZE];  // Danh sách các phần tử
} Queue;
```

Nếu dùng bộ nhớ động để lưu giữ hàng đợi, thì phần tử front luôn là phần tử thứ 0 của danh sách. Và rear sẽ bằng độ dài danh sách trừ đi 1. Cấu trúc động của hàng đợi:

```
typedef struct {
      int front, rear;  // Vị trí của đỉnh đầu, đỉnh cuối
      int *nodes;       // Danh sách các phần tử
} Queue;
```

Thao tác trên hàng đợi

- Thêm một phần tử vào cuối hàng đợi
- Lấy một phần tử ở vị trí đầu của hàng đợi

Thao tác thêm một phần tử vào cuối hàng đợi với bộ nhớ động được thực hiện tương tự với ngăn xếp. Chương trình 3.5a cài đặt thủ tục thêm một phần tử vào cuối hàng đợi động.

Chương trình 3.5a

```
void insert(Queue *queue, int node){
      int *tmpNodes = new int[queue->rear + 2];// Cấp phát vùng nhớ mới
      queue->rear ++;                          // Tăng chỉ số của node đuôi
      if(queue->front == -1)                   // Nếu hàng đợi cũ rỗng
            queue->front = 0;                  // thì cập nhật front
      for(int i=0; i<queue->rear; i++)         // Sao chép sang vùng nhớ mới
            tmpNodes[i] = queue->nodes[i];
      tmpNodes[queue->rear] = node;            // Thêm node mới vào đuôi
      delete [] queue->nodes;                  // Giải phóng vùng nhớ cũ
      queue->nodes = tmpNodes;                 // Trỏ vào vùng nhớ mới
      return;
}
```

Thao tác lấy ra một phần tử đầu của hàng đợi thực hiện theo các bước:

1. Kiểm tra tính rỗng (front = rear = -1) của hàng đợi. Nếu không rỗng mới thực hiện tiếp

2. Lấy phần tử nodes[0] ra.
3. Sao chép danh sách còn lại sang vùng nhớ mới
4. Giải phóng vùng nhớ cũ
5. Đưa danh sách trỏ vào vùng nhớ mới
6. Trả về giá trị phần tử lấy ra

Chương trình 3.5b cài đặt thủ tục lấy ra một phần tử của hàng đợi động.

Chương trình 3.5b

```
int remove(Queue *queue){
      if((queue-front < 0)||(queue-rear < 0)){// Kiểm tra hàng đợi rỗng
            cout << "Queue is empty!" << endl;
            return 0;
      }
      // Lưu giữ giá trị phần tử đầu
      int result = queue->nodes[queue->front];
      int *tmpNodes;
      if(queue->rear > 0){                  // Nếu có hơn 1 phần tử
            tmpNodes = new int[queue->rear];// Cấp phát vùng nhớ mới
            for(int i=0; i<queue->rear; i++)// Sao chép sang vùng nhớ mới
                  tmpNodes[i] = queue->nodes[i];
      }else                                 // Nếu chỉ có 1 phần tử
            queue->front --;                // Hàng đợi thành rỗng
      queue->rear --;                       // Giảm chỉ số của node đuôi
      delete [] queue->nodes;               // Giải phóng vùng nhớ cũ
      queue->nodes = tmpNodes;              // Trỏ vào vùng nhớ mới
      return result;                        // Trả về giá trị node đầu
}
```

Áp dụng

Hàng đợi được áp dụng trong các bài toán cần cơ chế quản lí cái nào vào trước sẽ được lấy ra trước. Chương trình 3.5c minh hoạ cơ chế quản lí tiến trình đơn giản nhất của hệ điều hành: các tiến trình được quản lí theo mã tiến trình, khi xuất hiện, tiến trình được đưa vào cuối của một hàng đợi. Khi nào CPU rảnh thì sẽ lấy tiến trình đầu hàng đợi ra để thực hiện.

Chương trình 3.5c

```
#include<stdio.h>
#include<conio.h>

typedef struct {
      int front, rear;              // Vị trí của đỉnh đầu, đỉnh cuối
```

```
      int *nodes;                    // Danh sách các phần tử
} Queue;

/* Khai báo các nguyên mẫu hàm */
void init(Queue *queue);
void insert(Queue *queue, int node);
int remove(Queue *queue);
void travese(Queue *queue);
void release(Queue *queue);

void init(Queue *queue){
      queue = new Queue;             // Cấp phát bộ nhớ cho con trỏ
      queue->front = -1;             // Khởi tạo danh sách rỗng
      queue->rear = -1;
      return;
}

void insert(Queue *queue, int node){
      int *tmpNodes = new int[queue->rear + 2];// Cấp phát vùng nhớ mới
      queue->rear ++;                        // Tăng chỉ số của node đuôi
      if(queue->front == -1)                 // Nếu hàng đợi cũ rỗng
            queue->front = 0;                // thì cập nhật front
      for(int i=0; i<queue->rear; i++)       // Sao chép sang vùng nhớ mới
            tmpNodes[i] = queue->nodes[i];
      tmpNodes[queue->rear] = node;          // Thêm node mới vào đuôi
      delete [] queue->nodes;                // Giải phóng vùng nhớ cũ
      queue->nodes = tmpNodes;               // Trỏ vào vùng nhớ mới
      return;
}

int remove(Queue *queue){
      if((queue-front < 0)||(queue-rear < 0)){// Kiểm tra hàng đợi rỗng
            cout << "Queue is empty!" << endl;
            return 0;
      }
      // Lưu giữ giá trị phần tử đầu
      int result = queue->nodes[queue->front];
      int *tmpNodes;
      if(queue->rear > 0){                   // Nếu có hơn 1 phần tử
            tmpNodes = new int[queue->rear];// Cấp phát vùng nhớ mới
            for(int i=0; i<queue->rear; i++)// Sao chép sang vùng nhớ mới
                  tmpNodes[i] = queue->nodes[i];
      }else                                  // Nếu chỉ có 1 phần tử
```

```
            queue->front --;                // Hàng đợi thành rỗng
      queue->rear --;                       // Giảm chỉ số của node đuôi
      delete [] queue->nodes;               // Giải phóng vùng nhớ cũ
      queue->nodes = tmpNodes;              // Trỏ vào vùng nhớ mới
      return result;                        // Trả về giá trị node đầu
}

void travese(Queue *queue){
      if(queue->front < 0){                 // Khi danh sách rỗng
            cout << "Danh sach rong!" << endl;
            return;
      }
      for(int i=queue->front; i<=queue.rear; i++)
            cout << queue->nodes[i] << " ";// Liệt kê các phần tử
      cout << endl;
      return;
}

void release(Queue *queue){
      if(queue->front > -1)          //Nếu danh sách không rỗng thì
            delete [] queue->nodes; //giải phóng vùng nhớ của danh sách
      delete queue;                 //Giải phóng vùng nhớ của con trỏ
}

void main(){
      clrscr();
      Queue *queue;
      init(queue);                          // Khởi tạo hàng đợi
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Them mot tien trinh vao hang doi" << endl;
            cout << "2: Dua mot tien trinh trinh vao thuc hien" << endl;
            cout<<"3: Xem tat ca cac tien trinh trong hang doi" << endl;
            cout << "5: Thoat!" << endl;
            cout << "========================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':             // Thêm vào hàng đợi
                        int maso;
                        cout << "Ma so tien trinh vao hang doi: ";
```

```
                        cin >> maso;
                        insert(queue, maso);
                        break;
                  case '2':               // Lấy ra khỏi hàng đợi
                        cout << "Tien trinh duoc thuc hien: " <<
                              remove(queue) << endl;
                        break;
                  case '3':               // Duyệt hàng đợi
                        cout<<"Cac tien trinh dang o trong hang doi la:"
                              <<endl;
                        travese(queue);
                        break;
            }while(function != '5');
      release(queue);                     // Giải phóng hàng đợi
      return;
}
```

3.4.3 Danh sách liên kết

Danh sách liên kết là một kiểu dữ liệu bao gồm một dãy các phần tử có thứ tự, các phần tử có cùng cấu trúc dữ liệu, ngoại trừ node đầu tiên của danh sách là node lưu thông tin về danh sách. Có hai loại danh sách liên kết:

- Danh sách liên kết đơn: mỗi node có một con trỏ trỏ đến node tiếp theo trong danh sách
- Danh sách liên kết kép: mỗi node có hai con trỏ, một trỏ vào node trước, một trỏ vào node tiếp theo trong danh sách.

Trong phần này sẽ trình bày danh sách liên kết đơn. Danh sách liên kết kép được coi như là một bài tập mở rộng từ danh sách liên kết đơn.

Định nghĩa danh sách đơn

Mỗi node của danh sách đơn chứa dữ liệu của nó, đồng thời trỏ đến node tiếp theo trong danh sách, cấu trúc một node như sau:

```
struct simple{
      Employee employee;      // Dữ liệu của node có kiểu Employee
      struct simple *next;    // Trỏ đến node kế tiếp
};
typedef struct simple SimpleNode;
```

Node đầu của danh sách đơn có cấu trúc riêng, nó không chứa dữ liệu như node thường mà chứa các thông tin:

- Số lượng node trong danh sách (không kể bản thân nó – node đầu)
- Con trỏ đến node đầu tiên của danh sách
- Con trỏ đến node cuối cùng của danh sách

Do vậy, cấu trúc node đầu của danh sách đơn là:

```
typedef struct{
      int nodeNumber;          // Số lượng các node
      SimpleNode *front, *rear;// Trỏ đến node đầu và cuối danh sách
} SimpleHeader;
```

Các thao tác trên danh sách liên kết đơn

Các thao tác cơ bản trên danh sách đơn bao gồm:

- Chèn thêm một node vào vị trí thứ n trong danh sách
- Loại ra một node ở vị trí thứ n trong danh sách

Việc chèn thêm một node vào vị trí thứ n trong danh sách được thực hiện theo các bước:

1. Nếu n<=0, chèn vào đầu. Nếu n>số phần tử của danh sách, chèn vào cuối. Trường hợp còn lại, chèn vào giữa.
2. Tìm node thứ n: giữ vết của hai node thứ n-1 và thứ n.
3. Tạo một node mới: cho node thứ n-1 trỏ tiếp vào node mới và node mới trỏ tiếp vào node thứ n.

Chương trình 3.6a cài đặt thủ tục chèn một node vào vị trí thứ n của danh sách.

Chương trình 3.6a

```
void insert(SimpleHeader *list, int position, int value){
      SimpleNode *newNode = new SimpleNode;
      newNode->value = value;
      if(position <= 0){                  // Chèn vào đầu ds
            newNode->next = list->front;  // Chèn vào trước node đầu
            list->front = newNode;        // Cập nhật lại node đầu ds
            if(list->nodeNumber == 0)     // Nếu ds ban đầu rỗng thì
                  list->rear = newNode;   // node đuôi trùng với node đầu
      }else if(position >= list->nodeNumber){// Chèn vào cuối ds
            list->rear->next = newNode;   // Chèn vào sau node cuối
            list->rear = newNode;         // Cập nhật lại node cuối ds
            if(list->nodeNumber == 0)     // Nếu ds ban đầu rỗng thì
                  list->front = newNode;  // node đầu trùng node đuôi
      }else{                              // Chèn vào giữa ds
            SimpleNode *prev = list->front, *curr = list->front;
            int index = 0;
            while(index < position){      // tìm node n-1 và n
                  prev = curr;
                  curr = curr->next;
                  index++;
            }
            newNode->next = curr;         // chèn vào trước node n
            prev->next = newNode;         // và chèn vào sau node n-1
      }
```

```
      list->nodeNumber++;                    // Cập nhật số lượng node
      return;
}
```

Việc xoá một node ở vị trí thứ n trong danh sách được thực hiện theo các bước:

1. Nếu n<0 hoặc n>số phần tử của danh sách, không xoá node nào.
2. Tìm node thứ n: giữ vết của ba node thứ n-1, thứ n và thứ n+1.
3. Cho node thứ n-1 trỏ tiếp vào node thứ n+1, xoá con trỏ của node thứ n.
4. Trả về node thứ n.

Chương trình 3.6b cài đặt thủ tục xoá một node ở vị trí thứ n của danh sách.

Chương trình 3.6b

```
SimpleNode* remove(SimpleHeader *list, int position){
      if((position < 0)||(position >= list->nodeNumber))
            return NULL;                     // Không xoá node nào cả
      SimpleNode* result;
      if(position == 0){                     // Xoá node đầu
            result = list->front;            // Giữ node cần xoá
            list->front = list->front->next;// Cập nhật node đầu
            if(list->nodeNumber == 1)        // Nếu ds chỉ có 1 node thì
                  list->rear = list->front;// Cập nhật node cuối ds
            }else if(position == list->nodeNumber - 1){
            result = list->rear;             // Giữ node cần xoá
            SimpleNode *curr = list->front;
            while(curr->next != list->rear)
                  curr = curr->next;         // Tìm node trước của node cuối
            curr->next = NULL;               // Xoá node rear hiện tại
            list->rear = curr;               // Cập nhật node cuối ds
      }else{
            SimpleNode *prev = list->front, *curr = list->front;
            int index = 0;
            while(index < position){         // Tìm node n-1 và n
                  prev = curr;
                  curr = curr->next;
                  index++;
            }
            result = curr;                   // Giữ node cần xoá
            prev->next = curr->next;     // Cho node n-1 trỏ đến node n+1
      }
      list->nodeNumber --;                   // Cập nhật số lượng node
      return result;                         // Trả về node cần xoá
```

```
}
```

Áp dụng

Chương trình 3.6c minh hoạ việc dùng danh sách liên kết đơn để quản lí nhân viên văn phòng với các thông tin rất đơn giản: tên, tuổi và tiền lương của mỗi nhân viên.

Chương trình 3.6c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

typedef struct{
      char name[25];                // Tên nhân viên
      int age;                      // Tuổi nhân viên
      float salary;                 // Lương nhân viên
} Employee;

struct simple{
      Employee employee;                  // Dữ liệu của node
      struct simple *next;          // Trỏ đến node kế tiếp
};
typedef struct simple SimpleNode;

typedef struct{
      int nodeNumber;               // Số lượng các node
      SimpleNode *front, *rear;     // Trỏ đến node đầu và cuối ds
} SimpleHeader;



/* Khai báo các nguyên mẫu hàm */
void init(SimpleHeader *list);
void insert(SimpleHeader *list, int position, Employee employee);
SimpleNode* remove(SimpleHeader *list);
void travese(SimpleHeader *list);
void release(SimpleHeader *list);

void init(SimpleHeader *list){
      list = new list;                    // Cấp phát bộ nhớ cho con trỏ
      list->front = NULL;                 // Khởi tạo danh sách rỗng
      list->rear = NULL;
```

```
	list->nodeNumber = 0;
	return;
}

void insert(SimpleHeader *list, int position, Employee employee){
	SimpleNode *newNode = new SimpleNode;
	newNode->employee = employee;
	if(position <= 0){                    // Chèn vào đầu ds
		newNode->next = list->front;  // Chèn vào trước node đầu
		list->front = newNode;        // Cập nhật lại node đầu ds
		if(list->nodeNumber == 0)     // Nếu ds ban đầu rỗng thì
			list->rear = newNode;   // node đuôi trùng với node đầu
	}else if(position >= list->nodeNumber){// Chèn vào cuối ds
		list->rear->next = newNode;   // Chèn vào sau node cuối
		list->rear = newNode;         // Cập nhật lại node cuối ds
		if(list->nodeNumber == 0)// Nếu ds ban đầu rỗng thì
			list->front = newNode;  // node đầu trùng node đuôi
	}else{                          // Chèn vào giữa ds
		SimpleNode *prev = list->front, *curr = list->front;
		int index = 0;
		while(index < position){// tìm node n-1 và n
			prev = curr;
			curr = curr->next;
			index++;
		}
		newNode->next = curr;   // chèn vào trước node n
		prev->next = newNode;   // và chèn vào sau node n-1
	}
	list->nodeNumber++;             // Cập nhật số lượng node
	return;
}

SimpleNode* remove(SimpleHeader *list, int position){
	if((position < 0)||(position >= list->nodeNumber))
		return NULL;            // Không xoá node nào cả
	SimpleNode* result;
	if(position == 0){              // Xoá node đầu
		result = list->front;   // Giữ node cần xoá
		list->front = list->front->next;// Cập nhật node đầu
		if(list->nodeNumber == 1)     // Nếu ds chỉ có 1 node thì
			list->rear = list->front;// Cập nhật node cuối ds
		}else if(position == list->nodeNumber - 1){
		result = list->rear;    // Giữ node cần xoá
```

```
            SimpleNode *curr = list->front;
            while(curr->next != list->rear)
                  curr = curr->next;// Tìm node trước của node cuối
            curr->next = NULL;      // Xoá node rear hiện tại
            list->rear = curr;      // Cập nhật node cuối ds
      }else{
            SimpleNode *prev = list->front, *curr = list->front;
            int index = 0;
            while(index < position){// Tìm node n-1 và n
                  prev = curr;
                  curr = curr->next;
                  index++;
            }
            result = curr;          // Giữ node cần xoá
            prev->next = curr->next;// Cho node n-1 trỏ đến node n+1
      }
      list->nodeNumber --;          // Cập nhật số lượng node
      return result;                // Trả về node cần xoá
}

void travese(SimpleHeader *list){
      if(list->nodeNumber <= 0){    // Khi danh sách rỗng
            cout << "Danh sach rong!" << endl;
            return;
      }
      SimpleNode *curr = list->front;
      while(curr != NULL){
            cout << curr->employee.name << " "
                  << curr->employee.age << " "
                  << curr->employee.salary << endl;// Liệt kê các phần tử
            curr = curr->next;
      }
      return;
}

void release(SimpleHeader *list){
      SimpleNode* curr = remove(list, 0);
      while(curr != NULL){
            delete curr;            //Giải phóng vùng nhớ của node
            curr = remove(list, 0);
      }
      delete list;                  //Giải phóng vùng nhớ của con trỏ
}
```

```
void main(){
      clrscr();
      SimpleHeader *list;
      init(list);                    // Khởi tạo ds
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Them mot nhan vien" << endl;
            cout << "2: Xoa mot nhan vien" << endl;
            cout << "3: Xem tat ca cac nhan vien trong phong" << endl;
            cout << "5: Thoat!" << endl;
            cout << "=====================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':          // Thêm vào ds
                        int position;
                        Employee employee;
                        cout << "Vi tri can chen: ";
                        cin >> position;
                        cout << "Ten nhan vien: ";
                        cin >> employee.name;
                        cout << "Tuoi nhan vien: ";
                        cin >> employee.age;
                        cout << "Luong nhan vien: ";
                        cin >> employee.salary;
                        insert(list, position, employee);
                        break;
                  case '2':          // Lấy ra khỏi ds
                        int position;
                        cout << "Vi tri can xoa: ";
                        cin >> position;
                        SimpleNode* result = remove(list, position);
                        if(result != NULL){
                              cout << "Nhan vien bi loai: " << endl;
                              cout << "Ten: " << result->employee.name
                                    << endl;
                              cout << "Tuoi: " << result->employee.age
                                    << endl;
                              cout << "Luong: "
                                    << result->employee.salary << endl;
```

```
                        }
                        break;
                case '3':          // Duyệt ds
                        cout<<"Cac nhan vien cua phong:"<<endl;
                        travese(list);
                        break;
            }while(function != '5');
      release(list);                // Giải phóng ds
      return;
}
```

TỔNG KẾT CHƯƠNG 3

Nội dung chương 3 đã trình bày các vấn đề liên quan đến các kiểu dữ liệu có cấu trúc trong C++:

- Khai báo cấu trúc thông qua từ khoá **struct**
- Tự định nghĩa kiểu dữ liệu cấu trúc bằng từ khoá **typedef**.
- Khai báo một biến có kiểu dữ liệu cấu trúc
- Khai báo các cấu trúc lồng nhau.
- Truy nhập đến các thuộc tính của cấu trúc
- Khai báo con trỏ cấu trúc, cấp phát và giải phóng bộ nhớ động của con trỏ cấu trúc. Truy nhập đến các thuộc tính của con trỏ cấu trúc.
- Khai báo và sử dụng mảng cấu trúc
- Khai báo mảng cấu trúc bằng con trỏ cấu trúc. Cấp phát và giải phóng vùng nhớ của mảng động các cấu trúc.
- Cài đặt một số cấu trúc đặc biệt:
  - Ngăn xếp
  - Hàng đợi
  - Danh sách liên kết

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

1. Để định nghĩa một cấu trúc sinh viên có tên là Sinhvien, gồm có tên và tuổi sinh viên. Định nghĩa nào sau đây là đúng:

   a.
   ```
   struct Sinhvien{
         char name[20];
         int age;
   };
   ```

   b.
   ```
   struct {
         char name[20];
         int age;
   } Sinh vien;
   ```

c.
```
typedef struct Sinhvien{
        char name[20];
        int age;
};
```

2. Một cấu trúc được định nghĩa như sau:

```
struct Employee{
        char name[20];
        int age;
};
```

Khi đó, cách khai báo biến nào sau đây là đúng:

a. `struct Employee myEmployee;`

b. `struct employee myEmployee;`

c. `Employee myEmployee;`

d. `employee myEmployee;`

3. Một cấu trúc được định nghĩa như sau:

```
typedef struct employee{
        char name[20];
        int age;
} Employee;
```

Khi đó, cách khai báo biến nào sau đây là đúng:

a. `Employee myEmployee;`

b. `employee myEmployee;`

c. `struct Employee myEmployee;`

d. `struct employee myEmployee;`

4. Với cấu trúc được định nghĩa như trong bài 3. Khi đó, cách khởi tạo biến nào sau đây là đúng:

a. `Employee myEmployee = {'A', 27};`

b. `Employee myEmployee = {"A", 27};`

c. `Employee myEmployee = ('A', 27);`

d. `Employee myEmployee = ("A", 27);`

5. Với cấu trúc được định nghĩa như trong bài 3. Khi đó, các cách cấp phát bộ nhớ cho biến con trỏ nào sau đây là đúng:

a. `Employee *myEmployee = new Employee;`

b. `Employee *myEmployee = new Employee();`

c. `Employee *myEmployee = new Employee(10);`

d. `Employee *myEmployee = new Employee[10];`

6. Định nghĩa một cấu trúc về môn học của một học sinh có tên Subject, bao gồm các thông tin:

- Tên môn học, kiểu char[];
- Điểm tổng kết môn học, kiểu float;

7. Định nghĩa cấu trúc về học sinh tên là Student bao gồm các thông tin sau:
   - Tên học sinh, kiểu char[];
   - Tuổi học sinh, kiểu int;
   - Lớp học sinh, kiểu char[];
   - Danh sách điểm các môn học của học sinh, kiểu là một mảng các cấu trúc Subject đã được định nghĩa trong bài tập 6.
   - Xếp loại học lực, kiểu char[];
8. Khai báo một biến có cấu trúc là Student đã định nghĩa trong bài 7. Sau đó, thực hiện tính điểm trung bình của tất cả các môn học của học sinh đó, và viết một thủ tục xếp loại học sinh dựa vào điểm trung bình các môn học:
   - Nếu điểm tb nhỏ hơn 5.0, xếp loại kém
   - Nếu điểm tb từ 5.0 đến dưới 6.5, xếp loại trung bình.
   - Nếu điểm tb từ 6.5 đến dưới 8.0, xếp loại khá
   - Nếu điểm tb từ 8.0 trở lên, xếp loại giỏi.
9. Viết một chương trình quản lí các học sinh của một lớp, là một dãy các cấu trúc có kiểu Stupid định nghĩa trong bài 7. Sử dụng thủ tục đã cài đặt trong bài 8 để thực hiện các thao tác sau:
   - Khởi tạo danh sách và điểm của các học sinh trong lớp.
   - Tính điểm trung bình và xếp loại cho tất cả các học sinh.
   - Tìm tất cả các học sinh theo một loại nhất định
10. Sử dụng cấu trúc ngăn xếp đã định nghĩa trong bài để đổi một số từ kiểu thập phân sang kiểu nhị phân: Chi số nguyên cho 2, mãi cho đến khi thương <2, lưu các số dư váo ngăn xếp. Sau đó, đọc các giá trị dư từ ngăn xếp ra, ta sẽ thu được chuỗi nhị phân tương ứng.
11. Mở rộng cấu trúc hàng đợi đã định nghĩa trong bài để trở thành hàng đợi có độ ưu tiên:
    - Cho mỗi node thêm một thuộc tính là độ ưu tiên của node đó
    - Khi thêm một node vào hàng đợi, thay vì thêm vào cuối hàng đợi như thông thường, ta tìm vị trí có độ ưu tiên phù hợp để chèn node vào, sao cho dãy các node trong hàng đợi là một danh sách có độ ưu tiên của các node là giảm dần.
    - Việc lấy ra là không thay đổi: lấy ra phần tử ở đầu hàng đợi, chính là phần tử có độ ưu tiên cao nhất.
12. Áp dụng hàng đợi có độ ưu tiên trong bài 11 để xây dựng chương trình quản lí tiến trình có độ ưu tiên của hệ điều hành, mở rộng ứng dụng trong bài ngăn xếp.
13. Mở rộng cấu trúc danh sách liên kết đơn trong bài thành danh sách liên kết kép:
    - Mỗi node có thêm một con trỏ prev để trỏ đến node trước nó
    - Đối với node header, cũng cần 2 con trỏ: trỏ đến node đầu tiên và node cuối cùng của danh sách

- Riêng với node đầu tiên (front) của danh sách, con trỏ prev của nó sẽ trỏ đến NULL. Giống như con trỏ next của node rear.

14. Cài đặt lại hai tháo tác thêm vào một node và xoá một node ở một vị trí xác định trong một cấu trúc danh sách liên kết kép định nghĩa trong bài 13.
15. Áp dụng các định nghĩa và thao tác trong các bài 13 và 14. Cài đặt lại chương trình quản lí nhân viên ở chương trình 3.6c bằng danh sách liên kết kép.

CHƯƠNG 4

VÀO RA TRÊN TỆP

Nội dung chương này tập trung trình bày các vấn đề liên quan đến các thao tác trên tệp dữ liệu trong ngôn ngữ C++:

- Khái niệm tệp, tệp văn bản và tệp nhị phân
- Các thao tác vào ra trên tệp
- Phương thức truy nhập tệp trực tiếp

4.1 KHÁI NIỆM TỆP

4.1.1 Tệp dữ liệu

Trong C++, khi thao tác với một tệp dữ liệu, cần thực hiện tuần tự theo các bước như sau:

1. Mở tệp tin
2. Thực hiện các thao tác đọc, ghi trên tệp tin đang mở
3. Đóng tệp tin

Để thực hiện các thao tác liên quan đến tệp dữ liệu, C++ cung cấp một thư viện `<fstream.h>` chứa các lớp và các hàm phục vụ cho các thao tác này. Do vậy, trong các chương trình làm việc với tệp tin, ta cần khai báo chỉ thị dùng thư viện này ngay từ đầu chương trình:

```
#include<fstream.h>
```

Khai báo biến tệp

Trong C++, khi khai báo một biến tệp, đồng thời ta sẽ mở tệp tương ứng theo cú pháp tổng quát bằng cách dùng kiểu **fstream** như sau:

```
fstream <Tên biến tệp>(<Tên tệp>, <Chế độ mở tệp>);
```

Trong đó:

- **Tên biến tệp**: có tính chất như một tên biến thông thường, nó sẽ được dùng để thực hiện các thao tác với tệp gắn với nó. Tên biến tệp cũng phải tuân thủ theo quy tắc đặt tên biến trong C++.
- **Tên tệp**: là tên tệp dữ liệu mà ta cần thao tác trên nó.
- **Chế độ mở tệp**: là các hằng kiểu bít đã được định nghĩa sẵn bởi C++. Nó chỉ ra rằng ta đang mở tệp tin ở chế độ nào: đọc hoặc ghi, hoặc cả đọc lẫn ghi.

Ví dụ, khai báo:

```
fstream myFile("abc.txt", ios::in);
```

là khai báo một biến tệp, có tên là myFile, dùng để mở tệp tin có tên là abc.txt và tệp tin này được mở ở chế độ để đọc dữ liệu (bít chỉ thị ios::in).

Lưu ý:

- Tên tệp tin có dạng một chuỗi kí tự, nếu khai báo tên tệp có đường dẫn thư mục "\" thì mỗi dấu "\" phải được viết thành "\\" để tránh bị nhầm lẫn với các kí tự đặc biệt trong C như "\n", "\d"…

Ví dụ, muốn mở một tệp tên là abc.txt trong thư mục myDir để đọc, ta phải khai báo như sau:

```
fstream myFile("myDir\\abc.txt", ios::in);
```

Các chế độ mở tệp tin

Các chế độ mở tệp tin được định nghĩa bởi các bít chỉ thị:

- **ios::in**: Mở một tệp tin để đọc.
- **ios::out**: Mở một tệp tin có sẵn để ghi.
- **ios::app**: Mở một tệp tin có sẵn để thêm dữ liệu vào cuối tệp.
- **ios::ate**: Mở tệp tin và đặt con trỏ tệp tin vào cuối tệp.
- **ios::trunc**: Nếu tệp tin đã có sẵn thì dữ liệu của nó sẽ bị mất.
- **ios::nocreate**: Mở một tệp tin, tệp tin này bắt buộc phải tồn tại.
- **ios::noreplace**: Chỉ mở tệp tin khi tệp tin chưa tồn tại.
- **ios::binary**: Mở một tệp tin ở chế độ nhị phân.
- **ios::text**: Mở một tệp tin ở chế độ văn bản.

Lưu ý:

- Khi muốn mở một tệp tin đồng thời ở nhiều chế độ khác nhau, ta kết hợp các bít chỉ thị tương ứng bằng phép toán hợp bít "|".

Ví dụ, muốn mở một tệp tin abc.txt để đọc (ios::in) đồng thời với để ghi (ios::out) dưới chế độ văn bản (ios::text), ta khai báo như sau:

```
fstream myFile("abc.txt", ios::in|ios::out|ios::text);
```

4.1.2 Tệp văn bản

Để mở một tệp tin dưới chế độ văn bản, ta dùng cú pháp sau:

```
fstream <Tên biến tệp>(<Tên tệp>, ios::text);
```

Khi đó, các thao tác đọc, ghi trên biến tệp được thực hiện theo đơn vị là các từ, được phân cách bởi dấu trống (space bar) hoặc dấu xuống dòng (enter).

Ví dụ, muốn mở tệp tin baitho.txt dưới chế độ văn bản, ta khai báo như sau:

```
fstream myBaiTho("baitho.txt", ios::text);
```

4.1.3 Tệp nhị phân

Để mở một tệp tin dưới chế độ nhị phân, ta dùng cú pháp sau:

```
fstream <Tên biến tệp>(<Tên tệp>, ios::binary);
```

Khi đó, các thao tác đọc, ghi trên biến tệp được thực hiện theo đơn vị byte theo kích thước các bản ghi (cấu trúc) được ghi trong tệp.

Ví dụ, muốn mở tệp tin baitho.txt dưới chế độ nhị phân, ta khai báo như sau:

```
fstream myBaiTho("baitho.txt", ios::binary);
```

4.2 VÀO RA TRÊN TỆP

4.2.1 Vào ra tệp văn bản bằng ">>" và "<<"

Ghi tệp văn bản bằng "<<"

Các bước thực hiện để ghi dữ liệu vào một tệp tin như sau:

1. Mở tệp tin theo chế độ để ghi bằng đối tượng ofstream (mở tệp tin chỉ để ghi):
   ```
   ofstream <Tên biến tệp>(<Tên tệp tin>, ios::out);
   ```
2. Ghi dữ liệu vào tệp bằng thao tác "<<":
   ```
   <Tên biến tệp> << <Dữ liệu>;
   ```
3. Đóng tệp tin bằng lệnh close():
   ```
   <Tên biến tệp>.close();
   ```

Chương trình 4.1 minh hoạ việc ghi dữ liệu vào tệp tin:

- Tên tệp tin được người dùng tự nhập vào từ bàn phím.
- Chương trình sẽ ghi vào tệp các kí tự do người dùng gõ vào từ bàn phím, mỗi kí tự được phân cách nhau bởi dấu trống (space bar).
- Chương trình dừng lại khi người dùng nhập kí tự 'e'. Và tệp tin được kết thúc bằng một dấu xuống dòng "endl".

Chương trình 4.1

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                    // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char fileName[length], input;
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;    // Nhập tên tệp tin

      /* Mở tệp tin */
      ofstream fileOut(fileName, ios::out);// Khai báo và mở tệp tin
      if(!fileOut){                        // Không mở được tệp
            cout << "Khong the tao duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Ghi dữ liệu vào tệp tin */
```

```
      do{
            cin >> input;                  // Đọc kí tự từ bàn phím
            fileOut << input << ' ';       // Ghi kí tự vào tệp tin
      }while((input != 'e')&&(fileOut));
      fileOut << endl;                     // Xuống dòng cuối tệp tin

      /* Đóng tệp tin */
      fileOut.close();                     // Đóng tệp tin
      return;
}
```

Đọc dữ liệu từ tệp văn bản bằng ">>"

Các bước thực hiện để đọc dữ liệu từ một tệp tin như sau:

1. Mở tệp tin theo chế độ để đọc bằng đối tượng ifstream (mở tệp tin chỉ để đọc):
   ```
   ifstream <Tên biến tệp>(<Tên tệp tin>, ios::in);
   ```
2. Đọc dữ liệu từ tệp bằng thao tác "**>>**":
   ```
   <Tên biến tệp> >> <Biến dữ liệu>;
   ```
3. Đóng tệp tin bằng lệnh close():
   ```
   <Tên biến tệp>.close();
   ```

Chương trình 4.2 minh hoạ việc đọc dữ liệu từ tệp tin vừa sử dụng trong chương trình 4.1 ra màn hình:

- Tên tệp tin được người dùng tự nhập vào từ bàn phím.
- Chương trình sẽ đọc các kí tự trong tệp và hiển thị ra màn hình, mỗi kí tự được phân cách nhau bởi dấu trống (space bar).
- Chương trình dừng lại khi kết thúc tệp tin.

Chương trình 4.2

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                 // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char fileName[length], output;
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;    // Nhập tên tệp tin

      /* Mở tệp tin */
```

```
      ifstream fileIn(fileName, ios::in); // Khai báo và mở tệp tin
      if(!fileIn){                        // Không mở được tệp
            cout << "Khong the mo duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Đọc dữ liệu từ tệp tin ra màn hình */
      while(fileIn){
            fileIn >> output;             // Đọc kí tự từ tệp tin
            cout << output;               // Ghi kí tự ra màn hình
      }
      cout << endl;                       // Xuống dòng trên màn hình

      /* Đóng tệp tin */
      fileIn.close();                     // Đóng tệp tin
      return;
}
```

Chương trình 4.3 minh hoạ việc copy toàn bộ nội dung của một tệp tin sang một tệp tin mới:

- Tên tệp tin nguồn và tệp tin đích được nhập từ bàn phím bởi người dùng.
- Tệp tin nguồn được mở ở chế độ đọc.
- Tệp tin đích được mở ở chế độ ghi.
- Đọc từng kí tự từ tệp tin nguồn và ghi ngay vào tệp tin đích.
- Đóng các tệp tin khi kết thúc.

Chương trình 4.3

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char sourceFile[length], targetFile[length], data;
      cout << "Ten tep tin nguon: ";
      cin >> setw(length) >> sourceFile;  // Nhập tên tệp tin nguồn

      cout << "Ten tep tin dich: ";
      cin >> setw(length) >> targetFile;  // Nhập tên tệp tin đích
```

```
	/* Mở tệp tin nguồn */
	ifstream fileIn(sourceFile, ios::in);// Khai báo và mở tệp nguồn
	if(!fileIn){                         // Không mở được tệp nguồn
		cout << "Khong the mo duoc tep tin nguon "
			<< sourceFile << endl;
		exit(1);
	}

	/* Mở tệp tin đích */
	ofstream fileOut(targetFile, ios::out);// Khai báo và mở tệp đích
	if(!fileOut){                        // Không mở được tệp đích
		cout << "Khong the tao duoc tep tin dich "
			<< targetFile << endl;
		exit(1);
	}

	/* Đọc dữ liệu từ tệp tin ra tệp đích */
	while(fileIn){
		fileIn >> data;              // Đọc kí tự từ tệp nguồn
		fileOut << data;             // Ghi kí tự ra tệp đích
	}

	/* Đóng các tệp tin */
	fileIn.close();                      // Đóng tệp tin nguồn
	fileOut.close();                     // Đóng tệp tin đích
	return;
}
```

Lưu ý:

- Tên biến tệp, sau khi dùng xong với một tệp xác định, có thể sử dụng để mở một tệp khác, với một chế độ mở tệp khác bằng phép toán **open**() của biến tệp.

```
<Tên biến tệp>.open(<Tên tệp mới>, <chế độ mở mới>);
```

Ví dụ, đoạn chương trình:

```
ofstream myFile("abc.txt", ios::out);
…	// ghi vao file abc.txt
myFile.close();
myFile.open("xyz.txt", ios::out|ios::app);
…	// Them vao cuoi file xyz.txt
myFile.close();
```

sẽ dùng biến tệp myFile (có kiểu ofstream) hai lần: một lần là dùng với tệp tin abc.txt ở chế độ mở để ghi từ đầu. Một lần khác là với tệp tin xyz.txt ở chế độ mở để ghi thêm vào cuối.

4.2.2 Vào ra tệp nhị phân bằng read và write

Ghi vào tệp nhị phân bằng write

Các bước thực hiện để ghi dữ liệu vào một tệp nhị phân như sau:

1. Mở tệp tin theo chế độ để ghi nhị phân bằng đối tượng fstream:
   ```
   fstream <Tên biến tệp>(<Tên tệp tin>, ios::out|ios::binary);
   ```
2. Ghi dữ liệu vào tệp bằng thao tác "**write()**":
   ```
   <Tên biến tệp>.write(char* <Dữ liệu>,
                        int <Kích thước dữ liệu>);
   ```
3. Đóng tệp tin bằng lệnh close():
   ```
   <Tên biến tệp>.close();
   ```

Trong đó, thao tác **write** nhận hai tham số đầu vào như sau:

- Tham số thứ nhất là con trỏ kiểu char trỏ đến vùng dữ liệu cần ghi vào tệp. Vì con trỏ bắt buộc có kiểu char nên khi muốn ghi dữ liệu có kiểu khác vào tệp, ta dùng hàm chuyển kiểu:
  ```
  reinterpret_cast<char *>(<Dữ liệu>);
  ```
- Tham số thứ hai là kích cỡ dữ liệu được ghi vào tệp. Kích cỡ này được tính theo byte, nên thông thường ta dùng toán tử:
  ```
  sizeof(<Kiểu dữ liệu>);
  ```

Lưu ý:

- Khi muốn đọc, ghi các dữ liệu có cấu trúc (struct) vào tệp thì ta phải dùng ở chế độ đọc/ghi tệp nhị phân mà không thể dùng chế độ đọc/ghi ở chế độ văn bản.
- Khi đọc/ghi dữ liệu có kiểu cấu trúc, để toán tử sizeof() thực hiện chính xác thì các thành viên của cấu trúc không được là kiểu con trỏ. Vì toán tử sizeof() đối với con trỏ chỉ cho kích cỡ của con trỏ mà không cho kích cỡ thật của vùng dữ liệu mà con trỏ trỏ tới.

Chương trình 4.4 minh hoạ việc ghi dữ liệu vào tệp tin nhị phân, dữ liệu là kiểu cấu trúc:

- Tên tệp tin và số lượng bản ghi được người dùng tự nhập vào từ bàn phím.
- Chương trình sẽ ghi vào tệp các bản ghi có cấu trúc do người dùng gõ vào từ bàn phím.

Chương trình 4.4

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<type.h>

const int length = 25;                    // Độ dài tối đa tên tệp tin

typedef struct {
      int day;                            // Ngày
      int month;                          // Tháng
```

```
      int year;                              // Năm
} Date;

typedef struct {
      char name[20];                         // Tên nhân viên
      Date birthDay;                         // Ngày sinh của nhân viên
      char role[20];                         // Chức vụ của nhân viên
      float salary;                          // Lương của nhân viên
} Employee;

void main(){
      clrscr();
      char fileName[length];                 // Tên tệp tin
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;       // Nhập tên tệp tin

      int recordNumber;                      // Số lượng bản ghi
      cout << "So luong ban ghi: ";
      cin >> recordNumber;                   // Nhập số lượng bản ghi

      /* Mở tệp tin */
      // Khai báo và mở tệp tin
      fstream fileOut(fileName, ios::out|ios::binary);
      if(!fileOut){                          // Không mở được tệp
            cout << "Khong the tao duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Ghi dữ liệu vào tệp tin */
      Employee myEmployee;
      for(int i=0; i<recordNumber; i++){
            cout << "Ban ghi thu " << i+1 << endl;
            cout << "Name: ";
            cin >> myEmployee.name;          // Nhập tên nhân viên
            cout << "Day of birth: ";
            cin >> myEmployee.birthDay.day;// Nhập ngày sinh
            cout << "Month of birth: ";
            cin >> myEmployee.birthDay.month;// Nhập tháng sinh
            cout << "Year of birth: ";
            cin >> myEmployee.birthDay.year;// Nhập năm sinh
            cout << "Role: ";
            cin >> myEmployee.role;          // Nhập chức vụ
            cout << "Salary: ";
```

```
            cin >> myEmployee.salary;      // Nhập tiền lương

            // Ghi dữ liệu vào tệp
            fileOut.write(reinterpret_cast<char *>(&myEmployee),
                        sizeof(Employee));
      }

      /* Đóng tệp tin */
      fileOut.close();                     // Đóng tệp tin
      return;
}
```

Đọc dữ liệu từ tệp nhị phân bằng read

Các bước thực hiện để đọc dữ liệu từ một tệp tin nhị phân như sau:

1. Mở tệp tin theo chế độ để đọc nhị phân bằng đối tượng fstream (mở tệp tin chỉ để ghi):
   ```
   fstream <Tên biến tệp>(<Tên tệp tin>, ios::in|ios::binary);
   ```
2. Đọc dữ liệu từ tệp bằng thao tác "**read()**":
   ```
   <Tên biến tệp>.read(char* <Dữ liệu ra>,
                        int <Kích thước dữ liệu>);
   ```
3. Đóng tệp tin bằng lệnh close():
   ```
   <Tên biến tệp>.close();
   ```

Chương trình 4.5 minh hoạ việc đọc dữ liệu từ tệp tin vào biến có cấu trúc:

- Tên tệp tin được người dùng tự nhập vào từ bàn phím.
- Chương trình sẽ đọc các cấu trúc nhân viên trong tệp và hiển thị ra màn hình.
- Chương trình dừng lại khi kết thúc tệp tin.

Chương trình 4.5

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<type.h>

const int length = 25;                 // Độ dài tối đa tên tệp tin

typedef struct {
      int day;                         // Ngày
      int month;                       // Tháng
      int year;                        // Năm
} Date;
```

```
typedef struct {
      char name[20];                        // Tên nhân viên
      Date birthDay;                        // Ngày sinh của nhân viên
      char role[20];                        // Chức vụ của nhân viên
      float salary;                         // Lương của nhân viên
} Employee;

void main(){
      clrscr();
      char fileName[length];                // Tên tệp tin
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;      // Nhập tên tệp tin

      /* Mở tệp tin */
      // Khai báo và mở tệp tin
      fstream fileIn(fileName, ios::in|ios::binary);
      if(!fileIn){                          // Không mở được tệp
            cout << "Khong the mo duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Đọc dữ liệu từ tệp tin ra màn hình */
      Employee myEmployee;
      while(fileIn){
            fileIn.read(reinterpret_cast<char *>(&myEmployee),
                        sizeof(Employee));      // Đọc kí tự từ tệp tin
            cout << myEmployee.name << " "
                  << myEmployee.birthDay.day << "/"
                  << myEmployee.birthDay.month << "/"
                  << myEmployee.birthDay.year << "    "
                  << myEmployee.role << " "
                  << myEmployee.salary << endl; // Ghi kí tự ra màn hình
      }

      /* Đóng tệp tin */
      fileIn.close();                       // Đóng tệp tin
      return;
}
```

4.3 TRUY NHẬP TỆP TRỰC TIẾP

4.3.1 Con trỏ tệp tin

Con trỏ tệp tin có vai trò như một đầu đọc trỏ vào một vị trí xác định của tệp và thao tác truy nhập tệp diễn ra tuần tự:

- Tại mỗi thời điểm, con trỏ tệp tin xác định một vị trí trên tệp mà tại đó, thao tác truy nhập tệp (đọc/ghi) được thực hiện.
- Sau thao tác truy nhập, con trỏ tệp tự động chuyển đến vị trí tiếp theo dựa vào kích thước đơn vị dữ liệu được truy nhập.

Cách truy nhập tệp tuần tự có nhược điểm là bao giờ cũng phải bắt đầu từ đầu tệp tin, đi tuần tự cho đến vị trí cần truy nhập. Khi tệp tin có kích thước lớn thì cách truy nhập này rất tốn thời gian.

Để tránh nhược điểm này, C++ cho phép truy nhập trực tiếp đến một vị trí xác định trên tệp tin bằng các phép toán:

- Truy nhập vị trí hiện tại của con trỏ tệp tin
- Dịch chuyển con trỏ tệp tin đến một vị trí xác định

4.3.2 Truy nhập vị trí hiện tại của con trỏ tệp

Cú pháp truy nhập đến vị trí hiện thời của con trỏ tệp phụ thuộc vào kiểu biến tệp đang dùng là để đọc hay ghi.

- Nếu biến tệp là kiểu mở tệp để đọc **ifstream** thì cú pháp là:

```
<Tên biến tệp>.tellg();
```

- Nếu biến tệp là kiểu mở tệp để ghi **ofstream** thì cú pháp là:

```
<Tên biến tệp>.tellp();
```

Chương trình 4.6a minh hoạ việc xác định vị trí hiện thời của con trỏ tệp sau một số thao tác đọc tệp trước đó.

Chương trình 4.6a

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                    // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char fileName[length], output;
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;    // Nhập tên tệp tin
```

```
      /* Mở tệp tin */
      ifstream fileIn(fileName, ios::in); // Khai báo và mở tệp tin
      if(!fileIn){                        // Không mở được tệp
            cout << "Khong the mo duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Đọc dữ liệu từ tệp tin ra màn hình
      *  Ghi vi trí con trỏ tệp ra màn hình cứ sau 5 lần đọc kí tự */
      int index = 0;
      while(fileIn){
            fileIn >> output;             // Đọc kí tự từ tệp tin
            cout << output;               // Ghi kí tự ra màn hình
            if(index % 5 == 0)            // Ghi ra vị trí con trỏ tệp
                  cout<< endl << "Vi tri con tro tep: "
                        << fileIn.tellg() << endl;
            index ++;
      }
      cout << endl;                       // Xuống dòng trên màn hình

      /* Đóng tệp tin */
      fileIn.close();                     // Đóng tệp tin
      return;
}
```

Chương trình 4.6b minh hoạ việc xác định vị trí hiện thời của con trỏ tệp sau một số thao tác ghi vào tệp trước đó.

Chương trình 4.6b

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                    // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char fileName[length], input;
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;    // Nhập tên tệp tin
```

```
	/* Mở tệp tin */
	ofstream fileOut(fileName, ios::out);// Khai báo và mở tệp tin
	if(!fileOut){                        // Không mở được tệp
		cout << "Khong the tao duoc tep tin " << fileName << endl;
		exit(1);
	}

	/* Ghi dữ liệu vào tệp tin
	*  Hiện ra màn hình vị trí con trỏ tệp sau khi ghi được 5 kí tự*/
	int index = 0;
	do{
		cin >> input;                // Đọc kí tự từ bàn phím
		fileOut << input << ' ';     // Ghi kí tự vào tệp tin
		if(index%5 == 0)             // Hiển thị vị trí con trỏ tệp
			cout << "Vi tri con tro tep: "
				<< fileOut.tellp() << endl;
		index++;
	}while((input != 'e')&&(fileOut));
	fileOut << endl;                     // Xuống dòng cuối tệp tin

	/* Đóng tệp tin */
	fileOut.close();                     // Đóng tệp tin
	return;
}
```

4.3.3 Dịch chuyển con trỏ tệp

Ngoài việc xác định vị trí hiện thời của con trỏ tệp, C++ còn cho phép dịch chuyển con trỏ tệp đến một vị trí bất kì trên tệp. Cú pháp dịch chuyển phụ thuộc vào kiểu biến tệp là đọc hay ghi.

- Nếu biến tệp có kiểu là mở tệp tin để đọc **ifstream**, cú pháp sẽ là:

```
<Tên biến tệp>.seekg(<Kích thước>, <Mốc dịch chuyển>);
```

- Nếu biến tệp có kiểu là mở tệp để ghi **ofstream**, cú pháp sẽ là:

```
<Tên biến tệp>.seekp(<Kích thước>, <Mốc dịch chuyển>);
```

Trong đó:

- **Kích thước**: là tham số mô tả khoảng cách dịch chuyển so với vị trí mốc dịch chuyển. Đơn vị tính của kích thước là byte, có kiểu là số nguyên.
- **Mốc dịch chuyển**: là vị trí gốc để xác định khoảng cách dịch chuyển của con trỏ tệp. Có ba tham số hằng về kiểu mốc dịch chuyển:
  - **ios::beg**: Mốc dịch chuyển là đầu tệp tin.
  - **ios::cur**: Mốc dịch chuyển là vị trí hiện thời của con trỏ tệp.
  - **ios::end**: Mốc dịch chuyển là vị trí cuối cùng của tệp tin.

Ví dụ:

```
ifstream fileIn("abc.txt", ios::in);
fileIn.seekg(sizeof(char)*7, ios::beg);
```

sẽ dịch chuyển con trỏ tệp tin đến kí tự (kiểu char) thứ 7+1 = 8 trong tệp tin abc.txt để đọc (giả sử tệp tin abc.txt lưu các kí tự kiểu char).

Lưu ý:

- Vì khoảng cách cách dịch chuyển có kiểu số nguyên (int) cho nên có thể nhận giá trị âm hoặc dương. Nếu giá trị dương, dịch chuyển về phía sau vị trí làm mốc, nếu giá trị âm, dịch chuyển về phía trước vị trí làm mốc.
- Nếu vị trí dịch chuyển đến nằm ngoài phạm vi tệp tin (phía sau vị trí cuối cùng của tệp hoặc phía trước vị trí đầu tiên của tệp) sẽ nảy sinh lỗi, khi đó <Tên biến tệp> = false.

Chương trình 4.7 cài đặt chương trình truy nhập tệp tin trực tiếp để đọc giá trị kí tự (kiểu char) trong tệp:

- Tên tệp tin (chứa dữ liệu kiểu char) do người dùng nhập vào từ bàn phím.
- Sau đó, mỗi khi người dùng nhập vào một số nguyên, chương trình sẽ dịch chuyển đến vị trí mới, cách vị trí cũ đúng bằng từng ấy kí tự, tính từ vị trí hiện thời của con trỏ tệp.
- Chương trình sẽ kết thúc khi người dùng nhập vào số 0.

Chương trình 4.7

```
#include<stdlib.h>
#include<iostream.h>
#include<fstream.h>
#include<conio.h>

const int length = 25;                          // Độ dài tối đa tên tệp tin

void main(){
      clrscr();
      char fileName[length], output;
      cout << "Ten tep tin: ";
      cin >> setw(length) >> fileName;     // Nhập tên tệp tin

      /* Mở tệp tin */
      ifstream fileIn(fileName, ios::in); // Khai báo và mở tệp tin
      if(!fileIn){                              // Không mở được tệp
            cout << "Khong the mo duoc tep tin " << fileName << endl;
            exit(1);
      }

      /* Đọc dữ liệu từ tệp tin ra màn hình
      *  Ghi vi trí con trỏ tệp ra màn hình cứ sau 5 lần đọc kí tự */
```

```
	int index = 1;
	do{
		cout << "So ki tu dich chuyen: ";
		cin >> index;

		// Dịch chuyển con trỏ tệp từ vị trí hiện thời
		fileIn.seekg(sizeof(char)*index, ios::cur);
		if(fileIn){			// Đúng
			fileIn >> output;	// Đọc kí tự từ tệp tin

			// Ghi kí tự ra màn hình
			cout << "Vi tri: " << fileIn.tellg() << output;
		}else{				// Ra khỏi phạm vi tệp
			fileIn.clear();		// Về vị trí đầu tệp
		}
	}while(index);

	/* Đóng tệp tin */
	fileIn.close();			// Đóng tệp tin
	return;
}
```

TỔNG KẾT CHƯƠNG 4

Nội dung chương 4 đã tập trung trình bày các vấn đề liên quan đến các thao tác trên tệp tin trong ngôn ngữ C++. Bao gồm:

- Các bước tuần tự khi thao tác với một tệp tin:
  - Mở tập tin
  - Đọc/ghi dữ liệu trên tệp tin
  - Đóng tệp tin
- Thao tác mở tệp tin với nhiều chế độ bằng kiểu fstream.
- Thao tác mở tệp tin chỉ để đọc với kiểu ifstream
- Thao tác mở tệp tin chỉ để ghi với thao tác ofstream.
- Đọc dữ liệu từ tệp tin văn bản với thao tác ">>".
- Ghi dữ liệu vào tệp tin văn bản bằng thao tác "<<".
- Đọc tệp tin nhị phân bằng thao tác read().
- Ghi vào tệp tin nhị phân bằng thao tác write().
- Xác định vị trí hiện thời của con trỏ tệp tin với các thao tác tellg() và tellp().
- Dịch chuyển vị trí của con trỏ tệp với các thao tác seekg() và seekp().
- Thiết lập lại trạng thái cho con trỏ tệp tin bằng thao tác clear().

- Đóng tệp tin đã sử dụng bằng thao tác close().

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

1. Muốn mở một tệp tin tên là abc.txt để đọc dữ liệu, lệnh mở tệp nào sau đây là đúng:

   a. `fstream myFile("abc.txt", ios::in);`

   b. `fstream myFile("abc.txt", ios::out);`

   c. `fstream myFile("abc.txt", ios::app);`

   d. `fstream myFile("abc.txt", ios::ate);`

2. Muốn mở một tệp tin abc.txt nằm trong thư mục xyz để ghi dữ liệu vào. Lệnh mở nào sau đây là đúng:

   a. `fstream myFile("xyz\abc.txt", ios::out);`

   b. `fstream myFile("xyz\\abc.txt", ios::out);`

   c. `fstream myFile("xyz/abc.txt", ios::out);`

   d. `fstream myFile("xyz//abc.txt", ios::out);`

3. Muốn mở một tệp tin abc.txt để ghi thêm dữ liệu vào cuối tệp, lệnh nào sau đây là đúng:

   a. `fstream myFile("abc.txt", ios::out);`

   b. `fstream myFile("abc.txt", ios::app);`

   c. `fstream myFile("abc.txt", ios::out|ios::app);`

   d. `fstream myFile("abc.txt", ios::out||ios::app);`

4. Xét hai lệnh khai báo sau:

   ```
   fstream myFile1("abc.txt", ios::out);
   ofstream myFile2("abc.txt", ios::out);
   ```

   Nhận định nào sau đây là đúng:

   a. myFile1 và myFile2 có chức năng giống nhau.

   b. myFile1 và myFile2 có chức năng khác nhau

5. Xét hai lệnh khai báo sau:

   ```
   fstream myFile1("abc.txt", ios::in);
   ifstream myFile2("abc.txt", ios::in);
   ```

   Nhận định nào sau đây là đúng:

   a. myFile1 và myFile2 có chức năng giống nhau.

   b. myFile1 và myFile2 có chức năng khác nhau

6. Xét đoạn chương trình sau:

   ```
   ofstream myFile("abc.txt", ios::out);
   if(myFile) myFile << "abc.txt";
   ```

   Chương trình sẽ làm gì?

   a. Ghi ra màn hình dòng chữ "abc.txt"

   b. Ghi vào tệp tin abc.txt dòng chữ "abc.txt"

   c. Đọc từ tệp tin abc.txt dòng chữ "abc.txt"

d. Chương trình sẽ báo lỗi.

7. Xét đoạn chương trình sau:

```
ifstream myFile("abc.txt", ios::in);
char text[20];
if(myFile) myFile >> text;
```

Chương trình sẽ làm gì, nếu tệp tin abc.txt có nội dung là dòng chữ "abc.txt"?

a. Ghi ra màn hình dòng chữ "abc.txt"

b. Ghi vào tệp tin abc.txt dòng chữ "abc.txt"

c. Đọc từ tệp tin abc.txt dòng chữ "abc.txt"

d. Chương trình sẽ báo lỗi.

8. Xét đoạn chương trình sau:

```
fstream myFile("abc.txt", ios::out);
if(myFile) myFile << "abc.txt";
myFile.close();
myFile.open("abc.txt", ios::in);
char text[20];
if(myFile) myFile >> text;
cout << text;
```

Chương trình sẽ làm gì, nếu tệp tin abc.txt có nội dung là dòng chữ "abc.txt"?

a. Ghi vào tệp tin abc.txt dòng chữ "abc.txt"

b. Đọc từ tệp tin abc.txt dòng chữ "abc.txt"

c. Ghi ra màn hình dòng chữ "abc.txt"

d. Cả ba đáp án trên.

e. Chương trình sẽ báo lỗi.

9. Xét đoạn chương trình sau:

```
ifstream myFile("abc.txt", ios::in);
if(myFile) cout << myFile.tellg();
```

Chương trình sẽ in ra màn hình kết quả gì?

a. 0

b. 1

c. 8

d. 16

10. Xét đoạn chương trình sau, nếu tệp abc.txt chứa một số lượng kí tự đủ lớn:

```
ifstream myFile("abc.txt", ios::in);
if(myFile){
        char c;
        myFile >> c;
        cout << myFile.tellg();
}
```

Chương trình sẽ in ra màn hình kết quả gì?

a. 0

b. 1

c. 8

d. 16

11. Xét đoạn chương trình sau, nếu tệp abc.txt chứa một số lượng kí tự đủ lớn:

```
ifstream myFile("abc.txt", ios::in);
if(myFile){
        myFile.seekg(sizeof(char)*5, ios::beg);
        myFile.seekg(sizeof(char)*5, ios::cur);
        cout << myFile.tellg();
}
```

Chương trình sẽ in ra màn hình kết quả gì?

a. 0

b. 5

c. 10

d. 80

12. Viết một chương trình gộp nội dung của hai tệp tin có sẵn vào một tệp tin thứ ba. Tên các tệp tin được nhập vào từ bàn phím.
13. Viết một chương trình tìm kiếm trên tệp nhị phân có cấu trúc được tạo bởi chương trình 4.4: Tìm tất cả các nhân viên có tên là X, X được nhập từ bàn phím. Hiển thị kết quả là tất cả các thông tin về các nhân viên được tìm thấy.
14. Viết một chương trình tìm kiếm trên tệp nhị phân có cấu trúc được tạo bởi chương trình 4.4: Tìm tất cả các nhân viên có năm sinh là X, X được nhập từ bàn phím. Hiển thị kết quả là tất cả các thông tin về các nhân viên được tìm thấy.
15. Viết một chương trình tìm kiếm trên tệp nhị phân có cấu trúc được tạo bởi chương trình 4.4: Tìm tất cả các nhân viên có lương cao hơn hoặc bằng một giá trị X, X được nhập từ bàn phím. Hiển thị kết quả là tất cả các thông tin về các nhân viên được tìm thấy.
16. Viết một chương trình sao chép một đoạn đầu nội dung của một tệp tin vào một tệp tin thứ hai. Tên các tệp tin và độ dài đoạn nội dung cần sao chép được nhập từ bàn phím.
17. Viết một chương trình sao chép một đoạn cuối nội dung của một tệp tin vào một tệp tin thứ hai. Tên các tệp tin và độ dài đoạn nội dung cần sao chép được nhập từ bàn phím.

CHƯƠNG 5

LỚP

Nội dung chương này tập trung trình bày các vấn đề liên quan đến lớp đối tượng trong C++:

- Khái niệm, khai báo và sử dụng lớp
- Khai báo và sử dụng các thành phần của lớp: các thuộc tính và các phương thức của lớp
- Phạm vi truy nhập lớp
- Khai báo và sử dụng các phương thức khởi tạo và huỷ bỏ của lớp
- Sử dụng lớp thông qua con trỏ đối tượng, mảng các đối tượng.

5.1 KHÁI NIỆM LỚP ĐỐI TƯỢNG

C++ coi lớp là sự trừu tượng hóa các đối tượng, là một khuôn mẫu để biểu diễn các đối tượng thông qua các thuộc tính và các hành động đặc trưng của đối tượng.

5.1.1 Định nghĩa lớp đối tượng

Để định nghĩa một lớp trong C++, ta dùng từ khóa **class** với cú pháp:

```
class <Tên lớp>{
};
```

Trong đó:

- **class**: là tên từ khóa bắt buộc để định nghĩa một lớp đối tượng trong C++.
- **Tên lớp**: do người dùng tự định nghĩa. Tên lớp có tính chất như tên kiểu dữ liệu để sử dụng sau này. Cách đặt tên lớp phải tuân thủ theo quy tắc đặt tên biến trong C++.

Ví dụ:

```
class Car{
};
```

là định nghĩa một lớp xe ô tô (Car). Lớp này chưa có bất kì một thành phần nào, việc định nghĩa các thành phần cho lớp sẽ được trình bày trong mục 5.2.

Lưu ý:

- Từ khóa **class** là bắt buộc để định nghĩa một lớp đối tượng trong C++. Hơn nữa, C++ có phân biệt chữ hoa chữ thường trong khai báo cho nên chữ **class** phải được viết bằng chữ thường.

  Ví dụ:

  ```
  class Car{  // Định nghĩa đúng
  };
  ```

  nhưng:

  ```
  Class Car{  // Lỗi từ khóa
  };
  ```

- Bắt buộc phải có dấu chấm phẩy ";" ở cuối định nghĩa lớp vì C++ coi định nghĩa một lớp như định nghĩa một kiểu dữ liệu, cho nên phải có dấu chấm phẩy cuối định nghĩa (tương tự định nghĩa kiểu dữ liệu kiểu cấu trúc).
- Để phân biệt với tên biến thông thường, ta nên (nhưng không bắt buộc) đặt *tên lớp bắt đầu bằng một chữ in hoa và các tên biến bắt đầu bằng một chữ in thường*.

5.1.2 Sử dụng lớp đối tượng

Lớp đối tượng được sử dụng khi ta khai báo các thể hiện của lớp đó. Một thể hiện của một lớp chính là một đối tượng cụ thể của lớp đó. Việc khai báo một thể hiện của một lớp được thực hiện như cú pháp khai báo một biến có kiểu lớp:

```
<Tên lớp> <Tên biến lớp>;
```

Trong đó:

- **Tên lớp**: là tên lớp đối tượng đã được định nghĩa trước khi khai báo biến.
- **Tên biến lớp**: là tên đối tượng cụ thể. Tên biến lớp sẽ được sử dụng như các biến thông thường trong C++, ngoại trừ việc nó có kiểu lớp đối tượng.

Ví dụ, muốn khai báo một thể hiện (biến) của lớp Car đã được định nghĩa trong mục 5.1.1, ta khai báo như sau:

```
Car myCar;
```

Sau đó, ta có thể sử dụng biến myCar trong chương trình như các biến thông thường: truyền tham số cho hàm, gán cho biến khác …

Lưu ý:

- Khi khai báo biến lớp, ta không dùng lại từ khóa **class** nữa. Từ khóa **class** chỉ được sử dụng khi định nghĩa lớp mà không dùng khi khai báo biến lớp.

Ví dụ, khai báo:

```
Car myCar;          // đúng
```

là đúng, nhưng khai báo:

```
class Car myCar;    // Lỗi cú pháp
```

là sai cú pháp.

5.2 CÁC THÀNH PHẦN CỦA LỚP

Việc khai báo các thành phần của lớp có dạng như sau:

```
class <Tên lớp>{
private:
      <Khai báo các thành phần riêng>
protected:
      <Khai báo các thành phần được bảo vệ>
public:
      <Khai báo các thành phần công cộng>
};
```

Trong đó:

- **private**: là từ khóa chỉ tính chất của C++ để chỉ ra rằng các thành phần được khai báo trong phạm vi từ khóa này là riêng tư đối với lớp đối tượng. Các đối tượng của các lớp khác không truy nhập được các thành phần này.
- **protected**: các thành phần được khai báo trong phạm vi từ khóa này đều được bảo vệ. Qui định loại đối tượng nào được truy nhập đến các thành phần được bảo vệ sẽ được mô tả chi tiết trong mục 5.3.
- **public**: các thành phần công cộng. Các đối tượng của các lớp khác đều có thể truy nhập đến các thành phần công cộng của một đối tượng bất kì.

Các thành phần của lớp được chia làm hai loại:

- Các thành phần chỉ dữ liệu của lớp, được gọi là *thuộc tính* của lớp
- Các thành phần chỉ hành động của lớp, được gọi là *phương thức* của lớp.

5.2.1 Thuộc tính của lớp

Khai báo thuộc tính

Thuộc tính của lớp là thành phần chứa dữ liệu, đặc trưng cho các tính chất của lớp. Thuộc tính của lớp được khai báo theo cú pháp sau:

```
<Kiểu dữ liệu> <Tên thuộc tính>;
```

Trong đó:

- **Kiểu dữ liệu**: có thể là các kiểu dữ liệu cơ bản của C++, cũng có thể là các kiểu dữ liệu phức tạp do người dùng tự định nghĩa như struct, hoặc kiểu là một lớp đã được định nghĩa trước đó.
- **Tên thuộc tính**: là tên thuộc tính của lớp, có tính chất như một biến thông thường. Tên thuộc tính phải tuân theo quy tắc đặt tên biến của C++.

Ví dụ, khai báo:

```
class Car{
private:
      int speed;
public:
      char mark[20];
};
```

là khai báo một lớp xe ô tô (Car), có hai thuộc tính: thuộc tính tốc độ (speed) có tính chất private, thuộc tính nhãn hiệu xe (mark) có tính chất public.

Lưu ý:

- Không được khởi tạo giá trị ban đầu cho các thuộc tính ngay trong lớp. Vì các thuộc tính chỉ có giá trị khi nó gắn với một đối tượng cụ thể, là một thể hiện (biến) của lớp.

  Ví dụ:

```
class Car{
private:
      int speed;         // đúng
      int weight = 500; // lỗi
```

```
};
```

- Khả năng truy nhập thuộc tính của lớp là phụ thuộc vào thuộc tính ấy được khai báo trong phạm vi của từ khóa nào: private, protected hay public.
- Thông thường, do yêu cầu đóng gói dữ liệu của hướng đối tượng, ta nên khai báo các thuộc tính có tính chất riêng tư (ptivate). Nếu muốn các đối tượng khác truy nhập được vào các thuộc tính này, ta xây dựng các hàm public truy nhập (get / set) đến thuộc tính đó.

Sử dụng thuộc tính

Thuộc tính có thể được sử dụng cho các chương trình nằm ngoài lớp thông qua tên biến lớp hoặc sử dụng ngay trong lớp bởi các phương thức của lớp.

- Nếu thuộc tính được dùng bên ngoài phạm vi lớp, cú pháp phải thông qua tên biến lớp (cách này chỉ sử dụng được với các biến có tính chất public):

```
<Tên biến lớp>.<tên thuộc tính>;
```

- Nếu thuộc tính được dùng bên trong lớp, cú pháp đơn giản hơn:

```
<Tên thuộc tính>;
```

Ví dụ, với định nghĩa lớp:

```
class Car{
private:
	int speed;
public:
	char mark[20];
};
```

ta khai báo một biến lớp:

```
Car myCar;
```

Thì có thể sử dụng thuộc tính nhãn hiệu xe khi in ra màn hình như sau:

```
cout << myCar.mark;
```

Lưu ý:

- Khi dùng thuộc tính bên trong các phương thức của lớp, mà tên thuộc tính lại bị trùng với tên biến toàn cục (tự do) của chương trình, ta phải chỉ rõ việc dùng tên thuộc tính của lớp (mà không phải tên biến toàn cục) bằng cách dùng chỉ thị phạm vi lớp "::" với cú pháp:

```
<Tên lớp>::<Tên thuộc tính>;
```

5.2.2 Phương thức của lớp

Khai báo khuôn mẫu phương thức

Một phương thức là một thao tác thực hiện một số hành động đặc trưng của lớp đối tượng. Phương thức được khai báo tương tự như các hàm trong C++:

```
<Kiểu trả về> <Tên phương thức>([<Các tham số>]);
```

Trong đó:

- **Kiểu trả về**: là kiểu dữ liệu trả về của phương thức. Kiểu có thể là các kiểu dữ liệu cơ bản của C++, cũng có thể là kiểu do người dùng định nghĩa, hoặc kiểu lớp đã được định nghĩa.

- **Tên phương thức**: do người dùng tự đặt tên, tuân theo quy tắc đặt tên biến của C++.
- **Các tham số**: Các tham số đầu vào của phương thức, được biểu diễn bằng kiểu dữ liệu tương ứng. Các tham số được phân cách bởi dấu phẩy ",". Các tham số là tùy chọn (Phần trong dấu ngoặc vuông "[]" là tùy chọn).

Ví dụ, khai báo:

```
class Car{
private:
      int speed;
      char mark[20];
public:
      void show();
};
```

là định nghĩa một lớp Car có hai thuộc tính cục bộ là speed và mark, và khai báo một phương thức show() để mô tả đối tượng xe tương ứng. Show() là một phương thức không cần tham số và kiểu trả về là void.

Lưu ý:

- Khả năng truy nhập phương thức từ bên ngoài là phụ thuộc vào phương thức được khai báo trong phạm vi của từ khóa nào: private, protected hay public.

Định nghĩa phương thức

Trong C++, việc cài đặt chi tiết nội dung của phương thức có thể tiến hành ngay trong phạm vi lớp hoặc bên ngoài phạm vi định nghĩa lớp. Cú pháp chỉ khác nhau ở dòng khai báo tên phương thức.

- Nếu cài đặt phương thức ngay trong phạm vi định nghĩa lớp, cú pháp là:

```
<Kiểu trả về> <Tên phương thức>([<Các tham số>]){
      … // Cài đặt chi tiết
}
```

- Nếu cài đặt phương thức bên ngoài phạm vi định nghĩa lớp, ta phải dùng chỉ thị phạm vi "::" để chỉ ra rằng đấy là một phương thức của lớp mà không phải là một hàm tự do trong chương trình:

```
<Kiểu trả về> <Tên lớp>::<Tên phương thức>([<Các tham số>]){
      … // Cài đặt chi tiết
}
```

Ví dụ, nếu cài đặt phương thức show() của lớp Car ngay trong phạm vi định nghĩa lớp, ta cài đặt như sau:

```
class Car{
private:
      int   speed;      // Tốc độ
      char  mark[20];   // Nhãn hiệu
public:
      void show(){      // Khai báo phương thức ngay trong lớp
            cout << "This is a " << mark << " having a speed of "
```

```
                    << speed << "km/h!" << endl;
            return;
        }
};
```

Nếu muốn cài đặt bên ngoài lớp, ta cài đặt như sau:

```
class Car{
private:
        int   speed;        // Tốc độ
        char  mark[20];     // Nhãn hiệu
public:
        void show();        // Giới thiệu xe
};
/* Khai báo phương thức bên ngoài lớp */
void Car::show(){
        cout << "This is a " << mark << " having a speed of "
                << speed << "km/h!" << endl;
        return;
}
```

Lưu ý:

- Nếu phương thức được cài đặt ngay trong lớp thì các tham số phải tường minh, nghĩa là mỗi tham số phải được biểu diễn bằng một cặp <Kiểu dữ liệu> <Tên tham số> như khi cài đặt chi tiết một hàm tự do trong chương trình.
- Thông thường, chỉ các phương thức ngắn (trên một dòng) là nên cài đặt ngay trong lớp. Còn lại nên cài đặt các phương thức bên ngoài lớp để chương trình được sáng sủa, rõ ràng và dễ theo dõi.

Sử dụng phương thức

Cũng tương tự như các thuộc tính của lớp, các phương thức cũng có thể được sử dụng bên ngoài lớp thông qua tên biến lớp, hoặc có thể được dùng ngay trong lớp bởi các phương thức khác của lớp định nghĩa nó.

- Nếu phương thức được dùng bên ngoài phạm vi lớp, cú pháp phải thông qua tên biến lớp (cách này chỉ sử dụng được với các phương thức có tính chất public):

```
<Tên biến lớp>.<Tên phương thức>([<Các đối số>]);
```

- Nếu thuộc tính được dùng bên trong lớp, cú pháp đơn giản hơn:

```
<Tên phương thức>([<Các đối số>]);
```

Ví dụ, với định nghĩa lớp:

```
class Car{
private:
        int   speed;        // Tốc độ
        char  mark[20];     // Nhãn hiệu
public:
        void show();        // Giới thiệu xe
```

```
};
/* Khai báo phương thức bên ngoài lớp */
void Car::show(){
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h!" << endl;
      return;
}
```

ta khai báo một biến lớp:

```
Car myCar;
```

Thì có thể sử dụng phương thức giới thiệu xe như sau:

```
myCar.show();
```

Lưu ý:

- Khi dùng phương thức bên trong các phương thức khác của lớp, mà phương thức lại bị trùng với các phương thức tự do của chương trình, ta phải chỉ rõ việc dùng phương thức của lớp (mà không phải dùng phương thức tự do) bằng cách dùng chỉ thị phạm vi lớp "::" với cú pháp:

```
<Tên lớp>::<Tên phương thức>([<Các đối số>]);
```

Chương trình 5.1 cài đặt đầy đủ một lớp xe ô tô (Car) với các thuộc tính có tính chất cục bộ:

- Tốc độ xe (speed)
- Nhãn hiệu xe (mark)
- Giá xe (price)

Và các phương thức có tính chất public:

- Khởi tạo các tham số (init)
- Giới thiệu xe (show)
- Các phương thức truy nhập (get/set) các thuộc tính

Sau đó, chương trình main sẽ sử dụng lớp Car này để định nghĩa các đối tượng cụ thể và sử dụng các phương thức của lớp này.

Chương trình 5.1

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            void  setSpeed(int);          // Gán tốc độ cho xe
```

```
            int   getSpeed();              // Đọc tốc độ xe
            void  setMark(char);           // Gán nhãn cho xe
            char[] getMark();              // Đọc nhãn xe
            void  setPrice(float);         // Gán giá cho xe
            float getPrice();              // Đọc giá xe
            void  init(int, char[], float);// Khởi tạo thông tin về xe
            void  show();                  // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
void Car::setSpeed(int speedIn){    // Gán tốc độ cho xe
      speed = speedIn;
}
int Car::getSpeed(){                // Đọc tốc độ xe
      return speed;
}
void Car::setMark(char markIn){     // Gán nhãn cho xe
      strcpy(mark, markIn);
}
char[] Car::getMark(){              // Đọc nhãn xe
      return mark;
}
void Car::setPrice(float priceIn){  // Gán giá cho xe
      price = priceIn;
}
float Car::getPrice(){              // Đọc giá xe
      return price;
}

void Car::init(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
      return;
}

void Car::show(){                   // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

// Hàm main, chương trình chính
```

```
void main(){
      clrscr();
      Car myCar;                    // Khai báo biến lớp

      // Khởi tạo lần thứ nhất
      cout << "Xe thu nhat: " << endl;
      myCar.init(100, "Ford", 3000);
      cout << "Toc do (km/h): " << myCar.getSpeed() << endl;
      cout << "Nhan hieu : " << myCar.getMark() << endl;
      cout << "Gia ($): " << myCar.getPrice() << endl;

      // Thay đổi thuộc tính xe
      cout << "Xe thu hai: " << endl;
      myCar.setSpeed(150);
      myCar.setMark("Mercedes");
      myCar.setPrice(5000);
      myCar.show();
      return;
}
```

Chương trình 5.1 sẽ in ra kết quả như sau:

```
Xe thu nhat:
Toc do (km/h): 100
Nhan hieu: Ford
Gia ($): 3000
Xe thu hai:
This is a Mercedes having a speed of 150km/h and its price is $5000
```

5.3 PHẠM VI TRUY NHẬP LỚP

5.3.1 Phạm vi truy nhập lớp

Trong C++, có một số khái niệm về phạm vi, xếp từ bé đến lớn như sau:

- **Phạm vi khối lệnh**: Trong phạm vi giữa hai dấu giới hạn "{}" của một khối lệnh. Ví dụ các lệnh trong khối lệnh lặp while(){} sẽ có cùng phạm vi khối lệnh.
- **Phạm vi hàm**: Các lệnh trong cùng một hàm có cùng mức phạm vi hàm.
- **Phạm vi lớp**: Các thành phần của cùng một lớp có cùng phạm vi lớp với nhau: các thuộc tính và các phương thức của cùng một lớp.
- **Phạm vi chương trình (còn gọi là phạm vi tệp)**: Các lớp, các hàm, các biến được khai báo và định nghĩa trong cùng một tệp chương trình thì có cùng phạm vi chương trình.

Trong phạm vi truy nhập lớp, ta chỉ quan tâm đến hai phạm vi lớn nhất, đó là phạm vi lớp và phạm vi chương trình. Trong C++, phạm vi truy nhập lớp được quy định bởi các từ khóa về thuộc tính truy nhập:

- **private**: Các thành phần của lớp có thuộc tính private thì chỉ có thể được truy nhập trong phạm vi lớp.
- **protected**: Trong cùng một lớp, thuộc tính protected cũng có ảnh hưởng tương tự như thuộc tính private: các thành phần lớp có thuộc tính protected chỉ có thể được truy nhập trong phạm vi lớp. Ngoài ra nó còn có thể được truy nhập trong các lớp con khi có kế thừa (sẽ được trình bày trong chương 6).
- **public**: các thành phần lớp có thuộc tính public thì có thể được truy nhập trong phạm vi chương trình, có nghĩa là nó có thể được truy nhập trong các hàm tự do, các phương thức bên trong các lớp khác…

Ví dụ, thuộc tính price của lớp Car có tính chất private nên chỉ có thể truy nhập bởi các phương thức của lớp Car. Không thể truy nhập từ bên ngoài lớp (phạm vi chương trình), chẳng hạn trong một hàm tự do ngoài lớp Car.

```
void Car::setPrice(float priceIn){
      price = priceIn;  // Đúng, vì setPrice là một phương thức
                        // của lớp Car
}
```

nhưng:

```
void freeFunction(Car myCar){
      myCar.price = 3000;// Lỗi, vì freeFunction là một hàm tự do
                         // nằm ngoài phạm vi lớp Car
}
```

Khi đó, hàm freeFunction phải truy nhập gián tiếp đến thuộc tính price thông qua phương thức truy nhập có tính chất public như sau:

```
void freeFunction(Car myCar){
      myCar.setPrice(3000);//  Đúng,  vì  setPrice  là  một  phương  thức
của
                           // lớp Car có thuộc tính public
}
```

Tuy nhiên, C++ cho phép một cách đặc biệt để truy nhập đến các thành phần private và protected của một lớp bằng khái niệm *hàm bạn* và *lớp bạn* của một lớp: trong các hàm bạn và lớp bạn của một lớp, có thể truy nhập đến các thành phần private và protected như bên trong phạm vi lớp đó.

5.3.2 Hàm bạn

Có hai kiểu hàm bạn cơ bản trong C++:

- Một hàm tự do là hàm bạn của một lớp
- Một hàm thành phần (phương thức) của một lớp là bạn của một lớp khác

Ngoài ra còn có một số kiểu hàm bạn mở rộng từ hai kiểu này:

- Một hàm là bạn của nhiều lớp

- Tất cả các hàm của một lớp là bạn của lớp khác (lớp bạn)

Hàm tự do bạn của một lớp

Một hàm bạn của một lớp được khai báo bằng từ khóa **friend** khi khai báo khuôn mẫu hàm trong lớp tương ứng.

```
class <Tên lớp>{
…     // Khai báo các thành phần lớp như thông thường
// Khai báo hàm bạn
friend <Kiểu trả về> <Tên hàm bạn>([<Các tham số>]);
};
```

Khi đó, định nghĩa chi tiết hàm bạn được thực hiện như định nghĩa một hàm tự do thông thường:

```
<Kiểu trả về> <Tên hàm bạn>([<Các tham số>]){
…     // Có thể truy nhập trực tiếp các thành phần private
      // của lớp đã khai báo
}
```

Lưu ý:

- Mặc dù hàm bạn được khai báo khuôn mẫu hàm trong phạm vi lớp, nhưng hàm bạn tự do lại không phải là một phương thức của lớp. Nó là hàm tự do, việc định nghĩa và sử dụng hàm này hoàn toàn tương tự như các hàm tự do khác.
- Việc khai báo khuôn mẫu hàm bạn trong phạm vi lớp ở vị trí nào cũng được: hàm bạn không bị ảnh hưởng bởi các từ khóa private, protected hay public trong lớp.
- Trong hàm bạn, có thể truy nhập trực tiếp đến các thành phần private và protected của đối tượng có kiểu lớp mà nó làm bạn (truy nhập thông qua đối tượng cụ thể).

Chương trình 5.2 minh họa việc định nghĩa một hàm bạn của lớp Car, hàm này so sánh xem hai chiếc xe, chiếc nào đắt hơn.

Chương trình 5.2

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            void  init(int, char[], float);// Khởi tạo thông tin về xe
            // Khai báo hàm bạn của lớp
            friend void moreExpensive(Car, Car);
};
```

```
/* Khai báo phương thức bên ngoài lớp */
void Car::init(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
      return;
}
/* Định nghĩa hàm bạn tự do */
void moreExpensive(Car car1, Car car2){
      if(car1.price > car2.price)//Truy nhập đến các thuộc tính private
            cout << "xe thu nhat dat hon" << endl;
      else if(car1.price < car2.price)
            cout << "xe thu nhat dat hon" << endl;
      else
            cout << "hai xe dat nhu nhau" << endl;
      return;
}

// Hàm main, chương trình chính
void main(){
      clrscr();
      Car car1, car2;                    // Khai báo biến lớp

      // Khởi tạo xe thứ nhất, thứ hai
      car1.init(100, "Ford", 3000);
      car2.init(150, "Mercedes", 3500);

      // So sánh giá hai xe
      moreExpensive(car1, car2);         // Sử dụng hàm bạn tự do
      return;
}
```

Chương trình 5.2 sẽ in ra thông báo:

```
xe thu hai dat hon
```

vì xe thứ hai có giá $3500, trong khi xe thứ nhất được khởi tạo giá là $3000.

Phương thức lớp là bạn của một lớp khác

Trong C++, một phương thức của lớp này cũng có thể làm bạn của một lớp kia. Để khai báo một phương thức f của lớp B là bạn của lớp A và f nhận một tham số có kiểu lớp A, ta phải khai báo tuần tự như sau (trong cùng một chương trình):

- Khai báo khuôn mẫu lớp A, để làm tham số cho hàm f của lớp B:

```
class A;
```

- Khai báo lớp B với hàm f như khai báo các lớp thông thường:

```
class B{
      …     // Khai báo các thành phần khác của lớp B
      void f(A);
};
```

- Khai báo chi tiết lớp A với hàm f của lớp B là bạn

```
class A{
      …     // Khai báo các thành phần khác của lớp A
      friend void B::f(A);
};
```

- Định nghĩa chi tiết hàm f của lớp B:

```
void B::f(A){
      …     // Định nghĩa chi tiết hàm f
}
```

Lưu ý:

- Trong trường hợp này, hàm f chỉ được định nghĩa chi tiết một khi lớp A đã được định nghĩa chi tiết. Do vậy, chỉ có thể định nghĩa chi tiết hàm f ngay trong lớp A (ở bước 3) hoặc sau khi định nghĩa lớp A (ở bước 4), mà không thể định nghĩa chi tiết hàm f ngay trong lớp B (ở bước 2).
- Hàm f có thể truy nhập đến các thành phần private và protected của cả hai lớp A và B. Tuy nhiên, muốn f truy nhập đến các thành phần của lớp A thì phải thông qua một đối tượng cụ thể có kiểu lớp A.

Chương trình 5.3 minh họa việc cài đặt và sử dụng một hàm permission() của lớp Person, là hàm bạn của lớp Car. Hàm này thực hiện việc kiểm tra xem một người có đủ quyền điều khiển xe hay không, theo luật sau:

- Với các loại xe thông thường, người điều khiển phải đủ 18 tuổi.
- Với các loại xe có tốc độ cao hơn 150km/h, người điều khiển phải đủ 21 tuổi.

Chương trình 5.3

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

class Car;                                  // Khai báo nguyên mẫu lớp

/* Định nghĩa lớp Person */
class Person{
      private:
            char  name[25];                 // Tên
            int   age;                      // Tuổi
```

```
      public:
            void init(char[], int);       // Khởi tạo thông tin về người
            int permission(Car);          // Xác định quyền điều khiển xe
};

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            void  init(int, char[], float);// Khởi tạo thông tin về xe
            // Khai báo hàm bạn của lớp
            friend int Person::permission(Car);
};

/* Khai báo phương thức bên ngoài lớp */
void Person::init(char nameIn[], int ageIn){
      strcpy(name, nameIn);
      age = ageIn;
      return;
}

void Car::init(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
      return;
}

/* Định nghĩa hàm bạn */
int Person::permission(Car car){
      if(age < 18)
            return 0;
      //Truy nhập thuộc tính private thông qua đối tượng car
      if((age < 21)&&(car.speed > 150))
            return 0;
      return 1;
}

// Hàm main, chương trình chính
void main(){
```

```
      clrscr();
      // Khai báo các biến lớp
      Car car;
      Person person;

      // Khởi tạo các đối tượng
      car.init(100, "Ford", 3000);
      person.init("Vinh", 20);

      // Xác định quyền điều khiển xe
      if(person.permission(car))          // Sử dụng hàm bạn
            cout << "Co quyen dieu khien" << endl;
      else
            cout << "Khong co quyen dieu khien" << endl;
      return;
}
```

Chương trình 5.3 sẽ hiển thị thông báo:

```
Co quyen dieu khien
```

Vì người chủ xe đã 20 tuổi và xe chỉ có tốc độ 100km/h.

5.3.3 Lớp bạn

Khi tất cả các phương thức của một lớp là bạn của một lớp khác, thì lớp của các phương thức đó cũng trở thành lớp bạn của lớp kia. Muốn khai báo một lớp B là lớp bạn của lớp A, ta khai báo theo tuần tự sau:

- Khai báo khuôn mẫu lớp B:

```
class B;
```

- Định nghĩa lớp A, với khai báo B là lớp bạn:

```
class A{
      …     // Khai báo các thành phần của lớp A
      // Khai báo lớp bạn B
      friend class B;
};
```

- Định nghĩa chi tiết lớp B:

```
class B{
      …     // Khai báo các thành phần của lớp B
};
```

Lưu ý:

- Trong trường hợp này, lớp B là lớp bạn của lớp A nhưng không có nghĩa là lớp A cũng là bạn của lớp B: tất cả các phương thức của lớp B có thể truy nhập các thành phần private của lớp A (thông qua các đối tượng có kiểu lớp A) nhưng các phương thức của lớp A lại không thể truy nhập đến các thành phần private của lớp B.

- Muốn các phương thức của lớp A cũng truy nhập được đến các thành phần private của lớp B, thì phải khai báo thêm là lớp A cũng là lớp bạn của lớp B.

5.4 HÀM KHỞI TẠO VÀ HUỶ BỎ

5.4.1 Hàm khởi tạo

Hàm khởi tạo được gọi mỗi khi khai báo một đối tượng của lớp. Ngay cả khi không được khai báo tường minh, C++ cũng gọi hàm khởi tạo ngầm định khi đối tượng được khai báo.

Khai báo hàm khởi tạo

Hàm khởi tạo của một lớp được khai báo tường minh theo cú pháp sau:

```
class <Tên lớp>{
      public:
            <Tên lớp>([<Các tham số>]);    // Khai báo hàm khởi tạo
};
```

Ví dụ:

```
class Car{
            int   speed;
            char  mark[20];
            float price;
      public:
            Car(int speedIn, char markIn[], float priceIn){
                  speed = speedIn;
                  strcpy(mark, markIn);
                  price = priceIn;
            }
};
```

là khai báo một hàm khởi tạo với ba tham số của lớp Car.

Lưu ý:

- Hàm khởi tạo phải có tên trùng với tên của lớp
- Hàm khởi tạo không có giá trị trả về
- Hàm khởi tạo có tính chất public
- Có thể có nhiều hàm khởi tạo của cùng một lớp

Sử dụng hàm khởi tạo của lớp

Hàm khởi tạo được sử dụng khi khai báo biến lớp. Khi đó, ta có thể vừa khai báo, vừa khởi tạo giá trị các thuộc tính của đối tượng lớp theo cú pháp sau:

```
<Tên lớp> <Tên đối tượng>([<Các đối số khởi tạo>]);
```

Trong đó:

- **Tên lớp**: là tên kiểu lớp đã được định nghĩa
- **Tên đối tượng**: là tên biến có kiểu lớp, tuân thủ theo quy tắc đặt tên biến của C++

- **Các đối số khởi tạo**: Là các đối số tương ứng với hàm khởi tạo của lớp tương ứng. Tương tự như việc truyền đối số khi dùng các lời gọi hàm thông thường.

Ví dụ, nếu lớp Car có hàm khởi tạo Car(int, char[], float) thì khi khai báo biến có kiểu lớp Car, ta có thể sử dụng hàm khởi tạo này như sau:

```
Car myCar(100, "Ford", 3000);
```

Lưu ý:

- Khi sử dụng hàm khởi tạo, phải truyền đúng số lượng và đúng kiểu của các tham số của các hàm khởi tạo đã được định nghĩa của lớp.
- Khi một lớp đã có ít nhất một hàm khởi tạo tường minh, thì không được sử dụng hàm khởi tạo ngầm định của C++. Do đó, khi đã khai báo ít nhất một hàm khởi tạo, nếu muốn khai báo biến mà không cần tham số, lớp tương ứng phải có ít nhất một hàm khởi tạo không có tham số.

  Ví dụ, nếu lớp Car chỉ có duy nhất một hàm khởi tạo như sau:

```
class Car{
public:
      Car(int, char[], float);
};
```

  thì không thể khai báo một biến như sau:

```
Car myCar;  // Khai báo lỗi
```

- Trong trường hợp dùng hàm khởi tạo không có tham số, ta không cần phải sử dụng cặp dấu ngoặc đơn "()" sau tên biến đối tượng. Việc khai báo trở thành cách khai báo thông thường.

Chương trình 5.4a minh họa việc định nghĩa và sử dụng lớp Car với hai hàm khởi tạo khác nhau.

Chương trình 5.4a

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;            // Tốc độ
            char  mark[20];         // Nhãn hiệu
            float price;            // Giá xe
      public:
            Car();                  // Khởi tạo không tham số
            Car(int, char[], float);// Khởi tạo đầy đủ tham số
            void  show();           // Giới thiệu xe
};
```

```
/* Khai báo phương thức bên ngoài lớp */
Car::Car(){                          // Khởi tạo không tham số
      speed = 0;
      strcpy(mark, "");
      price = 0;
}

// Khởi tạo có đầy đủ tham số
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

void Car::show(){                    // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

// Hàm main, chương trình chính
void main(){
      clrscr();
      Car myCar1;                    // Sử dụng hàm khởi tạo không tham số
      Car myCar2(150, "Mercedes", 5000);// Dùng hàm khởi tạo đủ tham số

      // Giới thiệu xe thứ nhất
      cout << "Xe thu nhat: " << endl;
      myCar1.show();

      // Giới thiệu xe thứ hai
      cout << "Xe thu hai: " << endl;
      myCar2.show();
      return;
}
```

Chương trình 5.4a sẽ hiển thị các thông tin như sau:

```
Xe thu nhat:
This is a  having a speed of 0km/h and its price is $0
Xe thu hai:
This is a Mercedes having a speed of 150km/h and its price is $5000
```

Lí do là xe thứ nhất sử dụng hàm khởi tạo không có tham số nên xe không có tên, tốc độ và giá đều là mặc định (0). Trong khi đó, xe thứ hai được khởi tạo đầy đủ cả ba tham số nên thông tin giới thiệu xe được đầy đủ.

Tuy nhiên, khi đối tượng có nhiều thuộc tính riêng, để tránh trường hợp phải định nghĩa nhiều hàm khởi tạo cho các trường hợp thiếu vắng một vài tham số khác nhau. Ta có thể sử dụng hàm khởi tạo với các giá trị khởi đầu ngầm định. Chương trình 5.4b cho kết quả hoàn toàn giống chương trình 5.4a, nhưng đơn giản hơn vì chỉ cần định nghĩa một hàm khởi tạo với các tham số có giá trị ngầm định.

Chương trình 5.4b

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;            // Tốc độ
            char  mark[20];         // Nhãn hiệu
            float price;            // Giá xe
      public:
            // Khởi tạo với các giá trị ngầm điịnh cho các tham số
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();           // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

void Car::show(){                   // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

// Hàm main, chương trình chính
void main(){
      clrscr();
      Car myCar1;                   // Các tham số nhận giá trị mặc định
```

```
        Car myCar2(150, "Mercedes", 5000);// Dùng hàm khởi tạo đủ tham số

        // Giới thiệu xe thứ nhất
        cout << "Xe thu nhat: " << endl;
        myCar1.show();

        // Giới thiệu xe thứ hai
        cout << "Xe thu hai: " << endl;
        myCar2.show();
        return;
}
```

5.4.2 Hàm hủy bỏ

Hàm hủy bỏ được tự động gọi đến khi mà đối tượng được giải phóng khỏi bộ nhớ. Nhiệm vụ của hàm hủy bỏ là dọn dẹp bộ nhớ trước khi đối tượng bị giải phóng. Cú pháp khai báo hàm hủy bỏ như sau:

```
class <Tên lớp>{
      public:
            ~<Tên lớp>([<Các tham số>]);  // Khai báo hàm khởi tạo
};
```

Ví dụ:

```
class Car{
            int   speed;
            char  *mark;
            float price;
      public:
            ~Car(){
                  delete [] mark;
            };
};
```

là khai báo một hàm hủy bỏ của lớp Car với thuộc tính mark có kiểu con trỏ. Hàm này sẽ giải phóng vùng nhớ đã cấp phát cho con trỏ kiểu char của thuộc tính nhãn hiệu xe.

Lưu ý:

- Hàm hủy bỏ phải có tên bắt đầu bằng dấu "~", theo sau là tên của lớp tương ứng.
- Hàm hủy bỏ không có giá trị trả về.
- Hàm hủy bỏ phải có tính chất public
- Mỗi lớp chỉ có nhiều nhất một hàm hủy bỏ. Trong trường hợp không khai báo tường minh hàm hủy bỏ, C++ sẽ sử dụng hàm hủy bỏ ngầm định.

- Nói chung, khi có ít nhất một trong các thuộc tính của lớp là con trỏ, thì nên dùng hàm hủy bỏ tường minh để giải phóng triệt để các vùng nhớ của các thuộc tính, trước khi đối tượng bị giải phóng khỏi bộ nhớ.

Chương trình 5.5 minh họa việc định nghĩa lớp Car với một hàm khởi tạo có các tham số với giá trị mặc định và một hàm hủy bỏ tường minh.

Chương trình 5.5

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;            // Tốc độ
            char  *mark;            // Nhãn hiệu
            float price;            // Giá xe
      public:
            // Khởi tạo với các giá trị ngầm định cho các tham số
            Car(int speedIn=0, char *markIn=NULL, float priceIn=0);
            void  show();           // Giới thiệu xe
            ~Car();                 // Hàm hủy bỏ tường minh
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char *markIn, float priceIn){
      speed = speedIn;
      mark = markIn;
      price = priceIn;
}

void Car::show(){                   // Phương thức giới thiệu xe
      cout << "This is a " << *mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

Car::~Car(){                        // Hàm hủy bỏ tường minh
      delete [] mark;
      cout << "The object has been destroyed!" << endl;
}
```

```
// Hàm main, chương trình chính
void main(){
      clrscr();
      Car myCar(150, "Mercedes", 5000);// Dùng hàm khởi tạo đủ tham số

      // Giới thiệu xe
      cout << "Gioi thieu xe: " << endl;
      myCar.show();
      return;
}
```

Chương trình 5.5 sẽ in ra thông báo như sau:

```
Gioi thieu xe:
This is a Mercedes having a speed of 150km/h and its price is $5000
The object has been destroyed!
```

Dòng cuối cùng là của hàm hủy bỏ, mặc dù ta không gọi hàm hủy bỏ trực tiếp, nhưng khi thoát khỏi hàm main() của chương trình chính, đối tượng myCar bị giải phóng khỏi bộ nhớ và khi đó, C++ tự động gọi đến hàm hủy bỏ mà ta đã định nghĩa tường minh. Do vậy, mà có dòng thông báo cuối cùng này.

5.5 CON TRỎ ĐỐI TƯỢNG VÀ MẢNG ĐỐI TƯỢNG

5.5.1 Con trỏ đối tượng

Con trỏ đối tượng là con trỏ trỏ đến địa chỉ của một đối tượng có kiểu lớp. Các thao tác liên quan đến con trỏ đối tượng bao gồm:

- Khai báo con trỏ đối tượng
- Cấp phát bộ nhớ cho con trỏ đối tượng
- Sử dụng con trỏ đối tượng
- Giải phóng bộ nhớ cho con trỏ đối tượng

Khai báo con trỏ đối tượng

Con trỏ đối tượng được khai báo tương tự như khai báo các con trỏ có kiểu thông thường:

```
<Tên lớp> *<Tên con trỏ đối tượng>;
```

Ví dụ, muốn khai báo một con trỏ đối tượng có kiểu của lớp Car, ta khai báo như sau:

```
Car *myCar;
```

Khi đó, myCar là một con trỏ đối tượng có kiểu lớp Car.

Cấp phát bộ nhớ cho con trỏ đối tượng

Con trỏ đối tượng cũng cần phải cấp phát bộ nhớ hoặc trỏ vào một địa chỉ của một đối tượng lớp xác định trước khi được sử dụng. Cấp phát bộ nhớ cho con trỏ đối tượng cũng bằng thao tác **new**:

```
<Tên con trỏ đối tượng> = new <Tên lớp>([<Các đối số>]);
```

Ví dụ, nếu lớp Car có hai hàm khởi tạo như sau:

```
class Car{
public:
      Car();
      Car(int, char[], float);
};
```

thì ta có thể cấp phát bộ nhớ theo hai cách, tương ứng với hai hàm khởi tạo của lớp:

```
myCar = new Car();                    // Khởi tạo không tham số
myCar = new Car(100, "Ford", 3000); // Khởi tạo đủ tham số
```

Lưu ý:

- Các đối số truyền phải tương ứng với ít nhất một trong các hàm khởi tạo của lớp.
- Khi sử dụng hàm khởi tạo không có tham số, ta vẫn phải sử dụng cặp ngoặc đơn "()" trong thao tác **new**.
- Khi lớp không có một hàm khởi tạo tường minh nào, sẽ dùng hàm khởi tạo ngầm định của C++ và cú pháp tương tự như sử dụng hàm khởi tạo tường minh không có tham số.
- Có thể vừa khai báo, vừa cấp phát bộ nhớ cho con trỏ đối tượng.

  Ví dụ:

```
      Car myCar = new Car();          // Khởi tạo không tham số
```

Sử dụng con trỏ đối tượng

Con trỏ đối tượng được sử dụng qua các thao tác:

- Trỏ đến địa chỉ của một đối tượng cùng lớp
- Truy nhập đến các phương thức của lớp

Con trỏ đối tượng có thể trỏ đến địa chỉ của một đối tượng có sẵn, cùng lớp theo cú pháp sau:

```
<Tên con trỏ đối tượng> = &<Tên đối tượng có sẵn>;
```

Ví dụ, ta có một con trỏ và một đối tượng của lớp Car:

```
Car *ptrCar, myCar(100, "Ford",3000);
```

Khi đó, có thể cho con trỏ ptrCar trỏ vào địa chỉ của đối tượng myCar như sau:

```
ptrCar = &myCar;
```

Khi muốn truy nhập đến các thành phần của con trỏ đối tượng, ta dùng cú pháp sau:

```
<Tên con trỏ đối tượng> -> <Tên thành phần lớp>([<Các đối số>]);
```

Ví dụ, đoạn chương trình sau sẽ thực hiện phương thức giới thiệu xe của lớp Car thông qua con trỏ ptrCar:

```
Car *ptrCar = new Car(100, "Ford",3000);
ptrCar->show();
```

Lưu ý:

- Danh sách các đối số phải tương thích với tên phương thức tương ứng.
- Các quy tắc phạm vi truy nhập vẫn áp dụng trong truy nhập các thành phần lớp thông qua con trỏ.

Giải phóng bộ nhớ cho con trỏ đối tượng

Con trỏ đối tượng cũng được giải phóng thông qua thao tác **delete**:

```
delete <Tên con trỏ đối tượng>;
```

Ví dụ:

```
Car *ptrCar = new Car();// Khai báo và cấp phát bộ nhớ
…                       // Sử dụng con trỏ ptrCar
delete ptrCar;          // Giải phóng bộ nhớ.
```

Lưu ý:

- Thao tác **delete** chỉ được dùng khi trước đó, con trỏ được cấp phát bộ nhớ qua thao tác **new**:

  ```
  Car *ptrCar = new Car();
  delete ptrCar;          // Đúng.
  ```

  Nhưng không được dùng **delete** khi trước đó, con trỏ chỉ trỏ vào một địa chỉ của đối tượng có sẵn (tĩnh):

  ```
  Car *ptrCar, myCar(100, "Ford", 3000);
  ptrCar = &myCar;
  delete ptrCar;          // Không được
  ```

Chương trình 5.6 minh họa việc dùng con trỏ đối tượng có kiểu lớp là Car.

Chương trình 5.6

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp */
class Car{
      private:
            int   speed;          // Tốc độ
            char  mark[20];       // Nhãn hiệu
            float price;          // Giá xe
      public:
            // Khởi tạo với các giá trị ngầm điịnh cho các tham số
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();         // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
```

```
}

void Car::show(){                    // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

// Hàm main, chương trình chính
void main(){
      clrscr();
      // Khai báo con trỏ, cấp phát bộ nhớ dùng hàm khởi tạo đủ tham số
      Car *myCar = new Car(150, "Mercedes", 5000);

      // Giới thiệu xe
      cout << "Gioi thieu xe: " << endl;
      myCar->show();

      // Giải phóng con trỏ
      delete myCar;
      return;
}
```

Chương trình 5.6 hiển thị ra thông báo là một lời giới thiệu xe;

```
Gioi thieu xe:
This is a Mercedes having a speed of 150km/h and its price is $5000
```

5.5.2 Mảng các đối tượng

Mảng các đối tượng cũng có thể được khai báo và sử dụng như mảng của các biến có kiểu thông thường.

Khai báo mảng tĩnh các đối tượng

Mảng các đối tượng được khai báo theo cú pháp:

```
<Tên lớp> <Tên biến mảng>[<Số lượng đối tượng>];
```

Ví dụ:

```
Car cars[10];
```

Là khai báo một mảng có 10 đối tượng có cùng kiểu lớp Car.

Lưu ý:

- Có thể khai báo mảng tĩnh các đối tượng mà chưa cần khai báo độ dài mảng, cách này thường dùng khi chưa biết chính xác độ dài mảng:

```
Car cars[];
```

- Muốn khai báo được mảng tĩnh các đối tượng, lớp tương ứng phải có hàm khởi tạo không có tham số. Vì khi khai báo mảng, tương đương với khai báo một dãy các đối tượng với hàm khởi tạo không có tham số.

Khai báo mảng động với con trỏ

Một mảng các đối tượng cũng có thể được khai báo và cấp phát động thông qua con trỏ đối tượng như sau:

```
<Tên lớp> *<Tên biến mảng động> = new <Tên lớp>[<Độ dài mảng>];
```

Ví dụ:

```
Car *cars = new Car[10];
```

Sau khi được sử dụng, mảng động các đối tượng cũng cần phải giải phóng bộ nhớ:

```
delete [] <Tên biến mảng động>;
```

Ví dụ:

```
Car *cars = new Car[10];// Khai báo và cấp phát động
…                       // Sử dụng biến mảng động
delete [] cars;         // Giải phóng bộ nhớ của mảng động
```

Sử dụng mảng đối tượng

Khi truy nhập vào các thành phần của một đối tượng có chỉ số xác định trong mảng đã khai báo, ta có thể sử dụng cú pháp:

```
<Tên biến mảng>[<Chỉ số đối tượng>].<Tên thành phần>([<Các đối số>]);
```

Ví dụ:

```
Car cars[10];
cars[5].show();
```

sẽ thực hiện phương thức show() của đối tượng có chỉ số thứ 5 (tính từ chỉ số 0) trong mảng cars.

Chương trình 5.7 sẽ cài đặt một chương trình, trong đó nhập vào độ dài mảng, sau đó yêu cầu người dùng nhập thông tin về mảng các xe. Cuối cùng, chương trình sẽ tìm kiếm và hiển thị thông tin về chiếc xe có giá đắt nhất trong mảng.

Chương trình 5.7

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            void  setSpeed(int);          // Gán tốc độ cho xe
```

```
            int   getSpeed();              // Đọc tốc độ xe
            void  setMark(char);           // Gán nhãn cho xe
            char[] getMark();              // Đọc nhãn xe
            void  setPrice(float);         // Gán giá cho xe
            float getPrice();              // Đọc giá xe
            // Khởi tạo thông tin về xe
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();                  // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

void Car::setSpeed(int speedIn){     // Gán tốc độ cho xe
      speed = speedIn;
}
int Car::getSpeed(){                 // Đọc tốc độ xe
      return speed;
}
void Car::setMark(char markIn){      // Gán nhãn cho xe
      strcpy(mark, markIn);
}
char[] Car::getMark(){               // Đọc nhãn xe
      return mark;
}
void Car::setPrice(float priceIn){   // Gán giá cho xe
      price = priceIn;
}
float Car::getPrice(){               // Đọc giá xe
      return price;
}

void Car::show(){                    // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

// Hàm main, chương trình chính
```

```
void main(){
      clrscr();
      int length;                    // Chiều dài mảng
      float maxPrice = 0;            // Giá đắt nhất
      int index = 0;                 // Chỉ số của xe đắt nhất
      Car *cars;                     // Khai báo mảng đối tượng

      // Nhập số lượng xe, tức là chiều dài mảng
      cout << "So luong xe: ";
      cin >> length;

      // Cấp phát bộ nhớ động cho mảng
      cars = new Car[length];

      // Khởi tạo các đối tượng trong mảng
      for(int i=0;i<length; i++){
            int speed;               // (Biến tạm) tốc độ
            char mark[20];           // (Biến tạm) nhãn hiệu
            float price;             // (Biến tạm) giá xe
            cout << "Xe thu " << i << ": " <<endl;
            cout << "Toc do (km/h): ";
            cin >> speed; cars[i].setSpeed(speed);    // Nhập tốc độ
            cout << "Nhan hieu : ";
            cin >> mark; cars[i].setMark(mark);       // Nhập nhãn xe
            cout << "Gia ($): ";
            cin >> price; cars[i].setPrice(price);    // Nhập giá xe

            if(maxPrice < price){
                  maxPrice = price;
                  index = i;
            }
      }

      // Tìm xe đắt nhất
      for(int i=0; i<length; i++)
            if(i == index){
                  cars[i].show();    // Giới thiệu xe đắt nhất
                  break;
            }

      // Giải phóng bộ nhớ của mảng
      delete [] cars;
      return;
```

```
}
```

TỔNG KẾT CHƯƠNG 5

Nội dung chương 5 đã tập trung trình bày các vấn đề cơ bản về lớp đối tượng trong ngôn ngữ C++:

- Khai báo, định nghĩa lớp bằng từ khóa class
- Sử dụng biến lớp như những đối tượng cụ thể
- Khai báo, định nghĩa các thuộc tính và các phương thức của lớp: định nghĩa các phương thức trong hoặc ngoài phạm vi khai báo lớp
- Khai báo phạm vi truy nhập lớp bằng các từ khóa chỉ phạm vi: private, protected và public. Giới thiệu các kiểu hàm có thể truy nhập phạm vi bất quy tắc: hàm bạn và lớp bạn với từ khóa friend.
- Khai báo, định nghĩa tường minh hàm khởi tạo của lớp và sử dụng khi khai báo biến lớp
- Khai báo, định nghĩa tường minh hàm hủy bỏ của lớp, nhằm dọn dẹp, giải phóng bộ nhớ trước khi đối tượng bị giải phóng khỏi bộ nhớ.
- Khai báo, cấp phát bộ nhớ, sử dụng và giải phóng bộ nhớ cho con trỏ đối tượng
- Khai báo, cấp phát bộ nhớ động, sử dụng và giải phóng vùng nhớ của mảng các đối tượng.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5

1. Trong các khai báo lớp sau, những khai báo nào là đúng:

    a. `class MyClass;`

    b. `Class MyClass;`

    c. `class MyClass{…};`

    d. `Class MyClass{…};`

2. Giả sử ta đã định nghĩa lớp MyClass, bây giờ ta khai báo một đối tượng thuộc kiểu lớp này. Khai báo nào là đúng:

    a. `class MyClass me;`

    b. `MyClass me;`

    c. `MyClass me();`

    d. `MyClass me{};`

3. Trong các khai báo thuộc tính ngay trong phạm vi của khai báo lớp như sau, những khai báo nào là đúng:

    a. `int myAge;`

    b. `private int myAge;`

    c. `public int myAge;`

    d. `private: int myAge;`

4. Trong các khai báo phương thức ngay trong phạm vi của khai báo lớp MyClass như sau, những khai báo nào là đúng:

   a. `public: void show();`

   b. `void show();`

   c. `void show(){cout << "hello!";};`

   d. `void MyClass::show(){cout << "hello!";}`

5. Xét đoạn chương trình sau:

```
class MyClass{
      int age;
      public:
            int getAge();
};
MyClass me;
```

   Khi đó, trong các lệnh sau, lệnh nào có thể thêm vào cuối đoạn chương trình trên:

   a. `cout << MyClass::age;`

   b. `cout << me.age;`

   c. `cout << me.getAge();`

   d. `cin >> me.age;`

6. Trong các khai báo hàm khởi tạo cho lớp MyClass như sau, những khai báo nào là đúng:

   a. `myClass();`

   b. `MyClass();`

   c. `MyClass(MyClass);`

   d. `void MyClas();`

   e. `MyClass MyClass();`

7. Trong các khai báo hàm hủy bỏ cho lớp MyClass như sau, những khai báo nào là đúng:

   a. `~myClass();`

   b. `~MyClass();`

   c. `~MyClass(MyClass);`

   d. `void ~MyClas();`

   e. `MyClass ~MyClass();`

8. Giả sử ta khai báo lớp MyClass có một thuộc tính age và một hàm khởi tạo:

```
class MyClass{
      int age;
      public:
            MyClass(MyClass me){
                  …
            };
            int getAge(){return age};
};
```

Trong các dòng lệnh sau, những dòng nào có thể xuất hiện trong hàm khởi tạo trên:

a. `age = MyClass.age;`

b. `age = me.age;`

c. `age = MyClass.getAge();`

d. `age = me.getAge();`

e. `age = 10;`

9. Xét khai báo lớp như sau:

```
class MyClass{
      public:
            MyClass(MyClass);
            MyClass(int);
};
```

Khi đó, trong số các khai báo đối tượng sau, những khai báo nào là đúng:

a. `MyClass me;`

b. `MyClass me();`

c. `MyClass me(10);`

d. `MyClass me1(10), me2(me1);`

10. Xét khai báo lớp như sau:

```
class MyClass{
      public:
            MyClass();
};
```

Khi đó, trong số các khai báo con trỏ đối tượng sau, những khai báo nào là đúng:

a. `MyClass *me;`

b. `MyClass *me();`

c. `MyClass *me = new me();`

d. `MyClass *me = new MyClass();`

11. Xét khai báo lớp như sau:

```
class MyClass{
      public:
            MyClass(int i=0, int j=0, float k=0);
};
```

Khi đó, trong số các khai báo con trỏ đối tượng sau, những khai báo nào là đúng:

a. `MyClass *me;`

b. `MyClass *me = new MyClass();`

c. `MyClass *me = new MyClass(1, 20);`

d. `MyClass *me = new MyClass(1, 20, 30);`

12. Khai báo một lớp nhân viên, có tên là Employee, với ba thuộc tính có tính chất private:

- Tên nhân viên, có dạng một con trỏ kiểu char

- Tuổi nhân viên, có kiểu int
- Lương nhân viên, có kiểu float.

13. Thêm vào lớp Employee trong bài 12 các phương thức có tính chất public:
    - set/get giá trị thuộc tính tên nhân viên
    - set/get giá trị thuộc tính tuổi nhân viên
    - set/get giá trị thuộc tính lương nhân viên.
14. Viết một hàm khởi tạo không có tham số cho lớp Employee trong bài 13, các thuộc tính của lớp nhận giá trị mặc định:
    - giá trị thuộc tính tên nhân viên, mặc định là chuỗi kí tự rỗng “”
    - giá trị thuộc tính tuổi nhân viên, mặc định là 18
    - giá trị thuộc tính lương nhân viên, mặc định là $100.
15. Viết thêm một hàm khởi tạo với đầy đủ ba tham số tương ứng với ba thuộc tính của lớp Employee trong bài 14.
16. Thêm vào lớp Employee một phương thức show() để giới thiệu về tên, tuổi và lương của đối tượng nhân viên.
17. Viết một hàm hủy bỏ tường minh cho lớp Employee nhằm giải phóng vùng nhớ của con trỏ char, là kiểu của thuộc tính tên nhân viên.
18. Viết một chương trình sử dụng lớp Employee được xây dựng sau bài 17 với các thao tác sau:
    - Khai báo một đối tượng kiểu Employee, dùng hàm khởi tạo không tham số
    - Dùng hàm show() để giới thiệu về đối tượng đó
19. Viết một chương trình sử dụng lớp Employee được xây dựng sau bài 17 với các thao tác sau:
    - Khai báo một đối tượng kiểu Employee, dùng hàm khởi tạo không tham số
    - Nhập từ bàn phím giá trị các thuộc tính tên, tuổi, lương nhân viên.
    - Gán các giá trị này cho các thuộc tính của đối tượng đã khai báo, dùng các hàm set
    - Dùng hàm show() để giới thiệu về đối tượng đó
20. Viết một chương trình sử dụng lớp Employee được xây dựng sau bài 17 với các thao tác sau:
    - Khai báo một đối tượng kiểu Employee, dùng hàm khởi tạo với đủ 3 tham số
    - Dùng hàm show() để giới thiệu về đối tượng đó
21. Viết một chương trình sử dụng lớp Employee được xây dựng sau bài 17 với các thao tác sau:
    - Khai báo một con trỏ đối tượng kiểu Employee
    - Cấp phát bộ nhớ, dùng hàm khởi tạo với đủ 3 tham số
    - Dùng hàm show() để giới thiệu về đối tượng mà con trỏ này đang trỏ tới
22. Viết một chương trình nhập dữ liệu cho một mảng động các đối tượng của lớp Employee trong bài 17. Chiều dài mảng động cũng được nhập từ bàn phím.

23. Viết một chương trình tìm kiếm trên mảng động đã được xây dựng trong bài 22: tìm kiếm và giới thiệu về nhân viên trẻ nhất và nhân viên già nhất trong mảng đó.
24. Viết một chương trình tìm kiếm trên mảng động đã được xây dựng trong bài 22: tìm kiếm và giới thiệu về nhân viên có lương cao nhất và nhân viên có lương thấp nhất trong mảng đó.
25. Viết một chương trình tìm kiếm trên mảng động đã được xây dựng trong bài 22: tìm kiếm và giới thiệu về nhân viên có tên xác định, do người dùng nhập từ bàn phím.

CHƯƠNG 6

TÍNH KẾ THỪA VÀ ĐA HÌNH

Nội dung chương này tập trung trình bày các vấn đề liên quan đến tính kế thừa và tương ứng bội (đa hình) trong ngôn ngữ C++:

- Khái niệm kế thừa, dẫn xuất và các kiểu dẫn xuất
- Khai báo, định nghĩa các hàm khởi tạo và hàm hủy bỏ trong lớp dẫn xuất
- Truy nhập tới các thành phần của lớp cơ sở và các lớp dẫn xuất
- Việc một lớp được kế thừa từ nhiều lớp cơ sở khác nhau
- Khai báo và sử dụng các lớp cơ sở trừu tượng trong kế thừa
- Tính đa hình trong kế thừa

6.1 KHÁI NIỆM KẾ THỪA

Lập trình hướng đối tượng có hai đặc trưng cơ bản:

- Đóng gói dữ liệu, được thể hiện bằng cách dùng khái niệm lớp để biểu diễn đối tượng với các thuộc tính private, chỉ cho phép bên ngoài truy nhập vào thông qua các phương thức get/set.
- Dùng lại mã, thể hiện bằng việc thừa kế giữa các lớp. Việc thừa kế cho phép các lớp thừa kế (gọi là lớp dẫn xuất) sử dụng lại các phương thức đã được định nghĩa trong các lớp gốc (gọi là lớp cơ sở).

6.1.1 Khai báo thừa kế

Cú pháp khai báo một lớp kế thừa từ một lớp khác như sau:

```
class <Tên lớp dẫn xuất>: <Từ khóa dẫn xuất> <Tên lớp cơ sở>{
      …     // Khai báo các thành phần lớp
};
```

Trong đó:

- **Tên lớp dẫn xuất**: là tên lớp được cho kế thừa từ lớp khác. Tên lớp này tuân thủ theo quy tắc đặt tên biến trong C++.
- **Tên lớp cở sở**: là tên lớp đã được định nghĩa trước đó để cho lớp khác kế thừa. Tên lớp này cũng tuân thủ theo quy tắc đặt tên biến của C++.
- **Từ khóa dẫn xuất**: là từ khóa quy định tính chất của sự kế thừa. Có ba từ khóa dẫn xuất là private, protected và public. Mục tiếp theo sẽ trình bày ý nghĩa của các từ khóa dẫn xuất này.

Ví dụ:

```
class Bus: public Car{
      …     // Khai báo các thành phần
};
```

là khai báo một lớp Bus (xe buýt) kế thừa từ lớp Car (xe ô tô) với tính chất kế thừa là public.

6.1.2 Tính chất dẫn xuất

Sự kế thừa cho phép trong lớp dẫn xuất có thể sử dụng lại một số mã nguồn của các phương thức và thuộc tính đã được định nghĩa trong lớp cơ sở. Nghĩa là lớp dẫn xuất có thể truy nhập trực tiếp đến một số thành phần của lớp cơ sở. Tuy nhiên, phạm vi truy nhập từ lớp dẫn xuất đến lớp cơ sở không phải bao giờ cũng giống nhau: chúng được quy định bởi các từ khóa dẫn xuất private, protected và public.

Dẫn xuất private

Dẫn xuất private quy định phạm vi truy nhập như sau:

- Các thành phần private của lớp cơ sở thì không thể truy nhập được từ lớp dẫn xuất.
- Các thành phần protected của lớp cơ sở trở thành các thành phần private của lớp dẫn xuất
- Các thành phần public của lớp cơ sở cũng trở thành các thành phần private của lớp dẫn xuất.
- Phạm vi truy nhập từ bên ngoài vào lớp dẫn xuất được tuân thủ như quy tắc phạm vi lớp thông thường.

Dẫn xuất protected

Dẫn xuất protected quy định phạm vi truy nhập như sau:

- Các thành phần private của lớp cơ sở thì không thể truy nhập được từ lớp dẫn xuất.
- Các thành phần protected của lớp cơ sở trở thành các thành phần protected của lớp dẫn xuất
- Các thành phần public của lớp cơ sở cũng trở thành các thành phần protected của lớp dẫn xuất.
- Phạm vi truy nhập từ bên ngoài vào lớp dẫn xuất được tuân thủ như quy tắc phạm vi lớp thông thường.

Dẫn xuất public

Dẫn xuất public quy định phạm vi truy nhập như sau:

- Các thành phần private của lớp cơ sở thì không thể truy nhập được từ lớp dẫn xuất.
- Các thành phần protected của lớp cơ sở trở thành các thành phần protected của lớp dẫn xuất.
- Các thành phần public của lớp cơ sở vẫn là các thành phần public của lớp dẫn xuất.
- Phạm vi truy nhập từ bên ngoài vào lớp dẫn xuất được tuân thủ như quy tắc phạm vi lớp thông thường.

Bảng 6.1 tóm tắt lại các quy tắc truy nhập được quy định bởi các từ khóa dẫn xuất.

Kiểu dẫn xuất	*Tính chất ở lớp cơ sở*	*Tính chất ở lớp dẫn xuất*
private	**private**	**Không truy nhập được**

	protected **public**	**private** **private**
protected	**private** **protected** **public**	**Không truy nhập được** **protected** **protected**
public	**private** **protected** **public**	**Không truy nhập được** **protected** **public**

6.2 HÀM KHỞI TẠO VÀ HUỶ BỎ TRONG KẾ THỪA

6.2.1 Hàm khởi tạo trong kế thừa

Khi khai báo một đối tượng có kiểu lớp được dẫn xuất từ một lớp cơ sở khác. Chương trình sẽ tự động gọi tới hàm khởi tạo của lớp dẫn xuất. Tuy nhiên, thứ tự được gọi sẽ bắt đầu từ hàm khởi tạo tương ứng của lớp cơ sở, sau đó đến hàm khởi tạo của lớp dẫn xuất. Do đó, thông thường, trong hàm khởi tạo của lớp dẫn xuất phải có hàm khởi tạo của lớp cơ sở.

Cú pháp khai báo hàm khởi tạo như sau:

```
<Tên hàm khởi tạo dẫn xuất>([<Các tham số>]):
      <Tên hàm khởi tạo cơ sở>([<Các đối số>]){
      …     // Khởi tạo các thuộc tính mới bổ sung của lớp dẫn xuất
};
```

Vì tên hàm khởi tạo là trùng với tên lớp, nên có thể viết lại thành:

```
<Tên lớp dẫn xuất>([<Các tham số>]):
      <Tên lớp cơ sở>([<Các đối số>]){
      …     // Khởi tạo các thuộc tính mới bổ sung của lớp dẫn xuất
};
```

Ví dụ:

```
Bus():Car(){
      …     // Khởi tạo các thuộc tính mới bổ sung của lớp Bus
}
```

là một định nghĩa một hàm khởi tạo của lớp Bus kế thừa từ lớp Car. Định nghĩa này được thực hiện trong phạm vi khai báo lớp Bus. Đây là một hàm khởi tạo không tham số, nó gọi tới hàm khởi tạo không tham số của lớp Car.

Lưu ý:

- Nếu định nghĩa hàm khởi tạo bên ngoài phạm vi lớp thì phải thêm tên lớp dẫn xuất và toán tử phạm vi "::" trước tên hàm khởi tạo.
- Giữa tên hàm khởi tạo của lớp dẫn xuất và hàm khởi tạo của lớp cơ sở, chỉ có môt dấu hai chấm ":", nếu là hai dấu "::" thì trở thành toán tử phạm vi lớp.

- Nếu không chỉ rõ hàm khởi tạo của lớp cơ sở sau dấu hai chấm ":" chương trình sẽ tự động gọi hàm khởi tạo ngầm định hoặc hàm khởi tạo không có tham số của lớp cơ sở nếu hàm đó được định nghĩa tường minh trong lớp cơ sở.

  Ví dụ, định nghĩa hàm khởi tạo:

```
Bus():Car(){
      …     // Khởi tạo các thuộc tính mới bổ sung của lớp Bus
};
```

  Có thể thay bằng:

```
Bus(){      // Gọi hàm khởi tạo không tham số của lớp Car
      …     // Khởi tạo các thuộc tính mới bổ sung của lớp Bus
};
```

Chương trình 6.1 định nghĩa lớp Car có 3 thuộc tính với hai hàm khởi tạo, sau đó định nghĩa lớp Bus có thêm thuộc tính label là số hiệu của tuyến xe buýt. Lớp Bus sẽ được cài đặt hai hàm khởi tạo tường minh, gọi đến hai hàm khởi tạo tương ứng của lớp Car.

Chương trình 6.1

```
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            Car();                        // Khởi tạo không tham số
            Car(int, char[], float);      // Khởi tạo đủ tham số
};

Car::Car(){                               // Khởi tạo không tham số
      speed = 0;
      strcpy(mark, "");
      price = 0;
}

// Khởi tạo đủ tham số
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car */
```

```
class Bus: public Car{
            int label;                      // Số hiệu tuyến xe
      public:
            Bus();                          // Khởi tạo không tham số
            Bus(int, char[], float, int); // Khởi tạo đủ tham số
};

Bus::Bus():Car(){                           // Khởi tạo không tham số
      label = 0;
}

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, int lIn):Car(sIn, mIn, pIn){
      label = lIn;
}
```

Trong hàm khởi tạo của lớp Bus, muốn khởi tạo các thuộc tính của lớp Car, ta phải khởi tạo gián tiếp thông qua hàm khởi tạo của lớp Car mà không thể gán giá trị trực tiếp cho các thuộc tính speed, mark và price. Lí do là các thuộc tính này có tính chất private, nên lớp dẫn xuất không thể truy nhập trực tiếp đến chúng.

6.2.2 Hàm hủy bỏ trong kế thừa

Khi một đối tượng lớp dẫn xuất bị giải phóng khỏi bộ nhớ, thứ tự gọi các hàm hủy bỏ ngược với thứ tự gọi hàm thiết lập: gọi hàm hủy bỏ của lớp dẫn xuất trước khi gọi hàm hủy bỏ của lớp cơ sở.

Vì mỗi lớp chỉ có nhiều nhất là một hàm hủy bỏ, nên ta không cần phải chỉ ra hàm hủy bỏ nào của lớp cơ sở sẽ được gọi sau khi hủy bỏ lớp dẫn xuất. Do vậy, hàm hủy bỏ trong lớp dẫn xuất được khai báo và định nghĩa hoàn toàn giống với các lớp thông thường:

```
<Tên lớp>::~<Tên lớp>([<Các tham số>]){
      … // giải phóng phần bộ nhớ cấp phát cho các thuộc tính bổ sung
}
```

Lưu ý:

- Hàm hủy bỏ của lớp dẫn xuất chỉ giải phóng phần bộ nhớ được cấp phát động cho các thuộc tính mới bổ sung trong lớp dẫn xuất, nếu có, mà không được giải phóng bộ nhớ được cấp cho các thuộc tính trong lớp cơ sở (phần này là do hàm hủy bỏ của lớp cơ sở đảm nhiệm).
- Không phải gọi tường minh hàm hủy bỏ của lớp cơ sở trong hàm hủy bỏ của lớp dẫn xuất.
- Ngay cả khi lớp dẫn xuất không định nghĩa tường minh hàm hủy bỏ (do không cần thiết) mà lớp cơ sở lại có định nghĩa tường minh. Chương trình vẫn gọi hàm hủy bỏ ngầm định của lớp dẫn xuất, sau đó vẫn gọi hàm hủy bỏ tường minh của lớp cơ sở.

Chương trình 6.2 cài đặt lớp Bus kế thừa từ lớp Car: lớp Car có một thuộc tính có dạng con trỏ nên cần giải phóng bằng hàm hủy bỏ tường minh. Lớp Bus có thêm một thuộc tính có dạng con

trỏ là danh sách các đường phố mà xe buýt đi qua (mảng động các chuỗi kí tự *char[]) nên cũng cần giải phóng bằng hàm hủy bỏ tường minh.

Chương trình 6.2

```
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
            char  *mark;                    // Nhãn hiệu xe
      public:
            ~Car();                         // Hủy bỏ tường minh
};

Car::~Car(){                                // Hủy bỏ tường minh
      delete [] mark;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car */
class Bus: public Car{
            char *voyage[];                 // Hành trình tuyến xe
      public:
            ~Bus();                         // Hủy bỏ tường minh
};

Bus::~Bus(){                                // Hủy bỏ tường minh
      delete [] voyage;
}
```

Trong hàm hủy bỏ của lớp Bus, ta chỉ được giải phóng vùng nhớ được cấp phát cho thuộc tính voyage (hành trình của xe buýt), là thuộc tính được bổ sung thêm của lớp Bus. Mà không được giải phóng vùng nhớ cấp phát cho thuộc tính mark (nhãn hiệu xe), việc này là thuộc trách nhiệm của hàm hủy bỏ của lớp Car vì thuộc tính mark được khai báo trong lớp Car.

6.3 TRUY NHẬP TỚI CÁC THÀNH PHẦN TRONG KẾ THỪA LỚP

6.3.1 Phạm vi truy nhập

Mối quan hệ giữa các thành phần của lớp cơ sở và lớp dẫn xuất được quy định bởi các từ khóa dẫn xuất, như đã trình bày trong mục 6.1.2, được tóm tắt trong bảng 6.2

Kiểu dẫn xuất	*Tính chất ở lớp cơ sở*	*Tính chất ở lớp dẫn xuất*
private	**private** **protected**	**Không truy nhập được** **private**

	public	**private**
protected	**private** **protected** **public**	**Không truy nhập được** **protected** **protected**
public	**private** **protected** **public**	**Không truy nhập được** **protected** **public**

Ta xét phạm vi truy nhập theo hai loại:

- Phạm vi truy nhập từ các hàm bạn, lớp bạn của lớp dẫn xuất
- Phạm vi truy nhập từ các đối tượng có kiểu lớp dẫn xuất

Truy nhập từ các hàm bạn và lớp bạn của lớp dẫn xuất

Nhìn vào bảng tổng kết 6.2, phạm vi truy nhập của hàm bạn, lớp bạn của lớp dẫn xuất vào lớp cơ sở như sau:

- Với dẫn xuất private, hàm bạn có thể truy nhập được các thành phần protected và public của lớp cơ sở vì chúng trở thành các thành phần private của lớp dẫn xuất, có thể truy nhập được từ hàm bạn.
- Với dẫn xuất protected, hàm bạn cũng có thể truy nhập được các thành phần protected và public của lớp cơ sở vì chúng trở thành các thành phần protected của lớp dẫn xuất, có thể truy nhập được từ hàm bạn.
- Với dẫn xuất public, hàm bạn cũng có thể truy nhập được các thành phần protected và public của lớp cơ sở vì chúng trở thành các thành phần protected và public của lớp dẫn xuất, có thể truy nhập được từ hàm bạn.
- Đối với cả ba loại dẫn xuất, hàm bạn đều không truy nhập được các thành phần private của lớp cơ sở, vì các thành phần này cũng không truy nhập được từ lớp dẫn xuất.

Truy nhập từ các đối tượng tạo bởi lớp dẫn xuất

Nhìn vào bảng tổng kết 6.2, phạm vi truy nhập của các đối tượng của lớp dẫn xuất vào lớp cơ sở như sau:

- Với dẫn xuất private, đối tượng của lớp dẫn xuất không truy nhập được bất cứ thành phần nào của lớp cơ sở vì chúng trở thành các thành phần private của lớp dẫn xuất, không truy nhập được từ bên ngoài.
- Với dẫn xuất protected, đối tượng của lớp dẫn xuất không truy nhập được bất cứ thành phần nào của lớp cơ sở vì chúng trở thành các thành phần protected của lớp dẫn xuất, không truy nhập được từ bên ngoài.
- Với dẫn xuất public, đối tượng của lớp dẫn xuất có thể truy nhập được các thành phần public của lớp cơ sở vì chúng trở thành các thành phần public của lớp dẫn xuất, có thể truy nhập được từ bên ngoài.

Bảng 6.3 tổng kết phạm vi truy nhập từ hàm bạn và đối tượng của lớp dẫn xuất vào các thành phần của lớp cơ sở, được quy định bởi các từ khóa dẫn xuất.

Kiểu dẫn xuất	*Tính chất ở lớp cơ sở*	*Tính chất ở lớp dẫn xuất*	*Truy nhập từ hàm bạn của lớp dẫn xuất*	*Truy nhập từ đối tượng của lớp dẫn xuất*
private	**private** **protected** **public**	**---** **private** **private**	**---** **ok** **ok**	**---** **---** **---**
protected	**private** **protected** **public**	**---** **protected** **protected**	**---** **ok** **ok**	**---** **---** **---**
public	**private** **protected** **public**	**---** **protected** **public**	**---** **ok** **ok**	**---** **---** **ok**

6.3.2 Sử dụng các thành phần của lớp cơ sở từ lớp dẫn xuất

Từ bảng tổng kết phạm vi truy nhập, ta thấy rằng chỉ có dẫn xuất theo kiểu public thì đối tượng của lớp dẫn xuất mới có thể truy nhập đến các thành phần (thuộc loại public) của lớp cơ sở. Khi đó, việc gọi đến các thành phần của lớp cơ sở cũng tương tự như gọi các thành phần lớp thông thường:

- Đối với biến đối tượng thông thường:

```
<Tên đối tượng>.<Tên thành phần>([Các đối số]);
```

- Đối với con trỏ đối tượng:

```
<Tên đối tượng>-><Tên thành phần>([Các đối số]);
```

Lưu ý:

- Cách gọi hàm thành phần này được áp dụng khi trong lớp dẫn xuất, ta không định nghĩa lại các hàm thành phần của lớp cơ sở. Trường hợp định nghĩa lại hàm thành phần của lớp cơ sở sẽ được trình bày trong mục 6.3.3.

Chương trình 6.3 minh họa việc sử dụng các thành phần lớp cơ sở từ đối tượng lớp dẫn xuất: lớp Bus kế thừa từ lớp Car. Lớp Bus có định nghĩa bổ sung một số phương thức và thuộc tính mới. Khi đó, đối tượng của lớp Bus có thể gọi các hàm public của lớp Bus cũng như của lớp Car.

Chương trình 6.3

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
```

```
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            void  setSpeed(int);          // Gán tốc độ cho xe
            int   getSpeed();             // Đọc tốc độ xe
            void  setMark(char);          // Gán nhãn cho xe
            char[] getMark();             // Đọc nhãn xe
            void  setPrice(float);        // Gán giá cho xe
            float getPrice();             // Đọc giá xe
            // Khởi tạo thông tin về xe
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();                 // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

void Car::setSpeed(int speedIn){    // Gán tốc độ cho xe
      speed = speedIn;
}
int Car::getSpeed(){                // Đọc tốc độ xe
      return speed;
}
void Car::setMark(char markIn){     // Gán nhãn cho xe
      strcpy(mark, markIn);
}
char[] Car::getMark(){              // Đọc nhãn xe
      return mark;
}
void Car::setPrice(float priceIn){  // Gán giá cho xe
      price = priceIn;
}
float Car::getPrice(){              // Đọc giá xe
      return price;
}

void Car::show(){                   // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
```

```
      return;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car */
class Bus: public Car{
            int label;              // Số hiệu tuyến xe
      public:
            // Khởi tạo đủ tham số
            Bus(int sIn=0, char mIn[]="", float pIn=0, int lIn=0);
            void setLabel(int);     // Gán số hiệu tuyến xe
            int getLabel();         // Đọc số hiệu tuyến xe
};

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, int lIn):Car(sIn, mIn, pIn){
      label = lIn;
}
void Bus::setLabel(int labelIn){    // Gán số hiệu tuyến xe
      label = labelIn;
}
int Bus::getLabel(){                // Đọc số hiệu tuyến xe
      return label;
}

// Chương trình chính
void main(){
      clrscr();
      Bus myBus;                    // Biến đối tượng của lớp Bus
      int speedIn, labelIn;
      float priceIn;
      char markIn[20];

      // Nhập giá trị cho các thuộc tính
      cout << "Toc do xe bus:";
      cin >> speedIn;
      cout << "Nhan hieu xe bus:";
      cin >> markIn;
      cout << "Gia xe bus:";
      cin >> priceIn;
      cout << "So hieu tuyen xe bus:";
      cin >> labelIn;

      myBus.setSpeed(speedIn);      // Phương thức của lớp Car
```

```
        myBus.setMark(markIn);          // Phương thức của lớp Car
        myBus.setPrice(priceIn);        // Phương thức của lớp Car
        myBus.setLabel(labelIn);        // Phương thức của lớp Bus
        myBus.show();                   // Phương thức của lớp Car
        return;
}
```

Trong chương trình 6.3, đối tượng myBus có kiểu lớp Bus, là lớp dẫn xuất của lớp cơ sở Car, có thể sử dụng các phương thức của lớp Car và lớp Bus một cách bình đẳng. Khi đó, lệnh myBus.show() sẽ gọi đến phương thức show() của lớp Car, do vậy, chương trình trên sẽ in ra màn hình kết quả như sau (tùy theo dữ liệu nhập vào ở 4 dòng đầu):

```
Toc do xe bus: 80
Nhan hieu xe bus: Mercedes
Gia xe bus: 5000
So hieu tuyen xe bus: 27
This is a Mercedes having a speed of 80km/h and its price is $5000
```

Trong dòng giới thiệu xe bus (vì ta đang dùng đối tượng myBus của lớp Bus), không có giới thiệu số hiệu tuyến xe. Lí do là vì ta đang dùng hàm show của lớp Car. Muốn có thêm phần giới thiệu về số hiệu tuyến xe buýt, ta phải định nghĩa lại hàm show trong lớp Bus. Mục 6.3.3 sẽ trình bày nội dung này.

6.3.3 Định nghĩa chồng các phương thức của lớp cơ sở

Định nghĩa chồng phương thức của lớp cơ sở

Một phương thức của lớp cơ sở bị coi là nạp chồng nếu ở lớp dẫn xuất cũng định nghĩa một phương thức có cùng tên.

Ví dụ, trong lớp Car, đã có phương thức show(), bây giờ, trong lớp Bus kế thừa từ lớp Car, ta cũng định nghĩa lại phương thức show():

```
class Car{
        …
        public:
                …
                void show();        // Phương thức của lớp cơ sở
};
class Bus: public Car{
        …
        public:
                …
                void show();        // Phương thức nạp chồng
};
```

khi đó, phương thức show() của lớp Bus được coi là phương thức nạp chồng từ phương thức show() của lớp Car.

Sử dụng các phương thức nạp chồng

Từ một đối tượng của lớp dẫn xuất, việc truy nhập đến phương thức đã được định nghĩa lại trong lớp dẫn xuất được thực hiện như lời gọi một phương thức thông thường:

- Đối với biến đối tượng thông thường:

```
<Tên đối tượng>.<Tên thành phần>([Các đối số]);
```

- Đối với con trỏ đối tượng:

```
<Tên đối tượng>-><Tên thành phần>([Các đối số]);
```

Ví dụ:

```
Bus myBus;
myBus.show();
```

sẽ gọi đến phương thức show() được định nghĩa trong lớp Bus.

Trong trường hợp, từ một đối tượng của lớp dẫn xuất, muốn truy nhập đến một phương thức của lớp cơ sở (đã bị định nghĩa lại ở lớp dẫn xuất) thì phải sử dụng chỉ thị phạm vi lớp trước phương thức được gọi:

- Đối với biến đối tượng thông thường:

```
<Tên đối tượng>.<Tên lớp cơ sở>::<Tên thành phần>([Các đối số]);
```

- Đối với con trỏ đối tượng:

```
<Tên đối tượng>-><Tên lớp cơ sở>::<Tên thành phần>([Các đối số]);
```

Ví dụ:

```
Bus myBus;
myBus.Car::show();
```

sẽ gọi đến phương thức show() được định nghĩa trong lớp Car từ một đối tượng của lớp Bus.

Chương trình 6.4 minh họa việc định nghĩa chồng hàm show() trong lớp Bus và việc sử dụng hai phương thức show() của hai lớp từ một đối tượng của lớp dẫn xuất.

Chương trình 6.4

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            int   getSpeed();             // Đọc tốc độ xe
            char[] getMark();             // Đọc nhãn xe
            float getPrice();             // Đọc giá xe
            // Khởi tạo thông tin về xe
```

```
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();                  // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

void Car::setSpeed(int speedIn){     // Gán tốc độ cho xe
      speed = speedIn;
}
int Car::getSpeed(){                 // Đọc tốc độ xe
      return speed;
}
char[] Car::getMark(){               // Đọc nhãn xe
      return mark;
}
float Car::getPrice(){               // Đọc giá xe
      return price;
}

void Car::show(){                    // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car */
class Bus: public Car{
            int label;               // Số hiệu tuyến xe
      public:
            // Khởi tạo đủ tham số
            Bus(int sIn=0, char mIn[]="", float pIn=0, int lIn=0);
            void show();             // Giới thiệu xe bus
};

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, int lIn):Car(sIn, mIn, pIn){
      label = lIn;
}
```

```
// Định nghĩa nạp chồng phương thức
void Bus::show(){                    // Giới thiệu xe bus
      cout << "This is a bus of type " << getMark() << ", on the line "
            << label << ", having a speed of " << getSpeed()
            << "km/h and its price is $" << getPrice() << endl;
      return;
}

// Chương trình chính
void main(){
      clrscr();
      Bus myBus(80, "Mercedes", 5000, 27);// Biến đối tượng của lớp Bus

      cout << "Gioi thieu xe:" << endl;
      myBus.Car::show();             // Phương thức của lớp Car
      cout << "Gioi thieu xe bus:" << endl;
      myBus.show();                  // Phương thức của lớp Bus
      return;
}
```

Chương trình 6.4 sẽ hiển thị các thông báo như sau:

```
Gioi thieu xe:
This is a Mercedes having a speed of 80km/h and its price is $5000
Gioi thieu xe bus:
This is a bus of type Mercedes, on the line 27, having a speed of
80km/h and its price is $5000
```

Lưu ý:

- Trong phương thức show() của lớp Bus, ta phải dùng các hàm get để truy nhập đến các thuộc tính của lớp Car. Không được truy nhập trực tiếp đến tên các thuộc tính (speed, mark và price) vì chúng có dạng private của lớp Car.

6.3.4 Chuyển đổi kiểu giữa lớp cơ sở và lớp dẫn xuất

Về mặt dữ liệu, một lớp dẫn xuất bao giờ cũng chứa toàn bộ dữ liệu của lớp cơ sở: Ta luôn tìm thấy lớp cơ sở trong lớp dẫn xuất, nhưng không phải bao giờ cũng tìm thấy lớp dẫn xuất trong lớp cơ sở. Do vậy:

- Có thể gán một đối tượng lớp dẫn xuất cho một đối tượng lớp cơ sở:

```
<Đối tượng lớp cơ sở> = <Đối tượng lớp dẫn xuất>; // Đúng
```

- Nhưng không thể gán một đối tượng lớp cơ sở cho một đối tượng lớp dẫn xuất:

```
<Đối tượng lớp dẫn xuất> = <Đối tượng lớp cơ sở>; // Không được
```

Ví dụ, ta có lớp Bus kế thừa từ lớp Car và:

```
Bus myBus;
```

```
Car myCar;
```

khi đó, phép gán:

```
myCar = myBus;    // đúng
```

thì chấp nhận được, nhưng phép gán:

```
myBus = myCar;    // không được
```

thì không chấp nhận được.

Lưu ý:

- Nguyên tắc chuyển kiểu này cũng đúng với các phép gán con trỏ: một con trỏ đối tượng lớp cơ sở có thể trỏ đến địa chỉ của một đối tượng lớp dẫn xuất. Nhưng một con trỏ đối tượng lớp dẫn xuất không thể trỏ đến địa chỉ một đối tượng lớp cơ sở.
- Nguyên tắc chuyển kiểu này cũng đúng với truyền đối số cho hàm: có thể truyền một đối tượng lớp dẫn xuất vào vị trí của tham số có kiểu lớp cơ sở. Nhưng không thể truyền một đối tượng lớp cơ sở vào vị trí một tham số có kiểu lớp dẫn xuất.

Chương trình 6.5 minh họa việc chuyển kiểu giữa các đối tượng của lớp cơ sở và lớp dẫn xuất.

Chương trình 6.5

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            int   getSpeed();             // Đọc tốc độ xe
            char[] getMark();             // Đọc nhãn xe
            float getPrice();             // Đọc giá xe
            // Khởi tạo thông tin về xe
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            void  show();                 // Giới thiệu xe
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}
```

```
int Car::getSpeed(){                    // Đọc tốc độ xe
      return speed;
}
char[] Car::getMark(){                  // Đọc nhãn xe
      return mark;
}
float Car::getPrice(){                  // Đọc giá xe
      return price;
}

void Car::show(){                       // Phương thức giới thiệu xe
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car */
class Bus: public Car{
            int label;                  // Số hiệu tuyến xe
      public:
            // Khởi tạo đủ tham số
            Bus(int sIn=0, char mIn[]="", float pIn=0, int lIn=0);
            void show();                // Định nghĩa chồng phương thức
};

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, int lIn):Car(sIn, mIn, pIn){
      label = lIn;
}

// Định nghĩa nạp chồng phương thức
void Bus::show(){                       // Giới thiệu xe bus
      cout << "This is a bus of type " << getMark() << ", on the line "
            << label << ", having a speed of " << getSpeed()
            << "km/h and its price is $" << getPrice() << endl;
      return;
}

// Chương trình chính
void main(){
      clrscr();
      Car myCar(100, "Ford", 3000); // Biến đối tượng lớp Car
      Bus myBus(80, "Mercedes", 5000, 27);// Biến đối tượng lớp Bus
```

```
        cout << "Gioi thieu xe o to lan 1:" << endl;
        myCar.show();
        cout << "Gioi thieu xe o to lan 2:" << endl;
        myCar = myBus;
        myCar.show();
        cout << "Gioi thieu xe bus:" << endl;
        myBus.show();
        return;
}
```

Chương trình 6.5 sẽ in ra kết quả thông báo như sau:

```
Goi thieu xe o to lan 1:
This is a Ford having a speed of 100km/h and its price is $3000
Gioi thieu xe lan o to lan 2:
This is a Mercedes having a speed of 80km/h and its price is $5000
Gioi thieu xe bus:
This is a bus of type Mercedes, on the line 27, having a speed of
80km/h and its price is $5000
```

Ở thông báo thứ nhất, đối tượng myCar gọi phương thức show() của lớp Car với các dữ liệu được khởi đầu cho myCar: (100, "Ford", 3000). Ở thông báo thứ hai, myCar gọi phương thức show() của lớp Car, nhưng với dữ liệu vừa được gán từ đối tượng myBus: (80, "Mercedes", 5000). Ở thông báo thứ ba, myBus gọi phương thức show() của lớp Bus với các dữ liệu của myBus: (80, "Mercedes", 5000, 27).

6.4 ĐA KẾ THỪA

C++ cho phép đa kế thừa, tức là một lớp có thể được dẫn xuất từ nhiều lớp cơ sở khác nhau, với những kiểu dẫn xuất khác nhau.

6.4.1 Khai báo đa kế thừa

Đa kế thừa được khai báo theo cú pháp:

```
class <Tên lớp dẫn xuất>: <Từ khoá dẫn xuất> <Tên lớp cơ sở 1>,
                          <Từ khoá dẫn xuất> <Tên lớp cơ sở 2>,
                          …
                          <Từ khoá dẫn xuất> <Tên lớp cơ sở n>{
        …       // Khai báo thêm các thành phần lớp dẫn xuất
};
```

Ví dụ:

```
class Bus: public Car, public PublicTransport{
        …       // Khai báo các thành phần bổ sung
};
```

là khai báo lớp Bus (xe buýt) kế thừa từ hai lớp xe Car (ô tô) và PublicTransport (phương tiện giao thông công cộng) theo cùng một kiểu dẫn xuất là public.

Lưu ý:

- Trong đa kế thừa, mỗi lớp cơ sở được phân cách nhau bởi dấu phẩy ",".
- Mỗi lớp cơ sở cơ thể có một kiểu dẫn xuất bởi một từ khoá dẫn xuất khác nhau.
- Nguyên tắc truy nhập vào các thành phần lớp cơ sở cũng hoàn toàn tương tự như trong kế thừa đơn.

6.4.2 Hàm khởi tạo và hàm huỷ bỏ trong đa kế thừa

Hàm khởi tạo trong đa kế thừa

Hàm khởi tạo trong đa kế thừa được khai báo tương tự như trong đơn kế thừa, ngoại trừ việc phải sắp xếp thứ tự gọi tới hàm khởi tạo của các lớp cơ sở: thông thường, thứ tự gọi đến hàm khởi tạo của các lớp cơ sở nên tuân theo thứ tự dẫn xuất từ các lớp cơ sở trong đa kế thừa.

Chương trình 6.6 minh hoạ việc định nghĩa hàm khởi tạo tường minh trong đa kế thừa: thứ tự gọi hàm khởi tạo của các lớp cơ sở trong hàm khởi tạo của lớp Bus là tương tự thứ tự dẫn xuất: hàm khởi tạo của lớp Car trước hàm khởi tạo của lớp PublicTransport.

Chương trình 6.6

```
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
            int   speed;                // Tốc độ
            char  mark[20];             // Nhãn hiệu
            float price;                // Giá xe
      public:
            Car();                      // Khởi tạo không tham số
            Car(int, char[], float);    // Khởi tạo đủ tham số
};

Car::Car(){                             // Khởi tạo không tham số
      speed = 0;
      strcpy(mark, "");
      price = 0;
}

// Khởi tạo đủ tham số
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
```

```
}

/* Định nghĩa lớp PublicTransport */
class PublicTransport{
            float ticket;                   // Giá vé phương tiện
      public:
            PublicTransport();              // Khởi tạo không tham số
            PublicTransport(float);         // Khởi tạo đủ tham số
};

PublicTransport::PublicTransport(){         // Khởi tạo không tham số
      ticket = 0;
}

// Khởi tạo đủ tham số
PublicTransport::PublicTransport(float ticketIn){
      ticket = ticketIn;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car và PublicTransport */
class Bus: public Car, public PublicTransport{ // Thứ tự khai báo
            int label;                      // Số hiệu tuyến xe
      public:
            Bus();                          // Khởi tạo không tham số
            Bus(int, char[], float, float, int);// Khởi tạo đủ tham số
};

// Khởi tạo không tham số
Bus::Bus(): Car(), Transport(){             // Theo thứ tự dẫn xuất
      label = 0;
}

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, float tIn, int lIn):
      Car(sIn, mIn, pIn), PublicTransport(tIn){ // Theo thứ tự dẫn xuất
      label = lIn;
}
```

Lưu ý:

- Trong trường hợp dùng hàm khởi tạo ngầm định hoặc không có tham số, ta có thể không cần gọi tường minh các hàm khởi tạo của các lớp cơ sở, trình biên dịch sẽ tự động gọi tới chúng theo đúng thứ tự dẫn xuất

Ví dụ, trong chương trình 6.6, hai cách định nghĩa hàm khởi tạo không tham số của lớp Bus sau là tương đương:

```
Bus::Bus(): Car(), Transport(){// Theo thứ tự dẫn xuất
      label = 0;
}
```

là tương đương với:

```
Bus::Bus(){                    // Theo thứ tự dẫn xuất ngầm định
      label = 0;
}
```

Hàm huỷ bỏ trong đa kế thừa

Vì hàm huỷ bỏ là duy nhất của mỗi lớp, hơn nữa hàm huỷ bỏ của lớp cơ sở sẽ được tự động gọi đến khi giải phóng đối tượng của lớp dẫn xuất. Cho nên hàm huỷ bỏ trong đa kế thừa hoàn toàn tương tự hàm huỷ bỏ trong đơn kế thừa:

- Hàm huỷ bỏ của lớp dẫn xuất chỉ giải phóng bộ nhớ cho các thành phần bổ sung, nếu có, của lớp dẫn xuất.
- Hàm huỷ bỏ của lớp dẫn xuất sẽ được gọi đến sớm nhất. Sau đó các hàm huỷ bỏ của các lớp cơ sở sẽ được gọi đến.
- Quá trình này được trình biên dịch thực hiện tự động.

6.4.3 Truy nhập các thành phần lớp trong đa kế thừa

Việc truy nhập đến các thành phần của các lớp trong đa kế thừa được dựa trên các nguyên tắc sau:

- Việc truy nhập từ đối tượng lớp dẫn xuất đến các thành phần của mỗi lớp cơ sở được tuân theo quy tắc phạm vi tương tự như trong đơn kế thừa.
- Trong trường hợp các lớp cơ sở đều có các thành phần cùng tên, việc truy xuất đến thành phần của lớp nào phải được chỉ rõ bằng toán tử phạm vi: “<Tên lớp>::” đối với thành phần lớp cơ sở đó.

Ví dụ, ta định nghĩa lớp Bus kế thừa từ hai lớp cơ sở: Car và PublicTransport. Nhưng cả ba lớp này đều định nghĩa một phương thức show() để tự giới thiệu:

```
class Car{
      public:
            void show();
};
class PublicTransport{
      public:
            void show();
};
class Bus: public Car, public PublicTransport{
      public:
            void show();
};
```

Khi đó, khai báo:

```
Bus myBus;
```

và lời gọi hàm:

```
myBus.show();                    // Gọi đến hàm của lớp Bus
myBus.Car::show();               // Gọi đến hàm của lớp Car
myBus.PublicTransport::show();// Gọi đến hàm của lớp PublicTransport
```

Chương trình 6.7 minh hoạ việc truy nhập đến các thành phần trùng nhau trong các lớp cơ sở và được định nghĩa lại trong lớp dẫn xuất.

Chương trình 6.7

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
            int   speed;                    // Tốc độ
            char  mark[20];                 // Nhãn hiệu
            float price;                    // Giá xe
      public:
            Car();                          // Khởi tạo không tham số
            Car(int, char[], float);        // Khởi tạo đủ tham số
            void show();                    // Giới thiệu
            float getSpeed(){return speed;};
            char[] getMark(){return mark;};
            float getPrice(){return price;};
};

Car::Car(){                                 // Khởi tạo không tham số
      speed = 0;
      strcpy(mark, "");
      price = 0;
}

// Khởi tạo đủ tham số
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}
```

```
// Giới thiệu
void Car::show(){
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}

/* Định nghĩa lớp PublicTransport */
class PublicTransport{
            float ticket;                 // Giá vé phương tiện
      public:
            PublicTransport();            // Khởi tạo không tham số
            PublicTransport(float);       // Khởi tạo đủ tham số
            void show();                  // Giới thiệu
            float getTicket(){return ticket;};
};

PublicTransport::PublicTransport(){       // Khởi tạo không tham số
      ticket = 0;
}

// Khởi tạo đủ tham số
PublicTransport::PublicTransport(float ticketIn){
      ticket = ticketIn;
}

// Giới thiệu
void PublicTransport::show(){
      cout << "This public transport had a ticket of $"
            << ticket << endl;
      return;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car và PublicTransport */
class Bus: public Car, public PublicTransport{ // Thứ tự khai báo
            int label;                    // Số hiệu tuyến xe
      public:
            Bus();                        // Khởi tạo không tham số
            Bus(int, char[], float, float, int);// Khởi tạo đủ tham số
            void show();                  // Giới thiệu
};

// Khởi tạo không tham số
```

```
Bus::Bus(): Car(), Transport(){           // Theo thứ tự dẫn xuất
      label = 0;
}

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, float tIn, int lIn):
      Car(sIn, mIn, pIn), PublicTransport(tIn){ // Theo thứ tự dẫn xuất
      label = lIn;
}

// Giới thiệu
void Bus::show(){
      cout << "This is a bus on the line " << label
            << ", its speed is " << getSpeed()
            << "km/h, mark is " << getMark()
            << ", price is $" << getPrice()
            << " and ticket is " << getTicket() << endl;
      return;
}

// phương thức main
void main(){
      clrscr();
      Bus myBus(100, "Mercedes", 3000, 1.5, 27);

      myBus.Car::show();                  // Hàm của lớp Car
      myBus.PublicTransport:: show();     // Hàm của lớp PublicTransport
      myBus.show();                       // Hàm của lớp Bus
      return;
}
```

Chương trình 6.7 sẽ in ra thông báo như sau:

```
This is a Mercedes having a speed of 100km/h and its price is $3000
This public transport had a ticket of $1.5
This is a bus on the line 27, its speed is 100km/h, mark is Mercedes,
price is $3000 and ticket is $1.5
```

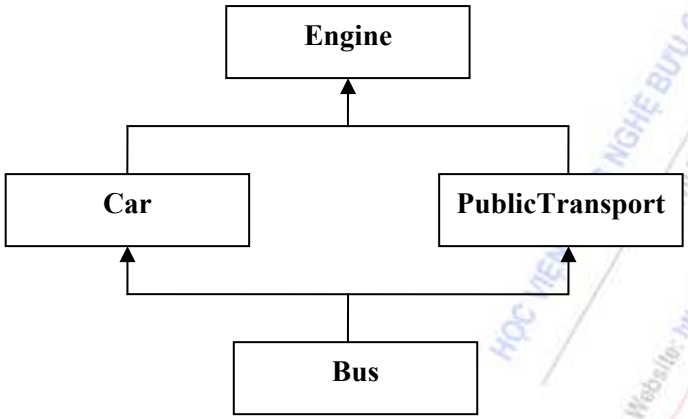
Dòng thứ nhất là kết quả của phương thức show() của lớp Car, dòng thứ hai, tương ứng là kết quả phương thức show() của lớp PublicTransport, dòng thứ ba là kết quả phương thức show() của lớp Bus.

6.5 LỚP CƠ SỞ TRỪU TƯỢNG

6.5.1 Đặt vấn đề

Sự cho phép đa kế thừa trong C++ dẫn đến một số hậu quả xấu, đó là sự đụng độ giữa các thành phần của các lớp cơ sở, khi có ít nhất hai lớp cơ sở lại cùng được kế thừa từ một lớp cơ sở khác. Xét trường hợp:

- Lớp Bus kế thừa từ lớp Car và lớp PublicTransport.
- Nhưng lớp Car và lớp PublicTransport lại cùng được thừa kế từ lớp Engine (động cơ). Lớp Engine có một thuộc tính là power (công suất của động cơ).



Khi đó, nảy sinh một số vấn đề như sau:

- Các thành phần dữ liệu của lớp Engine bị lặp lại trong lớp Bus hai lần: một lần do kế thừa theo đường Bus::Car::Engine, một lần theo đường Bus::PublicTransport::Engine. Điều này là không an toàn.
- Khi khai báo một đối tượng của lớp Bus, hàm khởi tạo của lớp Engine cũng được gọi hai lần: một lần do gọi truy hồi từ hàm khởi tạo lớp Car, một lần do gọi truy hồi từ hàm khởi tạo lớp PublicTransport.
- Khi giải phóng một đối tượng của lớp Bus, hàm huỷ bỏ của lớp Engine cũng sẽ bị gọi tới hai lần.

Để tránh các vấn đề này, C++ cung cấp một khái niệm là kế thừa từ *lớp cơ sở trừu tượng*. Khi đó, ta cho các lớp Car và PublicTransport kế thừa trừu tượng từ lớp Engine. Bằng cách này, các thành phần của lớp Engine chỉ xuất hiện trong lớp Bus đúng một lần. Lớp Engine được gọi là lớp cơ sở trừu tượng của các lớp Car và PublicTransport.

6.5.2 Khai báo lớp cơ sở trừu tượng

Việc chỉ ra một sự kế thừa trừu tượng được thực hiện bằng từ khoá **virtual** khi khai báo lớp cơ sở:

```
class <Tên lớp cơ sở>: <Từ khoá dẫn xuất> virtual <Tên lớp cơ sở>{
      …     // Khai báo các thành phần bổ sung
};
```

Ví dụ:

```
class Engine{
      …     // Các thành phần lớp Engine
```

```
};
class Car: public virtual Engine{
      …     // Khai báo các thành phần bổ sung
};
```

là khai báo lớp Car, kế thừa từ lớp cơ sở trừu tượng Engine, theo kiểu dẫn xuất public.

Lưu ý:

- Từ khoá virtual được viết bằng chữ thường.
- Từ khoá virtual không ảnh hưởng đến phạm vi truy nhập thành phần lớp cơ sở, phạm vi này vẫn được quy định bởi từ khoá dẫn xuất như thông thường.
- Từ khoá virtual chỉ ra một lớp cơ sở là trừu tượng nhưng lại được viết trong khi khai báo lớp dẫn xuất.
- Một lớp dẫn xuất có thể được kế thừa từ nhiều lớp cơ sở trừu tượng

6.5.3 Hàm khởi tạo lớp cơ sở trừu tượng

Khác với các lớp cơ sở thông thường, khi có một lớp dẫn xuất từ một lớp cơ sở trừu tượng, lại được lấy làm cơ sở cho một lớp dẫn xuất khác thì trong hàm khởi tạo của lớp dẫn xuất cuối cùng, vẫn phải gọi hàm khởi tạo tường minh của lớp cơ sở trừu tượng. Hơn nữa, hàm khởi tạo của lớp cơ sở trừu tượng phải được gọi sớm nhất.

Ví dụ, khi lớp Car và lớp PublicTransport được kế thừa từ lớp cơ sở trừu tượng Engine. Sau đó, lớp Bus được kế thừa từ hai lớp Car và PublicTranport. Khi đó, hàm khởi tạo của lớp Bus cũng phải gọi tường minh hàm khởi tạo của lớp Engine, theo thứ tự sớm nhất, sau đó mới gọi đến hàm khởi tạo của các lớp Car và PublicTransport.

```
class Engine{
      public:
            Engine(){… };
};
class Car: public virtual Engine{                //Lớp cơ sở virtual
      public:
            Car(): Engine(){… };
};
class PublicTransport: public virtual Engine{    //Lớp cơ sở virtual
      public:
            PublicTransport():Engine(){… };
};
class Bus: public Car, public PublicTransport{
      public:
            // Gọi hàm khởi tạo tường minh của lớp cơ sở trừu tượng
            Bus():Engine(), Car(), PublicTransport(){… };
};
```

Lưu ý:

- Trong trường hợp lớp Engine không phải là lớp cơ sở trừu tượng của các lớp Car và PublicTransport, thì trong hàm khởi tạo của lớp Bus không cần gọi hàm khởi tạo của lớp

Engine, mà chỉ cần gọi tới các hàm khởi tạo của các lớp cơ sở trực tiếp của lớp Bus là lớp Car và lớp PublicTransport.

Chương trình 6.8 minh hoạ việc khai báo và sử dụng lớp cơ sở trừu tượng: lớp Engine là lớp cơ sở trừu tượng của các lớp Car và lớp PublicTransport. Hai lớp này, sau đó, lại làm lớp cơ sở của lớp Bus.

Chương trình 6.8

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Engine */
class Engine{
            int   power;                  // Công suất
      public:
            Engine(){power = 0;};         // Khởi tạo không tham số
            Engine(int pIn){power = pIn;};// Khởi tạo đủ tham số
            void show();                  // Giới thiệu
            float getPower(){return power;};
};

// Giới thiệu
void Engine::show(){
      cout << "This is an engine having a power of "
            << power << "KWH" << endl;
      return;
}

/* Định nghĩa lớp Car dẫn xuất từ lớp cơ sở trừu tượng Engine*/
class Car: public virtual Engine{
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            Car();                        // Khởi tạo không tham số
            Car(int, int, char[], float); // Khởi tạo đủ tham số
            void show();                  // Giới thiệu
            float getSpeed(){return speed;};
            char[] getMark(){return mark;};
            float getPrice(){return price;};
};
```

```
Car::Car(): Engine(){                          // Khởi tạo không tham số
      speed = 0;
      strcpy(mark, "");
      price = 0;
}

// Khởi tạo đủ tham số
Car::Car(int pwIn, int sIn, char mIn[], float prIn): Engine(pwIn){
      speed = sIn;
      strcpy(mark, mIn);
      price = prIn;
}

// Giới thiệu
void Car::show(){
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h, its power is" << getPower()
            << "KWh and price is $" << price << endl;
      return;
}

/* Định nghĩa lớp PublicTransport dẫn xuất trừu tượng từ lớp Engine */
class PublicTransport: public virtual Engine{
            float ticket;                      // Giá vé phương tiện
      public:
            PublicTransport();                 // Khởi tạo không tham số
            PublicTransport(int, float);       // Khởi tạo đủ tham số
            void show();                       // Giới thiệu
            float getTicket(){return ticket;};
};

// Khởi tạo không tham số
PublicTransport::PublicTransport(): Engine(){
      ticket = 0;
}

// Khởi tạo đủ tham số
PublicTransport::PublicTransport(int pwIn, float tIn): Engine(pwIn){
      ticket = tIn;
}

// Giới thiệu
void PublicTransport::show(){
```

```
      cout << "This is a public transport havìn a ticket of $"
            << ticket << " and its power is " << getPower()
            << "KWh" << endl;
      return;
}

/* Định nghĩa lớp Bus kế thừa từ lớp Car và PublicTransport */
class Bus: public Car, public PublicTransport{ // Thứ tự khai báo
            int label;                    // Số hiệu tuyến xe
      public:
            Bus();                        // Khởi tạo không tham số
            Bus(int,int,char[],float,float,int);// Khởi tạo đủ tham số
            void show();                  // Giới thiệu
};

// Khởi tạo không tham số
Bus::Bus(): Engine(), Car(), Transport(){       // Theo thứ tự dẫn xuất
      label = 0;
}

// Khởi tạo đủ tham số
Bus::Bus(int pwIn, int sIn, char mIn[], float prIn, float tIn, int lIn):
      Engine(pwIn), Car(sIn, mIn, prIn), PublicTransport(tIn){
      label = lIn;
}

// Giới thiệu
void Bus::show(){
      cout << "This is a bus on the line " << label
            << ", its speed is " << getSpeed()
            << "km/h, power is" << Car::getPower()
            << "KWh, mark is " << getMark()
            << ", price is $" << getPrice()
            << " and ticket is " << getTicket() << endl;
      return;
}

// phương thức main
void main(){
      clrscr();
      Bus myBus(250, 100, "Mercedes", 3000, 1.5, 27);

      myBus.Car::Engine::show();          // Hàm của lớp Engine
```

```
        myBus.PublicTransport::Engine::show();// Hàm của lớp Engine
        myBus.Car::show();                    // Hàm của lớp Car
        myBus.PublicTransport:: show();       // Hàm của lớp PublicTransport
        myBus.show();                         // Hàm của lớp Bus
        return;
}
```

Chương trình 6.8 sẽ in ra thông báo như sau:

```
This is an engine having a power of 250KWh
This is an engine having a power of 250KWh
This is a Mercedes having a speed of 100km/h, its power is 250KWh and
price is $3000
This is a public transport having a ticket of $1.5 and its power is
250KWh
This is a bus on the line 27, its speed is 100km/h, power is 250KWh,
mark is Mercedes, price is $3000 and ticket is $1.5
```

Hai dòng đầu là kết quả của phương thức show() của lớp Engine: một lần gọi qua lớp Car, một lần gọi qua lớp PublicTransport, chúng cho kết quả như nhau. Dòng thứ ba là kết quả phương thức show() của lớp Car. Dòng thứ tư, tương ứng là kết quả phương thức show() của lớp PublicTransport. Dòng thứ năm là kết quả phương thức show() của lớp Bus.

6.6 ĐA HÌNH

6.6.1 Đặt vấn đề

Sự kế thừa trong C++ cho phép có sự tương ứng giữa lớp cơ sở và các lớp dẫn xuất trong sơ đồ thừa kế:

- Một con trỏ có kiểu lớp cơ sở luôn có thể trỏ đến địa chỉ của một đối tượng của lớp dẫn xuất.
- Tuy nhiên, khi thực hiện lời gọi một phương thức của lớp, trình biên dịch sẽ quan tâm đến kiểu của con trỏ chứ không phải đối tượng mà con trỏ đang trỏ tới: phương thức của lớp mà con trỏ có kiểu được gọi chứ không phải phương thức của đối tượng mà con trỏ đang trỏ tới được gọi.

Ví dụ, lớp Bus kế thừa từ lớp Car, cả hai lớp này đều định nghĩa phương thức show():

```
class Car{
      public:
            void show();
};
class Bus: public Car{
      public:
            void show();
};
```

khi đó, nếu ta khai báo một con trỏ lớp Bus, nhưng lại trỏ vào địa chỉ của một đối tượng lớp Car:

```
Bus myBus;
Car *ptrCar = &myBus;   // đúng
```

nhưng khi gọi:

```
ptrCar->show();
```

thì chương trình sẽ gọi đến phương thức show() của lớp Car (là kiểu của con trỏ ptrCar), mà không gọi tới phương thức show() của lớp Bus (là kiểu của đối tượng myBus mà con trỏ ptrCar đang trỏ tới).

Để giải quyết vấn đề này, C++ đưa ra một khái niệm là phương thức trừu tượng. Bằng cách sử dụng phương thức trừu tượng. Khi gọi một phương thức từ một con trỏ đối tượng, trình biên dịch sẽ xác định kiểu của đối tượng mà con trỏ đang trỏ đến, sau đó nó sẽ gọi phương thức tương ứng với đối tượng mà con trỏ đang trỏ tới.

6.6.2 Khai báo phương thức trừu tượng

Phương thức trừu tượng (còn gọi là phương thức ảo, hàm ảo) được khai báo với từ khoá **virtual**:

- Nếu khai báo trong phạm vi lớp:

```
virtual <Kiểu trả về> <Tên phương thức>([<Các tham số>]);
```

- Nếu định nghĩa ngoài phạm vi lớp:

```
virtual <Kiểu trả về> <Tên lớp>::<Tên phương thức>([<Các tham số>]){…}
```

Ví dụ:

```
class Car{
      public:
            virtual void show();
};
```

là khai báo phương thức trừu tượng show() của lớp Car: phương thức không có tham số và không cần giá trị trả về (void).

Lưu ý:

- Từ khoá virtual có thể đặt trước hay sau kiểu trả về của phương thức.
- Với cùng một phương thức được khai báo ở lớp cơ sở lẫn lớp dẫn xuất, chỉ cần dùng từ khoá virtual ở một trong hai lần định nghĩa phương thức đó là đủ: hoặc ở lớp cơ sở, hoặc ở lớp dẫn xuất.
- Trong trường hợp cây kế thừa có nhiều mức, cũng chỉ cần khai báo phương thức là trừu tượng (virtual) ở một mức bất kì. Khi đó, tất cả các phương thức trùng tên với phương thức đó ở tất cả các mức đều được coi là trừu tượng.
- Đôi khi không cần thiết phải định nghĩa chồng (trong lớp dẫn xuất) một phương thức đã được khai báo trừu tượng trong lớp cơ sở.

6.6.3 Sử dụng phương thức trừu tượng – đa hình

Một khi phương thức được khai báo là trừu tượng thì khi một con trỏ gọi đến phương thức đó, chương trình sẽ thực hiện phương thức tương ứng với đối tượng mà con trỏ đang trỏ tới, thay vì

thực hiện phương thức của lớp cùng kiểu với con trỏ. Đây được gọi là hiện tượng đa hình (tương ứng bội) trong C++.

Chương trình 6.9 minh hoạ việc sử dụng phương thức trừu tượng: lớp Bus kế thừa từ lớp Car, hai lớp này cùng định nghĩa phương thức trừu tượng show().

- Khi ta dùng một con trỏ có kiểu lớp Car trỏ vào địa chỉ của một đối tượng kiểu Car, nó sẽ gọi phương thức show() của lớp Car.
- Khi ta dùng cũng con trỏ đó, trỏ vào địa chỉ của một đối tượng kiểu Bus, nó sẽ gọi phương thức show() của lớp Bus.

Chương trình 6.9

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

/* Định nghĩa lớp Car */
class Car{
      private:
            int   speed;                  // Tốc độ
            char  mark[20];               // Nhãn hiệu
            float price;                  // Giá xe
      public:
            int   getSpeed(){return speed;};// Đọc tốc độ xe
            char[] getMark(){return mark;};// Đọc nhãn xe
            float getPrice(){return price;};// Đọc giá xe
            // Khởi tạo thông tin về xe
            Car(int speedIn=0, char markIn[]="", float priceIn=0);
            virtual void show();    // Giới thiệu xe, trừu tượng
};

/* Khai báo phương thức bên ngoài lớp */
Car::Car(int speedIn, char markIn[], float priceIn){
      speed = speedIn;
      strcpy(mark, markIn);
      price = priceIn;
}

// Phương thức trừu tượng giới thiệu xe
virtual void Car::show(){
      cout << "This is a " << mark << " having a speed of "
            << speed << "km/h and its price is $" << price << endl;
      return;
}
```

```
/* Định nghĩa lớp Bus kế thừa từ lớp Car */
class Bus: public Car{
            int label;              // Số hiệu tuyến xe
      public:
            // Khởi tạo đủ tham số
            Bus(int sIn=0, char mIn[]="", float pIn=0, int lIn=0);
            void show();            // Giới thiệu xe
};

// Khởi tạo đủ tham số
Bus::Bus(int sIn, char mIn[], float pIn, int lIn):Car(sIn, mIn, pIn){
      label = lIn;
}

// Định nghĩa nạp chồng phương thức trừu tượng
void Bus::show(){                   // Giới thiệu xe bus
      cout << "This is a bus of type " << getMark() << ", on the line "
            << label << ", having a speed of " << getSpeed()
            << "km/h and its price is $" << getPrice() << endl;
      return;
}

// Chương trình chính
void main(){
      clrscr();
      Car *ptrCar, myCar(100, "Ford", 3000);
      Bus myBus(150, "Mercedes", 5000, 27);// Biến đối tượng của lớp Bus

      ptrCar = &myCar;              // Trỏ đến đối tượng lớp Car
      ptrCar->show();               // Phương thức của lớp Car

      ptrCar = &myBus;              // Trỏ đến đối tượng lớp Bus
      ptrCar->show();               // Phương thức của lớp Bus
      return;
}
```

Chương trình 6.9 hiển thị kết quả thông báo như sau:

```
This is a Ford having a speed of 100km/h and its price is $3000
This is a bus of type Mercedes, on the line 27, having a speed of
150km/h and its price is $5000
```

Dòng thứ nhất là kết quả khi con trỏ ptrCar trỏ đến địa chỉ của đối tượng myCar, thuộc lớp Car nên sẽ gọi phương thức show() của lớp Car với các dữ liệu của đối tượng myCar: (100, Ford,

3000). Dòng thứ hai tương ứng là kết quả khi con trỏ ptrCar trỏ đến địa chỉ của đối tượng myBus, thuộc lớp Bus nên sẽ gọi phương thức show() của lớp Bus, cùng với các tham số của đối tượng myBus: (150, Mercedes, 5000, 27).

Lưu ý:

- Trong trường hợp ở lớp dẫn xuất không định nghĩa lại phương thức trừu tượng, thì chương trình sẽ gọi phương thức của lớp cơ sở, nhưng với dữ liệu của lớp dẫn xuất.

Ví dụ, nếu trong chương trình 6.9, lớp Bus không định nghĩa chồng phương thức trừu tượng show() thì kết quả hiển thị sẽ là hai dòng thông báo giống nhau, chỉ khác nhau ở dữ liệu của hai đối tượng khác nhau:

```
This is a Ford having a speed of 100km/h and its price is $3000
This is a Mercedes having a speed of 150km/h and its price is $5000
```

TỔNG KẾT CHƯƠNG 6

Nội dung chương 6 đã trình bày các vấn đề cơ bản liên quan đến thừa kế và tương ứng bội trong C++ như sau:

- Khai báo một lớp dẫn xuất kế thừa từ một lớp cơ sở bằng khai báo kế thừa ":" đi kèm với một từ khoá dẫn xuất.
- Có ba loại dẫn xuất khác nhau, được quy định bởi ba từ khoá dẫn xuất khác nhau: private, protected và public. Kiểu dẫn xuất phổ biến là dẫn xuất public.
- Sự kế thừa tạo ra mối quan hệ tương ứng giữa lớp cơ sở và lớp dẫn xuất: có thể chuyển kiểu ngầm định (trong phép gán, phép truyền đối số, phép trỏ địa chỉ) từ một đối tượng lớp cơ sở đến một đối tượng lớp dẫn xuất. Nhưng không thể chuyển kiểu ngược lại từ lớp dẫn xuất vào lớp cơ sở.
- Hàm khởi tạo của lớp dẫn xuất có thể gọi tường minh hoặc gọi ngầm định hàm khởi tạo của lớp cơ sở. Hàm khởi tạo lớp cơ sở bao giờ cũng được thực hiện trước hàm khởi tạo lớp dẫn xuất.
- Hàm huỷ bỏ của lớp dẫn xuất luôn gọi ngầm định hàm huỷ bỏ của lớp cơ sở. Trái với hàm khởi tạo, hàm huỷ bỏ lớp cơ sở luôn được thực hiện sau hàm huỷ bỏ của lớp dẫn xuất.
- Có thể truy nhập các phương thức của lớp cơ sở từ lớp dẫn xuất, phạm vi truy nhập là phụ thuộc vào kiểu dẫn xuất: private, protected hoặc public. Điều này cho phép sử dụng lại mã nguồn của lớp cơ sở, mà không cần định nghĩa lại ở lớp dẫn xuất.
- Trong lớp dẫn xuất, có thể định nghĩa chồng một số phương thức của lớp cơ sở. Khi có định nghĩa chồng, muốn truy nhập vào phương thức lớp cơ sở, phải dùng toán tử phạm vi lớp "<Tên lớp>::".
- C++ còn cho phép một lớp có thể được dẫn xuất từ nhiều lớp cơ sở khác nhau, gọi là đa kế thừa. Trong đa kế thừa, quan hệ giữa lớp dẫn xuất với mỗi lớp cơ sở là tương tự như trong đơn kế thừa.
- Trong đa kế thừa, hàm khởi tạo lớp dẫn xuất sẽ gọi tường minh (hoặc ngầm định) hàm khởi tạo các lớp cơ sở, theo thứ tự khai báo kế thừa. Hàm huỷ bỏ lớp dẫn xuất lại gọi ngầm định các hàm huỷ bỏ của các lớp cơ sở.

- C++ cung cấp khái niệm kế thừa từ lớp cơ sở trừu tượng để tránh trường hợp trùng lặp dữ liệu ở lớp dẫn xuất, khi các lớp cơ sở lại cùng được dẫn xuất từ một lớp khác.
- C++ cũng cho phép cơ chế tương ứng bội (đa hình) bằng cách định nghĩa một phương thức là trừu tượng trong sơ đồ thừa kế. Khi đó, một con trỏ lớp cơ sở có thể trỏ đến địa chỉ của một đối tượng lớp dẫn xuất, và phương thức được thực hiện là tuỳ thuộc vào kiểu của đối tượng mà con trỏ đang trỏ tới.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 6

1. Trong các khai báo sau, khai báo nào là đúng cú pháp kế thừa lớp:

   a. `class A:  public class B{…};`

   b. `class A:  public B{…};`

   c. `class A:  class B{…};`

   d. `class A::  public B{…};`

2. Trong các kiểu dẫn xuất sau, từ các phương thức lớp dẫn xuất, không thể truy nhập đến các thành phần private của lớp cơ sở:

   a. `private`

   b. `protected`

   c. `public`

   d. `Cả ba kiểu trên`

3. Trong các kiểu dẫn xuất sau, từ đối tượng của lớp dẫn xuất, có thể truy nhập đến các thành phần của lớp cơ sở:

   a. `private`

   b. `protected`

   c. `public`

   d. `Cả ba kiểu trên`

4. A là lớp dẫn xuất public từ lớp cơ sở B. Giả sử có các kiểu khai báo:

   ```
   A myA, *ptrA;
   B myB, *ptrB;
   ```

   Khi đó, các lệnh nào sau đây là không có lỗi:

   a. `myA = myB;`

   b. `myB = myA;`

   c. `ptrA = &myB;`

   d. `ptrB = &myA;`

   e. `ptrA = ptrB;`

   f. `ptrB = ptrA;`

5. A là lớp dẫn xuất public từ lớp cơ sở B. Giả sử có các kiểu khai báo và nguyên mẫu hàm:

   ```
   A myA;
   B myB;
   void show(A, B);
   ```

Khi đó, các lệnh gọi hàm nào sau đây là không có lỗi:

a. `show(myA, myA);`

b. `show(myA, myB);`

c. `show(myB, myA);`

d. `show(myB, myB);`

6. A là lớp dẫn xuất public từ lớp cơ sở B. Giả sử B có một hàm khởi tạo:

```
B(int, float);
```

Khi đó, định nghĩa hàm khởi tạo nào sau đây của lớp A là chấp nhận được:

a. `A::A(){…};`

b. `A::A(): B(){…};`

c. `A::A(int x, float y): B(){…};`

d. `A::A(int x, float y): B(x, y){…};`

7. A là lớp dẫn xuất public từ lớp cơ sở B. Giả sử B có hai hàm khởi tạo:

```
B();
B(int, float);
```

Khi đó, những định nghĩa hàm khởi tạo nào sau đây của lớp A là chấp nhận được:

a. `A::A(){…};`

b. `A::A(): B(){…};`

c. `A::A(int x, float y): B(){…};`

d. `A::A(int x, float y): B(x, y){…};`

8. A là lớp dẫn xuất public từ lớp cơ sở B. Giả sử B có hàm huỷ bỏ tường minh:

```
~B();
```

Khi đó, những định nghĩa hàm huỷ bỏ nào sau đây của lớp A là chấp nhận được:

a. `A::~A(){…};`

b. `A::~A(): ~B(){…};`

c. `A::~A(int x){…};`

d. `A::~A(int x): ~B(){…};`

9. Giả sử B là một lớp được khai báo:

```
class B{
      int x;
      public: int getx();
};
```

và A là một lớp dẫn xuất từ lớp B theo kiểu private:

```
class A: private B{
};
```

khi đó, nếu myA là một đối tượng lớp A, lệnh nào sau đây là chấp chận được:

a. `myA.x;`

b. `myA.getx();`

c. Cả hai lệnh trên.

d. Không lệnh nào cả.

10. Giả sử B là một lớp được khai báo:

```
class B{
      int x;
      public: int getx();
};
```

và A là một lớp dẫn xuất từ lớp B theo kiểu protected:

```
class A: protected B{
};
```

khi đó, nếu myA là một đối tượng lớp A, lệnh nào sau đây là chấp chận được:

a. `myA.x;`

b. `myA.getx();`

c. Cả hai lệnh trên.

d. Không lệnh nào cả.

11. Giả sử B là một lớp được khai báo:

```
class B{
      int x;
      public: int getx();
};
```

và A là một lớp dẫn xuất từ lớp B theo kiểu public:

```
class A: public B{
};
```

khi đó, nếu myA là một đối tượng lớp A, lệnh nào sau đây là chấp chận được:

a. `myA.x;`

b. `myA.getx();`

c. Cả hai lệnh trên.

d. Không lệnh nào cả.

12. Giả sử B là một lớp được khai báo:

```
class B{
      public: void show();
};
```

và A là một lớp dẫn xuất từ lớp B theo kiểu public, có định nghĩa chồng hàm show():

```
class A: public B{
      public: void show();
};
```

khi đó, nếu myA là một đối tượng lớp A, muốn thực hiện phương thức show() của lớp B thì lệnh nào sau đây là chấp chận được:

a. `myA.show();`

b. `myA.B::show();`

c. `B::myA.show();`

d. `A::B::show();`

13. Muốn khai báo một lớp A kế thừa từ hai lớp cơ sở B và C, những lệnh nào là đúng:

    a. `class A: B, C{…};`

    b. `class A: public B, C{…};`

    c. `class A: public B, protected C{…};`

    d. `class A: public B, public C{…};`

14. B là một lớp có hai hàm khởi tạo:

    ```
    B();
    B(int);
    ```

    C cũng là một lớp có hai hàm khởi tạo:

    ```
    C();
    C(int, int);
    ```

    Và A là một lớp kế thừa từ B và C:

    ```
    class A: public B, public C{…};
    ```

    Khi đó, hàm khởi tạo nào sau đây của lớp A là chấp nhận được:

    a. `A::A(){…};`

    b. `A::A():B(),C(){…};`

    c. `A::A(int x, int y): C(x, y){…};`

    d. `A::A(int x, int y, int z): B(x), C(y, z){…};`

    e. `A::A(int x, int y, int z): C(x, y), B(z){…};`

15. Muốn khai báo lớp A kế thừa từ lớp cơ sở trừu tượng B, những khai báo nào sau đây là đúng:

    a. `virtual class A: public B{…};`

    b. `class virtual A: public B{…};`

    c. `class A: virtual public B{…};`

    d. `class A: public virtual B{…};`

16. Lớp A là một lớp dẫn xuất, được kế thừa từ lớp cơ sở B. Hai lớp này đều định nghĩa hàm show(). Muốn hàm này trở thành trừu tượng thì những định nghĩa nào sau đây là đúng:

    a. `void A::show(){…}` và `void B::show(){…}`

    b. `virtual void A::show(){…}` và `void B::show(){…}`

    c. `void A::show(){…}` và `virtual void B::show(){…}`

    d. `virtual void A::show(){…}` và `virtual void B::show(){…}`

17. Khai báo lớp người (Human) bao gồm các thuộc tính sau:

    - Tên người (name)
    - Tuổi của người đó (age)
    - Giới tính của người đó (sex)

    Sau đó khai báo lớp Cá nhân (Person) kế thừa từ lớp Human vừa được định nghĩa ở trên.

18. Bổ sung các phương thức truy nhập các thuộc tính của lớp Human, các phương thức này có tính chất public.
19. Bổ sung thêm các thuộc tính của lớp Person: địa chỉ và số điện thoại. Thêm các phương thức truy nhập các thuộc tính này trong lớp Person.
20. Xây dựng hai hàm khởi tạo cho lớp Human: một hàm không tham số, một hàm với đủ ba tham số tương ứng với ba thuộc tính của nó. Sau đó, xây dựng hai hàm khởi tạo cho lớp Person có sử dụng các hàm khởi tạo của lớp Human: một hàm không tham số, một hàm đủ năm tham số (ứng với hai thuộc tính của lớp Person và ba thuộc tính của lớp Human).
21. Xây dựng một hàm main, trong đó có yêu cầu nhập các thuộc tính để tạo một đối tượng có kiểu Human và một đối tượng có kiểu Person, thông qua các hàm set thuộc tính đã xây dựng.
22. Xây dựng hàm show() cho hai lớp Human và Person. Thay đổi hàm main: dùng một đối tượng có kiểu lớp Person, gọi hàm show() của lớp Person, sau đó lại gọi hàm show() của lớp Human từ chính đối tượng đó.
23. Khai báo thêm một lớp người lao động (Worker), kế thừa từ lớp Human, có thêm thuộc tính là số giờ làm việc trong một tháng (hour) và tiền lương của người đó (salary). Sau đó, khai báo thêm một lớp nhân viên (Employee) kế thừa đồng thời từ hai lớp: Person và Worker. Lớp Employee có bổ sung thêm một thuộc tính là chức vụ (position).
24. Chuyển lớp Human thành lớp cơ sở trừu tượng của hai lớp Person và Worker. Xây dựng thêm hai hàm show() của lớp Worker và lớp Employee. Trong hàm main, khai báo một đối tượng lớp Employee, sau đó gọi đến các hàm show() của các lớp Employee, Person, Worrker và Human.
25. Chuyển hàm show() trong các lớp trên thành phương thức trừu tượng. Trong hàm main, khai báo một con trỏ kiểu Human, sau đó, cho nó trỏ đến lần lượt các đối tượng của các lớp Human, Person, Worker và Employee, mỗi lần đều gọi phương thức show() để hiển thị thông báo ra màn hình.

CHƯƠNG 7

MỘT SỐ LỚP QUAN TRỌNG

Nội dung chương này tập trung trình bày một số lớp cơ bản trong C++:

- Lớp vật chứa (Container)
- Lớp tập hợp (Set)
- Lớp chuỗi kí tự (String)
- Lớp ngăn xếp (Stack) và hàng đợi (Queue)
- Lớp danh sách liên kết (Lists)

7.1 LỚP VẬT CHỨA

Lớp vật chứa (Container) bao gồm nhiều lớp cơ bản của C++: lớp Vector, lớp danh sách (List) và các kiểu hàng đợi (Stack và Queue), lớp tập hợp (Set) và lớp ánh xạ (Map). Trong chương này sẽ trình bày một số lớp cơ bản của Container là: Set, Stack, Queue và List

7.1.1 Giao diện của lớp Container

Các lớp cơ bản của Container có một số toán tử và phương thức có chức năng giống nhau, bao gồm:

- ==: Toán tử so sánh bằng
- <: Toán tử so sánh nhỏ hơn
- **begin()**: Giá trị khởi đầu của con chạy iterator
- **end()**: Giá trị kết thúc của con chạy iterator
- **size()**: Số lượng phần tử đối tượng của vật chứa
- **empty()**: Vật chứa là rỗng
- **front()**: Phần tử thứ nhất của vật chứa
- **back()**: Phần tử cuối của vật chứa
- **[]**: Toán tử truy nhập đến phần tử của vật chứa
- **insert()**: Thêm vào vật chứa một (hoặc một số) phần tử
- **push_back()**: Thêm một phần tử vào cuối vật chứa
- **push_front()**: Thêm một phần tử vào đầu vật chứa
- **erase()**: Loại bỏ một (hoặc một số) phần tử khỏi vật chứa
- **pop_back()**: Loại bỏ phần tử cuối của vật chứa
- **pop_front()**: Loại bỏ phần tử đầu của vật chứa.

Ngoài ra, tuỳ vào các lớp cụ thể mà có một số toán tử và phương thức đặc trưng của lớp đó. Các toán tử và phương thức này sẽ được trình bày chi tiết trong nội dung từng lớp tiếp theo.

7.1.2 Con chạy Iterator

Iterator là một con trỏ trỏ đến các phần tử của vật chứa. Nó đóng vai trò là một con chạy cho phép người dùng di chuyển qua từng phần tử có mặt trong vật chứa. Mỗi phép tăng giảm một đơn vị của con chạy này tương ứng với một phép dịch đến phần tử tiếp theo hay phần tử trước của phần tử hiện tại mà con chạy đang trỏ tới.

Khai báo con chạy

Cú pháp chung để khai báo một biến con chạy iterator như sau:

```
Tên_lớp<T>::iterator Tên_con_chạy;
```

Trong đó:

- **Tên lớp**: là tên của lớp cơ bản ta đang dùng, ví dụ lớp Set, lớp List…
- **T**: là tên kiểu lớp của các phần tử chứa trong vật chứa. Kiểu có thể là các kiểu cơ bản trong C++, cũng có thể là các kiểu phức tạp do người dùng tự định nghĩa.
- **Tên con chạy**: là tên biến sẽ được sử dụng làm biến chạy trong vật chứa.

Ví dụ:

```
Set<int>::iterator iter;
```

là khai báo một biến con chạy iter cho lớp tập hợp (Set), trong đó, các phần tử của lớp vật chứa có kiểu cơ bản int. Hoặc:

```
List<Person>::iterator iter;
```

là khai báo một biến con chạy iter cho lớp danh sách (List), trong đó, các phần tử có kiểu lớp do người dùng tự định nghĩa là Person.

Sử dụng con chạy

Con chạy được sử dụng khi cần duyệt lần lượt các phần tử có mặt trong vật chứa. Ví dụ sau sẽ in ra các phần tử có kiểu int của một tập hợp:

```
Set<int> mySet;          // Khai báo một đối tượng của lớp Set,
                         // các phần tử có kiểu int
Set<int>::iterator i;    // Khai báo con chạy của lớp Set,
                         // các phần tử có kiểu int
…                        // Thêm phần tử vào
for(i=mySet.begin(); i<mySet.end(); i++)
      cout << mySet[i] << " ";
```

Lưu ý:

- Biến con chạy phải được khởi đầu bằng phương thức begin() và kết thúc bằng phương thức end() của đối tượng cần duyệt tương ứng.
- Một con chạy có thể được sử dụng nhiều lần cho nhiều đối tượng nếu chúng có cùng kiểu lớp vật chứa và các phần tử của chúng cũng cùng kiểu.

7.2 LỚP TẬP HỢP

Lớp tập hợp (Set) chứa các phần tử có cùng kiểu, không phân biệt thứ tự giữa các phần tử nhưng lại phân biệt giữa các phần tử: các phần tử là khác nhau từng đôi một. Muốn sử dụng lớp tập hợp, phải có chị thị đầu tệp:

```
#include<set.h>   // Thư viện riêng của lớp tập hợp
```

hoặc:

```
#include<stl.h>   // Thư viện chung cho các lớp vật chứa
```

7.2.1 Hàm khởi tạo

Lớp tập hợp có ba kiểu khởi tạo chính:

- Khởi tạo không tham số:

```
Set<T> Tên_đối_tượng;
```

- Khởi tạo bằng một mảng các đối tượng phần tử:

```
Set<T> Tên_đối_tượng(T*, chiều_dài_mảng);
```

- Khởi tạo bằng một đối tượng thuộc lớp tập hợp khác:

```
Set<T> Tên_đối_tượng(Set<T>);
```

Trong đó:

- **T**: là tên kiểu của các phần tử của tập hợp. Kiểu này có thể là kiểu cơ bản của C++, cũng có thể là các kiểu cấu trúc (struct) hoặc lớp (class) do người dùng tự định nghĩa.

Ví dụ:

```
Set<int> mySet;
```

là khai báo một đối tượng mySet của lớp tập hợp, mySet khởi đầu chưa có phần tử nào, các phần tử của đối tượng này có kiểu cơ bản int. Hoặc:

```
Person *myPerson = new Person[10];  // Khởi tạo mảng đối tượng
Set<Person> mySet(myPerson, 10);    // Khai báo tập hợp
```

là khai báo một đối tượng mySet của lớp tập hợp, có 10 phần tử tương ứng với các phần tử trong mảng động myPerson, các phần tử có kiểu lớp Person.

7.2.2 Toán tử

Các toán tử trên lớp tập hợp là tương ứng với các toán tử số học trên tập hợp thông thường: OR "|", AND "&", XOR "^" và phép trừ "-".

Phép toán "|" và "|="

Phép toán này trả về phép hợp (OR) của hai đối tượng của lớp tập hợp, kết quả cũng là một đối tượng của lớp tập hợp:

```
<Đối_tượng_1> = <Đối_tượng_2>  | <Đối_tượng_3>;
<Đối_tượng_1> |= <Đối_tượng_2>;
```

Ví dụ, mySet1 chứa hai phần tử kiểu int {1,2}, mySet2 chứa ba phần tử kiểu int {2,3,4}, và:

```
mySet3 = mySet1 | mySet2;
```

thì mySet3 sẽ chứa các phần tử kiểu int là {1,2,3,4}.

Phép toán "&" và "&="

Phép toán này trả về phép giao (AND) giữa hai đối tượng tập hợp, kết quả cũng là một đối tượng tập hợp:

```
<Đối_tượng_1> = <Đối_tượng_2>  & <Đối_tượng_3>;
<Đối_tượng_1> &= <Đối_tượng_2>;
```

Ví dụ, mySet1 chứa hai phần tử kiểu int {1,2}, mySet2 chứa ba phần tử kiểu int {2,3,4}, và:

```
mySet3 = mySet1 & mySet2;
```

thì mySet3 sẽ chứa các phần tử có kiểu int là {2}.

Phép toán "^" và "^="

Phép toán này trả về phép hợp ngoại (XOR) giữa hai đối tượng tập hợp, kết quả cũng là một đối tượng tập hợp:

```
<Đối_tượng_1> = <Đối_tượng _2>  ^ <Đối_tượng_3>;
<Đối_tượng_1> ^= <Đối_tượng_2>;
```

Ví dụ, mySet1 chứa hai phần tử kiểu int {1,2}, mySet2 chứa ba phần tử kiểu int {2,3,4}, và:

```
mySet3 = mySet1 ^ mySet2;
```

thì mySet3 sẽ chứa các phần tử có kiểu int là {1,3,4}.

Phép toán "-" và "-="

Phép toán này trả về phép hiệu (loại trừ) giữa hai đối tượng tập hợp, kết quả cũng là một đối tượng tập hợp:

```
<Đối_tượng_1> = <Đối_tượng_2>  - <Đối_tượng_3>;
<Đối_tượng_1> -= <Đối_tượng_2>;
```

Ví dụ, mySet1 chứa hai phần tử kiểu int {1,2}, mySet2 chứa ba phần tử kiểu int {2,3,4}, và:

```
mySet3 = mySet1 - mySet2;
```

thì mySet3 sẽ chứa các phần tử có kiểu int là {1}.

7.2.3 Phương thức

Lớp tập hợp có một số phương thức cơ bản sau:

- Thêm một phần tử vào tập hợp
- Loại một phần tử khỏi tập hợp
- Tìm kiếm một phần tử trong tập hợp

Thêm một phần tử vào tập hợp

Có hai cú pháp để thêm một phần tử vào tập hợp:

```
pair<iterator, bool> insert(T&);
iterator insert(<Vị trí con chạy>, T&);
```

Trong đó:

- Phương thức thứ nhất thêm một phần tử vào tập hợp, nếu phần tử đã có mặt trong tập hợp, trả về false và vị trí con chạy của phần tử đó. Nếu phần tử chưa tồn tại, trả về true và vị trí con chạy của phần tử mới thêm vào.
- Phương thức thứ hai cũng thêm vào một phần tử, nhưng chỉ kiểm tra xem phần tử đã tồn tại hay chưa bắt đầu tự vị trí con chạy được chỉ ra trong biến <vị trí con chạy>. Phương thức này trả về vị trí con chạy của phần tử mới thêm vào (nếu thành công) hoặc vị trí của phần tử đã có mặt.

Ví dụ, mySet là một tập hợp kiểu int:

```
mySet.insert(10);
```

sẽ thêm một phần tử có giá trị 10 vào tập hợp.

Lưu ý:

- Do tuân thủ theo lí thuyết tập hợp, nên khi chèn vào tập hợp nhiều lần với cùng một giá trị phần tử. Trong tập hợp chỉ tồn tại duy nhất một giá trị phần tử đó, không được trùng lặp.

Loại một phần tử khỏi tập hợp

Có ba cú pháp để loại bỏ phần tử khỏi tập hợp:

```
int erase(T&);
void erase(<vị trí con chạy>);
void erase(<vị trí bắt đầu>, <vị trí kết thúc>);
```

Trong đó:

- T: là tên kiểu các phần tử của tập hợp.
- Phương thức thứ nhất xoá phần tử có giá trị được chỉ rõ trong tham số đầu vào.
- Phương thức thứ hai loại bỏ phần tử ở vị trí của con chạy được xác định bởi tham số đầu vào.
- Phương thức thứ ba loại bỏ một số phần tử nằm trong phạm vi từ <vị trí bắt đầu> cho đến <vị trí kết thúc> của con chạy.

Ví dụ, mySet là một tập hợp các phần tử có kiểu int:

```
mySet.erase(10);
```

sẽ xoá phần tử có giá trị 10, hoặc:

```
mySet.erase((Set<int>::iterator)10);
```

sẽ xoá phần tử ở vị trí thứ 10 trong tập hợp, hoặc:

```
mySet.erase(mySet.begin(), mySet.end());
```

sẽ xoá toàn bộ các phần tử hiện có của tập hợp mySet.

Tìm kiếm một phần tử trong tập hợp

Có hai cú pháp để tìm kiếm một phần tử trong tập hợp:

```
iterator find(T&);
int count(T&);
```

Trong đó:

- Phương thức thứ nhất tìm phần tử có giá trị xác định bởi tham số đầu vào, kết quả là vị trí con chạy của phần tử đó.
- Phương thứ thứ hai chỉ để kiểm tra xem phần tử có xuất hiện trong tập hợp hay không: trả về 1 nếu có mặt, trả về 0 nếu không có mặt.

Ví dụ, mySet là một tập hợp các phần tử có kiểu int:

```
Set<int>::iterator index = mySet.find(10);
cout << index;
```

sẽ hiển thị vị trí con chạy của phần tử có giá trị 10 trong tập hợp mySet.

7.2.4 Áp dụng

Chương trình 7.1 minh hoạ một số thao tác trên một tập hợp các phần tử có kiểu char: thêm phần tử, loại bỏ phần tử, duyệt các phần tử.

Chương trình 7.1

```
#include<stdio.h>
#include<conio.h>
#include<set.h>
void main(){
      clrscr();
      Set<char> mySet;
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Them mot phan tu vao tap hop" << endl;
            cout << "2: Loai bo mot phan tu khoi tap hop" << endl;
            cout << "3: Xem tat ca cac phan tu cua tap hop" << endl;
            cout << "5: Thoat!" << endl;
            cout << "=====================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':               // Thêm vào
                        char phantu;
                        cout << "Ki tu them vao: ";
                        cin >> phantu;
                        mySet.insert(phantu);
                        break;
                  case '2':               // Loại ra
                        char phantu;
                        cout << "Loai bo ki tu: " << endl;
                        cin >> phantu;
```

```
                        mySet.erase(phantu);
                        break;
                case '3':               // Duyệt
                        cout<<"Cac phan tu cua tap hop la:"<<endl;
                        Set<char>::iterator i;
                        for(i=mySet.begin(); i<mySet.end(); i++)
                                cout << mySet[i] << " ";
                        break;
        }while(function != '5');
    return;
}
```

7.3 LỚP CHUỖI

Lớp chuỗi (String) cũng là một loại lớp chứa, nó chứa một tập các phần tử là một dãy các kí tự có phân biệt thứ tự, các phần tử không nhất thiết phải phân biệt nhau. Muốn sử dụng lớp String, cần thêm vào chỉ thị đầu tệp:

```
#include <string.h>
```

7.3.1 Hàm khởi tạo

Lớp String có ba hàm khởi tạo chính:

- Hàm khởi tạo không tham số:

```
String <tên biến>;
```

- Hàm khởi tạo từ một string khác:

```
String <tên biến>(const String &);
```

- Hàm khởi tạo từ một mảng các kí tự:

```
String <tên biến>(char*, <chiều dài mảng>);
```

Ví dụ:

```
String myStr;
```

là khai báo một chuỗi myStr chưa có phần tử nào (rỗng). Hoặc:

```
String myStr("hello!");
```

là khai báo một chuỗi myStr có các phần tử theo thứ tự là {'h', 'e', 'l', 'l', 'o', '!'}. Hoặc:

```
char* myChar = new char[10];
String myStr(myChar, 10);
```

là khai báo một chuỗi myStr có 10 phần tử tương ứng với các kí tự trong mảng myChar.

Lưu ý:

- Trong trường hợp khởi tạo bằng một chuỗi khác, ta có thể truyền vào một đối số có kiểu cơ bản khác, trình biên dịch sẽ tự động chuyển đổi số đó sang dạng string.

Ví dụ:

```
String myStr(12);
```

thì chuỗi myStr sẽ chứa hai phần tử kí tự{‘1’, ’2’}. Hoặc:

```
String myStr(15.55);
```

thì chuỗi myStr sẽ chứa năm phần tử kí tự {‘1’, ’5’, ’.’, ’5’, ’5’}.

7.3.2 Toán tử

Lớp String có các toán tử cơ bản là:

- Phép gán chuỗi
- Phép cộng chuỗi
- Phép so sánh chuỗi
- Phép vào/ra

Phép gán chuỗi “=”

Cú pháp phép gán chuỗi là tương tự cú pháp gán các đối tượng cơ bản:

```
<Tên biến 1>  = <Tên biến 2>;
```

Ví dụ:

```
String s1(12), s2;
s2 = s1;
```

thì chuỗi s2 cũng chứa hai phần tử như s1 {‘1’, ‘2’}.

Lưu ý:

- Có thể gán trực tiếp các đối tượng cơ bản cho chuỗi:

```
String myStr = 12;      // myStr có hai phần tử {‘1’, ‘2’}
```

- Nhưng phép gán lại có độ ưu tiên thấp hơn phép toán học:

```
String myStr = 12+1.5;  // tương đương myStr = 13.5
```

Phép cộng chuỗi “+” và “+=”

Phép cộng chuỗi sẽ nối chuỗi thứ hai vào sau chuỗi thứ nhất, kết quả cũng là một chuỗi:

```
<Tên biến 1> = <Tên biến 2> + <Tên biến 3>;
<Tên biến 1> += <Tên biến 2>;
```

Ví dụ:

```
String s1(12), s2(3);
s1+= s2;
```

thì s1 sẽ có ba phần tử kí tự {‘1’, ‘2’, ‘3’}.

Phép so sánh chuỗi

Các phép so sánh chuỗi đều là các phép toán hai ngôi, trả về kết quả ở dạng bool (true/false):

- Phép so sánh lớn hơn “>”: `chuỗi_1 > chuỗi_2;`
- Phép so sánh lớn hơn hoặc bằng “>=”: `chuỗi_1 >= chuỗi_2;`
- Phép so sánh nhỏ hơn “<”: `chuỗi_1 < chuỗi_2;`
- Phép so sánh nhỏ hơn hoặc bằng “<=”: `chuỗi_1 <= chuỗi_2;`

- Phép so sánh bằng "==": `chuỗi_1 == chuỗi_2;`
- Phép so sánh khác (không bằng) "!=": `chuỗi_1 != chuỗi_2;`

Lưu ý:

- Phép so sánh chuỗi thực hiện so sánh mã ASCII của từng kí tự ở hai chuỗi theo thứ tự tương ứng cho đến khi có sự khác nhau đầu tiên giữa hai kí tự.
- Phép so sánh là phép so sánh dựa trên từ điển, có phân biệt chữ hoa và chữ thường.

Ví dụ:

```
"12" < "a";        // có giá trị đúng
"a" <= "A";        // có giá trị sai
```

Phép vào/ra

- Phép xuất ra "<<": `cout << biến_chuỗi;`
- Phép nhập vào ">>": `cin >> biến_chuỗi;`

Ví dụ:

```
String s("hello!");
cout << s;
```

sẽ in ra màn hình dòng chữ "hello!".

7.3.3 Phương thức

Lớp chuỗi có một số phương thức cơ bản:

- Lấy chiều dài chuỗi
- Tìm một chuỗi con
- Thêm một chuỗi con
- Xoá một chuỗi con
- Chuyển kiểu kí tự

Lấy chiều dài chuỗi

Cú pháp:

```
<biến_chuỗi>.length();
```

trả về chiều dài của chuỗi (số lượng phần tử kí tự trong chuỗi).

Ví dụ:

```
String s("hello!");
cout << s.length();
```

sẽ in ra màn hình độ dài chuỗi s là 6.

Tìm một chuỗi con

Cú pháp:

```
<biến_chuỗi>.find(<Chuỗi con>, <Vị trí bắt đầu>, <Kiểu so khớp>);
```

Trong đó:

- Tham số thứ nhất là chuỗi con cần tìm.
- Tham số thứ hai là vị trí để bắt đầu tìm, mặc định là bắt đầu tìm từ phần tử có chỉ số 0.
- Tham số thứ ba chỉ ra cách so khớp có phân biệt chữ hoa với chữ thường: SM_IGNORE là không phân biệt, SM_SENSITIVE là có phân biệt.
- Phương thức này trả về kết quả dạng bool, tương ứng là có tìm thấy hay không.

Ví dụ:

```
s.find("12", 0, SM_IGNORE);
```

sẽ tìm trong chuỗi s xem có sự xuất hiện của chuỗi "12" hay không, vị trí bắt đầu tìm là 0, với cách tìm không phân biệt chữ hoa chữ thường.

Thêm một chuỗi con

Cú pháp:

```
<biến_chuỗi>.insert(<vị trí chèn>, <chuỗi con>);
```

Trong đó:

- Tham số thứ nhất là vị trí chỉ số mà tại đó, chuỗi con sẽ được chèn vào
- Tham số thứ hai là chuỗi con cần chèn, chuỗi con này cũng có thể là một kí tự.

Ví dụ:

```
s.insert(0, "12");
```

sẽ chèn vào đầu chuỗi s một chuỗi con có hai phần tử "12".

Xoá một chuỗi con

Cú pháp:

```
<biến_chuỗi>.delete(<vị trí bắt đầu>, <độ dài chuỗi xoá>);
```

Trong đó:

- Tham số thứ nhất là vị trí bắt đầu xoá chuỗi con
- Tham số thứ hai là độ dài chuỗi con bị xoá, giá trị mặc định là 1.

Ví dụ:

```
s.delete(0, 2);
```

sẽ xoá hai kí tự đầu của chuỗi s.

Chuyển kiểu kí tự

- Đổi chuỗi thành các kí tự hoa: `<biến_chuỗi>.toUpper();`
- Đổi chuỗi thành các kí tự thường: `<biến_chuỗi>.toLower();`

Ví dụ:

```
s.toUpper();      // chuyển chuỗi s thành kí tự hoa
s.toLower();      // chuyển chuỗi s thành kí tự thường
```

7.3.4 Áp dụng

Chương trình 7.2 minh hoạ một số thao tác cơ bản trên lớp chuỗi, có sử dụng thư viện lớp chuỗi chuẩn của C++: cộng thêm một chuỗi, chèn thêm một chuỗi con, xoá một chuỗi con, tìm một chuỗi con…

Chương trình 7.2

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
      clrscr();
      String myStr;
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Cong them mot chuoi" << endl;
            cout << "2: Chen them mot chuoi" << endl;
            cout << "3: Xoa di mot chuoi" << endl;
            cout << "4: Tim mot chuoi con" << endl;
            cout << "5: Chuyen thanh chu hoa" << endl;
            cout << "6: Chuyen thanh chu thuong" << endl;
            cout << "7: Xem noi dung chuoi" << endl;
            cout << "8: Xem chieu dai chuoi" << endl;
            cout << "9: Thoat!" << endl;
            cout << "====================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':                  // Thêm vào cuối
                        String subStr;
                        cout << "Chuoi them vao: ";
                        cin >> subStr;
                        myStr += subStr;
                        break;
                  case '2':                  // Chèn vào chuỗi
                        String subStr;
                        int position;
                        cout << "Chuoi them vao: ";
                        cin >> subStr;
                        cout << "Vi tri chen:";
                        cin >> position;
```

```
                        myStr.insert(position, subStr);
                        break;
                case '3':               // Xoá đi một chuỗi con
                        int position, count;
                        cout << "Vi tri bat dau xoa:";
                        cin >> position;
                        cout << "Do dai xoa:";
                        cin >> count;
                        myStr.delete(position, count);
                        break;
                case '4':               // Tìm chuỗi con
                        String subStr;
                        int position;
                        cout << "Chuoi con can tim:";
                        cin >> subStr;
                        cout << "Vi tri bat dau tim:";
                        cin >> position;
                        if(myStr.find(position, subStr))
                                cout << "Co xuat hien!" << endl;
                        else
                                cout << "Khong xuat hien!" << endl;
                        break;
                case '5':               // Chuyển thành chữ hoa
                        myStr.toUpper();
                        cout << myStr << endl;
                        break;
                case '6':               // Chuyển thành chữ thường
                        myStr.toLower();
                        cout << myStr << endl;
                        break;
                case '7':               // Duyệt
                        cout << "Noi dung chuoi:" << endl;
                        cout << myStr << endl;
                        break;
                case '8':               // Duyệt
                        cout << "Chieu dai chuoi:"
                                << myStr.length() << endl;
                        break;
        }while(function != '9');
    return;
}
```

7.4 LỚP NGĂN XẾP VÀ HÀNG ĐỢI

7.4.1 Lớp ngăn xếp

Lớp ngăn xếp (stack) cũng là một loại lớp vật chứa, nó chứa các phần tử cùng kiểu, không bắt buộc phải phân biệt nhau nhưng có phân biệt về thứ tự: các thao tác thêm phần tử và lấy phần tử ra đều được thực hiện ở một đầu ngăn xếp. Phần tử nào được thêm vào trước thì sẽ bị lấy ra sau.

Muốn dùng lớp Stack phải dùng chỉ thị đầu tệp:

```
#include<stack.h>
```

Hàm khởi tạo

Lớp Stack có hai cách khởi tạo:

- Khởi tạo không tham số:

  ```
  Stack<T> biến_ngăn_xếp;
  ```

- Khởi tạo bằng một ngăn xếp khác, có cùng kiểu phần tử:

  ```
  Stack<T> biến_ngăn_xếp(Stack<T>);
  ```

Trong đó:

- **T**: là kiểu của các phần tử chứa trong ngăn xếp. T có thể là các kiểu cơ bản, cũng có thể là các kiểu phức tạp do người dùng tự định nghĩa.

Ví dụ:

```
Stack<int> myStack;
```

là khai báo một biến myStack, chứa các phần tử có kiểu cơ bản int.

Toán tử

Lớp Stack chỉ dùng đến các toán tử gán "=" và toán tử so sánh bằng "==":

- Phép gán "=":

  ```
  <ngăn xếp 1> = <ngăn xếp 2>;
  ```

  Dùng để gán hai đối tượng ngăn xếp.

- Phép so sánh bằng "==":

  ```
  <ngăn xếp 1> == <ngăn xếp 2>;
  ```

  Dùng để kiểm tra xem hai đối tượng ngăn xếp có bằng nhau hay không. Kết quả trả về có kiểu bool (true/false).

Phương thức

- Thêm một phần tử:

  ```
  <biến ngăn xếp>.push(T);
  ```

sẽ thêm một phần tử có kiểu T vào đỉnh ngăn xếp.

- Loại một phần tử:

  ```
  <biến ngăn xếp>.pop();
  ```

sẽ trả về phần tử đang nằm ở đỉnh ngăn xếp.

- Kiểm tra tính rỗng:

```
<biến ngăn xếp>.empty();
```

sẽ trả về kết quả kiểu bool, tương ứng với trạng thái của ngăn xếp có rỗng hay không.

- Kích thước ngăn xếp:

```
<biến ngăn xếp>.size();
```

sẽ trả về số lượng các phần tử hiện đang có mặt trong ngăn xếp.

Áp dụng

Trong phần này, ta sẽ viết lại chương trình 3.4c đã được minh hoạ trong phần viết về cấu trúc ngăn xếp (chương 3). Nhưng thay vì phải định nghĩa cấu trúc ngăn xếp, ta dùng lớp ngăn xếp của thư viện C++: đảo ngược một xâu kí tự được nhập vào từ bàn phím.

Chương trình 7.3

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stack.h>

void main(){
      clrscr();
      Stack<char> myStack;
      char strIn[250];
      cout << "Nhap chuoi: ";
      cin >> strIn;      // Nhập chuỗi kí tự từ bàn phím
      for(int i=0; i<strlen(strIn); i++)  // Đặt vào ngăn xếp
            myStack.push(strIn[i]);
      while(!myStack.empty())          // Lấy ra từ ngăn xếp
            cout << myStack.pop();
      return;
}
```

7.4.2 Lớp hàng đợi

Lớp hàng đợi (queue) cũng là một loại lớp vật chứa, nó chứa các phần tử cùng kiểu, không bắt buộc phải phân biệt nhau nhưng có phân biệt về thứ tự: các thao tác thêm phần tử được thực hiện ở một đầu, các thao tác lấy phần tử ra được thực hiện ở một đầu còn lại của hàng đợi. Phần tử nào được thêm vào trước thì sẽ bị lấy ra trước.

Muốn dùng lớp Queue phải dùng chỉ thị đầu tệp:

```
#include<queue.h>
```

Hàm khởi tạo

Lớp Queue có hai cách khởi tạo:

- Khởi tạo không tham số:

```
Queue<T> biến_hàng_đợi;
```

- Khởi tạo bằng một hàng đợi khác, có cùng kiểu phần tử:

```
Queue<T> biến_hàng_đợi(Queue<T>);
```

Trong đó:

- **T**: là kiểu của các phần tử chứa trong hàng đợi. T có thể là các kiểu cơ bản, cũng có thể là các kiểu phức tạp do người dùng tự định nghĩa.

Ví dụ:

```
Queue<int> myQueue;
```

là khai báo một biến myQueue, chứa các phần tử có kiểu cơ bản int.

Toán tử

Lớp Queue chỉ dùng đến các toán tử gán "=" và toán tử so sánh bằng "==":

- Phép gán "=":

```
<hàng đợi 1> = <hàng đợi 2>;
```

Dùng để gán hai đối tượng hàng đợi.

- Phép so sánh bằng "==":

```
<hàng đợi 1> == <hàng đợi 2>;
```

Dùng để kiểm tra xem hai đối tượng hàng đợi có bằng nhau hay không. Kết quả trả về có kiểu bool (true/false).

Phương thức

- Thêm một phần tử:

```
<biến hàng đợi>.push(T);
```

sẽ thêm một phần tử có kiểu T vào cuối hàng đợi.

- Loại một phần tử:

```
<biến hàng đợi>.pop();
```

sẽ trả về phần tử đang nằm ở đỉnh đầu hàng đợi.

- Kiểm tra tính rỗng:

```
<biến hàng đợi>.empty();
```

sẽ trả về kết quả kiểu bool, tương ứng với trạng thái của hàng đợi có rỗng hay không.

- Kích thước hàng đợi:

```
<biến hàng đợi>.size();
```

sẽ trả về số lượng các phần tử hiện đang có mặt trong hàng đợi.

Áp dụng

Trong phần này, ta sẽ cài đặt lại chương trình trong phần cấu trúc hàng đợi (chương 3). Nhưng thay vì phải tự định nghĩa cấu trúc hàng đợi, ta dùng lớp Queue của thư viện C++ để mô phỏng chương trình quản lí tiến trình của hệ điều hành.

Chương trình 7.4

```
#include<stdio.h>
#include<conio.h>
#include<queue.h>

void main(){
      clrscr();
      Queue<int> myQueue;
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Them mot tien trinh vao hang doi" << endl;
            cout << "2: Dua mot tien trinh trinh vao thuc hien" << endl;
            cout<<"3: Xem tat ca cac tien trinh trong hang doi"<< endl;
            cout << "5: Thoat!" << endl;
            cout << "=======================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':               // Thêm vào hàng đợi
                        int maso;
                        cout << "Ma so tien trinh vao hang doi: ";
                        cin >> maso;
                        myQueue.push(maso);
                        break;
                  case '2':               // Lấy ra khỏi hàng đợi
                        cout << "Tien trinh duoc thuc hien: " <<
                              myQueue.pop() << endl;
                        break;
                  case '3':               // Duyệt hàng đợi
                        Queue<int>::iterator i;
                        for(i=myQueue.begin(); i<myQueue.end(); i++)
                              cout << myQueue[i] << " ";
                        break;
            }while(function != '5');
      return;
}
```

7.5 LỚP DANH SÁCH LIÊN KẾT

Lớp danh sách liên kết (List) cũng là một kiểu lớp vật chứa, nó chứa các phần tử cùng kiểu, có tính đến thứ tự. Muốn sử dụng lớp List của thư viện C++, phải khai báo chỉ thị đầu tệp:

```
#include<list.h>  // Dành riêng cho lớp List
```

hoặc:

```
#include<stl.h>   // Dùng chung cho các lớp vật chứa
```

7.5.1 Hàm khởi tạo

Lớp List có ba kiểu khởi tạo:

- Khởi tạo không tham số:

  ```
  List<T> biến_danh_sách;
  ```

- Khởi tạo tử một danh sách cùng kiểu:

  ```
  List<T> biến_danh_sách(List<T>);
  ```

- Khởi tạo từ một mảng các phần tử:

  ```
  List<T> biến_danh_sách(T* <Mảng phần tử>, int <chiều dài mảng>);
  ```

Trong đó:

- **T**: là kiểu của các phần tử chứa trong danh sách liên kết. T có thể là các kiểu cơ bản, cũng có thể là các kiểu phức tạp do người dùng tự định nghĩa.

Ví dụ:

```
List<int> myList;
```

sẽ khai báo một danh sách liên kết myList, các phần tử của nó có kiểu cơ bản int.

7.5.2 Toán tử

Toán tử gán "="

Cú pháp:

```
<danh sách 1> = <danh sách 2>;
```

sẽ copy toàn bộ các phần tử của <danh sách 2> vào <danh sách 1>.

Toán tử so sánh bằng "=="

Cú pháp:

```
<danh sách 1> = <danh sách 2>;
```

sẽ trả về một giá trị kiểu bool, tương ứng với việc hai danh sách này có bằng nhau hay không. Việc so sánh được tiến hành trên từng phần tử ở vị trí tương ứng nhau.

Lưu ý:

- Ngoài ra còn có các phép toán so sánh khác cũng có thể thực hiện trên danh sách: <, >, <=, >=, !=.

7.5.3 Phương thức

Thêm một phần tử vào danh sách

Cú pháp:

```
<biến danh sách>.insert(<vị trí chèn>, <phần tử>);
<biến danh sách>.push_front(<phần tử>);
<biến danh sách>.push_back(<phần tử>);
```

Trong đó:

- Phương thức thứ nhất chèn một phần tử vào một vị trí bất kì của danh sách, vị trí chèn được chỉ ra bởi tham số thứ nhất.
- Phương thức thứ hai chèn một phần tử vào đầu danh sách
- Phương thức thứ ba chèn một phần tử vào cuối danh sách.

Ví dụ:

```
List<int> myList;
myList.push_front(7);
```

sẽ chèn vào đầu danh sách myList một phần tử có giá trị là 7.

Xoá đi một phần tử

Cú pháp:

```
<biến danh sách>.erase(<vị trí xoá>);
<biến danh sách>.pop_front();
<biến danh sách>.pop_back();
```

Trong đó:

- Phương thức thứ nhất xóa một phần tử ở một vị trí bất kì của danh sách, vị trí xoá được chỉ ra bởi tham số thứ nhất.
- Phương thức thứ hai xoá một phần tử ở đầu danh sách
- Phương thức thứ ba xoá một phần tử ở cuối danh sách.

Ví dụ:

```
List<int> myList;
cout << myList.pop_front();
```

sẽ in ra màn hình phần tử đầu của danh sách myList.

Kiểm tra tính rỗng của danh sách

Cú pháp:

```
<biến danh sách>.empty();
```

trả về giá trị bool, tương ứng với trạng thái hiện tại của biến danh sách là rỗng hay không.

Xem kích thước danh sách

Cú pháp:

```
<biến danh sách>.size();
```

Ví dụ:

```
cout << myList.size();
```

sẽ in ra màn hình kích cỡ (số lượng các phần tử) của danh dách.

Xem nội dung một phần tử

Cú pháp:

```
<biến danh sách>.get();
<biến danh sách>.next();
```

Trong đó:

- Phương thức thứ nhất trả về phần tử ở vị trí hiện tại của con chạy
- Phương thức thứ hai trả về phần tử tiếp theo, và con chạy cũng di chuyển sang phần tử đó.

Ví dụ:

```
List<int> myList;
cout << myList.get();
```

sẽ in ra màn hình nội dung phần tử thứ nhất của danh sách myList.

7.5.4 Áp dụng

Trong phần này, ta cài đặt lại chương trình 3.6c đã được trình bày trong phần cấu trúc danh sách liên kết (chương 3). Nhưng thay vì tự tạo danh sách liên kết, ta dùng lớp List trong thư viện của C++ để quản lí nhân viên văn phòng.

Chương trình 7.5

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<list.h>

typedef struct{
      char name[25];                    // Tên nhân viên
      int age;                          // Tuổi nhân viên
      float salary;                           // Lương nhân viên
} Employee;
void main(){
      clrscr();
      List<Employee> myList;
      int function;
      do{
            clrscr();
            cout << "CAC CHUC NANG:" << endl;
            cout << "1: Them mot nhan vien" << endl;
            cout << "2: Xoa mot nhan vien" << endl;
            cout << "3: Xem tat ca cac nhan vien trong phong" << endl;
```

```
            cout << "5: Thoat!" << endl;
            cout << "====================================" << endl;
            cout << "Chon chuc nang: " << endl;
            cin >> function;
            switch(function){
                  case '1':               // Thêm vào ds
                        int position;
                        Employee employee;
                        cout << "Vi tri can chen: ";
                        cin >> position;
                        cout << "Ten nhan vien: ";
                        cin >> employee.name;
                        cout << "Tuoi nhan vien: ";
                        cin >> employee.age;
                        cout << "Luong nhan vien: ";
                        cin >> employee.salary;
                        myList.insert(position, employee);
                        break;
                  case '2':               // Lấy ra khỏi ds
                        int position;
                        cout << "Vi tri can xoa: ";
                        cin >> position;
                        Employee result = myList.erase(position);
                        if(result != NULL){
                              cout << "Nhan vien bi loai: " endl;
                              cout << "Ten: " << result.name << endl;
                              cout << "Tuoi: " << result.age << endl;
                              cout << "Luong: " << result.salary << endl;
                        }
                        break;
                  case '3':               // Duyệt ds
                        cout << "Cac nhan vien cua phong:" << endl;
                        Employee result = myList.front();
                        do{
                              cout << result.name << " "
                                    << result.age << " "
                                    << result.salary << endl;
                              result = myList.next();
                        }while(result != NULL);
                        break;
            }while(function != '5');
      return;
}
```

TỔNG KẾT CHƯƠNG 7

Nội dung chương 7 đã trình bày một số lớp cơ bản của lớp vật chứa (Container) trong thư viện của C++:

- Lớp tập hợp (Set)
- Lớp chuỗi (String)
- Lớp ngăn xếp (Stack)
- Lớp hàng đợi (Queue)
- Lớp danh sách liên kết (List)

Hầu hết các lớp vật chứa này đều có thể chứa các phần tử với kiểu bất kì: có thể là kiểu cơ bản, cũng có thể là kiểu phức tạp do người dùng tự định nghĩa. Ngoại trừ lớp chuỗi, chỉ chứa các phần tử có kiểu char.

Các lớp vật chứa này có một số phương thức tương tự nhau, nhưng cách thực hiện lại khác nhau:

- Thêm vào một phần tử
- Lấy ra một phần tử
- Kiểm tra tính rỗng
- Kiểm tra số lượng phần tử.

Người dùng có thể áp dụng các lớp có sẵn trong thư viện này của C++ để giải quyết các bài toán khác nhau mà không phải tự định nghĩa lại các lớp.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 7

1. Dùng thư viện lớp stack để kiểm tra một số tự nhiên nhập vào từ bàn phím có phải là một số parlindom hay không: Một số tự nhiên là parlindom nếu đổi ngược thứ tự các chữ số, ta vẫn thu được chính số đó, ví dụ, 87278 là một số như vậy.
2. Viết lại chương trình 7.1 dùng lớp tập hợp, nhưng dùng kiểu của các phần tử là lớp Car đã được định nghĩa trong chương 5.
3. Dùng thư viện lớp String để viết chương trình chia nhỏ một chuỗi ban đầu thành một số chuỗi con, ranh giới phân chia là một chuỗi con được nhập vào từ bàn phím. Ví dụ “abc acb”, mà chia nhỏ theo chuỗi con “c” sẽ thu được các chuỗi: “ab”, “ca”, “cb”.
4. Dùng thư viện lớp String để viết chương trình thay thế các chuỗi con của một chuỗi ban đầu bằng một chuỗi con khác. Các chuỗi con được nhập từ bàn phím.
5. Viết lại chương trình 7.5 , vẫn dùng thư viện lớp List, nhưng thay thế kiểu phần tử bằng kiểu lớp Car đã được định nghĩa trong chương 5.

HƯỚNG DẪN TRẢ LỜI CÂU HỎI VÀ BÀI TẬP

Chương 1

Chương 2

1. c và e.
2. a và d.
3. c.
4. b.
5. c.
6. c.
7. d.
8. b và c.
9. b.
10. c.
11. d.
12. c.
13. d.

Chương 3

1. a.
2. c.
3. a.
4. b.
5. b và d.
6. Gợi ý:

```
struct Subject{
      char subject[20];
      float note;
};
```

Hoặc:

```
typedef struct {
      char subject[20];
      float note;
}Subject;
```

7. Gợi ý:

```
typedef struct {
      char name[20];
```

```
        int age;
        char class[20];
        Subject* subjects;
        char type[20];
}Student;
```

Chương 4

1. a.
2. b.
3. c.
4. a.
5. a.
6. b.
7. c.
8. d.
9. a.
10. b.
11. c.

Chương 5

1. a và c.
2. b.
3. d.
4. a, b và c.
5. c.
6. b và c.
7. b và c.
8. b, d và e.
9. c và d.
10. a và d.
11. a, b, c và d.
12. Gợi ý (từ bài 12 đến bài 17):

```
class Employee{
        char* name;
        int age;
        float salary;
        public:
                Employee();
                Employee(char*    nameIn="",    int    ageIn=18,    float
salaryIn=100);
```

```
            void setName(char*);
            char* getName();
            void setAge(int);
            int getAge();
            void setSalary(float);
            float getSalary();
            void show();
            ~Employee();
};
```

Chương 6

1. b.
2. d.
3. c.
4. b, d và f.
5. b và d.
6. d.
7. a, b, c và d.
8. a và c.
9. d.
10. d.
11. b.
12. b.
13. c và d.
14. a, b, c và d.
15. c và d.
16. b và c.
17. Gợi ý (từ bài 17 đến bài 25):

```
class Human{
      char* name;
      int age;
      int sex;
      public:
            Human();
            Human(char*, int, int);
            void setName(char*);
            char* getName();
            void setAge(int);
            int getAge();
            void setSex(int);
```

```
                int getSex();
                void show();
                ~Human();
};
class Person: public virtual Human{
        char* address;
        char* phone;
        public:
                Person();
                Person(char*, int, int, char*, char*);
                void setAddress(char*);
                char* getAddress();
                void setPhone(char*);
                char* getPhone();
                void show();
                ~Person();
};
class Worker: public virtual Human{
        int hour;
        float salary;
        public:
                Worker();
                Worker(int, float);
                void setHour(int);
                int getHour();
                void setSalary(float);
                float getSalary();
                void show();
};
class Employee: public Person, public Worker{
        char* position;
        public:
                Employee();
                Employee(char*,  int,  int,  char*,  char*,  int,  float,
char*);
                void setPosition(char*);
                char* getPosition();
                void virtual show();
                ~Employee();
};
```

Chương 7

2. Gợi ý

```
void main(){
	clrscr();
	Set<Car> mySet;
	int function;
	do{
		clrscr();
		cout << "CAC CHUC NANG:" << endl;
		cout << "1: Them mot phan tu vao tap hop" << endl;
		cout << "2: Loai bo mot phan tu khoi tap hop" << endl;
		cout << "3: Xem tat ca cac phan tu cua tap hop" << endl;
		cout << "5: Thoat!" << endl;
		cout << "=========================================" << endl;
		cout << "Chon chuc nang: " << endl;
		cin >> function;
		switch(function){
			case '1':                    // Thêm vào
				Car car = new Car();
				cout << "O to them vao: " << endl;
				cout << "Toc do: ";
				int speed;
				cin >> speed;
				car.setSpeed(speed);
				cout << "Nhan hieu:";
				char[] mark;
				cin >> mark;
				car.setMark(mark);
				float price;
				cout << "Gia xe:";
				cin >> price;
				car.setPrice(price);
				mySet.insert(car);
				break;
			case '2':                    // Loại ra
				Set<Car>::iterator index;
				cout << "Loai bo xe o vi tri: ";
				cin >> index;
				if(index >= mySet.begin())&&(index < mySet.end())
					mySet.erase(mySet[index]);
				break;
			case '3':                    // Duyệt
				cout<<"Cac phan tu cua tap hop la:"<<endl;
				Set<char>::iterator i;
				for(i=mySet.begin(); i<mySet.end(); i++){
					cout << mySet[i].getSpeed() << " ";
					cout << mySet[i].getMark() << " ";
					cout << mySet[i].getPrice() << " ";
				}
```

```
                        break;
                }while(function != '5');
        return;
}
```

5. Gợi ý

```
void main(){
        clrscr();
        List<Car> myList;
        int function;
        do{
                clrscr();
                cout << "CAC CHUC NANG:" << endl;
                cout << "1: Them mot xe o to" << endl;
                cout << "2: Xoa mot xe o to" << endl;
                cout << "3: Xem tat ca cac o to" << endl;
                cout << "5: Thoat!" << endl;
                cout << "======================================" << endl;
                cout << "Chon chuc nang: " << endl;
                cin >> function;
                switch(function){
                        case '1':               // Thêm vào ds
                                int possition;
                                Car car = new Car();
                                cout << "Vi tri can chen: ";
                                cin >> possition;
                                cout << "Toc do: ";
                                int speed;
                                cin >> speed;
                                car.setSpeed(speed);
                                cout << "Nhan hieu:";
                                char[] mark;
                                cin >> mark;
                                car.setMark(mark);
                                float price;
                                cout << "Gia xe:";
                                cin >> price;
                                car.setPrice(price);
                                myList.insert(possition, car);
                                break;
                        case '2':               // Lấy ra khỏi ds
                                int possition;
                                cout << "Vi tri can xoa: ";
                                cin >> possition;
                                Car result = myList.erase(possition);
                                if(result != NULL){
                                        cout << "O to bi loai: " endl;
                                        cout<<"Toc do:"<<result.getSpeed()<<endl;
```

```
                                cout << "Nhan hieu: " << result.getMark()
                                     << endl;
                                cout<<"Gia: "<<result.getPrice()<<endl;
                        }
                        break;
                case '3':                   // Duyệt ds
                        cout << "Cac o to hien co:" << endl;
                        Car result = myList.front();
                        do{
                                cout << result.getSpeed() << " "
                                     << result.getMark() << " "
                                     << result.getPrice() << endl;
                                result = myList.next();
                        }while(result != NULL);
                        break;
        }while(function != '5');
    return;
}
```

TÀI LIỆU THAM KHẢO

Tài liệu tiếng Anh

[1] James P. Cohoon and Jack W.Davidson, ***C++ Program Design – An Introduction to Programming and Object-Oriented Design***, 2nd edition, WCB McGraw-Hill, 1999.

[2] Nell Dale, Chip Weems and Mark Headington, ***Programming and Problem Solving with C++,*** John & Barlett Publisher, 1996.

[3] Michael T. Goodrich, Roberto Tamassia and David Mount, ***Data Structures and Algorithms in C++,*** John Wiley & Sons Inc, 2004.

[4] Scott R.Ladd, ***C++ Components and Algorithms***. 2nd edition. . M&T Books, 1994.

[5] Scott R.Ladd, ***C++ Templates and Tools,*** M&T Books, 1994

[6] Robert Lafore, ***Object – Oriented Programming in C++***, Fourth edition, SAMS, 2001.

Tài liệu tiếng Việt

[1] Lê Đ. Hưng, Tạ T. Anh, Nguyễn H. Đức và Nguyễn T. Thuỷ, ***Lập trình hướng đối tượng với C++***, NXB Khoa học và Kỹ thuật, 2005.

[2] Nguyễn T. Thuỷ, Tạ T. Anh, Nguyễn Q. Huy và Nguyễn H. Đức, ***Bài tập lập trình hướng đối tượng với C++***, NXB Khoa học và Kỹ thuật, 2004.

Các địa chỉ web

1. http://www.angelfire.com/country/aldev0/cpphowto/
2. http://www.gnacademy.org/text/cc/Tutorial/tutorial.html
3. http://sophia.dtp.fmph.uniba.sk/cpptut/tutorial.us.html
4. http://www.brpreiss.com/books/opus4/html/book.html
5. http://www.fredosaurus.com/notes-cpp/index.html

MỤC LỤC

NGÔN NGỮ LẬP TRÌNH C++

Mã số : 412LTC340

Chịu trách nhiệm bản thảo

TRUNG TÂM ĐÀO TẠO BƯU CHÍNH VIỄN THÔNG 1

Trường Đại học Khoa học Tự nhiên
Khoa Công nghệ thông tin
Bộ môn Tin học cơ sở

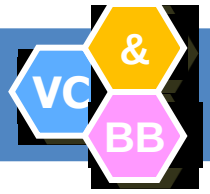
NHẬP MÔN LẬP TRÌNH

Đặng Bình Phương
dbphuong@fit.hcmuns.edu.vn

GIỚI THIỆU
NGÔN NGỮ LẬP TRÌNH C

Nội dung

1. Giới thiệu
2. Bộ từ vựng của C
3. Cấu trúc chương trình C
4. Một số ví dụ minh họa



Giới thiệu

Giới thiệu

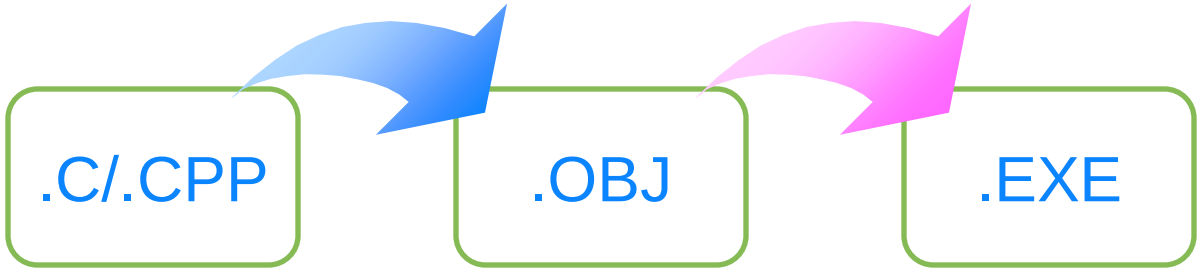
- Dennis Ritchie tại Bell Telephone năm 1972.
- Tiền thân của ngôn ngữ B, KenThompson, cũng tại Bell Telephone.
- Là ngôn ngữ lập trình có cấu trúc và phân biệt chữ Hoa - thường (case sensitive)
- ANSI C.

Giới thiệu

- Ưu điểm của C
  - Rất mạnh và linh động, có khả năng thể hiện bất cứ ý tưởng nào.
  - Được sử dụng rộng rãi bởi các nhà lập trình chuyên nghiệp.
  - Có tính khả chuyển, ít thay đổi trên các hệ thống máy tính khác nhau.
  - Rõ ràng, cô đọng.
  - Lập trình đơn thể, tái sử dụng thông qua hàm.

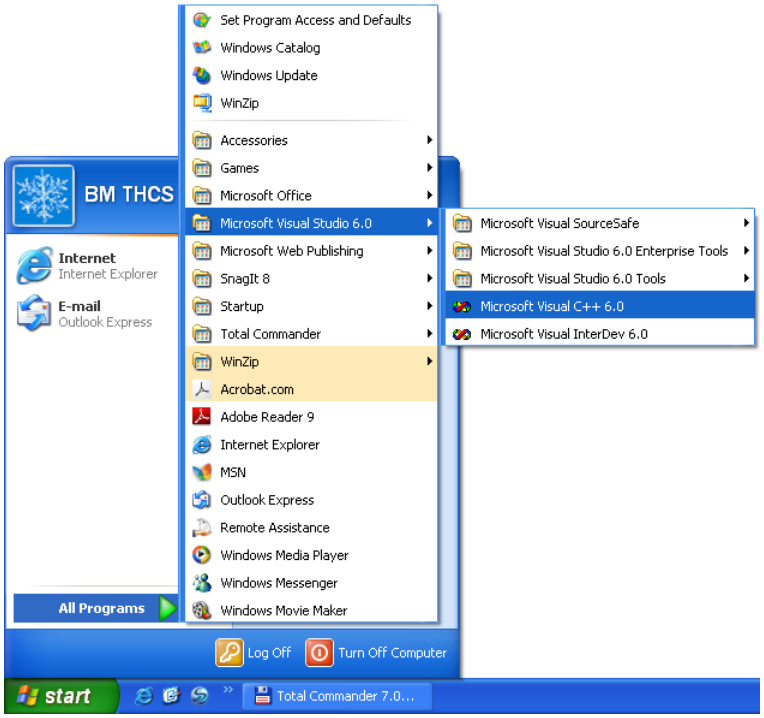
Giới thiệu

- Môi trường phát triển tích hợp IDE (Integrated Development Environment)
  - Biên tập chương trình nguồn (Trình EDIT).
  - Biên dịch chương trình (Trình COMPILE).
  - Chạy chương trình nguồn (Trình RUNTIME).
  - Sửa lỗi chương trình nguồn (Trình DEBUG).



Giới thiệu

- Môi trường lập trình
  - Borland C++ 3.1 for DOS.
  - Visual C++ 6.0, Win32 Console Application.



Bộ từ vựng của C

- Các ký tự được sử dụng
  - Bộ chữ cái 26 ký tự Latinh A, B, C, …, Z, a, b, c, …, z
  - Bộ chữ số thập phân : 0, 1, 2, …, 9
  - Các ký hiệu toán học : + – * / = < > ( )
  - Các ký tự đặc biệt : . , : ; [ ] % \ # $ ‘
  - Ký tự gạch nối _ và khoảng trắng ‘ ’

Bộ từ vựng của C

- Từ khóa (keyword)
  - Các từ dành riêng trong ngôn ngữ.
  - Không thể sử dụng từ khóa để đặt tên cho biến, hàm, tên chương trình con.
  - Một số từ khóa thông dụng:
    - const, enum, signed, struct, typedef, unsigned…
    - char, double, float, int, long, short, void
    - case, default, else, if, switch
    - do, for, while
    - break, continue, goto, return

Bộ từ vựng của C

- Tên/Định danh (Identifier)
  - Một dãy ký tự dùng để chỉ tên một hằng số, hằng ký tự, tên một biến, một kiểu dữ liệu, một hàm một hay thủ tục.
  - Không được trùng với các từ khóa và được tạo thành từ các chữ cái và các chữ số nhưng bắt buộc chữ đầu phải là chữ cái hoặc _.
  - Số ký tự tối đa trong một tên là 255 ký tự và được dùng ký tự _ chen trong tên nhưng không cho phép chen giữa các khoảng trắng.

Bộ từ vựng của C

- Ví dụ Tên/Định danh (Identifier)
  - Các tên hợp lệ: GiaiPhuongTrinh, Bai_Tap1
  - Các tên không hợp lệ: 1A, Giai Phuong Trinh
  - Phân biệt chữ hoa chữ thường, do đó các tên sau đây khác nhau:
    - A, a
    - BaiTap, baitap, BAITAP, bAItaP, …

Bộ từ vựng của C

- Dấu chấm phẩy ;
  - Dùng để phân cách các câu lệnh.
  - Ví dụ: printf(“Hello World!”); printf(“\n”);
- Câu chú thích
  - Đặt giữa cặp dấu /* */ hoặc // (C++)
  - Ví dụ: /*Ho & Ten: NVA*/, // MSSV: 0712078
- Hằng ký tự và hằng chuỗi
  - Hằng ký tự: ‘A’, ‘a’, …
  - Hằng chuỗi: “Hello World!”, “Nguyen Van A”
  - Chú ý: ‘A’ khác “A”

Cấu trúc chương trình C

```
#include "…";   // Khai báo file tiêu đề

int x;          // Khai báo biến hàm
void Nhap();    // Khai báo hàm

void main()     // Hàm chính
{
     // Các lệnh và thủ tục
}
```

Ví dụ

```
#include <stdio.h>
#include <conio.h>

void main()
{
      int x, y, tong;
      printf("Nhap hai so nguyen: ");
      scanf("%d%d", &x, &y);
      tong = x + y;
      printf("Tong hai so la %d", tong);
      getch();
}
```

Bài tập lý thuyết

1. Tên (định danh) nào sau đây đặt không hợp lệ, tại sao?
   - Tin hoc co SO A, 1BaiTapKHO
   - THucHaNH, NhapMon_L@pTrinH
2. Câu ghi chú dùng để làm gì? Cách sử dụng ra sao? Cho ví dụ minh họa.
3. Trình bày cấu trúc của một chương trình. Giải thích ý nghĩa của từng phần trong cấ trúc.

Bài giảng

Ngôn ngữ lập trình C

MỤC LỤC

BÀI MỞ ĐẦU

1. Lịch sử phát triển ngôn ngữ lập trình C

C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennish Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennish Ritchie và B.W Kernighan đã cho xuất bản quyển "Ngôn ngữ lập trình C" và được phổ biến rộng rãi đến nay.

Lúc ban đầu, C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các công việc lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của công việc lập trình, C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình để rồi các công ty lập trình sử dụng một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ C và chuẩn ANSI C cũng được khai sinh từ đó.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất "mềm dẻo", có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng khác. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp "rắc rối" là "hơi khó hiểu" do sự "mềm dẻo" của C. Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

Ngôn ngữ C có những đặc điểm cơ bản sau:

- *Tính cô đọng (compact)*: C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.
- *Tính cấu trúc (structured)*: C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp… Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.
- *Tính tương thích (compatible)*: C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.
- *Tính linh động (flexible)*: C là một ngôn ngữ rất uyển chuyển và cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
- *Biên dịch (compile)*: C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

2. Một số khái niệm dùng trong ngôn ngữ lập trình C.

2.1 Tập kí tự dùng trong ngôn ngữ C.

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Các từ lại được liên kết với nhau theo một qui tắc nào đó để tạo nên các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và thể hiện một thuật toán để giải một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau :

26 chữ cái hoa : A B C .. Z

26 chữ cái thường : a b c .. z

10 chữ số : 0 1 2 .. 9

Các ký hiệu toán học : + - * / = ( )

Ký tự gạch nối : _

Các ký tự khác : . , : ; [ ] {} ! \ & % # $ ...

Dấu cách (space) dùng để tách các từ. Ví dụ chữ VIET NAM có 8 ký tự, còn VIETNAM chỉ có 7 ký tự.

Chú ý :

Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên.

Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2+bx+c=0$, ta cần tính biệt thức Delta $\Delta= b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự Δ, vì vậy ta phải dùng ký hiệu khác để thay thế.

2.2 Tên, từ khóa.

2.2.1 Từ khóa

Từ khóa là các từ dành riêng (reserved words) của C mà người lập trình có thể sử dụng nó trong chương trình tùy theo ý nghĩa của từng từ. Ta không được dùng từ khóa để đặt cho các tên của riêng mình. Các từ khóa của Turbo C 3.0 bao gồm:

asm	auto	break	case	cdecl	char
class	const	continue	_cs	default	delete
do	double	_ds	else	enum	_es
extern	_export	far	_fastcal	float	for
friend	goto	huge	if	inline	int
interr	_loadds	long	near	new	operat
pascal	private	protecte	public	register	return
_save	_seg	short	signed	sizeof	_ss
static	struct	switch	templat	this	typed
union	unsigne	virtual	void	volatile	while

2.2.2 Tên

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn,...

Tên được đặt theo qui tắc sau :

Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của

tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng : Option-Compiler-Source-Identifier length khi dùng TURBO C.

Ví dụ :

Các tên đúng :

a_1 delta x1 _step GAMA

Các tên sai :

3MN	Ký tự đầu tiên là số
m#2	Sử dụng ký tự #
f(x)	Sử dụng các dấu ( )
do	Trùng với từ khoá
te ta	Sử dụng dấu trắng
Y-3	Sử dụng dấu -

Chú ý :

Trong TURBO C, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. trong C, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

BÀI 1: TỔNG QUAN VỀ NGÔN NGỮ C

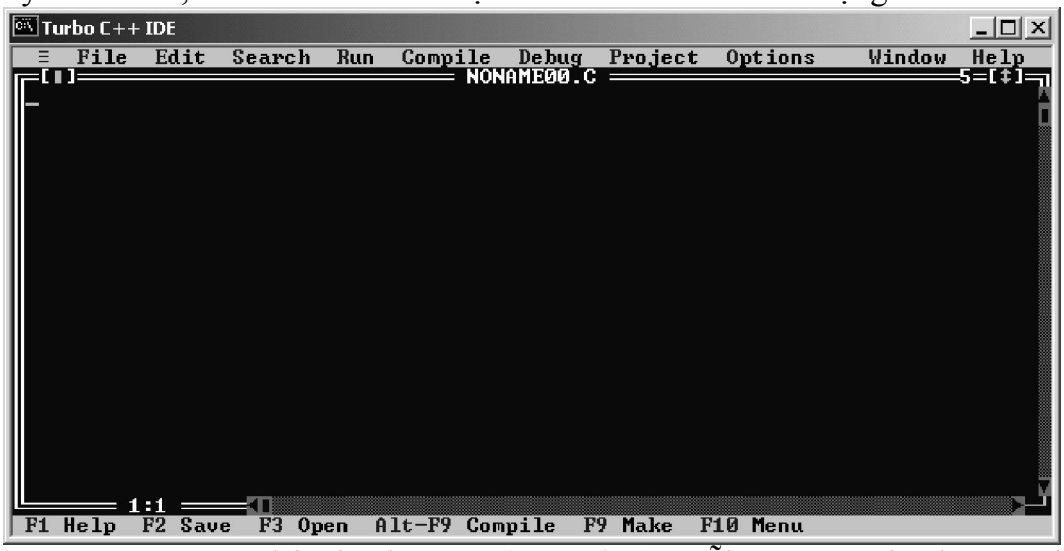
Mục tiêu:
Học viên sau khi học xong có khả năng:
– Biết cách đưa dữ liệu vào từ bàn phím và xuất dữ liệu lên màn hình;
– Sử dụng được hệ thống trợ giúp từ help file;
– Biết cách khởi động và thoát khỏi chương trình;

1. Các thao tác cơ bản

1.1 Khởi động và thoát khỏi môi trường C.

1.1.1 Khởi động

Chạy Turbo C cũng giống như chạy các chương trình khác trong môi trường DOS hay Windows, màn hình sẽ xuất hiện menu của Turbo C có dạng như sau:



Dòng trên cùng gọi là thanh menu (menu bar). Mỗi mục trên thanh menu lại có thể có nhiều mục con nằm trong một menu kéo xuống.

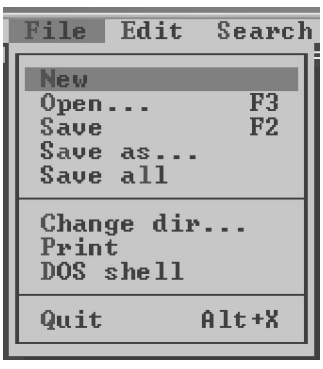
Dòng dưới cùng ghi chức năng của một số phím đặc biệt. Chẳng hạn khi gõ phím F1 thì ta có được một hệ thống trợ giúp mà ta có thể tham khảo nhiều thông tin bổ ích.

Muốn vào thanh menu ngang ta gõ phím F10. Sau đó dùng các phím mũi tên qua trái hoặc phải để di chuyển vùng sáng tới mục cần chọn rồi gõ phím Enter. Trong menu kéo xuống ta lại dùng các phím mũi tên lên xuống để di chuyển vùng sáng tới mục cần chọn rồi gõ Enter.

Ta cũng có thể chọn một mục trên thanh menu bằng cách giữ phím Alt và gõ vào một ký tự đại diện của mục đó (ký tự có màu sắc khác với các ký tự khác). Chẳng hạn để chọn mục File ta gõ Alt-F (F là ký tự đại diện của File).

1.1.2 Thoát khỏi

Dùng File/Quit hoặc Alt-X



1.2 Lưu và Mở file.

1.2.1 Lưu file

Sử dụng File/Save hoặc gõ phím F2. Có hai trường hợp xảy ra:

- Nếu chương trình chưa được ghi lần nào thì một hội thoại sẽ xuất hiện cho phép bạn xác định tên tập tin (FileName). Tên tập tin phải tuân thủ quy cách đặt tên của DOS và không cần có phần mở rộng (sẽ tự động có phần mở rộng là .C hoặc .CPP

sẽ nói thêm trong phần Option). Sau đó gõ phím Enter.

- Nếu chương trình đã được ghi một lần rồi thì nó sẽ ghi những thay đổi bổ sung lên tập tin chương trình cũ.

Chú ý: Để đề phòng mất điện trong khi soạn thảo chương trinh thỉnh thoảng bạn nên gõ phím F2.

Quy tắc đặt tên tập tin của DOS: *Tên của tập tin gồm 2 phần: Phần tên và phần mở rộng.*

- *Phần tên của tập tin phải bắt đầu là 1 ký tự từ a..z (không phân biệt hoa thường), theo sau có thể là các ký tự từ a..z, các ký số từ 0..9 hay dấu gạch dưới (_), phần này dài tối đa là 8 ký tự.*
- *Phần mở rộng: phần này dài tối đa 3 ký tự.*

Ví dụ: Ghi chương trình vừa soạn thảo trên lên đĩa với tên là CHAO.C

1.2.2 Mở.

a) soạn thảo một chương trình mới

Muốn soạn thảo một chương trình mới ta chọn mục New trong menu File (File ->New)

Trên màn hình sẽ xuất hiện một vùng trống để cho ta soạn thảo nội dung của chương trình. Trong quá trình soạn thảo chương trình ta có thể sử dụng các phím sau:

Các phím xem thông tin trợ giúp:

- F1: Xem toàn bộ thông tin trong phần trợ giúp.
- Ctrl-F1: Trợ giúp theo ngữ cảnh (tức là khi con trỏ đang ở trong một từ nào đo, chẳng hạn int mà bạn gõ phím Ctrl-F1 thì bạn sẽ có được các thông tin về kiểu dữ liệu int)

Các phím di chuyển con trỏ trong vùng soạn thảo chương trình:

Phím	Ý nghĩa	Phím tắt (tổ hợp
Enter	Đưa con trỏ xuống dòng	
Mũi tên đi lên	Đưa con trỏ lên hàng trước	Ctrl-E
Mũi tên đi	Đưa con trỏ xuống hàng sau	Ctrl-X
Mũi tên sang	Đưa con trỏ sang trái một ký	Ctrl-S
Mũi tên sang	Đưa con trỏ sang phải một ký	Ctrl-D
End	Đưa con trỏ đến cuối dòng	
Home	Đưa con trỏ đến đầu dòng	
PgUp	Đưa con trỏ lên trang trước	Ctrl-R
PgDn	Đưa con trỏ xuống trang sau	Ctrl-C
	Đưa con trỏ sang từ bên trái	Ctrl-A
	Đưa con trỏ sang từ bên phải	Ctrl-F

Các phím xoá ký tự/ dòng:

Phím	Ý nghĩa	Phím
Delet	Xoá ký tự tại ví trí con trỏ	Ctrl-
Back	Di chuyển sang trái đồng thời xoá ký tự đứng trước con	Ctrl-
	Xoá một dòng chứa con trỏ	Ctrl-
	Xóa từ vị trí con trỏ đến cuối dòng	Ctrl-
	Xóa ký tự bên phải con trỏ	Ctrl-

Các phím chèn ký tự/ dòng:

Insert	Thay đổi viết xen hay viết chồng
Ctrl-N	Xen một dòng trống vào trước vị trí con

Sử dụng khối :

Khối là một đoạn văn bản chương trình hình chữ nhật được xác định bởi đầu khối là góc trên bên trái và cuối khối là góc dưới bên phải của hình chữ nhật. Khi một khối đã được xác định (trên màn hình khối có màu sắc khác chỗ bình thường) thì ta có
thể chép khối, di chuyển khối, xoá khối... Sử dụng khối cho phép chúng ta soạn thảo chương trình một cách nhanh chóng. sau đây là các thao tác trên khối:

Phím	***Ý nghĩa***
Ctrl-	Đánh dấu đầu khối
Ctrl-	Đánh dấu cuối khối
Ctrl-	Chép khối vào sau vị trí con trỏ
Ctrl-	Chuyển khối tới sau vị trí con trỏ
Ctrl-	Xoá khối
Ctrl-	Ghi khối vào đĩa như một tập tin
Ctrl-	Đọc khối (tập tin) từ đĩa vào sau vị trí con
Ctrl-	Tắt/mở khối
Ctrl-	Đánh dấu từ chứa chon trỏ
Ctrl-	In một khối

Các phím, phím tắt thực hiện các thao tác khác:

Phím	**Ý nghĩa**	**Phím**
F10	Kích hoạt menu chính	Ctrl-K-D,
F2	Lưu chương trình đang soạn vào đĩa	Ctrl-
F3	Tạo tập tin mới	
Tab	Di chuyển con trỏ một khoảng đồng thời đẩy dòng văn	Ctrl-I
ESC	Hủy bỏ thao tác lệnh	Ctrl-
	Đóng tập tin hiện tại	Alt-
	Hiện hộp thoại tìm kiếm	Ctrl-
	Hiện hộp thoại tìm kiếm và thay thế	Ctrl-
	Tìm kiếm tiếp tục	Ctrl-

Ví dụ: Bạn hãy gõ đoạn chương trình sau:

```
#include <stdio.h>
#include<conio.h>
int    main ()
{
char ten[50];
printf("Xin cho biet ten cua ban !");
scanf("%s",ten);
```

```
printf("Xin chao ban %s",ten);
getch();
return 0;
       }
```

b) Mở một chương trình đã có sẵn

Với một chương trình đã có trên đĩa, ta có thể mở nó ra để thực hiện hoặc sửa chữa bổ sung. Để mở một chương trình ta dùng File/Open hoặc gõ phím F3. Sau đó gõ tên tập tin vào hộp File Name hoặc lựa chọn tập tin trong danh sách các tập tin rồi gõ Enter.

Ví dụ: Mở tập tin CHAO.C sau đó bổ sung để có chương trình mới như sau:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
char ten[50];
printf("Xin cho biet ten cua ban !");
scanf("%s",ten);
printf("Xin chao ban %s\n ",ten);
printf("Chao mung ban den voi Ngon ngu lap trinh C");
getch();
return 0;
}
```

Ghi lại chương trình này (F2) và cho thực hiện (Ctrl-F9). Hãy so sánh xem có gì khác trước?

1.3 Dịch , chạy chương trình

Để thực hiện chương trình hãy dùng Ctrl-F9 (giữ phím Ctrl và gõ phím F9).
Ví dụ: Thực hiện chương trình vừa soạn thảo xong và quan sát trên màn hình để thấy kết quả của việc thực thi chương trình sau đó gõ phím bất kỳ để trở lại với Turbo.

1.4 Sử dụng menu Help.

Trên menu Help bao gồm các lệnh gọi trợ giúp khi người lập trình cần giúp đỡ một số vấn đề nào đó như: Cú pháp câu lệnh, cách sử dụng các hàm có sẵn…

- Lệnh Contents: Hiện thị toàn bộ nội dung của phần help.
- Lệnh Index : Hiển thị bảng tìm kiếm theo chỉ mục.
- Các lệnh còn lại, bạn sẽ tìm hiểu khi thực hành trên máy

2. Cấu trúc của một chương trình C.

2.1. Tiền xử lý và biên dịch

Trong C, việc dịch (translation) một tập tin nguồn được tiến hành trên hai bước hoàn toàn độc lập với nhau:

- Tiền xử lý.
- Biên dịch.

Hai bước này trong phần lớn thời gian được nối tiếp với nhau một cách tự động theo cách thức mà ta có ấn tượng rằng nó đã được thực hiện như là một xử lý duy nhất. Nói chung, ta thường nói đến việc tồn tại của một bộ tiền xử lý (preprocessor?) nhằm chỉ rõ chương trình thực hiện việc xử lý trước. Ngược lại, các thuật ngữ trình biên dịch hay sự biên dịch vẫn còn nhập nhằng bởi vì nó chỉ ra khi thì toàn bộ hai giai đoạn, khi thì lại là giai đoạn thứ hai.

Bước tiền xử lý tương ứng với việc cập nhật trong văn bản của chương trình nguồn, chủ yếu dựa trên việc diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor); các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu (symbol) #.

Hai chỉ thị quan trọng nhất là:

- Chỉ thị sự gộp vào của các tập tin nguồn khác: ***#include***
- Chỉ thị việc định nghĩa các macros hoặc ký hiệu: ***#define***

Chỉ thị đầu tiên được sử dụng trước hết là nhằm gộp vào nội dung của các tập tin cần có (header file), không thể thiếu trong việc sử dụng một cách tốt nhất các hàm của thư viện chuẩn, phổ biến nhất là:

#include <stdio.h>

Chỉ thị thứ hai rất hay được sử dụng trong các tập tin thư viện (header file) đã được định nghĩa trước đó và thường được khai thác bởi các lập trình viên trong việc định nghĩa các ký hiệu như là:

#define NB_COUPS_MAX 100

#define SIZE 25

2.2 Cấu trúc một chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, khai báo biến ngoài, các hàm tự tạo, chương trình chính (hàm main).

Cấu trúc có thể như sau:

Các chỉ thị tiền xử lý (Preprocessor directives)
#include <Tên tập tin thư viện>
#define ….

Định nghĩa kiểu dữ liệu (phần này không bắt buộc): dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gợi nhớ hay đặt 1 kiểu dữ liệu cho riêng mình dựa trên các kiểu dữ liệu đã có.
Cú pháp: **typedef** <Tên kiểu cũ> <Tên kiểu mới>
Ví dụ: typedef int SoNguyen; // Kiểu SoNguyen là kiểu int

Khai báo các prototype (tên hàm, các tham số, kiểu kết quả trả về,… của các hàm *sẽ cài đặt trong phần sau, phần này không bắt buộc*): phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm.

Khai báo các biến ngoài (các biến toàn cục) *phần này không bắt buộc*: phần này khai báo các biến toàn cục được sử dụng trong cả chương trình.

Chương trình chính phần này bắt buộc phải có
<Kiểu dữ liệu trả về> main()
{
Các khai báo cục bộ trong hàm main: Các khai báo này chỉ tồn tại trong hàm mà thôi, có thể là khai báo biến hay khai báo kiểu.
Các câu lệnh dùng để định nghĩa hàm main
return <kết quả trả về>; // Hàm phải trả về kết quả
}

Cài đặt các hàm
<Kiểu dữ liệu trả về> function1(các tham số)
{
Các khai báo cục bộ trong hàm.
Các câu lệnh dùng để định nghĩa hàm return
<kết quả trả về>;
}
…

Một chương trình C bắt đầu thực thi từ hàm main (thông thường là từ câu lệnh đầu tiên đến câu lệnh cuối cùng).

2.3 Các tập tin thư viện thông dụng

Đây là các tập tin chứa các hàm thông dụng khi lập trinh C, muốn sử dụng các hàm trong các tập tin header này thì phải khai báo #include <Tên tập tin> ở phần đầu của chương trình

1) stdio.h: Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output). Gồm các hàm in dữ liệu (printf()), nhập giá trị cho biến (scanf()), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhận một dãy ký tự từ bàm phím (gets()),
in chuỗi ký tự ra màn hình (puts()), xóa vùng đệm bàn phím (fflush()), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…

2) conio.h : Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS

console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…

3) math.h: Tập tin định nghĩa các hàm tính toán gồm các hàm abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…

4) alloc.h: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

5) io.h: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…

6) graphics.h: Tập tin định nghĩacác hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …

Còn nhiều tập tin khác nữa.

2.4 Cú pháp khai báo các phần bên trong một chương trình C

2.4.1. Chỉ thị #include để sử dụng tập tin thư viện

Cú pháp:

#include <Tên tập tin> // Tên tập tin được đạt trong dấu <> hay **#include "Tên đường dẫn"**

Menu Option của Turbo C có mục INCLUDE DIRECTORIES, mục này dùng để chỉ định các tập tin thư viện được lưu trữ trong thư mục nào.

Nếu ta dùng #include<Tên tập tin> thì Turbo C sẽ tìm tập tin thư viện trong thư mục đã được xác định trong INCLUDE DIRECTORIES.

Ví dụ: include <stdio.h>

Nếu ta dùng #include "Tên đường dẫn" thì ta phải chỉ rõ tên ở đâu, tên thư mục và tập tin thư viện.

Ví dụ: #include"C:\\TC\\math.h"

Trong trường hợp tập tin thư viện nằm trong thư mục hiện hành thì ta chỉ cần đưa tên tập tin thư viện. Ví dụ: #include"math.h".

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
#include "math.h"
```

2.4.2. Chỉ thị #define để định nghĩa hằng số

Cú pháp:

#define <Tên hằng> <Giá trị>

Ví dụ:

```
#define MAXINT 32767
```

2.4.3. Khai báo các prototype của hàm

Cú pháp:

<Kiểu kết quả trả về> Tên hàm (danh sách đối số)

Ví dụ:

```
long giaithua( int n); //Hàm tính giai thừa của số nguyên n
```

```
double       x_mu_y(floatx, float y);         /*Hàm tính x mũ y*/
```

2.4.4. Cấu trúc của hàm "bình thường"

Cú pháp:

<Kiểu kết quả trả về> Tên hàm (các đối số)
{
Các khai báo và các câu lệnh định nghĩa hàm
return kết quả;
}

Ví dụ:

```
int       tong(int x, int y)       /*Hàm tính tổng 2 số nguyên*/
{
return (x+y);
}
float       tong(float x, float y)   /*Hàm tính tổng 2 số thực*/
{
return (x+y);
}
```

2.4.5. Cấu trúc của hàm main

Hàm main chính là chương trình chính, gồm các lệnh xử lý, các lời gọi các hàm khác.

Cú pháp:

<Kết quả trả về> main(đối số)
{
Các khai báo và các câu lệnh định nghĩa hàm return <kết quả>;

}

***Ví dụ 1*:**

```
int       main()
{
printf("Day la chuong trinh chinh");
getch();
return 0;
```

***Ví dụ 2*:**

```
int       main()
{
int a=5, b=6,c;
float          x=3.5, y=4.5,z;
printf("Day la chuong trinh chinh");
c=tong(a,b);
printf("\n Tong cua %d va %d la %d",a,b,c);
z=tong(x,y);
printf("\n Tong cua %f và %f là %f", x,y,z);
getch();
   return 0;
}
```

3.Câu lệnh nhập và xuất dữ liệu

3.1 Xuất dữ liệu lên màn hình

Hàm printf (nằm trong thư viện **stdio.h**) dùng để xuất giá trị của các biểu

thức lên màn hình.

Cú pháp:

printf(“Chuỗi định dạng ”, Các biểu thức);

Giải thích:

- *Chuỗi định dạng*: dùng để qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Một số định dạng khi đối với số nguyên, số thực, ký tự.

Định dạng	Ý nghĩa
%d	Xuất số nguyên
%[.số chữ số thập phân] f	Xuất số thực có <số chữ số thập phân> theo qu tắc
%o	Xuất số nguyên hệ bát phân
%x	Xuất số nguyên hệ thập lục phân
%c	Xuất một ký tự
%s	Xuất chuỗi ký tự
%e hoặc %E hoặc %g	Xuất số nguyên dạng khoa học (nhân 10 mũ x)
Ví dụ	
%d	In ra số nguyên
%4	In số nguyên tối đa 4 ký số, nếu số cần in nhiều hơn 4 ký số thì in hết
%f	In số thực
%6 f	In số thực tối đa 6 ký số (tính luôn dấu chấm), *nếu số cần in nhiều hơn ký*
%.3	In số thực có 3 số lẻ, nếu số cần in có nhiều hơn 3 số lẻ thì làm tròn.

- *Các biểu thức*: là các biểu thức mà chúng ta cần xuất giá trị của nó lên màn hình, mỗi biểu thức phân cách nhau bởi dấu phẩy (**,**).

Ví dụ:

```
include<stdio.h>
int main(){
int         bien_nguyen=1234, i=65;
float         bien_thuc=123.456703;
printf(“Gia   tri   nguyen   cua   bien   nguyen   =%d\n”,bien_nguyen);
printf(“Gia tri thuc cua bien thuc =%f\n”,bien_thuc); printf(“Truoc
khi lam tron=%f \n
Sau khi lam tron=%.2f”,bien_thuc, bien_thuc);
return 0;
}
```

Kết quả in ra màn hình như sau:

```
Gia tri nguyen cua bien nguyen =1234
Gia tri thuc cua bien thuc =123.456703
Truoc khi lam tron=123.456703
 Sau khi lam tron=123.46
```

Nếu ta thêm vào dòng sau trong chương trình:

```
printf(“\n       Ky tu co ma ASCII %d la %c”,i,i);
```

Kết quả ta nhận được thêm:

```
Ky tu co ma ASCII 65 la A_
```

```
printf(“ So nguyen la %d \n So thuc la %f”,i, (float)i );
```

```
So nguyen la 65
So thuc la 65.000000
```

```
printf(“\n  So  thuc  la  %f  \n  So  nguyen  la  %d”,bien_thuc,
```



```
So thuc la 123.456703
So nguyen la 123_
```

```
(int)bien_thuc);


printf("\n Viet binh thuong =%f \n Viet kieu khoa
hoc=%e",bien_thuc, bien_thuc);
```

Kết quả in ra màn hình:

```
Viet binh thuong=123.456703
Viet kieu khoa hoc=1.234567e+02
```

Lưu ý: Đối với các ký tự điều khiển, ta không thể sử dụng cách viết thông thường để hiển thị chúng.

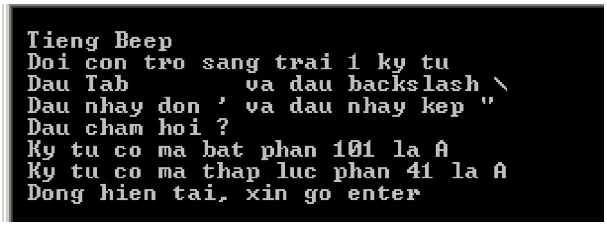
Ký tự điều khiển là các ký tự dùng để điều khiển các thao tác xuất, nhập dữ liệu. Một số ký tự điều khiển được mô tả trong bảng:

Ký tự điều khiển	**Ký tự được hiển thị**	**Ý nghĩa**
\a	BEL	Phát ra tiếng chuông
\b	BS	Di chuyển con trỏ sang trái 1 ký tự và xóa ký tự bên trái (backspace)
\f	FF	Sang trang
\n	LF	Xuống dòng
\r	CR	Trở về đầu dòng
\t	HT	Tab theo cột (giống gõ phím Tab)
\\	\	Dấu \
\’	‘	Dấu nháy đơn (‘)
\”	“	Dấu nháy kép (“)
\?	?	Đấu chấm hỏi (?)

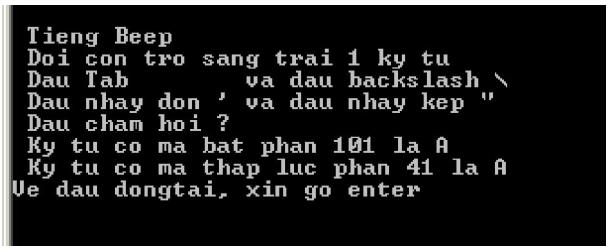
Ví dụ:

```
#include <stdio.h>
#include<conio.h>
int main ()
{       clrscr();
printf("\n Tieng Beep \a");
printf("\n Doi con tro sang trai 1 ky tu\b"); printf("\n Dau
Tab \tva dau backslash \\"); printf("\n Dau nhay don \' va
dau nhay kep \""); printf("\n Dau cham hoi \?");
    printf("\n Ky             tu co ma bat phan 101 la \101");
    printf("\n Ky tu co ma thap luc phan 41 la \x041");
    printf("\n             D    h    tai, xin go enter");
    getch();                o    i
                            n    e
                            g    n
    printf("\rVe dau dong");
    getch();
    return 0;
    }
```

Kết quả trước khi gõ phím Enter:



Kết quả sau khi gõ phím Enter:



3.2 Đưa dữ liệu vào từ bàn phím

Là hàm cho phép đọc dữ liệu từ bàn phím và gán cho các biến trong chương trình khi chương trình thực thi. Trong ngôn ngữ C, đó là hàm scanf nằm trong thư viện stdio.h.

Cú pháp:

scanf("Chuỗi định dạng", địa chỉ của các biến);

Giải thích:

- *Chuỗi định dạng*: dùng để qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Một số định dạng khi nhập kiểu số nguyên, số thực, ký tự.

Định dạng	**Ý nghĩa**
%[số ký số]d	Nhập số nguyên có tối đa <số ký số>
%[số ký số] f	Nhập số thực có tối đa <số ký số> tính cả dấu chấm
%c	Nhập một ký tự
Ví dụ:	
%d	Nhập số nguyên
%4d	Nhập số nguyên tối đa 4 ký số, nếu nhập nhiều hơn 4 ký số thì c nhận
%f	Nhập số thực
%6f	Nhập số thực tối đa 6 ký số (tính luôn dấu chấm), nếu nhập nhiều ho 6

- *Địa chỉ của các biến*: là *địa chỉ (&) của các biến* mà chúng ta cần nhập giá trị cho nó. Được viết như sau: **&<tên biến>**.

Ví dụ:

```
scanf("%d",&bien1);/*Doc gia tri cho bien1 co kieu nguyen*/
scanf("%f",&bien2); /*Doc gia tri cho bien2 co kieu thuc*/
scanf("%d%f",&bien1,&bien2);
/*Doc gia tri cho bien1 co kieu nguyen, bien2 co kieu thuc*/
scanf("%d%f%c",&bien1,&bien2,&bien3);
/*bien3 co kieu char*/
```

Lưu ý:

- Chuỗi định dạng phải đặt trong cặp dấu nháy kép (“”).
- Các biến (địa chỉ biến) phải cách nhau bởi dấu phẩy (,).
- Có bao nhiêu biến thì phải có bấy nhiêu định dạng.
- Thứ tự của các định dạng phải phù hợp với thứ tự của các biến.
- Để nhập giá trị kiểu char được chính xác, nên dùng hàm ***fflush(stdin)*** để loại bỏ các ký tự còn nằm trong vùng đệm bàn phím trước hàm scanf().
- Để nhập vào một chuỗi ký tự (không chứa khoảng trắng hay *kết thúc bằng khoảng trắng*), chúng ta phải khai báo kiểu *mảng ký tự* hay *con trỏ* ký tự, sử dụng định dạng %s và *tên biến thay cho địa chỉ biến*.
- Để đọc vào một chuỗi ký tự có chứa khoảng trắng (*kết thúc bằng phím Enter)* thì *phải dùng hàm gets*().

Ví dụ:

```
int         biennguyen;
float        bienthuc;
char         bienchar;
char        chuoi1[20], *chuoi2;
```

Nhập giá trị cho các biến:

```
scanf("%3d",&biennguyen);
```

- Nếu ta nhập 1234455 thì giá trị của biennguyen là 3 ký số đầu tiên (123). Các ký số còn lại sẽ còn nằm lại trong vùng đệm.

```
scanf("%5f",&bienthuc);
```

- Nếu ta nhập 123.446 thì giá trị của bienthuc là 123.4, các ký số còn lại sẽ còn nằm trong vùng đệm.

```
scanf("%2d%5f",&biennguyen, &bienthuc);
```

- Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 1223 3.142325

  - 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
  - 2 ký số tiếp theo trước khoảng trắng (23) sẽ được đọc vào cho bienthuc.

```
scanf("%2d%5f%c",&biennguyen, &bienthuc,&bienchar)
```

- Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 12345 3.142325:

  - 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
  - 3 ký số tiếp theo trước khoảng trắng (345) sẽ được đọc vào cho bienthuc.
  - Khoảng trắng sẽ được đọc cho bienchar.

- Nếu ta chỉ nhập 1 số gồm nhiều ký số như sau: 123456789:

  - 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
  - 5 ký số tiếp theo (34567) sẽ được đọc vào cho bienthuc.
  - bienchar sẽ có giá trị là ký số tiếp theo ‘8’.

```
scanf("%s",chuoi1);        hoặc scanf("%s",chuoi2)
```

- Nếu ta nhập chuỗi như sau: *Nguyen Van A* ↵ thì giá trị của biến chuoi1 hay chuoi2 chỉ là *Nguyen* .

```
scanf("%s%s",chuoi1, chuoi2);
```

Nếu ta nhập chuỗi như sau: *Duong Van H* ↵ thì giá trị của biến chuoi1 là *Duong* và giá trị của biến chuoi2 là *Van.*

Vì sao như vậy? C sẽ đọc từ đầu đến khi gặp khoảng trắng và gán giá trị cho biến đầu tiên, phần còn lại sau khoảng trắng là giá trị của các biến tiếp

theo.

```
gets(chuoi1);
```

Nếu nhập chuỗi : *Nguyen Van A* ↵ thì giá trị của biến chuoi1 là *Nguyen Van A*

4. Một vài chương trình đơn giản.

Ví dụ 1

Dòng	File Edit Search Run Compile Debug Project Option Window Help
1	/* Chuong trinh in ra cau bai hoc C dau tien */
2	#include <stdio.h>
3	
4	void main(void)
5	{
6	printf("Bai hoc C dau tien.");
7	}

- Dòng thứ 1: bắt đầu bằng /* và kết thúc bằng */ cho biết hàng này là hàng diễn giải (chú thích). Khi dịch và chạy chương trình, dòng này không được dịch và cũng không thi hành lệnh gì cả. Mục đích của việc ghi chú này giúp chương trình rõ ràng hơn. Sau này bạn đọc lại chương trình biết chương trình làm gì.
- Dòng thứ 2: chứa phát biểu tiền xử lý ***#include <stdio.h>***. Vì trong chương trình này ta sử dụng hàm thư viện của C là ***printf***, do đó bạn cần phải có khai báo của hàm thư viện này để báo cho trình biên dịch C biết. ***Nếu không khai báo chương trình sẽ báo lỗi***.
- Dòng thứ 3: hàng trắng viết ra với ý đồ làm cho bảng chương trình thoáng, dễ đọc.
- Dòng thứ 4: ***void main(void)*** là thành phần chính của mọi chương trình C (bạn có thể viết main() hoặc void main() hoặc main(void)). Tuy nhiên, bạn nên viết theo dạng void main(void) để chương trình rõ ràng hơn. Mọi chương trình C đều bắt đầu thi hành từ hàm main. Cặp dấu ngoặc () cho biết đây là khối hàm (function). Hàm void main(void) có từ khóa void đầu tiên cho biết hàm này không trả về giá trị, từ khóa void trong ngoặc đơn cho biết hàm này không nhận vào đối số.
- Dòng thứ 5 và 7: cặp dấu ngoặc móc {} giới hạn thân của hàm. Thân hàm bắt đầu bằng dấu { và kết thúc bằng dấu }.
- Dòng thứ 6: ***printf("Bai hoc C dau tien.");***, chỉ thị cho máy in ra chuỗi ký tự nằm trong nháy kép (""). Hàng này được gọi là một câu lệnh, kết thúc một câu lệnh trong C phải là dấu chấm phẩy(;).

Chú ý:

- Các từ include, stdio.h, void, main, printf phải viết bằng chữ thường.
- Chuỗi trong nháy kép cần in ra "Bạn có thể viết chữ HOA, thường tùy, ý".
- Kết thúc câu lệnh phải có dấu chấm phẩy.
- Kết thúc tên hàm không có dấu chấm phẩy hoặc bất cứ dấu gì.
- Ghi chú phải đặt trong cặp /* …. */.
- Thân hàm phải được bao bởi cặp { }.
- Các câu lệnh trong thân hàm phải viết thụt vào.

Ví dụ 2:

```
1   /* Chuong trinh nhap va in ra man hinh gia tri bien*/
2   #include <stdio.h>
3   #include <conio.h>
4
5   void main(void)
6   {
7   int i;
8   printf("Nhap vao mot so: ");
9   scanf("%d", &i);
10  printf("So ban vua nhap la: %d.\n", i);
    getch();
    }
```

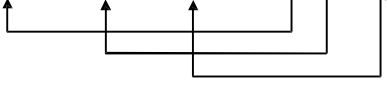
- Dòng thứ 7: ***int i;*** là lệnh khai báo, mẫu tự i gọi là tên biến. Biến là một vị trí trong bộ nhớ dùng lưu trữ giá trị nào đó mà chương trình sẽ lấy để sử dụng. Mỗi biến phải thuộc một kiểu dữ liệu. Trong trường hợp này ta sử dụng biến i kiểu số nguyên (integer) viết tắt là int.
- Dòng thứ 9: ***scanf("%d", &i)***. Sử dụng hàm scanf để nhận từ người sử dụng một trị nào đó. Hàm scanf trên có 2 đối mục. Đối mục ***"%d"*** được gọi là chuỗi định dạng, cho biết loại dữ kiện mà người sử dụng sẽ nhập vào. Chẳng hạn, ở đây phải nhập vào là số nguyên. Đối mục thứ 2 ***&i*** có dấu & đi đầu gọi là address operator, dấu & phối hợp với tên biến cho hàm scanf biến đem trị gõ từ bàn phím lưu vào biến i.
- Dòng thứ 10: ***printf("So ban vua nhap la: %d.\n", i);***. Hàm này có 2 đối mục. Đối mục thứ nhất là một chuỗi định dạng có chứa chuỗi văn bản ***So ban vua nhap la:*** và ***%d*** (ký hiệu khaibáo chuyển đổi dạng thức) cho biết số nguyên sẽ được in ra. Đối mục thứ 2 là i cho biết giá trị lấy từ biến i để in ra màn hình.

Ví dụ 3:

Dòn **F**ile **E**dit **S**earch **R**un **C**ompile **D**ebug **P**roject **O**ption **W**indow **H**elp

```
1   /* Chuong trinh nhap vao 2 so a, b in ra tong*/
2   #include <stdio.h>
3   #include <conio.h>
4
5   void main(void)
6   {
7   int a, b;
8   printf("Nhap vao so a: "); scanf("%d", &a);
9      printf("Nhap vao so b: "); scanf("%d", &b);
10  printf("Tong cua 2 so %d va %d la %d.\n", a, b, a+b);
11  getch();
12  }
```

Dòng thứ 10: ***printf("Tong cua 2 so %d va %d la %d.\n", a, b, a+b);***



5. Thực hành

5.1 Mục đích, yêu cầu

Học viên có khả năng:

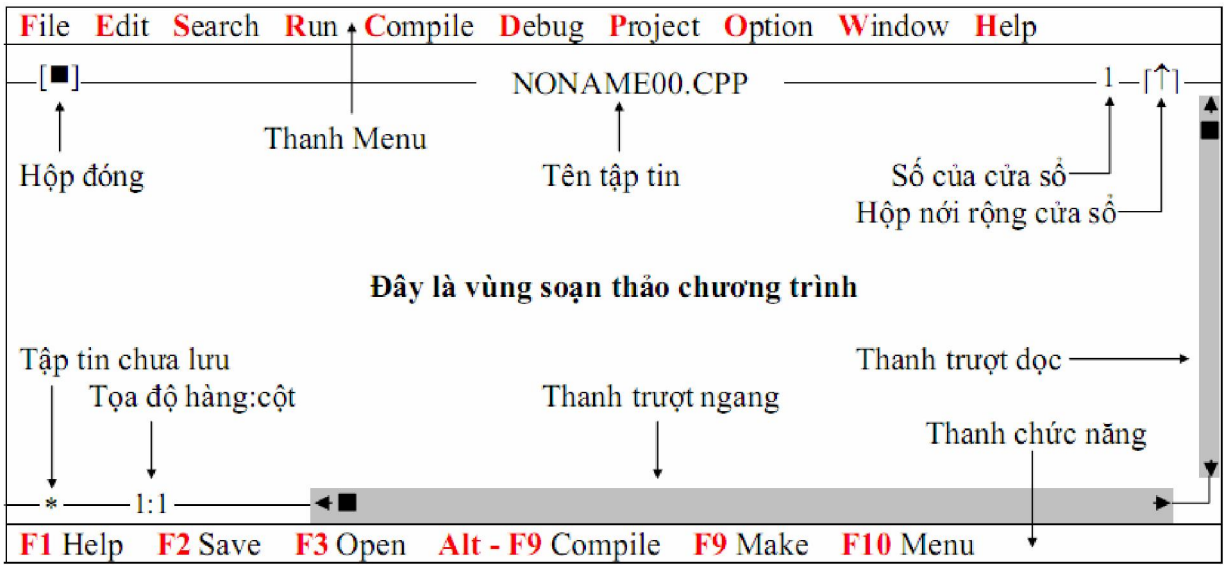
- Thực hiện được các thao tác cơ bản như Lưu, mở,… trong Borland C.
- Thực hiện được các quá trình nhập xuất trong một số chương trình cơ bản.

5.2 Nội dung thực hành

BÀI THỰC HÀNH 1: Làm quen với ngôn ngữ Lập trình C.

a) Khởi động C trên Window.

b) Tìm hiểu cửa sổ làm việc trên C



c) Soạn thảo chương trình xuất ra màn hình với nội dung như sau:

```
MOC GIUA DONG SONG XANH
 MOT BONG HOA TIM BIEC
OI CON CHIM CHIEN CHIEN
  HOT CHI MA VANG TROI
TUNG GIOT LONG LANH ROI
  TOI DUA TAY TOI HUNG
.....
                (Mua xuan nho nho)
```

d) Sau đó lưu lại với tên: ***baitho.cpp***

e) Sau đó thoát khỏi chương trình.

f) Gọi Turbo C và mở tập tin ***baitho.cpp***

BÀI THỰC HÀNH 2: Soạn thảo chương trình sau.

Viết chương trình nhập tên bạn và in ra lời chào.
Hướng dẫn: viết chương trình thêm vào những dòng có dấu chấm:

```
#include <stdio.h>
#include <conio.h>
```

```
void main(void)
{
char name[50], ch;
……
……
printf("\nchao %s!\n",name);
ch=getch();
}
```

BÀI THỰC HÀNH 3:

Viết chương trình nhập vào bán kính hình tròn và tính diện tích hình tròn.

Hướng dẫn

```
#include <stdio.h>
#include <conio.h>
#define PI3.14
void main(void)
{
float R;
…….("Nhap vao ban kinh hinh tron: ");
…….("%..", …….);
…….("Dien tich hinh tron %...\n"……);
getch();
}
```

BÀI 2: HẰNG,BIẾN VÀ MẢNG

Mục tiêu:

- Trình bày được các kiểu dữ liệu và so sánh được phạm vi biểu diễn của các kiểu dữ liệu;
- Vận dụng được các loại biến, hằng biểu thức cho từng chương trình cụ thể;
- Biết, hiểu và so sánh được các lệnh, khối lệnh;
- Viết được các chương trình đơn giản với các hằng, biến và mảng;

1. Kiểu dữ liệu

1.1 Kiểu số nguyên

Đây là các kiểu dữ liệu mà giá trị của nó là số nguyên. Dữ liệu kiểu số nguyên lại chia ra thành hai loại như sau:

 Các số nguyên có dấu (signed) để chứa các số nguyên âm hoặc dương.

Kiểu (Type)	*Độ lớn (Byte)*	*Miền giá trị (Range)*
char	1	–128 ... 127
int	2	–32.768 ... 32.768
short	2	–32.768 ... 32.768
long	4	–2.147.483.648 ... +2.147.483.647

- Các số nguyên không dấu (unsigned) để chứa các số nguyên dương (kể cả số 0)

Kiểu (Type)	*Độ lớn (Byte)*	*Miền giá trị (Range)*
unsigned char	1	0 ... 255
unsigned int	2	0 ... 65.535
unsigned short	2	0 ... 65.535
unsigned long	4	0 ... 4.294.967.295

1.2 Kiểu số thực

Đây là các kiểu dữ liệu mà giá trị của nó là số thực. Trong C định nghĩa các kiểu số thực chuẩn như sau:

	Độ lớn (Byte)	*Miền giá trị (Range)*
loat	4	$3.4*10^{-38}$ ... $3.4*10^{38}$
double	8	$1.7*10^{-308}$ ... $1.7*10^{308}$

Kiểu float là kiểu số thực có độ chính xác đơn (single-precision floating-point), kiểu double là kiểu số thực có độ chính xác kép (double-precision floating-point).

1.3 Kiểu luận lý

Trong C không hỗ trợ kiểu luận lý tường minh mà chỉ ngầm hiểu một cách không tường minh như sau:

- false (sai) là giá trị 0.
- true (đúng) là giá trị khác 0, thường là 1.

1.4 Kiểu ký tự

Đây chính là kiểu dữ liệu số nguyên **char** có độ lớn 1 byte và miền giá trị là 256 ký tự trong bảng mã ASCII.

2. Hằng.

Hằng là đại lượng không đổi trong suốt quá trình thực thi của chương trình.

Hằng có thể là một chuỗi ký tự, một ký tự, một con số xác định. Chúng có thể được biểu diễn hay định dạng với nhiều dạng thức khác nhau.

2.1 Hằng số thực

Số thực bao gồm các giá trị kiểu float, double, long double được thể hiện theo 2 cách sau:

- *Cách 1*: Sử dụng cách viết thông thường mà chúng ta đã sử dụng trong các môn Toán, Lý, …Điều cần lưu ý là sử dụng dấu thập phân là dấu chấm (.);

Ví dụ: 123.34 -223.333 3.00 -56.0

- *Cách 2*: Sử dụng cách viết theo số mũ hay số khoa học. Một số thực được tách làm 2 phần, cách nhau bằng ký tự e hay E

Phần giá trị: là một số nguyên hay số thực được viết theo cách 1.
Phần mũ: là một số nguyên

Giá trị của số thực là: *Phần giá trị nhân với 10 mũ phần mũ*.

Ví dụ: 1234.56e-3 = 1.23456 (là số $1234.56 * 10^{-3}$)
-123.45E4 = -1234500 (là $-123.45 * 10^{4}$)

2.2 Hằng số nguyên.

Số nguyên gồm các kiểu int (2 bytes) , long (4 bytes) được thể hiện theo những cách sau.

- *Hằng số nguyên 2 bytes (int) hệ thập phân*: Là kiểu số mà chúng ta sử dụng thông thường, hệ thập phân sử dụng các ký số từ 0 đến 9 để biểu diễn một giá trị nguyên.

Ví dụ: 123 (một trăm hai mươi ba), -242 (trừ hai trăm bốn mươi hai).

- *Hằng số nguyên 2 byte (int) hệ bát phân*: Là kiểu số nguyên sử dụng 8 ký số từ 0 đến 7 để biểu diễn một số nguyên.

Cách biểu diễn: 0<các ký số từ 0 đến 7> Ví dụ : 0345 (số 345 trong hệ bát phân)
-020 (số -20 trong hệ bát phân)

Cách tính giá trị thập phân của số bát phân như sau:

Số bát phân : $0d_nd_{n-1}d_{n-2}\ldots d_1d_0$ (d_i có giá trị từ 0 đến 7)

=> Giá trị thập phân= $\sum d_i * 8^i$

0345=229 , 020=16

- *Hằng số nguyên 2 byte (int) hệ thập lục phân*: Là kiểu số nguyên sử dụng 10 ký số từ 0 đến 9 và 6 ký tự A, B, C, D, E ,F để biểu diễn một số nguyên.

Ký tự	giá trị
A	10
B	11
C	12
D	13
E	14
F	15

Cách biểu diễn: 0x<các ký số từ 0 đến 9 và 6 ký tự từ A đến F>

Ví dụ:
0x345 (số 345 trong hệ 16)
0x20 (số 20 trong hệ 16)
0x2A9 (số 2A9 trong hệ 16)

Cách tính giá trị thập phân của số thập lục phân như sau:

Số thập lục phân : $0xd_nd_{n-1}d_{n-2}\ldots d_1d_0$ (d_i từ 0 đến 9 hoặc A đến F)
=> Giá trị thập phân= $\sum_{i=0}^{n} d_i * 16^i$

- *Hằng số nguyên 4 byte (long)*: Số long (số nguyên dài) được biểu diễn như số int trong hệ thập phân và kèm theo ký tự *l* hoặc L. Một số nguyên nằm ngoài miền giá trị của số int (2 bytes) là số long (4 bytes).

Ví dụ: 45345L hay 45345l hay 45345

- *Các hằng số còn lại*: Viết như cách viết thông thường (không có dấu phân cách giữa 3 số)

Ví dụ:
12 (mười hai)
12.45 (mười hai chấm 45)
1345.67 (một ba trăm bốn mươi lăm chấm sáu mươi bảy)

2.3 Hằng ký tự

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn (‘). Mỗi một ký tự tương ứng với một giá trị trong bảng mã ASCII. Hằng ký tự cũng được xem như trị số nguyên.

Ví dụ: ‘a’, ‘A’, ‘0’, ‘9’

Chúng ta có thể thực hiện các phép toán số học trên 2 ký tự (thực chất là thực hiện phép toán trên giá trị ASCII của chúng)

2.4 Hằng chuỗi ký tự

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép (“).

Ví dụ: “Ngon ngu lap trinh C”, “Khoa CNTT-TCDTNTAG”, “NVA-DVB”

Chú ý:
1. Một chuỗi không có nội dung “” được gọi là chuỗi rỗng.
2. Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL (‘\0’: mã Ascii là 0).
3. Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước.

Ví dụ: “I’m a student” phải viết “I\’m a student”
“Day la ky tu “dac biet”” phải viết “Day la ky tu \”dac biet\”“

3. Biến

Biến là một đại lượng được người lập trình định nghĩa và được đặt tên thông qua việc khai báo biến. Biến dùng để chứa dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này. Cách đặt tên biến giống như cách đặt tên đã nói trong phần trên.
Mỗi biến thuộc về một kiểu dữ liệu xác định và có giá trị thuộc kiểu đó.

3.1 Cú pháp khai báo biến:

<Kiểu dữ liệu> Danh sách các tên biến cách nhau bởi dấu phẩy;

Ví dụ:

Main()

```
{
int       a, b, c;              /*Ba biến a, b,c có kiểu int*/
long int       chu_vi;         /*Biến chu_vi có kiểu long*/
char beta, alfa;               /* khai báo biến kiểu char*/
float        nua_chu_vi;       /*Biến nua_chu_vi có kiểu float*/
double      dien_tich;         /*Biến dien_tich có kiểu double*/
`
.
.
}
```

Biến kiểu int chỉ nhận được các ía trị kiểu int. Các biến khác cùng có ý nghĩa tương tự, chẳng hạn biến kiểu chả chỉ chứa được một ký tự. Để lưu trữ một xâu kí tự cần sử dụng một mảng kiểu char.

3.2 Vị trí khai báo

Các khai báo cần đặt ngay sau dấu '{ đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác. Như vậy, sau một câu lệnh gán chẳng hạn thì không được khai báo nữa. Sau đây là một ví dụ sai về vị trí của khai báo.

```
main()
{
 int a, b,c;
 a=35;
 int d;         /*Vị trí khai báo sai*/
 .
 .
 }
```

3.3 Việc khởi tạo đầu cho các biến.

Nếu trong khia báo, ngay sau tên biến ta đặt dấu = và một giá trị nào đó thì đây chính là cách vừa khai báo vừa khởi đầu cho 1 biến. Ví dụ:

```
int a,b =20, c,d =40;
float e= -35.1, x =23.0, y,z t =36.1;
```

Tất nhiên điều này có thể đạt được nhờ toán tử gán và về thực tế hai cách này là tươn đơn g. vậy để đạt được ý định như ví dụ trên ta có thể dùng các câu lệnh:

```
int a, b,c,d;
float e,x,y,z,t;
b =20; d=40; e = -35.1; x= 23.0; t =36.1;
```

3.4 Lấy địa chỉ của biến.

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ biến dùng trong một số hàm như scanf. Để nhận địa chỉ biến ta dùng phép toán: **&tên_biến**

4. Khối lệnh

4.1 Định nghĩa

Khối lệnh là một dãy các câu lệnh được bao bởi các dấu {và }. Ví dụ;

```
{
A=2; b=3;
Printf("\n%6d%6d",a,b);
}
```

4.2 Một số chú ý quan trọng khi viết chương trình

Turbo C xem một khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác chỗ nào viết được một câu lệnh thì ở đó cũng có quyền đặt một khối lệnh.

4.3 Khai báo ở đầu khối lệnh.

Các khai báo biến, mảng chẳng những có thể đặt ở đầu của một hàm mà còn có thể viết ở đầu khối lệnh, Ví dụ:

```
{
int a,b,c[50];
float x,y,z,t[20][30];
    a=b=3;/*gán 3 cho a và b*/
    x=5.7; y=a*x;
    z=b*x;
    printf("\n y=%8.2f\nz=%8.2f",y,z);
    }
```

4.4 Sự lồng nhau của các khối lệnh.

Bên trong một khối lệnh lại có thể viết các khối lệnh khác. Sự lồng nhau theo cách như vậy là không hạn chế. Ta thấy một điều thú vị là: Thân hàm cũng là một khối lệnh. Đó là khối lệnh cực đại theo nghĩa nó chứa nhiều khối lệnh khác bên trong và không khối lệnh nào chứa nó.

Về các khối lệnh lồng nhau:

Giả sự cùng một tên biến hay tên mảng lại được khai báo ở cả hai khối lệnh lồng nhau theo cách

```
{
    int a;
    .
    .
    {
    int a;
    .
    .
    }
```

```
        .
        .
}
```

Khi đó làm thế nào để phân biệt được biến a trong và biến a ngoài. Về thực chất ở đây ta có hai đại lượng khác nhau cho hai biến này, phạm vi hoạt động và thời gian tồn tại của chúng cũng khác nhau. Biến a trong có phạm vi hoạt động tại các câu lệnh của khối lệnh trong và nó chỉ tồn tại trong thời gian máy làm việc với khối lệnh này. Còn phạm vi hoạt động của biến a ngoài bao gồm các câu lệnh bên trongkhoois lệnh ngoài nhưng không thuộc khối lệnh trong. Ta cũng lưu ý là: Sự thay đổi ía trị của biến a ngoài không có ảnh hưởng gì tới biến a trong và ngược lại. Ví dụ xét đoạn chương trình.

```
{
      int a=5, b=2;
      {
        int a=4;
        b=a+b;
      printf("\na=%3d b=%3d",a,b);
      }
      printf("\na=%3d b=%d",a,b);
}
```

Trong chương trình trên có sử dụng hai câu lệnh printf hoàn toàn giống nhau về mặt hình thức, nhưng thực chất chúng sẽ cho các kết quả khác nhau. Biến a ở câu lệnh printf thứ nhất có giá trị bằng 4 (biến a trong), còn biến a ở câu lệnh printf thứ 2 có giá trị là 5 (biến a ngoài). Biến b có cùng một ý nghĩa như nhau trong cả hai câu lệnh, nó có giá trị bằng 6. Về mặt hình thức thì chương trình trên có hai biến là a và b. Nhưng thực chất tồn tại ba biến khác nahu: a ngoài, a trong và b.

Như vậy đoạn chương trình trên sẽ có hiện trên màn hình hai dòng như sau:

a=4 b=6
a=5 b=6

5. Mảng

5.1 Khái niệm về mảng, cách khai báo.

Mảng có thể hiểu là một tập hợp nhiều phần tử có cùng một kiểu giá trị và có chung một tên. Mỗi phần tử mảng có vai trò như một biến và chứa được một giá trị. Có bao nhiêu kiểu biến thì cũng có bấy nhiêu kiểu mảng.

Mảng cần được khai báo để định rõ:

- Kiểu mảng (int, float, double,…)
- Tên mảng,
- Số chiều và kích thước mỗi chiều.

Khái niệm về kiểu mảng và tên mảng cũng giống như khai niệm kiểu biến và tên biến, điều này đã nói ở các mục trên. Ta sẽ giải thích khái niệm số chiều và kích thước mỗi chiều thông qua các ví dụ sau. Các khai báo:

int a[10],b[4][2];

float x[],y[3][3];

Sẽ xác định 4 mảng: a, b,x và y. Ý nghĩa của chúng như sau:

- Đối với mảng thứ nhất thì kiểu int, tên là a, số chiều là môt, kích thước bằng 10. Mảng có 10 phần tử được đánh số như sau: a[0],a[1], a[2],…,a[9]. Mỗi phần tử a[1] chứa được một giá trị kiểu int và mảng a có thể biểu diễn được một dãy 10 số nguyên.
- Đối với mảng thứ hai thì kiểu là int, tên là b số chiều là 2, kích thước của các chiều là 4 và 2. Mảng có 8 phần tử, chúng được đánh số và được sắp xếp như sau:

b[0][0] b[0][1]
b[1][0] b[1][1]
b[2][0] b[2][1]
b[3][0] b[3][1]

Mỗi phần tử b[i][j] chứa được một giá trị kiểu int, mảng b có thể biểu diễn được một bảng số nguyên 4 dòng, 2 cột

- Đối với mảng thứ ba thì kiểu là float, tên là x, số chiều là một, kích thước bằng 5. mảng có 5 phần tử được đánh số như sau:

x[], x[1],x[2],x[3],x[4]

Mỗi phần tử x[i] chứa được một giá trị kiểu float, và mảng x có thể biểu diễn được một dãy 5 số thực.

- Đối với mạng thứ 4 thì kiểu là float, tên y, số chiều là 2, kích thước của các chiều là 3. Mảng có 9 phần tử , chúng được đánh số và được sắp xếp như sau;

y[0][0]	y[0][1]	y[0][2]
y[1][0]	y[1][1]	y[1][2]
y[2][0]	y[2][1]	y[2][2]

Mỗi phần tử y[i][j] chứa được một giá trị kiểu float, và mảng y có thể biểu diễn được một bảng số thực 3 dòng, 3 cột.

Chú ý

- Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác, các phần tử mảng có địa chỉ liên tiếp nhau.
- Trong bộ nhớ, các phần tử của mảng hai chiều được sắp xếp theo hàng.

5.2 Chỉ số mảng

Một phần tử cụ thể của mảng được xác định nhờ các chỉ số của nó. Chỉ số của mảng phải có giá trị int không vượt quá kích thước của chiều tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử a,b,x,y đã được khai báo như trên và giả sử i, j là các biến nguyên trong đó i=2, j=1.

Khi đó:

a[j+i-1] là a[2]
b[j+i][2-1] là b[3][0]

x[j/i] là x[0]
y[i][j] là y[2][1]
Các cách viết sau là sai:
y[j] vì y là mảng hai chiều, cần hai chỉ số .
b[i][j][1] vì b là mảng hai chiều, chỉ cần hai chỉ số.
Chú ý về chỉ số:
Biểu thức dùng làm chỉ số có thể thực.
Khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng
Ví dụ:
a[2.4]=a[2]
a[1.9]=a[1]
Khi chỉ số vượt ra ngoài kích thước mảng, máy vẫn không báo lỗi, nhưng sẽ truy nhập đến một vùng nhớ bên ngoài mảng và có thể làm rối loạn chương trình.

5.3 Lấy địa chỉ phần tử mảng

Dưới đây khi nói mảng ta hiểu là mảng một chiều. Mảng có từ hai chiều trở lên ta nói mảng kèm số chiều. Có một vài hạn chế trên các mảng nhiều chiều. chẳng hạn, có thể lấy địa chỉ phần tử mảng một chiều, nhưng nói chung không cho phép lấy địa chỉ phần tử mảng nhiều chiều.

Như vậy máy sẽ chấp nhận phép tính: &a[i] nhưng không chấp nhận phép tính &y[i][j].

5.4 Địa chỉ đầu của mảng

Một chú ý quan trọng là: Tên mảng biểu thị địa chỉ đầu của mảng.
Như vậy ta có: a= &a[0].

6. Các loại biến và mảng

6.1 Biến , mảng tự động

6.1.1 Định nghĩa:

Các biến khai báo bên trong thân của một hàm (kể cả hàm main) gọi là biên (mảng) tự động hay cục bộ.

Các đối số của hàm cũng được xem là biến tự động.
Do thân hàm cũng là một khối lệnh, nên từ những qui định chung về khối lệnh đã nói trong mục trên có thể rút ra những điều sau:
+ Phạm vi hoạt động: Các biến (mảng) tự động chỉ có tác dụng bên trong thân của hàm mà tại đó chúng được khai báo.
+ Thời gian tồn tại: Các biến (mảng) tự động của một hàm sẽ tồn tại (được cấp phát bộ nhớ) trong khoảng thời gian từ khi máy bắt đầu làm việc với hàm đến khi máy ra khỏi hàm.
+ Nhận xét:
Do chương trình bắt đầu làm việc từ câu lệnh đầu tiên của hàm main ( ) và khi máy ra khỏi hàm main() thi chương tình kết thúc, nên các biến , mảng khai báo trong hàm main( ) sẽ tồn tại trong suốt thời gian làm việc của chương trình.

6.1.2 Khởi đầu

Chỉ có thể áp dụng cơ chế khởi đầu(khởi đầu trong khai báo) cho biến tự động .
Muốn khởi đầu cho một mảng tự động ta phải sử dụng toán tử gán.
Ví dụ:
Đoạn chương trình

```
main()
 {
 float a[4]={3.2,1.5,3.6,2.8};
 }
```

Sai ở chỗ đã sử dụng cơ chế khởi đầu cho mảng tự động a.

Biến, mảng tự động chưa khởi đầu thì giá trị của chúng là hoàn toàn là không xác định.

Là vô nghĩa và không thể làm việc được, vì nó thực hiện ý định đưa ra màn hình giá trị của biến a và phần tử mảng b[1] trong khi cả hai đều chưa được khởi đầu.

6.2 Biến, mảng ngoài

+ Định nghĩa: biến, mảng khai báo bên ngoài các hàm gọi là biến, mảng ngoài.

+ Thời gian tồn tại: BIến, mảng ngoài sẽ tồn tại (được cáp phát bộ nhớ) trong suốt thời gian làm việc của chương trình.

+ Phạm vi sử dụng: Phạm vi hoạt động của biến, mảng ngoài là từ vị trí khai báo của chúng cho đến cuối tệp chương trình. Như vậy, nếu một biến, mảng ngoài được khai báo ở đầu chương trình (đứng trước tất cả các hàm) thì nó có thể được sử dụng trong bất kỳ hàm nào miễn là hàm đó không có các biến tự động trùng tên với biến mảng ngoài này.

Chú ý: Nếu chương trình trình viết trên nhiều tập tin và các tập tin được dịch độc lập, thì phạm vi sử dụng của biến mảng ngoài có thể mở rộng từ tập tin này sang tập tin khác bằng từ khóa extern.

+ Các qui tắc khởi đầu

1/ Các biến, mảng ngoài có thể khởi đầu (1 lần) vào lúc dịch chương trình bằng cách sử dụng các biểu thức hằng. Nếu không được khởi đầu, máy sẽ gán cho chúng giá trị không.

Ví dụ:

```
char sao='*';
int a=6*365;
long b 34*3*2467;
float x=32.5;
float y[6]= {3.2,0,5.1,23,0,41};
int z[3][2]={
       {25,31},
       {46,54},
       {93,81}
       };
main()
{
.
.
}
```

2/ Khi khởi dầu mảng ngoài có thể không cần chỉ ra kích thước (số PT) của nó. Khi đó, máy sẽ dành cho mảng một khoảng nhớ đủ để thu nhận danh sách giá trị khởi đầu.

Ví dụ:

```
float a[]={2.6,3,15};
int t[][4]={
            {6,7,8,9},
            {3,12,4,14};

      };
```

3/ Khi chỉ ra kích thước của mảng, thì kích thước này cần không nhỏ hơn kích thước của bộ khởi đầu.
Ví dụ:

```
float h[6][3] = {

            {4, 3, 2}, {6, 1,9},

            {0.5.2}

            };
```

4/ Đối với mảng 2 chiều có thể khởi đầu theo các cách sau (Số giá trị khởi đầu trên mỗi hàng có thể khác nhau)

Ví dụ: float a[][3]= {

{0},

{2.5, 3.6,8},

{-6.3}

};

5/ Bộ khởi đầu của một mảng **char** có thể:
Hoặc là danh sách của hằng kí tự,

Hoặc là một hằng xâu kí tự.

Ví dụ:
char name[] = {‘H’, ‘a’, ‘n’, ‘g’, ‘\0’”};

char name[] = “Hang”;

6.4 Biến tĩnh, mảng tĩnh

Để khai báo một biến, mảng tĩnh ta viết thêm từ khóa **static** vào đằng trước.
Ví dụ:
static int a, b, c[10];
static float x,y[10][6];
Các khai báo dạng trên có thể đặt bên trong hoặc bên ngoài các hàm. Nếu đặt bên trong ta có các biến , mảng tĩnh trong, đặt bên ngoài ta có các biến, mảng tĩnh ngoài.
Các biến, mảng tĩnh (trong và ngoài) giống biến mảng ngoài ở chỗ:
+ Chúng được cấp phát bộ nhớ trong suốt thời gian hoạt động của chương trình, do đó, giá trị của chúng được lưu trữ từ đầu đến cuối chương trình.
+ Chúng có thể khởi đầu một lần khi dịch chương trình nhờ các biểu thức hằng. Các qui tắc khởi đầu đối với biến, mảng ngoài áp dụng được đối với biến, mảng tĩnh. Sự khác nhau giữa biến, mảng ngoài với biến mảng tĩnh chỉ phạm vi hoạt động sử dụng.

+ Các biến mảng tĩnh trong chỉ được sử dụng bên trong thân của hàm mà tại đó chúng được khai báo.

+ Phạm vi sử dụng của các biến mảng tĩnh ngoài được tính từ khi chúng khai báo đến cuối tệp gốc chứa chúng.

Trong trường hợp chương trình đặt trên một tệp, hoặc chương trình đặt trên nhiều tệp nhưng dùng toán tử #include để kết nối các tệp với nhau thì các biến, mảng tĩnh ngoài có cùng phạm vi sử dụng như các biến, mảng ngoài.

Ví dụ:

```
/* chuong trinh minh hoa cach khoi dau bien, mang tinh ngoai*/
#include<stdio.h>
static int a=41, t[][3]= {
                    {25, 30, 40},
                    {145,83,10}
                    };
static float y[8]= {-45.8,32.5};
static float x[10][2] ={
                    {-125.3,48.9},
                    {145.6,83.5}
                    };
static char n1[]={‘T’, ‘h’, ‘u’, ‘\0’};
static char n2= “Thu”;
static char n3[10] = ‘T’, ‘h’, ‘u’, ‘\0’};
static char n4[10]= “Thu”;
        main()
        {
        fprintf(stdprn “\na=%6d, t(1,2) =%6d, t(1,1)=%6d”,a, t[1][2],t[1][1]);
        fprintf(stdprn,”\n\nx(1,1) =%8.2f, x(2,0)=%8.2f”,x[1][1],x[2][0]);
        fprintf(stdprn, “\n\n%8s%8s%8s”,n1,n2,n3,n4);
        }
```

Kết quả thực hiện chương trình:

```
a=41, t(1,2)=10, t(1,1)=83

x(1,1)=83.50, x(2,0)=0.00
Thu     Thu     Thu     Thu
```

6.5 Bài tập áp dụng

Ví dụ:

```
Int i;              /*bien ben trong*/

Float  pi;/*bien ben ngoia*/

Int main()

{….
```

```
   }
```

Ví dụ 2:

```
 #include<stdio.h>
 #include<conio.h>
 Int main( )
 {int i, j;
 clrscr();
 i =4; j=5;
 printf ("Gia tri cua I la %d",i);
 printf("Gia tri cua j la %d",j);
ì f (j>i)
{
int hieu=j-i;                /*bien ben trong*/
printf("\n hieu so cua j tru i la %d", hieu);
}
else
{ int hieu =i-j              /*bien ben trong*/
 printf("\n Gia tri cua i tru j la %d", hieu);
}
getch( );
return 0;
}
```

Ví dụ 3:

Viết chương trình nhập vào n số nguyên. Tính và in ra trung bình cộng.

```
/* Tinh trung binh cong n so nguyen */

#include <stdio.h>
#include <conio.h>

void main(void)
{
int ia[50], i, in, isum = 0;
printf("Nhap vao gia tri n:
"); scanf("%d", &in);

//Nhap du lieu vao mang
for(i = 0; i < in; i++)
{
printf("Nhap vao phan tu thu %d: ", i + 1);
scanf("%d", &ia[i]);  //Nhap gia tri cho phan tu thu i
}

//Tinh tong gia tri cac phan
tu for(i = 0; i < in; i++)
isum += ia[i];        //cong don tung phan tu vao isum

printf("Trung binh cong: %.2f\n", (float) isum/in);
getch();
}
```

7. Thực hành

7.1 Mục đích yêu cầu:

- Làm quen và nắm vững các cách khai báo biến, các kiểu dữ liệu.
- Viết được các chương trình đơn giản với các hằng, biến và mảng.

7.2 Nội dung thực hành

Bài 1: Viết chương trình nhập vào bán kính của hình tròn. Tính chu vi à sdieenj tích của hình tròn theo công thức:

Chu vi CV= 2*pi*r

Diện tích S= pi*r*r
In kết quả ra màn hình.

Bài 2: Viết chương trình nhập vào các giá trị điện trở R1, R2, của một mạch điện: tính tổng trở theo công thức: $\frac{1}{R} = \frac{1}{R1} + \frac{1}{R2}$

Bài 3: Viết chương trình nhập vào điểm ba môn Toán, Lý, Hóa của một học sinh. In ra điểm trung bình của học sinh đó với hai số lẻ thập phân.
Bài 4: Viết chương trình nhập vào ngày, tháng, năm. In ra ngày tháng năm theo dạng dd/mm/yy. Ví dụ 20/11/99.
Bài 5: Viết chương trình nhập vào 1 dãy số dương rồi in tổng các số dương đó.

BÀI 3: BIỂU THỨC

1. Các phép toán

1.1 Phép toán số học

Trong ngôn ngữ C, các toán tử +, -, *, / làm việc tương tự như khi chúng làm việc trong các ngôn ngữ khác. Ta có thể áp dụng chúng cho đa số kiểu dữ liệu có sẵn được cho phép bởi C. Khi ta áp dụng phép / cho một số nguyên hay một ký tự, bất kỳ phần dư nào cũng bị cắt bỏ. Chẳng hạn, 5/2 bằng 2 trong phép chia nguyên.

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Chia lấy phần dư
--	Giảm 1 đơn vị
+	Tăng 1 đơn vị

1.2 Phép toán quan hệ và logic

Ý tưởng chính của toán tử quan hệ và toán tử Logic là đúng hoặc sai. Trong C mọi giá trị khác 0 được gọi là đúng, còn sai là 0. Các biểu thức sử dụng các toán tử quan hệ và Logic trả về 0 nếu sai và trả về 1 nếu đúng.

Toán tử	Ý nghĩa
Các toán tử quan hệ	
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Khác
Các toán tử Logic	
&&	AND
||	OR
!	NOT

Bảng chân trị cho các toán tử Logic:

P	q	p&&q	p||	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Các toán tử quan hệ và Logic đều có độ ưu tiên thấp hơn các toán tử số học. Do đó một biểu thức như: 10 > 1+ 12 sẽ được xem là 10 > (1 + 12) và kết quả là sai (0).

Ta có thể kết hợp vài toán tử lại với nhau thành biểu thức như sau:

10>5&&!(10<9)||3<=4 Kết quả là đúng

Thứ tự ưu tiên của các toán tử quan hệ là Logic

Cao nhất: !

> >= < <=

== !=

&& Thấp nhất:||

1.3 Chuyển đổi kiểu gía trị

Các toán tử Bitwise ý nói đến kiểm tra, gán hay sự thay đổi các Bit thật sự trong 1 Byte của Word, mà trong C chuẩn là các kiểu dữ liệu và biến char, int. Ta không thể sử dụng các toán tử Bitwise với dữ liệu thuộc các kiểu float, double, long double, void hay các kiểu phức tạp khác.

Toán tử	Ý nghĩa
&	AND
|	OR
^	XOR
~	NOT
>>	Dịch phải
<<	Dịch trái

Bảng chân trị của toán tử ^ (XOR)

p	q	p^q
0	0	0
0	1	1
1	0	1
1	1	0

1.4 Phép toán Tăng và giảm (++ & --)

Toán tử ++ thêm 1 vào toán hạng của nó và – trừ bớt 1. Nói cách khác:

x = x + 1 giống như ++x x = x – 1 giống như x—

Cả 2 toán tử tăng và giảm đều có thể tiền tố (đặt trước) hay hậu tố (đặt sau) toán hạng.

Ví dụ: x = x + 1 có thể viết x++ (hay ++x)

Tuy nhiên giữa tiền tố và hậu tố có sự khác biệt khi sử dụng trong 1 biểu thức. Khi 1 toán tử tăng hay giảm đứng trước toán hạng của nó, C thực hiện việc tăng hay giảm trước khi lấy giá trị dùng trong biểu thức. Nếu toán tử đi sau toán hạng, C lấy giá trị toán hạng trước khi tăng hay giảm nó. Tóm lại:

x = 10

y = ++x //y = 11

Tuy nhiên:

x = 10

y = x++; //y = 10

Thứ tự ưu tiên của các toán tử số học:

++ -- sau đó là * / % rồi mới đến + -

2. Câu lệnh gán và biểu thức.

2.1 Biểu thức

Biểu thức là một sự kết hợp giữa các toán tử (operator) và các toán hạng (operand) theo đúng một trật tự nhất định.

Mỗi toán hạng có thể là một hằng, một biến hoặc một biểu thức khác.

Trong trường hợp, biểu thức có nhiều toán tử, ta dùng cặp dấu ngoặc đơn () để chỉ định toán tử nào được thực hiện trước.

Ví dụ: Biểu thức nghiệm của phương trình bậc hai:

(-b + sqrt(Delta))/(2*a)

Trong đó 2 là hằng; a, b, Delta là biến.

2.2 Câu lệnh gán

Lệnh gán (assignment statement) dùng để gán giá trị của một biểu thức cho một biến.

Cú pháp: <Tên biến> = <biểu thức>

Ví dụ:

```
int main() {
int x,y;
x =10; /*Gán hằng số 10 cho biến x*/
y = 2*x; /*Gán giá trị 2*x=2*10=20 cho x*/
return 0;
}
```

Nguyên tắc khi dùng lệnh gán là kiểu của biến và kiểu của biểu thức phả giống nhau, gọi là có sự tương thích giữa các kiểu dữ liệu. Chẳng hạn ví dụ sau cho thấy một sự không tương thích về kiểu:

```
int main() {
int x,y;
x = 10; /*Gán hằng số 10 cho biến x*/
y = "Xin chao";
/*y có kiểu int, còn "Xin chao" có kiểu char* */
return 0;
}
```

Khi biên dịch chương trình này, C sẽ báo lỗi "*Cannot convert 'char *' to 'int'*" tức là C không thể tự động chuyển đổi kiểu từ char * (chuỗi ký tự) sang int.

Tuy nhiên trong đa số trường hợp sự tự động biến đổi kiểu để sự tương thích về kiểu sẽ được thực hiện. *Ví dụ*:

```
int main() { int x,y; float
   r; char ch;
   r = 9000;
   x = 10; /* Gán hằng số 10 cho biến x */
   y = 'd'; /* y có kiểu int, còn 'd' có kiểu char*/
   r = 'e'; /* r có kiểu float, 'e' có kiểu char*/
   ch = 65.7; /* ch có kiểu char, còn 65.7 có kiểu float*/
   return 0;
   }
```

Trong nhiều trường hợp để tạo ra sự tương thích về kiểu, ta phải sử dụng đến cách thức chuyển đổi kiểu một cách tường minh. Cú pháp của phép toán này như sau:

(Tên kiểu) <Biểu thức>

Chuyển đổi kiểu của <Biểu thức> thành kiểu mới <Tên kiểu>. Chẳng hạn như:

```
float f;
f = (float) 10/4; /* f lúc này là 2.5*/
```

Chú ý:

- Khi một biểu thức được gán cho một biến thì giá trị của nó sẽ thay thế giá trị cũ mà biến đã lưu giữ trước đó.
- Trong câu lệnh gán, dấu = là một toán tử; do đó nó có thể được sử dụng là một thành phần của biểu thức. Trong trường hợp này giá trị của biểu thức gán chính là giá trị của biến.

Ví dụ:

```
int x, y;
y = x = 3; /* y lúc này cùng bằng 3*/
```

- Ta có thể gán trị cho biến lúc biến được khai báo theo cách thức sau:

<Tên kiểu> <Tên biến> = <Biểu thức>;

Ví dụ: `int x = 10, y=x;`

3. Biểu thức điều kiện.

C có một toán tử rất mạnh và thích hợp để thay thế cho các câu lệnh của If-Then-Else. Cú pháp của việc sử dụng toán tử ? là:

E1 ? E2 : E3

Trong đó E1, E2, E3 là các biểu thức.

Ý nghĩa: Trước tiên E1 được ước lượng, nếu đúng E2 được ước lượng và nó trở thành giá trị của biểu thức; nếu E1 sai, E2 được ước lượng và trở thành giá trị của biểu thức.

Vídụ:
```
X = 10
Y = X > 9 ? 100 : 200
```

Thì Y được gán giá trị 100, nếu X nhỏ hơn 9 thì Y sẽ nhận giá trị là 200.
Đoạn mã này tương đương cấu trúc if như sau:

```
X = 10
if (X < 9) Y = 100
else Y = 200
```

4. Một số Ví dụ

4.1 Phép toán số học

Phép tóan cộng +

ví dụ:

int a=0,b=1,c=2;

a=12+1; //lấy 12+1=13 rồi gán cho a, kết quả(kq) a=13.

a=b+c; //lấy biến giá trị b cộng giá tri c tức 1+2 rồi gán cho a, kq a=3.

a=c+10; //lấy biến c cộng 10 rồi gán cho a, kq a=12.

a=a+10; //lấy biến a cộng 10 rồi gán cho a,kq a=12+10=22.

a=a+a; //lấy biến a cộng cho a rồi gán cho a,kq=22+22=44.

Chương trình sau cho phép bạn nhập vào 2 số và in kết quả phép tính cộng hai số ra màn hình.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float a,b,c;
printf("Chuong trinh tinh tong hai so\n")
printf("Ban hay nhap vao so thu nhat :");
scanf("%f",&a);
printf("Ban hay nhap vao so thu hai :");
scanf("%f",&b);
c=a+b;
printf("\nKet Qua của phep cong: %f+%f=%f",a,b,c);
return 0;
}
```

…….. ……

Tiếp theo các phép toán.

Phép tóan trừ -

ví dụ:

int a,b=0,c=0;

a=12-1; //lấy 12-1 gán vào a, Kết quả(kq) a=11.

b=a-3; //lấy a-3 gán vào b, kq b=11-3=8.

c=b-8; //lấy b-8 gán vào c, kq c=8-8=0.

Chương trình sau cho phép bạn nhập hai số sau đó xuất giá trị hiệu của hai số ra màn hình.

```
#include<stdio.h>
#include<conio.h>
int main()
{
float a,b,c;
printf("Chuong trinh tinh tong hai so\n")
printf("Ban hay nhap vao so thu nhat :");
scanf("%f",&a);
printf("Ban hay nhap vao so thu hai :");
scanf("%f",&b);
```

```
c=a-b;
printf("\nKet Qua của phep cong: %f-%f=%f",a,b,c);
return 0;
}
```

Phép toán nhân *

int a, b;
a=2*20; //lấy 2 nhân với 20 rồi gán vào cho a, kết quả (kq) a=40.
b=a*2; //lấy a=40 nhân với 2 rồi gán vào b, kq b=80.
a=a*a; // lấy a=40 nhân với a=40 rồi gán lại cho a, kq a=1600

Chương trình sau cho phép bạn nhập hai số sau đó in ra kết quả tích của hai số đó.

```
#include<stdio.h>
#include<conio.h>
int main()
{
       float a,b,c;
      printf("Chuong trinh tinh tich hai so\n")
      printf("Ban hay nhap vao so thu nhat :");
      scanf("%f",&a);
      printf("Ban hay nhap vao so thu hai :");
      scanf("%f",&b);
       c=a*b;
     printf("\nKet Qua của phep nhan: %f*%f=%f",a,b,c);
return 0;
}
```

4.2 Phép toán gán

Phép gán =

Phép tóan này dùng để gán một giá trị cho một biến với cú pháp sau:
a=value;

với **a phải** là một biến. value là biến hay là một hằng số bất kỳ.
ví dụ:

int a,b=10;
const int c=12;
a=b; //ok a=10.
a=11;//ok a=11.
11=a;//false 11 không phải là biến.
c=12;// false c là một hằng số không thể thay đổi được.

4.3 Phép toán logic

Phép toán so sánh bằng ==

Phép so sánh hai biểu thức A và B ký hiệu A==B là một mệnh đề. Mệnh đề này có giá trị là đúng(bằng 1) nếu biểu thức A bằng biểu thức B, ngược lại mệnh đề này sẽ có giá trị false (bằng 0).

int a,b;
a=1;
b=2;
b=a;//có giá trị là sai (0)

a=1;//có giá trị là đúng (1)
2=b;//có giá trị là đúng.
2=2;//có giá trị là đúng.

Ta có thể lấy giá trị của kết quả phép toán so sánh bằng cách gán giá trị trả về của phép so sánh vào một biến cụ thể.

```
int a;
a=(1==2);//Mệnh đề 1==2 là một mệnh đề sai,a sẽ được gán kết quả là a=0.
a=(a==0);//mệnh đề a==0 là một mệnh đề đúng do đó kết quả a=1.
```

Chương trình cho phép bạn nhập vào 2 số và in ra giá trị đúng sai của biểu thức so sánh hai số.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c;
printf("Chuong trinh tinh tong hai so\n");
printf("Ban hay nhap vao so thu nhat :");
scanf("%d",&a);
printf("Ban hay nhap vao so thu hai :");
scanf("%d",&b);
c=(b==a);
printf("\nKet Qua cua phep so sanh: (%d == %d)=%d",a,b,c);
return 0;
}
```

Phép so sánh không bằng !=

Phép so sánh A!=B là một mệnh đề, mệnh đề này đúng(bằng 1) khi A khác B (hay A không bằng B) và có giá trị sai (bằng 0) khi A==B.

```
int a,b;
a=1;
b=2;
1!=2; //Mệnh đề 1 khác 2 có giá trị đúng (1).
a!=b;//Mệnh đề a khác b có giá trị đúng.
2!=b;//Mệnh đề 2 khác b có giá trị sai (0).
```

Ta có thể lưu giá trị của phép so sánh không bằng vào biến.

Code:

```
int a;
a=(1!=2);//1 khác 2 là một mệnh đề đúng do đó biến a sẽ có giá trị 1.
a=(1!=a);//1 khác a (a=1) là một mệnh đề sai do đó biến a sẽ có giá trị là 0.
```

Chương trình cho phép bạn nhập vào hai số, in ra giá trị đúng sai của phép toán so sách khác nhau giữa hai số.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c;
printf("Chuong trinh tinh tong hai so\n");
```

```
printf("Ban hay nhap vao so thu nhat :");
scanf("%d",&a);
printf("Ban hay nhap vao so thu hai :");
scanf("%d",&b);
c=(b!=a);
printf("\nKet Qua cua phep so sanh: (%d != %d)=%d",a,b,c);
return 0;
}
```

Phép toán so sánh lớn hơn >, lớn hơn bằng >=, nhỏ hơn < nhỏ hơn bằng<=

1>2; //Mệnh đề(MĐ) có giá trị Sai.
1>=2; //MĐ có giá trị Sai.
1<2; //MĐ có giá trị đúng.
1<=2 //Mệnh đề có giá trị đúng.

Phép toán Or ||

Cho A B là các mệnh đề mệnh phép toán A or B (A||B) có kết quả là một mệnh đề, và mệnh đề này chỉ sai khi và chỉ khi A và B cùng là hai mệnh đề sai.

Ta có bảng chân trị sau:
Code:

A \ B	0	1
0	0	1
1	1	1

ví dụ:
Code:

(1<2)||(1==2); //Mệnh đề kết quả có giá trị sai do 1<2 và 1==2 là hai mệnh đề sai.
(1==1)||(1==2);// Mệnh đề kết quả là một mệnh đề đúng do 1==1 là MĐ đúng và 1==2 là một mệnh đề sai.

Phép toán and &&

Hai mệnh đề A và B, phép toán and hai mệnh đề A và B (A&&B) là một mệnh đề,mệnh đề này có giá trị là đúng khi và chỉ khi A và B là hai mệnh đề đúng.

4.4 Các ví dụ về biểu thức

Ví dụ

2
x
$3 + 7$
$2 \times y + 5$
$2 + 6 \times (4 - 2)$
$z + 3 \times (8 - z)$

Ví dụ :

Roland nặng 70 kilograms, và Mark nặng k kilograms. Viết một biểu thức cho tổng cân nặng của họ. Tổng cân nặng của hai người tính bằng kilograms là $\mathbf{70 + k}$.

Ví dụ:

Tính giá trị biểu thức **4 × *z* + 12** với ***z* = 15**.
Chúng ta thay thế mọi z với giá trị 15, và đơn giản hóa biểu thức theo quy tắc: thi hành phép toán trong dấu ngoặc trước tiên, kế đến lũy thừa, phép nhân và chia rồi phép cộng và trừ.
4 × *z* + 12 trở thành
4 × 15 + 12 = (phép nhân thực hiện trước phép cộng)
60 + 12 =
72

4.5 Phép toán tăng hoặc giảm

Ví dụ	**Cách thực thi**
iunt a=1; int b=1;	a = 1, b chưa khởi tạo a = 1, b = 1
a+=b++;	Thực hiện phép toán a+=b trước, sau đó mới thực hiện phép toán b++. Tức là a=2, b=2.
a+=++b;	Thực hiện phép toán ++b trước, sau đó mới thực hiện phép toán a+=b. Tức là b=2, a=3.

Lưu ý:

Phép toán	Ý nghĩa
a++;	Thực hiện phép toán trước, sau đó mới thực hiện toán tử
++a;	Thực hiện toán tử trước, sau đó mới thực hiện phép toán
a–;	Tương tự a++;
–a;	Tương tự ++a;

5. Thực hành

5..1 Mục đích, yêu cầu:

Kết thúc bài học này, bạn có thể:

- Sử dụng được các toán tử số học, so sánh và luận lý
- Chuyển đổi các kiểu dữ liệu
- Nắm được thứ tự ưu tiên giữa các toán tử.

Các bước trong phần này đã được nghiên cứu kỹ và giải thích chi tiết để chúng ta có thể hiểu rõ và áp dụng chúng một cách hoàn chỉnh.Ta hãy theo các bước cẩn thận.

5.2 Nội dung thực hành

Bài 1:

Trong chương này, ta sẽ viết một chương trình tính toán tiền lãi đơn giản (lãi thuần chưa tính tiền vốn vào) khi ta vay tiền.

Công thức để tính toán là **p * n * r /100**. Ở đây '**p**' có nghĩa là tiền vốn, '**n**' là số năm và '**r**' có nghĩa là tỉ lệ lãi suất.

Chương trình khai báo ba biến số thực 'float' có tên là p, n và r. Chú ý rằng, các biến được khai báo trên cùng một dòng mã thì ta dùng dấu phẩy (,) để phân chia chúng với nhau. Mỗi biến trên được gán một giá trị.

Xét dòng mã sau:

```
printf("\nAmount la: %f", p*n*r/100);
```

Trong *printf()* ở trên, chúng ta đã dùng '%f' để hiển thị giá trị của biến kiểu float (số thực), giá trị biến này là p*n*r/100, công thức dùng tính lãi đơn giản được đưa vào trong *printf()*. Đó là p, n và r được nhân với nhau và kết quả được chia cho 100. Như vậy *printf()* sẽ hiển thị lãi đơn.

Gọi Borland C.

Code mẫu: Tính lãi đơn

1. Tạo ra một tập tin mới.

2. Gõ đoạn mã sau trong 'Edit window':

```
#include <stdio.h>
#include <conio.h>

void main()
{
 float p, n, r;
 clrscr();
 p = 1000;
 n = 2.5;
 r = 10.5;

     printf("\n Lai suat la: %f", p*n*r/100);
}
```

3. Lưu tập tin với tên *simple.cpp*.

4. Biên dịch tập tin *simple.cpp*.

5. Thực thi chương trình *simple.cpp*.

6. Trở về trình soạn thảo.

Kết quả:

```
Lai suat la: 262.500000
```

Bài 2 Dùng toán tử số học

Trong phần này ta sẽ viết một chương trình có sử dụng toán tử số học.

Chương trình này khai báo bốn biến số nguyên tên là a, b, c và d. Giá trị sẽ gán cho các biến a, b và c là: a = 50, b = 24, c = 68

Xét dòng mã sau:

```
d = a*b+c/2;
```

a nhân với b. c chia cho 2. Kết quả của a*b được cộng với thương số của c/2. Giá trị này sẽ gán cho d qua toán tử (=). Biểu thức được tính như sau:

1. 50 * 24 = 1200
2. 68 / 2 = 34
3. 1200 + 34 = 1234
4. d = 1234

'printf( )' : hiển thị giá trị của biến d.

Xét biểu thức:

```
d = a*(b+c+(a-c)*b);
```

Ở đây dấu ngoặc đơn trong cùng có độ ưu tiên cao nhất. Do vậy, (a-c) được tính trước. Sau đó, tính tới các dấu ngoặc đơn ngoài. Kết quả của (a-c) được nhân cho b bởi vì '*' có độ ưu tiên cao hơn '-' và '+'. Biểu thức được tính như dưới đây:

1. d = 50 * (24 + 68 + (50 - 68) * 24)
2. d = 50 * (24 + 68 + (-18) * 24)
3. d = 50 * (24 + 68 + (-432))
4. d = 50 * (92 - 432)
5. d = 50 * (-340)
6. d = -17000

```
Các biểu thức khác được tính tùy vào các toán tử đã được dùng. Kết quả được
hiển thị bởi lệnh 'printf()'.
```

1.Tạo mới một tập tin.

2. Gõ đoạn mã sau trong 'Edit window':

```
#include <stdio.h>
#include <conio.h>

void main()
{
```

```
   int a,b,c,d;
   clrscr();
   a = 50;
   b = 24;
   c = 68;
   d = a*b+c/2;
   printf("\n Gia tri sau a*b+c/2 la: %d", d);
   d = a%b;
   printf("\n Gia tri sau a mod b la: %d", d);
   d = a*b-c;
   printf("\n Gia tri sau a*b-c la: %d", d);
   d = a/b+c;
   printf("\n Gia tri sau a/b+c la: %d", d);

   d = a+b*c;
   printf("\n Gia tri sau a+b*c la: %d", d);

   d = (a+b)*c;
   printf("\n Gia tri sau (a+b)*c la: %d", d);

   d = a*(b+c+(a-c)*b);
   printf("\n Gia tri sau a*(b+c+(a-c)*b) la: %d", d);
}
```

3. Lưu tập tin với tên *sohoc.cpp*

4. Biên dịch tập tin *sohoc.cpp*

5. Thực thi chương trình *sohoc.cpp*

6. Trở về trình soạn thảo.

Kết quả xuất:

```
   Gia tri sau  a*b+c/2 la: 1234
   Gia tri sau a mod b la: 2
   Gia tri sau  a*b-c la: 1132
   Gia tri sau  a/b+c la: 70
   Gia tri sau  a+b*c la: 1682
   Gia tri sau  (a+b)*c la: 5032
   Gia tri sau a*(b+c+(a-c)+b) la: -17000
```

Bài 3 Dùng toán tử so sánh và luận lý

Trong phần này chúng ta sẽ viết một chương trình sử dụng toán tử so sánh và toán tử luận lý.

Ba biến số nguyên tên là a, b và c được khai báo trong chương trình này. Các giá trị gán cho biến như sau: a = 5, b = 6 & c = 7.

Xét những dòng mã sau:

1. `a + b >= c;`

Đầu tiên, a+b sẽ được tính (toán tử số học có độ ưu tiên cao hơn toán tử so sánh), kết quả là 11. Kế đến giá trị 11 được so sánh với c. Kết quả là 1(true) bởi vì 11 > 7.

2. Xét biểu thức khác:

```
a > 10 && b <5;
```

Tính toán đầu tiên sẽ là a> 10 và b<5, bởi vì toán tử so sánh (> <) có quyền ưu tiên cao hơn toán tử luận lý AND (&&). Tính toán theo sau:

1.5 > 10 && 6<5
2.FALSE && FALSE
3.FALSE tức là, 0

1. Tạo một tập tin mới.

2. Gõ đoạn mã sau vào 'Edit window':

```
#include <stdio.h>
#include <conio.h>

void main()
{
          int a = 5, b = 6, c = 7;
          printf ("int a = 5, b = 6, c = 7;\n");
          printf("Gia tri cua a > b la \t%i\n\n", a > b);
          printf("Gia tri cua b < c la \t%i\n\n", b < c);
          printf("Gia tri cua a + b >= c la \t%i\n\n", a + b >= c);
          printf("Gia tri cua a - b <= b - c la \t%i\n\n", a-b <= b-c);
          printf("Gia tri cua b-a == b - c la \t%i\n\n", b - a == b -
c);
          printf("Gia tri cua a*b != c * c la \t%i\n\n", a * b < c * c);
          printf("Ket qua cua a>10 && b<5 = %d\n\n", a>10 && b<5);
          printf("Ket qua cua a > 100 || b < 50 = %d\n\n", a>100 ||
b<50);
}
```

3. Lưu tập tin với tên *sosanh.c*pp

4. Biên dịch tập tin *sosanh.c*pp

5. Thực thi chương trình *sosanh.cpp*

6. Trở về trình soạn thảo.

Kết quả xuất:

```
     int a = 5, b = 6, c = 7;

     Gia tri cua  a > b la                    0

     Gia tri cua  b < c  la                  1
```

```
Gia tri cua  a + b >= c  la            1

Gia tri cua  a - b <= b - c  la        1

Gia tri cua  b - a == b - c  la        0

Gia tri cua  a * b != c * c  la         1

Ket qua cua  a > 10 && b < 5 =  0

Ket qua cua  a > 100 || b < 50 = 1
```

Bài 4 Chuyển đổi kiểu dữ liệu

Trong phần này, ta sẽ viết một chương trình để hiểu rõ việc chuyển đổi kiểu dữ liệu.

Trong biểu thức đầu tiên, tất cả đều là số nguyên 'int' 40 / 17 * 13 / 3 sẽ có kết quả là 8 (40 / 17 làm tròn ra 2, 2 * 13 = 26, 26 / 3 làm tròn ra 8)

Biểu thức thứ hai như sau:

1. Định giá trị: 40 / 17 * 13 / 3.0
2. 40 / 17 làm tròn kết quả là 2
3. 2 * 13 = 26
4. Nhưng vì có số 3.0 đã ép kiểu phép chia cuối cùng thành số kiểu double, vì vậy 26.0 / 3.0 = 8.666667

Trong biểu thức thứ ba:

Nếu chúng ta di chuyển dấu chấm thập phân sang số 13 (40 / 17 * 13.0 / 3), kết quả vẫn sẽ là 8.666667 bởi vì:

1. 40 / 17 làm tròn là 2
2. Số 13.0 ép kiểu dữ liệu phép nhân thành double nhưng kết quả vẫn là 26 vì 2.0 * 13.0 = 26.0
3. Và 26.0 ép phép chia cuối cùng thành kiểu double, vì vậy 26.0 / 3.0 = 8.666667

Trong biểu thức cuối:

nếu chúng ta di chuyển dấu chấm thập phân sang số 17 (40 / 17.0 * 13 / 3), kết quả bây giờ sẽ là 10.196078 bởi vì:

1. 17.0 ép kiểu của phép chia đầu thành kiểu double và 40.0 / 17.0 = 2.352941
2. 2.352941 * 13.0 = 30.588233
3. và 30.588233 / 3.0 = 10.196078

1. Tạo một tập tin mới.

2. Gõ đoạn mã sau vào 'Edit window':

```
#include <stdio.h>
#include <conio.h>

void main()
{
 clrscr();
 printf("40/17*13/3 = %d", 40/17*13/3);
 printf("\n\n40/17*13/3.0 = %lf", 40/17*13/3.0);
 printf("\n\n40/17*13.0/3 = %lf", 40/17*13.0/3);
 printf("\n\n40/17.0*13/3 = %lf", 40/17.0*13/3);
}
```

3. Lưu tập tin với tên *type.c*.

4. Biên dịch tập tin *type.c*.

5. Thực thi chương trình *type.c*.

6. Trở về trình soạn thảo.

Kết quả xuất:

```
40/17*13/3 = 8

40/17*13/3.0 = 8.666667

40/17*13.0/3 = 8.666667

40/17.0*13/3 = 10.19607
```

Bài 4. CÁC CÂU LỆNH ĐIỀU KHIỂN

Mục tiêu:

– Trình bày được cú pháp và công dụng của các câu lệnh rẽ nhánh, lệnh lặp và các lệnh kết thúc vòng lặp;

– Phân biệt được công dụng giữa các câu lệnh if và switch ; giữa câu lệnh lặp for với while và do ... while. Từ đó có thể vận dụng vào việc phân tích bài tập và viết chương trình;

– Vận dụng được các lệnh kết thúc vòng lặp;

– Viết được các chương trình với các câu lệnh điều khiển;

– Rèn luyện đức tính làm việc khoa học, khả năng làm việc theo nhóm.

1. Câu lệnh rẽ nhánh

1.1. Câu lệnh if

1.1.1 Khái niệm

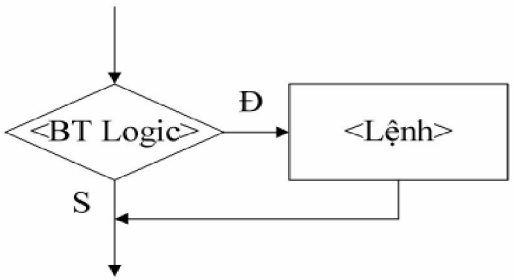
Câu lệnh if là câu lệnh làm cho chương trình thực hiện hay không thực hiện câu lệnh nào đó tùy vào điều kiện nêu ra.

1.1.2 Cú pháp

Câu lệnh If (thiếu)

Thực hiện <Lệnh> nếu <BT Logic> đúng (true, khác 0).
Ngược lại, <BT Logic> sai (false, bằng 0) sẽ không làm gì cả.

Sơ đồ khối



Cú pháp

```
if (<BT Logic>)
    <Lệnh>;
```

Trong đó:

- <BT Logic> cho kết quả đúng (true, khác 0) hoặc sai (false, bằng 0) và được đặt trong cặp dấu ngoặc đơn ( ).
- <Lệnh> là câu lệnh đơn hoặc khối lệnh (kẹp các câu lệnh đơn giữa { và }).

Ví dụ:

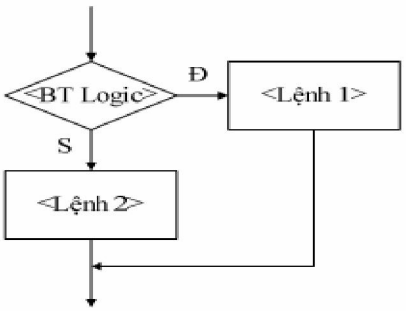
Nếu a khác không thì xuất thông báo "a khác 0".

```
if (a != 0)
printf("a khac 0.");
```

Câu lệnh If (đủ)

Thực hiện <Lệnh 1> nếu <BT Logic> đúng (khác 0).
Ngược lại, <BT Logic> sai (bằng 0) sẽ thực hiện <Lệnh 2>.

Sơ đồ khối



Cú pháp

```
if (<BT Logic>)
    <Lệnh 1>;
else
    <Lệnh 2>;
```

Ví dụ: Tùy theo giá trị của a để xuất thông báo tương ứng.

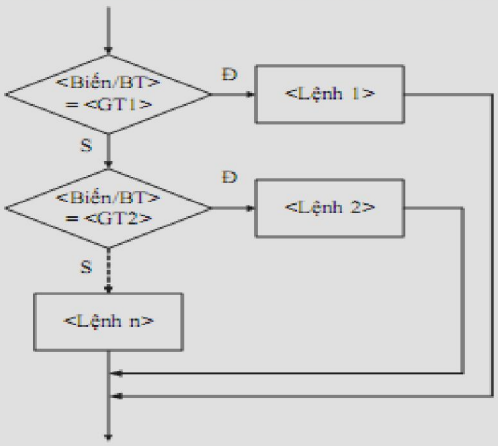
```
if (a != 0)
printf("a khac 0.");
else

printf("a bang 0.");
```

1.2. Câu lệnh switch

1.2.1 Khái niệm

Câu lệnh switch là câu lệnh làm chương trình thực hiện chọn lựa một trong nhiều hành động để thực hiện.

1.2.2 Cú pháp

<table>
<tr><td colspan="2">Câu lệnh switch
<Biến/BT> bằng <GT> nào sẽ thực hiện <Lệnh> đó.
Nếu không tìm thấy <GT> tương ứng sẽ thực hiện <Lệnh n>.</td></tr>
<tr><td>Sơ đồ khối</td><td> 
</td></tr>
<tr><td>Cú pháp</td><td><pre>switch (<Biến/BT>)
{
    case <GT1> : <Lệnh 1>;[break;]
    case <GT2> : <Lệnh 2>;[break;]
    ...
    [default:
        <Lệnh n>;]
}</pre></td></tr>
</table>

Trong đó:

- <Biến/BT> là biến hay biểu thức cho ra dữ liệu kiểu rời rạc, hữu hạn, đếm được như các kiểu số nguyên, ký tự, liệt kê và được đặt trong cặp dấu ngoặc đơn ( ). **Không được sử dụng kiểu số thực.**
- <Lệnh 1>, …, <Lệnh n> là một hay nhiều câu lệnh đơn.
- Cuối mỗi trường hợp (case) sẽ có hoặc không có lệnh break. Khi gặp lệnh này, lệnh switch sẽ kết thúc.
- Phần default có thể có hoặc không. Nếu không tìm thấy trường hợp nào phù hợp, phần này sẽ được thực hiện.

Ví dụ:

Lệnh switch sau sẽ xuất thông báo “Một” nếu a bằng 1, xuất thông báo “Hai” nếu a bằng 2, xuất thông báo “Ba” nếu a bằng bằng 3. Nếu không sẽ xuất “a<1 hoặc a>3” và “Khong biet doc!”.

```
switch (a)
{
case 1 : printf(“Mot”); break;
case 2 : printf(“Hai”); break;
case 3 : printf(“Ba”); break;
default:
printf(“a<1 hoac a>3”);
printf(“\nKhong biet doc!”);
}
```

1.3. Ví dụ

Ví dụ 1: Viết chương trình nhập vào 2 số nguyên a, b. Tìm và in ra số lớn nhất.

a. Phác họa lời giải

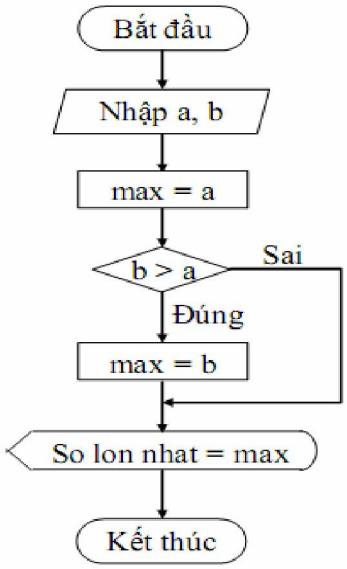
Trước tiên ta cho giá trị ***a là giá trị lớn nhất bằng cách gán a cho max*** (max là biến được khai báo cùng kiểu dữ liệu với a, b). Sau đó so sánh b với a, ***nếu b lớn hơn a ta gán b cho max*** và cuối cùng ta ***được kết quả max là giá trị lớn nhất***.

b. Mô tả quy trình xử lý (giải thuật)

Ngôn ngữ tự nhiên	Ngôn ngữ C
- Khai báo 3 biến a, b, max kiểu số nguyên	- int ia, ib, imax;
- Nhập vào giá trị a	- printf("Nhap vao so a: "); scanf("%d", &ia);
- Nhập vào giá trị b	- printf("Nhap vao so b: "); scanf("%d", &ib);
- Gán a cho max	- imax = ia;
- Nếu b > a thì gán b cho max	- if (ib > ia) imax = ib;
	- printf("So lon nhat = %d.\n", imax);

➢ **Biểu thức luận lý phải đặt trong cặp dấu ( ). if ib > ia → báo lỗi**

c. Mô tả bằng lưu đồ



d. Viết chương trình

```
/* Chuong trinh tim so lon nhat tu 2 so nguyen a, b */

#include <stdio.h>

#include <conio.h>

void main(void)

{
int ia, ib, imax; printf("Nhap vao so a: ");
scanf("%d", &ia);
printf("Nhap vao so b: ");
scanf("%d", &ib);

imax = ia;
```

```
if (ib>ia)
imax = ib;
printf("So lon nhat = %d.\n", imax);
getch();
}
```

Ví dụ 2: Viết chương trình nhập vào 2 số nguyên a, b. Nếu a lớn hơn b thì hoán đổi giá trị a và b, ngược lại không hoán đổi. In ra giá trị a, b.

a. Phác họa lời giải

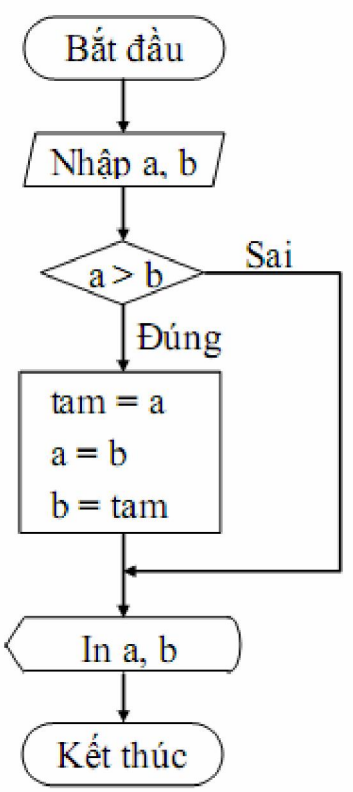
Nếu giá trị a lớn hơn giá trị b, bạn phải hoán chuyển 2 giá trị này cho nhau (nghĩa là a sẽ mang giá trị b và b mang giá trị a) bằng cách đem ***giá trị a gởi (gán) cho biến tam*** (biến tam được khai báo theo kiểu dữ liệu của a, b), kế đến bạn ***gán giá trị b cho a*** và cuối cùng bạn ***gán giá trị tam cho b***, rồi in ra a, b.

b. Mô tả quy trình thực hiện (giải thuật)

Ngôn ngữ tự nhiên	**Ngôn ngữ C**

- Khai báo 3 biến a, b, tam kiểu số nguyên
- Nhập vào giá trị a
- Nhập vào giá trị b
- Nếu a > b thì tam = a;
a = b;
b = tam;
	- int ia, ib, itam;
	- printf("Nhap vao so a: ");
	scanf("%d", &ia);
	- printf("Nhap vao so b: ");
	scanf("%d", &ib);
	- if (ia > ib)
	{
	itam = ia;
	ia = ib;
	ib = itam;
	}
	- printf("%d, %d\n", ia, ib);

c. Mô tả bằng lưu đồ



d. Viết chương trình

```
/* Chuong trinh hoan vi 2 so a, b neu a > b */

#include <stdio.h>
#include <conio.h>
void main(void)
{
int ia, ib, itam;
printf("Nhap vao so a: ");
scanf("%d", &ia);
printf("Nhap vao so b: ");
scanf("%d", &ib);
if (ia>ib)
{
itam = ia;    //hoa vi a va b
ia = ib;
ib = itam;
}
printf("%d, %d.\n", ia, ib);
getch();
}
```

Ví dụ 3: Viết chương trình nhập vào 2 số nguyên a, b. In ra thông báo "a lớn hơn b" nếu a>b, in ra thông báo "a nhỏ hơn b" nếu a<b, in ra thông báo "a băng b" nếu a=b.

a. Phác họa lời giải

Trước tiên so sánh a với b. Nếu a > b thì in ra thông báo "a lớn hơn b", ngược lại nếu a < b thì

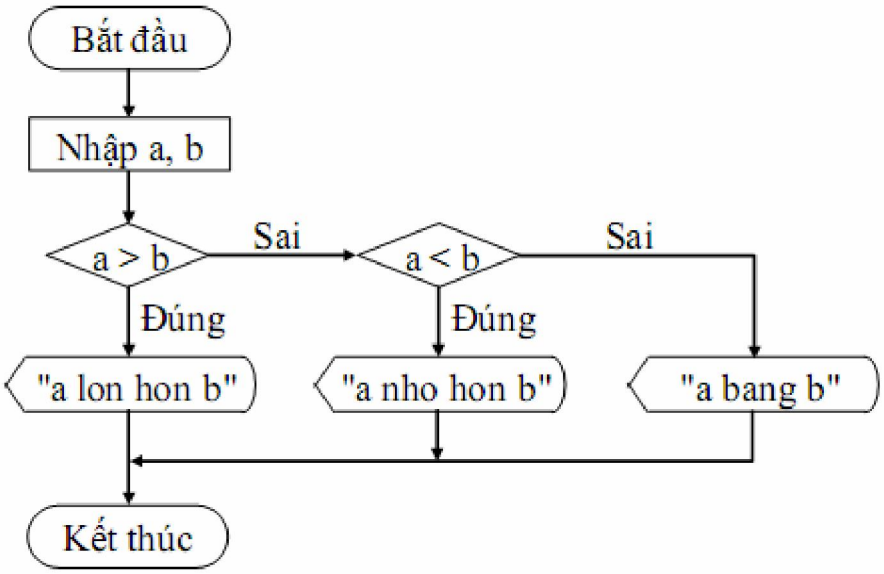
in ra thông báo "a nhỏ hơn b", ngược với 2 trường hợp trên thì in ra thông báo "a bằng b"

b. Mô tả quy trình thực hiện (giải thuật)

Ngôn ngữ tự nhiên	**Ngôn ngữ C**

- Khai báo 2 biến a, b kiểu số nguyên
- Nhập vào giá trị a
 	
- Nhập vào giá trị b
 	
- Nếu a > b thì
in ra thông báo "a lớn hơn b" Ngược lại Nếu a < b thì
in ra thông báo "a nhỏ hơn b" Ngược lại thì
in ra thông báo "a bằng b"
	- int ia, ib;
	- printf("Nhap vao so a: ");
	scanf("%d", &ia);
	- printf("Nhap vao so b: ");
	scanf("%d", &ib);
	- if (ia > ib)
	printf("a lon hon b.\n");
	else if (ia < ib)
	printf("a nho hon b.\n");
	else
	printf("a bang b.\n");

c. Mô tả bằng lưu đồ



d. Viết chương trình

```
/* Chuong trinh nhap vao 2 so nguyen a, b. In ra thong bao a > b, a < b, a = b
*/

#include <stdio.h>
#include <conio.h>

void main(void)
{
int a, b;
printf("Nhap vao so a: ");
scanf("%d", &a);
printf("Nhap vao so b: ");
scanf("%d", &b);
```

```
if (a>b)
printf("a lon hon b.\n");
else if (a<b)
printf("a nho hon b.\n");
else
printf("a bang b.\n");
getch();
}
```

Ví dụ 4: Viết chương trình nhập vào tháng và in ra quý. (tháng 1 -> quý 1, tháng 10 -> quý 4)

a. Phác họa lời giải

Nhập vào giá trị tháng, kiểm tra xem tháng có hợp lệ (trong khoảng 1 đến 12). Nếu hợp lệ in ra quý tương ứng (1->3: quý 1, 4->6: quý 2, 7->9: quý 3, 10->12: quý 4).

b. Viết chương trình

```
/* Chuong trinh nhap vao thang. In ra quy tuong ung */

#include <stdio.h>
#include <conio.h>

void main(void)
{
int ithang;
printf("Nhap vao thang: ");
scanf("%d",&ithang);
 switch(ithang)
{
case 1: case 2: case 3 : printf("Quy 1.\n");
                         break;
case 4: case 5: case 6:  printf("Quy 2.\n");
                         break;
case 7: case 8: case 9:  printf("Quy 3.\n");
                         break;
case 10: case 11: case 12:   printf("Quy 4.\n");
     break;
default                  : printf("Ban phai nhap vao so trong khoang 1..12\n");
};
getch();
}
```

2. Câu lệnh lặp

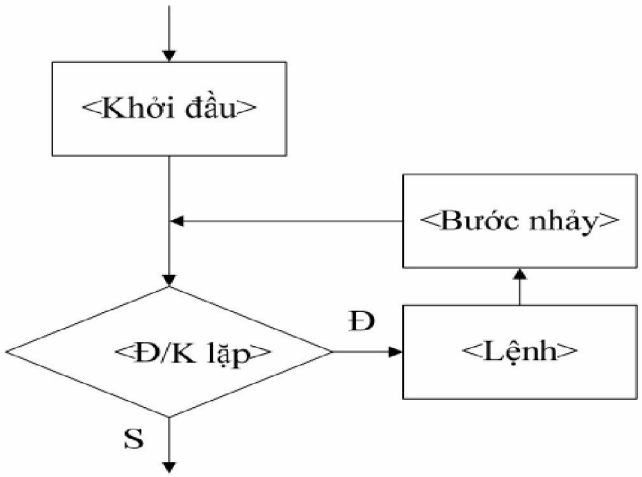
2.1 Câu lệnh For

2.1.1 Khái niệm

Câu lệnh for là câu lệnh làm cho chương trình thực hiện lặp lại một hay nhiều hành

động một số lần nhất định.

2.1.2 Cú pháp

Câu lệnh For	
Sơ đồ khối	
Cú pháp	`for (<Khởi đầu>;<Đ/K lặp>;<Bước nhảy>)` `<Lệnh>;`

Trong đó:

- <Khởi đầu>: là một biểu thức C bất kỳ, thường là một câu lệnh gán để đặt cho biến đếm một giá trị cụ thể.
- <Đ/K lặp>: là một biểu thức C bất kỳ, thường là một biểu thức quan hệ cho kết quả true-false. Nếu <Đ/K lặp> nhận giá trị false (bằng 0), câu lệnh for sẽ kết thúc. Ngược lại, <Lệnh> sẽ được thực hiện.
- <Bước nhảy>: là một biểu thức C bất kỳ, thường là một biểu thức nhằm tăng giá trị biến đếm trong biểu thức khởi đầu. Biểu thức này sẽ được thi hành sau khi <Lệnh> được hiện.
- <Lệnh>: là câu lệnh đơn hoặc khối lệnh.

Ví dụ:

Câu lệnh sau sẽ in các số từ 0 đến 9 ra màn hình.

```
int i;
for (i = 0; i < 10; i++)
printf("%d\n", i);
```

Ta cũng có thể vừa khai báo và khởi tạo biến trong phần khởi đầu của for:

```
for (int i = 0; i < 10; i++)
printf("%d\n", i);
```

Thực hiện nhiều câu lệnh đồng thời (khối lệnh):

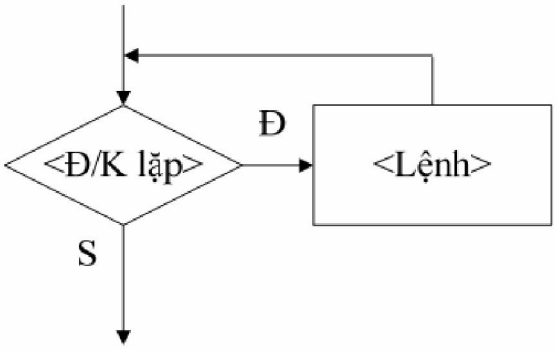
```
for (int i = 0; i < 10; i++)
{
printf("%d", i);
printf("\n");
}
```

2.2 Câu lệnh while

2.2.1 Khái niệm

Câu while là câu lệnh làm cho chương trình thực hiện lặp lại nhiều lần một số hành động cho trước trong khi vẫn còn thỏa một điều kiện để tiếp tục quá trình lặp.

2.2.2 Cú pháp

Câu lệnh while	
Sơ đồ khối	
Cú pháp	`while (<Đ/K lặp>)` `  <Lệnh>;`

Trong đó :

- <Đ/K lặp>: là một biểu thức C bất kỳ, thường là một biểu thức quan hệ cho kết quả true-false và được đặt trong cặp dấu ngoặc đơn ( ). Nếu <Đ/K lặp> nhận giá trị false (bằng 0), câu lệnh for sẽ kết thúc. Ngược lại,

<Lệnh> sẽ được thực hiện.

- <Lệnh>: là câu lệnh đơn hoặc khối lệnh.

Ví dụ, vòng lặp sau sẽ xuất giá trị của n bắt đầu từ 0 cho đến 9.

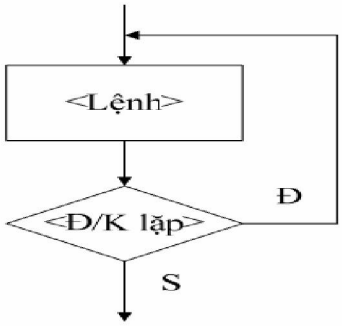
```
int n = 0;
while (n < 10)
{
printf("%d\n", n);
n++;
}
```

2.3 Câu lệnh do…while

2.3.1 Khái niệm

Câu lệnh do... while là câu lệnh làm cho chương trình thực hiện lặp lại nhiều lần một số hành động cho trước trong khi vẫn còn thỏa một điều kiện để tiếp tục quá trình lặp. Câu lệnh này ngược với câu lệnh while ở trên một chỗ là điều kiện lặp được kiểm tra sau khi các lệnh đã được thực hiện.

2.3.2 Cú pháp

Câu lệnh do... while	
Sơ đồ khối	
Cú pháp	do <Lệnh>; while (<Đ/K lặp>);

<Đ/K lặp> và <Lệnh> giống như lệnh while ở trên. Ví dụ, câu lệnh sau sẽ xuất các số từ 0 đến 9.

```
n = 0;
do
{
printf("%d\n", n);
n++;
}
while (n < 10);
```

Ứng dụng của câu lệnh này là tạo vòng lặp nhập dữ liệu cho một biến sao cho biến đó có giá trị giới hạn trong một phạm vi nào đó. Đoạn chương trình sau đây yêu cầu người sử dụng nhập vào giá trị cho biến n trong đoạn từ 1 đến 100.

```
int n;
do
{
printf("Nhap n : ");
scanf("%d", &n);
}
while (n < 1 || n > 100);
```

2.4 Ví dụ

Ví dụ 1:Viết chương trình in ra câu "Vi du su dung vong lap for" 3 lần

```
1   /* Chuong trinh in ra cau "Vi du su dung vong lap for" 3 lan */
2
3   #include <stdio.h>
4   #include <conio.h>
5
6   #define MSG "Vi du su dung vong lap for.\n"
7
8   void main(void)
9   {
10  int i;
11  for(i = 1; i<=3; i++)  /hoac for(i = 1; i<=3; i+=1)
12  printf("%s", MSG);
13  getch();
14  }
```

Ví dụ 2: Viết chương trình in ra câu "Vi du su dung vong lap while" 3 lần

```
1   /* Chuong trinh in ra cau "Vi du su dung vong lap while" 3 lan
2   */
3
4   #include <stdio.h>
5   #include <conio.h>
6
7   #define MSG "Vi du su dung vong lap while.\n"
8
9   void main(void)
10  {
11  int i = 0;
12  while (i++ < 3)
13  printf("%s", MSG);
14  getch();
    }
```

Kết quả in ra màn hình

Vi du su dung vong lap while.
Vi du su dung vong lap while.
Vi du su dung vong lap while.
_

Bạn thay 2 dòng 11 và 12 bằng câu lệnh
while(printf("%s", MSG), ++i < 3);
Chạy lại chương trình và quan sát kết quả.

Ví dụ 2: Viết chương trình kiểm tra password.

```
1       /* Chuong trinh kiem tra mat khau */
2
3       #include <stdio.h>
4
5       # define PASSWORD12345
6
7       void main(void)
8       {
9       int in;
10      do
11      {
12      printf("Nhap vao password: ");
13      scanf("%d", &in);
14      } while (in != PASSWORD)
15      }
```

Nhap vao password: 1123
Nhap vao password: 12346
Nhap vao password: 12345

Bạn thay các dòng từ 10 đến 14 bằng câu lệnh:

do{}while(printf("Nhap vao password: "),
scanf("%d", &in),
in != PASSWORD);

Chạy lại chương trình và quan sát kết quả.

3. Câu lệnh dừng vòng lặp

3.1 Câu lệnh break, continue

a) Câu lệnh Break:

Thông thường lệnh break dùng để thoát khỏi vòng lặp không xác định điều kiện dừng hoặc bạn muốn dừng vòng lặp theo điều kiện do bạn chỉ định. Việc dùng lệnh break để thoát khỏi vòng lặp thường sử dụng phối hợp với lệnh if. Lệnh break dùng trong for, while, do…while, switch. Lệnh break thoát khỏi vòng lặp chứa nó.

Ví dụ: Sử dụng lệnh break trong switch để nhảy bỏ các câu lệnh kế tiếp còn lại.

b) Lệnh Continue:

Được dùng trong vòng lặp for, while, do…while. Khi lệnh continue thi hành quyền điều khiển sẽ trao qua cho biểu thức điều kiện của vòng lặp gần nhất. Nghĩa là lộn ngược lên đầu vòng lặp, tất cả những lệnh đi sau trong vòng lặp chứa continue sẽ bị bỏ qua không thi hành.

Ví dụ:

```
/* Nhap vao 1 day so nguyen tu ban phim den khi gap so 0 thi dung. In ra tong
cac so nguyen duong */

#include <stdio.h>
#include <conio.h>

void main(void)
{
int in, itong = 0;
for(; ;)
```

```
{
printf("Nhap vao 1 so nguyen: ");
scanf("%d", &in);
if (in < 0)
continue;      //in < 0 quay nguoc len dau vong lap
if (in == 0)
break;         //in = 0 thoat vong lap
itong += in;
}
printf("Tong: %d.\n", itong);
getch();
}
```

3.2 Câu lệnh goto

a. Ý nghĩa

Một dạng khác của rẽ nhánh là câu lệnh nhảy **goto** cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình. Nhãn là một tên gọi do NSD tự đặt theo các qui tắt đặt tên gọi. Lệnh goto thường được sử dụng để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này thường được sử dụng rất hạn chế.

b. Cú pháp

Goto <nhãn> ;

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng sau nhãn và dấu hai chấm (:).

3.3 Ví dụ

Ví dụ1: Viết chương trình nhập vào các số nguyên và tính tổng chúng, khi nhập vào 0 thì dừng và in ra kết quả.

```
#include <stdio.h>

void main()
{
    int n;
    int tong=0;

    for(;;)
    {
        printf("\nNhap vao 1 so nguyen: ");
        scanf("%d", &n);
        if(n<0)
            continue;
        if(n==0)
            break;
        tong += n;
    }
    printf("\n=>Tong la: %d", tong);
}
```

Ví dụ 2: Viết chương trình cho người dùng nhập vào một sô và in số đó ra nếu là số âm ngược lại sẽ nhập lại

```
#include<stdio.h>

void main ()
```

```
{
     int n;
     Nhap: ;
     printf("Nhap va so n: ");
     scanf("%d",&n);
     if (n>0)
     goto Nhap;
     printf("ban vua nhap so %d: ",n);
}
```

4. Thực hành

4.1. Mục đích yêu cầu

- Vận dụng các câu lệnh rẽ nhánh, lệnh lặp để giải quyết các bài tập, viết chương trình.

4.2 Nội dung thực hành

Thực hiện các**: Bài tập trong Ví dụ**

Bài 1: giải phương tình bậc nhất ax+b=0

Bài 2: Nhập vào 1 tháng, năm, cho biết tháng đó có bao nhiêu ngày

Hướng dẫn:

// Thang co 31 ngay: 1, 3, 5, 7, 8, 10, 12

// Thang co 30 ngay: 4, 6, 9, 11

// Thang 2 co 28 hoac 29 ngay.

Bài 3: Giải phương trình bậc 2: $ax^2 + bx + c = 0$

Bài 4: Nhập vào độ dài 3 cạnh a, b, c của 1 tam giác.

a. Cho biết 3 cạnh đó có lập thành một tam giác không ?

b. Nếu có, tính chu vi của tam giác đó.

Bài 5: Viết chương trình tính tổng n số tự nhiên đầu tiên

S = 1 + 2 + 3 + ... + n

Bài 6: hãy tính tổng s =1 + 2 + 3 + ... + n

Bài 7: Viết chương trình nhập vào năm hiện tại, năm sinh. In ra tuoi.

Bài 8: Nhập vào 1 dãy số nguyên từ bàn phím đến khi gặp số 0 thì dừng. In ra tổng các số nguyên dương.

Bài 9: Viết chương trình nhập điểm thi từ bàn phím và hiển thị kết quả : kém nếu điểm 0, 1, 2 hoặc 3; Yếu nếu điểm là 4; Trung bình nếu điểm 5 hoặc 6; Khá nếu điểm 7 hoặc 8; Giỏi nếu điểm 9 hoặc 10.

BÀI 5: HÀM

Mục tiêu:

Học viên sau khi học xong có khả năng:

- Trình bày được khái niệm hàm;
- Trình bày được qui tắc xây dựng hàm và vận dụng khi thiết kế xây dựng chương trình;

1. Khái niệm hàm trong ngôn ngữ C

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình; người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một công việc nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn.

Trong C, chương trình con được gọi là hàm. Hàm trong C có thể trả về kết quả thông qua tên hàm hay có thể không trả về kết quả.

Hàm có hai loại: hàm chuẩn và hàm tự định nghĩa. Trong chương này, ta chú trọng đến cách định nghĩa hàm và cách sử dụng các hàm đó.

Một hàm khi được định nghĩa thì có thể sử dụng bất cứ đâu trong chương trình. Trong C, một chương trình bắt đầu thực thi bằng hàm main.

Ví dụ 1: Ta có hàm max để tìm số lớn giữa 2 số nguyên a, b như sau:

```
int max(int a, int b)
{
return (a>b) ? a:b;
}
```

Ví dụ 2: Ta có chương trình chính (hàm main) dùng để nhập vào 2 số nguyên a,b và in ra màn hình số lớn trong 2 số

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b)
{
return (a>b) ? a:b;
}

int main()
{
int a, b, c;
printf("\n Nhap vao 3 so a, b,c          ");
scanf("%d%d%d",&a,&b,&c);
printf("\n So lon la %d",max(a, max(b,c)));
getch();
return 0;
}
```

1.1. Hàm thư viện

Hàm thư viện là những hàm đã được định nghĩa sẵn trong một thư viện nào đó, muốn sử dụng các hàm thư viện thì phải khai báo thư viện trước khi sử dụng bằng lệnh
#include <tên thư viện.h>

Một số thư viện:

alloc.h assert.h bcd.h bios.h complex.h conio.h
ctype.h dir.h dirent.h dos.h errno.h
fcntl.h float.h fstream.h grneric.h
graphics.h io.h iomanip.h iostream.h limits.h locale.h
malloc.h math.h mem.h process.h setjmp.h
share.h signal.h stdarg.h stddef.h stdio.h
stdiostr.h stdlib.h stream.h string.h strstrea.h
sys\stat.h sys\timeb.h sys\types.h time.h values.h

Ý nghĩa của một số thư viện thường dùng:

1. stdio.h : Thư viện chứa các hàm vào/ ra chuẩn (**st**an**d**ard **i**nput/**o**utput). Gồm các hàm **printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…**

2. conio.h : Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm **clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…**

3. math.h: Thư viện chứa các hàm tính toán gồm các hàm **abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…**

4. alloc.h: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhơ. Gồm các hàm **calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …**

5. io.h: Thư viện chứa các hàm vào ra cấp thấp. Gồm các hàm **open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…**

6. graphics.h: Thư viện chứa các hàm liên quan đến đồ họa. Gồm **initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …**

...

Muốn sử dụng các hàm thư viện thì ta phải xem cú pháp của các hàm và sử dụng theo đúng cú pháp (xem trong phần trợ giúp của Turbo C).

1.2. Hàm người dùng

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình.

2. Xây dựng hàm

2.1 Định nghĩa hàm

Cấu trúc của một hàm tự thiết kế:

***<kiểu kết quả>* Tên hàm ([*<kiểu t số> <tham số>*][,*<kiểu t số><tham số>*][…])**

{

[Khai báo biến cục bộ và các câu lệnh thực hiện hàm]
[return [<Biểu thức>];]
}

Giải thích:

- *Kiểu kết quả:* là kiểu dữ liệu của kết quả trả về, có thể là : int, byte, char, float, void… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp *hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void*.

- *Kiểu t số*: là kiểu dữ liệu của tham số.

- *Tham số:* là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. ***Tham số này gọi là tham số hình thức***, ***khi gọi hàm chúng ta phải truyền cho nó các tham số thực tế.*** Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm (phần giới hạn bởi cặp dấu {}) là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

- Khi định nghĩa hàm, ta thường sử dụng câu lệnh return để trả về kết quả thông qua tên hàm.

2.2 Sử dụng hàm.

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi ta có một lời gọi đến hàm đó.

Cú pháp gọi hàm: <Tên hàm>([Danh sách các tham số])

Ví dụ: Viết chương trình cho phép tìm ước số chung lớn nhất của hai số tự nhiên.

```
#include<stdio.h>
unsigned int       ucln(unsigned  int a, unsigned  int b)
{
unsigned          int u;
if (a<b)
u=a;
       else
             u=b;
       while ((a%u !=0) || (b%u!=0))
u--;
return u;
}
int main()
{
unsigned int a, b, UC;
printf("Nhap a,b: ");scanf("%d%d",&a,&b);
UC = ucln(a,b);
printf("Uoc chung lon nhat la: ", UC);
return 0;
}
```

Lưu ý: Việc gọi hàm là một phép toán, không phải là một phát biểu.

2.3 Nguyên tắc hoạt động của hàm

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tương ứng*.
- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.

- Khi gặp lệnh return hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

3. Truyền tham số

Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị; nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức

Ví dụ 1: Giả sử ta muốn in ra nhiều dòng, mỗi dòng 50 ký tự nào đó. Để đơn giản ta viết một hàm, nhiệm vụ của hàm này là in ra trên một dòng 50 ký tự nào đó. Hàm này có tên là InKT.

```
#include <stdio.h>
#include <conio.h>
void InKT(char ch)
{
int i;
for(i=1;i<=50;i++) printf("%c",ch);
printf("\n");
}
int main()
{
char c = 'A';
InKT('*'); /* In ra 50 dau * */ InKT('+');
InKT(c);
return 0;
}
```

Lưu ý:

- Trong hàm InKT ở trên, biến ch gọi là tham số hình thức được truyền bằng giá trị (gọi là tham trị của hàm). Các tham trị của hàm coi như là một biến cục bộ trong hàm và chúng được sử dụng như là dữ liệu đầu vào của hàm.

- Khi chương trình con được gọi để thi hành, tham trị được cấp ô nhớ và nhận giá trị là bản sao giá trị của tham số thực. Do đó, mặc dù tham trị cũng là biến, nhưng việc thay đổi giá trị của chúng không có ý nghĩa gì đối với bên ngoài hàm, không ảnh hưởng đến chương trình chính, nghĩa là không làm ảnh hưởng đến tham số thực tương ứng.

Ví dụ 2: Ta xét chương trình sau đây:

```
#include <stdio.h>
```

```
#include <conio.h>
int    hoanvi(int a, int b)
{
int t;
t=a;  /*Đoạn này hoán vị giá trị của 2 biến a, b*/
a=b;
b=t;
printf("\Ben trong ham a=%d    , b=%d",a,b);
return 0;
}

int main()
{
int a, b;
clrscr();
printf("\n Nhap vao 2 so nguyen a, b:");
scanf("%d%d",&a,&b);
printf("\n Truoc khi goi ham hoan vi a=%d    ,b=%d",a,b);
hoanvi(a,b);
printf("\n Sau khi goi ham hoan vi a=%d    ,b=%d",a,b);
getch();
return 0;
}
```

Kết quả thực hiện chương trình:

```
Nhap vao 2 so nguyen a, b:6 5

Truoc khi goi ham hoan vi a=6  ,b=5
Ben trong ham a=5  , b=6
Sau khi goi ham hoan vi a=6  ,b=5
```

Giải thích:

- Nhập vào 2 số 6 và 5 (a=6, b=5)
- Trước khi gọi hàm hoán vị thì a=6, b=5
- Bên trong hàm hoán vị a=5, b=6
- Khi ra khỏi hàm hoán vị thì a=6, b=5

* Lưu ý

Trong đoạn chương trình trên, nếu ta muốn sau khi kết thúc chương trình con giá trị của a, b thay đổi thì ta phải đặt tham số hình thức là các con trỏ, còn tham số thực tế là địa chỉ của các biến.

Lúc này mọi sự thay đổi trên vùng nhớ được quản lý bởi con trỏ là các tham số hình thức của hàm thì sẽ ảnh hưởng đến *vùng nhớ đang được quản lý bởi tham số thực tế tương ứng* (cần để ý rằng vùng nhớ này chính là các biến ta cần thay đổi giá trị).

Người ta thường áp dụng cách này đối với các dữ liệu đầu ra của hàm.

Ví dụ: Xét chương trình sau đây:

```
#include <stdio.h>
#include <conio.h>
long   hoanvi(long *a, long *b)
/* Khai báo tham số hình thức *a, *b là các con trỏ kiểu long */
{
```

```
long    t;
t=*a;         /*gán nội dung của x cho t*/
*a=*b;        /*Gán nội dung của b cho a*/
*b=t;         /*Gán nội dung của t cho b*/
printf("\n Ben trong ham a=%ld    , b=%ld",*a,*b);
/*In ra nội dung của a, b*/
return 0;
}

int main()
{
long a, b;
clrscr();
printf("\n Nhap vao 2 so nguyen a, b:");
scanf("%ld%ld",&a,&b);
printf("\n Truoc khi goi ham hoan vi a=%ld    ,b=%ld",a,b);
hoanvi(&a,&b); /* Phải là địa chỉ của a và b */
printf("\n Sau khi goi ham hoan vi a=%ld    ,b=%ld",a,b);
getch();
return 0;
}
```

Kết quả thực hiện chương trình:

```
Nhap vao 2 so nguyen a, b: 5 6

Truoc khi goi ham hoan vi a=5  ,b=6
Ben trong ham a=6  , b=5
Sau khi goi ham hoan vi a=6  ,b=5
```

Giải thích:

- Nhập vào 2 số 5, 6 (a=
- Trước khi gọi hàm hoanvi thì a=5, b=6
- Trong hàm hoanvi (khi đã hoán vị) thì a=6, b=5
- Khi ra khỏi hàm hoán vị thì a=6, b=6

4. Các lệnh đơn nhằm kết thúc hàm và nhận giá trị trả về cho tên hàm.

Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

return ; **/*không trả về giá trị*/**

return <biểu thức>; **/*Trả về giá trị của biểu thức*/**

return (<biểu thức>); **/*Trả về giá trị của biểu thức*/**

Nếu hàm có kết quả trả về, ta bắt buộc phải sử dụng câu lệnh return để trả về kết quả cho hàm.

Ví dụ 1: Viết hàm tìm số lớn giữa 2 số nguyên a và b

```
int max(int a, int b)
{
return (a>b) ? a:b;
}
```

Ví dụ 2: Viết hàm tìm ước chung lớn nhất giữa 2 số nguyên a, b. Cách tìm: đầu tiên ta giả sử UCLN của hai số là số nhỏ nhất trong hai số đó. Nếu điều đó không đúng thì ta giảm đi một đơn vị và cứ giảm như vậy cho tới khi nào tìm thấy UCLN

```
int       ucln(int a, int b)
{
int u;
if (a<b)

       else
      u=a;
      u=b;

while ((a%u !=0) || (b%u!=0))
u--;
return u;
}
```

5. Một số bài tập áp dụng

Ví dụ 1: Chuyển số phút thành số **giờ: phút**

```
#include <stdio.h>
#include <conio.h>

// khai bao prototype void time(int & , int &);

// ham doi phut thanh
gio:phut
void time(int &ig, int &ip)
{
ig = ip / 60;
ip %= 60;
}

void main(void)
{
int igio, iphut;
printf("Nhap vao so phut : ");
scanf("%d", &iphut);
time(igio, iphut);
printf("%02d:%02d\n", igio, iphut);
getch();
}
```

Kết quả ra màn hình : `Nhap vao so phut: 185`

`03:05`

Ví dụ 2:

****

* Minh hoa ve ham *

****

Đây là kết quả của chương trình sau:

```
1   #include <stdio.h>
2   #include <conio.h>
3
4   // khai bao
5   prototype void
6   line();
7
8   // ham in 1
9   dong dau void
10  line()
11  {
12  int i;
13  for(i = 0; i <
14  19; i++)
15  printf("*");
16  printf("\n");
17  }
18
19  void main(void)
20  {
21  line();
22  printf("* Minh hoa ve ham *");
    line();
```

6. Thực hành

6.1 Mục đích yêu cầu

Mục đích của việc sử dụng hàm là làm cho chương trình viết ra được sáng sủa, ngắn gọn. Vì thế, học viên phải nắm vững cách định nghĩa các hàm và cách dùng chúng. Kết hợp các phần đã học trong các chương trước để viết các chương trình con.

6.2 Nội dung thực hành

Bài mẫu:

1. **Tạo một tập tin mới.**

2. **Nhập vào mã lệnh sau:**

```
#include<stdio.h>

void main()
{
      int a, b, c, sum;

      printf("\nEnter any three numbers: ");
      scanf("%d %d %d", &a, &b, &c);

      sum = calculatesum(a, b, c);
```

```
        printf("\nSum = %d", sum);
}

calculatesum(int x, int y, int z)
{
 int d;

 d = x + y + z;
 return (d);
}
```

3. **Lưu tập tin với tên hammau.C.**
4. **Biên dịch tập tin, hammau.C.**
5. **Thực thi chương trình, hammau.C.**
6. **Trở về trình soạn thảo.**

Bai 1: Viết chương trình có dùng hàm để giải phương trình bậc hai:
$ax^2 + bx + c = 0$ (a khác 0).
BàI 2: Viết chương trình có dùng hàm tìm số lớn nhất trong 3 số thực.
Bài 3: Viết chương trình có dùng hàm kiểm tra năm nhuận.
Bài 4: Nhập hai số a, b và xuất ra màn hình kết quả các phép tính cộng, trừ, nhân và chia của hai số.

Tóm tắt ngôn ngữ lập trình C

TỰA ĐỀ TRANG SÁCH

Cảm nghĩ: Tôi viết lên những dòng này không mục đích gì khác ngoài :

1. Ghi chép lại những gì mình biết.
2. Chia xẻ kiến thức với những người bạn thân của Tôi và mong được góp ý kiến những chỗ sai và thiếu sót để được bổ sung kiến thức .

Vì vậy những dòng chữ này không may ai có cầm phải nó xin đừng hỏi Tôi là ai, vì Tôi chỉ là kẻ vô danh với các bạn, và hãy xem đây là những dòng chữ nguyệch ngoạc mà thôi. (Tôi xin nói thật môn C này Tôi còn phải thi lại đấy !)

Xin chân thành cảm ơn !

Hà Nội, ngày 20 tháng 2 năm 2004

Tác giả

GIỚI THIỆU

Một cuốn C dày tới hơn 500 trang đó là cuốn sách hoàn hảo. Còn đây Tôi chỉ đưa ra được phương pháp đọc và hiểu được phần nào cuốn sách đó thôi. Tôi hi vọng cuốn sách C và thêm những lời giải thích này của Tôi, những ai chưa biết sẽ hiểu ra phần nào.

Tôi nhắc lại đây chỉ là những lời giải thích do Tôi hiểu khi đọc sách.

Chính vì vậy mà Tôi đặt cho nó cái tên là:

"Tóm tắt ngôn ngữ lập trình C"

PHẦN 1 : CẤU TRÚC MỘT CHƯƠNG TRÌNH C.

I. CÁC #INCLUDE<….> :

Đây là các thư viện của người lập trình đã được **Turbo C** định nghĩa sẵn cho người lập trình. Khi viết chương trình người lập trình cần khai báo nó.

Một số thư viện thường dùng là:

- Thư viện **#include<stdio.h>** : là thư viện Vào / Ra chuẩn.
- Thư viện **#include<conio.h>** : là thư viện
- Thư viện **#include<string.h>** : là thư viện cho phép sử dụng các xâu kí tự.
- Thư viện **#include<math.h>** : cho phép sử dụng các hàm toán học như sin,cos….
- Thư viện **#include<malloc.h>** : là thư viện cho phép khai báo động bộ nhớ.

Đây là một số thư viện thường dùng, ngoài ra còn một số thư viện khác nữa (*mong tự tìm hiểu*).

Chú ý :

Khi viết chương trình nhất thiết phải khai báo thư viện tương ứng, nếu không máy sẽ báo lỗi.

2. Hàm chính:

Đó là hàm **main()** tạm thời bạn hãy sử dụng nó còn phần giải thích xin được trình bày trong phần *"Xây dựng và sử dụng Hàm"*.

3.Thân chương trình:

Được bắt đầu bằng dấu khóa " { " ,tương đương với BEGIN trong ngôn ngữ Turbo Pascal và kết thúc trương trình bằng dấu " } " , tương đương với END.

Trong thân chương trình thường là các trình tự sau:

- Khai báo các biến,hằng…có sử dụng trong chương trình.
- Nhập giá trị cho các biến nếu cần.
- Xử lý các phép toán.
- Đưa ra kết quả sau xử lý.

Chú ý:

Đây chỉ là cấu trúc cơ bản của chương trình, cấu trúc sẽ khác và phức tạp hơn rất nhiều, nếu ta sử dụng cấu trúc tự định nghĩa.

II. KIỂU DỮ LIÊU VÀ VẤN ĐỀ KHAI BÁO BIẾN, HẰNG :

Kiểu dữ liệu có liên quan trặt trẽ với việc khai báo biến, hằng .Điều này là đương nhiên, bởi lẽ khi khai báo một biến nào đó bạn phải định kiểu dữ liệu tương ứng cho nó, còn việc hiểu biến, hằng là gì?, kiểu dữ liệu là gì ? Tôi xin trình bày một cách nôm na, chắc là không chặt trẽ, nhưng Tôi nghĩ nó rễ hiểu:

1.Biến :

Là giá trị đầu vào của chương trình, nếu nó chỉ là thành phần trung gian để tính toán (trong trường hợp này bạn phải nhập giá trị cho nó) . Ngược lại biến là giá trị đầu ra, nếu nó là nơi chứa kết quả sau khi tính toán.

Chú ý :

Biến có thể thay đổi giá trị trong chương trình.

2.Hằng :

Là giá trị đầu vào của chương trình, hằng không thể thay đổi được giá trị trong suốt thời gian tính toán.

3.Kiểu dữ liệu :

Đã được C định nghĩa sẵn, gồm có một số kiểu cơ bản hay sử dụng sau:

+ *Kiểu số nguyên :* Trong C nó được đặt tên là **int**, chiếm 2 byte trong bộ nhớ , có giá trị từ -32768 -> 32767 (2 byte). Tức là khi khai báo một biến nào đó, biến ấy không thể nhận giá trị vượt quá giới hạn này được.

VD: int a=2345 đúng.

int a=32768 sai do vượt quá giới hạn.

Tuy nhiên kết quả tính toán được của một biến nào đó cũng không được vượt quá giới hạn trên.

+ *Kiểu số thực:* Có tên là **float**, chiếm 4 byte trong bộ nhớ phạm vi biểu diễn từ -3.4E-38 đến 3.4E+38.

+ *Kiểu kí tự:* Có tên là **char**, chiếm 1 byte trong bộ nhớ.

Đây là 3 kiểu dữ liệu đơn giản nhất và hay sử dụng nhất, ngoài ra còn rất nhiều kiểu dữ liệu khác mục đích làm tăng phạm vi biểu diễn của 3 kiểu dữ liệu trên.

(nếu có nhu cầu xin tự tham khảo thêm).

III. LỆNH VÀO (NHẬP DỮ LIỆU), RA (XUẤT DỮ LIỆU) TRONG C:

1.Cách khai báo trong C:

- Nếu là số nguyên: **int a;** với khai báo này ta nhận được một biến a có kiểu là số nguyên.
- Nếu là số thực : **float b;** giải thích tương tự trên.
- Nếu là mảng: VD : **float a[10];** mảng tên là a kiểu thực và có 10 phần tử,
  int b[10]; mảng b kiểu nguyên có 10 phần tử.

Chú ý :

Khi nhập dữ liệu kiểu của biến tương ứng.

2.Nhập (vào dữ liệu), xuất dữ liệu trong C:

a. Nhập dữ liệu:

Cú pháp:

scanf("%.kiểu dữ liệu",&.biến);

VD : Giả sử có khai báo : float a;hoặc int b;

Câu lệnh nhập là : **scanf("%f ",&a);** hoặc **scanf("%d",&b);**

trong đó :
- % : Bắt buộc phải có.
- Kiểu dữ liệu : Có thể là số thực (kí hiệu là **f**).
  Có thể là số nguyên (kí hiệu là **d**).
  Có thể là kí tự (k/hiệu là **c**),
- & : Đại diện cho phép lấy địa chỉ.
- biến : Là 1 biến bất kì do bạn tự đặt.

b. Xuất dữ liệu:

Cú pháp :

printf(" %.đặc tả.kiểu dữ liệu",biến);

trong đó: - Đặc tả : Là kĩ thuật lấy phần nguyên và phần thập phân.

+ Nếu viết đặc tả = **4.2f** nghĩa là 4 số phần nguyên và 2 số phần thập phân (cách viết này chỉ dùng cho số thực).

+ Đặc tả = **4d** nghĩa là 4 chỗ cho chữ số đó. (chỉ sử dụng cho số nguyên).

VD: **printf("%4.2f",a);** **hoặc** **printf("%4d",b);**

IV.CHƯƠNG TRÌNH:

Trong 3 phần trên Tôi đã giới thiệu sơ qua những yếu tố cần thiết khi viết một chương trình. Trong phần này xẽ giới thiệu 1 cấu trúc chương trình hoàn hảo và một số câu lệnh cần thiết, bắt buộc trong chương trình. Có ví dụ minh họa:

a. Cấu trúc:

1. Khai báo các thư viện có liên quan.
2. Sau dấu { là phần khai báo các biến, hằng .. có sử dụng trong chương trình.
3. Nhập dữ liệu cho các biến cần thiết.
4. Xử lý theo yêu cầu của đề bài để có được kết quả.
5. Xuất dữ liệu ra màn hình.
6. Kết thúc trương trình bằng dấu }.

b.Các câu lệnh cần thiết:

1. Lệnh getch(); : Lệnh này yêu cầu dừng màn hình để xem kết quả.
2. Lệnh clrscr(); :Lệnh này dùng để xóa màn hình trước khi xuất dữ liệu.

c. Ví dụ:

Tôi xin đưa ra 1 ví dụ hết sức đơn giản là :

Cho hai số a, b là 2 số thực được nhập vào từ bàn phím . Yêu cầu tính tổng của hai số đó kết quả được bao nhiêu chứa vào biến c, sau đó in kết quả ra màn hình.

Tóm tắt : Dữ liệu đầu vào gồm có : 2 biến a và b kiểu số thực .

Đầu ra : In giá trị nhận được của biến c (c phải là số thực).

Viết chương trình:

```
#include<stdio.h>
#include<conio.h >
   main()   /*hàm chính bắt buộc phải có*/
   {
    clrscr();
    float a,b,c;   /*khai báo biến*/
    printf("nhập giá trị cho 2 biến:\n");scanf("%f",&a);
                                          scanf("%f",&b);    /*nhập */
    c=a+b;          /*tính toán*/
    printf("kết quả là c=%4.2f",c);          /* xuất dữ liệu*/
    getch();        /*dừng màn hình*/
   }
```

Bạn có thể tự lấy cho mình ví dụ đơn giản để làm quen với ngôn ngữ này.

Bài tập ví dụ:

Giải phương trình bậc nhất : ax + b = 0 ; với a,b là hai số thực nhập từ bàn phím,và in kết quả ra màn hình.

Tóm lai:

- Khi viết một chương trình bạn phải nắm được đề bài cho gì và yêu cầu làm gì. Để đơn giản hóa Tôi thiết nghĩ bạn nên giải bài toán một cách bình thường ra giấy. Sau đó bạn tìm cách đưa nó vào khuôn khổ của ngôn ngữ, với chú ý là.

Biến nhận kết quả (giá trị đầu ra) luôn nằm bên trái của phép toán.

Biến, hằng số tham gia thực hiện pháp toán (còn gọi là đầu vào) nằm bên phải biểu thức.

VD: Viết ax=b; /*viết sai*/

Viết x=b/a /*viết đúng*/

- Sau khi viết được chương trình này rồi, Tôi tin chắc rằng bạn xẽ đưa ra kết luận rằng Tôi phức tạp quá viết ra giấy còn nhanh và đơn giản gấp vạn lần. Bạn yên tâm cứ hãy cố gắng đi, còn rất nhiều điều thú vị đang chờ bạn khi đó Tôi tin chắc bạn xẽ có một cách nghĩ khác cho mà xem.

V. CÁC VÒNG LẶP XÁC ĐỊNH VÀ KHÔNG XÁC ĐỊNH:

1. Cách thức lưu trữ:

Trước hết Tôi muốn giải thích cách lưu trữ trong bộ nhớ của máy tính như sau:

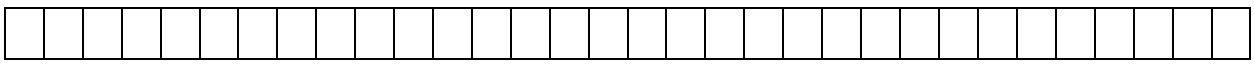
Khi mà bạn nhập dữ liệu vào máy tính đương nhiên máy tính phải cấp phát một khoảng nhớ để lưu trữ dữ liệu, và nó có địa chỉ xác định

a.Với số thực :

Cứ một giá trị bất kì nào được nhập vào đều chiếm 4 byte=32 bit số này được mã hóa bằng dãy nhị phân 0.1

VD : float a; => mô hình cấp phát là:

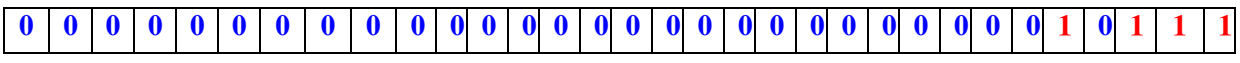
Bit cao nhất => (mỗi ô 1 bit) => bit thấp nhất



Với khai báo : **float a=23;** thì số 23 được mã hóa thành dãy nhị phân:

0000 0000 0000 0000 0000 0000 0001 0111 => mô hình lưu trữ là

bit cao nhất => => bit thấp nhất



b. Với số nguyên:

Máy sẽ cấp phát 2 byte=16 bit để lưu trữ .

Mô hình lưu trữ =>

Bit cao nhất => =>bit thấp nhất

VD: Với khai báo: **int a=23;** => **mô** hình cấp phát là:

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**1**

Tương tự như trên bạn có thể biểu diễn được với kiểu kí tự.

c. Nhận xét:

Thật đơn giản khi bạn nhập một số bất kì nào đó, thì lập tức máy xẽ tự mã hóa và cấp phát một vùng nhớ cho biến đó. Nhưng vấn đề không phải là nhập vào như thế nào, mà là địa chỉ của nó . Bạn phải biết được địa chỉ của nó thì bạn mới mong gọi được nó ra. Đây quả là một vấn đề hết sức phức tạp đối với những người bắt đầu làm quen với ngôn ngữ lập trình.Tôi xin đưa ra hai cách lấy địa chỉ như sau:

a. gọi theo tên biến : Là cách gọi bằng chính cái tên của biến do bạn đặt.

b. gọi theo địa chỉ : Là cách gọi theo chỗ nhớ của biến tương ứng. Cách này rất có hiệu quả trong việc lưu trữ biến nhiều phần tử, tuy nhiên bạn phải sử dụng 1 con trỏ để lưu trữ địa chỉ *(Tôi xẽ giới thiệu sau) VD* như biến mảng, biến cấu trúc).Trong Pascal đã giới thiệu rất kĩ về mảng tuy nhiên Tôi xin nhắc lại định nghĩa mảng.

Mảng là một số các phần tử phân bố kế tiếp nhau trong bộ nhớ.

VD : với khai báo **int a[20];** ⇔ Mảng a có 20 phần tử vì vậy

a[0] : Là phần tử đầu tiên trong mảng

a[1] : Là phần tử thứ 2 trong mảng.

……

a[19] : Là phần tử cuối cùng trong mảng.

2. Câu lệnh điều kiện:

Cú pháp : **If<điều kiện>**

```
{
   công việc;
}
```

```
else {
    công việc;
    }
```

VD:

Giải phương trình bậc nhất ax2+b=c (điều kiện a#0)với a,b,c được nhập từ bàn phím.

Viết chương trình

```
#include<stdio.h>
#include<conio.h>
  main()
  {
    float a,b,c,x;
    printf("nhập a,b,c=");scanf("%f%f%f",&a&b&c);
     x2=(c-b)/a;
   If((c-b)>0)
         {
         printf("phương trình có 2 nghiệm");
         printf("x1=%4.2f",sqrt(x));
         printf("x2=%4.2f",-sqrt(x));
         }
  if((c-b)=0) printf("co 1 nghiem x=0.");
  if((c-b)<0) printf("phuong trinh vo nghiem.");
     getch();
        }
```

3. Vòng lặp xác định:

Là vòng lặp cho bạn biết trước số lần lặp là bao nhiêu. Mục đích là để lưu trữ (nhập/xuất) các phần tử mảng cho thuận tiện

Cú pháp :

```
for(i=0;i<n;i++)
    {
    khối lệnh;
    }
```

trong đó: i : Là biến chạy cho biết địa chỉ của phần tử thứ i, i++ là sau 1 vòng lặp i sẽ tự động tăng thêm 1 đơn vị.

n : Là số phần tử của mảng.

VD:

Hãy nhập một mảng (10 phần tử) số nguyên bất kì tiên sau đó cho hiện ra màn hình dãy số vừa nhập.

Chương trình:

```
#include<stdio.h>
#include<conio.h>
    main()
    {
     int  a[10],i;
     for(i=0;i<10;i++)
       {
        printf("nhap gia tri cho mang:\n");
        printf("phan tu thu: %d \n",i);scanf("%d",&a[i]);
        }                    /*đến đây bạn đã có 1 mảng 10 phần tử*/
for(i=0;i<10;i++)
       {
        printf("hiện gia tri  mang vừa nhập:\n");
        printf("phần tử thứ %d %d",i,a[i]);
        }
      getch();
     }
```

Tương tự vậy bạn có thể làm ví dụ sau:

Bài Tập : Nhập mảng 10 số thực bất kì sau đó:

- Cho hiện lên màn hình dãy số vừa nhập.
- Tính tổng của 10 số đó và cho hiện tổng tính được lên màn hình .

Gợi ý: Công thức tính tổng s=s+a[i] ;

4. *Vòng lặp không xác định* :

Có 2 vòng cơ bản :

a. *Vòng lặp while….do :*

Cú pháp :

```
while (điều kiện)
    {
    khối lệnh;
    }
```

Giải thích : Vòng lặp trên có ý nghĩa là:

Trong khi (while) (điều kiện (còn đúng))

{

thì còn làm các công việc;

}

VD:

Tìm giá trị của n để có được giai thừa lớn hơn 120.

Chương trình:

```
#include<stdio.h>
#include<conio.h>
  main()
   {
   int n,gt;
   gt=1;
   n=0;
     while(gt<120)
       {
        n=n+1;
       gt=gt*n;
       }
     printf("gia tri n can tim la n= : %d",n);
     printf("giai thua tai n la:%d)",gt;
  getch();
  }
```

Nhận xét :

Khác với vòng lặp for bởi vì nó không biết nó phải lặp bao nhiêu lần. Nếu không có điều kiện(còn đúng) thì nó xẽ lặp mãi không dừng. Nhưng vấn đề không phải ở đó. Bạn nên hiểu rằng khi người ta đưa ra một vòng lặp thì chắc chắn nó sẽ có những công dụng khác nhau, giải quyết loại bài toán khác nhau.Tôi lấy ví dụ đơn giản bạn có thể dùng vòng for để nhập dữ liệu cho mảng, ngược lại trong 1 phép tính nào đó VD :y=x/5 điều kiện là hãy nhập vào giá trị của x để có được y<0.0004 và in giá trị của x tương ứng với loại bài toán này thì vòng for tỏ ra không có hiệu quả.

Tóm lại:

Để phân biệt được sự khác nhau giữa các vòng lặp là vô cùng khó khăn. Thực ra mà nói thì giữa chúng gần như không có danh giới, mà chỉ tỏ ra ưu việt hơn trong từng loại bài toán mà thôi. Vì vậy bạn cần nắm vững và phân biệt các loại bài toán để áp dụng nó một cách chính xác.

Bạn có thể xem các vòng lặp như là những công cụ làm việc, còn dữ liệu là nguyên liệu. Tôi lấy ví dụ như có bao giờ bạn dùng dao để trặt đá không trong khi đó bạn có búa, tuy nhiên bạn cố tình thì vẫn có thể làm được.

b. Vòng lặp do…while:

Cú pháp:
```
do
{
   công việc;
}
while <điều kiện>;
```

Vòng lặp này chỉ khác vòng lặp trên ở chỗ while được đặt ở cuối câu còn nội dung thì giống nhau.

Ghi chú:

Đối với 2 vòng lặp b, c bạn cần kết hợp thêm câu lệnh sau thì rất có hiệu quả. (tham khảo trong sách).

- **break ;** mục đích để thoát khỏi vòng lặp không cần đến điều kiện.
- **Continue;** Khi gặp lệnh này nó xẽ bỏ công việc đang thực hiện và quay lại thực hiện công việc tiếp theo.

Chú ý:

Bạn phải hết sức quan tâm đến phần này, vì nó là công cụ duy nhất để giải quyết bài toán. Nếu như không hiểu phần này coi như bạn chẳng biết gì cả. Cách tốt nhất bạn nên làm nhiều bài tập để hiểu cặn kẽ vấn đề.

VI. CON TRỎ VÀ HÀM:

1. Con trỏ:

Con trỏ rất có lợi trong mọi ngôn ngữ vì nó rất tiện lợi sử dụng trong việc *lưu trữ địa chỉ*. Có những việc nếu như không có con trỏ bạn không thể thực hiện được công việc như ý muốn. Ví dụ việc lấy kết quả từ một chương trình con sang chương trình chính, nếu không sử dụng con trỏ bạn không thể lấy kết quả ra khỏi chương trình đó

được. Tuy nhiên việc sử dụng con trỏ làm cho chúng ta cảm thấy rất phức tạp.Thực ra nó cũng đơn giản thôi bạn thấy gì sự khác nhau giữa hai chương trình sau:

Chương trình 1:

<không có con trỏ>

```
#include<stdio.h>
#include<conio.h>
  main()
   {
    int a=34;
    printf("gia tri của a la a= : %d",a);
    getch();
   }
```

kết quả in a=34;

Chương trình 2:

< có con trỏ>

```
#include<stdio.h>
#include<conio.h>
 main()
 {
    int a=34,*p;
    *p=a;
    printf("gia tri của a la a= : %d",p);
    getch();
  }
```

kết quả in a=844;

Vì sao vậy bởi vì con trỏ chỉ lưu địa chỉ của biến a chứ nó không lưu giá trị của biến a. Mà địa chỉ trong bộ nhớ thì đương nhiên không phải là giá trị của biến nào đó. Đến đây Tôi tin chắc bạn đã hình dung ra rồi trứ, nếu chưa Tôi xin được lấy một ví dụ vui thế này:Bạn của bạn nhờ bạn cất giữ hộ một cái va li đương nhiên bạn không thể mở nó ra được, chiếc va li đó chẳng may bị mất, khi trình báo với công an thì bạn chỉ có thể trình bày là chiếc va li để chỗ nào thôi chứ không biết được trong đó có gì, còn bạn của bạn lại biết bên trong có gì nhưng không biết nó nằm chỗ nào.

Tức là gì:

" Con trỏ chỉ lưu trữ địa chỉ chứ không lưu trữ giá trị của biến đó,đương nhiên giá trị phải nằm trong địa chỉ "

Hi vọng bạn không còn nhầm lẫn nữa. Và sau đây Tôi xẽ cho các bạn thấy sự tác dụng của con trỏ và việc sử dụng nó. Tôi xẽ nói tiếp về con trỏ sau, vì nó gắn liền với việc sử dụng hàm.

2.Hàm và chương trình:

Bạn có phân biệt được hai từ này không còn theo Tôi thì:

+ **HÀM** là 1 phần chương trình lớn, gồm có các hàm sau:

2.1. Hàm void() : Là hàm không chả lại kết quả trong tên hàm khi chương trình chính gọi đến nó.

VD: Có hàm **void tim(int n);** Khi đó trong chương trình chính đưa ra lời gọi hàm là **printf("kết quả tìm là%d",tim(n));** thì chắc trắn bạn không có được kết quả đúng.

2.2. Hàm main() : Hàm này bắt buộc cho mọi chương trình(còn gọi là hàm chính).

2.3. Hàm có kiểu : Trong đó có thể là các kiểu nguyên, thực,char.

VD: int gt(int n); hoặc hàm float tbc(float a,float b);

Loại hàm này sau khi thực hiện bạn đưa ra câu lệnh sau

return(biến tổng); thì tự khắc kết quả được tự động chứa vào tên hàm là "tbc" , khi đó ở chương trình chính bạn gọi tới nó thì nó sẽ cho kết quả mà nó nhận được.

2.4. Hàm tự định nghĩa:

(Tôi xin giới thiệu hàm này trong phần dữ liệu có cấu trúc vì phần này chưa cần sử dụng đến).

+ **CHƯƠNG TRÌNH** :

Là một dãy các hàm kết hợp lại với nhau trong đó có hàm chính được định nghĩa là hàm **main().**

3. *Sử dụng hàm*:

Từ đầu đến giờ cứ mỗi một ví dụ đều phải sử dụng hàm đó là hàm main(). Nhưng trong phần này Tôi muốn trình bày cách kết hợp nhiều hàm trong 1 chương trình, kết hợp với việc sử dụng con trỏ như thế nào.

a.Kết hợp hàm main và hàm có kiểu:

Tôi sẽ xây dựng 1 chương trình có 2 hàm trong đó hàm main sẽ gọi đến hàm void

VD (bt 4,trang 133,chương 5,sách của ông ẤT):

Viết chương trình tính

sinx=$x-x^3/3!+x^5/5!-$ …..+$(-1)^n x^{2n+1}/(2*n+1)!$ Với độ chính xác là 0.0001.

Gợi ý:

Ta thấy x luôn tỷ lệ với n, tức n tăng thì mũ của x cũng tăng theo, khi n tăng thêm 1 đơn vị thì ta lại có được 1 thương mới, nếu n lẻ thì thương này mang dấu trừ và ngược lại. Độ chính xác 0.0001 tức là khi có một thương mới, nếu thương này <0.0001 tức là điều kiện được thỏa mãn

Đầu vào : Nhập x bất kì.

n: Là biến chạy nhằm thỏa mãn độ chính xác.

Đầu ra : Giá trị của sinx ;

Tôi sử dụng 2 hàm:

- Hàm int gt(int n); để tính giai thừa n kiểu hàm là nguyên tên hàm là gt ,đối n kiểu nguyên.
- Hàm main() là hàm chính làm nhiệm vụ nhập x và gọi tới hàm gt để lấy kết quả tính toàn và cuối cùng là in giá trị của sinx

Để viết được chương trình này bạn phải hình dung được là:

- Mẫu số là biểu thức tính giai thừa. Vì vậy để đơn giản bài toán ta xây dựng 1 hàm tính giai thừa là mẫu số, khi n tăng tự nó xẽ đưa ra lời gọi giai thừa.
- Tử số tuy nó phức tạp nhưng rất may trong c đã có hàm pow(y,x) ⇔ y^x để tính x^n và $(-1)^n$ tức là $(-1)^n$<=> pow(-1,n);và x^n<=>pow(x,n) ;
- Chương trình sử dụng 1 biến kt để chứa giá trị thương nhận được, đồng thời nó cũng là điều kiện để vòng lặp kết thúc. Nếu kt>0.0001.biến sinx để chứa kết quả nhận được.

Sau đây là chương trình:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 int gt(int n);
```

```
/*bắt đầu hàm gt*/
 int gt(int n)
  {
     int i,tg;
     tg=1;
     for(i=1;i<=2*n+1; )
  {
     i=i+2;
     tg=tg*i;
  }
 return tg;     /*là câu lệnh chứa kết quả của tg vào tên hàm chú ý nếu là con trỏ thì phảI là return (&tg)*/
  }
/*bắt đầu hàm chính*/
   main()
  {
    clrscr();
    int n;
    float x,sinx,dk;
    printf("\n nhap gia tri cua x= ");scanf("%f",&x);
       n=1;
        do
              {
           dk=(pow(-1,n)* pow(x,n)) /gt(n); /*đây là lời gọi tới hàm gt*/
           sinx=sinx+dk;
           n=n+1;
             }
       while (abs(dk)<0.0001); /*abs là lấy giá trị tuyệt đối*/
  printf("\n gia tri cua sinx =%4.8f",sinx);
  printf("\n n=%d",n);
  getch();
     }
```

Nếu cũng với bài toán này Tôi khai báo là: int *gt(int n);tức là sử dụng con trỏ thì điều gì xẽ xảy ra: Bạn hãy tự chạy chương trình này thì bạn xẽ hiểu những gì Tôi nói trong phần con trỏ.

Sau đây là chương trình:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
    int *gt(int n);
        /*bắt đầu hàm gt*/
    int *gt(int n)
    {
      int i,tg;
      tg=1;
      for(i=1;i<=2*n+1; )
    {
      i=i+2;
      tg=tg*i;
    }
       return (&tg);
    }
/*bắt đầu hàm chính*/
       main()
      {
       clrscr();
       int n;
       float x,sinx,dk;
       printf("\n nhap gia tri cua x= ");scanf("%f",&x);
        n=1;
        do
        {
            dk=(pow(-1,n)* pow(x,n)) /gt(n);/*đây là lời gọi tới hàm gt*/
            sinx=sinx+dk;
            n=n+1;
        }
```

```
while (abs(dk)<0.0001);  /*abs là lấy giá trị tuyệt đối*/
printf("\n gia tri cua sinx =%4.8f",sinx);
printf("\n n=%d",n);
getch();
}
```

Chú ý:

Bạn có biết tại sao trong hàm gt Tôi lại sử dụng biến tg, mà lại không sử dụng trực tiếp biến gt không. Bởi vì không nên sử dụng lời gọi hàm lớn hơn 1 lần trong chương trình.

Tương tự vậy bạn hãy làm bài tập sau:

Tính giá trị của $e^x=1+x/1!+x^2/2!+....+x^n/n!$;

b. Hàm main và hàm void:

- Như ta đã biết hàm void không trả lại kết quả cho tên hàm vì vậy muốn lấy kết quả ra khỏi chương trình thì bắt buộc bạn phải dùng đến con trỏ.
- Tôi lại xin nhắc lại cho nhớ hai câu mà bạn không được pháp quên đó là:

1. Con trỏ chỉ lưu địa chỉ của biến chứ không lưu giá trị.

2. Khi nào cần lấy giá trị của biến ra khỏi chương trình thì bắt buộc phải sử dụng con trỏ.

Vì sao Tôi nhắc lại điều này vì từ bây giờ trở đi con trỏ là công cụ đắc lực khi giải quyết bài toán. Việc sử dụng hàm bạn đã quá quen thuộc vì vậy sau đây Tôi muốn bạn hiểu thông qua ví dụ sau:

VD: Viết chương trình hoán vị hai số a,b cho nhau sủ sụng 2 hàm:

- Hàm void vh(float *p1,float *p2) để hoán vị 2 số cho nhau.
- Hàm main() để nhập dữ liệu và in kết quả hoán vị.

Chương trình:

```
#include<stdio.h>
#include<conio.h>
 void hv(float *p1,float  *p2);        /*khai báo nguyên mẫu*/
 void hv(float *p1, float *p2);       /*bắt đầu hàm void*/
 {
     float tg;
     tg=*p1; *p1=*p2; *p2=tg;
 }
```

```
main()
{
  float a,b;
  printf("\n nhap gia tri cua a= ");scanf("%f",&a);
  printf("\n nhap gia tri cua b= ");scanf("%f",&b);
  vh(&a,&b);
  printf("\n nhap gia tri cua a= %4.2f",a);
  printf("\n nhap gia tri cua b= %4.2f",b);
  getch();
  }
```

nhập a=2.4;b=3.0;

kết quả chạy: a=3.00;b=2.40;

Để tỏ rõ uy lực của con trỏ bạn thử bỏ * ở p1 và p2 mà xem.

4. *Các lỗi thường gặp trong khi sử dung con trỏ:*

Đưa ra phần này Tôi, bản thân cảm thấy rất xấu hổ bởi vì vốn tiếng anh của Tôi vô cùng khiêm tốn. Có gì mong bỏ quá cho.

+ Máy báo : inconpatible type conversion : lỗi này là do lời gọi hàm sai lẽ ra là tên hàm (&biến) bạn chỉ viết tênhàm(biến)

+ Máy báo : illegal use of floating point :bạn đã viết thiếu dấu "*" trong 1 biểu thức nào đó vì đây là con trỏ.

+ Máy báo : cannot covert int to int :thiếu dấu & trong return(tênhàm)

+ Máy báo : too few parameters in callto :đối trong lời gọi hàm không tương thích với trong hàm.

+ Máy báo : linler error undefined symbol ….:sai đường dẫn giữa lời gọi hàm và hàm.

Ngoài ra còn rất nhiều lỗi khác nữa mong tự tìm hiểu.Đặc biệt có 1 lỗi mà chương trình không báo lỗi, nhưng chương trình cho kết quả sai, thì đó là do thuật toán của bạn sai hoặc là bạn sử dụng con trỏ không đúng, tức là bàn vẫn chưa hiểu gì về con trỏ .

VII. MẢNG VÀ CON TRỎ :

Mảng là một số các phần tử có cùng kiểu dữ liệu được lưu trữ kế tiếp nhau trong bộ nhớ.

Việc truy nhập vào mảng bắt buộc phải thông qua chỉ số của mảng.

Chú ý:

Trong C quy định chỉ số của 1 mảng bắt đầu từ 0. Và bạn đừng nghĩ mảng và biến là một (hi vọng Tôi nhắc điều này là không thừa).Nếu bạn lấy chỉ số là 1 thì khi truy nhập mảng xẽ truy nhập ra ngoài vùng nhớ của mảng=>kết quả sai.

VD : for(i=1;i<n;i++) là sai tuy nhiên máy không báo lỗi.

Còn for(i=0;i<n;i++) là đúng;

Trong ngôn ngữ C có 2 loại mảng đó là mảng tĩnh và mảng động :

***1. Mảng tĩnh*:**

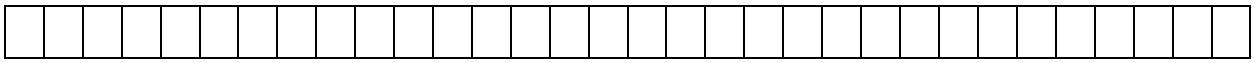
Là mảng cho ta biết trước số phần tử là bao nhiêu.

VD: int a[10]; với khai báo này bạn đã có một mảng số nguyên có10 phần tử do chính bạn sẽ nhập vào trong chương trình bằng câu lệnh:

```
for(i=0;i<10;i++)
    {
        printf("nhập mảng a[%d]=",i);scanf("%d",&a[i]);
    }
```

Khi đó máy sẽ lưu trong bộ nhớ như sau:(giả thiết đây là 1 khoảng trong bộ nhớ).

a[0] chiếm 8bit(=8 ô)=>……..vùng nhó liên tiếp……. =>a[9] chiếm 8bit.



Công việc về mảng thường có một số công việc sau:

- Nhập mảng.
- Sắp xếp mảng.
- Tìm phần tử mảng.
- Bổ sung phần tử cho mảng.
- Loại phần tử mảng.
- In mảng.

VD:

Lập một chương trình sử dụng 2 hàm . Hàm void làm nhiệm vụ sắp xếp mảng. Hàm main làm nhiệm vụ nhập mảng và hiển thị mảng.

***Nhận xét*:**

Trong hàm void làm nhiệm vụ sắp xếp. Sau khi sắp xếp nó xẽ trả kết quả lại cho hàm main.Vì vậy để lấy được kết quả này bắt buộc phải sử dụng con trỏ:

Chương trình:

```
#include<stdio.h>
#include<conio.h>
```

```
#include<alloc.h>
 void  sxep(float *p);
 float a[5];
 void  sxep(float *p)
  {
     int i,j;
     float tg;
     for(i=0;i<5;i++)
     for(j=i+1;j<=5;j++)
     if(p[i]>p[j])
      {
        tg=p[i];
        p[i]=p[j];
        p[j]=tg;
      }
   }
main()
  {
      int i;
      clrscr();
      for(i=0;i<=5;i++)
        {
           printf("\n nhap phan tu thu: %d \n",i);scanf("%f",&a[i]);
        }
      else*/ printf("\n ket qua sap xep:\n");
      sxep(a);
      for(i=0;i<5;i++)
      printf("\n a[i]=%4.2f",a[i]);
 getch();
  }
```

Sau khi chạy chương trình này bạn :

1.Thử bỏ * trong p xem điều gì xẽ sảy ra.

2.Trong lời gọi hàm bạn viết sxep(&a[i])

3.Thử viết for(i=0;I<=5;i++) hoặc for(i=1;i<5;i++)

4. Thay lời gọi trên bằng for(i=0;i<5;i++)

sxep(&a[i])đối chiếu với lời gọi trên và cho nhận xét. (Đây là hai cách gọi hàm đều đúng mà Tôi phải tìm hiểu mãi mới nhận ra được quy luật của nó bạn phản thử mới nhớ được).

Chú ý:

Bạn phải thử mới biết cái thú vị của nó.Nếu bạn không thử trong những hướng dẫn này, đến 1 lúc nào đó bạn làm chương trình nó không báo lỗi, mà lại cho kết quả sai, khi đó bạn xẽ tức điên lên cho mà xem.

2. Khai báo động: (mảng động)

Cú pháp: **a=(kiểu dữ liệu*)malloc(n*sizeof(a));**

Trong đó: a : Là mảng phần tử(a phảI là con trỏ mảng).

n : Là chỉ số lớn nhất của mảng.

+ Kiểu dữ liệu: Là kiểu dữ liệu của mảng.

Thực ra phần này tương tự với phần trên nó chỉ khác ở hai điểm sau:

- Tên mảng phải là con trỏ, số lượng phần tử (n) phải nhập vào.
- Không biết trước số lượng phần tử. Đây là lý do của việc phải sử dụng con trỏ,và vì n nhập từ bàn phím chứ không ấn định cho nên tiết kiệm được bộ nhớ.
- Trước khi sử dụng khai báo động phải khai báo thư viện là:

#include<stdio.h>

Do vậy Tôi mong các bạn hiểu phần này thông qua ví dụ sau:

VD:

Viết CT nhập n phần tử từ bàn phím. CT sử dụng hai hàm:

1.Hàm float tbc(float p[],int n);để tính trung bình cộng của mảng

2.Hàm main() để nhập dữ liệu và in kết quả.

Chương trình:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
float  tbc(float p[],int n);
float tbc(float p[],int n)
{
int i,d=0;
float s;
```

```
 for(i=0;i<n;i++)
   {
   s=s+p[i];
   d=d+1;
   }
return(s/d);
}
main()
{
int i,n;
float *a;
clrscr();
printf("\n nhap so phan tu cua mang n=:");scanf("%d",&n);
a=(float*)malloc(n*sizeof(float));
for(i=0;i<n;i++)
  {
   printf("\n nhap phan tu thu: %d \n",i);scanf("%f",&a[i]);
  }
tbc(a,n);
 printf("\n trung binh cong  la %8.2f",tbc(a,n));

getch();
}
```

Nhận xét:

Bạn có thấy điều gì khác biệt trong phần này không . Trong phần con trỏ Tôi nói: Con trỏ chỉ lưu giữ địa chỉ chứ không lưu kết quả. Còn trong phần này thì nó lại lưu kết quả, lúc in ra vẫn hoàn toàn đúng: Tôi xin lấy tiếp ví dụ vui trong phần con trỏ như sau:Lần trước là chiếc va li của người bạn nhờ bạn cất giữ hộ. Còn bây giờ chiếc va li ấy là của bạn thì đương nhiên bạn phải biết trong đó có gì, và nó được cất ở đâu. Có nghĩa là: Khi bạn nhập giá trị cho 1 biến a nào đó, bạn gán *(p)=a (tức là cất hộ),còn bạn nhập trực tiếp *p[i]=?(như ví dụ trên đã chứng minh) Thì thử hỏi hai điều này có khác nhau không.

PHẦN 2. CẤU TRÚC

I. CÁCH LƯU TRỮ CỦA MÁY TÍNH:

Phần này Tôi cứ loay hoay từ đầu CT đến giờ xem giới thiệu vào chỗ nào là hợp lý nhất, bạn biết vì sao không: Vì nó cực kì quan trọng, vì Tôi sợ bạn quên mất. Nếu bạn quên thì bạn mất đi cơ hội biết được máy tính làm gì, và làm như thế nào, khi bạn tác động vào nó. Nếu không hiểu đương nhiên những việc bạn làm chỉ là mò mẫm mà thôi.

Bạn chú ý câu nói này**: Trong ngôn ngữ C quy định lưu trữ trong máy tính bằng 2 dạng đó là: Lưu trữ kế tiếp và lưu trữ móc nối đối với dữ liệu có cấu trúc.**

Mô hình lưu trữ **:**

1.*Dạng kế tiếp*:

Ví dụ có khai báo trong phần II(xem ở dưới). Đầu chương trình bạn: Viết khai báo 1 biến cấu trúc tên là danh sách và 1 mảng cấu trúc tên là ds[10] .

a. *Với biến cấu trúc tên là danh sách*:

=>giả thiết các ô nhớ có thứ tự từ trái qua phải và từ trên xuống.

ô 1	2	3	..												
												..	..	150	ô 160

- 8 ô cho 1 kí tự=>mảng char họtên[20] cần 160 kí tự => lưu trữ xong trường họtên[20].
- Tương tự tiếp theo cần 160 kí tự cho trường quêquán[20]=>bắt đầu từ ô 161=>hết ô 320.

ô 161	162	..	..												
												..	..	319	320

- Tiếp theo đến trường ngày sinh có kiểu là int => cần 2 byte để lưu trữ=>bắt đầu từ ô 321=>hết ô thứ 336.

321															ô 336

⇨ Hết 1 biến cấu trúc.

b. ***Với mảng cấu trúc***.

Tương tự trên bạn có thể suy ra. Một biến cấu trúc cần 336 ô(=336 bit) để lưu trữ, thì mảng ds[10] cần 10*336 ô = 3360 ô để lưu trữ, các ô này kế tiếp nhau.

c.*Kết luận*:

Bạn phải phân biệt rõ đâu là biến cấu trúc, đâu là mảng cấu trúc. Vì điều này còn liên quan tới việc sử dụng con trỏ.

2. *Lưu trữ móc nối* :

Khác với kế tiếp. Lưu trữ móc nối thì các phần tử nằm không kế tiếp nhau trong bộ nhớ, nhưng chúng được liên kết với nhau thông qua con trỏ giữ địa chỉ của phần tử tương ứng:Tức là mỗi một phần tử có 2 trường một trường để chứa dữ liệu, còn 1 trường để chứa con trỏ liên kết với phần tử kề sau nó.

Mô hình chung:

(giả thiết mỗi ô là 1 phần tử cấu trúc chứ không phải là 1 bit như trên.).

+ Với 1 phần tử có cấu trúc:=>

Phần chứa dữ liệu	Link=null

Phần chứa dữ liệu : Bao gồm các trường kế tiếp nhau.

Phần link : Là con trỏ kết nối phần tử liền sau nó. Vì có 1 phần tử nên nó không giữ địa chỉ của phần tử nào nữa=> chỉ vào null=0(rỗng);

Với lớn hơn 1 phần tử có cấu trúc => (giả thiết đây là các vùng nhớ liên tục trong bộ nhớ.)

Bắt đầu vùng nhớ =>

Phần tử 1	Link 1	Vùng nhớ không liên quan	Vùng nhớ không liên quan	p.tử 2	Link 2
Không liên quan	.....	.....	Phần tử 3	Link 3	K.L.Q
Phần tử 4	Link 4	KLQ	Phần tử 5	Link 5	KLQ

=>kết thúc vùng nhớ.

Trong vùng nhớ trên có 5 phần tử có cấu trúc xen kẽ với các vùng nhớ khác, nhưng chúng được liên kết nhau bởi phần link: Link1 chứa phần tử 2,link 2 chứa phần tử 3..... cuối cùng là link 5, nó không chứa gì cả và nó chỉ vào null.

Chú ý:

Các trường của 1 phần tử cũng được cấp phát liên tiếp nhau trong bộ nhớ.Tức là phải phân biệt rõ các trường và các phần tử.

II. ĐỊNH NGHĨA VÀ KHAI BÁO MỘT CẤU TRÚC THEO KIỂU LƯU TRỮ KẾ TIẾP .

A. Cấu trúc không lồng nhau :

1. *Định nghĩa* :

Cấu trúc lầ tập hợp các biến, các mảng được biểu diễn thông qua một tên.

2. *Khai báo theo kiểu lưu trữ kế tiếp*:

Có rất nhiều cách khai báo khác nhau cho một cấu trúc. Chính vì vậy mà chúng ta khi bắt đầu tiếp xúc với phần này thường rơi vào hai trường hợp : Một là khó nhớ, hai là khai báo lộn sộn không theo một cấu trúc nào cả. Để khắc phục tình trạng này Tôi khuyên bạn nên làm quen với một cách khai báo duy nhất sau, và đây cũng là cách khai báo có nhiều tính năng nhất: Đây là cách khai báo theo kiểu tự định nghĩa:

Cú pháp:

```
typedef  struct
         {
         kiểu trường   Tên trường 1;
         kiểu trường   Tên trường 2
           ..............................
         kiểu trường    Tên trường n;
        }
         Tên cấu trúc;
```

trong đó:

- Trường 1,2...n: Là 1 cái tên do bạn đặt sao cho ý nghĩa gắn liền với kiểu trường.
- Kiểu trường thường là các kiểu int, float, char.
- Tên cấu trúc: là cái tên cũng do bạn tự đặt sao cho có ý nghĩa. Tên này sẽ sử dụng để làm kiểu dữ liệu trong CT <=>với 3 kiểu dữ liệu trên. Chú ý tên trường thường không có kí tự trống

VD: họ tên:là sai.
Họtên:là đúng.

VD: khai báo một cấu trúc gồm 3 trường họ tên, ngày sinh và quê quán:

```
Typedef struct
    {
    Char   họtên[20];   /*mảmg kí tự có 20 kí tự*/
    Char   quequan[20];   /*tương tự trên*/
```

```
    int     ngàysinh;
} sinhviên;
```

Với khai báo trên bạn có một cấu trúc tên là sinh viên có 3 trường là họ tên, quê quán và ngày sinh.

3. *Cách truy nhập tới cấu trúc lưu trữ kế tiếp* :

Khi bạn có khai báo trên thì trong chương trình bạn có thể dùng nó như một kiểu dữ liệu thông dụng

VD:

Đầu chương trình sau dấu { bạn viết: **danhsách:sinhviên, sv[10];** Với khai báo này bạn có 1 biến cấu trúc tên là danhsách có 10 phần tử và có kiểu là sinhviên. Mảng cấu trúc có kiểu là danh sách.

Bạn nên nhớ là trong C phân biệt chữ hoa và thường, cho nên khi đặt tên bạn phải thống nhất. Tốt nhất bạn nên viết thường cả, và thêm 1 chú ý nữa là đặt tên nên ngắn gọn, dễ nhớ VD:ht <=>họtên.qq<=>quêquán...

Cách truy nhập:

Có một cách truy nhập duy nhất đó là:

Tên biến cấu trúc.tên trường; (ngăn cách giữa tên biến và trường là dấu (.)).

Còn một kiểu cấu trúc lồng nữa Tôi xẽ giới thiệu sau, vì Tôi muốn bạn làm quen dần dần, để chánh sự choáng ngợp kiến thức.

4. Ví dụ :

Phần I Tôi đã cho bạn công cụ làm việc đó là các vòng lặp. Và phần II này bạn đã có nguyên liệu(là dữ liệu đầu vào). Bây giờ là lúc bạn có thể chế biến nó thành sản phẩm như ý muốn. Tôi xin làm mọt vài ví dụ mẫu, từ đó bạn có thể phát triển thành những gì mà bạn muốn.

Chú ý: Khi nhập dữ liệu kiểu char, bạn nên dùng câu lệnh gets(); để nhập , trướckhi nhập bạn phải xóa bộ đệm bàn phím bằng câu lệnh: fflush(stdin);

Còn khi in thì vẫn dùng printf như bình thường.

VD1:

Hãy nhập vào một danh sách cán bộ có 10 sinh viên mỗi cán bộ gồm 4 trường : họ tên, công, bậc lương, và lương(lương = bậc lương*công) .Sau khi nhập xong thì cuối cùng cho hiện danh sách đó ra màn hình. Với 1 chú ý là bạn phải dùng 1 biến trung gian để nhập giá trị cho các biến có kiểu là số thực nếu không khi nhập máy sẽ bị treo.

Chương trình :

```
#include<conio.h>
```

```
#include<string.h>
typedef struct
      {
      char  ht[20];
      int  cong;
      float luong;
      float bluong;
      }cb;
main()
{
clrscr();
cb a[10];
float l,bl;
int c, i;
for(i=0;i<3;i++)
   {
   printf(" nhap ho ten nguoi thu: %d \n",i);
   fflush(stdin);
    gets(a[i].ht);
    printf(" nhap ngay cong:\n"); scanf("%d",&c); a[i].cong=c;
   printf(" nhap bac luong:\n");scanf("%f",&bl); a[i].bluong=bl;
   a[i].luong=a[i].cong*a[i].bluong;
      }
printf("\n **************KET QUA******************");
for(i=0;i<3;i++)
    {
    printf("\n ho ten %d  nguoi thu: %s",i, a[i].ht );
    printf("\n ngay cong: %4d",a[i].cong);
    printf("\n bac luong:%4.2f",a[i].bluong);
    printf("\n luong:% 4.2f",a[i].luong);
    }
getch();
}
```

Có lẽ chương trình này không cần giải thích gì hơn, vì nó cũng khá đơn giản. Nhưng một điều chú ý cho là: Khi bạn nhập giá trị cho biến thực bạn không được nhập trực tiếp, mà phải nhập thông qua 1 biến trung gian vì nếu nhập trực tiếp như vậy máy xẽ bị treo.

VD2:

Ví dụ có kết hợp với việc sử dụng hàm và con trỏ:

Hãy nhập vào từ bàn phím số liệu cho n sinh viên. Cấu trúc mỗi sinh viên gồm 3 trường là: họ tên, quê quán và tuổi. Yêu cầu sử dụng 3 hàm

Hàm void sxếp(sinhviên *p; int n); để sắp xếp tăng dần theo thứ tự tăng dần của tuổi.Vì hàm này sau khi làm nhiệm vụ sắp xếp nó phải trả lại kết quả sắp xếp cho hàm main(), cho nên đối phải là con trỏ.

Hàm void in(sinhviên p;int n);để in kết quả sau khi sắp xếp.

Hàm main() là hàm chính để nhập dữ liệu và đưa ra lời gọi đến các hàm khác khi cần thiết.

Chương trình:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
typedef struct
      {
      char ht[40];
      char qq[40] ;
      int t;
      }sinhvien;
void  sxep(sinhvien *p,int n);
void in(sinhvien p);
void  sxep(sinhvien *p,int n)
{
int i,j;
sinhvien tg;
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
         if(p[i].t>p[j].t)
         {
```

```
            tg=p[i];
            p[i]=p[j];
            p[j]=tg;
            }
}
void in(sinhvien p)
      {
      printf("\n ten nguoi: 5%s",p.ht);
      printf("\n que quan %s: ",p.qq);
      printf("\n tuoi:%d",p.t);
      }
main()
{
clrscr();
sinhvien *p,ds[20];
int i,j,n;
printf("nhap so sinh vien n:="); scanf("%d",&n);
for(i=0;i<n;i++)
      {
      fflush(stdin);
      printf("\n nhap ten nguoi: %d",i);gets(ds[i].ht);
      printf("\n nhap que quan : ");    gets(ds[i].qq);
      printf("\n nhap tuoi:");          scanf("%d",&ds[i].t);
      }
sxep(ds,n);   /* lời gọi hàm*/
for(i=0;i<n;i++)
in(ds[i]);   /* lời gọi hàm*/
getch();
}
```

Giải thích quá trình làm việc: Sau khi soát lỗi xong, máy vào hàm chính bắt nhập vào n và thực hiện việc nhập dữ liệu vào biến cấu trúc có tên là danh sách. Nhập xong nó thấy có lời gọi hàm sxếp, biến tương ứng là ds và nó đem biến này gán vào đối tương ứng là con trỏ *p. Con trỏ p nhận địa chỉ của ds và thực hiện công việc sắp xếp,

sau khi sắp xếp nó lại chả kết quả về cho đối tương ứng của nó là ds, tiếp theo nó gặp vòng for và cứ 1 giá trị của i nó lại gọi hàm in(ds[i]) một lần, và thực hiện việc in. Vì nó gửi(truyền) dữ liệu từ ds cho p, và p làm nhiệm vụ in ngay tại hàm in(p) mà không đem kết quả trở lại, cho nên ta không cần dùng con trỏ *p

Chú ý:

Có 2 cách gọi hàm đó là:

1. *Truyền 1 lúc cả danh sách*:

   khi đó lời gọi hàm tương ứng là (lấy trên làm ví dụ):

   sxếp(ds); /*trong hàm in có vòng for*/

2. *Truyền từng phần tử 1*:

   khi đó lời gọi là:

   for(i=0;i<n;i++) /* trong hàm in(ds) không có vòng for*/

   in(ds[i]); /* **lời gọi hàm***/

Bạn nên cố gắng phân biệt hai trường hợp này, vì nó thể hiện tính đúng dắn của thuật toán. Chứng minh cụ thể bằng ví dụ trên nếu bạn đưa ra lời gọi hàm sxếp là:

for(i=0;i<n;i++)

Sxếp(**ds[i]**); vì truyền từng phần tử 1 nên nó lấy gì ra để so sánh.

Nhưng đối với hàm in(ds); thì khác, vì nó truyền 1 lần 1 phần tử, hay truyền 1 lần nhiều phần tử cũng được. Đôi lúc bạn phải tính đến sự thuận tiện của giải thuật. Vì giả sử Tôi kết hợp hàm tìm(ds) (chỉ tìm 1 người) với hàm in(ds), khi đó bạn dùng lời gọi hàm in theo cách 1 là vô cùng rở hơi (bạn thử nghĩ mà xem tại sao vậy?)

Kết luận:

Trong phần này bạn chỉ cần hiểu 2 điều sau là bạn đã nắm khá vững vấn đề:

1. Bạn hiểu được việc lưu trữ của máy tính => có mấy cách và phần này đang áp dụng cách nào.?
2. Bạn hiểu việc sử dụng hàm và con trỏ như thế nào?. Khi nào thì dùng con trỏ, và việc truyền giá trị cho đối tương ứng ra sao?.

Phần này Tôi không cho bài tập vì nếu như bạn hiểu được các ví dụ trên thì bạn có thể tự ra đề cho mình được. Hoặc là trong sách có rất nhiều mong tự tham khảo.

B. Cấu trúc lồng cấu trúc:

1. *Khai báo cấu trúc lồng*:

Để cho rễ hiểu Tôi khai báo thông qua 1 ví dụ sau: có khai báo

```
Typedef struct
    {
    int   ns;(ngày sinh)
    int   ts;(tháng sinh)
     int   ns;(năm sinh)
    } ngay;

Typedef struct
    {
    Char   họtên[20];  /*mảmg kí tự có 20 kí tự*/
    Char   quequan[20];  /*tương tự trên*/
    ngay      ngàysinh;
    }sinhviên;
```

Khi đó việc truy nhập tới cấu trúc của trường ngày như sau: (với giả sử biến ds[i] là biến cấu trúc kiểu sinhviên)

ds[i].ngàysinh.ngày

Còn mọi công việc giống hệt như cấu trúc không lồng nhau. Tuy vậy Tôi vẫn lấy ví dụ để bạn tham khảo thêm.

VD:

```
typedef struct
        {
        int ngay,thang,nam;
        }ngsinh;
typedef struct
        {
        char     ht[25];
        char     qq[25];
        ngsinh   ns;
        int      tuoi;
        }hocsinh;
void     vaosl(hocsinh *p,int n);
```

```
void     tim(hocsinh *p,int n);
void      in(hocsinh  p);
void vaosl(hocsinh *p,int n)
{
int i;
for(i=0;i<n;i++)
 {
  fflush(stdin);
 printf("\n nhap ho ten nguoi thu %d: ",i);fflush(stdin);
 gets(p[i].ht);
 printf("\n nhap que quan:");fflush(stdin);
 gets(p[i].qq);
 printf("\n ngay sinh:");scanf("%d",&p[i].ns.ngay);
 printf("\n thang sinh:");scanf("%d",&p[i].ns.thang);
 printf("\n nam sinh:");scanf("%d",&p[i].ns.nam);
 printf("\n nhap tuoi:");scanf("%d",&p[i].tuoi);
 }
}
void   tim(hocsinh *p,int n)
{
int i;
char ht[25],ch;
    while(1)
     {
        tt:printf("nhap nguoi can tim:");fflush(stdin);
         gets(ht);
         if(ht[0]==0) break;
         for(i=0;i<n;i++)
       if(strcmp(ht,p[i].ht)==0)  in(ds[i]); /*lời gọi hàm in(p)*/
          ch=getch();
          if(ch=='c'||ch=='C')goto tt;
      }
    }
 void  in(hocsinh  p)
```

```
 {
printf("*****************Ket qua chay*******************\n");
printf("\n Ho ten nguoi thu %d: %s ",i+1,p[i].ht);
printf("\n Que quan: %s ",p[i].qq);
printf("\n Ngay sinh : %d ",p[i].ns.ngay);
printf("\n Thang sinh : %d ",p[i].ns.thang);
printf("\n Nam sinh : %d ",p[i].ns.nam);
printf("\n Tuoi : %d ",p[i].tuoi);
printf("_____________________________________________\n");
 }
 main()
    {
       hocsinh ds[10];
       int i,n;
       clrscr();
       printf("nhap n:=");scanf("%d",&n);
       vaosl(ds,n);
       printf("*********Ket qua tim**********************\n");
       tim(ds,n);
       for(i=0;i<n;i++)
       in(ds[i]);
       getch();
     }
```

C. Khai báo động:

Khai báo động cũng là 1 hình thức lưu trữ kế tiếp. Nhưng nó khác khai báo tĩnh ở trên là ở chỗ cần đến đâu khai báo đến đấy, với mục đích là tiết kiệm bộ nhớ.

Nếu trong khai báo tĩnh là: sinhviên ds[20]; tức là bạn đã ấn định 20 chỗ để lưu trữ 1 danh sách có cấu trúc kiểu sinhviên.

Còn trong khai báo động bạn không làm như vậy, mà bạn phải khai báo thêm 1 con trỏ kiểu sinhviên để nhập dữ liệu.

Cú pháp :

Tên con trỏ =(tên kiểu cấu trúc*)malloc(n+1)*sizeof(tên kiểu cấu trúc);

trong đó:

- Tên con trỏ : Là tên do bạn đặt nó có kiểu là kiểu cấu trúc.
- n : Là số phần tử của cấu trúc do bạn nhập vào.
- Kiểu cấu trúc : Là kiểu mà bạn đã định nghĩa nó gồm các trường.

VD: Có khai báo:

```
typedef struct
  {
  char     ht[25];
  char     qq[25];
  int      tuổi  ;
  }hocsinh;
```

Bạn hãy nhập số người khoảng 5 người vào và cho hiển thị lên màn hình số người vừa nhập.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
typedef struct
      {
      char ht[20];
      char qq[20];
      int  tuoi;
     }sinhvien;
main()
{
int   songuoi,i;
sinhvien   *p;
tt:printf("nhap so nguoi:");scanf("%d",&songuoi);
if (songuoi>5)  goto tt; /*nếu số người nhập >5 thì quay lại chỗ tt: để nhập lại*/
p=(sinhvien*)malloc((songuoi+1)*sizeof(sinhvien));
for(i=0;i<songuoi;i++)
  {
  fflush(stdin);
  printf("\n nhap ho ten:");gets(p[i].ht);
  fflush(stdin);
```

```
    printf("\n nhap que quan:");gets(p[i].qq);
    printf("\n nhap tuoi:");scanf("%d",&p[i].tuoi);
    }
  for(i=0;i<songuoi;i++)
     {
     printf("\n ho ten:%s",p[i].ht);
     printf("\n que quan:%s",p[i].qq);
     printf("\n tuoi:%d",p[i].tuoi);
     }
  getch();
  }
```

Qua ví dụ trên cho thấy khai báo động và khai báo tĩnh chỉ khác nhau hai điểm sau:

- Nhập dữ liệu cho danh sách trong khai báo động bắt buộc phải nhập bằng con trỏ. Còn trong khai báo tĩnh thì có thể là con trỏ hoặc không cần con trỏ.
- Nếu là khai báo động thì có cú pháp khai báo động riêng.

Còn sau đó tất cả mọi công việc và cách làm đều như nhau.

Bây giờ Tôi xẽ lấy một ví dụ tổng hợp các công việc để bạn được rõ hơn. Nếu bạn chưa thành thạo việc truyền đối trong lời gọi hàm, thì hãy quan sát thật kĩ việc truyền đối trong chương trình này. Với một chú ý là tất cả các tham biến ,tham trị trong mỗi hàm là hoàn toàn khác nhau mặc dù chúng có tên giống nhau. Điển hình là biến con trỏ *p1 trong chương trình này.

VD:

Nhập vào một danh sách sinh viên có số người do bạn quyết định. Yêu cầu mỗi sinh viên được nhập có đủ các thông tin :họ tên, quê quán và tuổi.

Chương trình :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
typedef struct
     {
     char ht[20];
     char qq[20];
```

```
        int  tuoi;
       }sinhvien;
void nhap(sinhvien *p1,int songuoi);
void  in(sinhvien p1);
void  sxep(sinhvien *p1,int songuoi);
void  tim(sinhvien *p1,int songuoi);
void  nhap(sinhvien *p1,int songuoi)
{
int i;
for(i=0;i<songuoi;i++)
        {
        fflush(stdin);
        printf("\n nhap ho ten:");gets(p1[i].ht);
        fflush(stdin);
        printf("\n nhap que quan:");gets(p1[i].qq);
        printf("\n nhap tuoi:");scanf("%d",&p1[i].tuoi);
        }
 }
void tim(sinhvien *p1,int songuoi)
{
int i;
char hoten[20];
while(1)
  {
  fflush(stdin);
  printf("\n nhap ho ten:"); gets(hoten);
  if(hoten[0]==0) break;
  for(i=0;i<songuoi;i++)
     if(strcmp(p1[i].ht,hoten)==0)
      in(p1[i]);
  }
}
void sxep(sinhvien *p1,int songuoi)
{
```

```
 int i,j;
 sinhvien tg;
  for(i=0;i<songuoi-1;i++)
     for(j=i+1;j<songuoi;j++)
         if(p1[i].tuoi>p1[j].tuoi)
           {
           tg=p1[i];
           p1[i]=p1[j];
           p1[j]=tg;
           }
 }
    void  in(sinhvien  p1)
   {
   printf("\n ho ten:%s",p1.ht);
   printf("\n que quan:%s",p1.qq);
   printf("\n tuoi:%d",p1.tuoi);
   }
      main()
      {
      clrscr();
      int songuoi,i;
      sinhvien *p;
      tt:printf("nhap so nguoi:");scanf("%d",&songuoi);
      if (songuoi>5)  goto tt;
      p=(sinhvien*)malloc((songuoi+1)*sizeof(sinhvien));

      nhap(p,songuoi);
      sxep(p,songuoi);

      for(i=0;i<songuoi;i++)
      in(p[i]);
      tim(p,songuoi);
      getch();
}
```

III. CẤU TRÚC LƯU TRỮ MÓC NỐI :

Đây là phần (theo Tôi đánh giá) khó nhất trong chương trình C vì nó rất phức tạp từ việc khai báo đến sử dụng. Phần này để hiểu được yêu cầu bạn phải có một chút kiến thức môn giải thuật, nói vậy không phải để bạn nản lòng bạn phải biết đây là phần cực kì quan trọng và bao giờ nó cũng chiếm 50% số điểm của bài thi. Nhưng không sao chúng ta xẽ cố gắng phơi bày bản chất của nó thông qua lời giải thích và những ví dụ cụ thể.

1. *Đinh nghĩa cho một cấu trúc móc nối* :

Cấu trúc móc nối là cấu trúc có các trường thông tin và ít nhất một con trỏ trỏ vào chính nó. Và được khai báo như sau:

```
typedef struct     (tên cấu trúc)
                   {
                   kiểu trường 1      tên trường;
                   kiểu trường 2      tên trường;
                   kiểu trường 3      tên trường;
                     .....
                   struct  tên cấu trúc   *tên con trỏ;
                   } tên cấu trúc;
```

VD:

```
typedef struct  sinhvien
        {
         char     ht[20];
        char     qq[20];
        int      tuoi;
        struct   sinhvien   *tiep; /*nhất thiết phải có */
       }sinhvien;
```

Sau Khai báo này bạn đã có một kiểu dữ liệu có tên là tên cấu trúc.Các trường thì hoàn toàn giống các trường trong cấu trúc lưu trữ kế tiếp.

2. *Nhập dữ liêu cho một cấu trúc móc nối* :

Giả thiết bạn có khai báo sau:

```
typedef struct  sinhvien
    {
```

```
char     ht[20];
char     qq[20];
int      tuoi;
struct   sinhvien   *tiep;
}sinhvien;
```

Khai báo trên cho biết bạn. Có một cấu trúc sinhviên gồm 3 trường, và một con trỏ tên là tiếp. Bây giờ bạn có thể dùng cấu trúc này để khai báo các biến cấu trúc kiểu sinh viên. Với một chú ý là việc nhập dữ liệu hay bất cứ một thao tác nào trên cấu trúc đều phải sử dụng con trỏ. Khác với lưu trữ mảng, lưu trữ móc nối luôn có 1 con trỏ trỏ đến đầu danh sách, mục đích để dữ đầu mối, mỗi khi bạn thao tác trên cấu trúc đều bắt đầu từ đầu mối này.Nếu muốn duyệt qua danh sách bạn phải sử dụng thêm 1 con trỏ khác để duyệt.

Chú ý

Đây cũng là 1 hình thức khai báo động, cho nên đầu chương trình bạn phải khai báo thư viện "alloc.h".

Cú pháp

khai báo động như sau:

pd=(tên cấu trúc*)malloc(sizeof(tên cấu trúc));

Trong đó pd là con trỏ trỏ đầu danh sách. Nếu như từ phần tử thứ hai trở đi bạn phải dùng con trỏ khác để nhập dữ liệu.

Tôi xẽ mô tả bằng lời quá trình nhập dữ liệu và sau đó là ví dụ điển hình:

Với một cấu trúc được định nghĩa như trên bạn cần hai con trỏ để phục vụ cho việc nhập dữ liệu

Con trỏ 1 luôn nắm đầu danh sách. Nhưng lúc đầu danh sách còn rỗng, vì vậy nó làm nhiệm vụ nhập phần tử đầu tiên. Sau đó nó nhường lại quyền này cho con trỏ thứ 2 con trỏ này tiếp tục làm nhiệm vụ được giao đồng thời nắm dữ cuối danh sách để nhập tiếp dữ liệu, đến khi thôi thì nó cũng coi như hết nhiệm vụ và đương nhiên nó không còn liên quan gì tới danh sách này nữa.Sau khi nhập xong bạn đã có 1 danh sách: Có con trỏ pd đang nắm đầu danh sách.

VD:

Hãy nhập từ bàn phím danh sách một số sinh viên gồm các thông tin sau: họ tên , quê quán, và tuổi.

Chương trình :

```
#include<stdio.h>
```

```
#include<conio.h>
#include<string.h>
#include<alloc.h>
typedef struct  sinhvien
        {
        char      ht[20];
        char      qq[20];
        int       tuoi;
        struct   sinhvien   *tiep;
        }sinhvien;
  main ()
{
char hoten[20];    /* bắt buộc phải khai báo thêm vì bạn dùng vòng while(1)*/
sinhvien  *pd,*p;  /* hai biến trỏ trong thân hàm*/
pd=NULL;    /*đầu tiên pd không trỏ vào đâu cả*/
while(1)      /* gọi là vòng lặp vô tận để thoát khỏi vòng này phụ thuộc vào câu lệnh
                    break */
      {
      fflush(stdin);
      printf("nhap ho ten:");gets(hoten);
      if(hoten[0]==0)break;   /* nếu hoten=0  thì thoát khỏi vòng lặp */
      if(pd==NULL)            /* nếu là phần tử đầu tiên của danh sách*/
         {
         pd=(sinhvien*)malloc(sizeof(sinhvien)); /*xin bộ nhớ cho phần tử đầu tiên */
         p1=pd;  /* gán 1 con trỏ khác = pd để nó làm tiếp nhiệm vụ nhập */
         }
      else
         {  /* từ lần nhập thứ 2 trở đi */
          p1->tiep=(sinhvien*)malloc(sizeof(sinhvien));
          p1=p1->tiep;    /*p1->tiếp <=> link(p1) trong giải thuật */
          }
      strcpy(p1->ht,hoten); /* câu lệnh nhập trường ht vào trong danh sách nó khác
câu lệnh nhập qq ở dưới vì nó đã được nhập vào máy rồi*/
      printf("\n nhap que quan:");gets(p1->qq);
```

```
printf("\n nhap tuoi:");scanf("%d",&(p1->tuoi));
p1->tiep=NULL;  /*sau mỗi vòng lặp ấn định không còn phần tử tiếp theo vì có thể quay lại vòng khác bạn không nhập nữa*/
}
 }
```

Chú ý:

Bạn nên làm quen với những quy định sau:

1. p=NULL: tức là không trỏ vào đâu cả.Con trỏ chỉ có thể nắm địa chỉ của 1 phần tử trong danh sách.
2. p=p->tiêp <=> p=link(p) <=>a[i] =>a[i+1] :phần tử tiếp theo của danh sách. Nhưng với chú ý p->tiếp bao giờ cũng chạy trước p một bước.
3. p->tiếp=NULL:không trỏ vào đâu nữa và cũng là phần tử cuối cùng của 1 danh sách.
4. strcpy(p->ht,hoten); đẩy trường họtên vào trong danh sách bắt buộc hotên đã được nhập vào trong máy rồi. Chú ý trường ht[20] hoàn toàn khác với hoten[20].

3.*Thao tác trên cấu trúc:*

Có rất nhiều thao tác trên danh sách ví dụ như hiển thị danh sách lên màn hình,tìm tên một người nào đó có hoặc không có trong danh sách, loại bỏ một người ra khỏi danh sách hoặc bổ sung.....Nhưng ở trước hết bạn hãy làm quen với hai công 3 công việc đơn giản nhất đó là nhập, hiển thị và tìm người nào đó có trong danh sách. Còn nếu bạn muốn tìm hiểu sâu hơn thì hãy xem ví dụ tổng hợp cuối phần này.

Ở trên Tôi đã giới thiệu cách nhập 1 danh sách bây giờ Tôi giới thiệu tiếp việc in 1 danh sách và tìm người.

a. *Hiển thị danh sách:*

Sau khi bạn nhập vào 1 danh sách và bạn ấn định 1 con trỏ(có tên) dữ đầu danh sách khi đó bạn muốn truy nhập vào danh sách đó bạn phải dùng 1 con trỏ khác có cùng kiểu dữ liệu gán bằng nó để làm nhiệm vụ(nếu không làm như vậy bạn sẽ mất đầu mối tức là mất địa chỉ), bằng câu lệnh:

VD p1=pd;

Và phép duyệt danh sách tương ứng là: p1=p->tiếp xẽ cho phép bạn đi từ đầu đến cuối danh sách.

Cùng với ví dụ trên Tôi xẽ làm tiếp phần còn lại của công việc là hiển thị ra màn hình.

Tiếp ví dụ trên:

```
   p1=pd;
     while(p1!=NULL)
          {
          printf("\n in ho ten:%s",p1->ht);
          printf("\n in que quan:%s",p1->qq);
          printf("\n in tuoi:%d",p1->tuoi);
          p1=p1->tiep;
          }
}
```

Cứ 1 vòng lặp thì in được 1 người vì con trỏ chỉ dữ địa chỉ được 1 người.
Trong while() bạn thử thay =p1->tiep!=NULL xem điều gì xẽ sảy ra.

b. *Tìm người khi biết tên:*

Đầu tiên bạn phải nhập tên của 1 người bất kì sau đó kiểm tra xem người tên này có trong danh sách không bằng câu lệnh:

```
strcmp(p->ht,hoten)= =0;
```

Nếu đúng =0 thì là có trong danh sách còn ngược lại thì không. Sau đó in người tìm được ra. Vẫn tiếp ví dụ trên Tôi làm tiếp công việc tìm người trên danh sách đó:

```
tt:printf("\n nhap ten nguoi can tim:");fflush(stdin);gets(hoten);
if(hoten[0]==0) goto tt;
p1=pd;
while(p1!=NULL)
   {
   if(strcmp(hoten,p1->ht)==0)
        {
          printf("\n nguoi can tim la:%s",p1->ht);
          printf("\n que quan:%s",p1->qq);
          printf("\n tuoi:%d",p1->tuoi);
        }
    p1=p1->tiep;
    }
```

4. Sử dụng hàm:

Để tiện so sánh Tôi chia chương trình trên thành 3 hàm với 3 công việc khác nhau là : Hàm void nhap() để nhập dữ liệu.

Hàm void in() để in dữ liệu.

Hàm void tim() để tìm và in thông tin người tìm được.

Chương trình :

Đối với loại cấu trúc này thường sử dụng hàm không đối. Nhưng vì sao nó lại làm thay đổi được kết quả: Vì chúng ta chỉ thao tác trên cùng 1 cấu trúc. Pd là biến toàn cục, trong mỗi hàm bất kì nào đó bạn đều gán: Ví dụ p1=pd, thì danh sách này được đem ra làm việc và bạn có thể bỏ đi 1 phần tử nào đó(ví dụ phép loại bỏ).Khi đó danh sách ban đầu đã bị thay đổi không còn nguyên vẹn như ban đầu.

Với *p1,*pd là hai biến toàn cục tức là sử dụng cho mọi hàm được khai báo ở đầu chương trình. Xem CT ở dưới.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
typedef struct  sinhvien
        {
        char     ht[20];
        char     qq[20];
        int      tuoi;
        struct   sinhvien   *tiep;
        }sinhvien;
void nhap();
void tim();
void  in();
sinhvien *pd,*p1;

void nhap()
{
char hoten[20];
pd=NULL;
```

```
while(1)
      {
      fflush(stdin);
      printf("nhap ho ten:");gets(hoten);
      if(hoten[0]==0)break;
      if(pd==NULL)
         {
         pd=(sinhvien*)malloc(sizeof(sinhvien));
         p1=pd;
         }
      else {
          p1->tiep=(sinhvien*)malloc(sizeof(sinhvien));
          p1=p1->tiep;
          }

      strcpy(p1->ht,hoten);
      printf("\n nhap que quan:");gets(p1->qq);
      printf("\n nhap tuoi:");scanf("%d",&(p1->tuoi));
      p1->tiep=NULL;
      }
}
void  in()
{
p1=pd;
while(p1!=NULL)
      {
      printf("\n in ho ten:%s",p1->ht);
      printf("\n in que quan:%s",p1->qq);
      printf("\n in tuoi:%d",p1->tuoi);
      p1=p1->tiep;
      }
}

void tim()
```

```
{
char hoten[20];

 tt:printf("\n nhap ten nguoi can tim:");fflush(stdin);gets(hoten);
 if(hoten[0]==0) goto tt;
 p1=pd;
 while(p1!=NULL)
    {
    if(strcmp(hoten,p1->ht)==0)
          {
             printf("\n nguoi can tim la:%s",p1->ht);
             printf("\n que quan:%s",p1->qq);
             printf("\n tuoi:%d",p1->tuoi);
          }
     p1=p1->tiep;
     }
}
main()
{
clrscr();
nhap();
in();
tim();
getch();
}
```

VD2:

Cũng với vấu trúc trên Tôi thiết kế 1 chương trình sử dụng ba hàm

Hàm void làm nhiệm vụ loại bỏ 1 người có tên trong danh sách.

Trình tự làm việc của hàm này như sau : Đầu tiên hàm bắt nhập vào một cái tên bất kì, sau đó nó so sánh (dùng 1con trỏ p1 để đi tìm. Cùng với p1 có 1 con trỏ *ps chạy liền sau nó, con trỏ này xẽ làm nhiệm vụ liên kết người đứng trước và người đứng sau p1 nếu p1 bị loại) xem có tên nào trùng với tên này không. Nếu thì nó xẽ in là không có. Ngược lại, nếu nó kiểm tra đúng là có thì nó dừng lại nắm lấy địa chỉ tại người đó và bắt đầu thực hiện loại, bằng cách lấy liên kết của người trước nó trỏ vào người sau

nó, chứ không trỏ vào nó nữa. Thoát khỏi chương trình nó đã làm mất liên kết của 1 phần tử trong danh sách .

Hàm main làm nhiệm vụ nhập dữ liệu và gọi hàm.

Hàm in kết quả nhập và kết quả tìm.

Chương trình :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
typedef struct  sinhvien
    {
    char     ht[20];
    char     qq[20];
    int      tuoi;
    struct   sinhvien   *tiep;
    }sinhvien;

void loai();
void in();
sinhviên   *pd,*p1;   /*hai biến toàn cục*/
  void loai()    /*bắt đầu hàm loại */
 {
 char hoten[20];
 int kq;
 sinhvien *ps;
     tt:printf("\n nhap ten nguoi bi loai:");fflush(stdin);
     scanf("%s",&hoten);
     if(hoten[0]==0) goto tt;
     if((hoten!=0)&&(pd!=0))
   {
   p1=pd;
   ps=p1;
  while(kq= strcmp(p1->ht,hoten)!=0)  /*nếu biến kq còn khác 0 còn làm*/
       {
```

```
            ps=p1;
            p1=p1->tiep;
            }  /* thoát khỏi vòng lặp p1 đang trỏ đến người cần tìm,*ps trỏ người liền
sau nó chú ý liền sau tức là đã được duyệt rồi*/
        if(kq==0)
          {
         if(strcmp(pd->ht,p1->ht)==0)
                        {
                        pd=pd->tiep;
                        free(p1);
                        return;
                        }
                else
                        {
                        ps->tiep=p1->tiep;
                        free(p1);
                    }
          }
       else
          {
           printf("\n khong co nhuoi can tim:");
           getch();
          }
       }
    }

    void  in()/*hàm in*/
    {
    p1=pd;
    while(p1!=NULL)
       {
       printf("\n in ho ten:%s",p1->ht);
       printf("\n in que quan:%s",p1->qq);
       printf("\n in tuoi:%d",p1->tuoi);
```

```
    p1=p1->tiep;
    }
}
main()
{
char hoten[20];
pd=NULL;
while(1)
   {
   fflush(stdin);
   printf("nhap ho ten:");gets(hoten);
   if(hoten[0]==0)break;
   if(pd==NULL)
      {
      pd=(sinhvien*)malloc(sizeof(sinhvien));
      p1=pd;
      }
   else {
       p1->tiep=(sinhvien*)malloc(sizeof(sinhvien));
       p1=p1->tiep;
       }

   strcpy(p1->ht,hoten);
   printf("\n nhap que quan:");gets(p1->qq);
   printf("\n nhap tuoi:");scanf("%d",&(p1->tuoi));
   p1->tiep=NULL;
   }
in();    /* in danh sách vừa nhập*/
loai();  /*gọi hàm loại*/
in();    /*gọi hàm in để in danh sách sau khi loại*/
getch();
}
```

Đến đây Tôi xin kết thúc phần này ở đây. Hi vọng bạn hiểu thế nào là cấu trúc mảng và thế nào là cấu trúc móc nối và thao tác trên mỗi cấu trúc.

PHẦN 3. TỆP TIN VÀ THAO TÁC TRÊN TỆP TIN.

(SẼ HOÀN THIỆN SAU. MONG CÁC BẠN THÔNG CẢM)