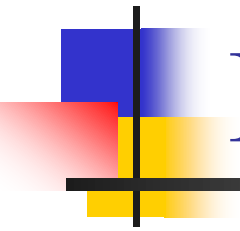


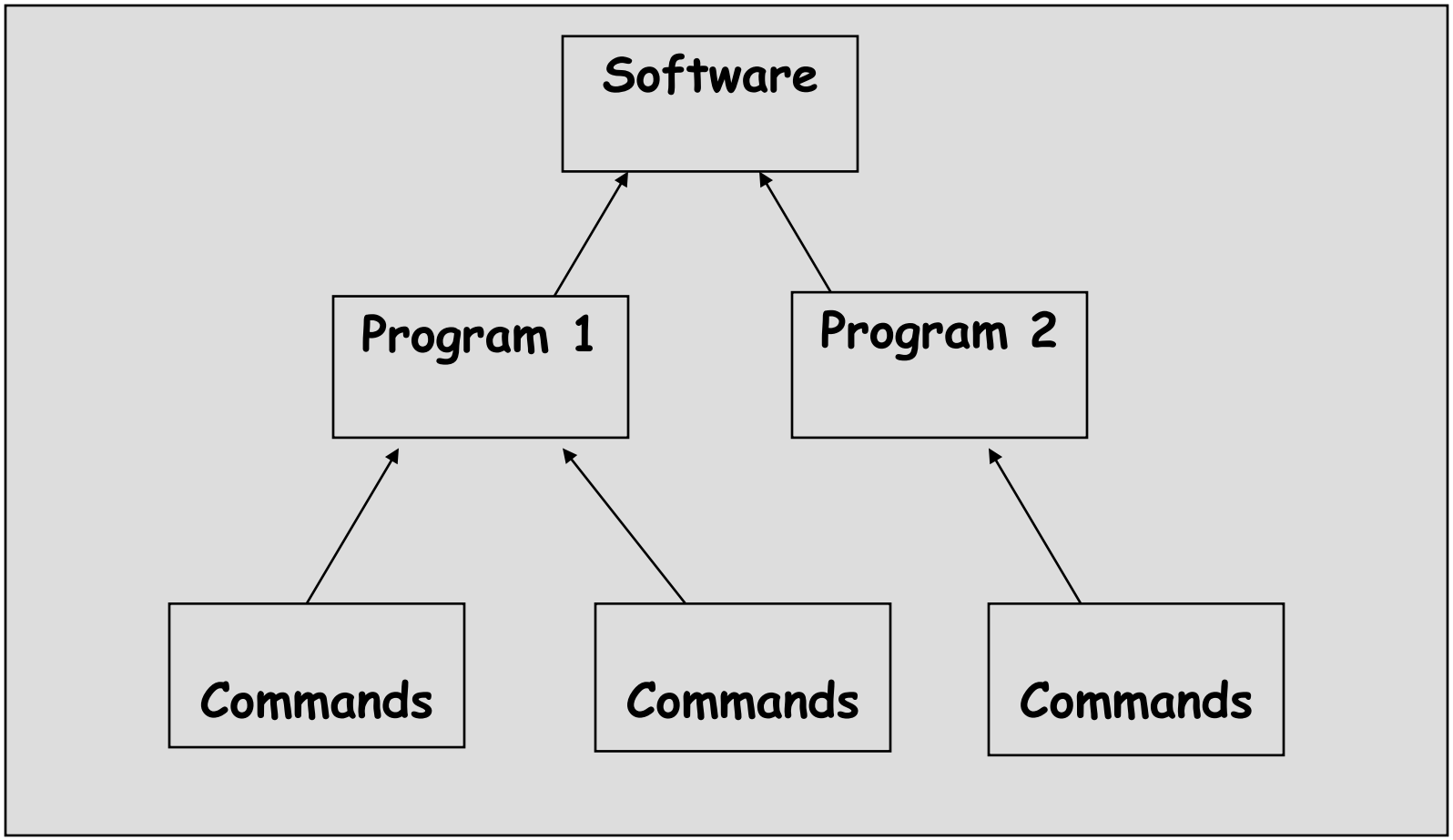
Khái niệm cơ bản C

Chương 1

Mục Tiêu

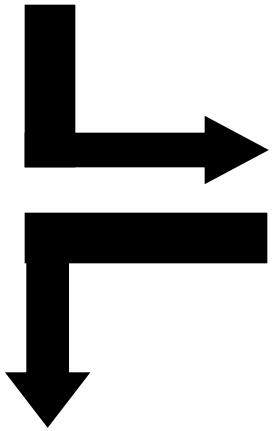
- Phân biệt sự khác nhau giữa Câu lệnh, Chương trình và Phần mềm
- Biết được quá trình hình thành ngôn ngữ C
- Biết được khi nào dùng C và tại sao
- Nắm được cấu trúc ngôn ngữ C
- Hiểu rõ khái niệm giải thuật (algorithms)
- Vẽ lưu đồ (flowchart)
- Sử dụng được các ký hiệu dùng trong lưu đồ

Phần mềm, chương trình, câu lệnh



Bắt đầu C

BPCL – Martin Richards



B – Ken Thompson

C – Dennis Ritchie

Các lĩnh vực ứng dụng của C

- C được dùng để lập trình hệ thống
- Một chương trình hệ thống làm thành một phần hệ điều hành hoặc các tiện ích hỗ trợ của hệ điều hành
- Hệ điều hành (Operating Systems), trình thông dịch (Interpreters), trình soạn thảo (Editors), trình Hợp Ngữ (Assembly) được gọi là chương trình hệ thống
- Hệ điều hành UNIX được phát triển dựa vào C
- Có các trình biên dịch dành cho hầu hết các loại hệ thống PC

Ngôn ngữ cấp trung

Ngôn ngữ cấp cao

C

Ngôn ngữ hợp ngữ

Ngôn ngữ có cấu trúc

- C cho phép tổng hợp mã lệnh và dữ liệu

- Nó có khả năng tập hợp và ẩn đi tất cả thông tin, lệnh khỏi phần còn lại của chương trình để dùng cho những tác vụ riêng

```
do
{
    i = i + 1;
    .
    .
    .
} while (i < 40);
```

- Chương trình C có thể được chia nhỏ thành những hàm (functions) hay những khối mã (code blocks).

Đặc điểm của C

- **C** có 32 từ khóa
- Những từ khóa này kết hợp với cú pháp của C hình thành ngôn ngữ C
- Các quy tắc được áp dụng cho các chương trình C
  - Tất cả từ khóa là chữ thường
  - Đoạn mã trong chương trình C có phân biệt chữ thường, chữ hoa, do while khác DO WHILE
  - Từ khóa không thể dùng đặt tên biến (variable name) hoặc tên hàm (function name)

```
main()
{
/* This is a sample Program*/
      int i,j;
      i=100;
      j=200;
        :
   }
```

Cấu trúc chương trình C

main()

- Chương trình C được chia nhỏ thành những đơn vị gọi là hàm
- Không kể có bao nhiêu hàm trong chương trình, Hệ điều hành luôn trao quyền điều khiển cho hàm main() khi một chương trình C được thực thi.
- Theo sau tên hàm là dấu ngoặc đơn
- Dấu ngoặc đơn có thể có chứa hay không chứa những tham số

Cấu trúc chương trình C (tt.)

Dấu phân cách {…}

- Sau phần đầu hàm là dấu ngoặc xoắn mở **{**
- Nó cho biết việc thi hành lệnh trong hàm bắt đầu
- Tương tự, dấu ngoặc xoắn đóng **}** sau câu lệnh cuối cùng trong hàm chỉ ra điểm kết thúc của hàm

Cấu trúc chương trình C (tt.)

Dấu kết thúc câu lệnh … ;

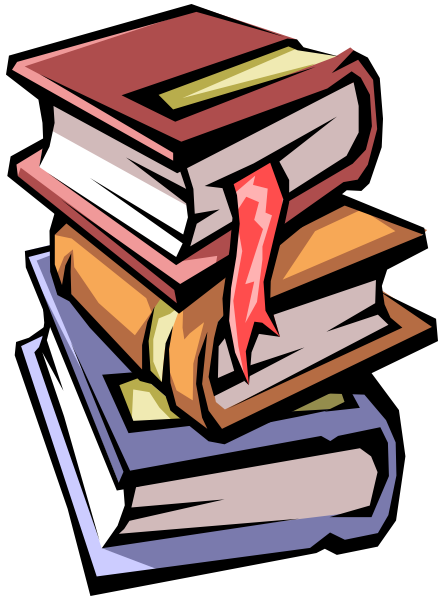
- Một câu lệnh trong C được kết thúc bằng dấu chấm phẩy **;**
- Trình biên dịch C không hiểu việc xuống dòng, khoảng trắng hay tab
- Một câu lệnh không kết thúc bằng dấu chấm phẩy sẽ được xem như dòng lệnh lỗi trong C

Cấu trúc chương trình C (tt.)

/*Dòng chú thích*/

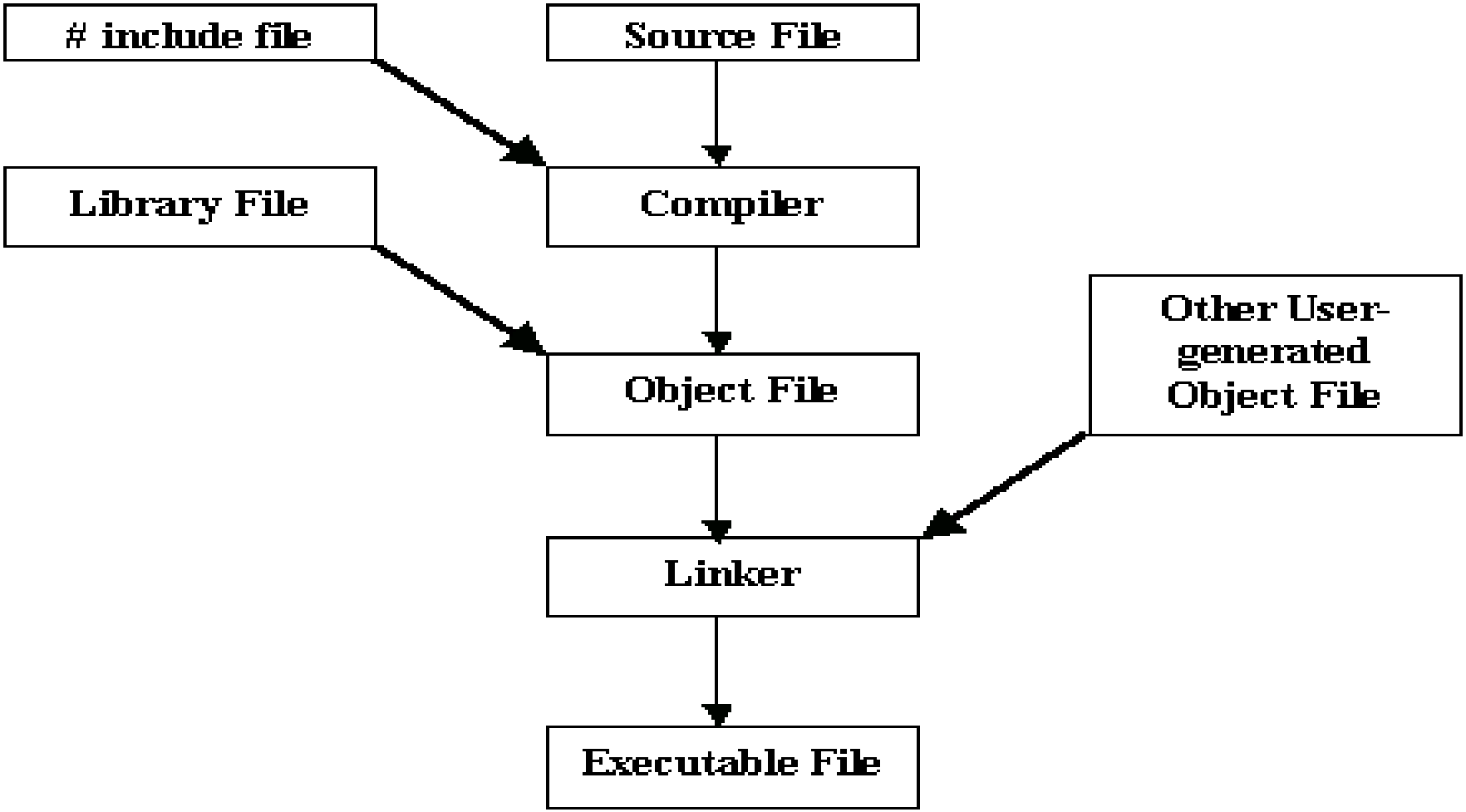
- Những chú thích thường được viết để mô tả công việc của một lệnh đặc biệt, một hàm hay toàn bộ chương trình
- Trình biên dịch sẽ bỏ qua phần chú thích
- Trong trường hợp chú thích nhiều dòng, nó sẽ bắt đầu bằng ký hiệu /* và kết thúc là */

Thư viện C



- 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 trình biên dịch yêu cầu tạo một thư viện riêng

Biên dịch và thi hành chương trình



Các bước lập trình giải quyết vấn đề

Giải thuật gồm một tập hợp các bước thực hiện nhằm giải quyết một vấn đề. Thí dụ sau đây mô tả một giải thuật

Rời phòng học

Phòng học

Đến cầu thang

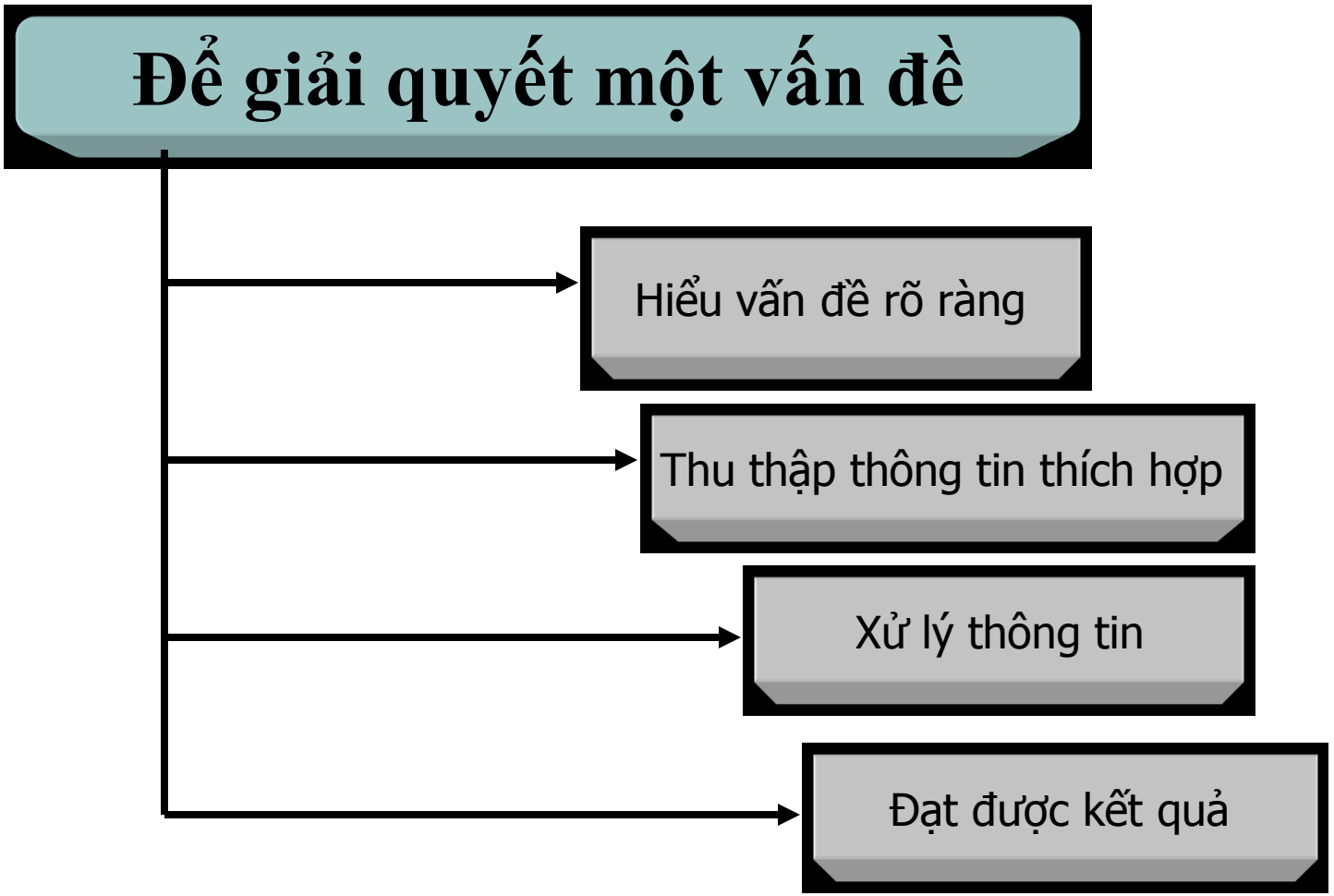
Xuống tầng hầm

Đi đến quán ăn tự phục vụ

Cafeteria

Đây là các bước thực hiên khi một người muốn đi đến quán ăn tự phục vụ từ phòng học

Giải quyết một vấn đề



Mã giả (Pseudocode)

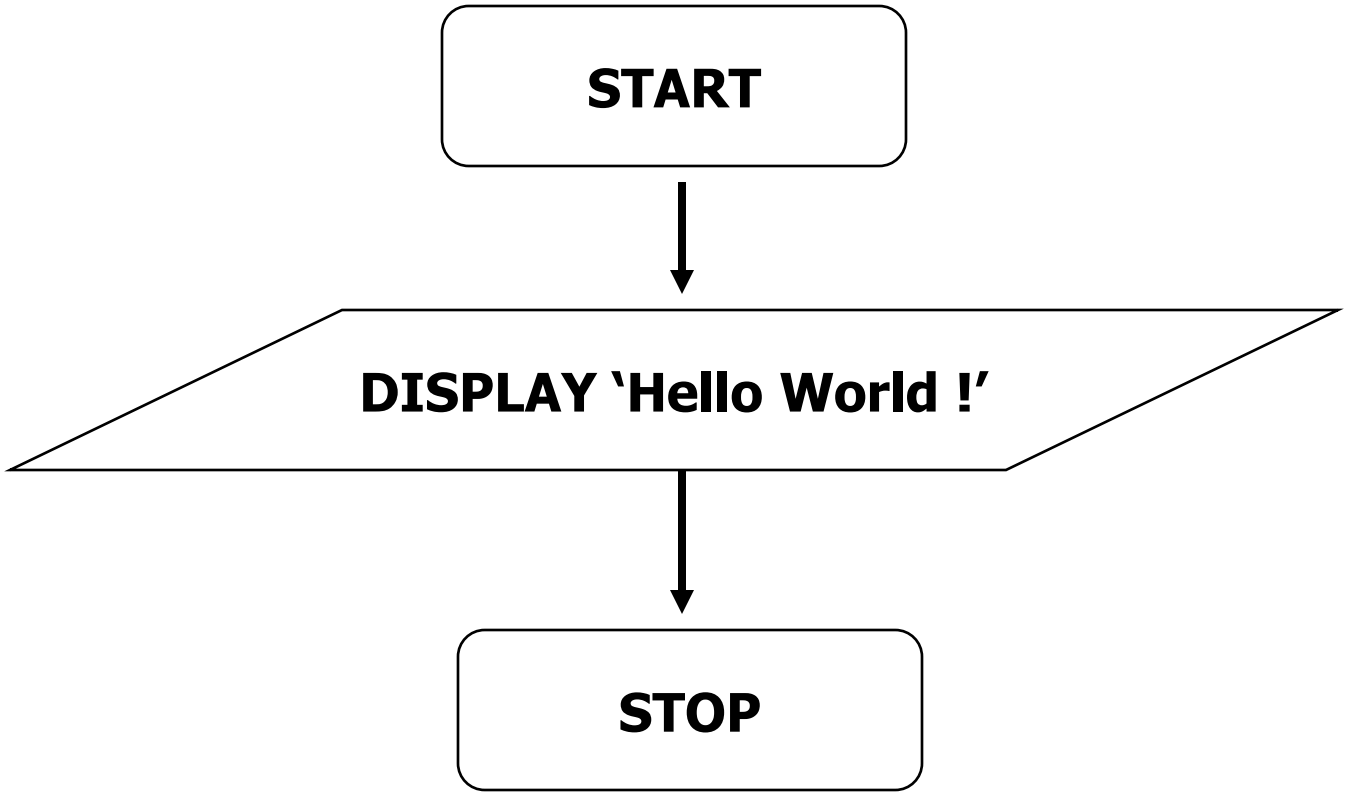
Không là mã thật. Một phương pháp viết giải thuật sử dụng một tập hợp các từ tương tự mã thật

```
BEGIN
DISPLAY 'Hello World !'
END
```

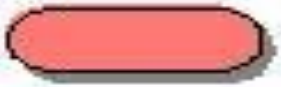
Mỗi đoạn mã giả phải bắt đầu với một từ BEGIN
Để hiển thị giá trị nào đó, từ DISPLAY được dùng
Mã giả kết thúc với từ END

Lưu đồ (Flowcharts)

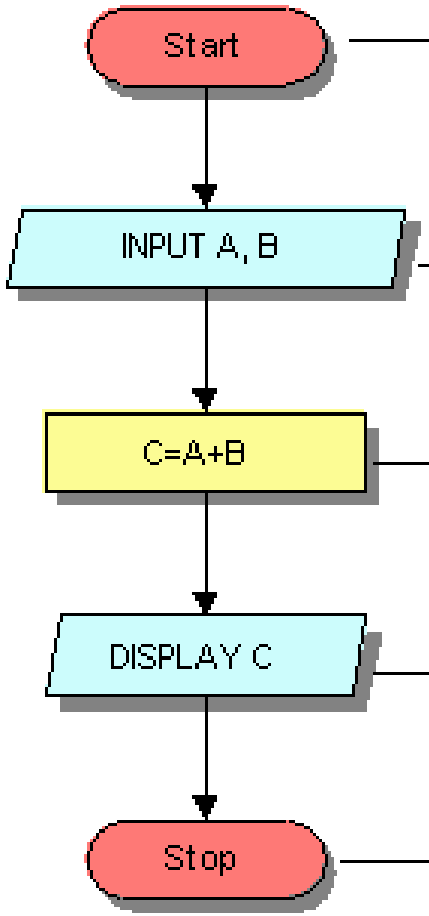
Lưu đồ là một hình ảnh minh hoạ cho giải thuật



Biểu tượng trong lưu đồ

Biểu Tượng	Mô Tả
	Bắt đầu hay kết thúc chương trình
	Những bước tính toán
	Các lệnh xuất hay nhập
	Quyết định và rẽ nhánh
	Bộ nối hai phần trong chương trình (đầu nối)
	Dòng chảy

Lưu đồ cộng hai số



Biểu tượng giới hạn :Dùng ở đây để bắt đầu lưu đồ

Biểu tượng xuất hay nhập: Dùng ở đây để nhập hai số

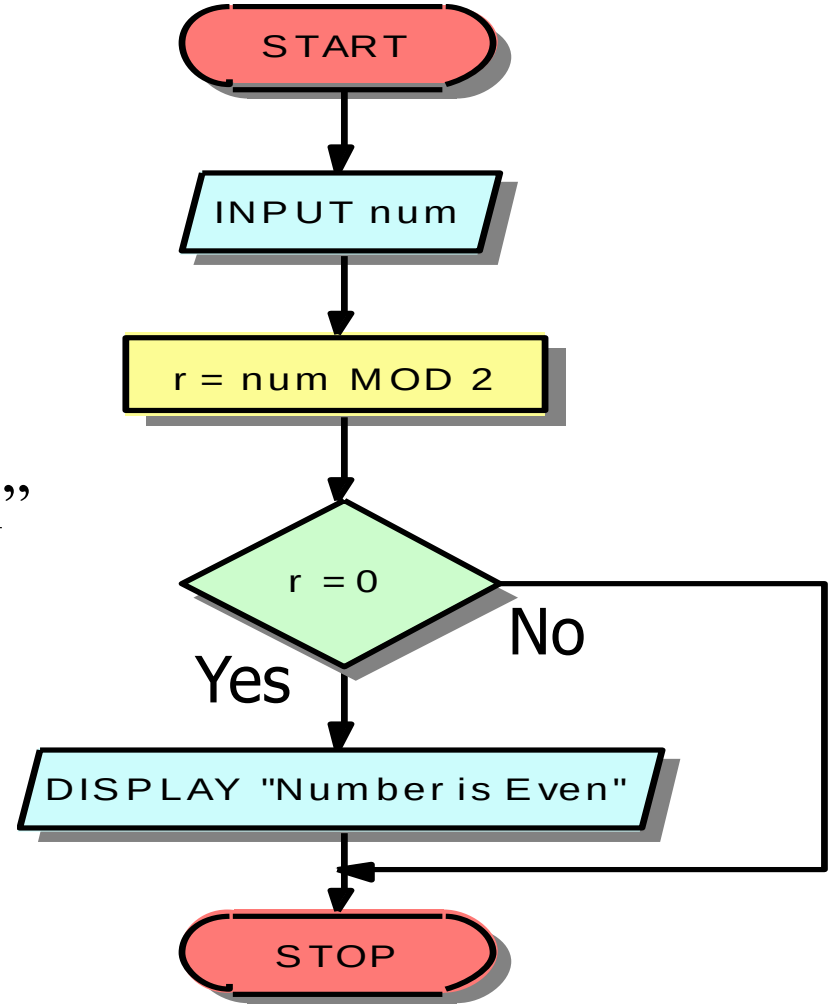
Biểu tượng xử lý : Dùng ở đây để cộng hai số

Biểu tượng xuất hay nhập: Dùng ở đây để hiện thị tổng C

Biểu tượng giới hạn: Dùng ở đây để kết thúc lưu đồ

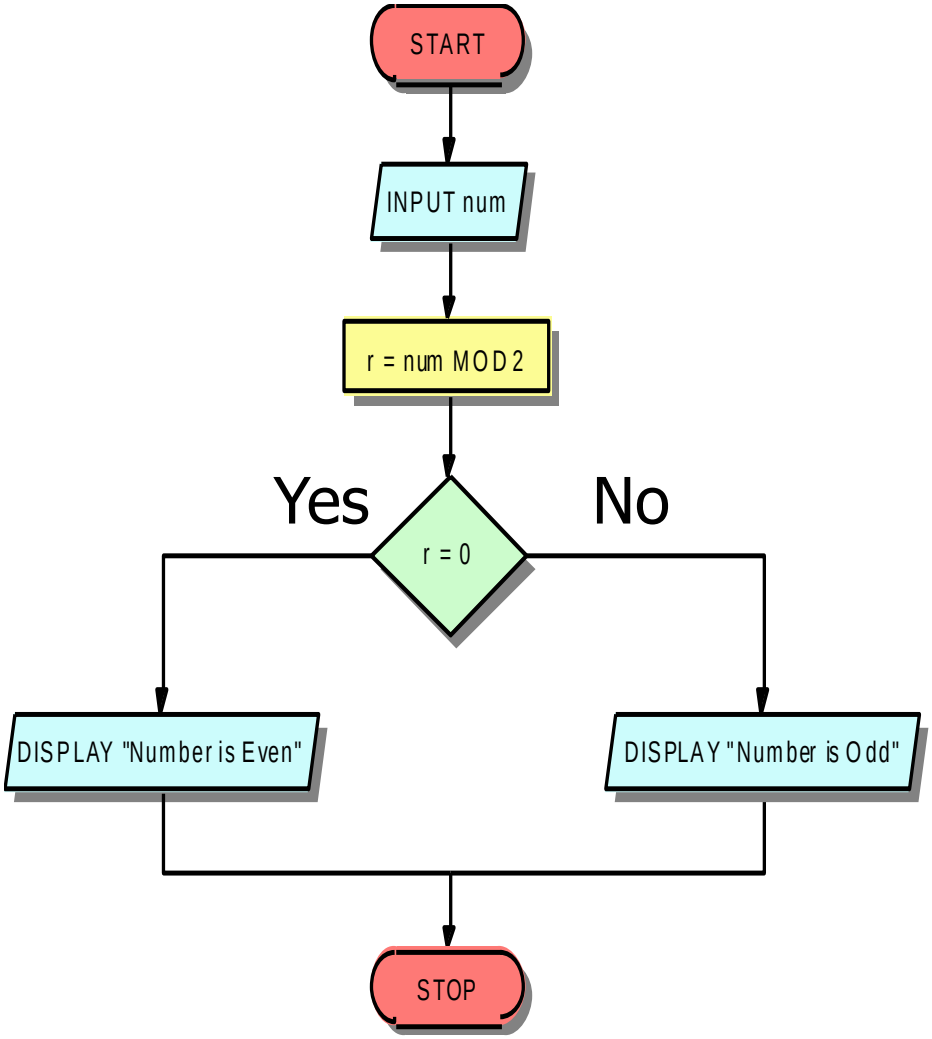
Cấu trúc IF

BEGIN
INPUT num
r = num MOD 2
IF r=0
    Display “Number is even”
END IF
END



Cấu trúc IF…ELSE

```
BEGIN
INPUT num
r=num MOD 2
IF r=0
   DISPLAY “Even Number”
ELSE
   DISPLAY “Odd Number”
END IF
END
```



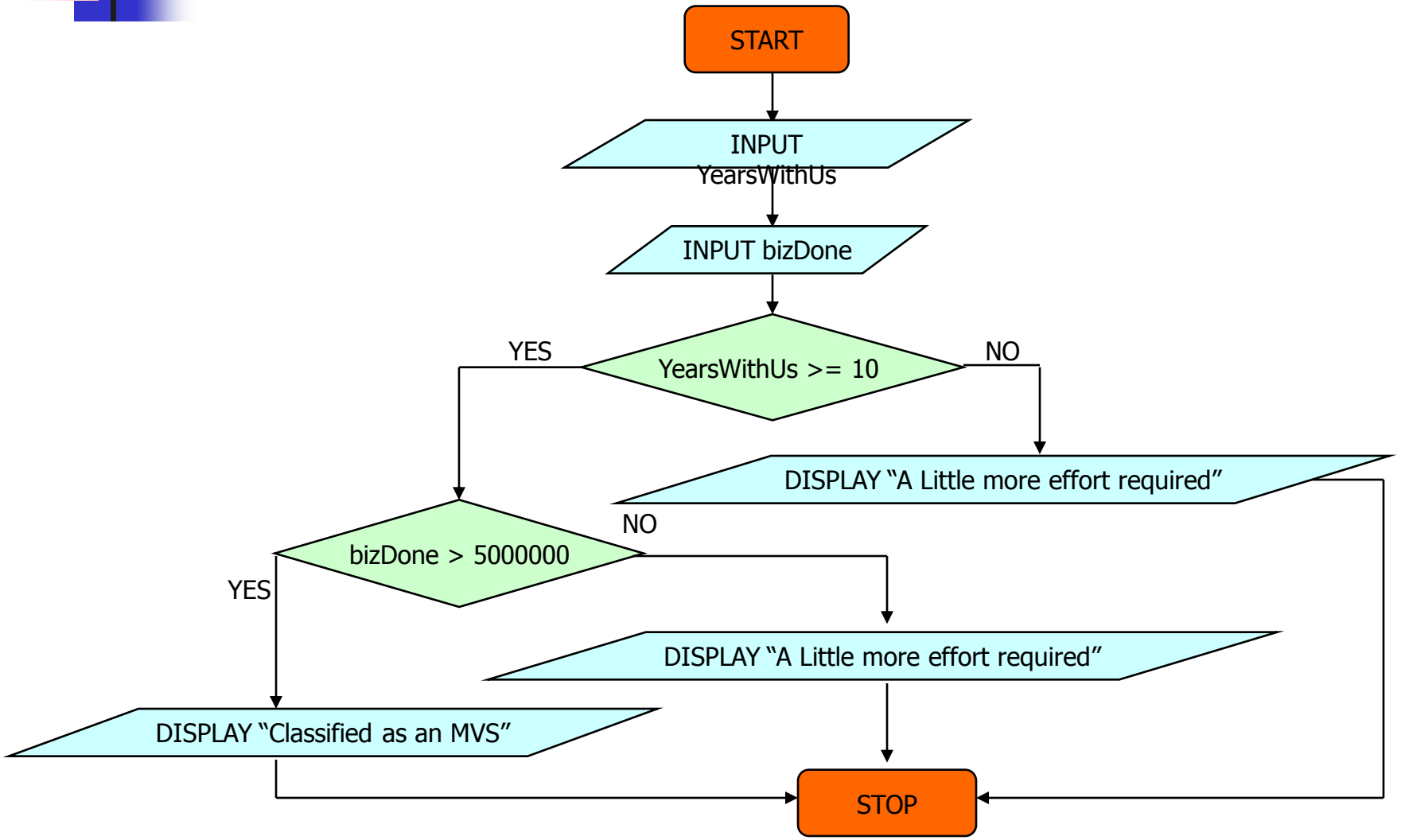
Đa điều kiện sử dụng AND/OR

```
BEGIN
INPUT yearsWithUs
INPUT bizDone
IF yearsWithUs >= 10 AND bizDone >=5000000
        DISPLAY “Classified as an MVS”
ELSE
        DISPLAY “A little more effort required!”
END IF
END
```

Cấu trúc IF lồng nhau

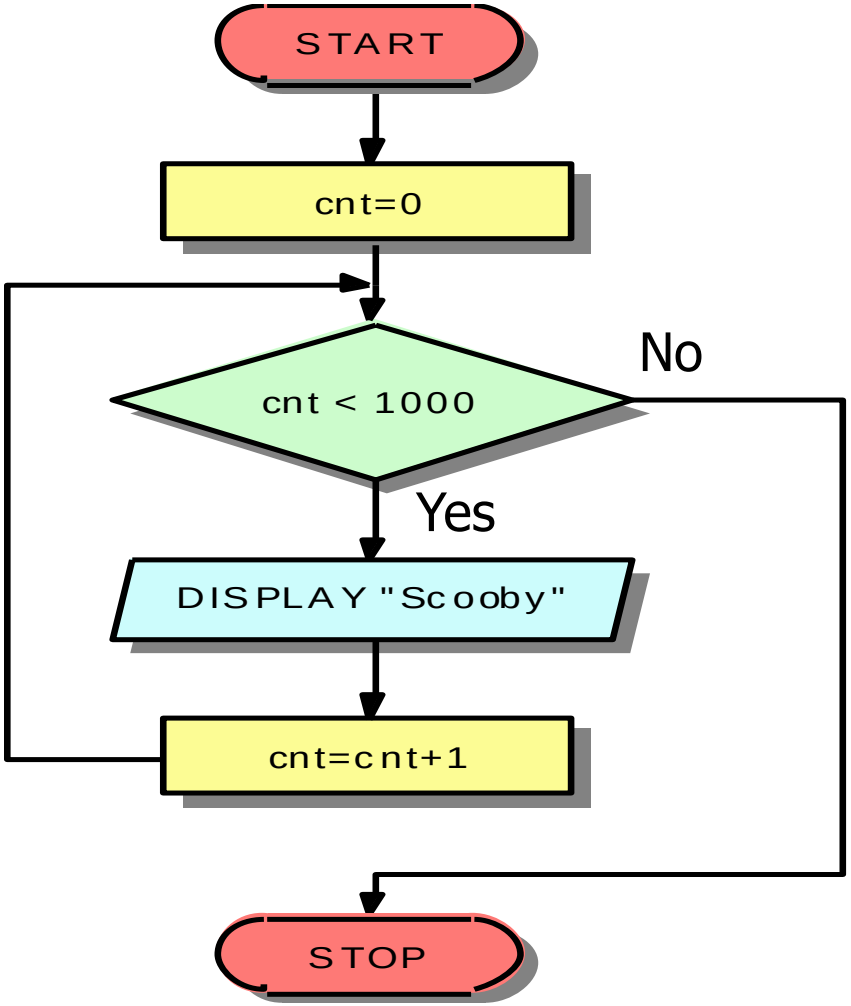
```
BEGIN
INPUT yearsWithUs
INPUT bizDone
IF yearsWithUs >= 10
        IF bizDone >=5000000
                DISPLAY “Classified as an MVS”
        ELSE
                DISPLAY “A little more effort required!”
END IF
ELSE
        DISPLAY “A little more effort required!”
END IF
END
```

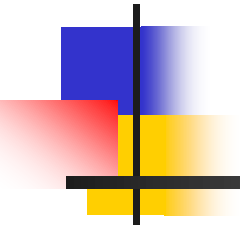
Cấu trúc IF lồng nhau (tt.)



Vòng lặp

BEGIN
cnt=0
WHILE (cnt < 1000)
DO
DISPLAY “Scooby”
cnt=cnt+1
END DO
END





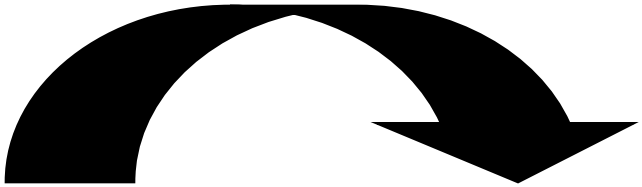
Biến và Kiểu Dữ Liệu

Chương 2

Mục Tiêu

- Hiểu được biến (variables)
- Phân biệt biến và hằng (constants)
- Liệt kê các kiểu dữ liệu khác nhau và sử dụng chúng trong chương trình C
- Hiểu và sử dụng các toán tử số học

Biến



Bộ nhớ

Dữ liệu **15**

	15		
	Dữ liệu trong bộ nhớ		

Mỗi vị trí trong bộ nhớ là duy nhất

Biến cho phép cung cấp một tên có ý nghĩa cho mỗi vị trí nhớ

Ví dụ

```
BEGIN
 DISPlAY ‘Enter 2  numbers’
INPUT A, B
C = A + B
DISPLAY  C
END
```

- A, B và C là các biến trong đoạn mã giả trên
- Tên biến giúp chúng ta truy cập vào bộ nhớ mà không cần dùng địa chỉ của chúng
- Hệ điều hành đảm nhiệm việc cấp bộ nhớ còn trống cho những biến này
- Để tham chiếu đến một giá trị cụ thể trong bộ nhớ, chúng ta chỉ cần dùng tên của biến

Hằng

- Một **hằng** (constant) là một giá trị không bao giờ thay đổi
- Các ví dụ
  - 5 **số / hằng số nguyên**
  - 5.3 **số / hằng số thực**
  - ‘Black’ **Hằng chuỗi**
  - ‘C’ **Hằng ký tự**
- **Biến lưu giữ các giá trị hằng**

Định danh

- Tên của các biến (variables), các hàm (functions), các nhãn (labels) và các đối tượng khác nhau do người dùng định nghĩa gọi là định danh
- Ví dụ về các định danh đúng
  - arena
  - s_count
  - marks40
  - class_one
- Ví dụ về các định danh sai
  - 1sttest
  - oh!god
  - start... end

Không hợp lệ !

- Các định danh có thể có bất cứ chiều dài nào theo quy ước, nhưng số ký tự trong một biến được nhận diện bởi trình biên dịch thì thay đổi theo trình biên dịch
- Các định danh trong C có phân biệt chữ hoa và chữ thường

Các nguyên tắc đặt tên định danh

Tên biến phải bắt đầu bằng một ký tự alphabet

Theo sau ký tự đầu có thể là các ký tự chữ, số …

Nên tránh đặt tên biến trùng tên các từ khoá

Tên biến nên mô tả được ý nghĩa của nó

Tránh dùng các ký tự gây lầm lẫn

Nên áp dụng các quy ước đặt tên biến chuẩn khi lập trình

Từ khóa

- Từ khóa: Tất cả các ngôn ngữ dành một số từ nhất định cho mục đích riêng
- Những từ này có một ý nghĩa đặc biệt trong ngữ cảnh của một ngôn ngữ cụ thể
- Sẽ không có xung đột nếu từ khóa và tên biến khác nhau. Ví dụ từ *integer* cho tên biến thì hoàn toàn hợp lệ ngay cả khi mà từ khóa là int

Kiểu dữ liệu

- Các kiểu dữ liệu khác nhau được lưu trữ trong biến là:
  - Số (Numbers)
    - Số nguyên.
      Ví dụ : 10 hay 178993455
    - Số thực.
      Ví dụ, 15.22 hay 15463452.25
    - Số dương
    - Số âm
  - Tên. Ví dụ : John
  - Giá trị luận lý :
    Ví dụ : Y hay N

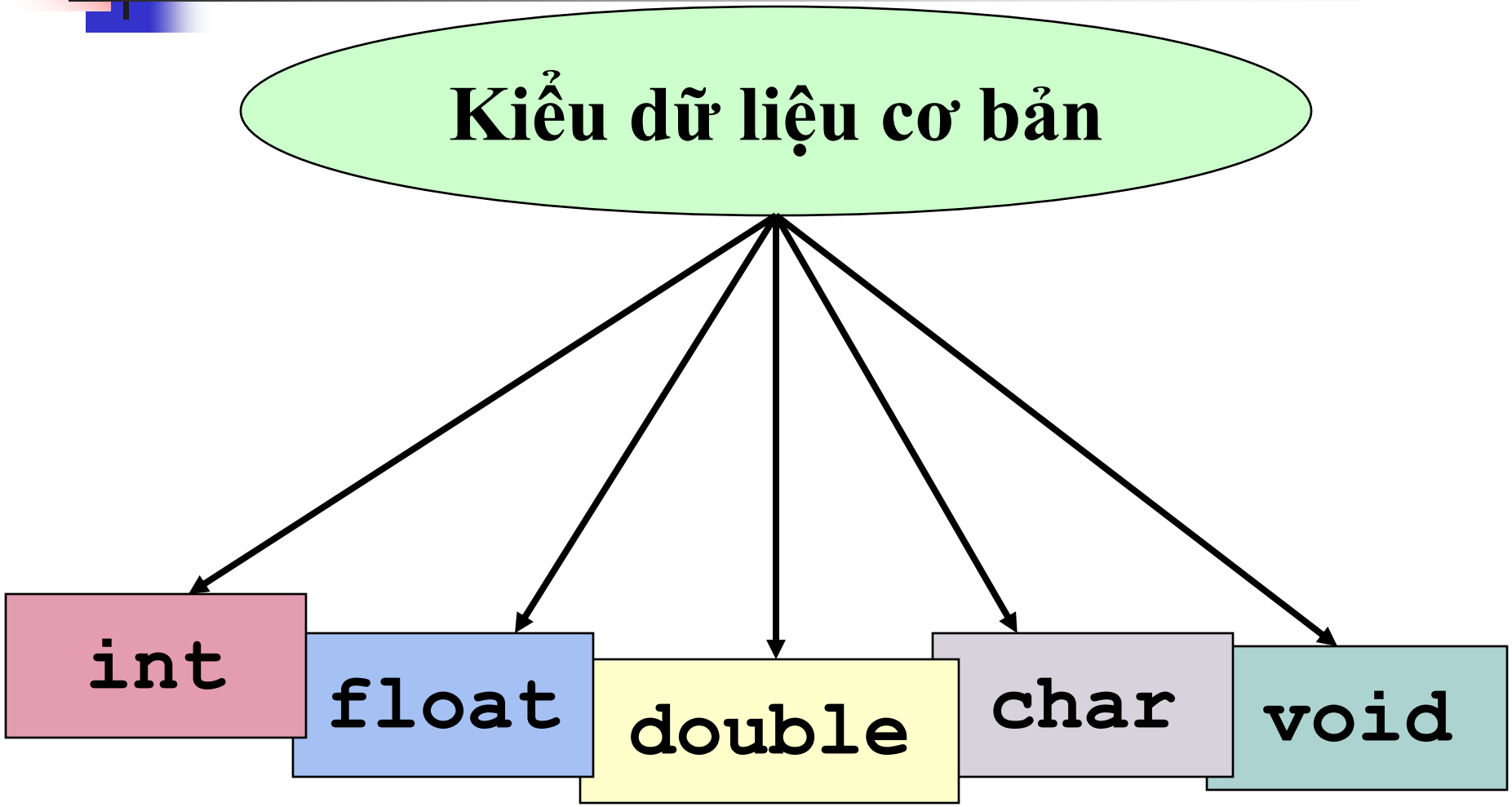
Kiểu dữ liệu (tt.)

- Kiểu dữ liệu mô tả loại dữ liệu sẽ được lưu trong biến
- Tên biến đặt sau kiểu dữ liệu
- Ví dụ : tên biến "varName" đứng sau kiểu dữ liệu "int"

kiểu dữ liệu tên biến

int varName

Kiểu dữ liệu cơ bản



Kiểu số nguyên (int)

- Lưu trữ dữ liệu số

  int num;
- Không thể lưu trữ bất cứ kiểu dữ liệu nào khác như “Alan” hoặc “abc”
- Chiếm 16 bits (2 bytes) bộ nhớ
- Biểu diễn các số nguyên trong phạm vi -32768 tới 32767
- Ví dụ : 12322, 0, -232

Kiểu số thực (float)

- Lưu trữ dữ liệu số chứa phần thập phân

  float num;
- Có độ chính xác tới 6 con số
- Chiếm 32 bits (4 bytes) bộ nhớ
- Ví dụ : 23.05, 56.5, 32

Kiểu số thực (double)

- Lưu trữ dữ liệu số chứa phần thập phân

  double num;

- Có độ chính xác tới 10 con số
- Chiếm 64 bits (8 bytes) bộ nhớ
- Ví dụ : 23.05, 56.5, 32

Kiểu ký tự (char)

- Lưu trữ một ký tự đơn

  char gender;

  gender='M';

- Chiếm 8 bits (1 byte) bộ nhớ
- Ví dụ: ‘a’, ‘m’, ‘$’ ‘%’ , ‘1’, ’5’

Kiểu void

- Không lưu bất cứ dữ liệu gì
- Báo cho trình biên dịch không có giá trị trả về

Những kiểu dữ liệu dẫn xuất

Bộ bổ từ (Modifiers) kiểu dữ liệu	+	Kiểu dữ liệu cơ bản	=	Kiểu dữ liệu dẫn xuất
unsigned	+	int	=	unsigned int (chỉ là số dương)
short	+	int	=	short int (chiếm ít bộ nhớ hơn int)
long	+	int/double	=	Long int /longdouble (chiếm nhiều bộ nhớ hơn int/double)

Các kiểu dữ liệu signed và unsigned

- Kiểu unsigned chỉ rõ rằng một biến chỉ có thể nhận giá trị dương

```
unsigned int varNum;
varNum=23123;
```

- varNum được cấp phát 2 bytes
- Bổ từ unsigned có thể được dùng với kiểu dữ liệu int và float
- Kiểu unsigned int hỗ trợ dữ liệu trong phạm vi từ 0 đến 65535

Những kiểu dữ liệu long (dài) và short (ngắn)

- **short int** chiếm giữ 8 bits (1 byte)
  - Cho phép số có phạm vi từ -128 tới 127
- **long int** chiếm giữ 32 bits (4 bytes)
  - -2,147,483,648 và 2,147,483,647
- **long double** chiếm 128 bits (16 bytes)

Kiểu dữ liệu & phạm vi giá trị

Kiểu	Dung lượng tính bằng bit	Phạm vi
char	8	-128 tới 127
Unsigned char	8	0 tới 255
signed char	8	-128 tới 127
int	16	-32,768 tới 32,767
unsigned int	16	0 tới 65,535
signed int	16	Giống như kiểu int
short int	16	Giống như kiểu int
unsigned short int	16	0 tới 65, 535

Kiểu dữ liệu & phạm vi giá trị (tt.)

Kiểu	Dung lượng tính bằng bit	Phạm vi
signed short int	16	Giống như kiểu short int
long int	32	-2,147,483,648 tới 2,147,483,647
signed long int	32	0 tới 4,294,967,295
unsigned long int	32	Giống như kiểu long int
float	32	6 con số thập phân
double	64	10 con số thập phân
long double	128	10 con số thập phân

Ví dụ về cách khai báo biến

```
main ()
{
   char abc;       /*abc of type character */
   int xyz;        /*xyz of type integer */
   float length;  /*length of type float */
   double area;  /*area of type double */
   long liteyrs;   /*liteyrs of type long int */
   short arm;     /*arm of type short integer*/
}
```

Chương 3

Mục Tiêu

- Hiểu được toán tử gán
- Hiểu được biểu thức số học
- Nắm được toán tử quan hệ và luận lý

(Relational and Logical Operators)

- Nắm được toán tử luận lý nhị phân và biểu thức

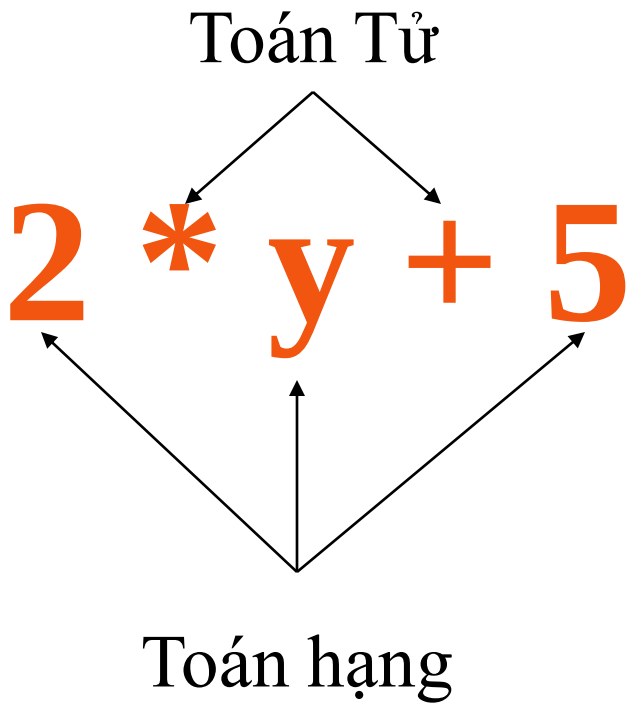
(Bitwise Logical Operators and Expression)

- Hiểu được khái niệm ép kiểu (Cast)
- Hiểu được độ ưu tiên của các toán tử

Biểu thức (Expressions)

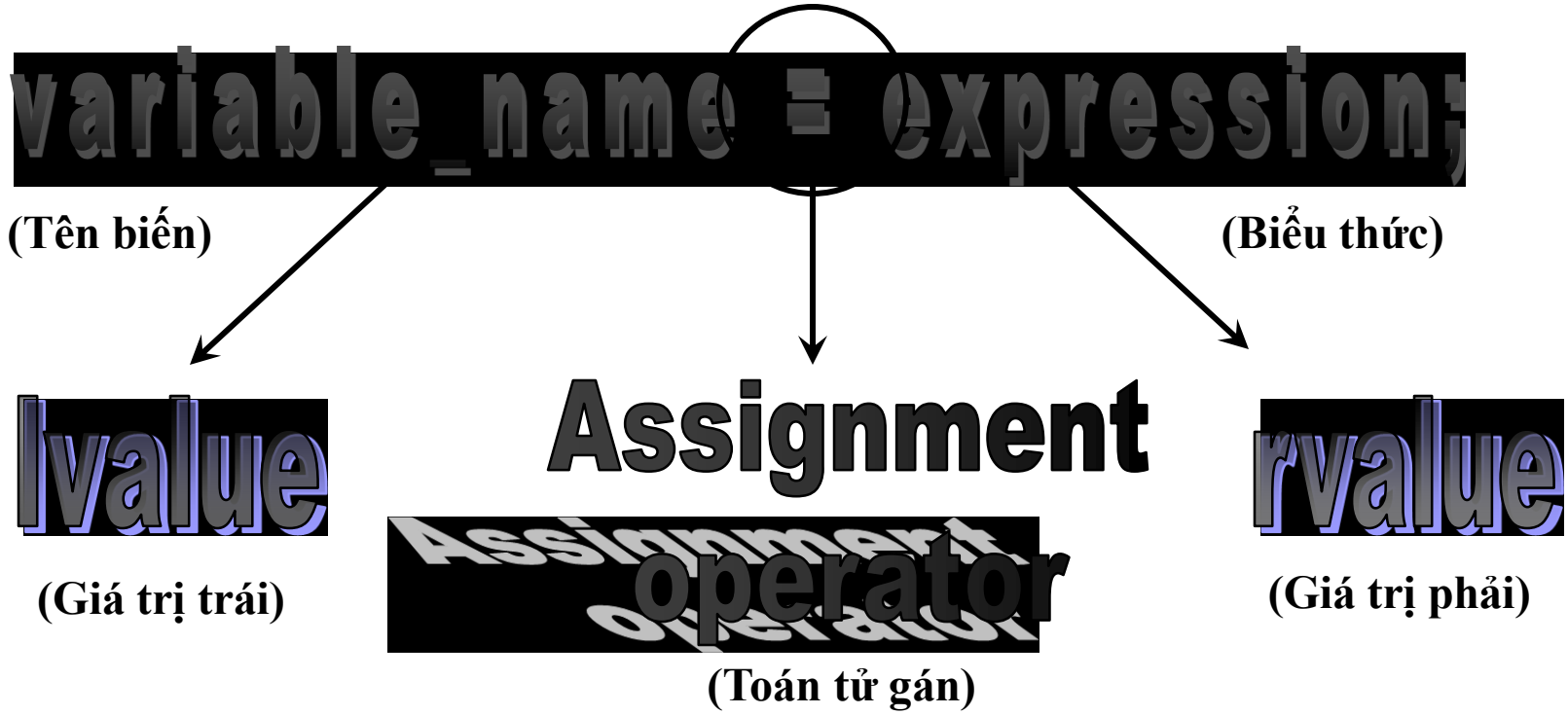
Sự kết hợp các toán tử và các toán hạng

Ví dụ:



Toán tử gán

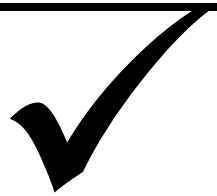
Toán tử gán (=) có thể được dùng với bất kỳ biểu thức C hợp lệ nào



Gán liên tiếp

Nhiều biến có thể được gán với cùng một giá trị trong một câu lệnh đơn

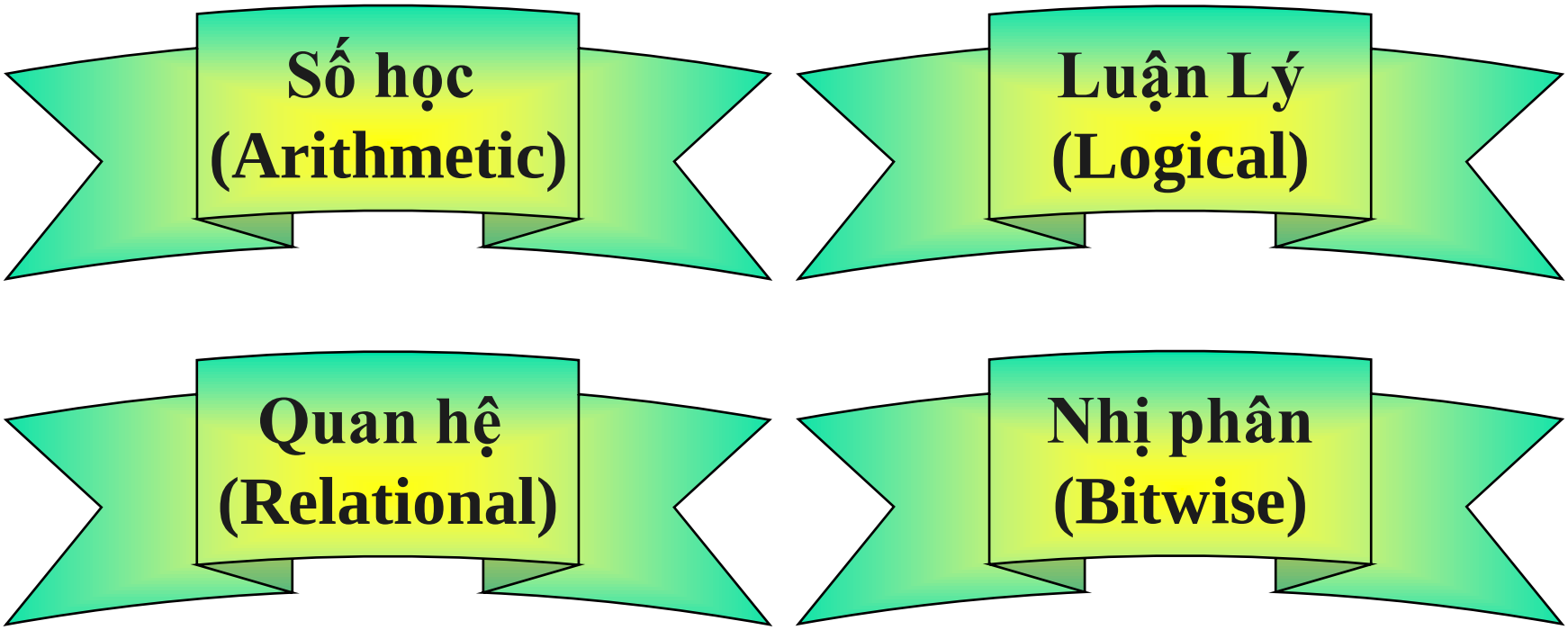
```
a = b = c = 10;
```



Tuy nhiên, không thể áp dụng quy tắc trên khi khai báo biến

```
int a = int b = int b = int c = 10
```

Bốn Kiểu Toán Tử



Biểu thức số học

Biểu thức số học có thể được biểu diễn trong C bằng cách sử dụng các toán tử số học

Ví dụ :

```
++i % 7

5 + (c = 3 + 8)

a * (b + c/d) - 22
```

Toán tử quan hệ và luận lý

Được dùng để :

Kiểm tra mối quan hệ giữa hai biến hay giữa một biến và một hằng

Toán tử quan hệ

Toán tử	Ý nghĩa
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Không bằng

Toán tử quan hệ và luận lý (tt.)

Toán tử luận lý là những ký hiệu dùng để kết hợp hay phủ định biểu thức chứa các toán tử quan hệ

Toán tử	**Ý nghĩa**
&&	**AND**: Kết quả là True khi cả 2 điều kiện đều đúng
||	**OR** : Kết quả là True khi chỉ một trong hai điều kiện là đúng
!	**NOT**: Tác động trên các giá trị riêng lẻ, chuyển đổi True thành False và ngược lại.

Ví dụ: if (a>10) && (a<20)

Những biểu thức dùng toán tử luận lý trả về 0 thay cho false và 1 thay cho true

Toán tử luận lý nhị phân

Dữ liệu chỉ được xử lý sau khi đã chuyển đổi giá trị SỐ thành giá trị NHỊ PHÂN

Toán tử	**Mô tả**
Bitwise AND (x & y)	Mỗi vị trí của bit trả về kết quả là 1 nếu bit của hai toán hạng là 1.
Bitwise OR (x | y)	Mỗi vị trí của bit trả về kết quả là 1 nếu bit của một trong hai toán hạng là 1.
Bitwise NOT (~ x)	Đảo ngược giá trị của toán hạng (1 thành 0 và ngược lại).
Bitwise XOR (x ^ y)	Mỗi vị trí của bit chỉ trả về kết quả là 1 nếu bit của một trong hai toán hạng là 1 mà không không phải cả hai toán hạng cùng là 1.

Toán tử luận lý nhị phân (tt.)

Ví dụ

- 10 & 15→ **1010 & 1111→1010** → 10
- 10 | 15→ **1010 | 1111→1111** → 15
- 10 ^ 15→ **1010 ^ 1111→0101** → 5
- ~ **10** → **~1010** →1…11110**101** → **-11**

Chuyển đổi kiểu

Qui tắc chuyển đổi kiểu tự động trình bày dưới đây nhằm xác định giá trị biểu thức:

a. char và short được chuyển thành int và float được chuyển thành double.

b. Nếu có một toán hạng là double, toán hạng còn lại sẽ được chuyển thành double, và kết quả là double.

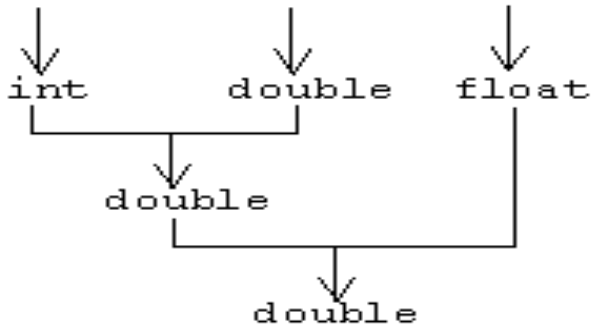
c. Nếu có một toán hạng là long, toán hạng còn lại sẽ được chuyển thành long, và kết quả là long.

d. Nếu có một toán hạng là unsigned, toán hạng còn lại sẽ được chuyển thành unsigned và kết quả cũng là unsigned.

e. Nếu tất cả toán hạng kiểu int, kết quả là int.

Ví dụ

```
char ch;
int i;
float f;
double d;
result = (ch/i) + (f*d) - (f+i);
```



Ép kiểu

Một biểu thức được ép thành một kiểu nhất định bằng cách dùng kỹ thuật ép kiểu (**cast).**

Cú pháp :

(kiểu dữ liệu) cast

Kiểu → Bất cứ kiểu dữ liệu hợp lệ trong C

Ví dụ:

```
float x,f;
f = 3.14159;
x = (int) f;
```

Giá trị số nguyên trả về bởi (int) f được chuyển thành số thực khi nó được toán tử GÁN xử lý. Song, giá trị của f vẫn không đổi.

Giá trị của x sẽ là 3 (số nguyên)

Độ ưu tiên của toán tử

- Độ ưu tiên tạo nên cấu trúc phân cấp của loại toán tử này so với loại toán tử khác khi tính giá trị một biểu thức số học
- Nó đề cập đến thứ tự thực thi các toán tử trong C
- Độ ưu tiên của các toán tử này được thay đổi bởi các dấu ngoặc đơn trong biểu thức

Loại toán tử	**Toán tử**	**Tính kết hợp**
Một ngôi	- ++ --	Phải đến trái
Hai ngôi	^	Trái đến phải
Hai ngôi	* / %	Trái đến phải
Hai ngôi	+ -	Trái đến phải
Hai ngôi	=	Phải đến trái

Độ ưu tiên của toán tử (tt.)

Ví dụ

-8 * 4 % 2 - 3

Trình tự	Thao tác	Kết quả
1.	- 8 (phép trừ một ngôi)	số âm của 8
2.	- 8 * 4	- 32
3.	- 32 % 2	16
4.	16-3	13

Độ ưu tiên của toán tử so sánh

Độ ưu tiên của toán tử so sánh (quan hệ) luôn được tính từ trái sang phải

Độ ưu tiên của toán tử luận lý

Thứ tự ưu tiên	Toán tử
1	NOT
2	AND
3	OR

Khi có nhiều toán tử luận lý trong một điều kiện, ta áp dụng quy tắc tính từ phải sang trái

Độ ưu tiên của toán tử luận lý (tt.)

Xét biểu thức sau:

False OR True AND NOT False AND True

Điều kiện này được tính như sau:

False OR True AND [NOT False] AND True

NOT có độ ưu tiên cao nhất.

False OR True AND [True AND True]

Ở đây, AND có độ ưu tiên cao nhất, những toán tử có cùng ưu tiên được tính từ phải sang trái.

False OR [True AND True]

[False OR True]

True

Độ ưu tiên giữa các toán tử

Khi một biểu thức có nhiều loại toán tử thì độ ưu tiên giữa chúng phải được thiết lập.

Thứ tự ưu tiên	Kiểu toán tử
1	Số học (Arithmetic)
2	So sánh (Comparison)
3	Luận lý (Logical)

Độ ưu tiên giữa các toán tử (tt.)

Ví dụ :

2*3+4/2 > 3 AND 3<5 OR 10<9

Việc tính toán như sau :

[2*3+4/2] > 3 AND 3<5 OR 10<9

Toán tử số học sẽ được tính trước

[[2*3]+[4/2]] > 3 AND 3<5 OR 10<9

[6+2] >3 AND 3<5 OR 10<9

[8 >3] AND [3<5] OR [10<9]

Độ ưu tiên giữa các toán tử (tt.)

Kế đến là toán tử so sánh có cùng độ ưu tiên. Ta áp dụng quy tắc tính từ trái sang phải.

True AND True OR False

Cuối cùng là toán tử kiểu luận lý. AND sẽ có độ ưu tiên cao hơn OR

[True AND True] OR False

True OR False

True

Thay đổi độ ưu tiên

- Dấu ngoặc đơn ( ) có độ ưu tiên cao nhất
- Độ ưu tiên của các toán tử có thể được thay đổi bởi dấu ngoặc đơn
- Toán tử có độ ưu tiên thấp hơn nếu đặt trong dấu ngoặc đơn sẽ được thực thi trước
- Khi các cặp ngoặc đơn lồng nhau **((( )))**, cặp ngoặc đơn trong cùng nhất sẽ được thực thi trước
- Nếu trong biểu thức có nhiều cặp ngoặc đơn thì việc thực thi sẽ theo thứ tự từ trái sang phải

Thay đổi độ ưu tiên (tt.)

Ví dụ :

5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR (2<6 AND 10>11))

Cách tính :

1) **5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR (True AND False))**

Dấu ngoặc đơn bên trong sẽ được tính trước

2) **5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR False)**

Thay đổi độ ưu tiên (tt.)

3) 5+9*3^2-4 >10 AND (2+16-8/4 > 6 OR False)

Kế đến dấu ngoặc đơn ở ngoài được tính đến

4) 5+9*3^2-4 > 10 AND (2+16-2 > 6 OR False)

5) 5+9*3^2-4 > 10 AND (18-2 > 6 OR False)

6) 5+9*3^2-4 > 10 AND (16 > 6 OR False)

7) 5+9*3^2-4 > 10 AND (True OR False)

8) 5+9*3^2-4 > 10 AND True

Thay đổi độ ưu tiên (tt.)

9) 5+9*9-4>10 AND True

Biểu thức bên trái được tính trước

10)5+81-4>10 AND True

11) 86-4>10 AND True

12) 82>10 AND True

13) True AND True

14) True

Nhập và Xuất trong C

Chương 4

Mục tiêu của bài học

- Tìm hiểu các hàm định dạng Nhập/Xuất
  scanf(), printf()

- Sử dụng các hàm Nhập/Xuất ký tự
  getchar(), putchar()

Nhập/Xuất chuẩn

- Thư viện chuẩn trong C cung cấp các hàm xử lý cho việc nhập và xuất.
- Thư viện chuẩn có các hàm I/O, dùng để quản lý việc nhập, xuất, các thao tác trên ký tự và chuỗi.
- Thiết bị nhập chuẩn thường là bàn phím.
- Thiết bị xuất chuẩn thường là màn hình (console).
- Nhập và xuất có thể được xử lý qua các tập tin thay vì từ các thiết bị chuẩn.

Tập tin Header <stdio.h>

- #include <stdio.h>
  - Đây là câu lệnh tiền xử lý
- stdio.h là tập tin header (header file)
- Chứa các macro sử dụng cho nhiều hàm nhập/xuất trong C
- Các macro trong stdio.h giúp các hàm printf(), scanf(), putchar(), getchar() thực thi

Nhập/Xuất được định dạng

- **printf( )** – Dùng cho xuất có định dạng
- **scanf( )** – Sử dụng để nhập có định dạng
- ***Các đặc tả định dạng*** - qui định dạng thức mà theo đó giá trị của biến được nhập vào và in ra

printf ( )

- Được dùng để hiển thị dữ liệu ra thiết bị xuất chuẩn như màn hình (console)

Cú pháp → printf ("control string", argument list);

- Danh sách đối số (argument list) chứa hằng, biến, biểu thức hoặc các hàm phân cách bởi dấu phẩy
- Phải có một lệnh định dạng trong "*control string*" cho mỗi đối số trong danh sách
- Các lệnh định dạng phải khớp với danh sách đối số về số lượng, kiểu và thứ tự.
- *control string* luôn được đặt trong dấu nhấy kép " ", đây là dấu phân cách

printf ( ) (tt.)

control string **chứa một trong ba kiểu phần tử sau:**

1. Các ký tự ***văn bản*** :
   gồm các ký tự có thể in được
2. Các ***lệnh định dạng*** :
   bắt đầu với ký hiệu % và theo sau là một mã định dạng tương ứng cho từng phần tử dữ liệu
3. Các ***ký tự không in được*** :
   gồm tab, blank và new_line

Mã định dạng

Định dạng	printf()	scanf()
Ký tự đơn (single character)	%c	%c
Chuỗi (string)	%s	%s
Số nguyên có dấu (signed decimal integer)	%d	%d
Kiểu float - dạng dấu chấm thập phân (decimal notation)	%f	%f hoặc %e
Kiểu float - dạng dấu chấm thập phân	%lf	%lf
Kiểu float - dạng lũy thừa (exponential notation)	%e	%f or %e
Kiểu float (%f hay %e , khi ngắn hơn)	%g	
Số nguyên không dấu (unsigned decimal integer)	%u	%u
Số nguyên hệ 16 không dấu - sử dụng "ABCDEF" (unsigned hexadecimal integer)	%x	%x
Số nguyên hệ 8 không dấu (unsigned octal integer)	%o	%o

Trong bảng trên : c, d, f, lf, e, g, u, s, o và x là các bộ đặc tả kiểu

Mã định dạng (tt.)

Mã định dạng	Các qui ước in
%d	Các con số trong số nguyên
%f	Các chữ số phần nguyên sẽ được in ra. Phần thập phân sẽ chỉ in 6 chữ số. Nếu phần thập phân ít hơn 6 chữ số, nó sẽ được thêm các chữ số 0 vào từ bên phải, ngược lại nó sẽ làm tròn số từ bên phải.
%e	Một con số bên trái của dấu chấm thập phân và 6 vị trí bên phải, như %f ở trên

Mã định dạng (tt.)

STT	Lệnh	Chuỗi điều khiển	Nội dung chuỗi điều khiển	Danh sách đối số	Giải thích danh sách đối số	Hiển thị trên màn hình
1.	printf("%d",300);	%d	Chỉ chứa lệnh định dạng	300	Hằng	300
2.	printf("%d",10+5);	%d	Chỉ chứa lệnh định dạng	10 + 5	Biểu thức	15
3.	printf("Good Morning Mr. Lee.");	Good Morning Mr. Lee.	Chỉ chứa các ký tự văn bản	Rỗng	Rỗng	Good Morning Mr. Lee.
4.	int count = 100; printf("%d",count);	%d	Chỉ chứa lệnh định dạng	count	Biến	100
5.	printf("\nhello");	\nhello	Chứa ký tự không được in và các ký tự văn bản	Rỗng	Rỗng	hello on a new line
6.	#define str "Good Apple " …….. printf("%s",str);	%s	Chỉ chứa lệnh định dạng	str	Hằng ký hiệu	Good Apple

7.	……..
	int count,stud_num;
	count=0;
	stud_num=100;
	printf("%d %d\n",count, stud_num);
		%d %d	Chứa lệnh định dạng và ký tự không được in	count, stud_num	Hai biến	0 , 100

Các ký tự đặc biệt

\\	In ra ký tự \
\"	In ra ký tự "
%%	In ra ký tự %

Ví dụ cho hàm printf()

Chương trình hiển thị số nguyên, thập phân, ký tự và chuỗi

```
#include <stdio.h>
void main()
{
        int a = 10;
        float b = 24.67892345;
        char ch = 'A';
        printf("Integer data = %d", a);
        printf("Float Data = %f",b);
        printf("Character = %c",ch);
        printf("This prints the string");
        printf("%s","This also prints a string");
}
```

Bổ từ trong hàm printf( )

1. Bổ từ ‘-‘

Phần tử dữ liệu sẽ được canh lề trái, phần tử sẽ được in bắt đầu từ vị trí bên trái trong cùng của trường.

2. Bổ từ xác định độ rộng trường

Có thể được sử dụng với kiểu float, double hoặc mảng ký tự (chuỗi). Độ rộng trường là một số nguyên xác định độ rộng nhỏ nhất cho phần tử dữ liệu.

Bổ từ trong hàm printf( ) (tt.)

3. Độ chính xác

Được sử dụng với kiểu float, double hoặc mảng ký tự (chuỗi). Nếu dùng với kiểu float hay double, chuỗi con số xác định số lượng lớn nhất các con số được in bên phải dấu chấm thập phân.

4. Bổ từ '0'

Mặc định thì khoảng trống sẽ được thêm vào một trường. Nếu người dùng muốn thêm số 0 vào trường thì bổ từ '0' được dùng

5. Bổ từ 'l'

Bổ từ này có thể được dùng hiển thị các đối số nguyên kiểu int hay double. Mã định dạng tương ứng là %ld

Bổ từ trong hàm printf( ) (tt.)

6. Bổ từ 'h'

Bổ từ này được sử dụng để hiển thị dạng short int. Mã định dạng tương ứng như là %hd

7. Bổ từ '*'

Nếu người dùng không muốn xác định độ rộng trường nhưng muốn chương trình xác định điều đó, bổ từ này được sử dụng

Ví dụ về các bổ từ

/* This program demonstrate the use of Modifiers in printf() */

```
#include <stdio.h>
void main(){
   printf("The number 555 in various forms:\n");
   printf("Without any modifier: \n");
   printf("[%d]\n",555);
   printf("With - modifier :\n");
   printf("[%-d]\n",555);
   printf("With digit string 10 as modifier :\n");
   printf("[%10d]\n",555);
   printf("With 0 as modifier : \n");
   printf("[%0d]\n",555);
   printf("With 0 and digit string 10 as modifiers :\n");
   printf("[%010d]\n",555);
   printf("With -,0 and digit string 10 as
   modifiers:\n");
   printf("[%-010d]\n",555);
}
```

scanf( )

- Được sử dụng để nhập dữ liệu
  Dạng tổng quát của hàm scanf()
  scanf("control string", argument list);

- Những định dạng dùng trong hàm printf() cũng được sử dụng với cùng cú pháp trong hàm scanf()

Sự khác nhau về danh sách đối số giữa printf( ) và scanf( )

- printf() sử dụng các tên biến, hằng, hằng biểu tượng và các biểu thức
- scanf() sử dụng các con trỏ tới biến

Danh sách đối số trong scanf() phải theo qui tắc :

- Đọc giá trị vào một biến có kiểu dữ liệu cơ sở, sử dụng ký hiệu **&** trước tên biến
- Đọc giá trị vào một biến có kiểu dữ liệu dẫn xuất, không sử dụng **&** trước tên biến

Sự khác nhau về các lệnh định dạng giữa printf( ) và scanf( )

- Không có tuỳ chọn %g
- Mã định dạng %f và %e là giống nhau

Ví dụ với hàm scanf( )

```
#include <stdio.h>
void main(){
   int a;
   float d;
   char ch, name[40];
   printf("Please enter the data\n");
   scanf("%d %f %c %s",&a,&d,&ch,name);
   printf("\n The values accepted are:
            %d,%f,%c,%s",a, d,ch,name);
}
```

Vùng đệm Nhập/Xuất

- Được sử dụng để đọc và viết các ký tự ASCII
- Một vùng đệm (buffer) là một không gian lưu trữ tạm thời trong bộ nhớ hoặc trên thẻ điều khiển thiết bị
- Bộ đệm Nhập/Xuất có thể chia làm :
  - Console I/O
  - Buffered File I/O

Console I/O

- Các hàm Console I/O chuyển các thao tác đến thiết bị xuất nhập chuẩn của hệ thống
- Trong ‘C’ các hàm console I/O đơn giản nhất là:
  - **getchar( ) -** đọc một và chỉ một ký tự từ bàn phím
  - **putchar( ) -** xuất một ký tự lên màn hình

getchar( )

- Dùng đọc dữ liệu nhập, một ký tự từ bàn phím
- Các ký tự đặt trong vùng đệm đến khi người dùng gõ phím enter
- Hàm getchar( ) không có đối số, nhưng vẫn phải có cặp dấu ngoặc ( )

Ví dụ hàm getchar()

```
/*Program to demonstrate the use of getchar()*/

#include <stdio.h>
void main()
{
  char letter;
  printf("\nPlease enter any character:");
  letter = getchar();
  printf("\nThe character entered by you
is %c", letter);
}
```

putchar( )

- Hàm xuất ký tự trong ‘C’
- Có một đối số

Đối số của một hàm putchar( ) có thể là :

- Một hằng ký tự đơn
- Một mã định dạng
- Một biến ký tự

Các tùy chọn và chức năng của putchar( )

Đối số	Hàm	Chức năng
Biến ký tự	putchar(c)	Hiển thị nội dung của biến ký tự c
Hằng ký tự	putchar('A')	Hiển thị ký tự A
Hằng số	putchar('5')	Hiển thị số 5
Mã định dạng	putchar('\t')	Xen một khoảng trống tại vị trí con trỏ
Mã định dạng	putchar('\n')	Xen một lệnh xuống dòng tại vị trí con trỏ

putchar( )

```
/* This program demonstrates the use of
   constants and escape sequences in
   putchar()*/
#include <stdio.h>
void main(){
   putchar('H'); putchar('\n');
   putchar('\t');
   putchar('E'); putchar('\n');
```

Ví dụ

```
   putchar('\t'); putchar('\t');
   putchar('L'); putchar('\n');
   putchar('\t'); putchar('\t'); putchar('\t');
   putchar('L'); putchar('\n');
   putchar('\t'); putchar('\t'); putchar('\t');
   putchar('\t');
   putchar('O');
}
```

Chương 5

Mục tiêu bài học

- Tìm hiểu về cấu trúc lựa chọn
  - Lệnh if
  - Lệnh if – else
  - Lệnh nhiều if
  - Lệnh if lồng nhau

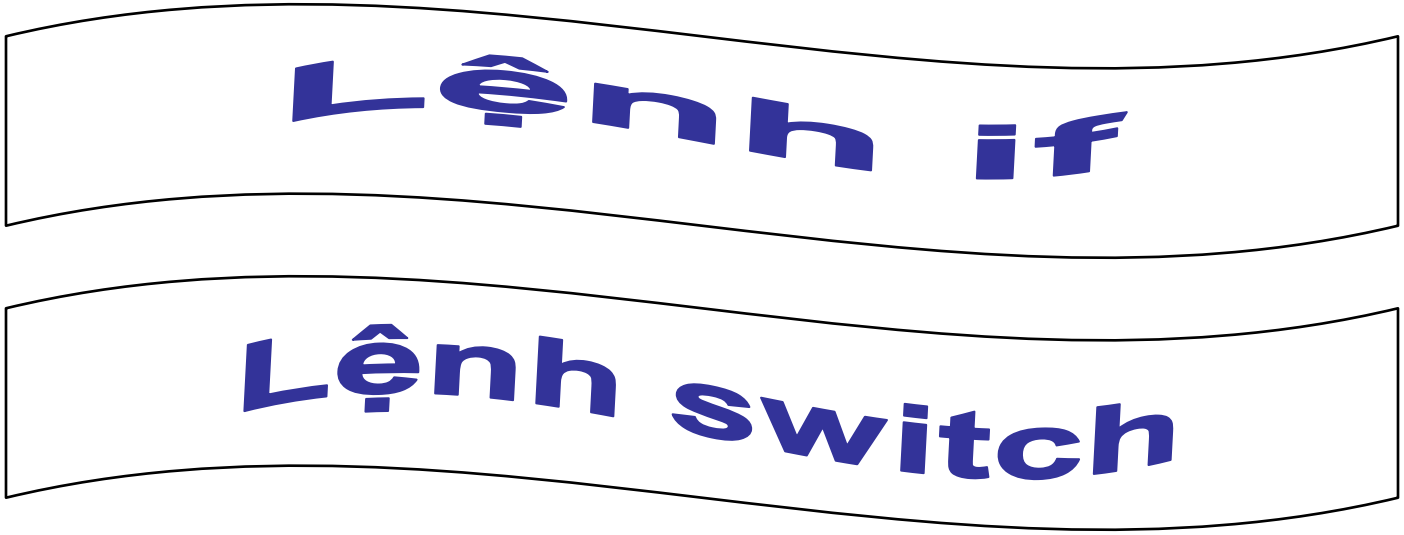
- Lệnh switch

Câu lệnh điều kiện

- Các câu lệnh điều kiện cho phép chúng ta thay đổi hướng thực hiện của chương trình
- Một câu lệnh điều kiện trả về giá trị đúng hoặc sai
- Ví dụ: Để xác định một số là số chẵn hay số lẻ chúng ta tiến hành như sau :
  1) Nhập vào một số
  2) Chia số đó cho 2 để xác định số dư
  3) Nếu số dư là 0, số đó là "SỐ CHẴN"
  4) Ngược lại số dư không bằng 0, số đó là "SỐ LẺ"

Các cấu trúc lựa chọn

C cung cấp hai dạng câu lệnh lựa chọn



Lệnh if

- Cú pháp:

```
if (expression)
    statement;
```

- Nếu biểu thức của lệnh if có giá trị đúng (true), khối lệnh theo sau lệnh if được thực thi

Lệnh if (tt.)

Chương trình hiển thị các giá trị dựa vào một điều kiện.

Ví dụ

```
#include <stdio.h>
void main(){
  int x, y;
  char a = 'y';
  x = y = 0;
  if (a == 'y'){
     x += 5;
     printf("The numbers are %d and
  \t%d",x,y);
  }
}
```

Lệnh if – else

```
if (expression)
        statement;
else
        statement;
```

Lệnh if – else (tt.)

- Nếu biểu thức của if trả về giá trị **true**, khối lệnh theo sau lệnh if được thực thi
- Nếu biểu thức của if **không** trả về giá trị true thì các lệnh theo sau else được thực thi
- Lệnh else là một tùy chọn. Lệnh else được dùng để thực thi các lệnh khi biểu thức trong if trả về giá trị **false**

Lệnh if – else (tt.)

Chương trình xác định một số là số chẵn hay lẻ

```
#include <stdio.h>
void main(){
  int num ,res ;
  printf("Enter a number :");
  scanf("%d",&num);
  res = num % 2;
  if (res == 0)
     printf("Then number is Even");
  else
     printf("The number is Odd");
}
```

Ví dụ

Lệnh if–else–if

Cú pháp:

```
if (expression)
    statement;
else if (expression)
    statement;
else if (expression)
    statement;
        …
else statement;
```

Lệnh if–else–if (tt.)

- Lệnh **if - else - if** còn được gọi là lệnh **if-else-if bậc thang**
- Các biểu thức được xác định giá trị theo hướng từ trên xuống

Lệnh if–else–if (tt.)

Chương trình hiển thị thông báo dựa vào một giá trị :

```
#include <stdio.h>
#include <conio.h>
   main() {
        int x;
        x = 0;
        clrscr ();
        printf("Enter Choice (1 - 3) : ");
        scanf("%d", &x);
        if (x == 1)
              printf ("\nChoice is 1");
        else if ( x == 2)
              printf ("\nChoice is 2");
        else if ( x == 3)
              printf ("\nChoice is 3");
        else printf ("\nInvalid Choice ");
   }
```

Ví dụ

Lệnh if lồng nhau

- Lệnh **if** lồng nhau là một lệnh **if** được đặt trong một lệnh **if** hoặc **else** khác
- Trong C, một lệnh **else** luôn kết hợp với một lệnh **if** gần nhất cùng khối lệnh nếu lệnh **else** đó chưa được kết hợp với một lệnh **if nào khác**

Lệnh if lồng nhau (tt.)

- Cú pháp:

```
if (expl)
{
     if (exp2) statement1;
     if (exp3) statement2;
     else statement3;              /*with if (exp3) */
}
else statement4;                  /* with if (expl) */
```

- Lệnh else bên trong được kết hợp với if(exp3)
- Theo chuẩn ANSI, một trình biên dịch hỗ trợ ít nhất là 15 mức lồng nhau

Lệnh if lồng nhau (tt.)

Ví dụ

```
#include <stdio.h>
#include <conio.h>
   void main (){
        int x, y;
        x = y = 0;
        clrscr ();
        printf ("Enter Choice (1 - 3) : ");
        scanf ("%d", &x);
        if (x == 1){
          printf("\nEnter value for y (1 - 5) : ");
          scanf ("%d", &y);
          if (y <= 5)
               printf("\nThe value for y is : %d", y);
               else
               printf("\nThe value of y exceeds 5 ");
        }
        else   printf ("\nChoice entered was not 1");
   }
```

Lệnh switch

- Lệnh **switch** là một bộ lựa chọn đa hướng, nó so sánh giá trị của một biểu thức với một danh sách các hằng số nguyên hoặc hằng ký tự

- Khi gặp một sự so sánh khớp, các lệnh kết hợp với hằng đó được thực thi

Lệnh switch (tt.)

- Cú pháp:

```
switch (expression)
{
        case constant1:
                statement sequence
                break;
        case constant2:
                statement sequence
                break;
        case constant3:
                statement sequence
                break;
        .
        .
        .
        default:
                statement sequence
}
```

Lệnh switch (tt.)

Chương trình kiểm tra một ký tự thường được nhập vào là một nguyên âm, là ký tự 'z' hay là một phụ âm

Ví dụ

```
#include <stdio.h>
#include <conio.h>
  main (){
      char ch;
      clrscr ();
      printf ("\nEnter a lower cased
           alphabet (a - z) : ");
      scanf("%c", &ch);
```

còn tiếp …

Lệnh switch (tt.)

```
if (ch < 'a' || ch > 'z')
   printf("\nCharacter not a lower cased alphabet");
else
   switch (ch) {
        case 'a' :
        case 'e' :
        case 'i' :
        case 'o' :
        case 'u' :
               printf("\nCharacter is a vowel");
               break;
        case 'z' :
               printf ("\nLast Alphabet (z) was entered");
               break;
        default :
               printf("\nCharacter is a consonant");
               break;
   }
}
```

Vòng lặp

Chương 6

Mục tiêu của bài học

- Tìm hiểu về vòng lặp ‘for’ trong C
- Làm việc với toán tử dấu phẩy (,)
- Tìm hiểu về các vòng lặp lồng nhau
- Tìm hiểu về vòng lặp ‘while’ và ‘do-while’
- Làm việc với các lệnh break và continue
- Tìm hiểu về hàm exit()

Vòng lặp là gì?

Một đoạn mã lệnh trong chương trình thực hiện lặp đi lặp lại cho đến khi một điều kiện xác định được thỏa mãn

3 kiểu cấu trúc vòng lặp

Vòng lặp for

Vòng lặp while

Vòng lặp do....while

Vòng lặp for

Cú pháp:

```
for (initialize counter; conditional test; re-evaluation
    parameter){
        statement
}
```

- *initialize counter* là một lệnh gán để khởi tạo biến điều khiển của vòng lặp trước khi đi vào vòng lặp
- *conditional test* là một biểu thức quan hệ để chỉ định khi nào vòng lặp sẽ kết thúc
- *re-evaluation parameter* định nghĩa cách thức thay đổi của biến điều khiển vòng lặp mỗi khi vòng lặp được thực thi

Vòng lặp for (tt.)

- Ba phần của vòng lặp **for** phải được phân cách bởi dấu chấm phẩy(;)
- Phần lệnh tạo nên thân vòng lặp có thể là một lệnh đơn hoặc một lệnh ghép (một tập nhiều lệnh)
- Vòng lặp **for** tiếp tục được thực thi khi biểu thức kiểm tra điều kiện vẫn có giá trị **true**. Khi điều kiện trở thành **false**, chương trình thực hiện lệnh theo sau vòng lặp **for**

Vòng lặp for - Ví dụ

```
/*This program demonstrates
the for loop in a C program*/
  #include <stdio.h>
  main() {
     int count;
     printf("\tThis is a \n");
     for(count = 1;count <=6;count++)
         printf("\n\t\t nice");
     printf("\n\t\t world. \n");
  }
```

Toán tử dấu phẩy

Vòng lặp **for** có thể được mở rộng bằng cách chứa nhiều giá trị khởi tạo và nhiều biểu thức tăng trị trong đặc tả của vòng lặp for

Cú pháp: **exprn1 , exprn2 ;**

```
#include <stdio.h>
  main()    {
      int i, j , max;
      printf("Please enter the maximum value \n");
      printf("for which a table can be printed:");
      scanf("%d", &max);
      for(i = 0 , j = max ; i <=max ; i++, j--)
            printf("\n%d + %d = %d",i, j, i + j);
  }
```

Vòng lặp for lồng nhau

Các vòng lặp for lồng nhau khi nó có dạng như sau

```
for (i = 1; i<max1; i++) {
      …
       for (j = 0; j < = max2; j++){
       …
       }
       …
}
```

Vòng lặp for lồng nhau - Ví dụ

```
#include <stdio.h>
  main()  {
     int i, j, k;
     i = 0;
     printf("Enter no. of rows :");
     scanf("%d", &i);
     printf("\n");
     for (j = 0; j < i ; j++){
       printf("\n");
       for (k = 0; k <= j; k++) /*inner for loop*/
       printf("*");
     }
  }
```

Vòng lặp while

Cú pháp

```
while (condition is true)
        statement ;
```

Vòng lặp while lặp lại các lệnh trong khi một biểu thức điều kiện mang giá trị True

Vòng lặp while -Ví dụ

```
/*A simple program using the while loop*/
  #include <stdio.h>
  main() {
      int count = 1;
      while( count <= 10)  {
        printf("\n This is iteration
                              %d\n",count);
        count++;
      }
      printf("\n The loop is completed.\n");
   }
```

Vòng lặp do…while

Cú pháp

```
do{
    statement;
} while (condition);
```

- Trong vòng lặp do while phần thân của vòng lặp được thực thi trước khi biểu thức điều kiện được kiểm tra
- Khi điều kiện mang giá trị False, vòng lặp **do while** sẽ được kết thúc, và điều khiển chuyển đến lệnh xuất hiện ngay sau lệnh while

Vòng lặp do...while - Ví dụ

```
#include <stdio.h>
  main (){
   int num1, num2;
   num2 = 0;
   do{
        printf( "\nEnter a number : ");
        scanf("%d",&num1);
        printf( " No. is %d",num1);
        num2++;
   } while (num1 != 0);
   printf ("\nThe total numbers entered were
              %d",--num2);
/*num2 is decremented before printing because
count for last integer (0) is not to be considered
*/
   }
```

Các lệnh chuyển điều khiển

return expression

- Lệnh return được sử dụng để trở về từ một hàm
- Thực hiện lệnh return để trở về vị trí mà tại đó hàm được gọi
- Lệnh return có thể có một giá trị đi cùng, giá trị này được trả về cho chương trình gọi

Các lệnh chuyển điều khiển (tt.)

goto label

- Lệnh goto chuyển điều khiển đến một câu lệnh bất kỳ khác bên trong cùng một hàm trong một chương trình C
- Điều này thật ra vi phạm đến qui luật của một ngôn ngữ lập trình cấu trúc.
- Chúng làm giảm độ tin cậy của chương trình và chương trình khó bảo trì.

Các lệnh chuyển điều khiển (tt.)

statement

- Lệnh break được sử dụng để kết thúc một mệnh đề ***case*** trong câu lệnh ***switch***
- Nó cũng có thể được sử dụng để kết thúc ngang giữa vòng lặp
- Khi gặp lệnh break, vòng lặp sẽ kết thúc ngay và điều khiển được chuyển đến lệnh kế tiếp bên ngoài vòng lặp

Lệnh break – Ví dụ

```
#include <stdio.h>
  main (){
    int count1, count2;
    for(count1 = 1, count2 = 0;
      count1 <=100; count1++){
      printf("Enter %d count2: ",
          count1);
      scanf("%d", &count2);
      if(j==100) break;
    }
  }
```

Các lệnh chuyển điều khiển (tt.)

continue statement

- Lệnh ***continue*** dùng để bắt đầu thực hiện lần lặp kế tiếp của vòng lặp
- Khi gặp lệnh ***continue***, các câu lệnh còn lại trong thân vòng lặp bị bỏ qua và điều khiển được chuyển đến lần lặp kế tiếp

Lệnh continue – Ví dụ

```
#include <stdio.h>
  main ()     {
    int num;
    for(num = 1;num<=100; num++){
      if(num % 9 == 0)
          continue;
      printf("%d\t",num);
    }
  }
```

Các lệnh chuyển điều khiển (tt)

hàm exit()

- Hàm exit() được sử dụng để thoát khỏi chương trình
- Sử dụng hàm này sẽ kết thúc ngay chương trình và điều khiển được chuyển về cho hệ điều hành

Bài 7

Mục tiêu của bài học

- Các phần tử của mảng và các chỉ số
- Khai báo mảng
- Cách quản lý mảng trong C
- Cách khởi tạo mảng
- Tìm hiểu chuỗi / mảng ký tự
- Tìm hiểu mảng hai chiều
- Cách khởi tạo mảng hai chiều

Các phần tử và chỉ số của mảng

- Mỗi phần tử được xác định bằng một số thứ tự (còn gọi là chỉ số) duy nhất trong mảng
- Số chiều của mảng được xác định bằng số các chỉ số cần thiết để định danh duy nhất từng phần tử
- Chỉ số là một số nguyên dương trong [ ] đặt ngay sau tên mảng
- Chỉ số của mảng (trong C) được bắt đầu là 0
- Mảng ***player*** với 11 phần tử :

player[0], player[1], player[2],…. player[10]

Khai báo mảng

- Các đặc tính riêng của mảng cần được định nghĩa.

  Lớp lưu trữ

  Kiểu dữ liệu của các phần tử

  Tên mảng đại diện cho vị trí phần tử đầu tiên

  Kích thước mảng một hằng số

Khai báo mảng (tt.)

Khai báo mảng giống như cách khai báo biến. Chỉ khác là tên mảng được theo sau bởi một hoặc nhiều biểu thức đặt trong cặp dấu ngoặc vuông [], để xác định kích thước của mảng.

```
int player[11];
```

Các qui tắc

- Các phần tử của mảng có cùng kiểu dữ liệu
- Mỗi phần tử của mảng có thể được sử dụng như một biến riêng lẻ
- Kiểu dữ liệu của mảng có thể là int, char, float hoặc double

Quản lý mảng trong C

- Trong ngôn ngữ C, mảng được "đối xử" không giống hoàn toàn với biến
- Hai mảng có cùng kiểu và cùng kích thước cũng không được xem là tương đương nhau
- Không thể gán trực tiếp một mảng cho một mảng khác.
- Không thể gán trị cho toàn bộ mảng, mà phải gán trị cho từng phần tử của mảng

Quản lý mảng trong C (tt)

```
/*Input values are accepted from the user
into the array ary[10]*/

#include <stdio.h>
void main(){
    int ary[10];
    int i, total, high;
    for(i=0; i<10; i++)    {
      printf("\n Enter value: %d : ", i+1);
      scanf("%d",&ary[i]);
    }
```

Quản lý mảng trong C (tt)

```
/* Displays highest of the entered values */
     high = ary[0];
     for(i=1; i<10; i++){
       if(ary[i] > high) high = ary[i];
     }
     printf("\nHighest value entered was %d", high);

/*prints average of values entered for ary[10] */
     for(i=0,total=0; i<10; i++) total = total +
ary[i];
     printf("\nThe average of the elements of ary
is%d",total/i);
}
```

Khởi tạo mảng

- Mỗi phần tử của một mảng auto cần được khởi tạo riêng rẽ.
- Trong ví dụ sau các phần tử của mảng được gán giá trị bằng cách sử dụng vòng lặp for

```
#include <stdio.h>
void main()  {
   char alpha[26];
   int i, j;
   for(i=65,j=0; i<91; i++,j++) {
    alpha[j] = i;
    printf("The character now assigned
  is%c\n",alpha[j]);
    }
   getchar();
}
```

Khởi tạo mảng (tt)

- Trong trường hợp mảng extern và static, các phần tử được tự động khởi tạo với giá trị 0

Chuỗi/Mảng ký tự

- Chuỗi có thể được định nghĩa như là một mảng kiểu ký tự, được kết thúc bằng ký tự null
- Mỗi ký tự trong chuỗi chiếm một byte và ký tự cuối cùng của chuỗi là “\0” (null)
- Ví dụ:

Chuỗi/Mảng ký tự (ví dụ)

```
#include <stdio.h>
void main(){
    char ary[5];
    int i;
    printf("\n Enter string : ");
    scanf("%s",ary);
    printf("\n The string is %s \n\n",ary);
    for (i=0; i<5; i++)
   printf("\t%d", ary[i]);
  }
```

Chuỗi/Mảng ký tự (tt)

Chạy chương trình:

Enter string:

Nếu dữ liệu nhập là “appl”, output của chương trình là:

The string is appl

97 112 112 108 0

Các hàm xử lý chuỗi

- Các hàm xử lý chuỗi được tìm thấy trong thư viện chuẩn <string.h>

Name	Function
strcpy(s1, s2)	Copies s2 into s1
strcat(s1, s2)	Concatenates s2 onto the end of s1
strlen(s1)	Returns the length of s1
strcmp(s1, s2)	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1> s2
strchr(s1, ch)	Returns a pointer to the first occurrence of ch in s1
strstr(s1, s2)	Returns a pointer to the first occurrence of s2 in s1

Mảng hai chiều

- Mảng đa chiều đơn giản nhất và thường được dùng nhất là mảng hai chiều
- Mảng hai chiều có thể xem như là một mảng với mỗi phần tử là mảng một chiều
- Về logic, một mảng hai chiều trông giống như một bảng lịch trình xe lửa, gồm các dòng và các cột
- Khai báo mảng hai chiều:

int temp[4][3];

Khởi tạo mảng đa chiều

```
int ary[3][4]
  ={1,2,3,4,5,6,7,8,9,10,11,12};
```

Kết quả của phép gán trên như sau:

ary [0] [0] = 1	ary [0] [1] = 2	ary [0] [2] = 3	ary [0] [3] = 4
ary[1] [0] = 5	ary [1][1] = 6	ary [1] [2] = 7	ary [1][3] = 8
ary[2] [0] = 9	ary [2][1] = 10	ary [2] [2] = 11	ary [2][3] = 12

Khởi tạo mảng đa chiều (tt)

```
int ary[3][4]
  ={{1,2,3},{4,5,6},{7,8,3}};
```

Kết quả của phép gán trên như sau:

ary[0][0] =1	ary[0][1]=2	ary[0][2]=3	ary[0][3]=0
ary[1][0]=4	ary[1][1]=5	ary[1][2]=6	ary[1][3]=0
ary[2][0]=7	ary[2][1]=8	ary[2][2]=3	ary[2][3]=0

Khởi tạo mảng đa chiều (tt)

Một mảng chuỗi hai chiều được khai báo theo cách sau:

```
char str_ary[25][80];
```

Mảng hai chiều - Ví dụ

```
#include <stdio.h>
#include <string.h>
void main (){
   int i, n = 0;
   int item;
   char x[10][12];
   char temp[12];
   clrscr();
   printf("Enter each string on a separate line\n\n");
   printf("Type 'END' when over \n\n");
      /* read in the list of strings */
   do{
      printf("String %d : ", n+1);
      scanf("%s", x[n]);
   } while (strcmp(x[n++], "END"));
/*reorder the list of strings */
```

còn tiếp....

Mảng hai chiều - Ví dụ (tt.)

```
n = n - 1;
   for(item=0; item<n-1; ++item) {
      /* find lowest of remaining strings */
      for(i=item+1; i<n; ++i) {
            if(strcmp (x[item], x[i]) > 0){
                  /*interchange two stings */
                  strcpy (temp, x[item]);
                  strcpy (x[item], x[i]);
                  strcpy (x[i], temp);
            }
       }}
/* Display the arranged list of strings */
printf("Recorded list of strings : \n");
     for(i = 0; i < n ; ++i)  {
      printf("\nString %d is %s", i+1, x[i]);
     }
}
```

Bài 8

Mục tiêu bài học

- Tìm hiểu về con trỏ và khi nào thì sử dụng con trỏ
- Cách sử dụng biến con trỏ và các toán tử con trỏ
- Gán giá trị cho con trỏ
- Phép toán trên con trỏ
- So sánh con trỏ
- Con trỏ và mảng một chiều
- Con trỏ và mảng nhiều chiều
- Tìm hiểu cách cấp phát bộ nhớ

Con trỏ là gì?

- Con trỏ là một biến, nó chứa địa chỉ ô nhớ của một biến khác
- Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là con trỏ *trỏ đến* biến thứ hai
- Con trỏ cung cấp phương thức truy xuất gián tiếp đến giá trị của một phần tử dữ liệu
- Các con trỏ có thể trỏ đến các biến có kiểu dữ liệu cơ bản như int, char, double, hay dữ liệu tập hợp như mảng hoặc cấu trúc.

Con trỏ được sử dụng để làm gì?

Các tình huống con trỏ có thể được sử dụng:

- Để trả về nhiều hơn một giá trị từ một hàm
- Để truyền mảng và chuỗi từ một hàm đến một hàm khác thuận tiện hơn
- Để làm việc với các phần tử của mảng thay vì truy xuất trực tiếp vào các phần tử này
- Để cấp phát bộ nhớ và truy xuất bộ nhớ (Cấp phát bộ nhớ trực tiếp)

Biến con trỏ

- Khai báo con trỏ: chỉ ra một kiểu cơ sở và một tên biến được đặt trước bởi dấu *

Cú pháp khai báo tổng quát:

```
type *name;
```

Ví dụ:

```
int *var2;
```

Các toán tử con trỏ

- Hai toán tử đặc biệt được sử dụng với con trỏ:

& và *****

- & là toán tử một ngôi và nó trả về địa chỉ ô nhớ của toán hạng

```
var2 = &var1;
```

- Toán tử * là phần bổ xung của toán tử &. Đây là toán tử một ngôi và nó trả về giá trị chứa trong vùng nhớ được trỏ đến bởi biến con trỏ

```
temp = *var2;
```

Gán trị đối với con trỏ

- Các giá trị có thể được gán cho con trỏ thông qua toán tử **&**.

ptr_var = &var;

- Ở đây địa chỉ của var được lưu vào biến ptr_var.
- Cũng có thể gán giá trị cho con trỏ thông qua một biến con trỏ khác trỏ có cùng kiểu.

ptr_var = &var;

ptr_var2 = ptr_var;

Gán trị đối với con trỏ (tt)

- Có thể gán giá trị cho các biến thông qua con trỏ

***ptr_var = 10;**

- Câu lệnh trên gán giá trị 10 cho biến var nếu ptr_var đang trỏ đến var

Phép toán con trỏ

- Chỉ có thể thực hiện phép toán cộng và trừ trên con trỏ

```
int var, * ptr_var;
ptr_var = & var;
var = 500;
ptr_var ++;
```

- Giả sử biến **var** được lưu trữ tại địa chỉ **1000**
- ptr_var lưu giá trị 1000. Vì số nguyên có kích thước là 2 bytes, nên sau biểu thức "ptr_var++;" ptr_var sẽ có giá trị là 1002 mà không là 1001

Phép toán con trỏ (tt)

++ptr_var or ptr_var++	Trỏ đến **số nguyên** kế tiếp đứng sau var
--ptr_var or ptr_var--	Trỏ đến **số nguyên** đứng trước var
ptr_var + i	Trỏ đến số nguyên thứ i sau var
ptr_var - i	Trỏ đến số nguyên thứ i trước var
++*ptr_var or (*ptr_var)++	Sẽ tăng trị **var** bởi 1
*ptr_var++	Sẽ tác động đến giá trị của số nguyên kế tiếp sau var

Phép toán con trỏ (tt)

- Mỗi lần con trỏ được tăng trị, nó trỏ đến ô nhớ của phần tử kế tiếp
- Mỗi lần con trỏ được giảm trị, nó trỏ đến ô nhớ của phần tử đứng trước nó
- Tất cả con trỏ sẽ tăng hoặc giảm trị theo kích thước của kiểu dữ liệu mà chúng đang trỏ đến

So sánh con trỏ

- Hai con trỏ có thể được so sánh trong một biểu thức quan hệ nếu chúng trỏ đến các biến có cùng kiểu dữ liệu
- Giả sử ptr_a và ptr_b là hai biến con trỏ trỏ đến các phần tử dữ liệu a và b. Trong trường hợp này, các phép so sánh sau là có thể:

So sánh con trỏ (tt)

ptr_a < ptr_b	Trả về giá trị true nếu **a** được lưu trữ ở vị trí trước **b**
ptr_a > ptr_b	Trả về giá trị true nếu **a** được lưu trữ ở vị trí sau **b**
ptr_a <= ptr_b	Trả về giá trị true nếu **a** được lưu trữ ở vị trí trước **b** hoặc ptr_a và ptr_b trỏ đến cùng một vị trí
ptr_a >= ptr_b	Trả về giá trị true nếu **a** được lưu trữ ở vị trí sau **b** hoặc ptr_a và ptr_b trỏ đến cùng một vị trí
ptr_a == ptr_b	Trả về giá trị true nếu cả hai con trỏ ptr_a và ptr_b trỏ đến cùng một phần tử dữ liệu.
ptr_a != ptr_b	Trả về giá trị true nếu cả hai con trỏ ptr_a và ptr_b trỏ đến các phần tử dữ liệu khác nhau nhưng có cùng kiểu dữ liệu.
ptr_a == NULL	Trả về giá trị true nếu ptr_a được gán giá trị NULL (0)

Con trỏ và mảng một chiều

- Địa chỉ của một phần tử mảng có thể được biểu diễn theo hai cách:
  - Sử dụng ký hiệu & trước một phần tử mảng.
  - Sử dụng một biểu thức trong đó chỉ số của phần tử được cộng vào tên của mảng.

Con trỏ và mảng một chiều-Ví dụ

```
#include<stdio.h>
void main(){
      static int ary[10]
                  ={1,2,3,4,5,6,7,8,9,10};
      int i;
      for (i= 0;i<10;i++){
      printf("\ni=%d,aryi]=%d,*(ary+i)=%d",
                  i,ary[i],*(ary + i));
      printf("&ary[i]=%X,ary+i=%X",&ary[i],
                  ary+i);
      /*%X gives unsigned hexadecimal*/
      }
}
```

Con trỏ và mảng một chiều-ví dụ tt

```
i=0   ary[i]=1   *(ary+i)=1   &ary[i]=194   ary+i = 194
i=1   ary[i]=2   *(ary+i)=2   &ary[i]=196   ary+i = 196
i=2   ary[i]=3   *(ary+i)=3   &ary[i]=198   ary+i = 198
i=3   ary[i]=4   *(ary+i)=4   &ary[i]=19A   ary+i = 19A
i=4   ary[i]=5   *(ary+i)=5   &ary[i]=19C   ary+i = 19C
i=5   ary[i]=6   *(ary+i)=6   &ary[i]=19E   ary+i = 19E
i=6   ary[i]=7   *(ary+i)=7   &ary[i]=1A0   ary+i = 1A0
i=7   ary[i]=8   *(ary+i)=8   &ary[i]=1A2   ary+i = 1A2
i=8   ary[i]=9   *(ary+i)=9   &ary[i]=1A4   ary+i = 1A4
i=9   ary[i]=10  *(ary+i)=10  &ary[i]=1A6   ary+i = 1A6
```

Con trỏ và mảng đa chiều

- Mảng hai chiều có thể được định nghĩa như là một con trỏ trỏ tới một nhóm các mảng một chiều liên tiếp nhau
- Khai báo một mảng hai chiều có thể như sau:

```
data_type  (*ptr_var) [expr 2];
```

thay vì

```
data_type (*ptr_var) [expr1] [expr 2];
```

Con trỏ và chuỗi

```
#include <stdio.h>
#include <string.h>
void main (){
      char a, str[81], *ptr;
      printf("\nEnter a sentence:");
      gets(str);
      printf("\nEnter character to search for:");
      a = getche();
      ptr = strchr(str,a);
      /* return pointer to char*/
      printf( "\nString starts at address: %u",str);
      printf("\nFirst occurrence of the character is
at address: %u ",ptr);
      printf("\n Position of first occurrence(starting
from  0)is: % d", ptr_str);
}
```

Con trỏ và chuỗi (tt)

```
Enter a sentence: We all live in a yellow submarine
Enter character to search for: Y
String starts at address: 65420.
First occurrence of the character is at address: 65437.
Position of first occurrence (starting from 0) is: 17
```

Cấp phát bộ nhớ

Hàm **malloc()** là một trong các hàm được sử dụng thường xuyên nhất để thực hiện việc cấp phát bộ nhớ từ vùng nhớ còn tự do.

Tham số của hàm **malloc()** là một số nguyên xác định số bytes cần cấp phát.

Cấp phát bộ nhớ (tt)

```
#include<stdio.h>
#include<malloc.h>
void main()
{
int *p,n,i,j,temp;
printf("\n Enter number of elements in the array :");
scanf("%d",&n);
p=(int*)malloc(n*sizeof(int));
for(i=0;i<n;++i) {
 printf("\nEnter element no. %d:",i+1);
 scanf("%d",p+i); }
for(i=0;i<n-1;++i)
  for(j=i+1;j<n;++j)
    if(*(p+i)>*(p+j)) {
        temp=*(p+i);
       *(p+i)=*(p+j);
       *(p+j)=temp;  }
for(i=0;i<n;++i)
       printf("%d\n",*(p+i));
 }
```

Hàm free()

Hàm **free()** được sử dụng để giải phóng bộ nhớ khi nó không cần dùng nữa.

Cú pháp:

```
void free(void*ptr);
```

Hàm này giải phóng không gian được trỏ bởi *ptr*, để dùng cho tương lai.

ptr phải được dùng trước đó với lời gọi hàm malloc(), calloc(), hoặc realloc().

Hàm free() - tt

```
#include <stdio.h>
#include <stdlib.h>
/*required for the malloc and free functions*/
int main(){
  int number;
  int *ptr;
  int i;
  printf("How many ints would you like store? ");
  scanf("%d", &number);
  ptr = (int *) malloc (number*sizeof(int));
   /*allocate memory */
   if(ptr!=NULL) {
    for(i=0 ; i<number ; i++){
      *(ptr+i) = i;
     }
```

Còn tiếp…

Hàm free() - tt

```
        for(i=number ; i>0 ; i--) {
              printf("%d\n",*(ptr+(i-1)));
              /* print out in reverse order */
        }
        free(ptr); /* free allocated memory */
        return 0;
}
else {
     printf("\nMemory allocation failed -
       not enough memory.\n");
     return 1;
   }
}
```

Hàm calloc()

calloc tương tự như **malloc**, nhưng điểm khác biệt chính là mặc nhiên giá trị 0 được lưu vào không gian bộ nhớ vừa cấp phát

calloc yêu cầu hai tham số

- Tham số thứ nhất là số lượng các biến cần cấp phát bộ nhớ
- Tham số thứ hai là kích thước của mỗi biến

Cú pháp:

```
void *calloc( size_t num, size_t size );
```

Hàm calloc() - tt

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  float *calloc1, *calloc2;
  int i;
  calloc1 = (float *) calloc(3,
sizeof(float));
  calloc2 = (float *)calloc(3, sizeof(float));
  if(calloc1!=NULL && calloc2!=NULL){
    for(i=0 ; i<3 ; i++){
      printf("calloc1[%d] holds %05.5f ",i,
                  calloc1[i]);
     printf("\ncalloc2[%d] holds %05.5f",
                 i,*(calloc2+i));
    }
```

Còn tiếp……

Hàm calloc() - tt

```
    free(calloc1);
    free(calloc2);
    return 0;
}
else{
      printf("Not enough memory\n");
      return 1;
    }
}
```

Hàm realloc()

Có thể cấp phát lại cho một vùng đã được cấp (thêm/bớt số bytes) bằng cách sử dụng hàm **realloc**, mà không làm mất dữ liệu.

realloc nhận hai tham số

- Tham số thứ nhất là con trỏ tham chiếu đến bộ nhớ
- Tham số thứ hai là tổng số byte muốn cấp phát
- Cú pháp:

```
void *realloc( void *ptr, size_t size );
```

Hàm realloc() - tt

```
#include<stdio.h>
#include <stdlib.h>
int main(){
  int *ptr;
  int i;
  ptr = (int *)calloc(5, sizeof(int *));
   if(ptr!=NULL) {
     *ptr = 1;  *(ptr+1) = 2;
     ptr[2] = 4;  ptr[3] = 8;  ptr[4] = 16;
     ptr = (int *)realloc(ptr, 7*sizeof(int));
     if(ptr!=NULL){
       printf("Now allocating more memory...\n");
       ptr[5] = 32; /* now it's legal! */
       ptr[6] = 64;
```

Hàm realloc() - tt

```
        for(i=0;i<7;i++) {
          printf("ptr[%d] holds %d\n", i, ptr[i]);
        }
        realloc(ptr,0);
        /* same as free(ptr); - just fancier! */
        return 0;
      }
      else {
      printf("Not enough memory-realloc failed.\n");
      return 1;
      }
   }
else {
      printf("Not enough memory-calloc failed.\n");
      return 1;
   }
}
```

Hàm

Bài 9

Mục tiêu của bài học

- Tìm hiểu cách sử dụng hàm
- Tìm hiểu cấu trúc của hàm
- Khai báo hàm và các nguyên mẫu hàm
- Tìm hiểu các kiểu khác nhau của biến
- Hàm được gọi như thế nào
- Truyền bằng giá trị
- Truyền bằng tham chiếu
- Tìm hiểu về các qui tắc về phạm vi của hàm
- Các hàm trong các chương trình có nhiều tập tin
- Các lớp lưu trữ
- Con trỏ hàm

Hàm

- Hàm là một đoạn chương trình thực hiện một tác vụ được định nghĩa cụ thể
- Các hàm được sử dụng để rút gọn cho một chuỗi các chỉ thị được thực hiện nhiều lần
- Hàm dễ viết và dễ hiểu
- Việc gở lỗi chương trình trở nên dễ dàng hơn khi cấu trúc của chương trình rõ ràng với hình thức lập trình theo module
- Chương trình cấu tạo từ các hàm cũng dễ dàng bảo trì, bởi vì sự sửa đổi khi có yêu cầu được giới hạn trong từng hàm của chương trình

Cấu trúc hàm

- Cú pháp tổng quát của một hàm trong C như sau:

```
type_specifier function_name(arguments)
{
        body of the function
}
```

  - *type_specifier* xác định kiểu dữ liệu của giá trị mà hàm sẽ trả về.
  - Một tên hàm hợp lệ được gán cho định danh của hàm
  - Các đối số xuất hiện trong cặp dấu ngoặc () được gọi là các tham số hình thức.

Các đối số của hàm

```
#include <stdio.h>
main()
{
    int i;
    for (i =1; i <=10; i++)
    printf ("\nSquare of %d is %d ", i,squarer (i));
}

squarer (int x)
/*  int x;  */
{
     int j;
     j = x * x;
     return (j);
}
```

Actual Arguments

Formal Arguments

- Chương trình tính bình phương của các số từ 1 đến 10
- Dữ liệu được truyền từ hàm main() đến hàm squarer()
- Hàm thao tác trên dữ liệu sử dụng các đối số

Sự trở về từ một hàm

```
squarer (int x)
/*  int x;  */
{
        int j;
        j = x * x;
        return (j);
}
```

- Lệnh return ngay lập tức chuyển điều khiển từ hàm trở về chương trình gọi.
- Giá trị đặt trong cặp dấu ngoặc () theo sau lệnh return được trả về cho chương trình gọi.

Kiểu dữ liệu của hàm

```
type_specifier function_name (arguments)
{
        body of the function
}
```

- *type_specifier* không xuất hiện trước hàm squarer(), vì squarer() trả về một giá trị kiểu số nguyên int
- *type_specifier* là không bắt buộc nếu kiểu của giá trị trả về là một số nguyên hoặc nếu không có giá trị trả về
- Tuy nhiên, để tránh sự không nhất quán, một kiểu dữ liệu nên được xác định.

Gọi hàm

- Dấu chấm phẩy được đặt cuối câu lệnh khi gọi hàm, nhưng không dùng cho định nghĩa hàm
- Cặp dấu ngoặc () là bắt buộc theo sau tên hàm, cho dù hàm có đối số hay không
- Nhiều nhất một giá trị được trả về
- Chương trình có thể có nhiều hơn một hàm
- Hàm gọi đến một hàm khác được gọi là *hàm gọi*
- Hàm đang được gọi đến được gọi là *hàm được gọi*

Khai báo hàm

- Việc khai báo hàm là bắt buộc khi hàm được sử dụng trước khi nó được định nghĩa
- Hàm address() được gọi trước khi nó được định nghĩa
- Một số trình biên dịch C sẽ thông báo lỗi nếu hàm không được khai báo trước khi gọi
- Điều này còn được gọi là sự khai báo không tường minh

```
#include <stdio.h>
Main(){
        …
        address()
        …
}
address(){
        …
}
```

Nguyên mẫu hàm

- Xác định kiểu dữ liệu của các đối số

```
char abc(int x, nt y);
```

Thuận lợi :

Bất kỳ sự chuyển kiểu không hợp lệ giữa các đối số được dùng để gọi hàm và kiểu đã được định nghĩa cho các tham số của hàm sẽ được thông báo.

```
char noparam (void);
```

Các biến

- Biến cục bộ
  - Được khai báo bên trong một hàm
  - Được tạo tại điểm vào của một khối và bị hủy tại điểm ra khỏi khối đó
- Tham số hình thức
  - Được khai báo trong định nghĩa hàm như là các tham số
  - Hoạt động như một biến cục bộ bên trong một hàm
- Biến toàn cục
  - Được khai báo bên ngoài tất cả các hàm
  - Lưu các giá trị tồn tại suốt thời gian thực thi của chương trình

Lớp lưu trữ

- Mỗi biến trong C có một tính chất được gọi là lớp lưu trữ
- Lớp lưu trữ định nghĩa hai đặc tính của biến:
- **Thời gian sống** của một biến là khoảng thời gian nó duy trì một giá trị xác định
- **Tầm vực** của một biến xác định các phần của một chương trình có thể nhận ra biến đó

Lớp lưu trữ - tt

- auto
- extern
- static
- register

Các qui luật phạm vi của hàm

- Các qui luật phạm vi – là những qui luật quyết định một đoạn mã lệnh có thể truy xuất đến một đoạn mã lệnh hay dữ liệu khác hay không
- Mã lệnh bên trong một hàm là cục bộ với hàm đó
- Hai hàm có phạm vi khác nhau
- Hai hàm có cùng mức phạm vi
- Một hàm không thể được định nghĩa bên trong một hàm khác

Gọi hàm

- Truyền tham trị
- Truyền tham chiếu

Truyền bằng giá trị

- Mặc nhiên trong C, tất cả các đối số được truyền bằng giá trị
- Khi các đối số được truyền đến hàm được gọi, các giá trị được truyền thông qua các biến tạm
- Mọi sự thao tác chỉ được thực hiện trên các biến tạm
- Các đối số được gọi là truyền bằng giá trị khi giá trị của biến được truyền đến hàm được gọi và bất kỳ sự thay đổi trên giá trị này không ảnh hưởng đến giá trị gốc của biến được truyền

Truyền bằng tham chiếu

- Với truyền tham chiếu, hàm cho phép truy xuất đến địa chỉ thực trong bộ nhớ của đối số và vì vậy có thể thay đổi giá trị của các đối số của hàm gọi
- Định nghĩa

  getstr(char *ptr_str, int *ptr_int);
- Gọi

  getstr(pstr, &var);

Sự lồng nhau của lời gọi hàm

```
main()
  {
    …
  palindrome();
    …
  }
```

```
palindrome()
    {
        …
        getstr();
        reverse();
        cmp();
        …
    }
```

Các hàm trong chương trình có nhiều tập tin

- Các hàm cũng có thể được định nghĩa là **static** hoặc **external**
- Các hàm tĩnh (static) chỉ được nhận biết bên trong tập tin chương trình và phạm vi của nó không vượt ra khỏi tập tin chương trình

  static fn _type fn_name (argument list);
- Hàm ngoại (external) được nhận biết bởi tất cả các tập tin của chương trình

  extern fn_type fn_name (argument list);

Con trỏ hàm

- Lưu địa chỉ bắt đầu của hàm
- Hàm có một vị trí vật lý trong bộ nhớ, vị trí này có thể gán cho một con trỏ

```
#include <stdio.h>
#include <string.h>
void check(char *a, char *b, int (*cmp)());
main() {
      char s1[80];
      int (*p)();
      p = strcmp;
      gets(s1);
      gets(s2);
      check(s1, s2, p);
}
```

```
void check(char *a, char *b, int (*cmp)())
{
printf("testing for equality \n");
if (!(*cmp)(a,b))
        printf("Equal");
else
        printf("Not Equal");
}
```

Chuỗi

Bài 10

Mục tiêu bài học

- Giải thích biến và hằng chuỗi.
- Giải thích con trỏ đến chuỗi.
- Thực hiện các thao tác nhập/xuất chuỗi.
- Giải thích các hàm thao tác chuỗi.
- Giải thích cách thức truyền mảng vào hàm.
- Mô tả cách thức sử dụng chuỗi như các đối số của hàm.

Các Biến Chuỗi

- Chuỗi là mảng ký tự kết thúc bởi ký tự null (‘\0’).
- Có thể gán các hằng chuỗi cho các biến chuỗi.
- Hằng chuỗi là một chuỗi các ký tự nằm trong dấu nháy kép.
- Ký tự null ‘\0’ được tự động thêm vào biểu diễn bên trong của chuỗi.
- Khi khai báo một biến chuỗi, hãy dành thêm một phần tử trống cho ký tự kết thúc.

Khai Báo Biến Chuỗi

- Khai báo một biến chuỗi tiêu biểu:

  char str[10];

- **str** là một biến mảng ký tự có thể lưu giữ tối đa 10 ký tự bao gồm cả ký tự kết thúc.

Các thao tác Nhập/Xuất chuỗi

- Sử dụng các hàm trong thư viện nhập/xuất chuẩn stdio.h để thực hiện các thao tác nhập/xuất chuỗi.
- Hàm gets() là cách đơn giản nhất để nhập vào một chuỗi thông qua thiết bị nhập chuẩn.
- Các ký tự được nhập vào cho đến khi ấn phím Enter
- Hàm gets() thay thế ký tự sang dòng mới '\n' bằng ký tự '\0'
- Cú pháp: **gets(str);**

Các thao tác Nhập/Xuất chuỗi - tt

- Hàm puts() được dùng để hiển thị một chuỗi trên thiết bị xuất chuẩn.
- Cú pháp : **puts(str);**
- Các hàm scanf() và printf() được sử dụng để nhập và hiển thị các kiểu dữ liệu hỗn hợp trong cùng một câu lệnh.
- Cú pháp để nhập chuỗi:

  scanf("%s", str);
- Cú pháp để hiển thị chuỗi:

  printf("%s", str);

Các hàm về chuỗi

Các hàm xử lý chuỗi nằm trong tập tin **string.h**. Một số thao tác được thực hiện bởi các hàm này là:

- Ghép chuỗi
- So sánh chuỗi
- Xác định vị trị một ký tự trong chuỗi
- Sao chép một chuỗi sang chuỗi khác
- Tính chiều dài chuỗi

Hàm strcat()

- Nối hai giá trị chuỗi vào một chuỗi.
- Cú pháp:

strcat(str1, str2);

- Nối str2 vào cuối chuỗi str1
- Trả về str1

Hàm strcmp()

- So sánh hai chuỗi và trả về một giá trị số nguyên dựa trên kết quả của sự so sánh.
- Cú pháp:

strcmp(str1, str2);

- Hàm trả về một giá trị:
  - Nhỏ hơn 0, nếu str1<str2
  - 0, nếu str1 giống str2
  - Lớn hơn 0, nếu str1>str2

Hàm strchr()

- Xác định vị trí xuất hiện của một ký tự trong một chuỗi.
- Cú pháp: **strchr(str, chr);**
- Hàm trả về :
  - con trỏ trỏ đến vị trí tìm được đầu tiên của ký tự (trỏ bởi **chr**) trong chuỗi **str**.
  - NULL nếu **chr** không có trong chuỗi

Hàm strcpy()

- Sao chép giá trị trong một chuỗi vào một chuỗi khác.
- Cú pháp:

strcpy(str1, str2);

- Giá trị của str2 được chép vào str1
- Hàm trả về **str1**

Hàm strlen()

- Xác định chiều dài của chuỗi.
- Cú pháp:

strlen(str);

- Hàm trả về một giá trị nguyên là độ dài của **str.**

Truyền Mảng vào Hàm

- Khi mảng được truyền vào hàm như một đối số, chỉ có địa chỉ của mảng được truyền.
- Tên mảng chính là là địa chỉ của mảng.

```
void main()  {
        int ary[10];
        …
        fn_ary(ary);
        …
}
```

Truyền Mảng vào Hàm - tt

```
#include<stdio.h>
void  main() {
int num[5], ctr, sum=0;
int sum_arr(int num_arr[]);
     /* Function declaration */
 clrscr();
 for(ctr=0;ctr<5;ctr++) {
     /*Accepts numbers into the array */
     printf("\nEnter number %d:",ctr+1);
     scanf("%d", &num[ctr]);
 }
```

Truyền Mảng vào Hàm -tt

```
sum=sum_arr(num);/*Invokes the function*/
printf("\nThe sum of the array is
  %d",sum);
getch();
}
int sum_arr(int num_arr[]){
  /*Function definition*/
  int i, total;
  for(i=0,total=0;i<5;i++)
      /* Calculates the sum */
      total+=num_arr[i];
  return total;
      /* Returns the sum to main() */
}
```

Truyền Mảng vào Hàm - tt

Kết quả của chương trình trên:

```
Enter number 1: 5
Enter number 2: 10
Enter number 3: 13
Enter number 4: 26
Enter number 5: 21
The sum of the array is 75
```

Ví dụ Truyền Mảng vào Hàm

```
#include<stdio.h>
#include<string.h>
void  main() {
 char lines[5][20];
 int ctr, longctr=0;
 int longest(char lines_arr[][20]);
     /* Function declaration */
 clrscr();
 for(ctr=0;ctr<5;ctr++)  {
     /*Accepts string values into the
 array*/
     printf("\nEnter string %d:",ctr+1);
     scanf("%s", lines[ctr]);
}
```

Vd Truyền Mảng vào Hàm - tt

```
 longctr=longest(lines);
 /*Passes the array to the function*/
 printf("\n The longest string is %s",
 lines[longctr]);
 getch();
}
int longest(char lines_arr[][20]) {
 /*Function definition*/
 int i=0, l_ctr=0, prev_len, new_len;
 prev_len=strlen(lines_arr[i]);
 /*Determines the length of the first
 element*/
```

Vd Truyền Mảng vào Hàm - tt

```
  for(i++;i<5;i++)   {
      new_len=strlen(lines_arr[i]);
/* Determines the length of the next element */

  if(new_len > prev_len)     l_ctr=i;
/* Stores the subscript of the longer string */
      prev_len=new_len;
  }
  return l_ctr;
/* Returns the subscript of the longest string */
}
```

Vd Truyền Mảng vào Hàm - tt

Kết quả của chương trình trên:

```
Enter string 1: The
Enter string 2: Sigma
Enter string 3: Protocol
Enter string 4: Robert
Enter string 5: Ludlum
The longest string is Protocol
```

Các kiểu dữ liệu nâng cao – Sắp xếp

Bài 11

Mục tiêu - 1

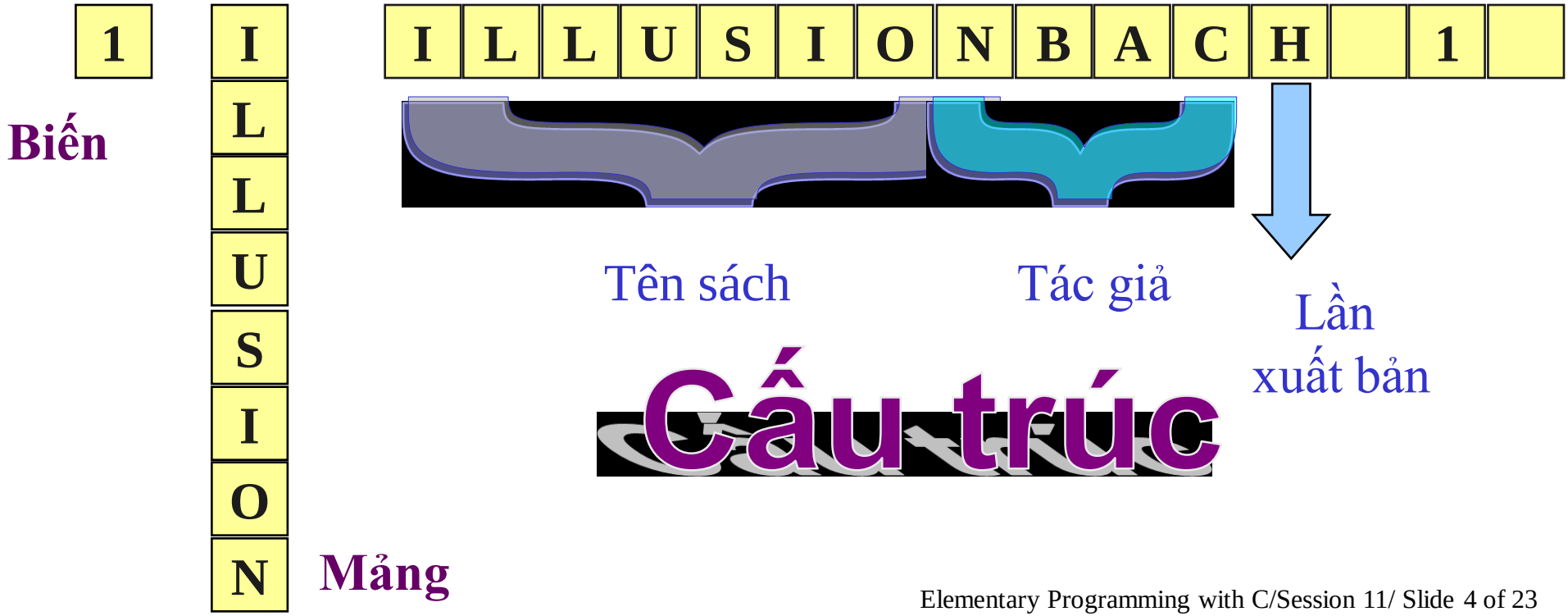
- Tìm hiểu kiểu dữ liệu cấu trúc và công dụng
- Định nghĩa cấu trúc
- Khai báo các biến kiểu cấu trúc
- Cách truy cập vào các phần tử của cấu trúc
- Khởi tạo biến cấu trúc
- Sử dụng biến cấu trúc trong câu lệnh gán
- Cách truyền tham số cấu trúc
- Sử dụng mảng các cấu trúc
- Tìm hiểu cách khởi tạo mảng các cấu trúc

Mục tiêu - 2

- Con trỏ cấu trúc
- Cách truyền tham số kiểu con trỏ cấu trúc
- Tìm hiểu từ khóa typedef
- Sắp xếp mảng bằng phương pháp Bubble sort và Insertion sort.

Cấu Trúc

- Một cấu trúc bao gồm các mẫu dữ liệu, không nhất thiết cùng kiểu, được nhóm lại với nhau.
- Một cấu trúc có thể bao gồm nhiều mẫu dữ liệu như vậy.



Định Nghĩa Cấu Trúc

- Việc định nghĩa cấu trúc sẽ tạo ra kiểu dữ liệu mới cho phép người dùng sử dụng chúng để khai báo các biến kiểu cấu trúc .
- Các biến trong cấu trúc được gọi là các phần tử của cấu trúc hay thành phần của cấu trúc
- Ví dụ:

```
struct cat {
        char bk_name [25];
        char author [20];
        int edn;
        float price;
};
```

Khai Báo Biến Cấu Trúc

- Khi một cấu trúc đã được định nghĩa, chúng ta có thể khai báo một hoặc nhiều biến kiểu này.
- Ví dụ: **struct cat books1;**
- Câu lệnh này sẽ dành đủ vùng nhớ để lưu trữ tất cả các mục trong một cấu trúc.

```
struct cat {
        char bk_name[25];
        char author[20];
        int edn;
        float price;
} books1, books2;
```

Cách khác

```
struct cat books1, books2;
```

hoặc

```
struct cat books1;
struct cat books2;
```

Truy Cập Phần Tử của Cấu Trúc

- Các phần tử của cấu trúc được truy cập thông qua việc sử dụng **toán tử chấm** (.), toán tử này còn được gọi là **toán tử thành viên - membership**.
- Cú pháp:

  structure_name.element_name
- Ví dụ:

  scanf(“%s”, books1.bk_name);

Khởi Tạo Cấu Trúc

- Giống như các biến khác và mảng, các biến kiểu cấu trúc có thể được khởi tạo tại thời điểm khai báo

```
struct employee
   {        int no;
    char name [20];
   };
```

- Các biến **emp1** và **emp2** có kiểu **employee** có thể được khai báo và khởi tạo như sau:

```
struct employee emp1 = {346, “Abraham”};
struct employee emp2 = {347, “John”};
```

Câu Lệnh Gán Sử Dụng Các Cấu Trúc - 1

- Có thể sử dụng câu lệnh gán đơn giản để gán giá trị của một biến cấu trúc cho một biến khác có cùng kiểu
- Chẳng hạn, nếu **books1** và **books2** là các biến cấu trúc có cùng kiểu, thì câu lệnh sau là hợp lệ

```
books2 = books1;
```

Câu Lệnh Gán Sử Dụng Các Cấu Trúc - 2

- Trong trường hợp không thể dùng câu lệnh gán trực tiếp, thì có thể sử dụng hàm tạo sẵn **memcpy()**
- Cú pháp:

  memcpy (char * destn, char &source, int nbytes);
- Ví dụ:

  memcpy (&books2, &books1, sizeof(struct cat));

Cấu Trúc Lồng Trong Cấu Trúc

- Một cấu trúc có thể lồng trong một cấu trúc khác. Tuy nhiên, một cấu trúc không thể lồng trong chính nó.

```
struct issue      {
    char borrower [20];
    char dt_of_issue[8];
    struct cat books;
}issl;
```

- Việc truy cập vào các phần tử của cấu trúc này tương tự như với cấu trúc bình thường khác,

```
issl.borrower
```

- Để truy cập vào phần tử của cấu trúc cat là một phần của cấu trúc issl ,

```
issl.books.author
```

Truyền tham số kiểu cấu trúc

- Tham số của hàm có thể là một cấu trúc.
- Là một phương tiện hữu dụng khi muốn truyền một nhóm các thành phần dữ liệu có quan hệ logic với nhau thông qua một biến thay vì phải truyền từng thành phần một
- Kiểu của tham số thực phải trùng với kiểu của tham số hình thức.

Mảng Cấu Trúc

- Một áp dụng thường gặp là mảng cấu trúc
- Một kiểu cấu trúc phải được định nghĩa trước, sau đó một biến mảng có kiểu đó mới được khai báo
- Ví dụ: **struct cat books[50];**
- Để truy cập vào thành phần author của phần tử thứ tư của mảng **books**:

  books[4].author

Khởi Tạo Các Mảng Cấu Trúc

- Mảng cấu trúc được khởi tạo bằng cách liệt kê danh sách các giá trị phần tử của nó trong một cặp dấu móc
- Ví dụ:

```
struct unit {
        char ch;
        int i;
};
struct unit series [3] =
        {{‘a’, 100}{‘b’, 200}{‘c’, 300}};
```

Con Trỏ Đến Cấu Trúc

- Con trỏ cấu trúc được khai báo bằng cách đặt dấu * trước tên của biến cấu trúc.
- Toán tử -> được dùng để truy cập vào các phần tử của một cấu trúc sử dụng một con trỏ
- Ví dụ:

```
struct cat *ptr_bk;
ptr_bk = &books;
printf("%s",ptr_bk->author);
```

- Con trỏ cấu trúc được truyền vào hàm, cho phép hàm thay đổi trực tiếp các phần tử của cấu trúc.

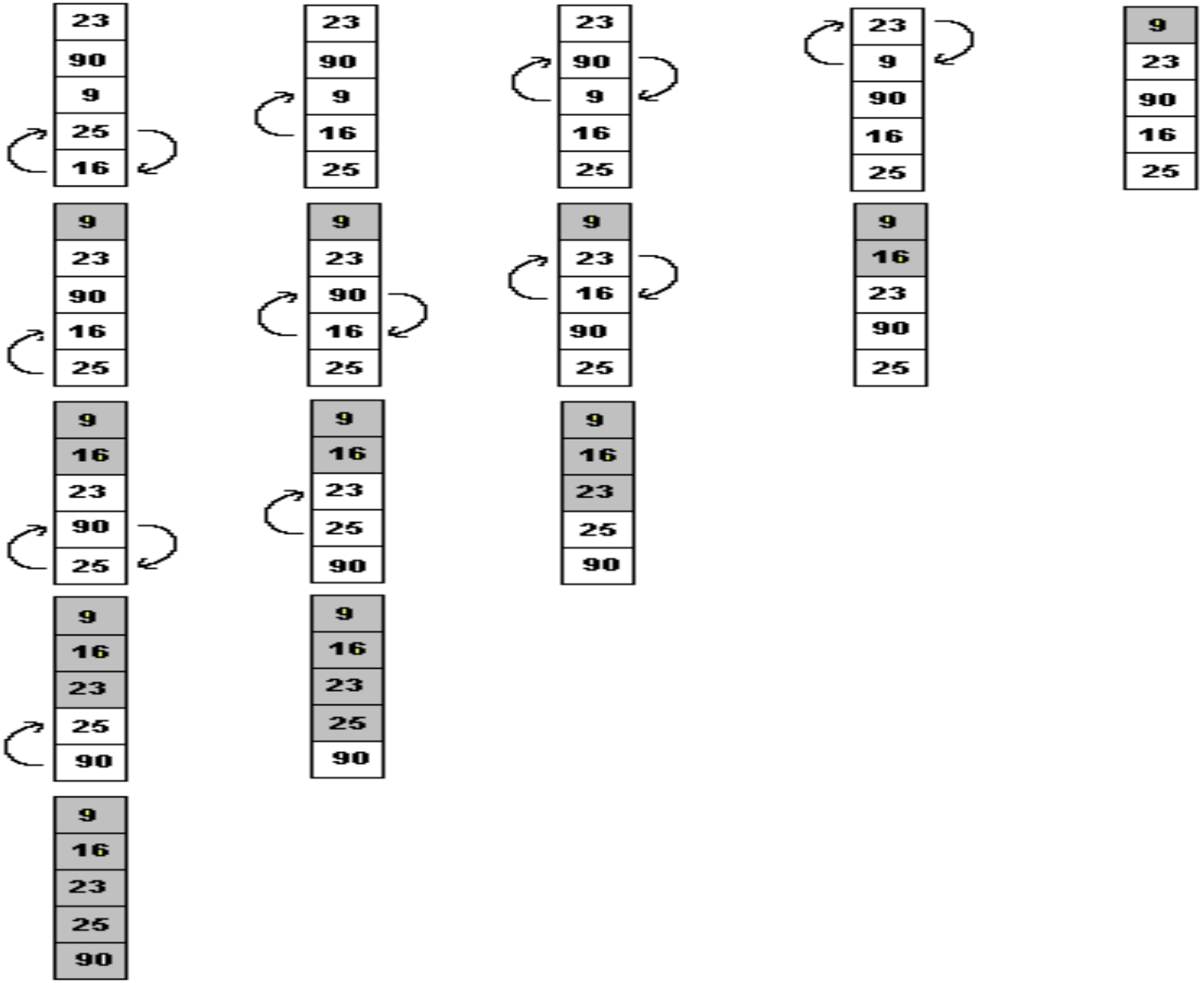
Từ Khóa typedef

- Một kiểu dữ liệu có thể được định nghĩa bằng cách sử dụng từ khóa typedef
- Nó không tạo ra một kiểu dữ liệu mới, mà định nghĩa một tên mới cho một kiểu đã có.
- Cú pháp: **typedef type name;**
- Ví dụ: **typedef float deci;**
- typedef không thể sử dụng với *storage classes*

Sắp xếp mảng

- Sắp xếp liên quan đến việc thay đổi vị trí các phần tử theo thứ tự xácđịnh như tăng dần hay giảm dần
- Dữ liệu trong mảng sẽ dễ dàng tìm thấy hơn nếu mảng được sắp xếp
- Hai phương pháp sắp xếp mảng được trình bày: Bubble Sort và Insertion Sort
- Trong phương pháp Bubble sort, việc so sánh bắt đầu từ phần tử dưới cùng và phần tử có giá trị nhỏ hơn sẽ chuyển dần lên trên (nổi bọt)
- Trong phương pháp Insertion sort, mỗi phần tử trong mảng được xem xét, và đặt vào vị trí đúng của nó giữa các phần tử đã được sắp xếp

Bubble Sort


23 90 9 25 16
23 90 9 16 25
23 90 9 16 25
23 9 90 16 25
9 23 90 16 25
9 23 90 16 25
9 23 16 90 25
9 16 23 90 25
9 16 23 25 90
9 16 23 25 90
9 16 23 25 90
9 16 23 25 90
9 16 23 25 90

Bubble Sort - tt

```
#include <stdio.h>
void main() {
  int i,j,temp,arr_num[5]={23,90,9,25,16};
  clrscr();
  for(i=3;i>=0;i--) /* Tracks every pass */
      for(j=4;j>=4-i;j--) {
            /* Compares elements */
            if(arr_num[j]<arr_num[j-1])
                  {     temp=arr_num[j];
                        arr_num[j]=arr_num[j-1];
                        arr_num[j-1]=temp;
                  }
            }
```

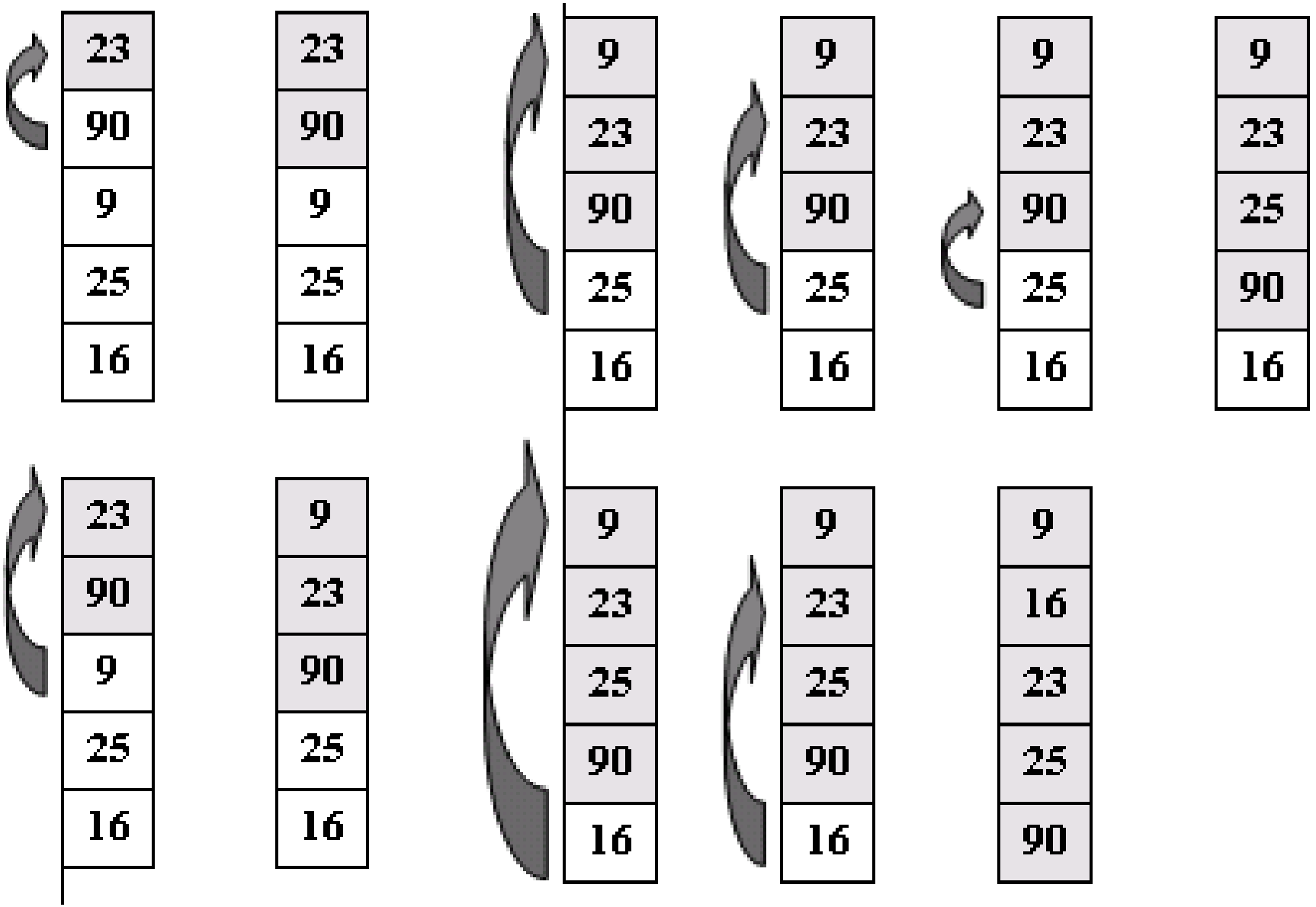
Contd…..

Bubble Sort - tt

```
  printf("\nThe sorted array");
  for(i=0;i<5;i++)
            printf("\n%d", arr_num[i]);

  getch();
}
```

Insertion Sort



Insertion Sort - tt

```
#include<stdio.h>
void main()  {
        int i, j, arr[5] = { 23, 90, 9, 25, 16 };
        char flag;
        clrscr();
        /*Loop to compare each element of the unsorted part of the array*/
        for(i=1; i<5; i++)
        /*Loop for each element in the sorted part of the array*/
                for(j=0,flag='n'; j<i&&flag=='n'; j++)  {
                        if(arr[j]>arr[i])  {
        /*Invoke the function to insert the number*/
                                insertnum(arr, i, j);
                                flag='y';
                        }
                }
        printf("\n\nThe sorted array\n");
        for(i=0; i<5; i++)
                printf("%d\t", arr[i]);
        getch();
}
```

Insertion Sort-3

```
insertnum(int arrnum[], int x, int y) {
      int temp;
      /*Store the number to be inserted*/
      temp=arrnum[x];
      /*Loop to push the sorted part of the
array down from the position where the number
has to inserted*/
      for(;x>y; x--) arrnum[x]=arrnum[x-1];
      /*Insert the number*/
      arrnum[x]=temp;
}
```

Quản lý tập tin

Bài 12

Mục tiêu

- Giải thích streams và file
- Thảo luận về các streams văn bản và streams nhị phân
- Giải thích các hàm xử lý tập tin
- Giải thích về con trỏ tập tin
- Thảo luận về con trỏ hiện hành
- Giải thích về các đối số dòng lệnh

Nhập/Xuất Tập Tin

- Tất cả các thao tác nhập/xuất trong C đều được thực hiện bằng các hàm trong thư viện chuẩn
- Tiếp cận này làm cho hệ thống tập tin của C rất mạnh và uyển chuyển
- Nhập/xuất trong C có thể theo 2 cách: dữ liệu có thể truyền ở dạng biểu diễn nhị phân bên trong của nó hay ở dạng văn bản mà con người có thể đọc được

Streams

- Hệ thống tập tin của C làm việc với rất nhiều thiết bị khác nhau bao gồm máy in, ổ đĩa, ổ băng từ và các thiết bị đầu cuối
- Mặc dù tất cả các thiết bị đều khác nhau, hệ thống tập tin có vùng đệm sẽ chuyển mỗi thiết bị về một thiết bị logic gọi là một stream
- Vì mọi streams đều hoạt động tương tự, nên việc quản lý các thiết bị khác nhau rất dễ dàng
- Có hai loại streams – stream văn bản và stream nhị phân

Streams Văn Bản

- Một streams văn bản là một chuỗi các ký tự có thể được tổ chức thành các dòng kết thúc bằng một ký tự sang dòng mới
- Trong một stream văn bản, có thể xảy ra một vài sự chuyển đổi ký tự khi môi trường yêu cầu
- Vì vậy, mối quan hệ giữa các ký tự được ghi (hay đọc) và những ký tự ở thiết bị ngoại vi có thể không phải là mối quan hệ một-một
- Và cũng vì sự chuyển đổi có thể xảy ra này, số lượng ký tự được ghi (hay đọc) có thể không giống như số lượng ký tự ở thiết bị ngoại vi

Streams Nhị Phân

- Một streams nhị phân là một chuỗi các byte với sự tương ứng một-một với thiết bị ngoại vi, nghĩa là, không có sự chuyển đổi ký tự.
- Số lượng byte đọc (hay ghi) cũng sẽ giống như số lượng byte ở thiết bị ngoại vi
- Các stream nhị phân là các chuỗi byte thuần túy, mà không có bất kỳ ký hiệu nào dùng để chỉ ra điểm kết thúc của tập tin hay kết thúc của mẫu tin
- Kết thúc của tập tin được xác định bằng kích thước của tập tin

Tập Tin

- Một tập tin có thể tham chiếu đến bất cứ thứ gì từ một tập tin trên đĩa đến một thiết bị đầu cuối hay một máy in
- Một tập tin kết hợp với một stream bằng cách thực hiện thao tác mở và ngưng kết hợp bằng thao tác đóng
- Khi một chương trình kết thúc bình thường, tất cả các tập tin đều tự động đóng
- Khi một chương trình kết thúc bất thường, các tập tin vẫn còn mở

Các Hàm Cơ Bản Về Tập Tin

Tên	Chức năng
fopen()	Mở một tập tin
fclose()	Đóng một tập tin
fputc()	Ghi một ký tự vào một tập tin
fgetc()	Đọc một ký tự từ một tập tin
fread()	Đọc từ một tập tin vào một vùng đệm
fwrite()	Ghi từ một vùng đệm vào tập tin
fseek()	tìm một vị trí nào đó trong tập tin
fprintf()	Hoạt động giống như printf(), nhưng trên một tập tin
fscanf()	Hoạt động giống như scanf(), nhưng trên một tập tin
feof()	Trả về true nếu đã đến cuối tập tin
ferror()	Trả về true nếu xảy ra một lỗi
rewind()	Đặt lại con trỏ định vị trí bên trong tập tin về đầu tập tin
remove()	Xóa một tập tin
fflush()	Ghi dữ liệu từ một vùng đệm bên trong vào một tập tin xác định

Con Trỏ Tập Tin

- Một con trỏ tập tin phải cần cho việc đọc và ghi các tập tin
- Nó là một con trỏ đến một cấu trúc chứa thông tin về tập tin. Thông tin bao gồm tên tập tin, vị trí hiện tại của tập tin, liệu tập tin có đang được đọc hay ghi, và liệu có bất kỳ lỗi nào xuất hiện hay đã đến cuối tập tin
- Định nghĩa lấy từ studio.h bao gồm một khai báo cấu trúc tên FILE
- Câu lệnh khai báo duy nhất cần thiết cho một con trỏ tập tin là: **FILE *fp**

Mở Một Tập Tin Văn Bản

- Hàm fopen() mở một stream để sử dụng và liên kết một tập tin với stream đó
- Hàm fopen() trả về con trỏ kết hợp với tập tin
- Nguyên mẫu của hàm fopen() là:

FILE *fopen(const char *filename, const char *mode);

Chế độ	Ý nghĩa
R	Mở một tập tin văn bản để đọc
w	Tạo một tập tin văn bản để ghi
a	Nối vào một tập tin văn bản
r+	Mở một tập tin văn bản để đọc/ghi
w+	Tạo một tập tin văn bản để đọc/ghi
a+f	Nối hoặc tạo một tập tin văn bản để đọc/ghi

Đóng Một Tập Tin Văn Bản

- Việc đóng một tập tin sau khi sử dụng là một điều quan trọng
- Thao tác này sẽ giải phóng tài nguyên và làm giảm nguy cơ vượt quá giới hạn số tập tin có thể mở.
- Đóng một stream sẽ làm sạch và chép vùng đệm kết hợp của nó ra ngoài, một thao tác quan trọng để tránh mất dữ liệu khi ghi ra đĩa
- Hàm fclose() đóng một stream đã được mở bằng hàm fopen()
- Nguyên mẫu của hàm fclose() là :

```
int  fclose(FILE *fp);
```

- Hàm fcloseall() đóng tất cả các streams đang mở

Ghi Một Ký Tự – Tập Tin Văn Bản

- Streams có thể được ghi vào tập tin theo cách từng ký tự một hoặc theo từng chuỗi
- Hàm fputc() được sử dụng để ghi các ký tự vào tập tin đã được mở trước đó bằng hàm fopen().
- Nguyên mẫu của hàm này là:

int fputc(int ch, FILE *fp);

Đọc Một Ký Tự – Tập Tin Văn Bản

- Hàm fgetc() được dùng để đọc các ký tự từ một tập tin đã được mở bằng hàm fopen() ở chế độ đọc
- Nguyên mẫu của hàm là:

```
int fgetc(int ch, FILE *fp);
```

- Hàm fgetc() trả về ký tự kế tiếp của vị trí hiện hành trong stream input, và tăng con trỏ định vị trí bên trong tập tin lên

Nhập Xuất Chuỗi

- Các hàm fputs() and fgets() ghi vào và đọc ra các chuỗi ký tự từ tập tin trên đĩa
- Hàm fputs() viết toàn bộ chuỗi vào stream đã định
- Hàm fgets() đọc một chuỗi từ stream đã cho cho đến khi đọc được một ký tự sang dòng mới hoặc sau khi đã đọc được length-1 ký tự.
- Nguyên mẫu của các hàm này là:

int fputs(const char *str, FILE *fp);

char *fgets(char *str, int length, FILE *fp);

Mở Một Tập Tin Nhị Phân

- Hàm fopen() mở một stream để sử dụng và liên kết một tập tin với stream đó.
- Hàm fopen() trả về một con trỏ tập tin kết hợp với tập tin.
- Nguyên mẫu của hàm fopen() là:

FILE *fopen(const char *filename,const char *mode);

Chế độ	Ý nghĩa
rb	Mở một tập tin nhị phân để đọc
wb	Tạo một tập tin nhị phân để ghi
ab	Nối vào một tập tin nhị phân
r+b	Mở một tập tin nhị phân để đọc/ghi
w+b	Tạo một tập tin nhị phân để đọc/ghi
a+b	Nối vào một tập tin nhị phân để đọc/ghi

Đóng Tập Tin Nhị Phân

- Hàm fclose() đóng một stream đã được mở bằng hàm fopen()
- Nguyên mẫu của hàm fclose() là:

```
int  fclose(FILE *fp);
```

Hàm fread() và fwrite()

- Hàm fread() và fwrite() là các hàm đọc hoặc ghi dữ liệu không định dạng.
- Chúng được dùng để đọc ra và viết vào tập tin toàn bộ khối dữ liệu.
- Hầu hết các chương trình ứng dụng hữu ích đều đọc và ghi các kiểu dữ liệu do người dùng định nghĩa, đặc biệt là các cấu trúc.
- Nguyên mẫu của các hàm này là:

```
size_t fread(void *buffer, size_t num_bytes,
                    size_t  count, FILE *fp);
size_t fwrite(const void *buffer, size_t num_bytes,
                    size_t  count, FILE *fp);
```

Sử Dụng feof()

- Hàm feof() trả về true nếu đã đến cuối tập tin, nếu không nó trả về false (0).
- Hàm này được dùng trong khi đọc dữ liệu nhị phân.
- Nguyên mẫu là:

int feof (FILE *fp);

Hàm rewind()

- Hàm rewind() đặt lại con trỏ định vị trí bên trong tập tin về đầu tập tin
- Nó lấy con trỏ tập tin làm đối số
- Cú pháp:

rewind(fp);

Hàm ferror()

- Hàm **ferror()** xác định liệu một thao tác trên tập tin có sinh ra lỗi hay không
- Vì mỗi thao tác đặt lại tình trạng lỗi, hàm **ferror**() phải được gọi ngay sau mỗi thao tác; nếu không, lỗi sẽ bị mất
- Nguyên mẫu của hàm là:

```
int ferror(FILE *fp);
```

Xóa Tập Tin

- Hàm remove() xóa một tập tin đã cho
- Nguyên mẫu của hàm là:

int remove(char *filename);

Làm Sạch các stream

- Hàm fflush() sẽ làm sạch vùng đệm và chép những gì có trong vùng đệm ra ngoài tùy theo kiểu tập tin
- Một tập tin được mở để đọc sẽ có vùng đệm nhập liệu trống, trong khi một tập tin được mở để ghi thì vùng đệm xuất của nó sẽ được ghi vào tập tin
- Nguyên mẫu của hàm là:

  int fflush(FILE *fp);
- Hàm fflush(), không có đối số, sẽ làm sạch tất cả các tập tin đang mở để xuất

Các Stream Chuẩn

Mỗi khi một chương trình C bắt đầu thực thi dưới DOS, hệ điều hành sẽ tự động mở 5 stream đặc biệt:

- Nhập chuẩn (stdin)
- Xuất chuẩn (stdout)
- Lỗi chuẩn (stderr)
- Máy in chuẩn (stdprn)
- Thiết bị phụ trợ chuẩn (stdaux)

Con Trỏ Kích Hoạt Hiện Hành

- Một con trỏ được duy trì trong cấu trúc FILE để lần theo vị trí nơi mà các thao tác nhập/xuất đang diễn ra
- Mỗi khi một ký tự được đọc từ hay ghi vào một stream, con trỏ kích hoạt hiện hành (gọi là curp) được tăng lên
- Vị trí hiện hành của con trỏ này có thể được tìm thấy bằng sự trợ giúp của hàm ftell().
- Nguyên mẫu của hàm là:

```
long int ftell(FILE *fp);
```

Đặt Lại Vị Trí Hiện Hành - 1

- Hàm fseek() định lại vị trí của curp dời đi một số byte tính từ đầu, từ vị trí hiện hành hay từ cuối stream là tùy vào vị trí được qui định khi gọi hàm fseek()
- Nguyên mẫu của hàm là:

```
int fseek (FILE *fp, long int offset,
           int origin);
```

Đặt Lại Vị Trí Hiện Hành - 2

- origin chỉ định vị trí bắt đầu tìm kiếm và phải có giá trị như sau:

Origin	Vị trí trong tập tin
SEEK_SET hay 0	Bắt đầu tập tin
SEEK_CUR hay 1	Vị trí của con trỏ trong tập tin hiện hành
SEEK_END hay 2	Cuối tập tin

fprintf() và fscanf()-1

- Hệ thống nhập xuất có vùng đệm bao gồm các hàm **fprintf**() và **fscanf**() tương tự như hàm **printf**() và **scanf**() ngoại trừ rằng chúng thao tác trên tập tin
- Nguyên mẫu của các hàm này là:

```
int fprintf(FILE * fp,
            const char *control_string,..);
int fscanf(FILE *fp,
            const char *control_string,...);
```

fprintf() và fscanf() - 2

- Mặc dù fprintf() và fscanf() là cách dễ nhất nhưng không phải luôn luôn là hiệu quả nhất
- Mỗi lời gọi phải mất thêm một khoảng thời gian overhead, vì dữ liệu được ghi theo dạng ASCII có định dạng chứ không phải theo định dạng nhị phân
- Vì vậy, nếu tốc độ và độ lớn của tập tin là vấn đề đáng ngại, thì fread() và fwrite() sẽ là lựa chọn tốt hơn