

www.mientayvn.com

Khi đọc qua tài liệu này, nếu phát hiện sai sót hoặc nội dung kém chất lượng xin hãy thông báo để chúng tôi sửa chữa hoặc thay thế bằng một tài liệu cùng chủ đề của tác giả khác. Tài liệu này bao gồm nhiều tài liệu nhỏ có cùng chủ đề bên trong nó. Phần nội dung bạn cần có thể nằm ở giữa hoặc ở cuối tài liệu này, hãy sử dụng chức năng Search để tìm chúng.

Bạn có thể tham khảo nguồn tài liệu được dịch từ tiếng Anh tại đây:

http://mientayvn.com/Tai_lieu_da_dich.html

Thông tin liên hệ:

Yahoo mail: thanhlam1910_2006@yahoo.com

Gmail: frbwrthes@gmail.com

Theo yêu cầu của khách hàng, trong một năm qua, chúng tôi đã dịch qua 16 môn học, 34 cuốn sách, 43 bài báo, 5 sổ tay (chưa tính các tài liệu từ năm 2010 trở về trước) Xem ở đây

**DỊCH VỤ
DỊCH
TIẾNG
ANH
CHUYÊN
NGÀNH
NHANH
NHẤT VÀ
CHÍNH
XÁC
NHẤT**

Chỉ sau một lần liên lạc, việc dịch được tiến hành

Giá cả: có thể giảm đến 10 nghìn/1 trang

Chất lượng: Tạo dựng niềm tin cho khách hàng bằng công nghệ 1. Bạn thấy được toàn bộ bản dịch; 2. Bạn đánh giá chất lượng. 3. Bạn quyết định thanh toán.

TRƯỜNG ĐẠI HỌC ĐÀ LẠT



GIÁO TRÌNH
**KỸ THUẬT LẬP TRÌNH
NÂNG CAO**

TRẦN HOÀNG THỌ

2002

MỤC LỤC

LỜI NÓI ĐẦU	4
PHẦN I.....	5
CHƯƠNG I	5
I. MỞ ĐẦU	5
1. Mô tả đệ quy	5
2. Các loại đệ quy	6
II. MÔ TẢ ĐỆ QUY CÁC CẤU TRÚC DỮ LIỆU.....	7
III. MÔ TẢ ĐỆ QUY GIẢI THUẬT	7
1. Giải thuật đệ quy.....	7
2. Chương trình con đệ quy.....	8
3. Mã hóa giải thuật đệ quy trong các ngôn ngữ lập trình.	11
4. Một số dạng giải thuật đệ quy đơn giản thường gặp	13
CHƯƠNG II	16
I. CÁC NỘI DUNG CẦN LÀM ĐỂ TÌM GIẢI THUẬT ĐỆ QUY CHO MỘT BÀI TOÁN.	16
1. Thông số hoá bài toán.....	16
2. Phát hiện các trường hợp suy biến (neo) và tìm giải thuật cho các trường hợp này.	16
3. Phân rã bài toán tổng quát theo phương thức đệ quy.	16
II. MỘT SỐ BÀI TOÁN GIẢI BẰNG GIẢI THUẬT ĐỆ QUY ĐIỂN HÌNH.	17
1. Bài toán tháp Hà Nội	17
2. Bài toán chia thưởng.....	19
3. Bài toán tìm tất cả các hoán vị của một dãy phần tử.....	21
4. Bài toán sắp xếp mảng bằng phương pháp trộn (Sort-Merge).	24
5. Bài toán tìm nghiệm xấp xỉ của phương trình $f(x)=0$	25
CHƯƠNG III	28
I. CƠ CHẾ THỰC HIỆN GIẢI THUẬT ĐỆ QUY.....	28
II. TỔNG QUAN VỀ VẤN ĐỀ KHỬ ĐỆ QUY.....	32
III. CÁC TRƯỜNG HỢP KHỬ ĐỆ QUY ĐƠN GIẢN.	33
1. Các trường hợp khử đệ quy bằng vòng lặp	33
2. Khử đệ quy hàm đệ quy arzac.....	41
3. Khử đệ quy một số dạng thủ tục đệ quy thường gặp.	45
Phần II	52
CHƯƠNG IV	52
I. CÁC GIAI ĐOẠN TRONG CUỘC SỐNG CỦA MỘT PHẦN MỀM	52
1) Đặc tả bài toán	52
2) Xây dựng hệ thống	52
3) Sử dụng và bảo trì hệ thống	53
II. ĐẶC TẢ	53
1. Đặc tả bài toán.....	53
2. Đặc tả chương trình (ĐTCT).....	54
3. Đặc tả đoạn chương trình	55
III. NGÔN NGỮ LẬP TRÌNH.....	57
CHƯƠNG V	59
I. CÁC KHÁI NIỆM VỀ TÍNH ĐÚNG.	59
II. HỆ LUẬT HOARE (HOARES INFERENCE RULES).	59
1. Các luật hệ quả (Consequence rules)	60

2. Tiên đề gán (The Assignment Axiom)	61
3. Các luật về các cấu trúc điều khiển	61
III. KIỂM CHỨNG ĐOẠN CHƯƠNG TRÌNH KHÔNG CÓ VÒNG LẶP.	64
IV. KIỂM CHỨNG ĐOẠN CHƯƠNG TRÌNH CÓ VÒNG LẶP.	68
1. Bất biến.....	68
2. Lý luận quy nạp và chứng minh bằng quy nạp.	70
3. Kiểm chứng chương trình có vòng lặp while.	71
CHƯƠNG VI.....	76
I. CÁC KHÁI NIỆM.	76
1. Đặt vấn đề.	76
2. Định nghĩa WP(S,Q).....	76
3. Hệ quả của định nghĩa.....	76
4. Các ví dụ.....	77
II. TÍNH CHẤT CỦA WP.	77
III. CÁC PHÉP BIẾN ĐỔI TÂN TỪ.....	78
1. Toán tử gán (tiên đề gán).	78
2. Toán tử tuần tự.....	78
3. Toán tử điều kiện.	79
4. Toán tử lặp.	80
IV. LƯỢC ĐỒ KIỂM CHỨNG HỢP LÝ VÀ CÁC ĐIỀU KIỆN CẦN KIỂM CHỨNG.....	84
1. Lược đồ kiểm chứng.	84
2. Kiểm chứng tính đúng.	85
3. Tập tối tiểu các điều kiện cần kiểm chứng.	93
PHỤ LỤC	96
I. LOGIC TOÁN.....	96
II. LOGIC MỆNH ĐỀ.....	96
1. Phân tích	96
2. Các liên từ logic.	97
3. Ý nghĩa của các liên từ Logic. Bảng chân trị.	97
4. Lý luận đúng.	98
5. Tương đương (Equivalence).	99
6. Tính thay thế, tính truyền và tính đối xứng.	100
7. Bài toán suy diễn logic.....	100
8. Các luật suy diễn (rules of inference).	102
III. LOGIC TÂN TỪ.	103
1. Khái niệm	103
2. Các lượng từ logic	105
3. Tập hợp và tân từ.....	107
4. Các lượng từ số học.....	107

+ Một số các trường hợp suy biến mà giá trị hàm tại đó đã được biết trước hoặc có thể tính một cách đơn giản (không đệ quy) .

Như :

$$FAC(0) = 1, FIBO(0) = FIBO(1) = 1, C_n^0 = 1, C_n^m = 0 \text{ với } m > n > 0.$$

+ Trường hợp tổng quát việc tính hàm sẽ được đưa về tính hàm ở giá trị “ bé hơn” (gần với giá trị neo) của đối số .

Như :

$$FAC(n) = n * FAC(n - 1) ;$$

$$FIBO(n) = FIBO(n - 1) + FIBO(n - 2) .$$

Trong tập biến của hàm có một nhóm mà độ lớn của nó quyết định độ phức tạp của việc tính giá trị hàm . Nhóm biến đó gọi là nhóm biến điều khiển . Giá trị biên của nhóm biến điều khiển ứng với trường hợp suy biến . Giá trị của nhóm biến điều khiển sẽ thay đổi qua mỗi lần gọi đệ quy với xu hướng tiến đến giá trị biên (tương ứng với các trường hợp suy biến của hàm) .

b) Các thủ tục đệ quy.

Thủ tục đệ quy là thủ tục có chứa lệnh gọi đến nó . Thủ tục đệ quy thường được sử dụng để mô tả các thao tác trên cấu trúc dữ liệu có tính đệ quy

Ví dụ 1 :

Xem dãy n phần tử a[1:n] là sự kết hợp giữa dãy a[1:n-1] và a[n] .

Do đó :

- Thủ tục tìm max trong dãy a[1:n] (thủ tục TMax) có thể thực hiện theo luật đệ quy :
 - + Tìm max trong dãy con a[1:n] (gọi đệ quy TMax(a[1:n-1])) .
 - + Tìm max của 2 số : TMax(a[1:n-1]) và a[n] (giải thuật không đệ quy).

Tức là :

$$TMax(a[1:n]) = \max(TMax(a[1:n-1]), a[n])$$

với $TMax(a[m:m]) = a[m]$; (trường hợp neo)

$$\max(x,y) = x > y ? x : y ; \quad (\text{giải thuật tính max 2 số : if (x>y) then } \max(x,y) = x \text{ else } \max(x,y) = y)$$

- Thủ tục tính tổng các phần tử (thủ tục TSUM) có thể thực hiện theo luật đệ quy :

+ Tìm tổng dãy con a[1:n] (gọi đệ quy TSUM(a[1:n-1])) .

+ Tìm tổng của 2 số : TSUM(a[1:n-1]) và a[n] (giải thuật không đệ quy).

quy).

Tức là :

$$TSUM(a[1:n]) = a[n] + TSUM(a[1:n-1])$$

với $TSUM(a[m:m]) = a[m]$

Ví dụ 2 :

Xem dãy a[m : n] là sự kết nối giữa hai dãy: dãy a[m:((m+n) div 2)] và dãy a[(((m+n) div 2)+1) :n] .

Do đó :

- Thủ tục tìm max trong dãy $a[1:n]$ (thủ tục T_{max1}) có thể thực hiện theo luật đệ quy :
 - + Tìm max trong dãy con trái $a[m:((m+n) \text{ div } 2)]$
(gọi đệ quy $T_{max1}(a[m:((m+n) \text{ div } 2)])$).
 - + Tìm max trong dãy con phải $a[((m+n) \text{ div } 2)+1 :n]$.
(gọi đệ quy $T_{max1}(a[((m+n) \text{ div } 2)+1 :n])$).
 - + Tìm max của 2 số : $T_{max1}(a[m:((m+n) \text{ div } 2)])$ và $T_{max1}(a[((m+n) \text{ div } 2)+1 :n])$. (giải thuật không đệ quy).

Tức là : $T_{max1}(a[m:n]) = \max(T_{max1}(a[m:((m+n) \text{ div } 2)]), T_{max1}(a[((m+n) \text{ div } 2)+1 :n]))$.

với $T_{max1}(a[m:m]) = a[m]$; (trường hợp neo)
 $\max(x,y) = x > y ? x : y$;

- Thủ tục tính tổng các phần tử (T_{SUM1}) có thể thực hiện theo luật đệ quy :
 - + Tìm tổng dãy con trái $a[m:((m+n) \text{ div } 2)]$
(gọi đệ quy $T_{SUM1}(a[m:((m+n) \text{ div } 2)])$).
 - + Tìm tổng dãy con phải $a[((m+n) \text{ div } 2)+1 :n]$.
(gọi đệ quy $T_{SUM1}(a[((m+n) \text{ div } 2)+1 :n])$).
 - + Tìm tổng của 2 số :
 $T_{SUM1}(a[m:((m+n) \text{ div } 2)])$ và $T_{SUM1}(a[((m+n) \text{ div } 2)+1 :n])$.

Tức là : $T_{SUM1}(a[m:n]) = T_{SUM1}(a[m:((m+n) \text{ div } 2)]) + T_{SUM1}(a[((m+n) \text{ div } 2)+1 :n])$

với $T_{SUM1}(a[m:m]) = a[m]$

Ví dụ 3 :

Cây nhị phân tìm kiếm kiểu T(BST) là một cấu trúc gồm : một nút kiểu T kết nối với 2 cây con nhị phân tìm kiếm kiểu T nên :

- Thủ tục quét cây nhị phân tìm kiếm theo thứ tự giữa (LNF) là :
 - + Quét cây con trái theo thứ tự giữa ;
 - + Thăm nút gốc ;
 - + Quét cây con phải theo thứ tự giữa ;
- Thủ tục tìm kiếm giá trị α_0 trên cây nhị phân tìm kiếm Root là :

Nếu Root $\equiv \emptyset$ thì thực hiện thao tác rỗng (không làm gì)

Con không

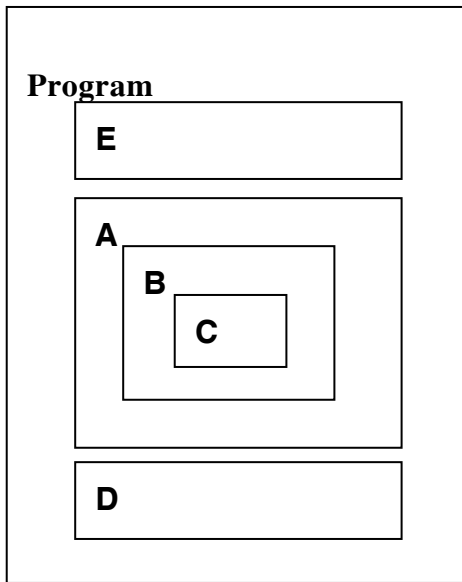
nếu giá trị tại nút gốc = α_0 thì thông báo tìm thấy và dừng

Còn không

nếu giá trị tại nút gốc < α_0 thì tìm ở cây con trái

Còn không thì tìm ở cây con phải .

Nhận xét :



- + Gọi các chương trình con trực tiếp của nó gọi được B nhưng không gọi được C
- + Gọi chính nó (gọi đệ quy).
- + Gọi chương trình con cùng cấp nhưng phải khai báo trước gọi được E nhưng không gọi được D , Muốn gọi D phải khai báo trước (khai báo FORWARD)

Để từ thủ tục hàm A có thể gọi đến D là thủ tục hàm cùng cấp nhưng được mô tả sau A, ta cần có một khai báo trước của D ở phía trước của A . Khai báo này gồm : tiêu đề của D, với danh sách thông số của D, tiếp theo là từ khoá FORWARD . Sau đó lúc mô tả lại D thì chỉ cần khai báo từ khoá PROCEDURE (hoặc FUNCTION) , tên của D (không có danh sách thông số) , phần thân của D.

Ví dụ : Với 2 thủ tục gọi đệ quy hỗ trợ nhau FIRST,SECOND sẽ được khai báo như sau :

```

procedure SECOND (i : integer) ; Forward ;
procedure FIRST (n : integer ; var X : real);
  var j, k : integer ;
  begin
    for j := 1 to n do begin
      writeln(' j = ', j) ;
      k := n - 2* j ;
      SECOND( k);
    end ;
  end ;
procedure second ;
  begin
    if (i > 0) then begin
      writeln(' i = ', i);
      FIRST( i - 1 ) ;
    end ;
  end ;
end ;

```



```
{ if(n == 0) return (m);
  else return ( USCLN( n , m mod n));
}
```

b) Đệ quy nhị phân.

Chương trình con đệ quy nhị phân là chương trình con đệ quy trực tiếp có dạng :

```
P ≡ { NẾU thỏa điều kiện dừng thì thực hiện S ;
      Còn không begin { thực hiện S* ; gọi P ; gọi P }
    }
```

Với S , S* là các thao tác không đệ quy .

Ví dụ 1 : Hàm FIBO(n) tính số hạng n của dãy FIBONACCI

+ Dạng hàm trong Pascal:

```
Function F(n : integer) : integer;
begin
  if( n < 2 ) then F := 1
    else F := F(n-1) + F(n-2)
end;
```

+ Dạng hàm trong C++ :

```
int F(int n)
{ if ( n < 2 ) return 1 ;
  else return (F(n -1) + F(n -2));
}
```

c) Đệ quy phi tuyến.

Chương trình con đệ quy phi tuyến là chương trình con đệ quy trực tiếp mà lời gọi đệ quy được thực hiện bên trong vòng lặp .

Dạng tổng quát của chương trình con đệ quy phi tuyến là :

```
P ≡ { for giá trị đầu to giá trị cuối do
      begin thực hiện S ;
            if ( thỏa điều kiện dừng ) then thực hiện S*
            else gọi P
          end ;
    }
```

Với S , S* là các thao tác không đệ quy .

Ví dụ :

Cho dãy $\{ X_n \}$ xác định theo công thức truy hồi :
 $X_0 = 1$; $X_n = n^2 X_0 + (n-1)^2 X_1 + \dots + 2^2 X_{n-2} + 1^2 X_{n-1}$

+ Dạng hàm đệ quy tính X_n trên ngôn ngữ mã giả là :

```
 $X_n \equiv$  if ( n=0 ) then return 1 ;
```

```
else { tg = 0 ;  
      for i = 0 to n-1 do tg = tg + (n-i)2 Xi ;  
      return tg ;  
    }
```

+ Dạng hàm đệ quy tính X_n trên ngôn ngữ Pascal là :

```
function X(n :integer) : integer ;  
  var i , tg : integer ;  
  begin  
    if ( n= 0 ) then X := 1  
    else  
      begin tg = 0 ;  
        for i:= 0 to n-1 do tg := tg + sqr(n-i) *X(i) ;  
        X := tg ;  
      end ;  
    end ;
```

+ Dạng hàm đệ quy tính X_n trên ngôn ngữ C++ là :

```
int X( int n ) ;  
{ if ( n == 0 ) return 1 ;  
  else { int tg = 0 ;  
        for (int i = 0 ; i<n ; i++ ) tg = tg + sqr(n-i) *X(i);  
        return ( tg ) ;  
      }
```

$$THN(1,X,Y,Z) \equiv \{ \text{Move}(X, Z) \}$$

Chú ý : Hoàn toàn tương tự ta cũng có thể quan niệm trường hợp suy biến là trường hợp $n=0$ tương ứng với bài toán $THN(0,X,Y,Z)$: chuyển 0 đĩa từ X sang Z lấy Y làm trung gian mà giải thuật tương ứng là không làm gì cả (thực hiện thao tác rỗng).

$$THN(0,X,Y,Z) \equiv \{ \phi \}$$

c) Phân rã bài toán :

Ta có thể phân rã bài toán $THN(k,X,Y,Z)$: chuyển k đĩa từ cột X sang cột Z lấy cột Y làm trung gian thành dãy tuần tự 3 công việc sau :

+ Chuyển (k - 1) đĩa từ cột X sang cột Y lấy cột Z làm trung gian :

$$THN(k-1,X,Z,Y) \text{ (bài toán THN với } n = k-1, X=X, Y=Z, Z=Y \text{)}$$

+ Chuyển 1 đĩa từ cột X sang cột Z : $\text{Move}(X, Z)$ (thao tác cơ bản).

+ Chuyển (k - 1) đĩa từ cột Y sang cột Z lấy cột X làm trung gian :

$$THN(k-1,Y,X,Z) \text{ (bài toán THN với } n = k-1, X=Y, Y=X, Z=Z \text{)}.$$

Vậy giải thuật trong trường hợp tổng quát ($n > 1$) là :

$$THN(n,X,Y,Z) \equiv \{ \begin{array}{l} THN(n-1,X,Z,Y); \\ \text{Move}(X, Z); \\ THN(n-1,Y,X,Z); \\ \end{array} \}$$

Với n đĩa thì cần bao nhiêu bước chuyển 1 đĩa? Thực chất trong thủ tục THN các lệnh gọi đệ qui chỉ nhằm sắp xếp trình tự các thao tác chuyển 1 đĩa

Số lần chuyển 1 đĩa được thực hiện là đặc trưng cho độ phức tạp của giải thuật .

Với n đĩa , gọi $f(n)$ là số các thao tác chuyển _một_ đĩa .

$$\text{Ta có : } f(0) = 0 .$$

$$f(1) = 1 .$$

$$f(n) = 2f(n-1) + 1 \quad \text{với } n > 0$$

$$\text{Do đó : } f(n) = 1 + 2 + 2^2 + \dots + 2^{n-1} = 2^n - 1$$

Để chuyển 64 đĩa cần $2^{64} - 1$ bước hay xấp xỉ 10^{20} bước . Cần khoảng 10 triệu năm với một máy tính nhanh nhất hiện nay để làm việc này .

d) Chương trình con mã hóa giải thuật THN trong NNLT Pascal :

procedure THN (n : integer ; X,Y,Z : char)

begin

if n > 0 then begin

THN (n-1 ,X,Z,Y) ;

Move(X, Z);

THN (n-1 ,Y,X,Z);

end ;

end ;

(Lấy trường hợp chuyển n = 0 làm trường hợp neo)

a) Thông số hóa.

Ta sẽ giải bài toán ở mức độ tổng quát : Tìm số cách chia m vật (phần thưởng) cho n đối tượng (học sinh) có thứ tự.

Gọi PART là số cách chia khi đó PART là hàm của 2 biến nguyên m, n (PART(m, n)).

Ta mã hoá n đối tượng theo thứ tự xếp hạng 1, 2, 3, ... n; Si là số phần thưởng mà học sinh i nhận được.

Khi đó các điều kiện ràng buộc lên cách chia là :

$$\begin{aligned} S_i &\geq 0 \\ S_1 &\geq S_2 \geq \dots \geq S_n \\ S_1 + S_2 + \dots + S_n &= m \end{aligned}$$

Ví dụ :

Với m = 5, n = 3 ta có 5 cách chia sau :

$$\begin{array}{ccc} 5 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & 2 & 0 \\ 3 & 1 & 1 \\ 2 & 2 & 1 \end{array}$$

Tức là PART(5,3) = 5

b) Các trường hợp suy biến :

+ m = 0 thì sẽ có duy nhất 1 cách chia : mọi học sinh đều nhận được 0 phần thưởng.

Vậy : PART(0, n) = 1 với mọi n

+ n = 0, m <> 0 thì sẽ không có cách nào để thực hiện việc chia.

Vậy : PART(m, 0) = 0 với mọi m <> 0.

(ta có thể thay trường hợp neo PART(m, 0) = 0 hoặc trường hợp neo PART(m, 1) = 1)

c) Phân rã bài toán trong trường hợp tổng quát :

+ m < n khi số phần thưởng m nhỏ hơn số học sinh n thì n - m học sinh xếp cuối sẽ luôn không nhận được gì cả trong mọi cách chia.

Vậy :

$$\text{khi } n > m \text{ thì } PART(m, n) = PART(m, m)$$

+ Trong trường hợp m >= n : số vật chia (phần thưởng) lớn hơn hoặc bằng số học sinh (đối tượng) ta phân các cách chia làm 2 nhóm :

* Nhóm thứ nhất không dành cho học sinh xếp cuối cùng phần thưởng nào cả

(S_n = 0). Số cách chia này sẽ bằng số cách chia m phần thưởng cho n - 1 học sinh.

Tức là : Số cách chia trong nhóm thứ nhất = PART(m, n - 1).

2	1	3	4	2	1	4	3	4	1	3	2	2	4	3	1
1	3	2	4	1	4	2	3	1	3	4	2	4	3	2	1
3	1	2	4	4	1	2	3	3	1	4	2	3	4	2	1
3	2	1	4	4	2	1	3	3	4	1	2	3	2	4	1
2	3	1	4	2	4	1	3	4	3	1	2	2	3	4	1

a) Thông số hóa bài toán .

Gọi HV(v, m) (với v : array[1 .. N] of T, m :integer ; m ≤ N ; T là một kiểu dữ liệu đã biết trước) là thủ tục xuất tất cả các dạng khác nhau của v có được bằng cách hoán vị m thành phần đầu của dãy v

Ví dụ : N = 4 , A[1] = 1 , A[2] = 2 , A[3] = 3 , A[4] = 4 thì lời gọi HV(A, 3) xuất tất cả hoán vị của A có được bằng cách hoán vị 3 phần tử đầu (có 6 h vị) :

1	2	3	4	1	3	2	4	3	2	1	4
2	1	3	4	3	1	2	4	2	3	1	4

Để giải bài toán đặt ra ban đầu ta gọi HV(A,N)).

b) Trường hợp neo.

Với m = 1 : HV(v,1) là thủ tục giải bài toán xuất tất cả các dạng của v có được bằng cách hoán vị 1 phần tử đầu . Vậy HV(v,1) là thủ tục xuất v.

HV(v,1) ≡ print(v) ≡ for k:= 1 to N do write(v[k])

c) Phân rã bài toán.

Ta có thể tìm hết tất cả các hoán vị m phần tử đầu của vector V theo cách sau :

- Giữ nguyên các phần tử cuối V[m] , . . . ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

- Đổi chỗ V[m] cho V[m-1] ,giữ nguyên các phần tử cuối V[m],... ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

- Đổi chỗ V[m] cho V[m-2] ,giữ nguyên các phần tử cuối V[m],.... ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

.....

- Đổi chỗ V[m] cho V[2] ,giữ nguyên các phần tử cuối V[m], . . . ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

- Đổi chỗ V[m] cho V[1] ,giữ nguyên các phần tử cuối V[m], . . . ,V[N] hoán vị m-1 phần tử đầu (gọi đệ quy HV(V ,m - 1) .

Vậy :

HV(V,m) ≡ { SWAP(V[m],V[m]) ; HV(V,m - 1) ;
 SWAP(V[m],v[m-1]) ; HV(V,m - 1) ;
 SWAP(V[m],v[m-2]) ; HV(V,m - 1) ;

```

.....
.....
SWAP( V[m],v[2] ) ; HV(V,m - 1) ;
SWAP( V[m],v[1] ) ; HV(V,m - 1) ;
}
( SWAP(x , y ) là thủ tục hoán đổi giá trị của 2 đối tượng dữ liệu x , y )

```

Vậy :

```

HV(V , m ) ≡ for k := m downto 1 do begin
                    SWAP( V[m], V[k] ) ;
                    HV(V,m - 1) ;
                end ;

```

d) Thủ tục hoán vị trên NNLT Pascal.

```

.....
const size = Val ; (* Val là hằng giá trị *)
type vector = array[1..size] of typebase ; (* typebase là một kiểu dữ liệu có thứ
tự *)

```

```

.....
procedure Swap( var x , y : typebase ) ;
var t : typebase ;
begin
    t := x ; x := y ; y := t ;
end ;
.....
procedure print( A : vector ) ;
var i : integer ;
begin
    for i:= 1 to size do write( A[i] : 3 ) ;
    writeln ;
end ;
.....

```

```

Procedure HV( V : vector ; m :integer ) ;
var k : integer ;
begin
    if ( m = 1 ) then print(V)
    else
        for k := m downto 1 do begin
            Swap(V[m] , V[k]);
            HV(V , m - 1) ;
        end ;
end ;

```

a) Thông số hoá:

Xét hàm ROOT với 3 thông số là g, a, b , (ROOT(g, a, b)) trả về giá trị nghiệm xấp xỉ ε của phương trình $g(x) = 0$ trên đoạn $[a, b]$ hoặc giá trị C nếu phương trình xét không có nghiệm xấp xỉ. Để giải bài toán ban đầu ta gọi hàm ROOT(f, a_0, b_0).

b) Trường hợp tầm thường:

đó là khi $b - a < \varepsilon$.

Khi đó :

```
if ( g(a).g(b) ) <= 0 then ROOT(g,a,b) = a ; // a là nghiệm xấp xỉ
    else ROOT(g,a,b) = E ; // không có nghiệm xấp xỉ
```

c) Phân rã trường hợp tổng quát:

khi $b - a \geq \varepsilon$ ta phân $[a, b]$ làm 2 đoạn $[a, c]$ và $[c, b]$ với $c = (a + b) / 2$.

- Nếu $ROOT(g, a, c) < E$ thì $ROOT(g, a, b) = ROOT(g, a, c)$ (bài toán tìm nghiệm trên đoạn $[a, c]$).

- còn không thì $ROOT(g, a, b) = ROOT(g, c, b)$ (bài toán tìm nghiệm trên đoạn $[c, b]$).

d) Hàm tìm nghiệm xấp xỉ trên NN Pascal có dạng:

```
const epsilon = ;
    E = ;
Function ROOT(a,b :real) :real ;
var c, R : real ;
begin
    if ((b-a) < epsilon) then if ( f(a)*f(b) <= 0 ) then ROOT := a
        else ROOT := L
    else
        begin
            c := (a + b)/2 ;
            if ( ROOT(a, c) < E ) then ROOT := ROOT(a,c)
                else ROOT := ROOT(c,b)
        end;
end;
```

e) Chương trình con tìm nghiệm xấp xỉ trong NN LT C++

```
const double epsilon = ;
const double E = ;
double ROOT(double a, double b)
{ if((b - a) < epsilon) if(f(a)*f(b) <= epsilon return (a) ;
    else return ( L) ;
else
{ double c = (a + b) / 2 ;
```

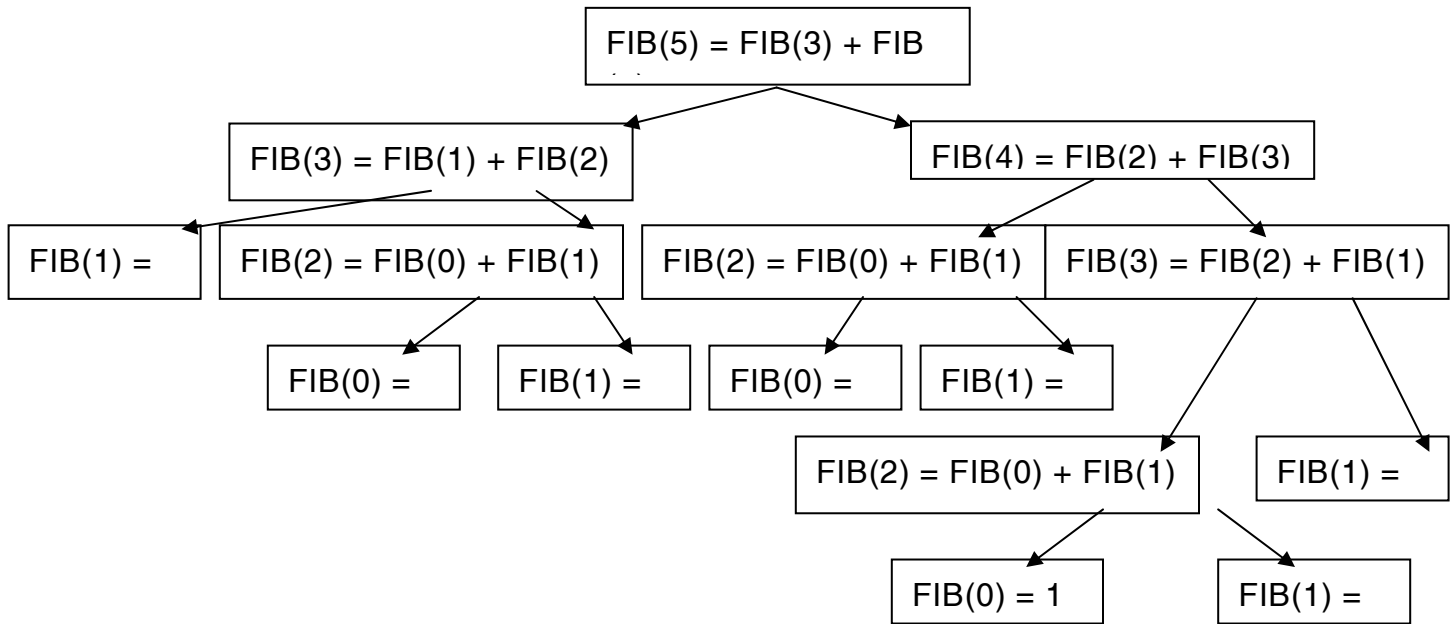
```
double R = ROOT(a,c);  
if( R< E ) return R ;  
    else return ( ROOT(c , b) );  
    }  
}
```


Đồng thời với quá trình xử lý ngược là quá trình xóa bỏ các thông tin về giải thuật xử lý trung gian (quá trình thu hồi vùng nhớ).

b) Xét giải thuật đệ quy tính giá trị hàm FIBONACCI .

```
FIB(n) ≡ if ((n = 0) or (n = 1)) then return 1 ;
           else return ( FIB(n - 1) + FIB(n - 2) ) ;
```

Sơ đồ tính FIB(5) :



c) Xét thủ tục đệ quy tháp Hà Nội THN (n, X, Y, Z)

```
THN (n : integer ; X , Y , Z : char)
≡ if (n > 0) then
    { THN(n-1,X,Z,Y) ;
      Move(X,Z) ;
      THN(n-1,Y,X,Z) ;
    }
```

Để chuyển 3 đĩa từ cột A sang cột C dùng cột B làm trung gian ta gọi : THN (3,A,B,C)

Sơ đồ thực hiện lời gọi THN (3,A,B,C) là :

<u>Lời gọi c/0</u>	<u>Lời gọi c/1</u>	<u>Lời gọi c/2</u>	<u>Lời gọi c/3</u>
			THN(0,A,C,B) A ---> C THN(0,B,A,C)
	THN(2,A,C,B)	A ---> B THN(1,C,A,B)	THN(0,C,B,A) C --->B THN(0,A,C,B)
THN(3,A,B,C)	A ---> C	THN(1,B,C,A)	THN(0,B,A,C) B ---> A THN(0,C,B,A)
	THN(2,B,A,C)	B ---> C THN(1,A,B,C)	THN(0,A,C,B) A ---> C THN(0,B,A,C)

Với THN(0 ,X , Y , Z) là trường hợp neo tương ứng với thao tác rỗng .

X -----> Y là thao tác chuyển 1 đĩa từ cột X sang cột Y (MOVE(X,Y)).

Các bước chuyển đĩa sẽ là :

A --> C ; A --> B ; C --> B ; A --> C ; B --> A ; B --> C ; A --> C ;

Lời gọi cấp 0 :

THN(3 , A , B , C) sẽ làm nảy sinh hai lời gọi cấp 1 : THN (2 ,A, C, B) ;

THN (2 , B , A , C) cùng với các thông tin của quá trình xử lý còn dang dở .

Các lời gọi cấp 1 :

THN(2 , A , C , B) , THN (2 , B , A , C) sẽ làm nảy sinh các lời gọi cấp 2 : THN (1 ,A, B, C) ; THN (1, C , A , B) ; THN (1 ,B, C, A) ; THN (1, A , B , C) ; cùng với các thông tin của quá trình xử lý còn dang dở .

Các lời gọi cấp 2 :

THN(1 ,A, B, C) ; THN(1, C , A , B) ; THN(1 ,B, C, A) ; THN(1, A , B , C) ; sẽ làm nảy sinh các lời gọi cấp 3 dạng : THN(0 ,X, Y, Z) (thao tác rỗng tương ứng với trường hợp suy biến) ; cùng với các thông tin của quá trình xử lý còn dang dở .

Quá trình gọi dừng lại khi gặp trường hợp suy biến .

Quá trình xử lý ngược với quá trình gọi bắt đầu khi thực hiện xong các trường hợp neo nhằm hoàn thiện các bước xử lý con dang dở song song với quá trình hoàn thiện các lời gọi là quá trình loại bỏ các lưu trữ thông tin giải thuật trung gian.

Do đặc điểm của quá trình xử lý một giải thuật đệ quy là : việc thực thi lời gọi đệ quy sinh ra lời gọi đệ quy mới cho đến khi gặp trường hợp suy biến (neo) cho nên để thực thi giải thuật đệ quy cần có cơ chế lưu trữ thông tin thỏa các yêu cầu sau :

+ Ở mỗi lần gọi phải lưu trữ thông tin trạng thái con dang dở của tiến trình xử lý ở thời điểm gọi. Số trạng thái này bằng số lần gọi chưa được hoàn tất .

+ Khi thực hiện xong (hoàn tất) một lần gọi, cần khôi phục lại toàn bộ thông tin trạng thái trước khi gọi .

+ Lệnh gọi cuối cùng (ứng với trường hợp neo) sẽ được hoàn tất đầu tiên , thứ tự dãy các lệnh gọi được hoàn tất ngược với thứ tự gọi, tương ứng dãy thông tin trạng thái được khôi phục theo thứ tự ngược với thứ tự lưu trữ .

Cấu trúc dữ liệu cho phép lưu trữ dãy thông tin thỏa 3 yêu cầu trên là cấu trúc lưu trữ thỏa luật LIFO (Last In First Out) . Một kiểu cấu trúc lưu trữ thường được sử dụng trong trường hợp này là cấu trúc chồng (stack).

Với một chồng S thường cho phép chúng ta thực hiện các thao tác sau trên nó :

- Thủ tục Creatstack(S) : Tạo chồng S rỗng .
- Thủ tục Push(x,S) : Lưu trữ thêm dữ liệu x vào đỉnh stack S
(x là dữ liệu kiểu đơn giản hoặc có cấu trúc)
- Thủ tục Pop(x,S) : Lấy giá trị đang lưu ở đỉnh S chứa vào trong đối tượng dữ liệu x và loại bỏ giá trị này khỏi S (lùi đỉnh S xuống một mức) .
- Hàm Empty(S) : (kiểu boolean) Kiểm tra tính rỗng của S : cho giá trị đúng nếu S rỗng , sai nếu S không rỗng .

Cài đặt cụ thể của S có thể thực hiện bằng nhiều phương pháp phụ thuộc vào từng ngôn ngữ lập trình và từng mục đích sử dụng cụ thể .

Ví dụ :

Cài đặt (bằng cấu trúc mảng) chồng S mà mỗi phần tử là một đối tượng dữ liệu thuộc kiểu T trong PASCAL như sau :

```
Const sizestack = ... ;
Type stackType = record
    St : array [1 .. sizestack] of T ;
    Top : 0 .. sizestack ;
end ;
```

Thủ tục Creatstack(S) : tạo chồng S rỗng :

```
Procedure Creatstack( var S : StackType )
Begin
    S.Top := 0 ;
End;
```

Thủ tục Push(x,S) : Chèn - Lưu trữ thêm dữ liệu x vào đỉnh stack S
(x là dữ liệu kiểu đơn giản hoặc có cấu trúc)

```
Procedure Push( var S : StackType ; x : T ) ;
Begin
```



```

                k := k + 1 ;
                F := F * k ;
            end ;
        FAC := F ;
    end ;

```

hoặc :

```

Function FAC ( n : integer ) : longint ;
var k : integer ;
    F : longint ;
Begin
    F := 1 ;
    For k:= 1 to n do F := F * k ;
    FAC := F ;
end ;

```

+ Trong NN LT C++

```

long int FAC ( int n )
{ int k = 0 ;
  long int F = 1 ;
  while ( k < n ) F = ++k * F ;
  return (F) ;
}

```

Hoặc :

```

long int FAC ( int n )
{ long int F = 1 ;
  for ( int k = 1 ; k <= n ; k++) F = k * F ;
  return (F) ;
}

```

Ví dụ 2 : Dạng hàm S_n không đệ quy

+ trên NN LT Pascal :

```

Function S(n : integer ) : integer ;
var k ,tg : integer ;
Begin
    k := 1 ; tg := 1 ;
    while ( k < n ) do begin
        k := k + 1 ;
        if odd (k) then tg := tg + (2 * k - 1 )
        else tg := tg - (2 * k - 1 ) ;
    end ;
    S := tg ;
end ;

```

```

+ Trong NN LT C++
int S ( int n )
{ int k = 1 , tg = 1 ;
  while ( k < n ) { k ++ ;
                    if (k%2) tg + = 2 * k - 1 ;
                    else   tg - = 2 * k + 1 ;
                  }
  return ( tg ) ;
}

```

b) Các thủ tục đệ qui dạng đệ qui đuôi.

Xét thủ tục P dạng :

$$P(X) \equiv \begin{cases} \text{if } B(X) \text{ then } D(X) \\ \text{else } \{ A(X) ; \\ \quad P(f(X)) ; \\ \} \end{cases}$$

Trong đó : X là tập biến (một hoặc một bộ nhiều biến).

$P(X)$ là thủ tục đệ quy phụ thuộc X

$A(X) ; D(X)$ là các nhóm thao tác (lệnh) không đệ quy

$f(X)$ là hàm biến đổi X

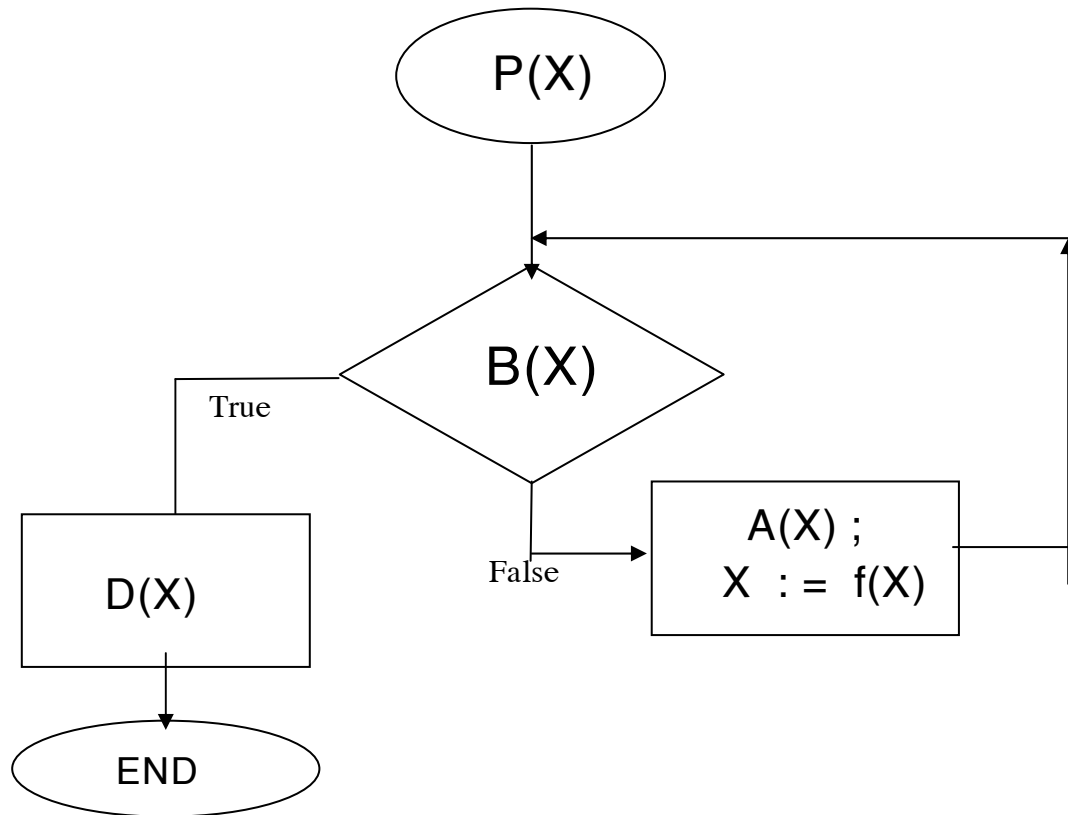
Xét quá trình thi hành $P(X)$:

gọi P_0 là lần gọi P thứ 0 (đầu tiên)	$P(X)$
P_1 là lần gọi P thứ 1 (lần 2)	$P(f(X))$
P_i là lần gọi P thứ i (lần $i + 1$)	$P(f(f(\dots f(X)\dots))$ ($P(f_i(X))$ hợp i lần hàm f)

Trong lần gọi P_i nếu $B(f_i(X))$ không đúng (false) thì thi hành lệnh A và gọi P_{i+1} ; nếu $B(f_i(X))$ đúng (true) thì thi hành lệnh D và kết thúc quá trình gọi .

Giả sử P được gọi đúng $n + 1$ lần . Khi đó ở trong lần gọi cuối cùng (thứ n) P_n thì $B(f_n(X))$ đúng , lệnh D được thi hành và chấm dứt thao tác gọi thủ tục P .

Sơ đồ khối quá trình thực hiện lệnh gọi thủ tục $P(X)$ có dạng sau :



Tương ứng với vòng lặp sau :

```

While ( not B(X) ) do begin
    A(X) ;
    X := f(X) ;
end ;
D(X) ;
  
```

Ví dụ 1 :

Để đổi 1 số nguyên không âm y ở cơ số 10 sang dạng cơ số k ($2 \leq k \leq 9$) với việc dùng mảng A ($A : \text{array}[1 \dots \text{size}] \text{ of } 0..k-1$, size là một hằng được khai báo trước) để chứa các ký số trong hệ k phân (với quy ước ký số có ý nghĩa thấp được chứa ở chỉ số cao) khi đó thủ tục đệ quy $\text{Convert}(x,m)$ để tạo dãy giá trị : $A[0], A[1], \dots, A[m]$ như sau (hãy tự giải thích) :

```

Convert(n,m) ≡ if n <> 0 then Begin
    A[m] := n mod k ;
    Convert(n div k, m-1) ;
End ;
  
```

Lệnh gọi $\text{Convert}(y,n)$ dùng để đổi số nguyên y trong cơ số 10 sang cơ số k lưu
dãy

ký số trong mảng A ;

Trong ví dụ này ta có :

X là (n, m) ;

$B(X)$ là biểu thức boolean $\text{not}(n < 0)$

$A(X)$ là lệnh gán $A[m] := n \bmod k$;

$f(X)$ là hàm $f(n,m) = (n \text{ div } k, m - 1)$;

$D(X)$ là lệnh rỗng

Đoạn lệnh lặp tương ứng với thủ tục $\text{Convert}(x,m)$ là :

While $(n < 0)$ **then begin**

$A[m] := n \bmod k$; { $A(X)$ }

$n := n \text{ div } k$; { $X := f(X)$ }

$m := m - 1$;

end ;

Ví dụ 2 :

Tìm USCLN của 2 số nguyên dựa vào thuật toán Euclide .

- Giải thuật đệ quy (dưới dạng thủ tục) tìm USCLN(m,n) bằng thuật toán Euclide :

$\text{USCLN}(m, n, \text{var } us) \equiv \text{if } (n = 0) \text{ then } us := m$
 $\text{else } \text{USCLN}(n, m \bmod n, us)$;

- Trong trường hợp này thì :

X là (m, n, us)

$P(X)$ là $\text{USCLN}(m, n, us)$

$B(X)$ là $n = 0$

$D(X)$ là lệnh gán $us := m$

$A(X)$ là lệnh rỗng

$f(X)$ là $f(m,n,us) = (n, m \bmod n, us)$

- Đoạn lệnh lặp tương ứng là :

While $(n < 0)$ **do begin**

$sd := m \bmod n$;

$m := n$;

$n := sd$;

end ;

$us := m$;

- Thủ tục không đệ quy tương ứng trong Pascal .


```

Procedure USCLN(m , n : integer ; var us : integer ) ;
  var sd : integer ;
  begin
    while ( n <> 0 ) do begin
      sd := m mod n ;
      m := n ;
      n := sd ;
    end ;
    us := m ;
  end ;

```

- Hàm con không đệ quy tương ứng trong C++

```

void USCLN(int m , int n , int& us )
  { while(n != 0 ) { int sd = m % n ;
    m = n ;
    n = sd ;
  }
  us = m ;
}

```

c) Các hàm đệ qui dạng đệ qui đuôi (tail-recursive).

Xét hàm đệ qui dạng :

$$f(X) = \begin{cases} f(g(X)) & \text{khi } C(X) \text{ đúng} \\ a(X) & \text{khi } C(X) \text{ sai} \end{cases}$$

Tức là :

$$f(X) \equiv \text{if}(C(X)) \text{ then return}(f(g(X))) \\ \text{else return}(a(x))$$

Tính $f(X_0)$.

Ta có :

$$\begin{aligned}
 f(X_0) &= f(g(X_0)) && \text{với } C(X_0) \text{ đúng.} \\
 &= f(g(g(X_0))) && \text{với } C(g(X_0)) \text{ đúng.} \\
 &= \dots \\
 &= f(g^k(X_0)) && \text{với } C(g^{k-1}(X_0)) \text{ đúng.} \\
 &= a(g^k(X_0)) && \text{với } C(g^k(X_0)) \text{ sai.}
 \end{aligned}$$

$$(g^k(x_0) = g(g(g(x_0))))$$

Đặt : $U_0 = X_0 = g^0(X_0)$

$$U_i = g^i(X_0) = g(g^{i-1}(X_0)) = g(U_{i-1}) \text{ với } i \geq 1$$

Ta có quan hệ sau :

$$U_0 = X_0$$

$$U_i = g(U_{i-1}) \quad i = 1 \dots k. \text{ Với } k \text{ là số nhỏ nhất mà } C(U_k) \text{ sai.}$$

Lúc đó : $f(X_0) = a(U_k)$

Vậy đoạn chương trình tính $f = f(X_0)$ là :

```

U := X0 ;
while C(U) do U := g(U) ;
f := a(U) ;
    
```

Ví dụ :

Với $m, n \geq 0$ ta có hàm đệ quy tính USCLN(m,n) là :

```

USCLN(m,n) ≡ if (m <> 0) then return(USCLN ( abs(m - n) , min(m , n) ) ;
                else return n ;
    
```

Trong trường hợp này :

X là (m, n) ;

$C(X) = C(m, n)$ là $m \neq 0$;

$g(X) = g(m, n) = (abs(m - n) , min(m, n))$;

$a(x) = a(m, n) = n$;

- Đoạn chương trình tính USCLN(a, b) là :

```

m := a ; n := b ;
while ( m <> 0 ) do begin
    t1 := m ;
    t2 := n ;
    m := abs(t1 - t2) ;
    n := min(t1,t2) ;
end ;
    
```

USCLN := n ;

- Hàm không đệ quy tương ứng trong Pascal.

```

Function USCLN(m , n : integer) : integer ;
var t1 , t2 : integer ;
begin
    while (n <> 0) do begin t1 := m ; t2 := n ;
        m := abs(t1 - t2) ;
        if(t1 < t2) then n := t1
            else n := t2 ;
    end ;
    USCLN := m ;
end ;
    
```

- Dạng hàm tương ứng trong C++

```

int USCLN(int m , int n)
{ while( n != 0) { int t1 = m ; int t2 = n ;
    
```

$$\begin{aligned}
 &= DS(A(U_{i+1}),FS(U_{i+1},U_i)) \\
 &= DS(V_{i+1},FS(U_{i+1},U_i)) \quad \text{với } 0 < i < k \quad (\text{vì } C(U_i) \text{ đúng}) \\
 V_k &= BS(U_k) \quad (\text{vì } C(U_k) = \text{false})
 \end{aligned}$$

Dựa vào 2 dãy số $\{U_i\}, \{V_i\}$ (mô tả bởi (2.1) và (2.2)) ta tính $A(X)$ theo giải thuật sau :

- Tính và ghi nhớ các U_i từ 0 đến k theo (2.1).
- (Với $C(U_0) = C(U_1) = \dots = C(U_{k-1}) = \text{True}$, $C(U_k) = \text{False}$)
- Sử dụng dãy giá trị U_i để tính lần ngược V_i từ k xuống 0 theo (2.2) , V_0 chính là giá trị cần tính ($V_0 = A(X)$).

c) Giải thuật không đệ quy tính giá trị hàm Arzac bằng sử dụng cấu trúc Stack .

Để thực hiện giải thuật trên thì dãy U_i phải được tính và lưu trữ trong một cấu trúc dữ liệu thích hợp , để khi cần đến (khi tính V_i) để lấy ra sử dụng . Đặc điểm quan trọng của dãy U_i là thỏa luật LIFO : thứ tự sử dụng ngược với thứ tự tạo sinh . Cấu trúc dữ liệu cho phép lưu trữ thuận lợi dãy phần tử thỏa luật LIFO (vào sau ra trước - Last In First Out) là cấu trúc Stack .

(Trên cấu trúc Stack 2 thao tác cơ bản đặc trưng là :

+ Push(S,X) : Chèn phần tử dữ liệu X vào đỉnh Stack S .

+ Pop(S,X) : Lấy ra khỏi stack S phần tử dữ liệu ở đỉnh và chứa nó

vào biến X).

Giải thuật không đệ quy tính $V_0 = A(X)$ dựa trên 2 công thức (2.1) , (2.2) và sử dụng Stack S là :

+ Bước 1 : tính U_i bắt đầu từ U_0 theo (2.1) lưu vào Stack S

CreateStack(S) ; (tạo stack rỗng S)

k := 0 ;

U := X ; ($U_0 = X$)

push(S,U) ; (chèn U_0 vào đỉnh stack S)

while C(U) do begin

k := k+1 ;

U := CS(U) ; ($U_{k+1} = CS(U_k)$)

push (S,U) ; (chèn U_{k+1} vào đỉnh Stack S)

end ;

+ Bước 2 : Lấy dữ liệu trong Stack S tính V_i theo (2.2)

pop(S,U) ; ($U = U_k$)

V := BS(U) ; ($C(U_k)$ sai ; $V = V_k = BS(U_k)$)

for i := k-1 downto 0 do

begin

Y := U ;

($Y = U_{i+1}$)

```

        pop(S,U) ;          ( U = Ui )
        V := DS(V,FS(Y,U)) ; ( C(Ui) đúng ; Vi = DS(Vi+1,FS(Ui+1,Ui)) )
    end ;
    { V = A(X0) }

```

Hoặc :

+ Bước 1 : tính U_i bắt đầu từ U₀ theo (2.1) lưu vào Stack S

```

CreateStack(S) ; ( tạo stack rỗng S )
U := X0 ; ( U0 = X0 )
push(S,U) ; ( chèn U0 vào đỉnh stack S )
while C(U) do begin
    U := CS(U) ; ( Uk+1 = CS(Uk) )
    push (S,U) ; ( chèn Uk+1 vào đỉnh Stack S )
end ;

```

+ Bước 2 : Lấy dữ liệu trong Stack S tính V_i theo (2. 2)

```

pop(S,U) ; ( U = Uk )
V := BS(U) ; ( C(Uk) sai ; V=Vk = BS (Uk) )
While(not emptystack(S)) do
    begin
        Y := U ; ( Y = Ui+1 )
        pop(S,U) ; ( U = Ui )
        V := DS(V,FS(Y,U)) ; ( C(Ui) đúng ; Vi =
DS(Vi+1,FS(Ui+1,Ui)) )
    end ;
    { V = A(X0) }

```

Cơ chế lưu trữ dãy dữ liệu LIFO bằng Stack là một đặc trưng của quá trình xử lý giải thuật đệ quy điều cần quan tâm là cấu trúc stack thường chiếm nhiều bộ nhớ . Vì vậy người ta luôn tìm cách tránh dùng nó khi còn tránh được .

d) Một số hàm Arzac đặc biệt mà việc khử đệ quy giải thuật tính giá trị hàm có thể không dùng Stack .

d1) Trường hợp thủ tục CS là song ánh .

Trường hợp CS là song ánh từ miền D lên miền D thì hàm CS có hàm ngược CS⁻¹ . Gọi hàm ngược của hàm CS là hàm CSM1 .

Ta có : CSM1(CS(X)) = CS⁻¹(CS(X)) = X với $\forall X \in D$.

Nên : CSM1(U_{i+1}) = CS⁻¹(CS(U_i)) = U_i với $i = k-1, \dots, 1, 0$

Khi đó ta không cần lưu giữ các giá trị trung gian của dãy { U_i } mà chỉ cần xuất phát từ U_k dùng hàm CSM1 để khôi phục lại các giá trị U_i với $i < k$.

Giải thuật tính A(X) sẽ trở thành :

+ Bước 1 : Dựa vào (2.1) tính U_k

```

U := X ; ( U0 = X )
k := 0 ;
while C(U) do begin
    k := k+1 ;
    U := CS(U) ; ( Uk+1 = CS(Uk) )
end ;

```

+ Bước 2 : Tính $V_k, V_{k-1}, \dots, V_1, V_0$ dựa vào U_k ,(2.2) và CSM1

```

V := BS(U) ; ( V = Vk = BS (Uk) )
for i := k-1 downto 0 do begin
    Y := U ; ( Y = Ui+1 )
    U := CSM1(U) ; ( Ui = CSM1(Ui+1) )
    V := DS(V,FS(Y,U)) ;
    ( Vi = DS(Vi+1,FS(Ui+1,Ui) )
end ;
{ V = V0 = A(X) }

```

d2) Trường hợp thủ tục DS có tính hoán vị .

Xét công thức tính :

$$V_i = DS(V_{i+1}, FS(U_{i+1}, U_i)) \quad \text{với mọi } i < k$$

Đặt : $U'_i = FS(U_{i+1}, U_i)$
 $DS(V_{i+1}, U'_i) = V_{i+1} T U'_i$

Ta có :

$$V_0 = DS(V_1, FS(U_1, U_0)) = DS(V_1, U'_0) = V_1 T U'_0$$

$$V_1 = DS(V_2, FS(U_2, U_1)) = DS(V_2, U'_1) = V_2 T U'_1$$

$$V_2 = DS(V_3, FS(U_3, U_2)) = DS(V_3, U'_2) = V_3 T U'_2$$

.....

$$V_i = DS(V_{i+1}, FS(U_{i+1}, U_i)) = DS(V_{i+1}, U'_i) = V_{i+1} T U'_i \quad (3-1)$$

.....

$$V_{k-1} = DS(V_k, FS(U_k, U_{k-1})) = DS(V_k, U'_{k-1}) = V_k T U'_{k-1}$$

$$V_k = BS(U_k)$$

Khi DS có tính hoán vị tức : $DS(DS(x,y),z) = DS(DS(x,z),y)$

(Viết trên ký hiệu T : $(x T y) T z = (x T z) T y$)

Thực hiện thế lần lượt V_1 rồi $V_2 \dots$ trong công thức V_0 .

Ta có :

```

else begin
    A(X) ; P(f(X)) ; B(X) ;
end ;
    
```

Với :

X là một biến đơn hoặc biến véc tơ.

C(X) là một biểu thức boolean của X .

A(X) , B(X) , D(X) là các nhóm lệnh không đệ quy (không chứa lệnh gọi đến P).

f(X) là hàm của X .

Tiến trình thực hiện thủ tục P(X) sẽ là :

+ Nếu C(X) đúng thì thực hiện D(X) .

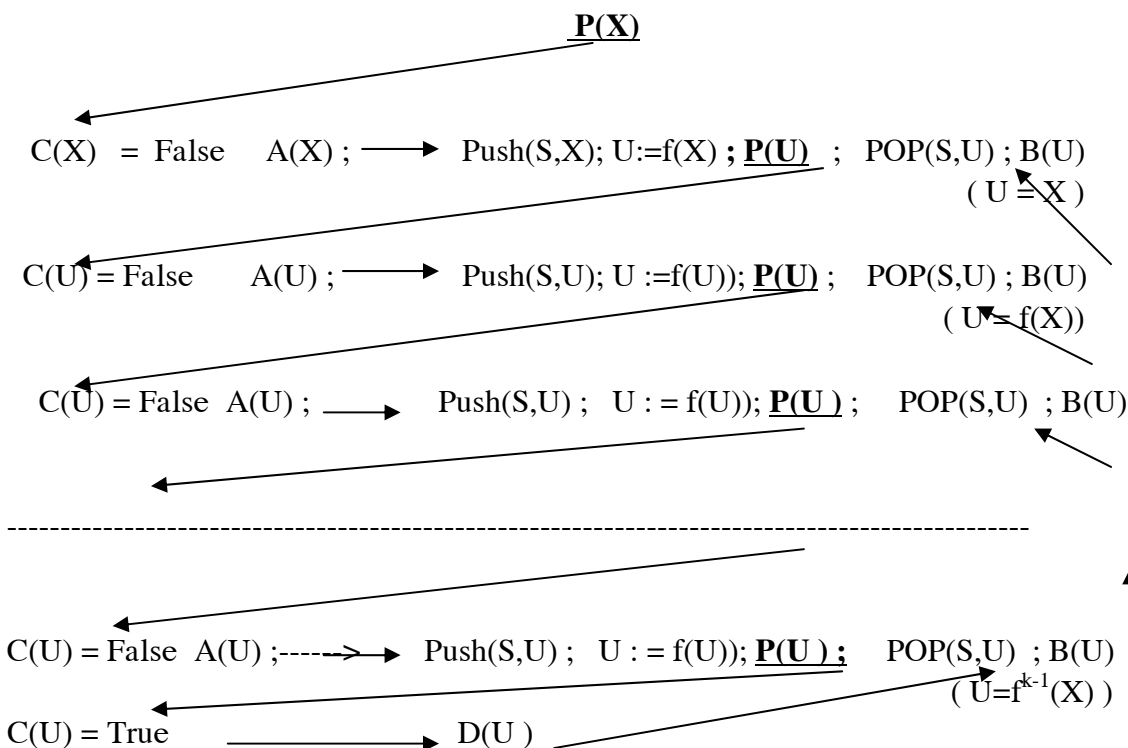
+ Còn không (C(X) sai) thì thực hiện A(X) ; gọi P(f(X)) ; thực hiện B(X) . (B(X) chỉ được thực hiện khi P(f(X)) thực hiện xong) .

Mỗi lần thành phần đệ quy P(Y) được gọi thì thông tin giải thuật B(Y) lại được sinh ra (nhưng chưa thực hiện) .

Giả sử quá trình đệ quy kết thúc sau k lần gọi đệ quy thì ta phải thực hiện một dãy k thao tác B theo thứ tự : B(f^{k-1}(X)) , B(f^{k-2}(X)) , . . . ,B(f(f(X))) ,B(f(X)),B(X).

Để thực hiện dãy thao tác { B(fⁱ(X)) } theo thứ tự ngược với thứ tự phát sinh ta cần dãy dữ liệu {fⁱ(X)} truy xuất theo nguyên tắc LIFO. Ta sẽ dùng một Stack để lưu trữ dãy { fⁱ(X) } ≡ { X , f(X) , f(f(X)) , . . . , f^{k-1}(X) }

Trình tự thực hiện P(X) được diễn tả bằng mô hình sau :



Giải thuật thực hiện P(X) với việc sử dụng Stack có dạng :

```

P(X)  ≡  {  Creat_Stack (S) ; ( tạo stack S )
          While(not(C(X)) do begin A(X) ;
                                Push(S,X) ; ( cất giá trị X vào stack S )
                                X := f(X) ;
                                end ;
          D(X) ;
          While(not(EmptyS(S))) do begin
                                POP(S,X) ; ( lấy dữ liệu từ S )
                                B(X) ;
                                end ;
          }
    
```

Ví dụ :

Thủ tục đệ quy chuyển biểu diễn số từ cơ số thập phân sang nhị phân có dạng :

```

Binary(m)  ≡  if ( m > 0 ) then begin
                Binary( m div 2 ) ;
                write( m mod 2 ) ;
            end;
    
```

Trong trường hợp này :

X là m .
 P(X) là Binary(m) .
 A(X) ; D(X) là lệnh rỗng .
 B(X) là lệnh Write(m mod 2) ;
 C(X) là (m ≤ 0) .
 f(X) = f(m) = m div 2 .

Giải thuật thực hiện Binary(m) không đệ quy là :

```

Binary ( m )  ≡  {  Creat_Stack (S) ;
                  While ( m > 0 ) do begin
                                sdu := m mod 2 ;
                                Push(S,sdu) ;
                                m := m div 2 ;
                                end;
                  While( not(EmptyS(S)) do begin
                                POP(S,sdu) ;
                                Write(sdu) ;
                                end;
                  }
    
```

c) Nhiều lệnh gọi đệ quy trực tiếp.

c1) Thủ tục đệ quy với 2 lần gọi trực tiếp
Thủ tục đệ quy 2 lần gọi trực tiếp có dạng :

```

P(X) ≡ if C(X) then D(X)
           else begin
               A(X) ; P(f(X)) ;
               B(X) ; P(g(X)) ;
           end ;

```

Quá trình thực hiện thủ tục P(X) sẽ là :

- Nếu C(X) đúng thì thực hiện D(X) .
- Nếu C(X) sai thì thực hiện A(X) ; gọi P(f(X)) ; thực hiện B(X) ; gọi P(g(X)) , khi gọi P(g(X)) thì lại phát sinh lệnh A(g(X)) như vậy ngoài việc phải lưu vào stack các giá trị fⁱ(X) ta còn phải lưu vào stack các giá trị gⁱ(X) tương ứng . Khi ta lấy dữ liệu từ stack để thực hiện lệnh B(U) mà chưa gặp điều kiện kết thúc thì ta thực hiện P(g(U)) và lại phải lưu giá trị g(U) vào stack ,... Điều kiện dừng là khi truy xuất tới phần tử lưu đầu tiên trong stack .

Như vậy là ngoài dữ liệu X , con phải lưu vào ngăn xếp thêm thứ tự lần gọi (cụm gọi)

Thuật toán khử đệ quy tương ứng với thủ tục đệ quy P(X) là :

```

{ Creat_Stact (S) :
  Push (S, (X,1)) ;
  Repeat
    While ( not C(X) ) do begin
      A(X) ;
      Push (S, (X,2)) ;
      X := f(X) ;
    end ;
    D(X) ;
    POP (S, (X,k)) ;
    if ( k <> 1 ) then begin
      B(X) ;
      X := g(X) ;
    end ;
  until ( k = 1 ) ;
}

```


Ví dụ : Khử đệ quy thủ tục Tháp Hà Nội .

+ Dạng đệ quy của thủ tục Tháp Hà Nội là :

```

THN(n , X , Y , Z ) ≡ if( n > 0 ) then begin
                                THN ( n - 1 , X , Z , Y ) ;
                                Move ( X , Z ) ;
                                THN ( n - 1 , Y , X , Z ) ;
                                end ;
    
```

Với n là số đĩa , X là cột đầu , Z là cột cuối , Y là cột giữa , Move(X,Z) là thao tác chuyển 1 đĩa từ cột X tới cột Z .

Trong trường hợp này :

Biến X là bộ (n , X , Y , Z) .
 C(X) là biểu thức boolean (n <= 0) .
 D(X) , A(X) là thao tác rỗng .
 B(X) = B(n,X,Y,Z) là thao tác Move(X,Z) ;
 f(X) = f(n ,X ,Y ,Z) = (n - 1 , X , Z , Y) .
 g(X) = g(n ,X , Y , Z) = (n - 1 , Y , X , Z) .

Giải thuật không đệ quy tương đương là :

```

{ Creat_Stack (S) ;
  Push (S ,(n,X,Y,Z,1)) ;
  Repeat
    While ( n > 0 ) do begin
                                Push (S ,(n,X,Y,Z,2)) ;
                                n := n - 1 ;
                                Swap (Y,Z) ; (* Swap(a,b) là thủ tục hoán
                                đổi nội dung 2 biến a , b *)
                                end ;
    POP (S,(n,X,Y,Z,k)) ;
    if ( k <> 1 ) then begin
                                Move (X ,Z) ;
                                n := n - 1 ;
                                Swap (X ,Y) ;
                                end ;
    until ( k = 1 ) ;
}
    
```

c2) Trường hợp n lần gọi đệ quy trực tiếp .

Thủ tục đệ quy trong trường hợp này có dạng :

```

P(X) ≡  if C(X) then  D(X)
          else begin
              A1(X) ; P(f1(X)) ;
              A2(X) ; P(f2(X)) ;
              .....
              Ai(X) ; P(fi(X)) ;
              .....
              An(X) ; P(fn(X)) ;
              An+1(X) ;
          end ;
    
```

Cũng giống như trong trường hợp (3a) là khi quay trở lại sau khi thực hiện một lần đệ quy, cần biết đó là lệnh gọi thuộc nhóm thứ mấy trong dãy lệnh gọi để biết thao tác cần thực hiện tiếp. Vì vậy trong chồng cần giữ thêm vị trí nhóm lệnh gọi.

Dạng lặp tương ứng là :

```

{ Creat_Stack (S) ;
  Push(S,(X,1)) ;
  Repeat
    While (not C(X)) do begin
      A1(X) ;
      Push (S,(X,2)) ;
      X := f1(X) ;
    end ;
    D(X) ;
    POP(S,(X,k)) ;
    While( k = n+1 ) do begin
      An+1 ;
      POP(S,(X,k)) ;
    end ;
    if ( k > 0 ) then begin
      Ak(X) ;
      Push (S,(X,k+1));
      X := f k (X)
    end ;
  until (k = 1) ;
}
    
```

Ví dụ : Khử đệ quy cho thủ tục hoán vị .
 + Thủ tục hoán vị dưới dạng đệ quy :

```
HVI(V ,n) ≡ if (n = 1) then Print ( V )
                else for i := n downto 1 do
                    begin
                        Swap (V[n],V[i] ) ;
                        HVI(V ,n - 1) :
                    end ;
```

trong trường hợp này thì :

X là bộ (V ,n) . (* vector V và số nguyên n *)

C(X) là (n = 1) .

D(X) là Print (V) . (* xuất vector V *)

Ai(X) là thủ tục Swap(V[n],V[i]) (i = 1 .. n) .

An+1 là thao tác rỗng .

fi(X) = f(V , n) = (V , n - 1) .(với i = 1 .. n)

Dạng lập của thủ tục là :

```
{ Creat_Stack (S) ;
  Push (S,(V ,n ,1)) ;
  Repeat
    While ( n > 1 ) do begin
        Swap(V[n] ,V[1] ;
        Push (S ,V , n ,2) ;
        n := n - 1 ;
    end ;

    Print (V) ;
    POP (S ,(V ,n ,k)) ;
    While ( k = n + 1 ) do POP(S ,(V ,n ,k) ;
    if(k <> 1 ) then begin
        Swap(V[n] ,V[k] ;
        Push (S ,(V ,n ,k+1) ;
        n := n - 1 ;
    end ;

  until(k = 1 ) ;
```

Ở từng thời điểm trước hoặc sau khi thực hiện một lệnh, người ta quan tâm đến tập hợp các trạng thái có thể của chương trình. Tập hợp các trạng thái này sẽ được biểu thị bởi các tân từ bậc nhất với các biến là các biến của chương trình.

Ví dụ 1 : Đoạn chương trình sau tính tích của hai số nguyên dương a và b

```
{ ( a > 0 ) and ( b > 0 ) } // ràng buộc trên trạng thái đầu .
    x := a ;
    y := b ; u := 0 ;
{ ( x = a ) and ( y = b ) and ( ( u + x*y ) = ( a*b ) ) } // ràng buộc trung gian trên
    repeat { ( u+x*y = a*b ) and ( y > 0 ) } // ràng buộc trung gian trên trạng
        u := u + a ;
        y := y - 1 ;
    { ( u+x*y = a*b ) and ( y >= 0 ) } // ràng buộc trung gian trên trạng
thái
    until ( y = 0 ) // ràng buộc trung gian trên trạng thái cuối
        { u = a*b } // ràng buộc trên trạng thái xuất
```

Mỗi tân từ trong ví dụ trên mô tả một tập các trạng thái có thể có ở điểm đó.

Ví dụ 2 : Đoạn chương trình hoán đổi nội dung của 2 biến x và y, dùng biến t làm trung gian.

```
{ ( x = x0 ) and ( y = y0 ) } // x , y mang giá trị ban đầu bất kỳ nào đó
    t := x ;
    x := y ;
    y := t
{ ( x = y0 ) and ( y = x0 ) } // x , y sau cùng mang giá trị hoán đổi của nhau.
```

Trong ví dụ này để biểu diễn quan hệ giữa nội dung các biến với nội dung của một số biến bị gán trị, người ta cần phải dùng các biến giả (ghost variable). Ví dụ ở đây là x_0 và y_0 biểu thị nội dung của x và y trước khi thực hiện đoạn chương trình.

Ví dụ 3 :

Nhân 2 số nguyên dương x, y, kết quả chứa trong u.

```
{ ( x = x0 > 0 ) and ( y = y0 > 0 ) }
    u := 0 ;
    repeat
        u := u + x ;
        y := y - 1 ;
    until ( y = 0 )
    { u = x0 * y0 }
```

Thực ra ở đây tập hợp các trạng thái xuất thực sự là nhỏ hơn, biểu thị bởi một điều kiện chặt hơn, đó là : $\{ (u = x_0 * y_0) \text{ and } (y = 0) \text{ and } (x = x_0) \}$

Tổng quát ta cần khảo sát một đoạn chương trình S với 2 điều kiện đi trước P và đi sau Q. Cần chứng minh rằng nếu xuất phát từ trạng thái thoả P thì hành lệnh S thì

Ví dụ : while (x > 0) do begin y := x;
 while (y > 0) do y := y - 1;
 x := x - 1;
 end ;

- <Dãy lệnh> chính là dãy tuần tự các <lệnh đơn> ngăn cách bởi dấu ';

$\{true\} x := 5 ; y := 2 \{ x = 5, y = 2 \}$
 Ta có : $\{ 5 = 5 \} x := 5 \{ x = 5 \}$ (a) // tiên đề gán
 $true \Rightarrow (5 = 5)$ và $(x = 5) \Rightarrow ((x = 5) \text{ and } (2 = 2))$ (b) // hiển nhiên
 $\{true\} x := 5 \{ (x = 5) \text{ and } (2 = 2) \}$ (c) //theo luật hệ quả
 $\{ x = 5, 2 = 2 \} y := 2 \{ (x = 5) \text{ and } (y = 2) \}$ (d) // tiên đề gán
 $\{true\} x := 5 ; y := 2 \{ x = 5, y = 2 \}$ // theo luật tuần tự

b) Luật về điều kiện (chọn) (Rule for conditionals)

b1)

$\frac{\{ P \text{ and } B \} S_1 \{ Q \}, \{ P \text{ and } (\text{not } B) \} S_2 \{ Q \}}{\{ P \} \text{ if } B \text{ then } S_1 \text{ else } S_2 \{ Q \}}$	(3.2a)
--	--------

Ý nghĩa luật này là :

Nếu có :

$\{ P \text{ and } B \}$ + Nếu xuất phát từ trạng thái thỏa P and B
 S_1 thì hành S_1 thì sẽ tới trạng thái thỏa Q
 $\{ Q \}$

Và

$\{ P \text{ and not } B \}$ + Nếu xuất phát từ trạng thái thỏa P and not B
 S_2 thì hành S_2 thì sẽ tới trạng thái thỏa Q
 $\{ Q \}$

Thì suy ra :

$\{ P \}$ Nếu xuất phát từ trạng thái thỏa P
 if B then S_1 else S_2 thì hành lệnh if B then S_1 else S_2
 $\{ Q \}$ thì sẽ tới trạng thái thỏa Q .

b2)

$\frac{\{ P \text{ and } B \} S \{ Q \}, P \text{ and } (\text{not } B) \Rightarrow Q}{\{ P \} \text{ if } B \text{ then } S \{ Q \}}$	(3.2b)
--	--------

Ví dụ 1 : Kiểm chứng tđcđk đặc tả :

$\{ i > 0 \} \text{ if } (i = 0) \text{ then } j := 0 \text{ else } j := 1 \{ j = 1 \}$

Ta có : $((i > 0) \text{ and } (i = 0)) \equiv \text{false}$ (a) // hiển nhiên

$\{ (i > 0) \text{ and } (i = 0) \} j := 0 \{ j = 1 \}$ (b) // $\{\text{false}\} S \{ Q \}$ đúng với \forall

S, Q

$((i > 0) \text{ and } \text{not}(i = 0)) \equiv \text{true}$ (c) // hiển nhiên

$\{true\} j := 1 \{j=1\}$ (d) //tiên đề gán
 $\{(i > 0) \text{ and } \text{not}(i = 0)\} j := 1 \{j=1\}$ (e) // c,d ,luật hệ quả

Từ b ,e và tiên đề 3.2a ta suy ra ĐPCM.

Ví dụ 2 : Kiểm chứng tđcđk đặc tả :

$\{i \geq j - 1\} \text{ if } (i > j) \text{ then } j := j+1 \text{ else } i := i+1 \{i \geq j\}$

Ta có : $\{i \geq j+1\} j := j+1 \{i \geq j\}$ (a) //tiên đề gán

$((i \geq j-1) \text{ and } (i > j)) \implies (i \geq j+1)$ (b) // biến đổi với chú ý i , j

nguyên

$\{(i \geq j-1) \text{ and } (i > j)\} j := j + 1 \{i \geq j\}$ (c) // a,b ,luật hệ quả

$\{i+1 \geq j\} i := i+1 \{i \geq j\}$ (d) // tiên đề gán

$((i \geq j-1) \text{ and } \text{not}(i > j)) \implies (i+1 \geq j)$ (e) // biến đổi

$\{(i \geq j-1) \text{ and } \text{not}(i > j)\} i := i + 1 \{i \geq j\}$ (g) // d ,e , luật hệ quả

Từ c , g dựa vào 3.2a suy ra ĐPCM.

Ví dụ 3 : Kiểm chứng tđcđk đặc tả :

$\{true\} \text{ if } \text{odd}(x) \text{ then } x := x+1 \{\text{even}(x)\}$

Ta có : $\{\text{even}(x+1)\} x := x+1 \{\text{even}(x)\}$ (a) //tiên đề gán

$true \text{ and } \text{odd}(x) \implies \text{even}(x+1)$ (b) // hiển nhiên

$\{true \text{ and } \text{odd}(x)\} x := x+1 \{\text{even}(x)\}$ (c) // a ,b , luật hệ quả

$true \text{ and } \text{not } \text{odd}(x) \implies \text{even}(x)$ (d) // hiển nhiên

Từ (c) và (d) dựa vào 3.2b suy ra ĐPCM .

b3) Luật về lệnh lặp While

$$\frac{\{I \text{ and } B\} S \{I\}}{\{I\} \text{ while } B \text{ do } S \{I \text{ and } (\text{not } B)\}} \quad (3.3)$$

Luật này nói rằng nếu I không bị thay đổi bởi một lần thực hiện lệnh S thì nó cũng không bị thay đổi bởi toàn bộ lệnh lặp While B do S. Với ý nghĩa này I được gọi là bất biến (invariant) của vòng lặp.

Chú ý rằng khẳng định : $\{P\} \text{ while } B \text{ do } S \{Q\}$ thỏa dựa vào hệ luật Hoare chỉ xác định tđcđk (conditionnal correctness). Để chứng minh tính đúng (correctness) thực sự ta cần bổ sung chứng minh lệnh lặp dừng.

Ví dụ 1 :

Kiểm chứng tính đúng có điều kiện của đặc tả :

$\{i \leq n\} \text{ while } (i < n) \text{ do } i := i+1 \{i=n\}$

Xem I là $(i \leq n)$ thì :

$\{I \text{ and } (i < n)\} i := i+1 \{I\}$ (a) // dễ kiểm chứng

Nên $\{I\} \text{ while } (i < n) \text{ do } i := i+1 \{I \text{ and } \text{not}(i < n)\}$ (b) // luật 3.3

$\{(x = x_0) \text{ and } (y = y_0)\} \quad t := x ; x := y ; y := t \quad \{(x = y_0) \text{ and } (y = x_0)\} \text{ (h) // c, g, luật tuần tự}$

Ví dụ 1 : Kiểm chứng tính đúng của đặc tả :

$$\{ (m :: k=2*m) \text{ and } (y * z^k = c) \}$$

$$k := k \text{ div } 2 ;$$

$$z := z * z ;$$

$$\{ y * z^k = c \}$$

Chứng minh :

(a) $\{y * (z*z)^k = c\} \quad z := z * z \quad \{y*z^k = c\}$ (tiên đề gán)

(b) $\{y * (z*z)^{k \text{ div } 2} = c\} \quad k := k \text{ div } 2 \quad \{y*(z*z)^k = c\}$ (tiên đề gán)

(c) $\{y * (z*z)^{k \text{ div } 2} = c\} \quad k := k \text{ div } 2 ; z := z*z \quad \{y*z^k = c\}$ (a, b, luật tuần tự)

(d) $(m :: k = 2*m) \text{ and } (y * z^k = c) \implies (y*z^{2m} = c) \text{ and } (m = k \text{ div } 2)$
 $\implies y * (z*z)^{k \text{ div } 2} = c$

c, d, luật hệ quả suy ra ĐPCM.

Nhận xét :

Với dãy tuần tự các lệnh gán, việc chứng minh $\{P\} S_1; \dots; S_n \{Q\}$ thường được bắt đầu từ lệnh cuối cùng, dùng tiên đề gán để được đkđ, rồi cứ thế lần ngược về đến S1.

$\{P_n\} S_n \{Q\}$ (n) tìm P_n từ S_n, Q và tiên đề gán

$\{P_{n-1}\} S_{n-1} \{P_n\}$ (n-1) tìm P_{n-1} từ S_{n-1}, P_n và tiên đề gán

$\{P_{n-1}\} S_{n-1}; S_n \{Q\}$ luật về dãy lệnh tuần tự

...

...

$\{P_1\} S_1; \dots; S_n \{Q\}$ (1) sau n-1 lần tương tự như trên.

Sau đó dùng các tính chất của dữ kiện chứng minh logic rằng :

$$P \implies P_1 \text{ (0)}$$

Từ (1), (0), dựa vào luật hệ quả ta có : $\{P\} S_1; \dots; S_n \{Q\}$ (ĐPCM)

2) Bài toán 2 :

a) Kiểm chứng đặc tả : $\{P\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \{Q\}$

Với S_1, S_2 là nhóm các lệnh gán, B là biểu thức boolean.

Cách chứng minh :

+ Bước 1 : Tìm P_1, P_2 thỏa : $\{P_1\} S_1 \{Q\}$ (1a)
 $\{P_2\} S_2 \{Q\}$ (1b)

+ Bước 2 : Chứng minh (dùng các tính chất logic và đại số)

$$P \text{ and } B \implies P_1 \text{ (2a)}$$

$$P \text{ and } (\text{not } B) \implies P_2 \text{ (2b)}$$

+ Bước 3 : Dùng luật hệ quả suy ra :

$$\{P \text{ and } B\} S_1 \{Q\} \text{ (3a) // 1a, 2a, và luật hệ quả}$$

$$\{P \text{ and } (\text{not } B)\} S_2 \{Q\} \text{ (3b) // 1b, 2b, và luật hệ quả}$$

+ Bước 4 : Dùng (3a), (3b), luật điều kiện suy ra :

{P} if B then S₁ else S₂ {Q} (ĐPCM)

b) Kiểm chứng đặc tả : {P} S₀ ; if B then S₁ else S₂ {Q} (*)

với S₁, S₂, S₀ là dãy các lệnh gán

Ví dụ : Kiểm chứng đặc tả :

```
{y > 0}
x := y-1 ;
if (y > 3) then x := x*x
else y := y-1
{x >= y}
```

Để khẳng định được (*) ta cần chỉ ra 1 khẳng định R mà :

{P} S₀ {R}

và {R} if B then S₁ else S₂ {Q} rồi dùng luật hệ quả để có (*)

Làm thế nào để tìm được R ? Do S₁ và S₂ là nhóm lệnh gán tuần tự nên ta có thể tìm được (bằng tiên đề gán và luật về dãy lệnh tuần tự) U và V để :

{U} S₂ {Q} và {V} S₃ {Q} .

Dĩ nhiên ta muốn U và V là các điều kiện tổng quát nhất có thể (ở đây là yếu nhất). R được xây dựng thế nào từ U và V ? Khả năng tổng quát nhất cho R để sau khi điểm điều kiện B sẽ có được U hoặc V là : $R \equiv (B \implies U) \text{ and } (\text{not } B \implies V)$

Như sau này sẽ chỉ ra U, V, R được xây dựng như vậy là yếu nhất (weakest precondition) để đạt được Q tương ứng với lần lượt các lệnh S₁, S₂ và if B then S₁ else S₂, và được ký hiệu là : WP(S₂,Q), WP(S₃,Q) và WP(if B then S₂ else S₃, Q) tương ứng.

Ví dụ 1 : Kiểm chứng đặc tả :

```
{ y > 0 }
x := y - 1 ;
if ( y > 3 ) then x := x * x
else y := y - 1 ;
{ x >= y }
```

Trong ví dụ này :

P là tân từ : (y > 0) ; Q là tân từ : (x >= y)

B là biểu thức boolean : (y > 3)

S₀ là lệnh gán : x := y - 1 ;

Do S₁ và S₂ là lệnh gán : x := x * x ;

S₂ là lệnh gán : y := y - 1 ;

Ta có :

(a) $\{x^2 \geq y\} \quad x := x*x \quad \{x \geq y\} \quad \text{suy ra} \quad U \equiv \text{WP}(S_1, Q) \equiv x^2 \geq y$

(b) $\{x \geq y-1\} \quad y := y-1 \quad \{x \geq y\} \quad \text{suy ra} \quad V \equiv \text{WP}(S_2, Q) \equiv x \geq y-1$

$$\begin{aligned} \text{Đặt } R &\equiv (B \implies U) \text{ and } (\text{not } B \implies V) \\ &\equiv ((y > 3) \implies (x^2 \geq y)) \text{ and } ((y \leq 3) \implies (x \geq y-1)) \end{aligned}$$

Ta chứng minh được dễ dàng.

$$R \text{ and } (y > 3) \implies (x^2 \geq y) \quad (c)$$

$$R \text{ and } (\text{not}(y > 3)) \implies (x \geq y-1) \quad (d)$$

nên theo luật hệ quả

$$\{R \text{ and } y > 3\} S_2 \{x \geq y\} \quad (\{R \text{ and } B\} S_2 \{Q\}) \quad (e)$$

$$\{R \text{ and } \text{not}(y > 3)\} S_3 \{x \geq y\} \quad (\{R \text{ and } (\text{not } B)\} S_3 \{Q\}) \quad (g)$$

Theo luật về lệnh chọn

$$\{R\} \text{ if } (y > 3) \text{ then } x := x*x \text{ else } y := y-1 \{x \geq y\} \quad (h)$$

theo tiên đề gán.

$$\begin{aligned} &\{ ((y > 3) \implies ((y-1)^2 \geq y)) \text{ and } ((y \leq 3) \implies ((y-1) \geq (y-1))) \} \\ &\quad x := y-1 \\ &\quad \{ R \} \quad (i) \end{aligned}$$

Dễ kiểm chứng được

$$(y > 3) \implies ((y-1)^2 \geq y) \equiv \text{true} \quad (j)$$

$$(y \leq 3) \implies (y-1) \geq y-1 \equiv \text{true} \quad (k)$$

nên

$$(y > 0) \implies ((y > 3) \implies ((y-1)^2 \geq y)) \text{ and } ((y \leq 3) \implies ((y-1) \geq (y-1))) \quad (l)$$

Theo luật hệ quả

$$\begin{aligned} &\{y > 0\} \\ &\quad x := y-1; \\ &\quad \text{if } (y > 3) \text{ then } x := x*x \text{ else } y := y-1 \\ &\quad \{x \geq y\} \quad (m) // \text{ĐPCM} \end{aligned}$$

Ví dụ 2 : kiểm chứng đặctả :

$$\{ \text{true} \}$$

$$\begin{aligned} &\text{if } (i \leq j) \text{ then if } (j < k) \text{ then } m := k \text{ else } m := j \\ &\quad \text{else if } (i < k) \text{ then } m := k \text{ else } m := i \\ &\quad \{(m \geq i) \text{ and } (m \geq j) \text{ and } (m \geq k)\} \end{aligned}$$

$$\text{Đặt } Q(m) \equiv (m \geq i) \text{ and } (m \geq j) \text{ and } (m \geq k)$$

Ta có :

$$(a) \{Q(i)\} m := i \{Q(m)\} \quad (\text{tiên đề gán})$$

$$(b) \{Q(k)\} m := k \{Q(m)\} \quad (\text{tiên đề gán})$$

$$\text{Đặt } R1 \equiv ((i < k) \implies Q(k)) \text{ and } ((i \geq k) \implies Q(i))$$

dùng luật về lệnh chọn ta sẽ chứng minh được :

$$\{R1\} \text{ if } (i < k) \text{ then } \dots \{Q(m)\} \quad (c)$$

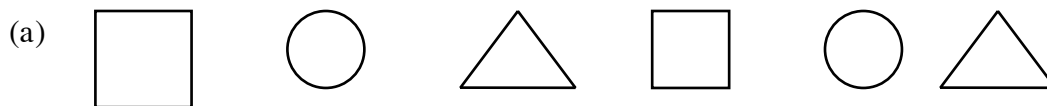
$$\text{Tương tự đặt } R2 \equiv (j < k) \implies (Q(k) \text{ and } (j \geq k)) \implies Q(j)$$

kicked, jumped, walked ... bằng học luật không đổi (thêm s, thêm ed), kèm theo với việc học thuộc một số trường hợp ngoại lệ. Hãy tưởng tượng việc học sẽ khó như thế nào nếu không có các luật không đổi (bất biến) này.

Việc nhận thức được các bất biến thường dẫn tới những lời giải đơn giản cho các bài toán khó.

Đầu óc con người dường như có một khả năng đặc biệt để nhận thức các bất biến hay các "khuôn mẫu".

Hãy quan sát 2 dãy các hình sau :



Hình kế tiếp trong mỗi dãy hình trên là gì ? Tính chất bất biến của mỗi dãy là gì ?

(a) Lặp lại bộ 3 hình vuông, tròn, tam giác.

(b) Về dạng thì là sự lặp lại của cặp 2 hình vuông lớn và nhỏ. Về màu thì là sự lặp lại của một màu trắng và 2 màu sậm.

Trong lĩnh vực chương trình cho máy tính, ta cũng cần nhận thức các sự việc bằng cách phát hiện các bất biến. Đối với một chương trình, ta có nhiều lần máy tính thi hành nó, mỗi lần thi hành được gọi là một quá trình (process) và tác động trên các dữ kiện khác nhau. Tính bất biến của các quá trình này chính là đặc tả của chương trình.

Bên trong một chương trình có thể có các vòng lặp. Việc thực hiện vòng lặp làm biến thiên nhiều lần trạng thái các biến chương trình (các đối tượng dữ liệu), mà số lần biến thiên thường không biết trước được. Làm thế nào để hiểu được tác động của vòng lặp và đi đến chứng minh vòng lặp thực hiện một tính chất (giữ một bất biến) nào đó thể hiện bởi đặc tả của nó.

Mô hình biến đổi trạng thái chương trình của vòng lặp **while B do S**



{ P } là trạng thái trước vòng lặp .

{ P_i } là trạng thái sau lần lặp thứ i .

Nếu (α) và (β) thoả thì S không thể lặp vô tận được (vì sau hữu hạn lần thì B sẽ nhận giá trị false).

Ví dụ : Chỉ ra các vòng lặp sau đây dừng :

```
{ n >= 0 }
k := n ;
while ( k <> 0 ) do begin { k > 0 }
    k := k-1 ;
    r := 2*r + p[k];
    if ( r >= q ) then r := r - q
end ;
```

vì bất biến $\{ k > 0 \}$ luôn được giữ đúng ở đầu vòng lặp. Ở đây hàm f chính là bằng k. f giảm sau mỗi lần lặp (vì $k := k - 1$). Vậy vòng lặp dừng .

```
{ x >= 0 ; y >= 0 }
a := x ;
b := y ;
while ( a <> b ) do
    { max(a,b) > 0 }
    if ( a > b ) then a := a - b
    else b := b - a
```

Ở đây hàm $f = \max(a,b)$. Ta luôn có bất biến $\max(a,b) > 0$ ở đầu vòng lặp, f giảm sau mỗi lần lặp.

b) Các ví dụ về chứng minh chương trình có vòng lặp .

Ví dụ 1 : Xét đặc tả đoạn chương trình tính tích 2 số nguyên A và B với $B \geq 0$ bằng phép cộng :

```
{ B >= 0 }
R := 0 ;
X := B ;
while ( X <> 0 ) do begin
    R := R + A ;
    X := X - 1 ;
end ;
{ R = A*B }
```

đkd P \equiv B >= 0
đkc Q \equiv R = A * B

Bước 1: Kiểm chứng tính đúng có điều kiện của đặc tả {P} S {Q}

+ Kiểm chứng đoạn lệnh trước vòng lặp : Chứng minh đặc tả sau đúng .

```
{ B >= 0 }
R := 0 ;      (*)
X := B ;
{ X = B , R = 0 , B >= 0 }
```

Ta có :

$$\{0 = 0, B \geq 0\} R := 0 \{R = 0, B \geq 0\} \quad (1) \text{ tiên đề gán}$$

$$\{0 = 0, B \geq 0\} \Rightarrow \{B \geq 0\} \quad (2) \text{ hiển nhiên}$$

$$\{B \geq 0\} R := 0 \{R = 0, B \geq 0\} \quad (3) \text{ luật hệ quả dựa vào (1),(2)}$$

$$\{B = B, R = 0, B \geq 0\} X := B \{X = B, R = 0, B \geq 0\} \quad (4) \text{ tiên đề gán}$$

$$\{B = B, R = 0, B \geq 0\} \Rightarrow \{R = 0, B \geq 0\} \quad (5) \text{ hiển nhiên}$$

$$\{R = 0, B \geq 0\} X := B \{X := B, R = 0, B \geq 0\} \quad (6) \text{ Luật hệ quả dựa vào}$$

(4),(5)

$$\{B \geq 0\} X := 0; X := B \{X = B, R = 0, B \geq 0\} \quad (7) \text{ luật tuần tự dựa vào (3),(6)}.$$

Như vậy với điều kiện đầu $B \geq 0$ thì sau khi thực hiện xong 2 lệnh khởi động, ta có khẳng định $X = B, R = 0, B \geq 0$ đặc tả (*) đúng .

+ Kiểm chứng vòng lặp :

- Phát hiện được bất biến của vòng lặp.

Bất biến ở đây là : “ số lần X bị giảm đi chính là số lần A được cộng vào R “

$$\text{Tức là : } I \equiv (R = A*(B-X)) \text{ and } (X \geq 0)$$

Khẳng định $(X \geq 0)$ được thêm vào để chứng minh vòng lặp dừng.

- Chứng minh I là bất biến của vòng lặp :

$$\{(R + A = A*B - A*X + A) \text{ and } (X > 0)\}$$

$$R := R + A$$

$$\{(R = A*B - A*X + A) \text{ and } (X > 0)\} \quad (8) \text{ tiên đề gán}$$

$$\{(R = A*(B - X)) \text{ and } (X > 0)\}$$

$$R := R + A$$

$$\{(R = A*B - A*X + A) \text{ and } (X > 0)\} \quad (9) \text{ biến đổi từ (8)}$$

$$\{(R = A*B - A*X + A) \text{ and } (X > 0)\} \equiv$$

$$\{(R = A*(B - (X - 1))) \text{ and } ((X - 1) \geq 0)\} (10)$$

$$\{(R = A*(B - (X - 1))) \text{ and } ((X - 1) \geq 0)\}$$

$$X := X - 1$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0)\} \quad (11) \text{ tiên đề gán}$$

$$\{(R = A*B - A*X + A) \text{ and } (X > 0)\}$$

$$X := X - 1$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0)\} \quad (12) \text{ luật hệ quả}$$

$$\{(R = A*(B - X)) \text{ and } (X > 0)\}$$

$$R := R + A; X := X - 1$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0)\} \quad (13) \text{ luật tuần tự dựa vào (9),(12)}$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0) \text{ and } (X <> 0)\} \implies \{(R = A*(B - X)) \text{ and } (X > 0)\} (14)$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0) \text{ and } (X <> 0)\}$$

$$R := R + A; X := X - 1$$

$$\{(R = A*(B - X)) \text{ and } (X \geq 0)\} \quad (15) \text{ luật hệ quả dựa vào (13),(14)}$$

Hay $\{I \text{ and } C\} R := R + A; X := X - 1 \{I\}$

với C là điều kiện vòng lặp $X <> 0$

Do luật lặp ta có :

{I} while ... {I and not C}

$(X=B) \text{ and } (R=0) \text{ and } (B>=0) \implies (R=A*(B-X)) \text{ and } (X>=0)$ (16)

Kết hợp 15,16 và 5 dùng luật hệ quả rồi luật tuần tự, ta có :

$\{X=B \text{ and } R=0 \text{ and } B>=0\} \text{ while } \dots \{I \text{ and not } C\}$ (17)

$\{B>=0\} \text{ R}:=0 ; \text{ X}:=B ; \text{ while } \dots \{I \text{ and not } C\}$ (18)

Ở đây $I \text{ and not } C \equiv (R = A*(B-X)) \text{ and } (X >= 0) \text{ and } (X = 0)$

mà $(R = A*(B-X)) \text{ and } (X >= 0) \text{ and } (X=0) \implies R=A*B$

Dùng luật hệ quả ta có đpcm

Bước 2 : Chứng minh tính dừng :

Đặt $f = X$, ta có :

(i) $I \text{ and } C \equiv (R = A*(B-X)) \text{ and } (X >= 0) \text{ and } (X <> 0) \implies X > 0 \implies f > 0$

(ii) Mỗi lần lặp, f bị giảm một đơn vị. Vậy vòng lặp phải dừng.

Từ (i) và (ii) ta kết luận được tính dừng từ bước 1 và bước 2 suy ra tính đúng đầy đủ của đoạn chương trình đối với P,Q.

Nhận xét từ chứng minh trên :

+ Đối với dãy các lệnh gán, nên phát xuất quá trình suy diễn từ điều kiện cuối.

+ Đối với vòng lặp cần xác định đúng bất biến của nó.

Chú ý : Ta có thể kiểm chứng tđcđk của đoạn chương trình trên bằng cách:

- Xây dựng một lược đồ chứng minh hợp lý bằng cách dựa vào các tiên đề và các khẳng định đã có trước đó chèn bổ sung các khẳng định trung gian ở những điểm khác nhau trong đoạn chương trình .

$\{P\} \equiv \{B \geq 0\}$ (0)

$\{(0 = A*(B - B)) \text{ and } (B \geq 0)\}$ (3)

$R := 0 ;$

$\{(R = A*(B - B)) \text{ and } (B \geq 0)\}$ (2)

$X := B ;$

$\{I\} \equiv \{(R = A*(B - B)) \text{ and } (X \geq 0)\}$ (1a)

while $(X <> 0)$ do begin

$\{I \text{ and } C\} \equiv \{(R = A*(B - X)) \text{ and } (X \geq 0) \text{ and } (X <> 0)\}$ (1b)

$\{(R + A = A*(B - (X - 1)) \text{ and } ((X - 1) \geq 0)\}$ (5)

$R := R + A ;$

$\{(R = A*(B - (X - 1)) \text{ and } ((X - 1) \geq 0)\}$ (4)

$X := X - 1 ;$

$\{I\} \equiv \{(R = A*(B - X)) \text{ and } (X \geq 0)\}$ (1d)

end

$\{I \text{ and not } C\} \equiv \{(R = A*(B - X)) \text{ and } (X \geq 0) \text{ and not}(X <> 0)\}$ (1c)

$\{Q\} \equiv \{R = A*B\}$ (6)

Lý lẽ để bổ sung là :

(1a) do phát hiện được bất biến I

(1b),(1c),(1d) dựa vào tiên đề về lệnh lặp xuất phát từ I

2, 3 dựa vào tiên đề gán xuất phát từ (1a)

4, 5 dựa vào tiên đề gán xuất phát từ (1d)

Các cặp khẳng định đi liền nhau là các điều kiện cần kiểm chứng :

$$(0) \implies (3) : (B \geq 0) \implies ((0 = A * (B-B)) \text{ and } (B \geq 0))$$

$$(1b) \implies (5) : ((R = A*(B - X)) \text{ and } (X \geq 0) \text{ and } (X <> 0)) \implies$$

$$((R + A = A*(B -(X - 1)) \text{ and } (X - 1)$$

$\geq 0)$)

$$(1c) \implies (6) : ((R = A*(B-X)) \text{ and } (X \geq 0) \text{ and } (X = 0)) \implies (R=A*B)$$

Để dàng chứng minh các điều trên.

Gọi k là số lần lặp (số lần thực hiện lệnh S ở thân vòng lặp). Ta xây dựng quy nạp theo k :

Bước 0 : ($k = 0$) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng 0 lần và tới trạng thái thỏa Q là : $P_0 \equiv (\text{not } B) \text{ and } Q$

Bước 1 : ($k = 1$) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng một lần và tới trạng thái thỏa Q là : $P_1 \equiv B \text{ and } WP(S, P_0)$
(tức là sau khi thực hiện một lần S thì trạng thái chương trình sẽ thỏa P_0).

Bước 2 : ($k = 2$) tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng 2 lần và tới trạng thái thỏa Q là : $P_2 \equiv B \text{ and } WP(S, P_1)$
(tức là sau khi thực hiện một lần S thì trạng thái chương trình sẽ thỏa P_1).

.....
.....

Bước k : Một cách tổng quát với $k \geq 1$ thì tân từ yếu nhất mô tả tập lớn nhất các trạng thái bảo đảm S lặp đúng k lần và tới trạng thái thỏa Q là : $P_k \equiv B \text{ and } WP(S, P_{k-1})$

Như vậy một trạng thái đầu làm W dừng ở một trạng thái thỏa Q khi và chỉ khi nó thỏa khẳng định sau : $\exists(k : k \geq 0 : P_k)$

Tức là :
$$WP(\text{while } B \text{ do } S, Q) \equiv \exists(k : k \geq 0 : P_k)$$

Với $P_k = \begin{cases} \text{not } B \text{ and } Q & \text{với } k = 0 \\ WP(S, P_{k-1}) & \text{với } k > 0 \end{cases}$

Ví dụ :

Cho S là đoạn chương trình :

$$j := j * i ; k := k + j ; n := n + 1 ;$$

và W là $\text{while } (n <> m) \text{ do } S$

$$Q \text{ là : } (k = (i^{m+1} - 1) / (i - 1) \text{ and } j = i^m)$$

(đoạn chương trình nhằm tính $k = 1 + i^1 + i^2 + \dots + i^m$)

Giả sử rằng $(i <> 0)$ và $(i <> 1)$, xác định $WP(W, Q)$.

Dãy các khẳng định P_n được xác định :

$$P_0 \equiv \text{not}(n <> m) \text{ and } Q$$

$$P_r \equiv (n <> m) \text{ and } wp(S, P_{r-1}) \text{ với } r = 1, 2, 3, \dots$$

Thực hiện tính toán ta có :

$$P_0 \equiv (n = m) \text{ and } (k = (i^{m+1} - 1) / (i - 1)) \text{ and } (j = i^m)$$

$$P_1 \equiv (n <> m) \text{ and } (n + 1 = m) \text{ and } ((k + j * i) = (i^{m+1} - 1) / (i - 1)) \text{ and } (j * i = i^m)$$

$$\equiv (n = m - 1) \text{ and } (k = (i^m - 1) / (i - 1)) \text{ and } (j = i^{m-1})$$

Tương tự :

$$P_2 \equiv (n = m - 2) \text{ and } (k = (i^{m-1} - 1)/(i-1)) \text{ and } (j = i^{m-2})$$

Ta có thể chứng minh bằng quy nạp giả thuyết sau (với mọi số tự nhiên r)

$$P_r \equiv (n = m - r) \text{ and } (k = (i^{m-r+1} - 1)/(i-1)) \text{ and } (j = i^{m-r})$$

$$P_n \equiv (n = m - n) \text{ and } (k = (i^{n+1} - 1)/(i-1)) \text{ and } (j = i^n)$$

Vậy :

$$\begin{aligned} WP(W,Q) &\equiv \exists(r : r \geq 0 : (n = m - r) \text{ and } (k = (i^{n+1} - 1)/(i-1)) \text{ and } (j = i^n)) \\ &\equiv (n \leq m) \text{ and } (k = (i^{n+1} - 1)/(i-1)) \text{ and } (j = i^n) \end{aligned}$$

b) Mối liên hệ giữa toán tử lặp và tiên đề lệnh lặp của hệ luật Hoare .

Ta tách việc khảo sát tính đúng một vòng lặp thành hai phần :

+ Phần quan sát sự biến đổi của vòng lặp để dẫn tới khẳng định nó dừng (tính dừng) .

+ Phần quan sát sự bất biến của vòng lặp để chứng minh kết quả cuối cùng của nó (đcđk)

Với ý tưởng đó ta tách WP(W,Q) (với W là while do S) thành các thành phần tương ứng và khảo sát mối quan hệ giữa WP(W,Q) và các khẳng định của hệ luật Hoare.

α) Với lệnh bất kỳ S, điều kiện yếu nhất để đảm bảo S dừng là không ràng buộc gì sau khi dừng. Tức là WP(S,true) là tân từ mô tả tập hợp tất cả các trạng thái mà xuất phát từ đó thì bảo đảm S dừng.

$$\text{Ta có : } WP(W,true) \equiv \exists(k : k \geq 0 : P_k)$$

$$\text{Với } P_0 \equiv (\text{not } B) \text{ and true} \equiv (\text{not } B)$$

$$P_k \equiv B \text{ and } WP(S,P_{k-1}) \text{ với } k > 0$$

(P_0 là điều kiện để không thực hiện S lần nào, P_1 là điều kiện để thực hiện S đúng một lần, P_k là điều kiện để thực hiện S đúng k lần.

Ví dụ :

$$\begin{aligned} W &\equiv \text{while } (n \lt m) \text{ do begin} \\ &\quad j := j * i ; k := k + j ; n := n + 1 ; \\ &\text{end ;} \end{aligned}$$

Ta tính điều kiện đầu để W dừng như sau :

$$P_0 \equiv \text{not } (n \lt m) \equiv (n = m)$$

$$P_1 \equiv B \text{ and } WP(S,P_0) \equiv (n \lt m) \text{ and } (n+1 = m) \equiv (n+1 = m)$$

Giả thiết quy nạp rằng $P_k \equiv (n+k = m)$.

Ta có :

$$\text{Giả thiết đúng với } k = 0 \text{ vì } P_0 \equiv (n = m) \equiv (n + 0 = m)$$

$$\text{Giả sử giả thiết đã đúng với } k. \text{ Tức là : } P_k \equiv ((n+k) = m)$$

Chứng minh giả thiết đúng với k+1. Thực vậy:

$$P_{k+1} \equiv B \text{ and } WP(S,P_k) \equiv (n \lt m) \text{ and } ((n+1)+k = m) \equiv (n+(k+1) = m)$$

$$\begin{aligned} (WP(S,P_k) &\equiv WP(j := j * i ; k := k + j ; n := n + 1, ((n+k) = m)) = (n + (k+1)) \\ &= m) \end{aligned}$$

$$\text{Vậy : } P_i \equiv (n+i = m)$$

$$\text{Tức là : } WP(W,true) \equiv \exists(i : i \geq 0 ; n+i = m) \equiv (n \geq m)$$

β) Quan hệ giữa $\{ P \} S \{ Q \}$ và WP(S,Q)

Theo định nghĩa về tính đúng và $WP(S, Q)$ ta có : S đúng có điều kiện dựa trên điều kiện đầu P và điều kiện cuối Q (đặc tả $\{P\} S \{Q\}$ đcđk) nếu và chỉ nếu hội (and) của P và điều kiện yếu nhất bảo đảm sự dừng của S mạnh hơn điều kiện yếu nhất bảo đảm S dừng trong một trạng thái thoả tâm từ Q .

Tức là : $\{P\} S \{Q\}$ thoả đk khi và chỉ khi $P \text{ and } WP(S, \text{true}) \implies WP(S, Q)$

Như vậy :

$\{I \text{ and } B\} S \{I\}$ thoả có đk khi và chỉ khi $I \text{ and } B \text{ and } WP(S, \text{true}) \implies WP(S, I)$

$\{I\} \text{ while } B \text{ do } S \{I \text{ and not } B\}$ thoả có đk khi và chỉ khi

$\{I\} \text{ and } WP(\text{while } B \text{ do } S, \text{true}) \implies WP(W, I \text{ and not } B)$

Như vậy chứng minh S giữ bất biến I chính là chứng minh

$I \text{ and } B \text{ and } wp(W, \text{true}) \implies wp(S, I)$

Chứng minh W dừng ứng với đkđ P chính là chứng minh : $P \implies WP(W, \text{true})$

γ) Định lý bất biến cơ sở (Fundamental invariance theorem) của Dijkstra phát biểu một dạng khác của luật về vòng lặp của Hoare .

Định lý: Giả sử $I \text{ and } B \text{ and } WP(S, \text{true}) \implies WP(S, I)$ (I là bất biến của vòng lặp)

thì : $I \text{ and } WP(W, \text{true}) \implies WP(\text{while } B \text{ do } S, I \text{ and not } B)$

$(\{I\} \text{ while } B \text{ do } S \{I \text{ and not } B\})$

Chứng minh : Ta sẽ chứng minh bằng quy nạp trên k rằng

$I \text{ and } P_k(\text{true}) \implies P_k(I \text{ and not } B)$ (a)

với : $P_0(Q) \equiv \text{not } B \text{ and } Q$

$P_k(Q) \equiv B \text{ and } wp(S, P_{k-1}(Q))$

Chú ý là $P_k(Q)$ là đkđ yếu nhất bảo đảm vòng lặp $\text{while } B \text{ do } S$ dừng sau đúng k lần lặp trong một trạng thái thoả mãn Q .

(i) Cơ sở $I \text{ and } P_0(\text{true}) \equiv I \text{ and } (\text{not } B \text{ and } \text{true})$ (định nghĩa)
 $\equiv \text{not } B \text{ and } (I \text{ and not } B)$
 $\equiv P_0(I \text{ and not } B)$

(ii) Bước quy nạp : Giả sử (a) đã đúng với k . Tức là :

$I \text{ and } P_k(\text{true}) \implies P_k(I \text{ and not } B)$

Ta chứng minh (a) đúng với $k+1$.

Thực vậy : $I \text{ and } P_{k+1}(\text{true}) \equiv I \text{ and } B \text{ and } WP(S, P_k(\text{true}))$ (định nghĩa)

$\equiv B \text{ and } I \text{ and } B \text{ and } WP(S, P_k(\text{true}))$

$\equiv B \text{ and } I \text{ and } B \text{ and } WP(S, \text{true}) \text{ and}$

$WP(S, P_k(\text{true}))$

(vì $WP(S, P_k(\text{true})) \equiv WP(S, \text{true}) \text{ and}$

$WP(S, P_k(\text{true}))$)

$\equiv B \text{ and } (I \text{ and } B \text{ and } WP(S, \text{true})) \text{ and}$

$WP(S, P_k(\text{true}))$

$\implies B \text{ and } WP(S, I) \text{ and } WP(S, P_k(\text{true}))$

Việc chứng minh tính đúng đầy đủ của lđkc phụ thuộc vào cấu trúc đoạn lệnh S và hệ tiên đề mà ta đã sử dụng để xây dựng lược đồ kiểm chứng hợp lý.

- Trường hợp 1 : Nếu đoạn lệnh S không chứa một lệnh lặp nào cả thì tính dừng được xem là hiển nhiên, khi đó 2 hệ tiên đề là hoàn toàn tương đương .

- Trường hợp 2 : Nếu đoạn lệnh S có chứa lệnh lặp thì tính dừng không phải bao giờ cũng được thỏa nên ta cần phải chỉ ra . Khi đó 2 hệ tiên đề là không tương đương .

+ Nếu trong suốt quá trình xây dựng lược đồ kiểm chứng ta chỉ sử dụng hệ tiên đề Dijkstra thì không phải kiểm chứng lại tính dừng nữa .

+ Nếu trong quá trình xây dựng lược đồ kiểm chứng ta có sử dụng (dù chỉ một lần) tiên đề của hệ Hoare thì phải kiểm chứng lại tính dừng (vì tiên đề Hoare không bảo đảm tính dừng) .

b) Kiểm chứng tính đúng đặc tả $\{P\} S \{Q\}$ khi S là một dãy lệnh tuần tự.

$$(S \equiv \{ S_1 ; S_2 ; \dots ; S_n \})$$

Kiểm chứng tính đúng đặc tả : $\{ P \} S_1 ; S_2 ; \dots ; S_n \{ Q \}$

Ví dụ :

Kiểm chứng đặc tả :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } (y * z^k = x^n) \} \quad (1)$$

$$k := k \text{ div } 2 ;$$

$$z := z * z ;$$

$$\{ (0 \leq k) \text{ and } (y * z^k) = x^n \} \quad (2)$$

Bài giải :

Cách 1 : Xây dựng lđkc hợp lý dựa vào hệ Hoare .

- Bước 1 : Xây dựng lược đồ kiểm chứng hợp lý.

+ Xây dựng lược đồ kiểm chứng chi tiết :

Từ (1) ta suy ra :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } (y * z^k) = x^n \} \quad (2a)$$

$$\{ (0 \leq k \text{ div } 2) \text{ and } (y * (z * z)^k \text{ div } 2 = x^n) \} \quad (2d)$$

$$k := k \text{ div } 2 ; \quad (2)$$

$$\{ (0 \leq k) \text{ and } (y * (z * z)^k = x^n) \} \quad (2c)$$

$$z := z * z ;$$

$$\{ (0 \leq k) \text{ and } (y * z^k = x^n) \} \quad (2b)$$

Diễn giải : Từ (2b) và lệnh gán $z := z * z$ dùng tiên đề gán ta suy ra (2c)

Từ (2c) và lệnh gán $k := k \text{ div } 2$ dùng tiên đề gán ta suy ra (2d)

+ Xây dựng lđkc hợp lý từ lđkc chi tiết :

Từ (2) ta suy ra :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } (y * z^k) = x^n \} \quad (2a)$$

$$\{ (0 \leq k \text{ div } 2) \text{ and } (y * (z * z)^k \text{ div } 2 = x^n) \} \quad (2d)$$

$$k := k \text{ div } 2 ; \quad (3)$$

$$\begin{aligned} & z := z * z ; \\ \{ (0 \leq k) \text{ and } (y * z^k = x^n) \} \end{aligned} \quad (2b)$$

Diễn giải : Từ (2b),(2c),(2d) và 2 lệnh gán tuần tự $z := z * z ; k := k \text{ div } 2$ dùng tiên đề tuần tự ta bỏ đi (2c) .

- Bước 2 : chứng minh đkc hợp lý (3) đúng :

$$\{ (0 \leq k \text{ div } 2) \text{ and } (y * (z * z)^k \text{ div } 2 = x^n) \} \quad (2d)$$

$$k := k \text{ div } 2 ; \quad (3a)$$

$$z := z * z ;$$

$$\{ (0 \leq k) \text{ and } (y * z^k = x^n) \} \quad (2b)$$

Ta có : Tính đúng của (3a) được khẳng định dựa vào cách xây dựng .

Kiểm chứng hai khẳng định đi liền nhau :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } (y * z^k) = x^n \} \{ (0 \leq k \text{ div } 2) \text{ and } (y * (z * z)^k \text{ div } 2 = x^n) \}$$

có quan hệ hàm ý (\implies) (hiển nhiên) (3b).

Từ (3a),(3b) áp dụng luật hệ quả ta suy ra (3) đúng .

Nhân xét :

Ta có thể hình thức hóa quá trình chứng minh bằng cách đưa vào ký hiệu :

$$I(z,k) \equiv (0 \leq k) \text{ and } (y * z^k = x^n)$$

Khi đó (2) có thể viết thành :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } I(z,k) \}$$

$$\{ I(z * z, k \text{ div } 2) \}$$

$$k := k \text{ div } 2 ;$$

$$\{ I(z * z, k) \}$$

$$z := z * z ;$$

$$\{ I(z,k) \}$$

(3) có thể viết thành :

$$\{ \text{even}(k) \text{ and } (0 < k) \text{ and } I(z,k) \}$$

$$\{ I(z * z, k \text{ div } 2) \}$$

$$k := k \text{ div } 2 ;$$

$$z := z * z ;$$

$$\{ I(z,k) \}$$

Điều kiện cần kiểm chứng là :

$$\text{even}(k) \text{ and } (0 < k) \text{ and } I(z,k) \implies I(z * z, k \text{ div } 2)$$

Chú ý : Khi có một cặp $\{P\} \{Q\}$ xuất hiện trong lược đồ thì khẳng định hàm ý (\implies) tương ứng là một điều kiện cần kiểm chứng (đkckc - verification condition). Các điều kiện này là cốt lõi của chứng minh về đkcđk, phần còn lại của chứng minh chỉ là việc áp dụng máy móc các quy luật.

Trong ví dụ trên, đkckc là :

$$\text{even}(k) \text{ and } (0 < k) \text{ and } I(z,k) \implies I(z^*z, k \text{ div } 2)$$

Đây chỉ là cách nói hình thức của sự kiện là $(z^*z)^k \text{ div } 2 = z^k$ khi k là số nguyên chẵn.

Cách 2 : Xây dựng lđkc hợp lý dựa vào hệ Dijkstra.

Bước 1 : Xây dựng lđkc hợp lý.

- Tính WP($k := k \text{ div } 2 ; z := z^*z, I(z,k)$)

Ta có : WP($k := k \text{ div } 2 ; z := z^*z, I(z,k)$)

$$\equiv \text{WP}(k := k \text{ div } 2, \text{WP}(z := z^*z, I(z,k)))$$

$$\equiv \text{WP}(k := k \text{ div } 2, I(z^*z, k)) \equiv I(z^*z, k \text{ div } 2)$$

+ Chèn WP($k := k \text{ div } 2 ; z := z^*z, I(z,k)$) vào (1) ta được lđcm hợp lý :

{ even(k) and (0 < k) and I(z,k) }

{ I(z^*z, k div 2) }

k := k div 2 ;

z := z^*z ;

{ I(z,k) }

Bước 2 : Kiểm chứng tính đúng của lđkc hợp lý .

Ta có : { I(z^*z, k div 2) }

k := k div 2 ;

z := z^*z ;

{ I(z,k) } (a) đúng

even(k) and (0 < k) and I(z,k) \implies I(z^*z, k div 2) (b) đúng.

Từ (a) , (b) ta suy ra đặc tả đúng

c) Kiểm chứng khi đoạn chương trình có chứa câu lệnh điều kiện

{P} if B then S1 else S2 {Q}

Khi đó ta thêm các khẳng định trung gian dạng:

{P} if B then {P and B} S1 {Q}

else {P and not B} S2 {Q}

(hoặc :

{P} if B then {P and B} S {Q}

else {P and not B} {Q}

khi không có phần else)

vào nơi có lệnh điều kiện tương ứng ta có lđkc trung gian thích hợp .

Ví dụ : Kiểm chứng đoạn chương trình :

{(0 < k) and (y * z^k = x^n)

if even(k) then begin

k := k div 2 ;

z := z^*z ;

```

        end
    else begin
        k := k - 1 ;
        y := y*z ;
    end

```

$\{(0 \leq k) \text{ and } (y * z^k = x^n)\}$

Cách 1: Dùng hệ tiên đề Hoare.

+ Bước 1 : Xây dựng ldc hợp lý.

Đặt $I(y,z,k) \equiv (0 \leq k) \text{ and } (y*z^k = x^n)$

Đặc tả có dạng : $\{(0 < k) \text{ and } I(y,z,k)\}$

```

    if even(k) then begin
        k := k div 2 ;
        z := z*z ;
    end
    else begin
        k := k - 1 ;
        y := y*z ;
    end
    {I(y,z,k)}

```

Chèn các khẳng định trung gian (dựa vào tiên đề gán của Hoare)

```

    {(0 < k) and I(y,z,k)}
    if even(k) then {even(k) and (0 < k) and I(y,z,k)}
    begin
        { I(y ,z*z , k div 2 ) }
        k := k div 2 ;
        { I(y , z*z , k ) }
        z := z*z ;
    end;
    {I(y,z,k)}
    else
    {(not even(k)) and (0 < k) and I(y,z,k)}
    begin
        { I(y*z ,z , k -1 ) }
        k := k - 1 ;
        { I(y*z , z , k ) }
        y := y*z ;
    end
    {I(y,z,k)}

```

Bỏ đi các khẳng định trung gian tầm thường (dựa vào luật tuần tự của Hoare) ta có ldc hợp lý :

```

    {(0 < k) and I(y,z,k)}
    if even(k) then {even(k) and (0 < k) and I(y,z,k)}

```

```

{ I(y ,z*z , k div 2 ) }
begin  k := k div 2 ;
      z := z*z ;
end
{I(y,z,k)}
else
{(not even(k)) and (0 < k ) and I(y,z,k)}
{ I(y*z ,z , k -1 ) }
  k := k -1 ;
  y := y*z ;
  {I(y,z,k)}

```

+ Bước 2: Chứng minh lược đồ kc đúng.

Các cặp khẳng định đứng liền nhau xuất hiện trong lược đồ :

{ even(k) and (0 < k) and I(y,z,k) } { I(y ,z*z , k div 2) } (a)

và { (not even(k)) and (0 < k) and I(y,z,k) } { I(y*z ,z , k -1) } (b)

Các hàm ý tương ứng cần phải chứng minh đúng :

{ even(k) and (0 < k) and I(y,z,k) } ==> { I(y ,z*z , k div 2) } (a*) (kiểm chứng ?)

và { (not even(k)) and (0 < k) and I(y,z,k) } ==> { I(y*z ,z , k -1) } (b*) (Kiểm chứng ?)

Từ (a*) và (b*) ta suy ra điều phải kiểm chứng.

Cách 2: Dùng hệ tiên đề Dijkstra.

- Bước 1 : Xây dựng ldc hợp lý.

Đặt $I(y,z,k) \equiv (0 \leq k) \text{ and } (y \cdot z^k = x^n)$

$S_1 \equiv \text{begin } k := k \text{ div } 2 ; z := z \cdot z ; \text{end} ;$

$S_2 \equiv \text{begin } k := k - 1 ; y := y \cdot z ; \text{end} ;$

$B \equiv \text{even}(k)$

Đặc tả có dạng : { (0 < k) and I(y,z,k) }

if B then S₁ else S₂

{I(y,z,k)}

+ Tính WP(if B then S₁ else S₂ , I(y,z,k))

$\equiv (B \implies \text{WP}(S_1 , I(y,z,k))) \text{ and } (\text{not } B \implies \text{WP}(S_2 , I(y,z,k)))$

$\equiv (\text{ dành cho người đọc })$

+ Chèn khẳng định WP(if B then S₁ else S₂ , I(y,z,k)) vào đặc tả theo tiên đề chọn của Dijkstra ta được ldc hợp lý dạng :

{(0 < k) and I(y,z,k)}

{ WP(if B then S₁ else S₂ , I(y,z,k)) }

if B then S₁ else S₂

{I(y,z,k)}

- Bước 2 : Chứng minh ldc đúng.

Cặp khẳng định đứng liền nhau trong lđkc là :

$\{(0 < k) \text{ and } I(y,z,k)\} \{ \text{WP}(\text{if } B \text{ then } S_1 \text{ else } S_2, I(y,z,k)) \}$

Ta cần chứng minh tính đúng của hàm ý tương ứng :

$\{(0 < k) \text{ and } I(y,z,k)\} \implies \{ \text{WP}(\text{if } B \text{ then } S_1 \text{ else } S_2, I(y,z,k)) \} (*)$
 (CM * danh cho người đọc)

Từ (*) suy ra điều phải kiểm chứng.

d) Kiểm chứng khi đoạn chương trình có chứa lệnh lặp while B do S.

Cách thứ 1: Sử dụng hệ tiên đề Dijkstra.

- Bước 1 : Xây dựng WP(While B do S) và chèn vào trước lệnh lặp .
 { WP(while B do S) while B do S
- Bước 2: Xây dựng lđkc hợp lý từ lđkc trên.
- Bước 3 : Chứng minh tính đúng của các điều kiện cần kiểm chứng.

Cách thứ 2 : Sử dụng hệ tiên đề Hoare.

- Bước 1 : Phát hiện bất biến I của vòng lặp và chèn các khẳng định trung gian tương ứng vào trước giữa và sau lệnh lặp (tiên đề Hoare) .

{(Invariant) I}
 while B do
 {I and B} S {I}
 {I and not B }

- Bước 2. Xây dựng lđkc hợp lý từ lđkc trên.
- Bước 3 : Chứng minh tính đúng của các điều kiện cần kiểm chứng .

Bước 4 : Chứng minh lệnh lặp dừng .

Ví dụ : Kiểm chứng đặc tả :

{ 0 <= n }
 y := 1 ; z := x ; k := n ;
 while (0 <> k) do
 begin
 k := k -1 ; y := y*z
 end
 { y = xⁿ }

Biết bất biến của vòng lặp : $I(y,z,k) \equiv (k \geq 0) \text{ and } (y*z = x^n)$

Bài giải theo cách 1:

Dựa vào hệ Hoare ta xây dựng lđkc chi tiết xuất phát từ điều kiện đầu , điều kiện cuối và bất biến .

{0 <= n}
 {I(1,x,n)}
 y := 1 ;
 {I(y,x,n)}
 z := x ;
 {I(y,z,n)}


```

k := n ;
{ I(y,z,k) }
while (0 <> k ) do begin
    { I(y,z,k) and (k <> 0 ) }
    { I(y*z,z,k-1) }
    k := k -1 ;
    { I(y*z,z,k) }
    y := y*z ;
    { I(y,z,k) }
end
{ I(y,z,k) and (k = 0 ) }
{ y = x^n }

```

Bỏ đi các khẳng định trung gian tầm thường ta có lđcm hợp lý dạng :

```

{ 0 <= n }
{ I(1,x,n) }
y := 1 ;
z := x ;
k := n ;
{ I(y,z,k) }
while (0 <> k ) do
begin
    { I(y,z,k) and (k <> 0 ) }
    { I(y*z,z,k-1) }
    k := k -1 ;
    y := y*z ;
    { I(y,z,k) }
end
{ I(y,z,k) and (k = 0 ) }
{ y = x^n }

```

Các điều kiện cần kiểm chứng là :

$$(0 \leq n) \implies I(1,x,n)$$

$$I(y,z,k) \text{ and } (k = 0) \implies y = x^n .$$

$$I(y,z,k) \text{ and } (k \neq 0) \implies I(y*z,z,k-1)$$

Thay $I(y,z,k) \equiv (0 \leq k) \text{ and } (y * z^k = x^n)$ ba đkckc trên sẽ trở thành :

(Phần chuẩn bị) $(0 \leq n) \implies (1 * x^n = x^n) \text{ and } (0 \leq n)$ (hiển nhiên)

(Phần kết thúc lặp) $(y * z^k = x^n) \text{ and } (0 \leq k) \text{ and } (k = 0) \implies y = x^n$ (hiển nhiên)

(Phần thân vòng lặp) $(y * z^k = x^n) \text{ and } (0 \leq k) \text{ and } (k \neq 0)$

$$\implies ((y*z)*z^{k-1} = x^n) \text{ and } (0 \leq k-1)$$

$$\equiv (y * z^k = x^n) \text{ and } (0 < k)$$

$$\implies (y*z^k = x^n) \text{ and } (0 \leq k-1) \text{ (hiển nhiên)}$$

$$I(k, tg) \text{ and } (k = n) \implies tg = S(i : 0 \leq i < n : a[i]) \quad (2)$$

Cuối cùng, ta có thể tìm được

$$\text{pre}(S_o, I(k, tg)) \equiv 0 = S(i : 0 \leq i < 0 : a[i])$$

và tập hợp các đkckc cho S_o bao gồm chỉ một điều kiện

$$(0 \leq n) \implies \text{pre}(S_o, I(k, tg)) \quad (3)$$

Như vậy, có 3 đkckc cho CT, đó là các điều kiện (1), (2), (3).

Quân tử luôn nói thật và tiểu nhân luôn nói dối. Một người hỏi một dân cư A trên đảo : "có phải anh là một quân tử?". A đáp : "nếu tôi là quân tử thì tôi thua tiền anh". Hãy chứng minh rằng : A nhất định phải thua tiền.

Ta mô hình hóa bài toán như sau :

Đặt các mệnh đề P : A là quân tử. Q : A phải trả tiền.

Kết luận phải chứng minh là Q.

Khảo sát giả thiết của bài toán:

+ Mệnh đề khẳng định : " A là tiểu nhân " là not P

+ A phát biểu một mệnh đề S. giả thiết cho biết : Nếu A là quân tử thì S phải đúng tức là : $P \implies S$

+ Nếu A là tiểu nhân thì S phải sai : $\text{not } p \implies \text{not } s$

+ S là một hàm ý : " Nếu A là quân tử thì A phải trả tiền".

Ta biểu thị S bởi : $p \implies q$

Như vậy tiền đề là : $(P \implies S) \text{ and } (\text{not } P \implies \text{not } S)$

theo luật tương đương (k) ta có thể viết là : $P \iff S$.

Bài toán được phát biểu dưới dạng thuần túy logic như sau :

Cho tiền đề $P \iff (P \implies Q)$

Có thể suy diễn được kết luận Q không ?

Ta sẽ xác lập rằng (lý luận trên là đúng) mệnh đề $(P \iff (P \implies Q)) \implies Q$ là đúng với mọi bộ giá trị đúng sai của các mệnh đề thành phần .

Có hai cách : (a) Dùng bảng giá trị đúng sai .

P	Q	$(P \iff (P \implies Q)) \implies Q$
T	T	(T \iff T) \implies T
F	T	(F \iff T) \implies T
T	F	(T \iff F) \implies F
F	F	(F \iff T) \implies F

(b) Dùng cách thay thế bằng các mệnh đề tương đương .

$$\begin{aligned}
 P \iff (P \implies Q) &\equiv P \iff (\text{not } P \text{ or } Q) && \text{(hàm ý)} \\
 &\equiv [(P \text{ and } (\text{not } P \text{ or } Q)) \text{ or } (\text{not } P \text{ and } \text{not } (\text{not } P \text{ or } Q))] \\
 &&& \text{(tương đương)}
 \end{aligned}$$

$$\begin{aligned}
 \text{mà } \text{not } P \text{ and } \text{not } (\text{not } P \text{ or } Q) &\equiv \text{not } P \text{ and } (\text{not } \text{not } P \text{ and } \text{not } Q) \\
 &\equiv \text{not } P \text{ and } (P \text{ and } \text{not } Q) \\
 &\equiv (\text{not } P \text{ and } P) \text{ and } \text{not } Q \\
 &\equiv \text{false and } \text{not } Q \equiv \text{false}
 \end{aligned}$$

$$\begin{aligned}
 \text{và } P \text{ and } (\text{not } P \text{ or } Q) &\equiv (P \text{ and } \text{not } P) \text{ or } (P \text{ and } Q) \\
 &\equiv \text{false or } (P \text{ and } Q) \\
 &\equiv P \text{ and } Q
 \end{aligned}$$

$$\text{Như vậy } P \iff (P \implies Q) \equiv P \text{ and } Q$$

$$\text{Từ đó } [P \iff (P \implies Q)] \implies Q \equiv (P \text{ and } Q) \implies Q$$

a) Các hằng: là các đối tượng cụ thể tồn tại trong lĩnh vực mà người ta đang khảo sát .

Ví dụ : + Các hằng số 5,6,10,2,...
+ Các hằng logic T(đúng) , F(sai)

Trong trường hợp tổng quát ,người ta thường đại diện cho các hằng bằng các chữ cái viết thường ở đầu bảng từ vựng: a,b,c...,a₁ ,b₁ , c₁ ,...

b) Các biến (Variable): là các tên tượng trưng . Mỗi biến được ấn định một miền giá trị là tập các đối tượng mà nó có thể nhận.

Ví dụ: + Các biến số nguyên n, j, k ,. . . với các tập trị là các tập con của tập số nguyên Z .

+ Các biến số thực x, y, z, . với các tập trị là các tập con của tập số thực R .

+ Các biến véc tơ V, W, . . . với các tập trị là các tập con của tập tích Đề Các $R \times R \times R \times \dots \times R$ (R^n)

Thường dùng các chữ cái viết thường ở cuối bảng từ vựng để biểu thị các biến : x,y,z,...,x₁ ,y₁ ,z₁ ,... Từ đây về sau ,mỗi biến nếu không được nói rõ đều được xem là biến nguyên .

c) Các toán tử (Operotors , hay hàm (functions)) là các ánh xạ từ các tập hợp đối tượng vào các tập hợp đối tượng trong lĩnh vực đang khảo sát. Ta sẽ thường dùng các toán tử toán học sau : + , - , * , / , div , mod

Một toán tử có thể có một hay nhiều toán hạng (ngôi) .

Ví dụ : + Toán tử "đối" (biểu thị bởi -) là một ngôi : -x
+ Toán tử - ,+ , - , * , / , div , mod là hai ngôi : 2 + 3 , x * y

d) Các hàm logic hay các tân từ (predicates) . Đó là các ánh xạ từ tập hợp các đối tượng vào tập boolean {true,false}, ta sẽ thường dùng các tân từ là các quan hệ toán học như sau :

- + Các quan hệ so sánh : = , <> , > , >= , < , <=
- + Các quan hệ tập hợp : \subseteq , \supseteq , ...
- + Các quan hệ khác : odd(x) kiểm tra xem x có lẻ không ?
even(x) kiểm tra xem x có chẵn không ?

e) Các liên từ logic : đây là các toán tử trên tập boolean mà ta gặp trong logic mệnh đề: and , or , not , ==> , <==>.

f) Các lượng từ phổ dụng \forall và tồn tại \exists (sẽ nói rõ ở mục sau)

Các biến logic , các tân từ trong đó có chứa các hằng hay biến hay hàm được gọi là các công thức cơ sở (formule elementaire)

Ví dụ : Các công thức cơ sở

- Biến logic : hôm-nay-trời-đẹp , tôi-về-lúc-8-giờ ,...
- tân từ : $5 > 2$
 $x > 5$
 $x + 5 > y - 3$

: $a[i] = x$ thoả điều sau $a[i]$ bằng x

Dạng chung là : \exists (danh sách biến : $R : P$)

Mệnh đề tồn tại chỉ đúng khi có một phần tử trong miền xác định bởi R thoả P .

khi $R = \text{true}$ thì ta có thể viết : \exists (danh sách biến :: P)

Ví dụ : cho hai array a và b

- Khẳng định : "trong array a không có thứ tự tăng"

$$\exists (i : 0 \leq i < n - 1 : a[i] > a[i+1])$$

- Khẳng định : "có ít nhất một phần tử của a lớn hơn mọi phần tử của b "

$$\exists (i : 0 \leq i < n : (j : 0 \leq j < n : a[i] > b[j]))$$

- Khẳng định "n là chẵn" : $\exists (m :: n = 2 * m)$

c) Một số tính chất:

- $(i : R : P) \equiv (i :: R \text{ and } P)$
- $(i : R : P) \equiv \text{not} (i :: R \text{ and not } P)$
- $(i : R : P) \equiv \text{not} (i : R : \text{not } P)$

d) Các biến tự do và bị buộc (free and band variables), phép thế(substitution)

Trong biểu thức $Q(i: r(i) : p(i))$ (ở đây ta xét Q là \forall hay \exists) biến i được gọi là bị buộc (band) vào lượng từ Q .

Những xuất hiện của một biến i không bị buộc vào một lượng từ nào đó trong biểu thức R , được gọi là tự do (free) trong R .

Ví dụ trong biểu thức : $\exists (d : p = q * d)$

các biến p và q là tự do , còn biến d là bị buộc . Các biến bị buộc chỉ đóng vai trò "giữ chỗ" và có thể được đổi tên , nếu tên này không trùng với một biến tự do đã có. Vì vậy , biểu thức trên tương đương với :

$$\exists (m :: p = q * m) \text{ nhưng hoàn toàn khác với : } (p :: p = q * p)$$

Về nguyên tắc , một tên biến có thể vừa tự do và bị buộc trong cùng một biểu thức

Ví dụ : Trong biểu thức $\forall (0 < i) \text{ and } (i : 0 \leq i < n : a[i] = 0)$

xuất hiện thứ nhất của i là tự do , còn xuất hiện còn lại là bị buộc.

Mặc dù ý nghĩa của biểu thức là rõ ràng nhưng nên tránh vì dễ gây nên nhầm lẫn .

Xét một tên từ chứa biến tự do .

Ví dụ : $\text{is-divisor}(q) \quad \exists (d :: p = q * d)$

Ta có thể thay các xuất hiện tự do của một biến bằng một biểu thức để được một tên từ mới.

Ví dụ: thế $2 * q$ cho q ta sẽ được tên từ $\text{is-divisor}(2 * q)$ mà dạng biểu thức của nó là : $\text{is-divisor}(2 * q) \quad \exists (d :: p = (2 * q) * d)$

Chú ý rằng trong $\exists (d :: p = q * d)$ biến p cũng tự do , nhưng vì ta không quan tâm đến phép thế cho p nên trong tên từ $\text{is-divisor}(q)$ ta chỉ nêu q để giảm bớt đi các chi tiết không cần thiết trong diễn giải.

Mỗi lượng từ mà ta xét ngoại trừ N là sự khái quát của các phép toán hai ngôi có tính giao hoán và kết hợp thành phép toán trên một tập bất kỳ.

Ví dụ :

S là khái quát của phép cộng ($+$), P là khái quát của phép nhân ($*$).

TRƯỜNG ĐẠI HỌC NÔNG NGHIỆP I - HÀ NỘI
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

TS. DƯƠNG XUÂN THÀNH



Giáo trình LẬP TRÌNH NÂNG CAO

(Trên ngôn ngữ Pascal)

(Soạn theo chương trình đã được Bộ GD&ĐT phê chuẩn)

Hà nội, 2005

Lời mở đầu

Cuốn giáo trình này được biên soạn theo đúng đề cương chi tiết môn học đã được Bộ Giáo dục và Đào tạo phê chuẩn. Thời gian học môn học này là 60 tiết trong đó có 10 tiết thực hành trên máy. Tác giả là người đã trực tiếp giảng dạy lập trình Pascal trong nhiều năm cho sinh viên chuyên tin và sinh viên các ngành khác.

Đối tượng sử dụng giáo trình là sinh viên chuyên ngành Tin học hệ đại học chính quy, tuy nhiên giáo trình cũng có thể sử dụng như là một tài liệu tham khảo cho sinh viên chuyên Tin học cao đẳng và những người muốn nghiên cứu nâng cao về lập trình.

Mục đích biên soạn cuốn giáo trình là cung cấp cho người đọc một tài liệu đơn giản, cô đọng những kiến thức về lập trình nâng cao. Người đọc có thể tự học mà không nhất thiết phải có thầy hướng dẫn.

Giáo trình bao gồm 6 chương và 4 phụ lục.

Chương 1: Chương trình con - Thủ tục và hàm, sinh viên đã được học qua trong chương trình Tin học đại cương, do vậy ở đây chủ yếu đi sâu vào khái niệm tham số, cách thức mà hệ thống dành bộ nhớ cho việc lưu trữ các tham số và việc gọi chương trình con từ chương trình con khác.

Chương 2: Các kiểu dữ liệu có cấu trúc, tập trung vào các kiểu dữ liệu mà sinh viên chưa được học như bản ghi có cấu trúc thay đổi, tập hợp..

Chương 3: Đơn vị chương trình và thư viện chuẩn, là chương chưa được học ở Tin học đại cương, ở đây hướng dẫn cách thiết kế các Đơn vị chương trình (Unit), cách thức sử dụng các Unit và tạo lập thư viện chương trình.

Chương 4: Con trỏ và cấu trúc động, là một chương khó, vì nó vừa liên quan đến quản lý bộ nhớ, vừa liên quan đến kiến thức của môn học Cấu trúc dữ liệu và Giải thuật do vậy trong chương này đã trình bày nhiều ví dụ để người đọc tham khảo.

Chương 5: Giải thuật đệ quy, được trình bày “hơi dài dòng” do đặc thù của tính đệ quy. Bài toán Tháp Hanoi được mô tả khác hoàn toàn so với tất cả các sách về Pascal đã có.

Chương 6: Đồ họa, ngoài việc giới thiệu các thủ tục vẽ thông thường, còn dành một phần trọng tâm cho việc xử lý ảnh Bitmap. Trong chương này có sử dụng một vài ví dụ của các tác giả khác (xem phần tài liệu tham khảo) nhưng đã được cải tiến đi rất nhiều.

Phụ lục 1: Bảng mã ASCII

Phụ lục 2: Tóm tắt các thủ tục và hàm của Turbo Pascal 7.0

Phụ lục 3: Định hướng biên dịch

Phụ lục 4: Thông báo lỗi

Các phụ lục đưa ra nhằm giúp người lập trình tiện tra cứu các thủ tục, hàm và xử lý các lỗi khi Pascal thông báo lỗi trên màn hình

Do phải bám sát đề cương và sự hạn chế về số trang tác giả nên trong giáo trình chưa đưa vào được phần xử lý âm thanh, lập trình hướng đối tượng....

Việc biên soạn lần đầu không thể tránh được thiếu sót, tác giả mong nhận được sự góp ý của bạn đọc và đồng nghiệp để lần xuất bản sau sẽ tốt hơn. Mọi góp ý xin gửi về địa chỉ:

Bộ môn Công nghệ Phần mềm, Khoa Công nghệ Thông tin,

Đại học Nông nghiệp I, Trâu quỳ, Gia lâm, Hà nội.

Xin trân trọng cảm ơn.

Hà nội, tháng 5 năm 2005

Ts. Dương Xuân Thành

Chương I

Chương trình con - Thủ tục và hàm

Khái niệm chương trình con đã được trình bày trong môn học Tin học đại cương, do vậy trong chương này chúng ta nhắc lại sơ qua một số khái niệm cũ và dành thời gian cho việc tìm hiểu sâu về tham số (tham biến và tham trị), lời gọi chương trình con, cách thức bố trí chương trình con trong thân chương trình mẹ. Sau khi học chương này bạn đọc cần nắm được các nội dung chủ yếu sau:

- Thế nào là biến toàn cục, biến địa phương
- Các biến toàn cục và biến địa phương được bố trí ở đâu
- Tầm tác dụng của từng loại biến
- Thứ tự xây dựng các chương trình con có ảnh hưởng thế nào đến toàn bộ chương trình
- Thế nào là tính đệ quy của chương trình con
- Lời gọi chương trình con thế nào là được phép
- Cách khai báo trước để gọi chương trình con không theo thứ tự thiết kế

1. Khái niệm về chương trình con

Chương trình con trong Pascal được hiểu là một chương trình nằm trong lòng một chương trình khác. Chương trình con gồm hai loại: **Thủ tục (Procedure)** và **hàm (Function)**. Các chương trình con được dùng rộng rãi khi xây dựng các chương trình lớn nhằm làm cho chương trình dễ theo dõi, dễ sửa chữa. Một đặc điểm nổi bật của chương trình con là nó có tính đệ quy nhờ thế mà nhiều bài toán sẽ được giải quyết dễ dàng.

Khi một chương trình con được gọi thì các biến được khai báo trong chương trình con (ta gọi là biến cục bộ) sẽ được cấp phát bộ nhớ. Kết thúc chương trình con, các biến cục bộ được giải phóng, điều này sẽ được lặp lại mỗi khi chương trình con được gọi và nó đồng nghĩa với việc thời gian xử lý bài toán sẽ tăng lên.

Bản thân tên gọi của hai loại chương trình con đã nói lên phần nào sự khác nhau giữa chúng. Function (Hàm) là một loại chương trình con cho kết quả là một giá trị vô hướng. Khi gọi tên Function với các tham số hợp lệ ta sẽ nhận được các giá trị, bởi vậy tên hàm có thể đưa vào các biểu thức tính toán như là các toán hạng. Procedure là loại chương trình con khi thực hiện không cho ra kết quả là một giá trị, mỗi Procedure nhằm thực hiện một nhóm công việc nào đó của chương trình mẹ, vì vậy tên của Procedure không thể đưa vào các biểu thức tính toán. Bằng cách xây dựng các chương trình con người lập trình có thể phân mảnh chương trình cho nhiều người cùng làm dưới sự chỉ đạo thống nhất của người chủ trì. Trong Turbo Pascal đã có sẵn một số chương trình con, ví dụ: $\sin(x)$, \sqrt{x} là các Function, còn $\text{read}()$, $\text{write}()$, $\text{gotoxy}(x1,x2)$ là các Procedure.

Trong một chương trình các chương trình con được bố trí ngay sau phần khai báo biến. Cấu trúc tổng quát một chương trình Pascal như sau:

```
PROGRAM tên_chương_trình;  
USES tên các UNIT; (*khai báo các đơn vị chương trình cần thiết*)  
LABEL          (*khai báo nhãn*);  
CONST          (*Khai báo hằng*);  
TYPE           (*định nghĩa kiểu dữ liệu mới*);  
VAR            (*khai báo biến*);
```

```
PROCEDURE Tên_CTC1 (danh sách tham số hình thức);  
Begin  
.....          (*thân thủ tục thứ nhất*);  
End;
```

```
PROCEDURE Tên_CTC2 (danh sách tham số hình thức);  
Begin  
.....          (*thân thủ tục thứ hai*);  
End;
```

```

FUNCTION Tên_HAM1(danh sách tham số hình thức):kiểu hàm;
Begin
.....          (*thân hàm thứ nhất*)
End;
.....

BEGIN          (*bắt đầu chương trình mẹ*)
.....
END.

```

Ghi chú:

1. Các chương trình con về nguyên tắc cũng bao gồm các phần khai báo báo như đối với một chương trình mẹ, phần nào không cần thiết thì không khai. Điều khác nhau cơ bản là thân chương trình con nằm giữa hai từ khoá Begin và End; (sau End là dấu ";" chứ không phải là dấu "." như trong chương trình mẹ) ngoài ra chương trình con còn có thể thêm phần khai báo các tham số hình thức, các tham số hình thức được đặt trong dấu () và viết ngay sau tên chương trình con.
2. Nếu chương trình con là Function thì cuối chương trình cần có lệnh gán giá trị vào tên chương trình con.

2. Tham số trong chương trình con

Các chương trình con có thể không cần tham số mà chỉ có các biến riêng (biến cục bộ). Trong trường hợp cần nhận các giá trị mà chương trình mẹ truyền cho thì chương trình con cần phải có các tham số (**Parameter**). Tham số được khai báo ngay sau tên chương trình con và được gọi là **tham số hình thức**.

Những giá trị lưu trữ trong các biến toàn cục của chương trình mẹ, nếu được truyền cho các thủ tục hoặc hàm thông qua lời gọi tên chúng thì được gọi là **Tham số thực**.

Tham số hình thức bao gồm hai loại:

2.1 Tham biến (Variabic parameter)

Tham biến là những giá trị mà chương trình con nhận từ chương trình mẹ, các giá trị này có thể biến đổi trong chương trình con và khi chương trình con kết thúc các giá trị này sẽ được trả về cho tham số thực.

Cách khai báo tham biến:

Tên chương trình con (Var tên tham biến : kiểu dữ liệu);

2.2 Tham trị (Value parameter)

Những tham số truyền vào cho chương trình con xử lý nhưng khi quay về chương trình mẹ vẫn phải giữ nguyên giá trị ban đầu thì được gọi là **tham trị**.

Cách khai báo tham trị:

Tên chương trình con (tên tham trị : kiểu dữ liệu);

Dưới đây là một ví dụ khai báo tham số:

PROCEDURE VIDU(x,y,z: integer; lam:boolean; var qq: char);

Câu lệnh khai báo chương trình con trên đây đồng thời khai báo các tham số hình thức trong đó x, y, z, lam là các tham trị, với x, y, z có kiểu integer, lam có kiểu boolean, qq là tham biến vì nó được viết sau từ khoá VAR.

Ví dụ 1.1: Lập chương trình tìm số lớn nhất trong n số nguyên được nhập từ bàn phím.

```
Program Tim_cuc_dai;
Uses Crt;
TYPE dayso = array[1..100] of integer;    (* Định nghĩa kiểu dữ liệu dayso là kiểu mảng
                                           gồm nhiều nhất là 100 phần tử*).
VAR   a: dayso                            (*khai báo biến của chương trình mẹ*)
      n: integer;
PROCEDURE nhapso(m:integer; var x:dayso);
      (* Nhập dãy số cần tìm cực đại vào mảng một chiều x[i]*)
      Var i : integer;                    (*khai báo biến cục bộ của chương trình con*)
Begin
  writeln('Nhap day so kieu integer);
  For i:=1 to m Do                        (* m được truyền từ chương trình mẹ qua tham số thực n*)
  Begin
    write('a[', i , ' ] = '); realln (x[i]);
  End; End;
FUNCTION Max(m: integer; b:dayso); integer;
(* Hàm MAX dùng để tìm số lớn nhất trong dãy số đã nhập, kiểu giá trị của hàm là kiểu integer *)
VAR
  i,t: integer;                          (* Biến riêng của hàm Max *)
Begin
  t:=b[1];                                (* Gán phần tử nhất của mảng b[i] cho biến t *)
  For i:=2 to m Do
  if t<b [i] then t:=b[i];
  Max:=t;                                  (* Gán giá trị cho chính hàm Max*)
End;
BEGIN                                     (* Thân chương trình mẹ *)
  Write('Ban can nhap bao nhieu so ? '); Readln(n);
  NHAPSO(N, A);                          (* Gọi chương trình con NHAPSO với 2 tham số thực là n và a. Hai tham
                                           số này sẽ thay thế cho hai tham số hình thức m, x trong chương trình con *)
  Writeln (' So lon nhat trong day so da nhap = ', MAX(n,a):5);
      (* Viết ra giá trị của hàm MAX với 2 tham số thực n,a độ dài số là 5 ký tự *)
  Repeat until keypressed;
END.
```

Ví dụ 1.1 là một chương trình bao gồm hai chương trình con, chương trình con thứ nhất là một thủ tục (Procedure), chương trình con thứ hai là một hàm (Function).

Chương trình mẹ có lệnh đọc số phần tử n của mảng dayso (tức là số lượng con số sẽ nhập vào). Vì mảng Dayso được khai báo có 100 phần tử nên không thể đọc vào nhiều quá 100 con số.

Sau đó là lệnh gọi chương trình con NHAPSO với 2 tham số thực là n , a , ở đây a là tham biến nghĩa là giá trị của mảng a sẽ được thay đổi trong chương trình con bởi tham số hình thức $x[i]$. Chương trình con nhập vào tham biến $x[i]$ kiểu mảng n con số thông qua tham trị m ($m=n$).

Lệnh viết giá trị lớn nhất của dãy số có kèm lời gọi hàm MAX vì hàm MAX thực chất trong trường hợp này chỉ là một con số.

Hàm MAX dùng để tìm số lớn nhất trong các số đã nhập, lời gọi hàm trong chương trình mẹ kèm theo việc truyền hai tham số thực là n và a thay thế cho hai tham số hình thức là m và b . Tên hàm được dùng như là một biến trong bản thân hàm khi ta dùng phép gán giá trị $MAX:=t$;

Chú ý:

1. Kiểu dữ liệu trong khai báo tham số hình thức chỉ có thể là: **số nguyên, số thực, ký tự, hoặc Boolean**. Nếu muốn đưa các kiểu dữ liệu có cấu trúc vào trong khai báo tham số thì phải định nghĩa trước kiểu dữ liệu này ở phần khai báo kiểu sau từ khoá Type (xem ví dụ 1.1).

2. Với kiểu dữ liệu chuỗi, nếu chúng ta khai báo tham số thực trong chương trình mẹ và tham biến trong chương trình con đều là STRING (không quy định độ dài tối đa của chuỗi) thì không cần phải định nghĩa trước kiểu dữ liệu ở phần TYPE. Để thấy rõ vấn đề chúng ta xét ví dụ sau đây:

Ví dụ: 1.2

```
Program Chuong_trinh_me;
```

```
Var s:string; m:byte
```

```
Procedure Chuong_trinh_con( Var a:string; n:byte);
```

Cách khai báo trên là được phép trong Pascal .

Nếu chúng ta quy định độ dài chuỗi như một trong ba dạng sau thì sẽ bị báo lỗi:

Dạng thứ nhất

```
Program Chuong_trinh_me;
```

```
Var s:string[30]; m:byte
```

```
Procedure Chuong_trinh_con( Var a:string[30]; n:byte);
```

Dạng thứ hai

```
Program Chuong_trinh_me;
```

```
Var s:string[30]; m:byte
```

```
Procedure Chuong_trinh_con( Var a:string; n:byte);
```

Dạng thứ ba

```
Program Chuong_trinh_me;
```

```
Var s:string; m:byte
```

```
Procedure Chuong_trinh_con( Var a:string[30]; n:byte);
```

Tuy nhiên có một ngoại lệ khi tham số hình thức trong các chương trình con không phải là tham biến mà là tham trị thì có thể khai báo theo dạng thứ hai.

Muốn quy định độ dài chuỗi trong các khai báo tham biến thì phải khai báo kiểu dữ liệu theo mẫu sau:

```
Program Chuong_trinh_me;
```

```
Type S1 = string[30];
```

```
Var s:s1; m:byte
```

```
Procedure Chuong_trinh_con( Var a:s1; n:byte);
```

3. Truyền tham số cho chương trình con

Trở lại ví dụ 1.1 ta thấy trong mỗi chương trình con có những tham số riêng của mình. Chương trình con nhập số đã sử dụng hai tham số hình thức là m và x. Hai tham số này được chuẩn bị để nhận các giá trị mà chương trình mẹ truyền cho thông qua lời gọi chương trình con với các tham số thực là n và b. Vì m được khai báo kiểu không có từ khoá Var nên nó là tham trị, nghĩa là khi chương trình con kết thúc thì giá trị của tham số thực n vẫn không thay đổi, tham số x là tham biến vì nó được khai báo sau từ khoá Var.

Khi tham số hình thức trong chương trình con là tham biến thì tham số thực trong chương trình mẹ phải là biến chứ không thể là hằng. Trong mọi trường hợp cả hai tham số thực và tham số hình thức đều phải cùng kiểu dữ liệu.

Các tham số thực truyền cho tham biến thì giá trị của nó có thể thay đổi trong chương trình con, khi ra khỏi chương trình con nó vẫn giữ nguyên các giá trị đã thay đổi đó. Trong ví dụ 1.1 tham số thực a là một mảng của n phần tử và tất cả các phần tử đều còn rỗng, khi truyền a vào tham biến x thì ở thời điểm ban đầu các phần tử của x cũng rỗng. Phép gán trong chương trình con NHAPSO sẽ làm thay đổi giá trị các phần tử của x, sau khi ra chương trình con nó giữ nguyên các giá trị đã gán tức là các giá trị ta nhập từ bàn phím vào các phần tử của mảng.

Khi tham số hình thức là tham trị thì tham số thực phải là một giá trị. Chương trình con nhận giá trị này như là giá trị ban đầu và có thể thực hiện các phép tính làm biến đổi giá trị đó, quá trình này chỉ tác động trong nội bộ chương trình con, khi ra khỏi chương trình con giá trị của tham số thực không biến đổi. Cụ thể trong ví dụ trên biến n nhận giá trị đọc từ bàn phím trong chương trình mẹ và được truyền cho tham số m trong cả hai chương trình con. Sau lời gọi chương trình con NHAPSO giá trị này có thể bị thay đổi nhưng khi rời NHAPSO đến lời gọi hàm MAX thì n lại vẫn giữ giá trị ban đầu.

Như vậy nếu muốn bảo vệ giá trị một tham số nào đó khi truyền chúng cho chương trình con thì phải qui định chúng là tham trị. Còn nếu muốn nhận lại giá trị mà chương trình con đã sinh ra thay thế cho những giá trị ban đầu có trong chương trình mẹ (hoặc là dùng những giá trị của chương trình con thay thế cho biến chưa được gán giá trị trong chương trình mẹ như ví dụ 1.1) thì tham số phải là tham biến.

Để thấy rõ hơn ý nghĩa của các tham số chúng ta xét ví dụ sau đây:

Người mẹ trao cho con trai một chiếc nhẫn và một túi tiền. Trước khi con đi làm ăn ở phương xa mẹ dặn: "Chiếc nhẫn là tín vật dùng để nhận lại gia đình và họ hàng khi con trở về, còn túi tiền là vốn ban đầu cho con kinh doanh".

Trong quá trình làm ăn, người con có thể cầm cố chiếc nhẫn nhưng khi trở về nhà nhất thiết phải mang chiếc nhẫn đó về, còn túi tiền khi quay về có thể nhiều lên cũng có thể ít đi, thậm chí không còn đồng nào. Trong ví dụ này chiếc nhẫn đóng vai trò tham trị, còn túi tiền đóng vai trò tham biến.

Vấn đề đặt ra là Pascal làm thế nào để đảm bảo các tính chất của tham trị và tham biến. Điều này sẽ được làm rõ khi nghiên cứu việc bố trí bộ nhớ (mục 5).

Khi lựa chọn tham số cần lưu ý một số điểm sau:

a. Kiểu của tham số trong chương trình con phải là các *kiểu vô hướng đơn giản* đã được định nghĩa sẵn trong Pascal hoặc đã được định nghĩa trong phần đầu của chương trình mẹ. Trong chương trình con không thể định nghĩa kiểu dữ liệu mới.

b. Chương trình con có thực sự cần tham số hay không? Nếu chương trình con chỉ sử dụng các biến toàn cục và biến địa phương cũng đáp ứng được yêu cầu của bài toán thì không nên dùng tham số. Nếu chương trình con thực hiện nhiều công việc trên cùng một loại đối tượng (đối tượng ở đây có thể là hằng, biến, hàm, thủ tục, kiểu), nghĩa là lời gọi chương trình con được lặp lại nhiều lần trên cùng một hoặc một nhóm đối tượng thì cần dùng đến tham số.

c. Nếu không muốn thay đổi giá trị của các tham số thực trong chương trình mẹ khi truyền nó cho chương trình con thì phải dùng tham số hình thức dưới dạng tham trị (trong phần khai báo kiểu không có từ khoá Var). Nếu cần thay đổi giá trị của tham số thực trong chương trình mẹ và nhận lại giá trị mà chương trình con đã xử lý thì tham số trong chương trình con phải là tham biến (tên tham số phải đặt sau từ khoá Var).

4. Biến toàn cục và biến địa phương

4.1 Biến toàn cục

Biến khai báo ở đầu chương trình mẹ được gọi là biến toàn cục. Nó có tác dụng trong toàn bộ chương trình, kể cả các chương trình con. Khi thực hiện chương trình máy dành các ô nhớ ở vùng dữ liệu (Data) để lưu giữ giá trị của biến.

Mở rộng ra tất cả các **đối tượng** trong Pascal (Kiểu dữ liệu, Hằng, Biến, Hàm, Thủ tục) khai báo trong chương trình mẹ được gọi là đối tượng toàn cục. Như vậy một kiểu dữ liệu đã được định nghĩa trong chương trình mẹ thì đương nhiên được phép sử dụng trong các chương trình con của nó.

Trong ví dụ 1.1 biến a và n là biến toàn cục, hai biến này có thể sử dụng trực tiếp trong thủ tục NHAPSO và trong hàm MAX mà không cần khai báo lại.

Pascal dành các ô nhớ ở vùng Data (vùng dữ liệu) cho các đối tượng toàn cục.

4.2 Biến địa phương

Những đối tượng khai báo trong chương trình con chỉ có tác dụng trong nội bộ chương trình con đó, chúng được gọi là đối tượng địa phương. Đối tượng hay được sử dụng nhất là Biến.

Biến địa phương có thể trùng tên với biến toàn cục song hệ thống dành các ô nhớ trong vùng nhớ ngăn xếp (Stack) để lưu giữ các giá trị của biến. Kết thúc chương trình con hệ thống sẽ giải phóng ô nhớ của biến địa phương dùng vào việc khác, giá trị của các biến lưu trữ trong các ô nhớ này sẽ không còn.

Trường hợp trong chương trình con lại có các chương trình con khác thì biến địa phương của chương trình con cấp trên lại được xem là biến toàn cục đối với chương trình con cấp dưới.

Một biến sau khi được khai báo trong một chương trình sẽ chỉ có tầm tác dụng trong bản thân chương trình đó và các chương trình con của nó. Biến này không có tác dụng trong các chương trình cùng cấp khác hoặc trong các chương trình con của chương trình khác. Điều này có nghĩa là các chương trình con và chương trình mẹ có thể có nhiều biến trùng tên, nhưng tầm tác dụng thì khác nhau do đó tính toàn vẹn của dữ liệu luôn được bảo đảm.

Ví dụ 1.3

```
Program Chuong_trinh_con;
Uses crt;
Var i,n:byte; c1:string[30];
Procedure Bien_dia_phuong;
Var i,n:byte; c1:string[30];
    Begin
        n:=3;
        C1:='Thu do Ha noi';
        Writeln('Gia tri n trong chuong trinh con: ',n);
        Writeln('Chuoai C1 trong chuong trinh con: ',C1);
    end;
Begin          (* thân chương trình mẹ *)
    Clrscr;
    Bien_dia_phuong;
    Writeln;
    n:=0;
    for i:= 1 to 10 do n:= n+i;
    c1:='Happy Birth Day';
    Writeln('Gia tri n trong chuong trinh me: ',n);
    Writeln('Chuoai C1 trong chuong trinh me: ',C1);
    Readln;
End.
```

Ví dụ 1.3 thiết kế một chương trình mẹ và một chương trình con dưới dạng thủ tục. Phần khai báo biến trong cả hai là như nhau. Phép gán dữ liệu vào biến n và chuỗi C1 là khác nhau.

Sau lời gọi chương trình con Bien_dia_phuong màn hình xuất hiện:

Gia tri n trong chuong trinh con: 3

Chuoai C1 trong chuong trinh con: Thu do Ha noi

Tiếp đó là kết quả các lệnh viết trong chương trình mẹ:

Giá trị n trong chương trình mẹ: 55

Chuỗi C1 trong chương trình mẹ: Happy Birth Day

Lời gọi chương trình con được thực hiện trước, kết quả là biến n mang giá trị 3, còn chuỗi C1 là 'Thu do Ha noi'. Khi trở về chương trình mẹ biến n = 55 còn chuỗi C1 = 'Happy Birth Day'. Điều này có nghĩa là biến n và C1 trong chương trình con không ảnh hưởng đến biến n và C1 trong chương trình mẹ.

5. Cách thức bố trí bộ nhớ

Khi một chương trình Pascal dạng EXE được chạy máy sẽ cấp phát một vùng nhớ cơ sở 640 Kb. Vùng nhớ này sẽ bao gồm các ô nhớ nằm liền nhau nghĩa là địa chỉ các ô nhớ tăng liên tục.

Phân loại vùng nhớ	Tên vùng	Dung lượng
Cao	Heap	0 - 655360 Bytes
	Stack Segment	16 - 64 Kb
	Data Segment	64 Kb
	Code Segment	Mỗi đoạn có 64 Kb
Thấp	Program Segment Prefix	256 Bytes

Hình 1.1

Chương trình được bố trí trong bộ nhớ như sau:

* Program Segment Prefix: ghi địa chỉ các hàm, biến, thủ tục

* Code Segment: lưu mã chương trình chính và mã các Unit liên quan đến chương trình, vùng này có thể gồm nhiều đoạn, mỗi đoạn 64 Kb.

* Data Segment: lưu trữ các biến, hằng, kiểu của chương trình chính, vùng này chỉ có 64 Kb nên nếu chương trình chính có quá nhiều hằng, biến thì có thể gặp lỗi: Too many variables

* Stack Segment: Lưu mã chương trình con và biến địa phương

* Heap: vùng nhớ tự do dùng cho việc cấp phát động

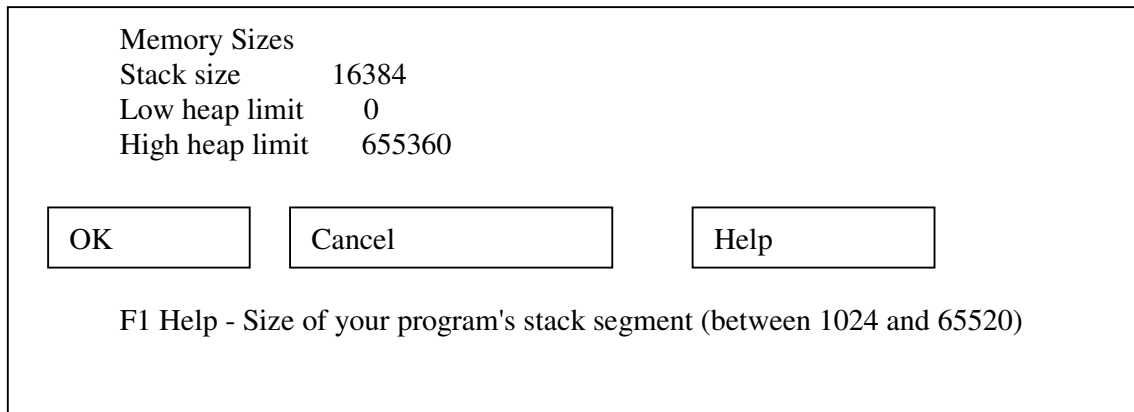
Các tham trị và biến cục bộ khai báo trong chương trình con được bố trí vào các ô nhớ của Stack. Khi chương trình mẹ gọi và truyền tham số cho chương trình con thì giá trị của các tham số này sẽ được sao chép vào các ô nhớ đã bố trí ở stack. Mọi biến đổi diễn ra trong stack không ảnh hưởng đến các giá trị của tham số thực trong chương trình mẹ.

Với các tham biến, Pascal không bố trí ô nhớ riêng mà sử dụng con trỏ trỏ vào địa chỉ của ô nhớ chứa biến toàn cục. Khi chương trình con làm thay đổi giá trị của các tham biến thì

cũng có nghĩa là nó làm thay đổi giá trị của các biến toàn cục trong chương trình mẹ. Kết thúc chương trình con chỉ các biến địa phương là bị giải phóng còn biến toàn cục không bị giải phóng cho nên chúng ta nói chương trình con đã mang các giá trị mới về cho chương trình mẹ.

Cần chú ý rằng Pascal 7.0 chỉ dành 16 Kb cho vùng Stack, dung lượng này đáp ứng đầy đủ các ứng dụng thông thường. Với những ứng dụng sử dụng tính đệ quy mà số lần gọi đệ quy là quá lớn thì sẽ có thể bị lỗi: Stack Overflow (bộ nhớ ngăn xếp bị tràn).

Gặp trường hợp này cần phải mở rộng vùng nhớ Stack bằng cách sau:
Trên thanh thực đơn chọn Options/Memory Size sẽ xuất hiện cửa sổ (hình 1.2)



Hình 1.2

Stack size 16384: dung lượng hiện thời của Stack

Size of your program's stack segment (between 1024 and 65520)

Có thể thay đổi dung lượng Stack trong khoảng 1024 - 65520 Bytes

Muốn thay đổi dung lượng của Stack chúng ta chỉ việc gõ dung lượng mới thay vào vị trí 16384 hiện thời.

Các tham số:

Low heap limit 0

High heap limit 655360 Là vùng nhớ tự do dành cho cấp phát động, không nên nhầm chúng với giá trị tối thiểu và tối đa của Stack.

6. Tính đệ qui của chương trình con

Thông thường lời gọi một chương trình con chỉ được thực hiện khi chương trình con đó đã được thiết kế hoàn chỉnh. Tuy nhiên Pascal cho phép một chương trình con ngay trong quá trình xây dựng lại có thể gọi tới chính nó, không những thế từ một chương trình con còn có thể gọi tới các chương trình con khác cùng cấp hoặc chương trình con cấp cao hơn nó.

Một chương trình con có thể có lời gọi tới chính tên chương trình con đó, tính chất này được gọi là tính "Đệ qui của chương trình con". Đệ qui có thể sử dụng trong cả Procedure và Function. Giống như mảng trong Pascal tương đương với ma trận trong toán, đệ qui trong Pascal tương đương với tính Quy nạp trong toán học. Về điều này chúng ta sẽ đề cập đến trong chương V "Giải thuật Đệ qui".

7. Lời gọi chương trình con

Một chương trình mẹ có thể có nhiều chương trình con trực thuộc, bên trong mỗi chương trình con lại có thể có các chương trình con riêng. Nói cách khác trong Pascal tồn tại một lớp chương trình con ngang cấp nhau, mỗi chương trình con này lại có thể đóng vai trò chương trình mẹ của một lớp chương trình con khác.

Khi thiết kế, mỗi chương trình con phải là một khối riêng biệt không thể lồng nhau hoặc có các lệnh nhảy Goto từ chương trình con này tới chương trình con khác.

7.1 Gọi chương trình con từ trong chương trình mẹ

Lời gọi chương trình con có thể đặt bất kỳ chỗ nào trong chương trình mẹ. Nếu chương trình con là một thủ tục thì lời gọi chương trình con (tức là tên chương trình con) có thể tạo nên một câu lệnh, ví dụ:

```
Readln; Gotoxy(5,8);
```

Nếu chương trình con là hàm thì tên hàm không thể tạo nên một câu lệnh, nói khác đi tên hàm phải nằm trong một biểu thức hay trong một thủ tục nào đó, ví dụ với hàm khai căn bậc hai SQRT() chúng ta không thể viết

```
sqrt(9);
```

Điều này là dễ hiểu vì hàm cho ta giá trị vô hướng, giá trị này không phải là một lệnh do đó Pascal không biết phải làm gì với giá trị đó.

Cách gọi hàm như sau là hợp lệ:

```
a:= sqrt(9) + 5;
```

```
Writeln('Can bac hai cua 9 bang ',sqrt(9));
```

7.2 Gọi chương trình con từ chương trình con khác

Các chương trình con cùng cấp có thể gọi tới nhau và truyền tham số cho nhau.

Nguyên tắc gọi là: những chương trình con xây dựng sau có thể gọi tới các chương trình con đã xây dựng trước nó, đồng thời các chương trình con cấp dưới cũng có thể gọi tới các chương trình con cấp trên nếu chúng cùng một gốc. Điều này có nghĩa là các chương trình con xây dựng trước không thể gọi tới các chương trình con xây dựng sau nếu không có chỉ báo FORWARD (xem mục 8). Xét một số ví dụ sau:

Ví dụ 1.6

```
Program Goi_ctc;
```

```
Uses crt;
```

```
Type dayso=array[1..60] of byte; s1=string[30];
```

```
Var
```

```

a:s1; b:dayso; i,j,n:byte;
Procedure nhapso(m:byte; var c:dayso);
Begin
  For i:=1 to m do
  Begin
    Write('c['i,'] = '); readln(c[i]);
  End;
End;
Function tinh tong(m:byte; d:dayso):real;
var tong:real;
Begin
  tong:=0;
  for i:= 1 to m do Tong:=tong+d[i];
  tinh tong:=tong;
End;
Procedure viet(k:byte; e:dayso);
Begin
  Write('Tong cac phan tu mang = ',tinh tong(k,e):8:0);
  readln;
End;

BEGIN
  clrscr;
  write('Nhap so phan tu n '); readln(n);
  nhapso(n,b);
  viet(n,b);
END.

```

Ví dụ 1.6 thiết kế ba chương trình con là Nhapso, Tinh tong và Viet. Thủ tục Nhapso dùng để nhập các phần tử vào mảng một chiều. Hàm Tinh tong dùng để tính tổng các phần tử mảng và thủ tục Viet dùng để hiện kết quả tính tổng lên màn hình.

Chương trình mẹ gọi chương trình con Viet và truyền các tham số là số phần tử mảng n và giá trị của các phần tử của mảng (mảng b). Chương trình con Viet lại gọi hàm Tinh tong và truyền các tham số cho hàm này. Đây là trường hợp một chương trình con gọi một chương trình con cùng cấp.

Việc các chương trình con gọi tới nhau phải tuân theo quy định sau đây:

Một chương trình con chỉ có thể gọi tới một chương trình con cùng cấp đã thiết kế trước chương trình con hiện thời.

Trong ví dụ 1.6 nếu chúng ta đưa hàm Tinh tong xuống dưới thủ tục Viet thì khi chạy chương trình sẽ bị báo lỗi:

Unknown Identifier.

Ví dụ 1.7 dưới đây trình bày cách thức mà một chương trình con cấp dưới gọi tới một chương trình con cấp trên cùng gốc.

Ví dụ 1.7

Program gọi_ctc;

Uses crt;

Type dayso=array[1..60] of byte;

Var b:dayso; i,n:byte;

{Hai chương trình con Nhapso và Tinh tong cùng cấp với thủ tục Xuly}

Procedure nhapso(m:byte; var c:dayso);

Begin

For i:=1 to m do

Begin

Write('c[' ,i,'] = '); readln(c[i]);

End;

End;

Function tinh tong(m:byte; d:dayso):real;

Var tong:real;

Begin

tong:=0;

for i:= 1 to m do Tong:=tong+d[i];

tinh tong:=tong;

End;

Procedure xuly(j:byte; ds:dayso);

Procedure viet(k:byte; e:dayso);

Var i:byte;

Begin

Writeln('Tong cac phan tu mang = ', tinh tong(k,e):8:0);

Writeln('Day so sap xep giam dan ');

for i:=1 to k do write(e[i], ' ');

readln;

End; { Kết thúc thủ tục Viet}

Procedure sapxep(m:byte; d:dayso);

Var p,q:byte; Tg:byte;

Begin

For p:= 1 to m-1 do

```

For q:=p+1 to m do
If d[p]<d[q] then
  Begin
    tg:=d[p]; d[p]:=d[q]; d[q]:=tg;
  End;
viet(m,d);
End; { Kết thúc thủ tục sapxep}

Begin {than thu tục Xuly}
  Writeln('Thu tục xu ly dung de sap xep va viet ket qua');
  sapxep(j,ds);
end; { Kết thúc thủ tục Xuly}

BEGIN {Than chương trình me}
  clrscr;
  write('Nhap so phan tu n '); readln(n);
  nhapso(n,b);
  xuly(n,b);
END.

```

Ví dụ 1.7 có ba chương trình con cùng cấp là Nhapso, Tinhtong và Xuly. Trong thủ tục Xuly có hai chương trình con là Viet và Sapxep trong đó chương trình con Viet được thiết kế trước, Sapxep được thiết kế sau. Chương trình con Sapxep có lời gọi đến chương trình con Viet cùng cấp với nó, mục đích của lời gọi này là truyền cho chương trình con Viet những dữ liệu mảng đã sắp xếp giảm dần. Chương trình con Viet có lời gọi đến hàm Tinhtong là một chương trình con cấp cao hơn nó, vì Tinhtong đã được thiết kế trước Xuly nên lời gọi là hợp lý. Nếu đảo vị trí của hai chương trình con Viet và Sapxep, nghĩa là đưa chương trình con Viet xuống sau Sapxep thì sẽ bị báo lỗi, về điều này chúng ta sẽ nghiên cứu ở mục 8.

8. Khai báo trước bằng Forward

Như đã nêu trong mục 7 việc các chương trình con gọi tới nhau bị hạn chế bởi thứ tự xây dựng các chương trình con đó. Vì những lý do khác nhau người ta không thể thay đổi thứ tự xây dựng các chương trình con nhưng lại muốn các chương trình con phải gọi được tới nhau không phụ thuộc vào thứ tự xây dựng chúng. Để làm việc này Pascal cho phép sử dụng từ khoá Forward. Nghĩa đen của từ Forward là "phía trước" thường được dùng để báo hiệu một cái gì đó ta sẽ gặp sau này ví dụ: phía trước 200 mét là công trường.

Cú pháp:

Tên chương trình con (danh sách tham số); Forward;

Dòng khai báo trên đây phải được đặt trong chương trình trước khi xây dựng tất cả các chương trình con. Khi tên một chương trình con đã được khai báo với cú pháp nêu trên thì việc nó nằm trước hay sau một chương trình con sẽ gọi tới nó là không quan trọng. Số lượng chương trình con khai báo trước với từ khoá Forward là không hạn chế.

Cần lưu ý rằng nếu có nhiều chương trình con cần khai báo trước thì mỗi tên chương trình con phải đi với một từ khoá Forward, không thể ghép nhiều tên chương trình con với cùng một từ Forward.

Ví dụ 1.8

```
Program Tu_khoa_Forward;
uses crt;
Type dayso=array[1..60] of byte;
var
  a:string; b:dayso;
  i,j,n:byte;

Function c2(m:byte; d:dayso):real; forward;
Procedure c4(p:byte; var q:dayso); forward;

Procedure c1(m:byte; var c:dayso);
Begin
  For i:=1 to m do
  Begin
    Write('c[' ,i, ' ] = '); readln(c[i]);
  End;
End;

Procedure c3(k:byte; e:dayso);
  Var i:byte;
Begin
  c4(k,e);
  writeln('Mang sau khi sap xep');
  for i:= 1 to k do write(e[i], ' ');
  writeln;
  Write('Tong cac phan tu mang = ',c2(k,e):8:0);
  readln;
End;

Function c2(m:byte; d:dayso):real;
var tong:real;
```



```

Begin
tong:=0;
for i:= 1 to m do Tong:=tong+d[i];
c2:=tong;
End;

```

```

Procedure c4(p:byte; var q:dayso);
  Var i,j:byte; tg:integer;
  Begin
  for i:= 1 to (p-1) do
  for j:= i+1 to p do
  if q[i]>q[j] then
  Begin
  tg:=q[i]; q[i]:=q[j]; q[j]:=tg;
  End;
  End;

```

```

BEGIN
  clrscr;
  write('Nhap so phan tu n '); readln(n);
  c1(n,b);
  c3(n,b);
END.

```

Ví dụ 1.8 có 4 chương trình con trong đó c1 và c3 thiết kế trước còn c2 và c4 thiết kế sau. Trong c3 có lời gọi đến c2 và c4 do vậy phải khai báo trước c2 và c4 . Nếu không muốn khai báo trước thì cần đưa c2 và c4 lên trên c3.

Bài tập ứng dụng chương 1

1. Lập chương trình tính diện tích toàn phần và thể tích các hình : Trụ tròn, nón.

Yêu cầu: Thiết kế menu theo mẫu sau, Menu có thể dùng con trỏ dịch chuyển để chọn các chức năng: **Hình Trụ** **Hình Nón** **Ket thuc**

Việc tính toán diện tích, thể tích mỗi hình ứng với một chương trình con

Tất cả các hình đều chung một chương trình con hiện kết quả. Chức năng Ket thuc dùng để quay về cửa sổ Pascal.

2. Lập một chương trình tạo thực đơn với các chức năng:

Tính giai thừa **Tính tổ hợp** **Trở về**

Dùng các chương trình con đã lập để giải bài toán sau: Cho n điểm trên màn hình, qua hai điểm bất kỳ bao giờ cũng nối được bởi một đoạn thẳng. Tính xem có bao nhiêu đoạn thẳng được tạo ra. Tìm đoạn ngắn nhất và dài nhất, chúng nối các điểm nào?

3. Thiết kế thực đơn với các chức năng:

1.giai he pt bac nhat 2. giai pt bac hai 3.Ket thuc

Yêu cầu: Bấm số để chọn chức năng trên thực đơn. Chức năng Ket thuc dùng để quay về cửa sổ Pascal.

Chương trình có 2 chương trình con để giải hệ phương trình bậc nhất 2 ẩn và giải phương trình bậc 2

4. A,B là mảng hai chiều của các số thực, số dòng và cột của từng mảng nhập từ bàn phím, lập chương trình bao gồm các chương trình con: nhập dữ liệu vào mảng, kiểm tra xem có thể nhân hai mảng hay không, nếu có thì chạy chương trình con nhân hai mảng, nếu không thì thông báo không thể nhân hai mảng. Hiện kết quả nhân dưới dạng ma trận.

5. Cho hai chuỗi s1, s2, viết chương trình bao gồm các chương trình con:

NHAP dùng để nhập vào s1, s2 các ký tự của bảng mã gồm cả chữ cái và chữ số,

TACH dùng để tách riêng các chữ số và chữ cái, những chữ số tách ra lưu vào mảng một chiều theo thứ tự của s1 trước, s2 sau.

CONG dùng để cộng các chữ số tách ra từ hai chuỗi

Thông báo kết quả theo mẫu:

Chuỗi s1 sau khi tách:.....

Chuỗi s2 sau khi tách:.....

Tổng các chữ số:.....

6. Lập chương trình với 4 chương trình con dùng để chuyển đổi các số giữa 4 hệ đếm:

Hệ 10 sang hệ 2, 8, 16

Hệ 2 sang hệ 8, 10, 16

Hệ 8 sang hệ 2, 10, 16

Hệ 16 sang hệ 2, 8, 10

Chương 2

Các kiểu dữ liệu có cấu trúc

Trong chương này không trình bày chi tiết các kiểu dữ liệu có cấu trúc đơn giản như kiểu mảng, chuỗi. Nội dung trọng tâm của chương là kiểu bản ghi (Record) có cấu trúc thay đổi, kiểu tệp và kiểu tập hợp. Chương này bạn đọc cần nắm được các vấn đề sau:

- Cách thức định nghĩa một kiểu dữ liệu mới
- Khai báo biến với các kiểu dữ liệu do người lập trình tự định nghĩa
- Cách sử dụng toán tử CASE khi khai báo bản ghi có cấu trúc thay đổi
- Cách thức ghi và đọc dữ liệu cho ba loại tệp: tệp văn bản, tệp có kiểu và tệp không kiểu, chú trọng cách ghi dữ liệu kiểu số vào tệp văn bản và lấy số liệu ra để xử lý
- Xử dụng dữ liệu kiểu tập hợp trong lập trình

1. Dữ liệu kiểu bản ghi (record)

1.1 Khái niệm cơ bản

Kiểu bố trí dữ liệu thông dụng nhất mà con người nghĩ ra là bố trí dưới dạng bảng. Bảng được coi là một đối tượng (để quản lý hoặc nghiên cứu), bảng bao gồm một số cột và một số dòng. Số cột, dòng trong bảng phụ thuộc vào phần mềm quản lý mà chúng ta sử dụng. Trong từng cột dữ liệu có tính chất giống nhau. Các phần mềm quản trị dữ liệu như Excel, Foxpro... đều ứng dụng khái niệm bảng và Pascal cũng không phải là ngoại lệ. Để có được một bảng trước hết Pascal xây dựng nên một dòng gọi là "bản ghi", tập hợp nhiều dòng sẽ cho một bảng, mỗi bản ghi được ghi vào bộ nhớ dưới dạng một tệp.

Bản ghi (Record) là một cấu trúc bao gồm một số (có định hoặc thay đổi) các phần tử có kiểu khác nhau nhưng liên quan với nhau. Các phần tử này gọi là các trường (Field). Ví dụ bảng điểm của lớp học bao gồm các trường Hoten, Ngaysinh, Gioitinh, Lop, Diachi, Toan, Ly, Hoa,...., dữ liệu điền vào các trường hình thành nên một bản ghi (Record). Có thể có những trường trong một bản ghi lại là một bản ghi, ví dụ trường Ngaysinh ở trên có thể là một bản ghi của ba trường là Ngay, Thang, Nam. Bản ghi không phải là kiểu dữ liệu đã có sẵn trong Pascal mà do người sử dụng tự định nghĩa do đó chúng phải được khai báo ở phần TYPE.

Bản ghi bao gồm hai loại:

* Bản ghi có cấu trúc không đổi : là loại bản ghi mà cấu trúc đã được định nghĩa ngay từ khi khai báo và giữ nguyên trong suốt quá trình xử lý.

* Bản ghi có cấu trúc thay đổi: là loại bản ghi mà cấu trúc của nó (tên trường, số trường, kiểu trường) thay đổi tùy thuộc vào những điều kiện cụ thể. Loại bản ghi này khi khai báo thì vẫn khai báo đầy đủ song khi xử lý thì số trường có thể giảm đi (so với cấu trúc đã khai báo) chứ không thể tăng lên.

Điểm mạnh của Bản ghi là cho phép xây dựng những cấu trúc dữ liệu đa dạng phục vụ công việc quản lý, tuy vậy muốn lưu trữ dữ liệu để sử dụng nhiều lần thì phải kết hợp kiểu Bản ghi với kiểu Tệp.

1.2 Khai báo

Kiểu dữ liệu của các trường trong Record có thể hoàn toàn khác nhau và được khai báo sau tên trường, những trường có cùng kiểu dữ liệu có thể khai báo cùng trên một dòng phân cách bởi dấu phẩy ",". Cuối mỗi khai báo trường phải có dấu ";" .

Kiểu dữ liệu Record được khai báo như sau:

```
TYPE
<Tên kiểu > = RECORD
    <Tên trường 1>: Kiểu;
    <Tên trường 2>: Kiểu;
    .....
    <Tên trường n>: Kiểu;
END;
```

Ví dụ 2.1

Khai báo kiểu dữ liệu BANGDIEM bao gồm một số trường nhằm phục vụ việc quản lý điểm.

```
TYPE
BANGDIEM = RECORD
    Hoten: String[25];
    Gioitinh: Char;
    Lop: String[5];
    Diachi: String[30];
    Toan,Ly,Hoa: Real;
END;
```

Với khai báo như trên dung lượng bộ nhớ dành cho các trường (tính bằng Byte) sẽ là: Hoten 26, Gioitinh 1, Lop 6, Diachi 31, Toan 6, Ly 6, Hoa 6. (Các trường kiểu String bao giờ cũng cần thêm 1 Byte chứa ký tự xác định độ dài chuỗi).

Tổng độ dài của Record bằng $26+1+6+31+18=82$ Bytes.

Có thể dùng hàm Sizeof(tên kiểu) để xác định độ dài một kiểu dữ liệu, ví dụ:

Write(sizeof(bangdiem)) sẽ nhận được số 82

Ví dụ 2.2

Xây dựng kiểu dữ liệu quản lý hồ sơ công chức. Chúng ta sẽ tạo ra bốn kiểu dữ liệu mới đặt tên là Diadanh, Donvi, Ngay và Lylich.

```
Type
Diadanh = Record
    Tinh, Huyen, Xa, Thon: String[15];
End;
```

```
Donvi = Record
    Truong: String[30];
    Khoa, Bomon: String[20]
End;
```

```
Ngay = Record
    Ng: 1..31;
    Th: 1..12;
    Nam: Integer;
End;
```

```
Lylich = Record
    Mhs: Word;
```

```
Hoten: String[25];
Ngaysinh: Ngay;
Quequan: Diadanh;
Coquan: Donvi;
End;
```

Trong cách khai báo trên trường Ngaysinh thuộc kiểu Ngay, Quequan thuộc kiểu Diadanh, Coquan thuộc kiểu Donvi, nói cách khác ba trường này lại chính là ba Record.

Để khắc phục cách khai báo nhiều kiểu bản ghi như trên có thể sử dụng các bản ghi lồng nhau. Kiểu bản ghi lồng nhau có thể khai báo trực tiếp, nghĩa là không cần khai báo riêng rẽ các bản ghi con.

Ví dụ 2.3

```
Uses crt;
Type
  Lylich=record
  Mhs:word;
  Hoten:string[25];
  Ngaysinh:record
    Ng:1..31;
    Th:1..12;
    Nam:Integer;
  End;
  Quequan:record
    Tinh,Huyen,xa,thon:string[15];
  End;
  Coquan:record
    Truong:string[30];
    Khoa, Bomon:string[20];
  End;
End;
.....
```

Ngoài cách khai báo kiểu rồi mới khai báo biến, Pascal cho phép khai báo trực tiếp biến kiểu bản ghi theo cú pháp sau:

```
Var
  Tên biến:Record
    Tên trường 1:kiểu trường;
    Tên trường 2:kiểu trường;
  .....
```

End;

1.3 Truy nhập vào các trường của bản ghi

Sau khi đã khai báo kiểu dữ liệu ta phải khai báo biến, giả sử cần quản lý danh sách cán bộ một trường đại học chúng ta phải khai báo một biến chứa danh sách viết tắt là DS. Khi đó ta phải khai

VAR

DS: Lylich;

Giống như hai kiểu dữ liệu Mảng và Chuỗi, việc xử lý được thực hiện trên các phần tử của mảng hoặc chuỗi. ở đây mặc dù DS là một biến nhưng chúng ta không thể xử lý chính biến đó mà chỉ có thể xử lý các trường của biến DS. Để truy nhập vào trường cần viết:

<Tên biến>.<Tên trường mẹ>.<tên trường con>....

Ví dụ để nhập dữ liệu cho trường Hoten ta viết các lệnh:

Write(' Ho va ten can bo: '); Readln(DS.hoten);

Lệnh Readln(DS.hoten); cho phép ta gán Họ tên cán bộ vào trường Hoten của bản ghi hiện thời.

Để nhập ngày tháng năm sinh chúng ta phải truy nhập vào các trường con

Readln(Ds.Ngay.Ngays);

Readln(Ds.Ngay.Thang);

Readln(Ds.Ngay.Nam);

Lệnh viết dữ liệu ra màn hình cũng có cú pháp giống như lệnh nhập.

Writeln(DS.Hoten);

Writeln(Ds.Ngay.Ngays);

.....

Chú ý:

Khi khai báo biến DS kiểu LYLICH chúng ta có thể nhập dữ liệu vào biến DS nhưng chỉ nhập được một bản ghi nghĩa là chỉ nhập dữ liệu được cho một người. Nếu muốn có một danh sách gồm nhiều người thì phải có nhiều bản ghi, để thực hiện điều này chúng ta có thể xây dựng một mảng các bản ghi. Trình tự các bước như sau:

* Định nghĩa kiểu dữ liệu bản ghi

* Khai báo biến mảng với số phần tử là số người cần quản lý, kiểu phần tử mảng là kiểu Bản ghi đã định nghĩa. (xem ví dụ 2.4)

Với tất cả các trường khi truy nhập ta luôn phải ghi tên biến rồi đến tên trường mẹ, tên trường con, ... điều này không chỉ làm mất thời gian mà còn khiến cho chương trình không đẹp, Pascal khắc phục nhược điểm này bằng cách đưa vào lệnh WITH... DO.

1.4 Lệnh WITH...DO

Cú pháp của lệnh:

WITH <Tên biến kiểu RECORD> DO <Các lệnh>

Khi sử dụng lệnh WITH...DO chuỗi lệnh viết sau DO chỉ cần viết tên trường có liên quan mà không cần viết tên biến. Xét ví dụ nhập điểm cho lớp học với giả thiết lớp có nhiều nhất là 40 học sinh.

Ví dụ 2.4:

```
Program Nhapdiem;
```

```
Uses CRT;
```

```
Type
```

```
BANGDIEM = RECORD
```

```
    Hoten: String[25];
```

```
    Gioitinh: ('T','G');    (* kiểu liệt kê, 'T' = Trai, 'G' = Gái *)
```

```
    Lop: String[5];
```

```
    Diachi: String[50];
```

```
    Toan,Ly,Hoa: Real;
```

```
End;
```

```
Var
```

```
    DS_LOP: Array[1..40] of BANGDIEM    (* danh sách lớp là mảng 40 phần tử*)
```

```
    i,j: Integer;
```

```
    lam: Char;
```

```
BEGIN
```

```
    clrscr;
```

```
    lam:='C'; i:=1;
```

```
    Repeat
```

```
    With DS_LOP[i] Do
```

```
    Begin
```

```
        Write(' Ho va ten hoc sinh: '); Readln(Hoten);
```

```
        Write(' Trai hay gai T/G: '); Readln(Gioitinh);
```

```
        Write(' Thuoc lop: '); Readln(Lop);
```

```
        Write(' Cho o thuong tru: '); Readln(Diachi);
```

```
        Write(' Diem toan: '); Readln(Toan);
```

```
        Write(' Diem ly: '); Readln(Ly);
```

```
        Write(' Diem hoa: '); Readln(Hoa);
```

```
    End;
```

```
    i:=i+1;
```

```
    Write(' NHAP TIEP HAY THOI ? C/K '); Readln(lam);
```

```
    Until upcase(lam)='K';
```

```
    clrscr;
```

```
    For j:=1 to i-1 do
```

```
    With DS_LOP[j] DO
```

```
        Writeln(Hoten:15,' ',Gioitinh:2,' ',Lop:4,' ',Diachi:10,' Toan:', Toan:4:2,'  
Ly:',Ly:4:2,' Hoa:',Hoa:4:2);
```



```
Repeat Until Keypressed;  
END.
```

Ví dụ 2.4 sử dụng mảng DS_LOP gồm 40 phần tử, mỗi phần tử là một Record. Mỗi lần nhập xong dữ liệu cho một phần tử lại hỏi "Nhập tiếp hay thôi?", như vậy số phần tử cụ thể của mảng tùy thuộc vào câu trả lời C hay là K.

Vấn đề đáng quan tâm ở đây là cách sử dụng lệnh With ... Do, ví dụ 2.4 sử dụng biến mảng DS_LOP vì vậy trước tiên phải truy nhập vào phần tử thứ i của mảng DS_LOP ($1 \leq i \leq 40$), với mỗi phần tử chúng ta tiếp tục truy nhập vào các trường của chúng.

Với lệnh

```
With DS_LOP[i] Do
```

chúng ta có thể nhập trực tiếp dữ liệu vào các trường Hoten, Gioitinh, ...

```
Write(' Ho va ten hoc sinh: '); Readln(Hoten);
```

```
Write(' Trai hay gai T/G: '); Readln(Gioitinh);
```

.....

Để viết dữ liệu ra màn hình chúng ta cũng dùng lệnh With ... Do

```
With DS_LOP[j] DO
```

```
Writeln(Hoten:15,' ',Gioitinh:2,' ',Lop:4,' ',Diachi:10,' Toan:', Toan:4:2,' Ly:',Ly:4:2,'  
Hoa:',Hoa:4:2);
```

Đối với các bản ghi lồng nhau như ví dụ 2.3 thì lệnh With.. Do cũng phải lồng nhau, xét ví dụ sau:

Ví dụ 2.5

```
Program Kieu_Record;
```

```
Uses crt;
```

```
Type Tencb=record
```

```
    Mahoso:word;
```

```
    Hoten:string[25];
```

```
    Ngaysinh: Record
```

```
        ngay:1..31;
```

```
        thang:1..12;
```

```
        nam:integer;
```

```
    End;
```

```
End;
```

```
Var
```

```
    DS: array[1..50] of tencb;
```

```
    i,j:byte; tl:char;
```

```
Begin
```

```
    Clrscr;
```

```
    i:=1;
```

```
    Repeat
```

```

    With ds[i] do
Begin
    Write('Ma ho so: '); Readln(mahoso);
    Write('Ho va ten: '); readln(hoten);
    With ngaysinh do
        Begin
            Write('Ngay sinh: '); readln(ngay);
            Write('Thang sinh: '); readln(thang);
            Write('Nam sinh: '); readln(nam);
        End;
    End;
End;
    Write('Nhap tiep hay thoi? C/K '); readln(tl);
    If upcase(tl)='C' then i:=i+1;
    Until upcase(tl)='K';
    clrscr;
    Writeln('DANH SACH CAN BO CO QUAN');
    For j:= 1 to i do
Begin
    With ds[j] do
        Begin
            Write(mahoso,' ',hoten,' ');
            With ngaysinh do Writeln(ngay,' ',thang,' ',nam);
        End;
    End;
End;
Readln;
END.

```

Trong ví dụ 2.5 để nhập dữ liệu vào bản ghi và viết dữ liệu (tức là danh sách cán bộ cơ quan) ra màn hình, chương trình cần phải sử dụng hai lệnh With .. Do lồng nhau, lệnh thứ nhất với biến DS, còn lệnh thứ hai với biến Ngaysinh. Tuy nhiên nếu có một chút tinh ý thì các khối chương trình nhập và viết dữ liệu ra màn hình có thể thu gọn lại, ví dụ khối viết dữ liệu ra màn hình sẽ như sau:

```

Writeln('DANH SACH CAN BO CO QUAN');
For j:= 1 to i do
    With ds[j] do
        Writeln(mahoso,' ',hoten,' ',ngaysinh.ngay,' ',ngaysinh.thang,' ',ngaysinh.nam);

```

1.5 Bản ghi có cấu trúc thay đổi

a. Xây dựng kiểu dữ liệu

Bản ghi có cấu trúc cố định dùng để mô tả một đối tượng mà các cá thể của nó (tức là các xuất hiện của đối tượng) có các thuộc tính (các trường) như nhau. Ví dụ DS là bản ghi mô tả một loại đối tượng là "**học viên**", mỗi xuất hiện của DS ứng với một học viên cụ thể và tất cả các học viên đều có các thuộc tính như nhau bao gồm (xem ví dụ 2.5)

Mahoso, Hoten, Ngay, Thang, Nam.

Trong thực tế nhiều khi ta gặp những đối tượng mà thuộc tính của chúng lại gồm hai loại:

- Thuộc tính chung cho mọi xuất hiện
- Thuộc tính riêng cho một số xuất hiện đặc biệt

Dưới đây là một số ví dụ minh họa:

- Kiểu dữ liệu quản lý vé ngành đường sắt

Trên một tuyến đường sắt có nhiều đoàn tàu chạy trong ngày, có những chuyến tốc hành chỉ dừng lại ở một vài ga dọc đường, có những chuyến tàu thường dừng lại tất cả các ga lẻ. Với tàu tốc hành, hành khách chỉ được mang theo hành lý không quá 20 Kg và sẽ có suất ăn trên tàu. Với tàu thường hành khách phải mua vé hàng hoá nếu có vận chuyển hàng hoá và không có suất ăn trên tàu.

* Thuộc tính chung: tên đoàn tàu (TDT), tuyến đường (TD), giờ đi (GD), loại tàu (LT) (ví dụ: tốc hành - TH, tàu thường - TT)

* Thuộc tính riêng với tàu tốc hành: Số suất ăn (SXA), số ga lẻ dừng dọc đường (SGD), còn tàu thường có thuộc tính riêng là cước hàng hoá (CHH). Như vậy việc xây dựng các bản ghi dữ liệu sẽ phải chú ý đến các thuộc tính chung cho các loại tàu và các thuộc tính riêng của từng loại (xem ví dụ 2.6).

Ví dụ 2.6:

Type QLDS = record

Ten_doan_tau: string[3];

Tuyen_duong: string[15];

Gio_di: real;

Case Loai_tau : (Toc_hanh, Tau_thuong) of

Toc_hanh: (So_xuat_an:Word; So_ga_do: Byte);

Tau_thuong: (cuoc_hang_hoa:real);

End;

Ví dụ 2.6 cho ta một kiểu dữ liệu bản ghi có cấu trúc thay đổi một mức, sự thay đổi ở đây thể hiện qua thuộc tính Loai_tau. Như vậy tổng số trường của mỗi bản ghi tùy thuộc vào đoàn tàu đó thuộc loại gì. Nếu là tàu tốc hành thì mỗi bản ghi có 5 trường, còn tàu thường chỉ có 4 trường. Độ dài của các bản ghi được tính căn cứ vào độ dài của các trường có trong bản ghi đó. Như đã biết độ dài từng trường được tính như sau:

Ten_doan_tau: 4,

Tuyen_duong: 16

Gio_di: 6

So_xuat_an: 2

So_ga_do: 1

Ve_hang_hoa: 6

Bản ghi với tàu tốc hành sẽ là $4+16+6+2+1 = 29$ Byte

Bản ghi với tàu thường là: $4+16+6+6 = 32$ Byte

- Các bước định nghĩa kiểu bản ghi có cấu trúc thay đổi:

- Để định nghĩa một kiểu bản ghi có cấu trúc thay đổi, chúng ta khai báo các thuộc tính chung trước, tiếp đó tìm trong các thuộc tính chung một thuộc tính dùng để phân loại.

- Thuộc tính phân loại có thể bao gồm một hoặc một số chỉ tiêu phân loại, tất cả các chỉ tiêu của thuộc tính phân loại phải đặt trong cặp dấu mở-đóng ngoặc đơn, ví dụ: Loai_tau : (Toc_hanh, Tau_thuong)

- Sử dụng toán tử Case ... of để phân loại, ví dụ:

Case Loai_tau : (Toc_hanh, Tau_thuong) of

Toc_hanh:

Tau_thuong:.....

- Với mỗi chỉ tiêu phân loại, chúng ta có thể khai báo tên một số trường thay đổi hoặc khai báo một bản ghi con với cấu trúc thay đổi, ví dụ:

Case Loai_tau : (Toc_hanh, Tau_thuong) of

Toc_hanh: (So_xuat_an, So_ga_do: Word);

Tau_thuong: (Case Cuoc :(cuoc_hanh_ly,cuoc_hang_hoa) of);

- Các trường thay đổi nếu có cùng kiểu dữ liệu thì tên trường viết cách nhau bởi dấu phẩy.

- Dữ liệu các trường phân loại phải thuộc kiểu đơn giản, cụ thể là:

kiểu nguyên, thực, logic, chuỗi, liệt kê, khoảng con.

Để phân loại chúng ta dùng toán tử Case ... Of.

Cần chú ý rằng toán tử Case .. Of ở đây không giống như cấu trúc Case .. Of đã nêu trong phần các cấu trúc lập trình nghĩa là cuối phần khai báo không có từ khoá "End;"

Trong vùng nhớ cấp phát cho chương trình sẽ có hai đoạn dành cho hai loại trường, đoạn thứ nhất dành cho các trường cố định, trong ví dụ 2.6 đoạn này có dung lượng là 26 byte. Đoạn thứ hai dành cho các trường thay đổi, đoạn này sẽ có dung lượng bằng dung lượng của chỉ tiêu phân loại lớn nhất.

Trong ví dụ 2.6 trường phân loại là Loai_tau, chỉ tiêu phân loại là toc_hanh và Tau_thuong. Với chỉ tiêu Toc_hanh, chúng ta khai báo hai trường thay đổi là So_xuat_an và So_ga_do còn với chỉ tiêu Tau_thuong có một trường là Cuoc_hang_hoa. Như vậy dung lượng của trường thay đổi của tàu tốc hành cần 3 byte còn tàu thường cần 6 byte, đoạn nhớ dành cho trường thay đổi sẽ có dung lượng 6 byte.

Chương trình quản lý đường sắt được thiết kế bao gồm một chương trình con lấy tên là NHAP dùng để nhập dữ liệu cho các đoàn tàu, phần thân chương trình chính sẽ yêu cầu nhập số chuyên tàu trên toàn tuyến và cho hiện dữ liệu ra màn hình (xem ví dụ 2.7).

Ví dụ 2.7

```
Program quan_ly_duong_sat;
Uses crt;
Type doan_tau = record
    ten_doan_tau:string[3];
    tuyen_duong:string[15];
    gio_di:real;
    loai:string[10];
Case loai_tau: (toc_hanh,tau_thuong) of
    toc_hanh:(so_xuat_an:word;so_ga_do:byte);
    tau_thuong:(cuoc_hang_hoa:real);
End;
    dt = array[1..5] of doan_tau;
Var
    dt1:dt;
    n,i,j:byte;tg:doan_tau;
Procedure Nhap(m:byte;var qlds:dt);
Begin
    For i:= 1 to m do
        with qlds[i] do
            Begin
                Write('Loai tau: ');readln(loai);
                if loai ='toc hanh' then
                    Begin
                        write('ten doan tau: '); readln(ten_doan_tau);
                        write('tuyen duong: '); readln(tuyen_duong);
                        write('gio xuất phát: '); readln(gio_di);
                        write('so xuất an: '); readln(so_xuat_an);
                        write('so ga do doc duong: '); readln(so_ga_do);
                        writeln;
                    end
                else
                    if loai = 'tau thuong' then
                        Begin
                            write('ten doan tau: '); readln(ten_doan_tau);
                            write('tuyen duong: '); readln(tuyen_duong);
                            write('gio xuất phát: '); readln(gio_di);
                            write('tien cuoc hang hoa: '); readln(cuoc_hang_hoa);
```

```

        writeln;
    End;
End; End;

Begin
    clrscr;
    write('co bao nhieu doan tau tren toan tuyen: '); readln(n);
    writeln;
    nhap(n,dt1);
    clrscr;
    writeln('danh sach tau chay toan tuyen');
    for i:= 1 to n-1 do      {sap xep du lieu tang dan theo loai tau}
    for j:= i+1 to n do
    with dt1[i] do
    if dt1[i].loai < dt1[i+1].loai then
    begin
        tg:=dt1[i]; dt1[i]:=dt1[j]; dt1[j]:=tg;
    end;
    Writeln(' DANH MUC CAC DOAN TAU TREN TUYEN');
    for i:= 1 to n do
    with dt1[i] do
    writeln(loai:10,' ',ten_doan_tau:3,' ',tuyen_duong:10,' ',gio_di:4:2,' ',so_xuat_an:3,'
',so_ga_do:3,cuoc_hang_hoa:10:2);
    readln;
END.

```

- *Kiểu dữ liệu quản lý điểm của sinh viên*

SV là kiểu dữ liệu bản ghi dùng để quản lý điểm của sinh viên. Các trường cố định của SV bao gồm: MHS (mã hồ sơ), HOTEN (họ và tên), NS (ngày sinh), GIOI (nam, nữ), Khoa (Sư phạm - SP, Kinh tế - KT, Cơ điện - CD). Các môn học tùy thuộc vào khoa mà sinh viên đang theo học, giả sử chúng ta quy định khoa Sư phạm có các môn: Toán, Lý, Tin cơ bản, lập trình nâng cao, khoa Kinh tế có các môn: Kế toán máy, Marketing, khoa Cơ điện có các môn: Cơ học máy, Sức bền vật liệu, Hình hoạ. Tất cả sinh viên nếu là Nam thì học thêm môn Bơi lội, nếu là Nữ thì học thêm Thể dục nghệ thuật.

Rõ ràng là chúng ta không thể tạo ra kiểu bản ghi cố định cho sinh viên trong toàn trường, bởi lẽ số môn học không giống nhau. Các trường MHS, HOTEN, NS là chung cho mọi sinh viên, trường KHOA và GIOI dùng để phân loại sinh viên từ đó xác định các môn học. Vì rằng mỗi kiểu bản ghi chỉ có thể khai báo *duy nhất một trường phân loại ngang hàng với các trường cố định* nên cùng một lúc chúng ta không thể phân loại theo cả KHOA và GIOI. Giải pháp duy nhất là chọn một trường phân loại với hai chỉ tiêu phân loại đại diện cho Khoa và Gioi, giả sử tên trường phân loại bây giờ lấy tên là MONHOC và hai chỉ tiêu phân loại là PL1 và PL2. PL1 đại diện cho Gioi còn PL2 đại diện cho Khoa.

Các chỉ tiêu phân loại lại có thể trở thành thuộc tính phân loại với một số chỉ tiêu nào đó mà chúng ta gọi là *chỉ tiêu con* chẳng hạn xem PL1 là thuộc tính phân loại với hai chỉ tiêu con là Nam và Nu, PL2 là thuộc tính phân loại với ba chỉ tiêu con là SP,KT,CD. Mỗi chỉ tiêu con bao gồm một số trường cụ thể hoặc nó lại được sử dụng như trường phân loại mới....

Một bản ghi kiểu SV có thể có cấu trúc thuộc một trong các dạng sau:

* Sinh viên khoa Sư phạm

1/ Mhs, Hoten, Ns, Boi_loi, Toan, Ly, Tincoban, Lap_trinh_nang_cao

2/ Mhs, Hoten, Ns, The_duc, Toan, Ly, Tincoban, Lap_trinh_nang_cao

* Sinh viên khoa Cơ điện

3/ Mhs, Hoten, Ns, Boi_loi, Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa

4 / Mhs, Hoten, Ns, The_duc, Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa

* Sinh viên khoa Kinh tế

5 / Mhs, Hoten, Ns, Boi_loi, Ke_toan_may, Marketing

6 / Mhs, Hoten, Ns, The_duc, Ke_toan_may, Marketing

Có thể nhận thấy rằng tên các trường phân loại không có trong cấu trúc bản ghi. Nếu chúng ta muốn trong mỗi bản ghi lại có cả tên khoa và giới tính thì phải đưa thêm vào các trường cố định mới.

Kiểu dữ liệu bản ghi SV được khai báo như sau:

Ví dụ 2.6

Type

SV = record

Mhs: Byte;

Hoten: String[20];

NS : Record

Ngay:1..31; Thang: 1..12; Nam: Word;

End;

Case monhoc:(pl1,pl2) of

pl1:(case gioi:(nam,nu) of

Nam: (Boi_loi:real);

Nu: (The_duc: real));

pl2:(case KHOA: (SP,CD,KT) of

SP: (Toan, Ly, Tincb, Ltnc: Real);

CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);

KT: (Ke_toan_may, Marketing:real));

End;

Từ cách khai báo trên chúng ta rút ra một số nhận xét quan trọng sau đây:

Nhận xét 1

Trong một kiểu record các trường cố định được khai báo trước, trường phân loại khai báo sau, như vậy trường phân loại phải là trường khai báo cuối cùng. Các trường thay đổi khai báo bên trong trường phân loại.

Nhận xét 2

Mỗi kiểu dữ liệu Record có cấu trúc thay đổi chỉ được phép có duy nhất một trường phân loại, nghĩa là không thể có hai toán tử case of ngang hàng khi khai báo. Nếu chúng ta khai báo kiểu SV như trong ví dụ 2.8 sau đây thì sẽ nhận được thông báo lỗi :

Error in Expression

Ví dụ 2.8

Type SV = record

Mhs: Byte;

Hoten: String[20]

NS : Record

 Ngày:1..31;

 Thang: 1..12;

 Nam: Word;

End;

 Case GIOI: (Nam, Nu) of

 Nam: Boi_loi:real;

 Nu:The_duc: real;

 Case KHOA: (SP,CD,KT) of

 SP: (Toan, Ly, Tin_cb, Ltnc: Real);

 CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);

 KT: (Ke_toan_may, Marketing:real);

End;

Lỗi xuất hiện do chúng ta đã chọn hai trường phân loại là GIOI và KHOA ngang hàng nhau. Để khắc phục lỗi này chúng ta phải lựa chọn lại cấu trúc của Record. Thay vì có hai trường phân loại cùng cấp chúng ta chọn một trường là MONHOC, chỉ tiêu phân loại là PL1 và PL2. Lúc này tên môn học cụ thể sẽ tùy thuộc vào giá trị mà PL1 và PL2 có thể nhận (xem ví dụ 2.9)

Nhận xét 3

Vì mỗi trường lại có thể là một bản ghi cho nên bên trong trường phân loại lại có thể chứa các trường phân loại khác, đây là trường hợp bản ghi thay đổi nhiều mức.

Với kiểu dữ liệu SV đã định nghĩa chúng ta xây dựng chương trình quản lý điểm của sinh viên như ví dụ 2.9 sau đây.

Ví dụ 2.9

Program QUAN_LY_DIEM;

Uses crt;

Type


```

SV = record
Mhs: Byte;
Hoten: String[20];
NS : Record
    Ngay:1..31;
    Thang: 1..12;
    Nam: Word;
End;
Case monhoc:(pl1,pl2) of
    pl1:( case gioi:(nam,nu) of
        Nam: (Boi_loi:real);
        Nu: (The_duc: real));
    pl2:( case KHOA: (SP,CD,KT) of
        SP: (Toan, Ly, Tincb, Ltnc: Real);
        CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);
        KT: (Ke_toan_may, Marketing:real));
    End;
Var
Ds:Array[1..100] of sv; tg:sv;
i,j,k:byte; pl,tk:string[3]; tl,gt:char;
BEGIN
clrscr; i:=1;
writeln(' Nhap diem cho sinh vien ');
Repeat
With ds[i] do
    Begin
        Write('Nhap ma ho so '); readln(mhs);
        Write('Nhap ho va ten '); readln(hoten);
        With ns do
            Begin
                Write('Nhap ngay sinh '); readln(ngay);
                Write('Nhap thang sinh '); readln(thang);
                Write('Nhap nam sinh '); readln(nam);
            End;
        Write('Cho biet gioi tinh nam "T" - nu "G" '); Readln(gt);
        if upcase(gt)='T' then ds[i].gioi:=nam else ds[i].gioi:=nu;
    Case ds[i].gioi of
        nam: begin
            Write('Nhap diem mon boi loi '); Readln(boi_loi);
            end;
        nu: begin

```

```

        Write('Nhap diem mon The duc '); readln(the_duc);
        end; End; {ket thuc Case ds[i].gioi...}
Write('Nhap ten khoa '); Readln(tk);
for k:= 1 to length(tk) do tk[k]:=upcase(tk[k]); { chuyển tên khoa thành chữ in}
if tk='SP' then ds[i].khoa:=sp
else if tk='CD' then ds[i].khoa:=cd
else ds[i].khoa:=kt;
Case ds[i].khoa of
    sp:Begin
        Write('Nhap diem mon Toan '); Readln(toan);
        Write('Nhap diem mon Ly '); Readln(ly);
        Write('Nhap diem mon Tin Co ban '); Readln(tincb);
        Write('Nhap diem mon Lap trinh nang cao '); Readln(ltnc);
    End;
    cd: Begin
        Write('Nhap diem mon Co hoc '); Readln(co_hoc_may);
        Write('Nhap diem mon Suc ben vat lieu '); Readln(suc_ben_vat_lieu);
        Write('Nhap diem mon Hinh hoa '); Readln(hinh_hoa);
    End;
    kt: Begin
        Write('Nhap diem mon Ke toan may '); Readln(ke_toan_may);
        Write('Nhap diem mon marketing '); Readln(marketing);
    End;
End; {ket thuc Case..}
                                { Sap xep du lieu tang dan theo ten Khoa}
for j:=1 to i-1 do
for k:=j+1 to i do
if ds[j].khoa<ds[j+1].khoa then
Begin
tg:=ds[j]; ds[j]:=ds[k]; ds[k]:=tg;
End;

End; {ket thuc with}
writeln;
Write('Nhap tiep hay thoi? C/K '); readln(tl);
if upcase(tl)='C' then i:=i+1;
Until upcase(tl)='K';

{Hien du lieu da nhap }
Write('Co xem du lieu khong? C/K '); readln(tl);

```

```

    if upcase(tl)='C' then
Begin
    Writeln('DU LIEU DA NHAP ');
    For j:= 1 to i do
        with ds[j] do
Begin
            write(mhs:3,' ',hoten:20);
            with ns do write(' ',ngay,'/',thang,'/',nam);
            case ds[j].khoa of
sp:writeln(' Khoa SP ',toan:4:1,ly:4:1,Tincb:4:1,ltnc:4:1);
cd:writeln(' Khoa CD ', co_hoc_may:4:1, suc_ben_vat_lieu:4:1,
hinh_hoa:4:1);
kt:writeln(' Khoa KT ',ke_toan_may:4:1,marketing:4:1);
            end;
        End;
    End;
    Readln;
END.

```

Ví dụ 2.9 tuy đã chạy hoàn chỉnh song có một số nhược điểm sau:

- * Tổ chức chương trình chưa hợp lý
- * Dữ liệu đưa ra màn hình chưa đẹp

Bạn đọc có thể thiết kế lại bằng cách đưa vào chương trình con Nhap và chương trình con Hien. Chương trình con Hien có thể thiết kế để dữ liệu đưa ra dưới dạng bảng theo mẫu sau đây:

DANH SACH SINH VIEN TRUONG

Ma ho so	Ho va Ten	Gioi	Khoa	So mon hoc	Tong diem	Trung binh
.....						

b. Truy nhập

Việc truy nhập vào các trường cố định của bản ghi có cấu trúc thay đổi hoàn toàn giống như bản ghi thường. Còn việc truy nhập và các trường thay đổi cần phải chú ý một số điểm sau:

- Không dùng phép gán hoặc nhập dữ liệu từ bàn phím cho các trường phân loại. Nếu trong ví dụ 2.9 chúng ta đưa vào lệnh

```
monhoc:='Toan';
```

thì sẽ nhận được thông báo lỗi: Type Mismatch

còn nếu đưa vào lệnh

Read(monhoc);

thì sẽ nhận được thông báo: Cannot Read or Write Variables of this Type

- Lệnh With ... Do có tác dụng với tất cả các trường kể cả các trường thay đổi bên trong các trường phân loại. Cụ thể trong ví dụ 2.9 với lệnh

With ds do chúng ta có thể đưa vào trực tiếp lệnh

Write('Nhap diem mon boi loi '); Readln(boi_loi);

- Tên các trường phân loại không thể đưa ra màn hình như là một tên trường bình thường nghĩa là cũng giống như trong mục 2.2 không thể viết lệnh Write(monhoc). Trong trường hợp cần thiết chúng ta có thể sử dụng các biến trung gian.

2. Dữ liệu kiểu tệp (file)

2.1 Khái niệm về tệp

Tệp dữ liệu là một dãy các phần tử cùng kiểu được sắp xếp một cách tuần tự. Tệp dữ liệu được cất giữ ở bộ nhớ ngoài (đĩa mềm hoặc đĩa cứng) dưới một tên nào đó, cách đặt tên tuân theo quy định của DOS nghĩa là phần tên tệp dài không quá 8 ký tự và phần đuôi không quá 3 ký tự.

Tệp tập hợp trong nó một số phần tử dữ liệu có cùng cấu trúc giống như mảng (Array) song khác mảng là số phần tử của tệp chưa được xác định.

Trong Pascal có 3 loại tệp được sử dụng là:

a. Tệp có kiểu

Tệp có kiểu là tệp mà các phần tử của nó có cùng độ dài và cùng kiểu dữ liệu. Với những tệp có kiểu chúng ta có thể *cùng một lúc đọc dữ liệu từ tệp ra hoặc nhập dữ liệu vào tệp.*

b. Tệp văn bản (Text)

Tệp văn bản dùng để lưu trữ dữ liệu dưới dạng các ký tự của bảng mã ASCII, các ký tự này được lưu thành từng dòng, độ dài của các dòng có thể khác nhau. Khi ghi một số nguyên, ví dụ số 2003 (kiểu word) vào tệp văn bản, Pascal cần 4 byte cho bốn ký tự chứ không phải là 2 byte cho một số. Việc ghi các số nguyên hoặc thực vào tệp văn bản sẽ phải qua một công đoạn chuyển đổi, điều này sẽ do Pascal tự động thực hiện. Để phân biệt các dòng Pascal dùng hai ký tự điều khiển là CR - về đầu dòng và LF - xuống dòng mới.

Trong bảng mã ASCII ký tự CR = CHR(13) còn LF = CHR(10)

c. Tệp không kiểu

Tệp không kiểu là một loại tệp không cần quan tâm đến kiểu dữ liệu ghi trên tệp. Dữ liệu ghi vào tệp không cần chuyển đổi.

Khi khai báo một biến kiểu tệp thì đồng thời ta cũng phải tạo ra một mối liên hệ giữa biến này và một tệp dữ liệu lưu trên thiết bị nhớ ngoài. Cách khai báo này đã được chuẩn hoá trong Pascal.

Kiểu dữ liệu của các phần tử trong biến kiểu tệp có thể là bất kỳ kiểu nào trừ kiểu của chính nó tức là trừ kiểu tệp. Ví dụ nếu kiểu phần tử là một mảng một chiều của các số nguyên ta có tệp các số nguyên, nếu kiểu phần tử là Record ta có tệp các Record...

Tác dụng lớn nhất của kiểu dữ liệu tệp là ta có thể lưu trữ các dữ liệu nhập vào từ bàn phím cùng các kết quả xử lý trong bộ nhớ RAM ra tệp để dùng nhiều lần. Các kiểu dữ liệu đã học chỉ xử lý trong RAM và in kết quả ra màn hình hoặc ra máy in, khi kết thúc chương trình hoặc mất điện cả dữ liệu nhập vào và kết quả xử lý đều bị mất.

d. Khai báo

Tệp có kiểu được định nghĩa sau từ khoá TYPE còn biến kiểu tệp được khai báo sau từ khoá VAR. Thủ tục khai báo biến kiểu tệp gồm 2 cách:

- Định nghĩa kiểu tệp với từ khoá FILE OF trong phần mô tả kiểu sau từ TYPE, tiếp theo là khai báo biến tệp trong phần khai báo biến

Ví dụ 2.10

Type

MSN = Array[1..100] of integer; (*định nghĩa mảng 100 số nguyên*)

TSN = File of MSN; (* định nghĩa dữ liệu kiểu tệp TSN có các phần tử là mảng số nguyên *)

CHUVIET = File of String[80]; (* định nghĩa CHUVIET là tệp các chuỗi có độ dài 80 ký tự *)

BANGDIEM = Record

Hoten: String[25];

Gioitinh: Char;

Lop: String[5];

Diachi: String[50];

Toan,Ly,Hoa: Real;

END;

TBD = File of BANGDIEM;

VAR

Tep1: TSN; (* biến tep1 có các phần tử là mảng số nguyên *)

Tep2: CHUVIET; (* biến Tep2 có các phần tử là chuỗi ký tự *)

Tep3: TBD; (* biến Tep3 có các phần tử là record*)

.....

- Định nghĩa trực tiếp biến kiểu tệp trong phần khai báo biến

Ví dụ 2.11

Var

Tep4: File of Array[1..5] of String[80];

Tep5: File of BANGDIEM;

Trong phần khai báo này biến Tep4 là một biến kiểu tệp, tệp này có 5 phần tử, mỗi phần tử là một chuỗi dài tối đa 80 ký tự. Biến Tep5 là một biến tệp mà các phần tử của nó có kiểu Bangdiem.

e. Truy nhập vào tệp

Các phần tử của tệp được lưu giữ tuần tự thành một dãy và việc truy nhập vào từng phần tử phụ thuộc vào thiết bị ghi, đọc của máy vi tính. Turbo Pascal có thể xử lý hai loại tệp là: **Tệp truy nhập tuần tự** và **tệp truy nhập trực tiếp**.

* **Tệp truy nhập tuần tự**: để truy nhập vào một phần tử nào đó ta bắt buộc phải đi qua các phần tử trước đó. Nếu muốn thêm các phần tử vào tệp thì chỉ có thể thêm vào cuối tệp. Tệp kiểu này dễ hình dung, dễ xử dụng song không linh hoạt, tốn thời gian xử lý. Việc truy nhập tuần tự thường được thực hiện thông qua một vòng lặp. *Tệp văn bản là tệp thuộc kiểu này.*

* **Tệp truy nhập trực tiếp**: là tệp có thể truy nhập vào phần tử bất kỳ trong tệp, trên những thiết bị nhớ ngoài cố điển như băng từ, băng đục lỗ không thể tạo tệp kiểu này vì không thể đọc ngay vào giữa băng. Chỉ những máy sử dụng đĩa (mềm hoặc cứng) thì mới có thể tạo tệp truy nhập trực tiếp vì có thể điều chỉnh để đầu từ đặt đúng vào một cung từ chứa dữ liệu nào đó.

Muốn truy nhập trực tiếp phải dùng thủ tục Seek(số hiệu phần tử).

f. Mở tệp

Để mở một tệp chuẩn bị lưu trữ dữ liệu Pascal xử dụng hai thủ tục chuẩn sau đây:

ASSIGN(biến tệp, tên tệp); (*liên kết biến tệp với một tên tệp sẽ ghi vào thiết bị nhớ ngoài *)

REWRITE(biến tệp); (* tạo một biến tệp rỗng chuẩn bị nhập dữ liệu vào *)

Trong đó:

* **Biến tệp**: là tên biến tệp đã khai báo sau từ khoá VAR

* **Tên tệp**: Là tên do ta chọn để ghi dữ liệu vào đĩa (theo quy định của DOS). Tên tệp có thể bao gồm cả đường dẫn tới thư mục mà chúng ta lựa chọn. Đường dẫn và tên tệp phải đặt trong dấu nháy đơn. Ví dụ:

ASSIGN(f,'a:\baitap.txt');

Sau thủ tục REWRITE ta sẽ có một tệp rỗng vì chưa có phần tử nào được đọc vào. Nếu trên thiết bị nhớ ngoài ta đã có sẵn một tệp trùng tên với tên ghi ở thủ tục ASSIGN thì tệp ngoài sẽ bị xoá đi.

Sau hai thủ tục chuẩn bị trên đây, để tiến hành ghi dữ liệu vào tệp ta lại dùng thủ tục WRITE(...) như đã biết.

Cách viết:

WRITE(biến tệp, các giá trị cần ghi vào tệp);

Bước cuối cùng là phải đóng tệp lại bằng thủ tục

CLOSE(biến tệp);

Sau thủ tục này dữ liệu sẽ được lưu trên đĩa và tệp sẽ bị đóng.

2.2 Tệp văn bản

a. Khai báo tệp văn bản

Tệp văn bản được khai báo trực tiếp trong phần khai báo biến:

Var

Bien_tep : Text;

Ví dụ

Var

Hoso: Text;

T1,T2,T3:Text;

b. Truy nhập vào tệp

Truy nhập vào tệp được hiểu là nhập dữ liệu vào tệp, ghi lại dữ liệu trên thiết bị nhớ ngoài, đọc dữ liệu đã có ra màn hình hoặc máy in và xử lý dữ liệu đó.

Đối với tệp văn bản việc ghi dữ liệu vào tệp có thể thực hiện qua hai cặp thủ tục sau:

- Mở tệp mới để ghi

Assign(bien_tep, Đường dẫn\ten_tep);

Rewrite(bien_tep);

Cặp thủ tục trên dùng để mở một tệp mới chuẩn bị nhận dữ liệu, nếu trong bộ nhớ ngoài đã có tệp trùng tên thì tệp ở ngoài sẽ bị xoá. Nếu bỏ qua *đường dẫn* thì tệp sẽ được lưu vào thư mục hiện hành tức là thư mục mà từ đó Pascal đã được khởi động (C:\TP\BIN). Giả sử chúng ta muốn lưu tệp với tên là BT1.DAT vào thư mục BAITAP trên đĩa A thì phải viết lệnh:

Assign(bien_tep, 'A:\BAITAP\BT1.DAT');

- Mở tệp đã có để ghi thêm

Assign(bien_tep, ten_tep);

Append(Bien_tep);

Cặp thủ tục trên mở một tệp đã lưu trong thiết bị nhớ ngoài để nhập thêm dữ liệu, nếu tệp không có thì máy sẽ thông báo lỗi. Dữ liệu nhập thêm sẽ được ghi vào cuối tệp.

- Mở tệp để đọc dữ liệu

Dưới đây là cặp thủ tục dùng để mở tệp đã tồn tại, chuẩn bị cho việc đọc dữ liệu từ tệp ra thiết bị ngoài:

Assign(bien_tep, ten_tep);

Reset(bien_tep);

c. Ghi dữ liệu vào tệp

Sau khi đã mở tệp chúng ta có thể dùng thủ tục Write hoặc Writeln để ghi dữ liệu vào tệp.

Cú pháp:

```
Write(Bien_tep, giá trị 1, giá trị 2, ...);
```

Hoặc

```
Writeln(Bien_tep, giá trị 1, giá trị 2, ...);
```

Sự khác nhau giữa Write và Writeln là ở chỗ thủ tục Write sẽ ghi dữ liệu liên tục trên các ô nhớ, không có các ký hiệu về đầu dòng và xuống dòng. Thủ tục Writeln sẽ đưa thêm vào cuối dòng các ký tự điều khiển CR (về đầu dòng) và LF (nhảy xuống dòng dưới).

Giá trị 1, giá trị 2... trong lệnh ghi có thể là chuỗi ký tự viết thường minh, tên biến hoặc biểu thức, các giá trị phải được phân cách ít nhất bằng một khoảng trống.

Thủ tục Writeln(Bien_tep); sẽ tạo nên một dòng rỗng không chứa ký tự nào cả.

Dữ liệu trước khi ghi vào tệp văn bản có thể thuộc các kiểu đơn giản sau đây: **Số nguyên, số thực, ký tự, chuỗi, logic**. Dù thuộc kiểu gì thì khi đã ghi vào tệp đều được chuyển đổi thành kiểu ký tự. Điều này cũng đồng nghĩa với quy định rằng các kiểu dữ liệu có cấu trúc như: **Array, Set, Record, File** không thể lưu trữ trên tệp văn bản.

Khi có thủ tục Close(bien_tep) Pascal sẽ đưa thêm vào dòng cuối cùng ký hiệu kết thúc tệp EOF (End of File).

Ví dụ 2.12

```
Var  
T1:text;  
Begin  
Assign(T1,'DULIEU.DAT');  
Rewrite(T1);  
Writeln(t1,'Tep van ban');  
Write(T1,123);  
Write(T1,' ',123.45);  
Writeln(T1);  
Close(T1);  
End.
```

Ví dụ 1.13 tạo ra tệp văn bản DULIEU.DAT tệp này sẽ được lưu trữ tại thư mục hiện hành tức là C:\TP\BIN.

Lệnh Writeln(t1,'Tep van ban'); sẽ ghi vào biến tệp T1 (cũng có nghĩ là ghi vào tệp DULIEU.DAT) dòng chữ "Tep van ban" tiếp đó là các ký tự điều khiển CR, LF.

Lệnh Write(T1,123); sẽ tự động chuyển đổi số 123 thành ký tự và ghi vào tệp không kèm theo CR, LF.

Lệnh Write(T1,' ',123.45); ghi vào tệp một khoảng trống trước dãy 123.45 tiếp đó chuyển đổi và ghi số 123.45 vào tệp theo quy cách ngầm định của Pascal.

Dữ liệu sẽ được ghi vào tệp như sau

```
Tep van ban  
123 1.234500000E+02  
(dòng trống)
```


Từ ví dụ trên có thể nêu một số nhận xét sau:

* Các ký tự nhập vào tệp phải nằm trong cặp dấu nháy đơn

* Các số không cần để trong dấu nháy, Pascal sẽ tự động chuyển đổi chúng khi ghi vào tệp. Nếu chúng ta không chỉ rõ quy cách đọc hoặc ghi số thì các số sẽ được ghi theo quy cách chuẩn, cụ thể là:

- Số nguyên sẽ được ghi chính xác như giá trị của nó trong lệnh Write(...)

- Số thực sẽ được chuyển thành dạng khoa học nghĩa là dạng số với lũy thừa cơ số 10, ví dụ 123.45 chuyển thành 1.234500000E+02. Trong cách viết khoa học Pascal ngầm định như sau:

- Phần nguyên là 1 ký tự

- Phần lẻ là 10 ký tự kể cả dấu chấm thập phân

- Phần còn lại là 4 ký tự cho lũy thừa cơ số 10, Ví Dụ:

$$E+02 = 10^2, \quad E-03 = 10^{-3}$$

* Các số có thể chỉ rõ quy cách ghi như sau

```
Write(T1,123:5);
```

Ghi vào tệp số nguyên 123 với độ dài 5 ký tự, Pascal sẽ để trống hai vị trí bên trái

```
Write(T1,' ',123.45:6:2);
```

Số 123.45 được ghi vào tệp với độ dài 6 ký tự và 2 ký tự chứa số lẻ

d. Đọc dữ liệu từ tệp văn bản

Dữ liệu lưu trữ trong tệp văn bản có thể cho hiện lên màn hình bằng lệnh TYPE của DOS, bằng các phím chức năng F3, F4 của NC hoặc đưa vào Word để xem và sửa chữa nếu cần.

Với Pascal tệp văn bản thuộc loại tệp truy nhập tuần tự, do vậy nếu cố tình dùng lệnh truy nhập ngẫu nhiên Seek(...) thì máy sẽ báo lỗi:

Error 63: Invalid File Type

Sau khi tiến hành mở tệp con trở tệp sẽ được đặt tại dòng đầu, Cách thức mà Pascal dùng để đọc dữ liệu là như sau: dùng thủ tục Read, hoặc Readln để đọc dữ liệu từ dòng hiện thời và gán vào biến tương ứng, viết biến đó ra màn hình hoặc máy in.

```
Read(Bien_tep, Dong); hoặc Readln(Bien_tep, Dong);
```

```
Write(Dong); hoặc Writeln(Dong);
```

Trong hai ví dụ trên Biến trung gian là Dong, biến này phải được khai báo trước và phải thuộc kiểu String.

Để có thể viết toàn bộ dữ liệu từ một tệp văn bản ra các thiết bị ngoài thì các lệnh đọc viết trên phải được lặp đi lặp lại từ dòng 1 đến dòng cuối cùng nghĩa là chương trình phải sử dụng một trong hai vòng lặp:

```
While not eof(Bien_tep) Do
```

```
Begin
```

```
Readln(Bien_tep, Dong);
```

```
Writeln(Dong);
```

```

End;
Hoặc
For i := 1 to Filesize(Bien_tep) Do
  Begin
    Readln(Bien_tep, Dong);
    Writeln(dong);
  End;

```

Nếu trên một dòng chúng ta đã ghi nhiều giá trị ứng với các biến khác nhau thì không thể đọc riêng từng biến. Cách mà chúng ta ra lệnh cho Pascal ghi dữ liệu vào tệp sẽ ảnh hưởng rất nhiều đến việc chúng ta gọi dữ liệu ra và xử lý dữ liệu đó.

Ví dụ sau đây sẽ giải thích cụ thể điều này.

Ví dụ 2.14

```

Var
  F:text;
  Hoten:string[25]; heso:real; socon:byte; t:string;
Begin
  Hoten:=' Tran Van Tam'; Heso:=4.25; Socon:= 2; {gán dữ liệu cho các trường}
  Assign(f,'hoso.txt');
  Rewrite(f);
  Writeln(f,hoten,' ',heso:4:2,' ',socon);
  Close(f);
  Reset(f);
  Readln(f,t); {đọc dữ liệu từ dòng hiện thời và gán vào biến t }
  Writeln(t);
Readln;
End.

```

Chương trình trên đây có nhiều điều cần phải lưu ý:

- Lệnh ghi vào tệp:

Writeln(f,hoten,' ',heso:4:2,' ',socon); sẽ ghi tất cả dữ liệu trên một dòng.

	T	r	a	n		V	a	n		T	a	m		4	.	2	5		2	CR	LF	EOF
--	---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	--	---	----	----	-----

Nhìn vào hình vẽ dễ dàng nhận thấy rằng chuỗi Hoten chiếm 13 Byte, Heso chiếm 4 Byte và Socon chiếm 1 Byte. Trong thực tế các số kiểu Real có độ dài là 6 Bytes nhưng vì chúng ta đã yêu cầu Pascal ghi Heso dài 4 ký tự trong đó có hai số lẻ nên khi ghi vào tệp, Heso chỉ chiếm 4 Bytes. Giữa các biến có hai khoảng trống do đó tổng chiều dài của dòng là 20 Bytes.

Lệnh đọc dữ liệu từ tệp Readln(f,t) sẽ đọc cả dòng hiện thời vào biến t và lệnh viết Writeln(t) sẽ đưa ra màn hình cả dòng nghĩa là ta lại nhận được toàn bộ dòng như trên. Cần

phân biệt rằng dữ liệu viết ra là các ký tự của bảng mã chứ không phải các chữ và số như ta đã nhập vào.

Trường hợp biến t không đủ độ dài để chứa hết cả dòng thì số ký tự thừa sẽ bị cắt bỏ.

- Nếu chúng ta viết lệnh đọc:

```
Readln(f,Hoten,' ',Heso,' ',socon);
```

thì máy sẽ báo lỗi: Error 106 : Invalid Numeric Format

nghĩa là không thể đọc trực tiếp cùng một lúc từ tệp ra màn hình cả biến kiểu ký tự và biến kiểu số.

- Nếu chúng ta viết lại lệnh ghi dữ liệu:

```
Writeln(f,hoten);
```

```
Writeln(f,heso:5:2,' ',socon);
```

nghĩa là ghi riêng dữ liệu kiểu "chữ" trên một dòng còn dữ liệu kiểu "số" trên dòng khác, giữa các số có một khoảng cách, sau đó dùng lệnh đọc:

```
Readln(f,hoten);
```

```
Readln(f,heso, Socon);
```

thì lại nhận được thông báo lỗi ở dòng `Readln(f,heso, Socon);`

```
Error 106 : Invalid Numeric Format
```

nghĩa là các số ghi vào tệp trên cùng một dòng thì cũng không thể đọc chúng như là các biến.

- Chúng ta sửa lại lệnh ghi một lần nữa

```
Writeln(f,hoten);
```

```
Writeln(f,heso:5:2);
```

```
Writeln(f,socon);
```

Rồi dùng lệnh đọc

```
Readln(f,Hoten);
```

```
Readln(f,Heso);
```

```
Readln(f,socon);
```

sau đó là lệnh viết

```
Writeln(Hoten, ' ', Heso:5:2,' ',socon,' ',Heso*290000);
```

thì chương trình sẽ không báo lỗi .

Đến đây có thể rút ra kết luận là:

- Muốn lấy lại kiểu của dữ liệu nhập vào tệp văn bản thì mỗi biến phải nhập trên một dòng.

- Với các biến kiểu số đã ghi riêng rẽ trên một dòng khi gọi ra Pascal sẽ tự động chuyển đổi từ dạng ký tự thành dạng số và ta có thể đưa các số này vào các biểu thức tính toán bình thường. Trong ví dụ trên chúng ta tính Lương bằng cách lấy `Heso*290000`.

- Trong bộ nhớ của máy dữ liệu được ghi liên tục trong các ô nhớ, để phân biệt các dòng Pascal dùng cặp ký tự điều khiển CR và LF. Nói cách khác dữ liệu được lưu trữ liên tục chứ không phải dưới dạng bảng, khi lấy dữ liệu ra chúng ta cũng lấy liên tục nhưng lại có thể bố trí trên màn hình sao cho trực quan và dễ theo dõi.

Ví dụ 2.15: xây dựng một chương trình đơn giản để quản lý công chức. Dữ liệu nhập vào bao gồm Họ tên, Hệ số lương và Số con. Dữ liệu xuất ra màn hình bao gồm Họ tên, Hệ số lương, Số con và Lương tháng, Lương tháng ở đây tính theo quy định của nhà nước = heso*290000.

Chương trình đặt ra hai khả năng lựa chọn:

- a. Nếu tệp dữ liệu đã tồn tại thì nhập thêm người
- b. Nếu tệp chưa có thì mở tệp mới

Trong cả hai trường hợp đều yêu cầu cho biết số người cần nhập.

Dữ liệu in ra dưới dạng bảng.

Ví dụ 2.15

Program Quan_ly_can_bo;

Uses crt;

Var f:text; hoten:string[20]; c1,heso:real; c2,i,n,socon:byte;

ten:string[12];

Begin

clrscr;

Write('Cho biet ten tep '); readln(ten);

{ \$I- }

assign(f,ten);

reset(f);

{ \$I+ }

if Ioresult=0 then

Append(f)

else

rewrite(f);

write('Nhap bao nhieu nguoi '); readln(n);

for i:= 1 to n do

Begin

Write(' Ho ten '); Readln(hoten);

Write(' He so '); readln(heso);

Write(' So con '); Readln(socon);

Writeln(f,hoten);

Writeln(f,heso:4:2);

writeln(f,socon);

End;

close(f);

```

assign(f,ten);
reset(f);
writeln('-----:-----:-----:-----:');
writeln(' | Ho va ten      | Hs | socon| Luong | ');
writeln('-----:-----:-----:-----:');
while not eof(f) do
  begin
    readln(f,hoten);
    readln(f,heso);
    readln(f,socon);
writeln(' | hoten:19, | heso:4:2, | ',socon:4, ' | ',heso*290000:10:2, ' | ');
    end;
    readln;
  End.

```

Ví dụ 2.15 có sử dụng định hướng chương trình dịch {\$I+}, {\$I-}. Khi định hướng là {\$I+} thì chương trình sẽ kiểm tra lỗi vào ra IO (Input, Output) nếu phát hiện thấy lỗi thì dừng chương trình, đây là chế độ ngầm định của Pascal. Nếu định hướng là {\$I-} thì việc kiểm tra lỗi vào ra tạm thời không thực hiện, nghĩa là nếu phát hiện thấy lỗi thì tạm treo các thủ tục vào ra và tìm xem hàm IORESULT cho kết quả là gì. Nếu hàm này cho kết quả bằng 0 thì có nghĩa là việc kiểm tra IO không có gì sai sót và chương trình tiếp tục làm việc. Nếu hàm Ioresult cho kết quả khác 0 thì có nghĩa là việc kiểm tra IO phát hiện thấy lỗi và chương trình cần phải được sửa chữa.

Ví dụ 2.16: Tạo tệp văn bản Baitho.txt để lưu trữ một bài thơ có n dòng, dòng cuối cùng ghi " Nam 2003". Khi nhập bài thơ cần hỏi trước bài thơ có bao nhiêu dòng.

Ví dụ 2.16

```

Program tep_van_ban;
Uses crt;
var i,n:integer; f:text; t:string;
Begin
  clrscr;
  assign(f,'baitho.txt');rewrite(f);
  writeln('Bai tho gom bao nhieu cau? '); readln(n);
  writeln('Hay nhap bai tho cua ban');
  for i:= 1 to n do
  begin
    write('Cau ',i, ' '); readln(t); writeln(f,t);

```

```

end;
writeln(f,'Nam ', 2003);
close(f);
clrscr;
gotoxy(15,5); i:=6;
clrscr;
writeln('Du lieu viet tu tep baitho.txt ');
reset(f);
while not eof(f) do
begin
readln(f,t); gotoxy(15,i);
writeln(t);
i:=i+1;
end;
readln;
End.

```

Ví dụ 2.16 có thể cải tiến theo nhiều cách để có được một chương trình đẹp dùng cho việc lưu trữ văn bản, chẳng hạn chúng ta sẽ đưa vào các chương trình con GHIMOI, GHITHEM và DOC phục vụ việc ghi dữ liệu vào tệp mới, ghi thêm dữ liệu vào tệp đã có hoặc đọc từ tệp ra. Trước khi ghi hoặc đọc cần kiểm tra xem tệp đã tồn tại chưa, khi nhập văn bản vào tệp không hạn chế số dòng, muốn kết thúc việc nhập thì khi bắt đầu một dòng mới chỉ cần bấm dấu "*" v.v... (xem ví dụ 2.17)

Ví dụ 2.17:

```

Program tep_van_ban;
Uses crt;
var i,j,n:integer; f:text; t:string;
    tl,tl1,tl2:char; ten:string[12];

Procedure Ghimoi(ten:string);
Begin
clrscr;
assign(f,ten);
rewrite(f);
writeln('Hay nhap bai tho cua ban');
i:=1;
repeat
write('Cau ',i,' '); readln(t);
if t<>'*' then writeln(f,t);

```

```

i:=i+1;
Until t='*';
close(f);
End;

```

```

Procedure Ghithem (ten:string);
Begin
clrscr;
assign(f,ten);
append(f);
writeln('Hay bo xung bai tho cua ban');
i:=1;
repeat
write('Cau ',i,' '); readln(t);
if t<>'*' then writeln(f,t);
i:=i+1;
Until t='*';
close(f);
end;

```

```

Procedure Doc(tep:string);
Begin
clrscr;
gotoxy(15,5); i:=6;
writeln('Du lieu viet tu tep ',tep);
assign(f,tep);
reset(f);
while not eof(f) do
begin
readln(f,t); gotoxy(15,i);
writeln(t);
i:=i+1;
end;
readln;
end;
BEGIN { Thân chương trình mẹ }
Clrscr;
Write('cho biet ten tep '); Readln(ten);
Write('Ban Ghi hay Doc du lieu G/D '); Readln(tl1);
If upcase(tl1)='G' then

```

```

Begin
write('Ghi tep moi hay ghi them vao tep cu M/C ');
Readln(tl2);
If upcase(tl2)='M' then Ghimoi(ten)
else
Ghithem(ten);
End
Else
Doc(ten);
END.

```

Trong ví dụ 2.17 có sử dụng một biến toàn cục là TEN, kiểu biến là string[12]. Khi lấy Ten làm tham số thực để truyền cho các chương trình con thì cần chú ý định nghĩa kiểu dữ liệu trước. Cụ thể nếu chúng ta viết dòng lệnh:

```
Procedure Doc(tep:string);
```

Dưới dạng mới là

```
Procedure Doc(tep:string[12]);
```

thì máy sẽ báo lỗi.

2.3 Tập có kiểu

Tập có kiểu là tập mà mọi phần tử đều có cùng độ dài và cùng kiểu. Kiểu các phần tử của tập có thể là số nguyên, thực, ký tự, chuỗi, mảng hoặc bản ghi. Cách thức khai báo biến kiểu tập đã trình bày trong mục II. Sự khác nhau cơ bản giữa tập có kiểu và tập văn bản là tập có kiểu ***có thể vừa ghi vào vừa đọc ra***, còn với tập văn bản chúng ta buộc phải kết thúc ghi bằng lệnh Close(bien_tep) thì mới có thể đọc tập, còn khi đang đọc tập chúng ta cũng phải kết thúc đọc và đóng tập thì mới có thể ghi thêm dữ liệu vào tập.

a. Đọc và ghi

- Ghi lên tập

```
Write(bientep, bien1, bien2, ...)
```

bien1, bien2, ... là các biến cùng kiểu với biến tập.

- Đọc tập

```
Read(Bientep, bien1, bien2, ....)
```

Đọc tập thực chất là đọc các phần tử của tập và gán cho các bien1, bien2,... cùng kiểu. Việc đọc diễn ra trong bộ nhớ do đó người sử dụng muốn biết được các giá trị cụ thể thì phải viết các biến đã đọc lên màn hình.

Chú ý:

Khác với tập văn bản, việc ghi và đọc tập có kiểu không sử dụng các lệnh Writeln hoặc Readln nghĩa là tập có kiểu không ghi dữ liệu thành các dòng. Các phần tử của tập có kiểu được ghi liên tục trong các ô nhớ và chỉ có ký hiệu kết thúc tập EOF.

Khi chúng ta đọc hoặc ghi xong một phần tử thì con trỏ tập sẽ tự động chuyển đến vị trí kế tiếp.

b. Truy nhập vào tệp

Seek(bientep,i); i=0,1,2,.....

Thủ tục Seek sẽ định vị con trỏ tại phần tử thứ i của tệp, Pascal quy định vị trí các phần tử là 0, 1, 2... như vậy phần tử thứ nhất là phần tử có vị trí 0.

Lệnh Seek(bientep, 5); sẽ định vị con trỏ tệp tại phần tử thứ 6.

Các tệp sau khi mở theo ngầm định con trỏ tệp luôn định vị tại phần tử đầu tiên nghĩa là phần tử có số thứ tự là 0. Trong quá trình đọc nếu chúng ta dùng một cấu trúc lặp để đọc thì khi đọc xong một phần tử con trỏ tệp sẽ tự động chuyển đến phần tử kế tiếp.

```
while not eof(BT) do {Kiểm tra xem con trỏ tệp đã nằm ở phần tử cuối chưa}
begin
  Read(bt,i); write(i:5);
end;
```

c. Các hàm xử lý tệp

*. **Filesize(bientep)** cho biết số phần tử có trong tệp

*. **FilePos(bientep)** cho biết vị trí hiện thời của con trỏ tệp

*. **Eof(Bientep)** cho giá trị True nếu con trỏ tệp ở vị trí cuối tệp (vị trí cuối tệp được hiểu là vị trí sau phần tử cuối cùng), nếu con trỏ tệp định vị tại bất kỳ phần tử nào của tệp thì hàm eof() cũng cho giá trị False.

Giả sử tệp DS có 8 phần tử khi đó hàm

Filesize(DS) sẽ cho kết quả là số 8,
còn thủ tục

Seek(DS,Filesize(DS))

sẽ định vị con trỏ tại phần tử thứ 8 (số thứ tự của phần tử này là 7) nghĩa là cuối tệp (bởi vì phần tử đầu tiên có số thứ tự là 0).

Ví dụ 2.18

Tạo một tệp lấy tên là TEPCK.DAT để vừa ghi vừa sửa dữ liệu

Program Tep_co_kieu;

Uses crt;

Var bt: file of byte; i:byte; n:real;

Begin

clrscr;

assign(bt,'TEPCK.DAT');

rewrite(bt);

for i:= 0 to 5 do write(bt,i); {ghi vào tệp năm số nguyên}

reset(bt);

writeln(' Du lieu luu tru trong tep TEPCK.DAT ');

while not eof(BT) do {viết dữ liệu từ tệp ra màn hình}

begin

Read(bt,i); write(i:5);

```

end;
writeln;
seek(bt,3); {định vị con trỏ tại phần tử thứ 4}
textcolor(magenta);
read(bt,i); {đọc phần tử thứ 4 vào biến i}
writeln('So cu trong tep o vi tri 3: ',i);
i:=33;
seek(bt,3);
write(bt,i); {sửa phần tử thứ 4, giá trị mới là 33}
seek(bt,3); read(bt,i)
Writeln('So moi trong tep o vi tri 3: ', i);
writeln('Vi tri hien thoi cua con tro: ',filepos(bt));
readln;
close(bt);
End.

```

Nhận xét:

Ví dụ 2.18 cho thấy khi ghi hoặc đọc dữ liệu vào tệp có kiểu thì chúng ta luôn luôn phải dùng đến các biến có cùng kiểu với biến tệp.

Đoạn chương trình

```
i:=33; seek(bt,3); write(bt,i);
```

dùng để ghi vào phần tử thứ 4 giá trị mới bằng 33. Nếu chúng ta viết lại đoạn lệnh này với ý tưởng ghi trực tiếp giá trị 33 vào phần tử thứ 4

```
seek(bt,3); write(bt,33);
```

thì Pascal sẽ báo lỗi "Variable Identifier Expertied" nghĩa là Pascal không hiểu số 33 thuộc kiểu dữ liệu gì.

Ví dụ 2.19

Lập chương trình để nhập điểm cho học sinh và ghi kết quả vào tệp có tên là DIEM.DAT. Chương trình định nghĩa một kiểu dữ liệu mới là Bangdiem (Kiểu Record) với các trường: Stt, Hoten, Diachi, Gioitinh, Lop, Toan, Ly, Hoa. Biến tệp Ds thuộc loại tệp có kiểu. Ví dụ 2.19 chỉ mới làm công việc là nhập dữ liệu vào tệp mà chưa đọc dữ liệu ra.

```

Program Kieutep;
Uses Crt;
Type
Bangdiem = Record
  Stt: Integer;
  Hoten, Diachi: String[25];
  Gioitinh, Lop: string[5];
  Toan,Ly,Hoa: Real;

```

```

End;
Tepdiem = File of Bangdiem;
Var
DS: Tepdiem;
NGUOI: Bangdiem;
i,j: Integer; lam,TL: Char;

Begin
Assign(DS,'diem.dat');
Rewrite(DS);
Nguoi.stt:=1;
textbackground(white);
textcolor(5);
lam:='L'; TL:='C'; While lam='L' do
Begin
clrscr;
i:=10; j:=5;
gotoxy(i,j); write(' So thu tu: ');
gotoxy(i,j+1); write(' Ho va ten: ');
gotoxy(i,j+2); write(' Nam hay nu: ');
gotoxy(i,j+3); write(' Thuoc lop: ');
gotoxy(i,j+4); write(' Dia chi: ');
gotoxy(i,j+5); write(' Diem Toan: ');
gotoxy(i,j+6); write(' Diem Ly: ');
gotoxy(i,j+7); write(' Diem Hoa: ');
i:=i+15;
With Nguoi do
Begin
Gotoxy(i,j); Readln(STT);
Gotoxy(i,j+1); Readln(Hoten);
Gotoxy(i,j+2); Readln(Gioitinh);
Gotoxy(i,j+3); Readln(lop);
Gotoxy(i,j+4); Readln(Diach);
Gotoxy(i,j+5); Readln(Toan);
Gotoxy(i,j+6); Readln(Ly);
Gotoxy(i,j+7); Readln(Hoa);
End;
Gotoxy(i,j+9); Write('Co ghi vao danh sach khong? C/K ');
Readln(TL);
If upcase(TL) ='C' then

```

```

Begin
Write(DS,NGUOI);
NGUOI.STT:= NGUOI.STT+1;
End;
textbackground(white);
textcolor(blue);
Gotoxy(28,22);
Write(' NHAP TIEP HAY THOI ? L/T ');
Readln(lam);
lam:=upcase(lam);
textbackground(white);
textcolor(5);
End;
Close(DS);
End.

```

Ví dụ 2.20 Nhập dữ liệu quản lý điểm tuyển sinh trung cấp vào tệp, tính tổng điểm và kết quả cho từng người sau đó đọc dữ liệu ra màn hình

Ví dụ 2.20

```

Program Diem_trung_cap;
Uses crt;
Type dong=record
    mhs:byte;
    hoten:string[12];
    toan,ly,tong:real;
    ketqua:string[7];
end;
ds = file of dong;
Var nguoi:dong; dsl:ds; i,n:byte;
Begin
clrscr;
assign(dsl,'dsl.txt');
rewrite(dsl);
write('Nhap bao nhieu nguoi ? '); readln(n);
for i:= 1 to n do
Begin
With nguoi do
Begin
write('Ma ho so ', i); mhs:=i;writeln;

```

```

write('Ho va ten '); readln(hoten);
write('Diem toan '); readln(toan);
write('Diem ly '); readln(ly);
tong:=toan+ly;
if tong>10 then ketqua:='Do' else ketqua:='Truot';
end;
write(dsl,nguoi);
writeln;
End;
close(dsl);
clrscr;
reset(dsl);
writeln('    Du lieu luu tru trong tep DSL.TXT ');
while not eof(dsl) do
with nguoi do
begin
read(dsl,nguoi);
writeln(mhs:3,hoten:15,toan:5:2,ly:5:2,tong:7:2,ketqua:6);
end;
readln;
End.

```

2.4 Tập không kiểu

Như đã biết tệp văn bản chứa đựng trong nó chỉ các ký tự của bảng mã, tất cả các kiểu dữ liệu khác đều phải chuyển về kiểu này. Tệp có kiểu đòi hỏi khắt khe về kiểu dữ liệu của biến và của tệp. Tệp không kiểu là loại tệp không quan tâm đến kiểu dữ liệu. Sự khác nhau giữa tệp có kiểu và tệp không kiểu là ở chỗ sau khi đã khai báo biến tệp việc mở tệp để ghi hoặc để đọc cần chỉ rõ độ lớn (tính bằng Byte) của từng phần tử tệp. Điều này có nghĩa là giữa các phần tử không có ký hiệu phân cách như thường thấy (tệp văn bản là CR và LF, tệp có kiểu là khoảng cách).

a. Khai báo biến tệp

Để khai báo một biến tệp không kiểu chúng ta dùng mẫu:

```
Var Bientep: File;
```

Ví dụ:

```
Var BT:file;
```

b. Mở tệp để ghi - đọc

Sau khi đã khai báo biến tệp, muốn mở tệp không kiểu để ghi chúng ta dùng cặp thủ tục:

Assign(bientep, tentep);

Rewrite(Bientep, n);

Còn mở tệp để đọc là các thủ tục

Assign(bientep, tentep);

Reset(Bientep, n);

ở đây n là độ lớn tính theo Byte do người sử dụng quy định để ghi (hoặc đọc) dữ liệu của từng phần tử (Record).

Nếu chọn n=1 nghĩa là mỗi phần tử có độ dài 1 byte thì dữ liệu ghi vào tệp chỉ có thể thuộc kiểu Byte, char, shortint, còn nếu chọn n=2 thì dữ liệu có thể là Integer, word. Với số thực Real phải chọn n=6.

Giả sử mỗi phần tử ghi vào tệp là một chuỗi dài tối đa là 10 ký tự thì phải chọn n=11 bởi vì mỗi chuỗi khi ghi vào bộ nhớ cần thêm một byte để chứa ký tự cho biết độ dài thực của chuỗi.

Nếu bỏ qua tham số n thì Pascal sẽ dùng giá trị ngầm định là 128.

c. Đọc và ghi tệp không định kiểu

*** Đọc tệp không định kiểu**

BlockRead(bientep, biennho, i, j);

Bản thân từ khoá BlockRead cho thấy việc đọc tệp không kiểu là đọc từng khối (block). Khối ở đây được hiểu là một vùng nhớ lưu trữ dữ liệu. Khối có thể là một biến, một chuỗi, một Record hay một mảng.

- bientep: là biến tệp liên kết với tên tệp đã chỉ ra trong thủ tục assign(...)

- biennho: là một biến đã được khai báo cùng kiểu với các phần tử của tệp, biến nhớ đóng vai trò vùng nhớ đệm để lưu trữ dữ liệu đọc từ phần tử của tệp ra. Từ biến nhớ này chúng ta có thể cho dữ liệu hiện trên màn hình hay in ra máy in.

- i là số phần tử quy định cho mỗi lần đọc

- j là một biến kiểu Word dùng để ghi lại số phần tử thực sự đã được đọc. Tham số j có thể bỏ nếu không cần thiết.

- Ghi tệp không định kiểu

*** BlockWrite(bientep, biennho, i);**

Thủ tục BlockWrite chỉ có 3 tham số, ý nghĩa của các tham số này cũng giống như với thủ tục BlockRead

d. Truy nhập tệp không kiểu

Tệp không kiểu cũng được truy nhập như tệp có kiểu nghĩa là cũng dùng thủ tục Seek(bientep,n) để truy nhập vào phần tử thứ n+1 của tệp.

Điều cần đặc biệt lưu ý là với tệp không kiểu là mỗi lần con trỏ tệp dịch chuyển nó sẽ dịch chuyển một số byte đúng bằng số byte đã quy định trong lệnh Rewrite() hoặc Reset()

Ví dụ 2.21

Program tep_khong_kieu1;

Uses crt;

```

Var bt:file; a:array[1..15] of byte;
    i,j,k:byte; n:word;

Begin
Clrscr;
assign(bt,'tep0kieu.dat');
rewrite(bt,1); {độ dài từng phần tử là 1 byte}
For i:= 1 to 15 do
begin
a[i] := i;
blockwrite(bt,a[i],1); {ghi vào tệp 15 số, mỗi lần ghi 1 số}
end;
j:=a[2]*a[3];
for i:= 0 to 14 do
begin
seek(bt,i);
Blockread(bt,k,1); {đọc từng phần tử của tệp vào biến K}
textcolor(red);
write(k, ' ');
end;
writeln;
seek(bt,filesize(bt)-1); {chuyển con trỏ đến phần tử cuối}
blockwrite(bt,j,1); {sửa giá trị phần tử cuối}
seek(bt,filesize(bt)-1);
blockread(bt,k,1); {đọc lại phần tử cuối}
write(k); {viết phần tử cuối lên màn hình}
close(bt);
readln;
end.

```

Ví dụ 2.21 tạo một biến tệp không kiểu là BT, biến tệp này kết nối với một tệp lưu trữ dữ liệu Tep0kieu.dat.

Đầu tiên chương trình ghi lên tệp 15 số nguyên (1 byte), sau đó đọc dữ liệu từ tệp và viết chúng ra màn hình (phần tử cuối là 15). Tiếp đó sửa dữ liệu ở phần tử cuối và lại viết phần tử đó ra màn hình (bây giờ là 6).

Ví dụ 2.22

Chương trình dưới đây tạo tệp không kiểu BT, các phần tử tệp là chuỗi có độ dài bằng 3, lệnh mở tệp để ghi phải quy định độ dài phần tử bằng 4 vì trong bộ nhớ string[3] sẽ chiếm 4 bytes.

```

Program tep_khong_kieu2;
Uses crt;
Var bt:file; a:array[1..30] of string[3];
    i,j:byte; k:string[3];

Begin
  Clrscr;
  assign(bt,'tep0kieu.dat');
  rewrite(bt,4); { độ lớn của mỗi phần tử là 4 bytes }
  For i:= 1 to 24 do
  begin
    a[i] := chr(i+64)+chr(i+65)+chr(i+66);
    blockwrite(bt,a[i],1);
  end;
  reset(bt,4);
  for i:= 0 to 23 do
  begin
    seek(bt,i);
    Blockread(bt,k,1);
    textcolor(green);
    write(k, ' ');
  end;
  close(bt);
  readln;
end.

```

Ví dụ 2.23

Nhập vào tệp các phần tử là Record và sau đó viết chúng ra màn hình. Trong phần khai báo Record chúng ta chọn Hoten là string[15] và Diem thuộc kiểu Real như vậy độ dài của mỗi phần tử phải là $15+1+6=22$.

```

Program tep_khong_kieu4;
Uses crt;
Type hs = record
    hoten:string[15];
    diem:real;
end;
Var
    bt:file; k,nguai:hs; i,j:byte;

```



```

Begin
Clrscr;
assign(bt,'tep0kieu.dat');
rewrite(bt,22);
write('Nhap bao nhieu nguoi? ');
readln(n);
  For i:= 1 to n do
  with nguoi do
  Begin
  write('Ho va ten : '); readln(hoten);
  Write('Diem tong : '); readln(diem);
  blockwrite(bt,nguoi,1);
  end;
  for i:= 0 to n-1 do
  begin
  seek(bt,i);
  Blockread(bt,k,1);
  textcolor(red);
  with k do
  writeln(hoten,' ',diem:5:2);
  end;
  close(bt);
  readln;
  end.

```

Để tìm hiểu thêm về tệp không kiểu chúng ta có thể mở Help của Pascal và xem trong đó ví dụ về việc dùng tệp không kiểu để sao chép dữ liệu từ tệp này sang tệp khác.

3. Dữ liệu kiểu tập hợp

3.1 Khái niệm tập hợp

a. Khái niệm tập hợp

Một tập hợp bao gồm n phần tử cùng kiểu dữ liệu ($0 \leq n \leq 255$), kiểu của các phần tử phải là kiểu vô hướng đếm được (nguyên, ký tự, logic, liệt kê, đoạn con). Số thứ tự các phần tử trong tập hợp luôn nằm trong khoảng 0 - 255.

b. Khai báo kiểu và gán dữ liệu vào tập hợp

Để mô tả kiểu tập hợp, Pascal dùng từ khoá **Set of** tiếp đó là **kiểu dữ liệu** cơ bản của các phần tử tạo nên tập hợp. Cách viết kiểu dữ liệu này phải tuân theo những quy định mà Pascal đã thiết kế:

- * Với Pascal 7.0 số phần tử không quá 256 và số thứ tự của các phần tử phải tăng dần
- * Kiểu dữ liệu cơ bản của các phần tử của tập hợp có thể viết tường minh hoặc thông qua các giá trị cụ thể. Ví dụ :

```
Type
a = set of Char;
b = set of Byte;
c = set of 1..100;
d = set of 'a'..'z';
Maytinh = set of (Compact, Mitac, IBM, CMS);
```

Trong ví dụ trên kiểu dữ liệu của các phần tử của tập hợp a, b được khai báo tường minh, a là tập các ký tự, b là tập các số nguyên. Kiểu dữ liệu các phần tử của tập c và d được xác lập qua các giá trị viết sau SET OF, tập c sẽ là tập các số nguyên trong khoảng từ 1 đến 100, tập d là tập các chứa các chữ cái thường còn Maytinh là tập hợp của 4 phần tử đã liệt kê.

Để thấy rõ hơn chúng ta xét ví dụ sau:

Ví dụ 2.24

```
Program Taphop1;
Uses crt;
Type
mau= set of (xanh, hong, tim, vang, den, nau);
sn1 = set of 0..100;
sn2 = 100..200;
chucai1 = set of 'a'..'z';
chucai2 = set of 'A'..'Z';
Kytu = set of Char;
Var
a,b:mau; b1:sn1;
b2 : set of sn2; c :chucai1; d:chucai2; e:kytu;
i,j,k:word;
tl : set of boolean;
BEGIN
clrscr;
i:=300; j:=160; k:=i+j;
a:=[xanh.. vang];
b:=[tim,nau];
b1:=[15,255,i+j,i*j];
b2:=[1, 200, 155, 123, 145];
c:=['z','à','k'];
```

```

d:=[ '2', '3', '4'];
tl:=[true,false];
readln;
END.

```

Ví dụ 2.24 khai báo năm kiểu tập hợp là Mau, Sn1, Chucai1, Chucai2, Kytu còn Sn2 là kiểu dữ liệu đoạn con. Biến a thuộc kiểu tập hợp MAU, biến B1 và B2 thuộc kiểu tập hợp số nguyên, biến c và d thuộc kiểu tập hợp chữ cái, biến e thuộc kiểu ký tự còn biến T1 thuộc kiểu boolean. Các phép gán trong ví dụ sẽ xác định một số tập hợp, chẳng hạn câu lệnh

```

a:=[xanh, hong, den, tim];

```

xác định tập hợp a gồm các phần tử xanh, hong, den, tim.

```

e:=[ '0'..'9', 'a'..'z'];

```

xác định tập hợp e gồm 10 ký tự 0, 1,...9 và 26 ký tự 'a', 'b', ... 'z'

Khi khai báo kiểu tập hợp cần chú ý một số điều sau đây:

* **Thứ nhất:** Thứ tự các phần tử trong tập hợp luôn theo chiều tăng dần, nếu chúng ta khai báo:

```

sn1 = set of 200..100;

```

hoặc

```

chucai = set of 'z' .. 'a';

```

thì Pascal sẽ thông báo lỗi: Lower bound greater than upper bound.

* **Thứ hai:** Kiểu dữ liệu của các phần tử mặc dù có thể là số nguyên song khi khai báo lại chỉ có thể là kiểu Byte.

Lệnh SN0 = set of Byte; là đúng
 Các lệnh khai báo sau là sai:
 SN1 = set of Word;
 SN2 = set of Integer;
 SN3 = set of Shortint;
 Lỗi mà Pascal thông báo là:
 Error 23: Set base type out of Range

* **Thứ ba:** Những giá trị mà chúng ta khai báo sau từ khoá SET OF sẽ dẫn tới một trong hai khả năng:

- Nếu là khai báo kiểu liệt kê, kiểu Boolean thì đây chính là các giá trị mà tập hợp có thể nhận. Trở lại ví dụ 2.24 các phép gán

```

a:=[xanh, tim, luc];
tl:=[tru, fal];

```

sẽ bị báo lỗi vì phần tử "luc" không có trong danh sách liệt kê của màu, còn Tru và Fal không có trong kiểu boolean.

- Với những kiểu còn lại (nguyên, ký tự, đoạn con) những khai báo sau từ khoá Set of không phải là giá trị của các phần tử của tập hợp mà là thứ tự của các phần tử đó, Kiểu của các dữ liệu đó sẽ cho biết các phần tử của tập hợp sẽ phải thuộc kiểu gì

Ví dụ: 2.25

Type

sn1 = set of 0..100; {các phần tử tập hợp phải là số nguyên}

kytu = set of char; {các phần tử tập hợp phải thuộc kiểu ký tự}

Var

a : sn1; b : kytu;

.....

Với khai báo như trong ví dụ 2.25 chúng ta không thể dùng phép gán:

b := ['a', 12, 'c']; (các phần tử của tập b phải là số)

c := [1, 2, 100]; (các phần tử của tập c phải là ký tự)

* **Thứ tự:** Không được gán trực tiếp các số nguyên có giá trị lớn hơn 255, ví dụ phép gán

b2 := [100, 200, 256]; sẽ bị báo lỗi Error 76: Constant out of Range

* **Thứ tự:** Các phần tử trong tập hợp có thể là hằng, biến hay biểu thức. Nếu là tập số nguyên thì giá trị các biến hay biểu thức trong phép gán có thể lớn hơn 255

Trong ví dụ 2.25 phép gán sau không bị báo lỗi

i := 300; j := 160; k := i + j;

b1 := [15, 255, i + j, i * j];

3.2 Phân loại tập hợp

a. Tập hợp cùng kiểu

Các tập hợp được coi là cùng kiểu nếu chúng được xây dựng trên cùng một kiểu cơ bản.

Giả sử chúng ta khai báo các tập hợp như ví dụ sau:

Ví dụ 2.26

Type

mau = set of (xanh, hong, tim, vang, den, nau);

mau_xe = set of (xanh..vang);

sn1 = set of 0..100;

sn2 = 100..200;

sn3 = set of Byte;

chucai1 = set of 'a'..'z';

chucai2 = set of 'A'..'Z';

```

Kytu = set of Char;
Var
  a:sn1; b:sn2; c:sn3; mau_oto:mau; chu:kytu;
.....
thì các nhóm tập hợp sau đ đây là cùng kiểu
mau, mau_xe
sn1, sn2, sn3
chucai1, kytu
chucai2, kytu

```

b. Hình thành một tập hợp

Một tập hợp được hình thành khi chúng ta gán cho *biến tập hợp* các giá trị cụ thể, các giá trị này được đặt trong hai dấu ngoặc vuông.

Ví dụ 2.27

```

c := [2, 15, 30, 100..200];
mau_oto := [xanh, den, vang, tim];
a := [];
chu := ['A'..'Z'];

```

Nếu một tập hợp không chứa một phần tử nào cả thì nó được gọi là **tập rỗng** và được ký hiệu là []. Trong ví dụ 2.28, a là một biến tập hợp rỗng.

Tập hợp rỗng được xem là cùng kiểu với mọi kiểu tập hợp.

3.3 Các phép tính trên tập hợp

Các phép tính giới thiệu sau đây chỉ thực hiện được trên các tập hợp cùng kiểu, nghĩa là các phần tử của mọi tập hợp phải cùng thuộc một kiểu cơ bản.

a. Phép gán

Phép gán được dùng để hình thành nên một tập hợp. Có thể gán một tập hợp cho một biến tập hợp cùng kiểu hoặc gán một biến tập hợp cho một biến tập khác cùng kiểu. (xem ví dụ 2.28)

b. Phép hợp

Phép hợp được ký hiệu bởi toán tử " + ". Hợp của hai tập hợp A và B khi đó có thể viết là $A + B$

Hợp của hai tập A và B là một tập hợp gồm các phần tử thuộc A hoặc thuộc B.

Giả sử các tập A, B, C được hình thành như sau:

```
A := [1..8,10..15];
```

$B := [5..9, 20,30];$

$C := A + B;$

Tập C được hình thành bởi phép hợp A và B nên sẽ gồm các phần tử [1..15,20,30]

c. Phép giao

Giao của hai tập A và B ký hiệu là $A*B$ là một tập hợp gồm các phần tử đồng thời thuộc cả A và B.

$C := A*B;$ tập C sẽ có các phần tử [5, 6, 7, 8]

d. Phép trừ

Phép trừ hai tập hợp A và B ký hiệu là $A - B$ cho ta một tập gồm các phần tử thuộc A nhưng không thuộc B

$C := A - B;$ tập C sẽ có các phần tử [1..4, 10..15]

còn $[1..5] - [1..10] = []$

e. Phép thuộc về

Phép thuộc về ký hiệu là IN là một phép thử xem một giá trị hay một biến (không phải là biến tập) có thuộc về một tập khác hay không? Nếu có giá trị phép thử là True, ngược lại phép thử cho kết quả False.

Ví dụ 2.28

$T1 := ['a'..'z'];$

$T2 := [true,false];$

$T3 := 'c';$

Write(T3 in T1); cho kết quả True

Write(false in T2); cho kết quả True

Write('1' in T1); cho kết quả False

3.4. Các phép so sánh trên tập

Các phép so sánh nói trong mục này chỉ được thực hiện trên các tập cùng kiểu. Kết quả của phép so sánh thuộc kiểu Boolean nghĩa là cho ra một trong hai giá trị True hoặc False.

a. Phép bằng, ký hiệu là =

Hai tập A và B gọi là bằng nhau nếu mọi phần tử của A đều thuộc B và mọi phần tử của B đều thuộc A không phụ thuộc vào thứ tự các phần tử.

Ví dụ phép so sánh

$[1..6] = [1,2,4,5,3,6]$ cho kết quả True

$['a', 'b', 'c'] = ['A', 'B', 'C']$ cho kết quả False

b. Phép không bằng, ký hiệu là <>

Hai tập A và B gọi là không bằng nhau nếu tập A có ít nhất một phần tử không thuộc tập B hoặc tập B có ít nhất một phần tử không thuộc tập A.

[1..6] <> [1,2,4,5,3,6] cho kết quả False

['a', 'b', 'c'] <> ['A', 'B', 'C'] cho kết quả True

c. Phép nhỏ hơn hoặc bằng, ký hiệu là <=

Tập A gọi là nhỏ hơn hoặc bằng tập B nếu mọi phần tử của tập A đều thuộc tập B.

[1..6] <= [1,2,4,5,3,6,8,9] cho kết quả True

[1..6] <= [1,2,4,5,3] cho kết quả False

d. Phép lớn hơn hoặc bằng, ký hiệu là >=

Tập A gọi là lớn hơn hoặc bằng tập B nếu mọi phần tử của tập B đều thuộc tập A.

[1..6] >= [1,2,4,5] cho kết quả True

[1,5] >= [1,2,4,5] cho kết quả False

Chú ý: Trong Pascal không tồn tại các phép so sánh nhỏ hơn < và lớn hơn >

Các ví dụ về dữ liệu kiểu tập hợp

Ví dụ 2.29 : Tìm số nguyên tố

Ta đã biết số nguyên tố là các số chỉ chia hết cho 1 và chính nó (ngoại trừ số 1) để tìm các số nguyên tố trong khoảng từ 1 đến n chúng ta sử dụng nguyên lý sàng Eratosthène. Nguyên lý này có thể phát biểu như sau: nếu a là một số nguyên tố trong tập [1..n] thì không thể tồn tại một số b thuộc tập [1..n] mà b là bội số của a. Nói cách khác nếu ta đã tìm được a là một số nguyên tố thì ta phải loại trừ khỏi tập [1..n] tất cả các số là bội số của a.

```
Program Tim_so_nguyen_to;
Uses crt;
Const n=255;
Type Songuyen = 1..n; {định nghĩa kiểu dữ liệu Songuyen gồm n số}
Var snt,sang: set of songuyen;
    so:songuyen; i:integer;
Begin
  clrscr;
  Writeln('Cac so nguyen to tu 2 den ',n);
  snt:=[]; {Tập snt ban đầu là tập rỗng}
  sang:=[2..n]; {Xác lập tập Sang gồm số nguyên tố đầu tiên (2) và các số khác}
  so:=2; {Gán vào biến So số nguyên tố đầu tiên = 2}
  Repeat
    While not (so in sang) do so:=so+1;
    snt:=snt+[so]; {Bổ xung số nguyên tố vừa tìm được vào tập Snt}
```

```

i:=so;
Write(so, ' '); {Viết số nguyên tố vừa tìm được ra màn hình}
While i<=n do
Begin
sang:=sang-[i]; i:=i+so; {Xoá các số là bội số của số nguyên tố vừa tìm được}
End;
Until sang = [];
Readln;
End.

```

Ví dụ 2.30

Trò chơi gieo xúc xắc: nguyên tắc chơi như sau: người chơi chọn một số trong khoảng từ 1 đến 6, máy sẽ gieo 3 con xúc xắc để được 3 số. Nếu số của người chơi nằm trong tập số mà máy đã gieo thì người chơi thắng, ngược lại người chơi thua. Nếu ba lần chơi mà người chơi đều thắng thì tuyên bố người chơi trúng số độc đắc.

```

Program xuc_xac;
Uses crt;
Var xucxac: set of 1..6;
    So,i,dem,a,b,c: 1..6;tl:char;
Begin
Clrscr;
Repeat
clrscr;
For i:= 1 to 3 do
Begin
Textcolor(5);
Write('Moi ban chon mot so tu 1 den 6 '); readln(so);
a:=random(6); b:=random(6); c:=random(6);
xucxac:=[a,b,c];
Writeln('So ma ban da chon ',so);
Writeln('So ma may da gieo ',a,' ',b,' ',c);
If so in xucxac then
Begin
textcolor(14);
writeln('Ban thang van thu ',i);

```



```
dem:=dem+1;
End
else
Writeln('Ban thua van thu ',i);
If dem = 3 then Writeln('Ban trung doc dac - Xin chuc mung');
End;
Writeln;
Write('Choi tiep hay thoi? C/K ');
Readln(tl);
Until upcase(tl)='K';
End.
```

Bài tập ứng dụng chương 2

1. Nhập vào thư viện n đầu sách ($n \leq 50$), các thông số cần đưa vào là: Tên sách, Tên tác giả, Năm xuất bản, Tên nhà xuất bản, Số trang, Giá tiền. Hiện danh mục sách đã nhập lên màn hình.

2. Dữ liệu tuyển sinh gồm các trường: SBD, HOTEN, TOAN, LY, HOA, TONG, KETQUA.

Lập chương trình nhập dữ liệu cho một phòng thi không quá 40 thí sinh, số liệu trong trường $Tong = Toan + Ly + Hoa$, trường Ketqua điền vào chữ "DO" nếu tổng ≥ 20 , còn lại là "TRUOT".

Hiện lên màn hình những người DO

3. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số chứng từ.

Lập chương trình nhập hàng vào kho, sau mỗi mặt hàng nhập lại hỏi: "Nhập tiếp hay thôi? C/K". Nếu bấm "K" thì kết thúc nhập và hiện danh mục hàng đã nhập lên màn hình.

4. TOADO là kiểu bản ghi chứa tọa độ (cột, dòng) của ba điểm trên màn hình. Lập chương trình kiểm tra xem ba điểm này có thẳng hàng không, nếu không thẳng hàng thì tạo nên tam giác gì (Vuông, Cân, ...)

5. Dữ liệu quản lý vé xe lửa gồm: Số toa, số ghế, giá vé, ga đi, ga đến, Đã bán vé chưa. Mỗi toa tàu có 50 ghế đánh số thứ tự từ 1 đến 50.

Lập chương trình nhập vào những ghế đã bán vé, thông báo còn bao nhiêu chỗ chưa bán vé.

6. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số chứng từ. Ngày nhập bao gồm Ngay, Tháng, Năm.

Lập chương trình nhập hàng vào kho, kết thúc nhập khi mã hàng bằng dấu "*" .

Cho một tháng nào đó, hiện những hàng đã nhập trong tháng đó.

7. Dữ liệu quản lý đất đai bao gồm hai trường Chủ đất và Đất. Chủ đất bao gồm: Mã hồ sơ, Họ tên, Mã đất, Địa chỉ. Đất bao gồm: Mã đất, Diện tích, Loại đất, Mục đích sử dụng.

Nếu mục đích sử dụng là trồng trọt thì có thuế nông nghiệp

Nếu mục đích sử dụng là công ích (trường học, bệnh viện, sân vận động) thì không phải nộp thuế.

Nếu mục đích sử dụng là kinh doanh bất động sản thì có thuế đất và thuế môn bài.

Nhập dữ liệu cho n người ($n \leq 30$) sau đó hiện lên màn hình 10 người sử dụng nhiều đất nhất.

8. Viết chương trình nhập vào tệp văn bản F một số dòng văn bản, đếm xem trong tệp các ký tự a,b,...z mỗi ký tự xuất hiện bao nhiêu lần?

9. Thông tin về thí sinh thi đại học bao gồm Hoten, namsinh, diemthi, ketqua. Thông tin trên được ghi trong một tệp văn bản, mỗi trường trên một dòng. Lập chương trình in ra màn hình mỗi thí sinh trên một dòng

Ví dụ:

Trong tệp	In ra màn hình
Tran Tam 1980 8 Kha	Tran Tam 1980 8 Kha

10. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số hoá đơn. Lập chương trình nhập dữ liệu từ bàn phím và ghi vào tệp có kiểu. Cho biết một số hoá đơn, hiện lên màn hình số liệu ứng với hoá đơn đó.

11. T1 và T2 là các tệp văn bản, nội dung T1, T2 nhập từ bàn phím, hãy nối T1 với T2 thành T3, hiện T3 lên màn hình.

12. T1 là tệp văn bản chứa 15 ký tự dạng số Integer. Lập chương trình chuyển T1 thành tệp số nguyên T2, tính tổng các số và viết kết quả lên màn hình.

13. Giả sử thư mục hiện hành là C:\TP\BIN. Cho biết một tên tệp từ bàn phím, kiểm tra xem trong thư mục hiện hành có tệp đó chưa, nếu có thì hiện nội dung tệp, nếu chưa thì mở tệp mới để ghi dữ liệu.

14. Hai tập A và B gọi là liên thông nếu chúng có ít nhất một phần tử chung. Nếu các cặp tập hợp sau là liên thông (A,B), (B,C), (C,D), (D,E) thì ta nói có một đường nối từ A đến E.

Lập chương trình tìm đường nối giữa n điểm A_1, A_2, \dots, A_n trên màn hình biết rằng tọa độ giữa các các điểm lần lượt là $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. In ra tất cả đường nối theo mẫu:

$A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n$

Nếu giữa các cặp điểm A_i, A_j không có đường nối thì trả lời không có.

Ví dụ:

$A_1(1,2), A_2(2,3), A_3(3,4), A_4(4,1), A_5(5,6)$

$A_1 \rightarrow A_5$ không có đường nối

$A_1 \rightarrow A_2$ có hai đường nối

$A_1 \rightarrow A_2$

$A_1 \rightarrow A_4 \rightarrow A_3 \rightarrow A_2$

Chương 3

Đơn vị chương và trình Thư viện chuẩn

Khái niệm Đơn vị chương trình giống như một cuốn sách và Thư viện chuẩn giống như một tủ sách. Thư viện chuẩn của Pascal không chứa hết mọi đơn vị chương trình, lý do là vì tiết kiệm bộ nhớ. Chương này bạn đọc nên dành nhiều thời gian cho việc khai thác các Unit đã thiết kế trong Pascal, tuy nhiên đã là một người lập trình thì không thể không biết tự tạo ra các Unit. Những phần người đọc cần nắm được là:

- Các thủ tục hoặc hàm có trong các Unit System, Graph, Crt, Dos..
- Phương pháp xây dựng các Unit
- Những nguyên tắc cần tuân thủ khi tham chiếu đến các Unit
- Biên dịch Unit từ chương trình nguồn sang dạng mã máy (đuôi tệp là TPU)

1. Khái niệm đơn vị chương trình (Unit)

Thuật ngữ Unit trong ngôn ngữ lập trình Pascal được dịch là "Đơn vị chương trình". Mỗi Unit được xem như một modul nhỏ chứa đựng một số công cụ cần thiết giúp cho người lập trình có thể dễ dàng thiết kế chương trình. Các Unit có thể gộp chung lại thành thư viện chương trình hoặc để phân tán trong một thư mục quy ước của Pascal.

Lệnh tham chiếu đến Unit được đặt ở đầu chương trình với cú pháp:

USES tênUnit;

Ví dụ: USES CRT, GRAPH;

Các Unit được tổ chức trong Pascal dưới hai dạng:

* Các file độc lập với phần mở rộng là TPU (Turbo Pascal Unit), ví dụ GRAPH.TPU

* File thư viện chuẩn với phần mở rộng TPL (Turbo Pascal Library), ví dụ TURBO.TPL

Thư viện chuẩn của Pascal chứa đựng một số Unit cơ bản, hay được dùng đến, chúng được đóng gói lại và được để cùng chỗ với tệp khởi động TURBO.EXE. Thông thường chương trình Pascal được cài đặt trên ổ đĩa C, với Pascal 7.0 đường dẫn đến nơi lưu trữ các tệp TURBO.EXE và TURBO.TPL là:

C:\TP\BIN

Khi tệp TURBO được gọi, nghĩa là chương trình Pascal được khởi động thì tệp TURBO.TPL cũng tự động được tải vào bộ nhớ. Lúc này các Unit chứa trong thư viện chuẩn sẽ sẵn sàng được tham chiếu đến. Việc truy cập đến các Unit trong thư viện chuẩn nhanh hơn so với truy cập vào các Unit độc lập vì chúng đã thường trú trong bộ nhớ. Khi có lời gọi một Unit nào đó bao giờ Pascal cũng ưu tiên tìm kiếm chúng trong thư viện chuẩn, nếu không tìm thấy thì tiếp tục tìm kiếm ở bên ngoài.

2. Thư viện chuẩn

Thư viện chuẩn của Pascal có tên là TURBO.TPL, thư viện này ngầm định ban đầu chứa năm Unit là:

2.1 Crt

CRT bao gồm các thủ tục quản lý màn hình, bàn phím và âm thanh, nhờ có Unit này mà người lập trình có thể thiết kế giao diện chương trình tương đối đẹp

2.2 Dos

Unit này chứa các chức năng quản lý tệp, đĩa, ngày tháng. Ngoài ra cũng có thể dùng Unit này để gọi trực tiếp các lệnh của hệ điều hành DOS hoặc các ngắt hệ thống.

2.3 Overlay

Unit này được sử dụng khi chương trình nguồn có dung lượng lớn. Mặc dù đã được biên dịch sang dạng mã máy song đôi khi do bộ nhớ trong (RAM) không đủ nên không thể tải cùng một lúc toàn bộ Code của chương trình nguồn. Sử dụng OVERLAY có thể tải từng phần

của chương trình nguồn vào bộ nhớ trong để chạy. Với những máy PC có Ram 128MB, hoặc 256 MB thì không mấy khi phải sử dụng Unit này.

2.4 Printer

Unit này định nghĩa tên máy in là LST. Việc kết xuất thông tin bằng lệnh Write khi tham chiếu đến LST sẽ cho phép ta in ra máy in các kết quả bài toán.

2.5 System

Đây là Unit cơ bản và quan trọng nhất của Pascal, nó chứa các thủ tục vào ra như Read, Write..., các hàm sơ cấp thông dụng như Ln, Sqrt, Sin, Cos... Khi Pascal được khởi động và thư viện chuẩn được nạp vào bộ nhớ thì Unit này cũng tự động liên kết với mọi chương trình vì thế ở đầu chương trình không cần đến lời gọi USES SYSTEM.

Ngoài những Unit có sẵn trong thư viện chuẩn, Pascal còn cho phép người lập trình thiết kế thêm các Unit mới, các Unit này sau đó sẽ được lưu vào tệp Turbo.tpl và được xem như một bộ phận cấu thành của ngôn ngữ. Việc chuyển các Unit của người lập trình vào thư viện chuẩn được thực hiện thông qua trình tiện ích TPUMOVER sẽ đề cập đến ở mục sau.

3. Các Unit khác

Pascal có khoảng 20 Unit độc lập lưu trữ trong thư mục TPUNIT. Các Unit này được phân chia thành 4 nhóm gồm:

- * Interface Unit: nhóm Unit chung cho mọi ứng dụng
- * Objects Unit: danh mục (theo vần a,b,c..) của các Unit thuộc kiểu Turbo vision
- * System Unit: Unit hệ thống
- * Turbo vision Unit: Unit phục vụ cho lập trình hướng đối tượng

Muốn tìm hiểu về một Unit nào đó ta có thể đọc phần giới thiệu Unit trong các tệp cùng tên với phần mở rộng là INT, các tệp này lưu trữ trong thư mục TP\DOC. Các tệp đuôi INT này là tệp văn bản nên có thể đọc chúng bằng bất kỳ phần mềm soạn thảo văn bản nào ví dụ Word, NC... Cũng xin lưu ý thêm là một số Unit mặc dù có tên trong thư mục TPUNIT như GRAPH3.TPU, TURBO3.TPU nhưng do đã quá lỗi thời nên không được giới thiệu trong thư mục TP\DOC. Dưới đây là một vài Unit thông dụng

3.1 Graph

Unit GRAPH được lưu trữ trong một File riêng có tên là GRAPH.TPU. Unit này chứa các thủ tục và hàm cho phép lựa chọn loại màn hình, số dòng, cột và các thủ tục vẽ các hình cơ bản, tô màu nét vẽ hoặc màu nền các hình khép kín...

3.2 Memory

Unit Memory chứa các phục vụ quản lý vùng nhớ cấp phát cho chương trình, các chương trình với phần mở rộng EXE thì vùng nhớ cơ sở được cấp phát là 640 Kb. Đoạn thấp nhất của vùng nhớ này (256 byte) dành cho việc đánh địa chỉ các biến, hàm, thủ tục trong chương trình.

3.3 Menus

Unit Menus chứa các phục vụ thiết kế thực đơn như chọn phím nóng (Hot key), tạo dải thực đơn (Menu Bar), khung thực đơn (menu box)...

3.4 Views

Unit Views chứa các thủ tục và hàm liên quan đến việc quan sát các khung hoặc cửa sổ như thanh cuộn (Scroll Bar), con trỏ (Cursor), chuyển trang (Nextview, Preview)...

4. Xây dựng các Unit

Các Unit cũng được viết và lưu vào thư mục ngầm định TP\BIN như các chương trình Pascal khác. Tập lệnh nguồn của Unit sẽ có phần mở rộng là PAS. Để soạn thảo một Unit chúng ta có thể dùng một phần mềm soạn thảo văn bản bất kỳ nhưng nói chung người ta vẫn hay dùng công cụ soạn thảo (Text Editor) của Pascal. Mặc dù được lưu dưới dạng một tập đuôi PAS nhưng các Unit không thể chạy bằng tổ hợp lệnh Ctrl – F9 như các chương trình bình thường.

Các chương trình Pascal khi biên dịch sang dạng mã máy sẽ có phần mở rộng là EXE (Execute), còn các Unit khi biên dịch sẽ có phần mở rộng là TPU (Turbo Pascal Unit).

Cấu trúc một Unit bao gồm bốn phần cơ bản sau đây:

4.1 Phần tiêu đề

Phần này chỉ gồm một dòng mở đầu là từ khoá UNIT sau đó là tên Unit, kết thúc dòng bằng dấu chấm phẩy.

Unit tenUnit;

Tên Unit phải viết cách từ khoá Unit ít nhất một khoảng trống, các ký tự của tên phải viết liền nhau theo các quy định giống như tên chương trình.

Cần phải đặc biệt lưu ý rằng Pascal quy định bắt buộc tên Unit sau này sẽ được dùng làm tên tệp để lưu trữ tệp nguồn của Unit (đuôi Pas) đồng thời cũng là tên của đơn vị chương trình (đuôi Tpu) do vậy khi đặt tên không nên chọn tên quá dài, nên chọn các tên gợi nhớ đến nội dung của Unit để tiện sử dụng về sau.

4.2 Phần khai báo chung

Phần này bắt đầu bằng từ khoá INTERFACE

Bản thân từ khoá Interface có nghĩa là dùng chung, điều này quy ước rằng ở đây chúng ta phải khai báo các kiểu dữ liệu mới, các biến, hằng, hàm, thủ tục mà sau này các chương trình tham chiếu đến Unit sẽ sử dụng. Có thể hình dung rằng những gì viết trong phần này sẽ là những tài nguyên mà Unit có thể cung cấp cho các chương trình ngoài, nó giống như bản thực đơn mà nhà hàng trình bày cho thực khách.

Ví dụ chúng ta xây dựng một Unit lấy tên là HHP (hình học phẳng) trong đó có các hàm tính diện tích, chu vi các hình chữ nhật, tam giác, tròn là dtcn, dttg, dttr, cvcn, cvtg, cvtr. Khi đó phần khai báo chung sẽ là:

```
INTERFACE  
Function dtcn( a,b: real) : real;  
Function dttg( a,b,c: real) : real;  
Function dttr( a: real) : real;  
Function cvcn( a,b: real) : real;  
Function cvtg( a,b,c: real) : real;  
Function cvtr( a: real) : real;
```

Ví dụ trên khai báo ba hàm tính diện tích và ba hàm tính chu vi, sau này khi các chương trình tính toán tham chiếu đến Unit hhp thông qua lệnh USES HHP sẽ có thể dùng trực tiếp các hàm này. Các tham số khai báo trong cả sáu hàm đều là các tham trị nghĩa là giá trị của chúng sẽ không biến đổi khi quay về làm việc với chương trình chính. Việc sử dụng tham số là tham trị hay tham biến sẽ tùy thuộc vào các bài toán cụ thể.

4.3 Phần nội dung

Phần nội dung bắt đầu bằng từ khoá IMPLEMENTATION. Tại đây chúng ta sẽ xây dựng nên các hàm, thủ tục mà tên của chúng đã được giới thiệu ở phần Interface. Việc xây dựng các chương trình con đã được giới thiệu ở phần lập trình cơ bản nên chúng ta không đề cập đến ở đây.

Điều cần nhấn mạnh là những hàm và thủ tục chúng ta đã giới thiệu ở Interface thì bắt buộc phải được xây dựng ở đây bởi lẽ chúng chính là những gì mà các chương trình tham chiếu đến Unit cần sử dụng. Ngược lại trong phần này chúng ta có thể xây dựng các hàm và thủ tục khác hoặc khai báo các biến, hằng mới không có trong phần Interface, chúng được xem là phần riêng của Unit mà các chương trình tham chiếu đến Unit không được phép sử dụng. Ví dụ:

```
IMPLEMENTATION  
Function dtcn( a,b: real) : real;  
Var s : real;  
Begin  
S:=a*b; Dtcn:=s;  
End;  
...  
Function cvcn( a,b: real) : real;  
Var s : real;  
Begin  
S:=a*b; Dtcn:=s;  
End;
```


4.4 Phần khởi động

Phần khởi động đặt giữa hai từ khoá BEGIN và END, sau End là dấu chấm giống như các chương trình Pascal bình thường. Trong phần này chúng ta có thể đưa vào các lệnh gán giá trị ban đầu cho biến hoặc lời gọi các thủ tục riêng nào đó cần cho quá trình tạo hàm và thủ tục của phần IMPLEMENTATION .

Phần khởi động là không bắt buộc phải có, trong trường hợp không có phần này thì chúng ta bỏ đi từ khoá BEGIN song vẫn phải có từ khoá END. để báo hiệu kết thúc Unit. Dưới đây là cấu trúc tổng thể của một Unit.

```
Unit Tên_Unit ;
Interface .....
Uses Tên_Unit; (tên các Unit sẽ dùng trong các chương trình con của Unit này)
Const .... (khai báo hằng)
Type ... (khai báo kiểu dữ liệu)
Var ... (khai báo biến cho các chương trình con trong Unit )
Tên các Procedure và Function của Unit

Implementation
(nội dung của từng chương trình con)

Begin
Phần khởi tạo giá trị ban đầu (tùy chọn)
End.
```

Về bản chất Unit cũng là một chương trình của Pascal, nó được xây dựng trên cơ sở các từ khoá và từ vựng mà Pascal đã thiết kế do vậy từ bên trong Unit chúng ta lại có thể tham chiếu đến các Unit khác không phân biệt là Unit chuẩn của Pascal hay Unit do người sử dụng tạo ra.

Ví dụ 3.1 Xây dựng Unit HHP chứa hàm tính diện tích hình chữ nhật.

```
Unit HHP;
Interface
Function dtcn( a,b: real) : real;
Implementation
Function dtcn( a,b: real) : real;
Uses crt, dos;
Var s : real;
Begin
    S:=a*b;
    Dtcn:=s;
End;
```

Sau khi đã soạn thảo xong Unit chúng ta cần ghi Unit thành tệp với **tên tệp trùng với tên đã chọn ở phần tiêu đề**. Các Unit lúc này mới chỉ ở dạng tệp văn bản nên khi ghi vào đĩa Pascal sẽ gán phần đuôi là .Pas. Các chương trình Pascal có thể tham chiếu đến các Unit này nhưng Pascal sẽ phải làm công việc biên dịch trước do đó sẽ mất một thời gian. Tốt nhất để có thể sử dụng các Unit chúng ta phải dịch chúng sang dạng mã máy thông qua chức năng Compile trên thực đơn chính của Pascal. Trước khi dịch cần chú ý rằng tùy chọn Destination của thực đơn Compile phải là Disk chứ không phải là Memory. Có thể dùng một trong ba cách dịch đã biết trong chức năng này là Make, Build, hoặc Run tuy nhiên cách hay được dùng là từ màn hình soạn thảo Unit, bấm tổ hợp phím Alt + F9, nếu không gặp một lỗi nào Pascal sẽ dịch Unit sang dạng mã máy và tự động lưu tệp đã biên dịch vào thư mục TP\BIN. Tệp này sẽ có tên giống như tên tệp nguồn nhưng phần mở rộng sẽ là TPU.

Các TPU có thể lưu vào thư mục khác do người sử dụng tự chọn hoặc thư mục UNITS của Pascal. Nếu lưu vào thư mục khác thì người sử dụng cần khai báo đường dẫn đến nơi lưu trữ thông qua lựa chọn Options – Directories – Unit Directories.

Trong trường hợp việc dịch gặp lỗi Pascal sẽ thông báo lỗi như khi dịch một chương trình thông thường.

Chú ý:

Nếu tên tệp nguồn của Unit trong thư mục TP\BIN khác với tên ghi sau từ khoá Unit thì việc dịch không thông báo lỗi, song Pascal không tạo ra được TPU.

Lệnh tham chiếu Uses TênUnit trong các chương trình trước hết sẽ tham chiếu đến các TPU, nghĩa là đến các Unit đã biên dịch, nếu không tìm thấy các TPU, Pascal sẽ tìm các chương trình nguồn của Unit và tự động biên dịch chúng trước khi thực hiện chương trình.

Ví dụ 3.2: xây dựng Unit

Chúng ta sẽ xây dựng một Unit lấy tên là HHP (hình học phẳng). Unit này sẽ tạo nên một số hàm dùng để tính diện tích, chu vi các hình tròn, chữ nhật và tam giác. Đối với tam giác chỉ xét trường hợp biết kích thước ba cạnh, bạn đọc có thể thêm vào đây các hàm mới để tính cho hai trường hợp còn lại của tam giác (cạnh – góc – cạnh và góc – cạnh – góc) hoặc tính thể tích, diện tích cho các hình không gian...

Unit HHP;

Interface

```
Function Dtcn( a,b:real): real;  
Function Dttr( a:real): real;  
Function Dttg( a,b,c:real): real;  
Function Cvcn( a,b:real): real;  
Function Cvtr( a:real): real;  
Function Cvtg( a,b,c:real): real;
```

Implementation

```
Function Dtcn( a,b:real): real;  
Var dt:real;
```

```

Begin
    dt:=a*b;
    dtcn:=dt;
End;

Function Dttr( a:real): real;
Var dt:real;
Begin
    dt:=pi*a*a;
    dttr:=dt;
End;

Function Dttg( a,b,c:real): real;
Var p,dt:real;
Begin
    p:=(a+b+c)/2;
    dt:=sqrt(p*(p-a)*(p-b)*(p-c));
    dttg:=dt;
End;

Function Cvcn( a,b:real): real;
Var cv:real;
Begin
    cv:=(a+b)*2;
    cvcn:=cv;
End;

Function cvtr( a:real): real;
Var cv:real;
Begin
    cv:=2*pi*a;
    cvtr:=cv;
End;

Function cvtg( a,b,c:real): real;
Var cv:real;
Begin
    cv:=a+b+c;
    cvtg:=cv;
End;
End.

```

4.5 Phần hướng dẫn

Mỗi Unit khi xây dựng xong cần có phần hướng dẫn để người sử dụng không gặp phải các lỗi khi chương trình tham chiếu đến Unit, chẳng hạn trong Unit HHP khi tính diện tích chu vi tam giác sẽ có thể xảy ra trường hợp số đo các cạnh a, b, c không tạo thành tam giác. Trong trường hợp này người thiết kế Unit phải hướng dẫn người sử dụng kiểm tra điều kiện tạo nên tam giác trước khi gọi các Function của HHP.

Thông thường phần hướng dẫn của Pascal đặt trong thư mục DOC dưới dạng các tệp văn bản (phần mở rộng là TXT). Do bộ mã dùng trong Pascal không có các ký tự tiếng Việt nên phần hướng dẫn chỉ có thể viết bằng tiếng Việt không dấu.

Mặc dù có phần hướng dẫn song nhiều khi người lập trình không đọc kỹ hoặc đọc mà vẫn quên các điều kiện của bài toán do vậy cần cố gắng hạn chế những ràng buộc mà người sử dụng phải tuân theo. Để làm được điều này cần phải chú ý đến các lớp bài toán có thể tham khảo đến Unit, đồng thời phải lường trước những sai sót mà người sử dụng có thể mắc phải. Trong thực tế ngay cả người lập trình chuyên nghiệp cũng không thể nào tránh được sai lầm do vậy cần chú ý đến các thông báo lỗi khi người sử dụng đưa ra các lệnh không phù hợp.

Ví dụ 3.3 sau đây nêu cách sử dụng Unit HHP tính diện tích chu vi tam giác

```
Program tinh_dtcv;  
Uses crt,hhp;  
Var m,n,q:real;  
Begin  
  clrscr;  
  Write('Cho biet ba canh '); readln(m,n,q);  
  if (m+n>q) and (n+q>m) and (q+m>n) then  
  Begin  
    Writeln('Dien tich tam giac la ',dttg(m,n,q):5:2);  
    Writeln('Chu vi tam giac la ',cvtg(m,n,q):5:2)  
  End  
  Else  
    Writeln('So lieu da cho khong tao thanh tam giac');  
  readln;  
End.
```

Trong ví dụ trên nếu người chạy chương trình nhập vào giá trị của ba cạnh là 1, 3, 8 và trong chương trình không kiểm tra điều kiện tạo nên tam giác thông qua câu lệnh

```
if (m+n>q) and (n+q>m) and (q+m>n) then ...
```

thì sẽ dẫn tới diện tích là căn bậc hai của một số âm. Chúng ta có thể thiết kế lại Unit sao cho khi các tham số truyền cho chương trình con không phù hợp thì sẽ xuất hiện thông báo lỗi và yêu cầu nhập vào các giá trị khác. Muốn thế chúng ta phải thay thế các Function tính diện tích và chu vi tam giác thành các Procedure.

Thông thường các Unit được thiết kế chưa thể bao quát ngay mọi yêu cầu mà các bài toán đặt ra vì vậy việc chỉnh sửa, nâng cấp là điều khó tránh khỏi. Để làm việc này người thiết kế Unit nên lưu trữ các cấu trúc dữ liệu, các gợi ý và cách thức tổ chức Unit để khi cần có thể

tra cứu dễ dàng. Điều này có ý nghĩa khi chúng ta nên cung cấp cho người dùng không chỉ các TPU mà cả tệp nguồn với mục đích người sử dụng có thể tự nâng cấp khi cần.

5. Tham chiếu đến các Unit

Một chương trình có thể tham chiếu đến nhiều Unit cùng một lúc, các Unit này có thể là Unit chuẩn của Pascal hay là Unit tự tạo. Trong các Unit tự tạo người thiết kế cũng có thể đưa vào các lệnh tham chiếu đến các Unit khác. Pascal có một nguyên tắc đặt ra là *trong một chương trình các Unit không được tham chiếu vòng tròn*.

Giả sử chúng ta đã thiết kế ba Unit là U1, U2, U3 với phân tiêu đề và Interface như sau:

```
Unit U1 ;  
Interface  
Uses U2;  
....  
End.
```

```
Unit U2;  
Interface  
Uses U3;  
....  
End.
```

```
Unit U3;  
Interface  
Uses U1;  
....  
End.
```

Nếu trong một chương trình chúng ta có lệnh

```
Uses U1, U2, U3;
```

Thì sẽ dẫn đến một tình trạng là U1 tham chiếu đến U2, U2 tham chiếu đến U3 và U3 tham chiếu lại U1, trong trường hợp này Pascal sẽ thông báo lỗi: Circular Unit Reference (Unit tham chiếu vòng tròn). Đây có thể là một lỗi khó hiểu đối với người lập trình nếu như người đó không biết được bản thân các Unit mà mình sử dụng đang tham chiếu đến những Unit nào.

Điều này có thể giải thích là do Unit U1 cần phải tham chiếu đến U3 thông qua U2, nhưng U3 lại phải tham chiếu đến U1 trong khi bản thân U1 chưa hoàn thiện do đó sẽ gây lỗi.

Một vấn đề khác cũng cần phải chú ý là thứ tự tham chiếu đến các Unit, ví dụ lệnh:

```
Uses CRT, GRAPH; quy ước rằng CRT được tham chiếu trước, GRAPH tham chiếu sau.
```

Trong trường hợp các TPU và chương trình tham chiếu đến chúng có một số biến , hàm trùng tên và có một số đại lượng được khởi tạo giá trị ban đầu thì Pascal sẽ ưu tiên sử dụng các giá trị được khởi tạo cuối cùng. Chúng ta sẽ giải thích điều này qua ví dụ sau;

Ví dụ 3.4

```
Unit u1;
Interface
Var traloi:char;
Procedure P11;
Implementation
    Procedure P11;
    Begin Writeln('Day la Unit U1');
    End;
Begin
    traloi:='K';
End.

Unit u2;
Interface
Var traloi:char;
Procedure P11;
Implementation
    Procedure P11;
    Begin
    Writeln('Day la Unit U2');
    End;

Begin
traloi:='C';
End.

Program dung_unit;
Uses crt,u1,u2;
Begin
    Clrscr;
    if upcase(traloi)='C' then
    Begin
    p11;
    Writeln('Bien Traloi mang gia tri : ',traloi);
    Writeln('Thu tu tham chieu la U1, U2 ');
```

```

End
Else
Begin
p11;
Writeln('Bien tra loi mang gia tri : ',traloi);
Writeln('Thu tu tham chieu la U2, U1 ')
End;
Readln;
End.

```

Ví dụ 3.4 xây dựng hai Unit U1 và U2, cả hai đều có biến Traloi, và thủ tục P11. Thủ tục P11 chỉ làm một công việc là viết lên màn hình dòng chữ “Day la unit ...”. Trong U1 biến Traloi được gán giá trị là ‘K’ còn trong U2 biến này mang giá trị ‘C’.

Chương trình Dung_Unit có lệnh tham chiếu

```
uses crt, u1, u2;
```

Do Unit U2 được tham chiếu sau U1 nên giá trị của biến Traloi được khởi tạo cuối cùng sẽ thuộc về U2 nghĩa là Traloi = “C”. Lúc này biểu thức điều kiện *if upcase(traloi)='C'* trong chương trình Dung_unit mang giá trị True do đó trên màn hình chúng ta nhận được kết quả

Day la Unit U2

Bien Traloi mang gia tri : C

Thu tu tham chieu la U1, U2

Nếu chúng ta đảo ngược lại thứ tự tham chiếu

```
Uses crt, u2, u1;
```

thì sẽ nhận được kết quả

Day la Unit U1

Bien Traloi mang gia tri : K

Thu tu tham chieu la U2, U1

Ví dụ trên cũng chỉ ra rằng nếu trong các TPU chúng ta thiết kế có các hàm hoặc thủ tục trùng tên với nhau và trùng tên với hàm, thủ tục có trong chương trình chính, đồng thời các biến (hàm) được gán giá trị ban đầu ngay trong phần Implementation thì khi chương trình tham chiếu đến TPU nó sẽ lấy giá trị của biến (hàm) được gán cuối cùng. Nếu trong bản thân chương trình chính cũng có lệnh gán giá trị cho biến (hàm) (trùng tên với các biến (hàm) có trong TPU) thì Pascal coi đây là giá trị được gán cuối cùng và giá trị này sẽ được sử dụng trong các ứng dụng.

Nói tóm lại việc thiết kế Unit đúng chưa cho chúng ta kết quả đúng nếu chúng ta không không nắm được nguyên lý sử dụng chúng. Cách tốt nhất là tránh dùng các hàm và biến trùng tên nhau. Trong trường hợp các Unit được thiết kế phục vụ một lớp bài toán nào đó và bắt buộc phải sử dụng một số thủ tục hoặc hàm trùng tên thì cần chỉ rõ cách thức sử dụng chúng trong phần hướng dẫn sử dụng.

6. Trình tiện ích TPUMOVER

Thuật ngữ TPUMOVER được ghép nối bởi hai thành tố TPU (Unit của Pascal) và MOVER (chuyển động, dịch chuyển) vì thế có thể hiểu TPUMOVER nghĩa là dịch chuyển các Unit. Trình tiện ích TPUMOVER được lưu trữ dưới dạng một File chương trình với phần mở rộng là EXE. Nhờ TPUMOVER người sử dụng có thể tự tổ chức các Unit do mình tạo ra thành một thư viện dưới dạng File với tên File theo quy định của hệ điều hành DOS và phần mở rộng là TPL (ngầm định của Pascal). Không những thế người sử dụng còn có thể thêm, bớt các Unit của mình vào chính thư viện chuẩn TURBO.TPL của Pascal hoặc trích một Unit nào đó trong thư viện để trở thành một TPU độc lập.

Tổ chức các Unit tự tạo thành thư viện riêng của người lập trình sẽ tránh được việc các TPU nằm rải rác trong thư mục nhưng lại gặp một rắc rối là mỗi lần muốn dùng Unit ta lại phải tách chúng ra ngoài. Sở dĩ như vậy là vì Pascal chỉ tự động nạp thư viện chuẩn TURBO.TPL khi khởi động máy. Cách tốt nhất là những Unit hay được dùng đến chúng ta gộp chúng vào thư viện chuẩn TURBO.TPL. Việc đưa các Unit tự tạo vào TURBO.TPL sẽ cho phép tăng tốc độ xử lý bởi vì các Unit trong thư viện chuẩn đã được tự động nạp vào bộ nhớ Ram khi Pascal khởi động, việc tìm kiếm ở Ram bao giờ cũng nhanh hơn tìm kiếm ở bộ nhớ ngoài.

Muốn biết cú pháp của Tpumover chúng ta có thể đọc trong tệp Until.doc mục số 4, tệp này nằm trong thư mục TP\DOC hoặc từ màn hình soạn thảo chương trình của Pascal chúng ta gõ Tpumover rồi bấm tổ hợp phím Ctr – F1. Pascal đưa ra cú pháp của Tpumover như sau:

Tpumover Filename Operations

- Filename là tên thư viện chuẩn (Turbo) hoặc tên thư viện mà chúng ta muốn tạo ra
 - Operations là danh sách tùy chọn của Tpumover, bao gồm
 - +Unitname: tên Tpu muốn đưa vào thư viện (dấu cộng ở đầu)
 - Unitname: tên Tpu muốn xóa khỏi thư viện (dấu trừ ở đầu)
 - *Unitname: tên Tpu muốn tách thành Unit độc lập (dấu sao ở đầu)

Nếu chúng ta bỏ qua Operations thì Tpumover sẽ hiện lên danh sách các Unit đang có trong thư viện Filename.

Ví dụ 3.5:

Giả sử chúng ta muốn đưa thêm hai Unit U1, U2 đã thiết kế vào thư viện chuẩn Turbo. Để chạy Tpumover chúng ta có hai cách:

a. Từ màn hình nền của Window:

Chọn Start – Run

Trong cửa sổ Run gõ vào

C:\TP\BIN\Tpumover.exe Turbo +u2 +u1

Chọn OK

b. Từ dấu nhắc hệ thống C:\> của Dos:

Gõ các lệnh vào thư mục Bin

C:\>CD TP ↵


```
C:\TP>CD BIN ↵
```

Tiếp đó gõ lệnh

```
C:\TP\BIN>Tpumover Turbo +U1 +U2 ↵
```

Hai Unit U1, U2 lúc này đã được đưa vào trong thư viện chuẩn Turbo.tpl.

Để xoá Unit U2 khỏi Turbo chúng ta gõ lệnh:

```
C:\TP\BIN>Tpumover Turbo -U2 ↵
```

Muốn đưa Unit U1 thành một Unit độc lập ta gõ lệnh:

```
C:\TP\BIN>Tpumover Turbo *U1 ↵
```

Unit U1 sau khi tách ra sẽ trở thành tệp U1.TPU trong thư mục BIN nhưng bên trong thư viện chuẩn Turbo.tpl vẫn tồn tại Unit U1. Khi có lệnh tham chiếu

Uses U1;

Pascal sẽ ưu tiên tìm U1 trong Turbo, nếu không thấy thì mới tìm ở ngoài.

7. Công dụng của Unit

7.1 Làm gọn nhẹ chương trình

Vì các hàm và thủ tục cần thiết cho chương trình đều đã cất trong Unit nên kích thước chương trình sẽ giảm đáng kể.

7.2 Tăng tốc độ thực hiện chương trình

Như đã biết trong phần lập trình cơ bản, mỗi chương trình trước khi chạy đều cần phải biên dịch. Vì các Unit mà chương trình tham chiếu đến đã được biên dịch trước cho nên thời gian dịch chương trình có Unit sẽ giảm và hệ quả là tốc độ thực hiện chương trình tăng.

7.3 Cho phép viết được các chương trình lớn

Những chiếc PC đầu tiên sử dụng bộ xử lý Intel 8086, bộ xử lý này chỉ xử lý 16 bit dữ liệu. Số dương 16 bits lớn nhất khi không xét dấu là $1111\ 1111\ 1111\ 1111_2 = 65535_{10}$. Nói cách khác Intel 8086 chỉ có thể quản lý được 65535 địa chỉ (tức là 64 KB). Với những bộ nhớ có dung lượng lớn hơn 64 KB người ta phải chia chúng thành các đoạn nhỏ mỗi đoạn có kích thước 64 KB và dùng hai tham số cho một địa chỉ, tham số thứ nhất Segment là địa chỉ của đoạn trong bộ nhớ và tham số thứ hai Offset là địa chỉ của ô nhớ trong mỗi đoạn. Các máy thế hệ sau này dù xử lý 32 bit nhưng kích thước mỗi đoạn vẫn chỉ là 64 KB. Đây chính là lý do khiến cho các chương trình Pascal không thể vượt quá 64 KB. Nhờ các Unit chúng ta có thể chuyển những hàm, thủ tục, thường trình vào Unit. Trong bộ nhớ các Unit này sẽ được nạp vào các đoạn khác với đoạn chứa chương trình. Điều đó có nghĩa là kích thước chương trình có thể vượt quá 64 KB.

7.4 Bảo đảm an toàn

Sau khi đã biên dịch các TPU được lưu trữ dưới dạng mã máy, việc tham chiếu đến TPU là tham chiếu đến phần Interface, như vậy chúng ta không sợ bị thay đổi những thiết kế trong phần Implementation. Đây chính là cách mà chúng ta bảo vệ các cấu trúc dữ liệu cũng

như cấu trúc các thủ tục, hàm đã thiết kế. Việc nâng cấp thay đổi chỉ có thể thực hiện trên tệp nguồn và sau đó lại phải biên dịch trước khi có thể sử dụng.

Cũng cần phải lưu ý rằng Unit không phải chỉ hoàn toàn có lợi, tất cả các Unit trong thư viện chuẩn Turbo đều được nạp vào bộ nhớ khi Pascal khởi động và thường trú ở đó cho nên chúng chiếm dụng một không gian nhớ nhất định. Việc đưa quá nhiều Unit vào thư viện chuẩn sẽ làm giảm đi kích thước bộ nhớ mà chương trình được phép sử dụng.

8. Một số Unit chuẩn

Pascal cung cấp cho người lập trình khoảng 20 Unit chuẩn do đó việc giới thiệu toàn bộ các Unit trong cuốn sách này là điều không thể. Bạn đọc có thể tham khảo danh mục các thủ tục và hàm của từng Unit trong phụ lục ở cuối sách. Trong mục này chúng ta sẽ tìm hiểu một số Unit thông dụng nhất.

8.1 System

System là đơn vị cơ bản của Turbo Pascal trong đó cài đặt toàn bộ các thủ tục và hàm mà chúng ta đã sử dụng từ đầu chương trình, system đã được thiết kế để chạy tự động khi có một chương trình được thực hiện do đó không cần lệnh gọi USES ở đầu chương trình.

Trong UNIT SYSTEM có một số thủ tục hay dùng khi lập trình như sau:

a. Thủ tục *EXIT*:

Khi gặp EXIT trong một chương trình thì kết thúc chương trình và ra khỏi chương trình đó, nếu đặt exit trong chương trình chính thì dừng toàn bộ công việc.

b. Thủ tục *CHDIR* (tên thư mục):

Thủ tục CHDIR (change Directory) cho phép chuyển thư mục có tên viết trong dấu ngoặc đơn thành thư mục hiện thời.

c. Thủ tục *RMDIR* (tên thư mục):

Thủ tục RMDIR (remove Directory) cho phép xóa một thư mục rỗng trên ổ đĩa hiện thời.

d. Thủ tục *Rename* (<tên cũ>; <Tên mới>):

Thủ tục này đổi tên tệp ngoại vi từ tên cũ thành tên mới.

8.2 Dos

DOS là UNIT chứa các thủ tục liên quan đến hệ điều hành và quản lý tệp:

a. *SETDATE*(năm, tháng, ngày):

Thủ tục này sẽ đặt lại ngày tháng năm cho đồng hồ của máy, ví dụ: SETDATE (1998, 11, 25);

Lưu ý rằng chỉ có thể đặt lại cho các năm trong phạm vi từ năm 1980 - 2099 nếu đưa vào giá trị khác máy sẽ báo lỗi.

b. *SETTIME* (giờ, phút, giây, phần trăm giây):

Đặt lại giờ cho đồng hồ của máy.

c. *GETTIME* (giờ, phút, giây, phần trăm giây):

Cho biết giờ hệ thống, muốn viết giờ hệ thống ra màn hình chúng ta có thể viết một chương trình như sau:

```

Program Xemgio;
Var x1, x2, x3, x4: wrod;
begin
Gettime (x1, x2, x3, x4);
Write ('Bay gio là ',x1:2,' gio ', x2:2,' phut ', x3:2,' giay');
end.

```

Chương trình trên không viết phần trăm giây, xin mời các bạn sửa chương trình để có thể thông báo toàn bộ các thông số , mỗi số trên một dòng.

8.3 Crt

CRT là Unit liên quan đến các thủ tục trình bày màn hình. Crt không nằm trong thư viện chuẩn nên không được nạp tự động. Trong CRT có các thủ tục sau:

a. Gotoxy(m, n):

Chuyển con trỏ tới toạ độ cột m và dòng n. Cần lưu ý rằng $1 \leq m \leq 80$; $1 \leq n \leq 25$.

b. CLRSCR: xoá sạch màn hình đưa con trỏ về toạ độ (1,1)

c. CLREOL: xoá các ký tự từ vị trí con trỏ hiện thời đến hết dòng

d. SOUND(n): Tạo ra một âm thanh với tần số n Hz.

Để hiểu rõ hơn về Thủ tục này xin xem chương Âm thanh

e. DELAY (n): Làm chậm (hoặc kéo dài) lệnh phía trước n mili giây (1 giây = 1000 mili giây). Giá trị lớn nhất cho phép là 65535 ms (tức là giá trị số nguyên kiểu Word lớn nhất mà máy có thể xử lý)

f. NOSOUND: Tắt tiếng kêu do SOUND (n) tạo ra

Ba thủ tục sound, delay, nosound được sử dụng để tạo ra các bản nhạc, cần lưu ý rằng sau thủ tục sound và delay âm thanh được tạo ra sẽ kéo dài liên tục nếu không có nosound mặc dù trong delay ta đã chỉ định khoảng thời gian kéo dài là n ms, chỉ khi có thủ tục nosound thì âm thanh mới kéo dài đúng n miligiây

g. LOWVIDEO :làm tối màn hình.

h. HIGHVIDEO : làm sáng thêm màn hình.

i. NORMVIDEO: trả về trạng thái sáng bình thường.

j. Thuộc tính màu

Có hai thủ tục gán thuộc tính màu cho ký tự viết ra màn hình là:

TEXTCOLOR(Mã màu) : màu chữ

TEXTBACKGROUND(Mã màu): màu nền

Cả hai thủ tục phải viết trước các lệnh Write hoặc Writeln và chúng sẽ có tác dụng cho đến khi chúng ta đặt lại mã màu mới.

Tham số "mã màu" trong hai thủ tục định màu ở trên có thể tham khảo trong bảng 3.1.

Pascal sử dụng một Byte trong bộ nhớ để lưu trữ thuộc tính màu, thuộc tính màu ở đây được hiểu bao gồm: màu chữ, màu nền và khả năng nhấp nháy của chữ.

Trong byte thuộc tính bốn Bit thấp được dùng lưu trữ màu chữ, ba Bit tiếp theo lưu trữ màu nền, Bit cao nhất lưu trữ thuộc tính nhấp nháy. Như đã biết trong hệ đếm cơ số 2 sử dụng khuôn 4 Bit chúng ta có thể biểu diễn các số từ 0 đến 15, còn khuôn 3 Bit chỉ biểu diễn được các số từ 0 đến 7. Điều này có nghĩa là màu chữ có thể chọn một trong 16 giá trị (từ 0 đến 15)

còn màu nền chỉ có 8 giá trị (từ 0 đến 7). Nếu Bit thứ 8 được bật, nghĩa là cộng thêm vào thuộc tính màu giá trị 128 ($1000\ 0000_2 = 128$) thì chữ sẽ nhấp nháy.

Bảng 3.1

Mã màu	Tên hằng	Màu tiếng Việt	Mã màu	Tên hằng	Màu tiếng Việt
0	Black	Đen	4	Red	Đỏ
1	Blue	Xanh lam	5	Magenta	Tím
2	Green	Xanh lá mạ	6	Brown	Nâu
3	Cyan	Xanh lơ	7	LightGray	Xám nhạt

Mã màu	Tên màu	Màu tiếng Việt	Mã màu	Tên hằng	Màu tiếng Việt
8	DackGray	Xám sẫm	12	LightRed	Đỏ sáng
9	LightBlue	Xanh lam sáng	13	LightMagenta	Tím sáng
10	LightGreen	Xanh lá mạ sáng	14	Yellow	Vàng
11	Lightcyan	Xanh lơ sáng	15	White	Trắng

Giá trị thuộc tính màu được tính theo công thức

Thuộc tính màu = Mã màu nền*16 + Mã màu chữ

Ví dụ: chọn chữ màu trắng mã màu là 15, nền màu xám sáng mã màu nền là 7 thì thuộc tính màu sẽ bằng

$$7 * 16 + 15 = 127$$

Trong thực tế không cần phải cộng thêm vào mã màu chữ số 128 mà chỉ cần cho mã màu chữ lớn hơn 15 thì chữ đã nhấp nháy rồi. Pascal có sẵn một biến là TextAttr kiểu Byte để lưu trữ thuộc tính màu, nếu thuộc tính màu nhỏ hơn 128 thì

Mã màu chữ = TextAttr Mod 16

Mã màu nền = TextAttr Div 16

Nếu thuộc tính màu lớn hơn 128 như cách chọn màu sau đây:

Textcolor(16);

TextbackGround(7);

Thì thuộc tính màu là 240 và mã màu chữ thực tế sẽ là 0, còn mã màu nền là 7. Sở dĩ như vậy là vì Pascal đã tự động cộng thêm 128 vào mã màu, hay nói cách khác

$$\text{Mã màu chữ} = (240 - 128) \text{ Mod } 16 = 0$$

$$\text{Mã màu nền} = (240 - 128) \text{ div } 16 = 7$$

Ví dụ 3.6 viết lên màn hình một chuỗi ký tự "*", chương trình có sử dụng 3 biến : biến Ttm lưu trữ thuộc tính màu, biến Mc là mã màu chữ, Mn là mã màu nền. Người sử dụng có thể chạy chương trình với các tham số màu khác nhau và chương trình sẽ thông báo thuộc tính màu và mã màu thực tế .

Ví dụ 3.6

```
Program Ma_mau;
Uses crt;
Var Ttm,Mc,Mn:byte;
Begin
  clrscr;
  textcolor(26);
  textbackground(7);
  writeln('*****');
  ttm:=textattr;
  mc:= textattr mod 16;
  writeln('Thuoc tinh mau ',ttm);
  Writeln('Mau chu : ', mc);
  if ttm>127 then
    mn:=(ttm-128) div 16
  else
    mn:= ttm div 16;
  Writeln('Mau nen : ', mn);
  repeat until keypressed;
End.
```

j. Hàm KEYPRESSED:

Hàm Keypressed kiểm tra xem trong quá trình làm việc có phím nào được bấm hay không, nếu có thì hàm mang giá trị True, còn nếu không thì mang giá trị False.

Hàm Keypressed thường được dùng trong các câu lệnh để kiểm tra điều kiện, ví dụ:

If keypressed then.... nếu có phím được bấm thì làm công việc...

Hoặc: Repeat Until Keypressed;

lặp lại công việc cho đến khi có phím được bấm

Cũng có thể dùng câu lệnh Repeat until keypressed để dừng chương trình, khi muốn tiếp tục thì chỉ cần bấm một phím bất kỳ. (Cần phân biệt với lệnh Readln, khi muốn tiếp tục thì phải bấm Enter).

k. Hàm READKEY:

Hàm Readkey nhận diện phím được bấm, giá trị mà nó nhận về là một ký tự. Có hai khả năng xảy ra đối với hàm Readkey:

*. Nếu vùng đệm bàn phím đang rỗng, nghĩa là các ký tự đưa vào đã được xử lý hết thì Readkey sẽ dừng chương trình chờ cho đến khi một phím được bấm, phím được bấm một mặt được đưa vào xử lý, mặt khác được lưu trong hàm Readkey.

*. Nếu vùng đệm bàn phím không rỗng, nghĩa là đã có các ký tự được bấm đang nằm chờ thì Readkey sẽ lấy ra ký tự đầu tiên trong vùng đệm và vùng đệm bàn phím sẽ giảm đi một ký tự.

1. Hàm GETKEY:

Hàm Getkey hoạt động tương tự như hàm Readkey nhưng giá trị mà nó nhận là một số dương đối với các phím thông thường, nếu là các phím có mã quét mở thì giá trị Getkey nhận gồm hai số, số đầu là số 0 còn số thứ hai là mã quét mở của phím, ví dụ:

Nếu bấm chữ J thì Getkey cho số 75, còn nếu bấm phím dịch chuyển con trỏ (mũi tên) sang trái thì Getkey cho 0 - 75.

Chúng ta cần có sự phân biệt rõ ràng phím trên bàn phím và ký tự trong bảng mã. Bàn phím kiểu Mỹ chỉ có 101 phím, còn bảng mã có 256 ký tự. Chỉ có một phần nhỏ ký tự trong bảng mã nhìn thấy được trên bàn phím (bao gồm các chữ cái, chữ số, toán tử số học, toán tử so sánh và một vài ký tự khác), các ký tự còn lại muốn cho hiện trên màn hình phải dùng tổ hợp 2 phím Alt - Số thứ tự.

Dưới đây là bảng đối chiếu một số phím và mã phím thường dùng

Bảng 3.2

Phím	Mã
BackSpace	8
Tab	9
Enter	13
Esc	27
Shift - Tab	0-15
↑	0-72
←	0-75
→	0-77
↓	0-80
Insert	0-82
Delete	0-83

Phím	Mã
Home	0-71
End	0-79
PageUp	0-73
PageDown	0-81
Ctrl - End	0-117
Ctrl - Home	0-119
Ctrl - ←	0-115
Ctrl - →	0-116
Ctrl - PageUp	0-132
Ctrl - PageDown	0-118

Để có thể nhận diện các phím có mã quét mở rộng chúng ta có thể dùng hai biến kiểu Char, một biến chứa ký tự có mã bằng 0, còn biến kia chứa ký tự của phần mã quét mở, nói cách khác cần phải gọi hàm Readkey hai lần mới bắt được phím có mã quét mở.

Ví dụ 3.7:

```
Var C1,C2:char;  
Begin  
C1:=Readkey; If C1=chr(0) then C2:=Readkey;  
End.
```

Chúng ta cũng có thể dùng biến kiểu Byte để bắt các phím có mã quét mở. Trong thực tế vì đã biết số thứ nhất trong mã quét mở bằng không nên chỉ cần tìm số thứ hai của mã quét mở như ví dụ sau:

```

Var mq:byte;
Begin
Mq=readkey;
If ord(mq) = 0 then mq := ord(readkey);
End.

```

Nếu biến mq có giá trị 75 thì phím được bấm không phải là chữ J mà là phím ← bởi vì mã quét ở đây là 0 - 75.

Chú ý:

Khi xây dựng chương trình có việc đọc mã quét, cần đặc biệt chú ý trong vùng đệm bàn phím đã có mã phím nằm chờ hay không, nếu có thì phải làm sạch vùng đệm bàn phím trước khi đọc mã. Việc làm sạch vùng đệm bàn phím có thể thực hiện bằng câu lệnh lặp sau:

```

While Keypressed do mq:= readkey;
Biến mq phải được khai báo trước và thuộc kiểu Char.

```

Ví dụ 3.8:

Viết chương trình hiện mã quét của các phím, nếu là phím có mã quét mở rộng thì hiện cả hai số dưới dạng 0_00.

```

Program Batphim;
Uses crt;
Var k,k1,k2:byte;
Function bp(Var mq1,mq2:byte):byte;
Begin
mq1:=ord(readkey);
if mq1=0 then
begin
mq2 := ord(readkey);
bp := mq2;
end
else
bp:=mq1;
end;

Begin
Clrscr;
Writeln('Xin moi bam 1 phim, bam phim END de ket thuc ');
Repeat
K := bp(k1,k2);
if k1<>0 then
Writeln('Ma phim ban vua bam = ',k1)

```

```

else
  Writeln('Ma phim ban vua bam = ',k1,'_',k2);
Until k=79;
End.

```

Chương trình đặt tên là BATPHIM có một chương trình con là hàm BP, hàm BP có hai tham biến dùng để lưu các mã quét mq1, mq2.

Lời gọi trong chương trình mẹ:

K:=bp(k1,k2); sẽ truyền cho hàm BP hai tham số thực k1,k2, các tham số này sẽ chứa mã quét mq1, mq2 khi quay về chương trình mẹ. Biến K dùng để lưu mã quét mq2 của phím END khi muốn kết thúc chương trình .

Chạy chương trình trên, khi xuất hiện thông báo:

" Xin moi bam 1 phim, bam phim END de ket thuc",

chúng ta có thể bấm một phím nào đó trên bàn phím và sẽ nhận được thông báo mã phím vừa bấm.

Chú ý: Các phím chức năng phụ trên bàn phím không thấy được mã quét bao gồm:

CAPSLOCK, SHIFT, CTRL, ALT, INS, DEL, NUMLOCK, PRINT SCREEN, SCROLL LOCK, PAUSE, F11, F12

Việc bắt phím thường được sử dụng khi thiết kế các Menu (thực đơn) trong trường hợp bài toán cần có một vài phương án lựa chọn công việc. Dưới đây là ví dụ thiết kế thực đơn với bốn tùy chọn

Tamgiac Chunhat Tron Ketthuc

Để chuyển con trỏ đến một chức năng nào đó ta bấm các phím ← hoặc →, để chọn chức năng bấm phím Enter, kết thúc công việc bấm End. Chương trình có sử dụng Unit HHP đã thiết kế.

Ví dụ 3.9

```

Program Menu;
Uses crt,hhp;
label h1,h2,h3;
Var c1,c2,c3,c4:string[20];
    l1,l2,l3,l4,mq1,mq2:byte; chon:char;
Procedure tr; { thu tuc tinh hinh tron }
    Var a,dt,cv:real;
Begin
  clrscr;
  write('Hay cho biet ban kinh hinh tron '); Readln(a);
  dt:=pi*a*a; cv:=2*pi*a;
  writeln('Dien tich hinh la : ',dt:12:4);
  writeln('Chu vi hinh la : ',cv:12:4);

```



```
writeln('Bam Enter de quay ve thuc don ');
repeat until keypressed;
End;
```

```
Procedure cn; { thu tuc tinh hinh chu nhat }
Var a,b,dt,cv:real;
Begin
  clrscr;
  write('Hay cho biet hai canh a, b '); Readln(a,b);
  dt:=a*b; cv:=(a+b)*2;
  writeln('Dien tich hinh la : ',dt:12:4);
  writeln('Chu vi hinh la : ',cv:12:4);
  writeln('Bam Enter de quay ve thuc don ');
  repeat until keypressed;
End;
```

```
Procedure tg; { thu tuc tinh hinh tam giac }
Var a,b,c,p,dt,cv:real;
Begin
  clrscr;
  write('Hay cho biet ba canh a,b,c '); Readln(a,b,c);
  if (a+b>c) and (b+c>a) and (c+a>b) then
  Begin
    p:=(a+b+c)/2;
    dt:=sqrt(p*(p-a)*(p-b)*(p-c));
    cv:=a+b+c;
    writeln('Dien tich hinh la : ',dt:12:4);
    writeln('Chu vi hinh la : ',cv:12:4);
  end
  else
    writeln('Cac kich thuc 'a:6:2,'b:6:2,'c:6:2,' khong tao thanh tam giac');
    writeln('Bam Enter de quay ve thuc don ');
  repeat until keypressed;
End;
```

```
Procedure w1; { thiet ke chuc nang thu nhat }
Begin
  textbackground(5);
  textcolor(10);
  window(1,2,11+2,2); write(c1);
```

```
    gotoxy(1,1);  
End;
```

```
Procedure w2; { thiet ke chuc nang thu hai }
```

```
Begin  
    textbackground(5);  
    textcolor(10);  
    window(11+2,2,11+12+3,2); write(c2);  
    gotoxy(1,1);  
End;
```

```
Procedure w3; { thiet ke chuc nang thu ba }
```

```
Begin  
    textbackground(5);  
    textcolor(10);  
    window(11+12+3,2,11+12+13+3,2); write(c3);  
    gotoxy(1,1);  
End;
```

```
Procedure w4; { thiet ke chuc nang ket thuc }
```

```
Begin  
    textbackground(5);  
    textcolor(10);  
    window(11+12+13+4,2,11+12+13+14+8,2); write(c4);  
    gotoxy(1,1);  
End;
```

```
BEGIN      {Than chuong trinh chinh}
```

```
    Clrscr;  
    c1:='Tamgiac'; l1:=length(c1);  
    c2:='Chunhat'; l2:=length(c2);  
    c3:='Tron'; l3:=length(c3);  
    c4:='Ketthuc'; l4:=length(c4);  
    clrscr;  
    textcolor(red); textbackground(green);  
    write('Bam ->, <- chuyen thuc don | Bam Enter de chon | Bam End ket thuc ');  
h1: w1;w2;w3;w4;w1;  
    while keypressed do chon:=readkey; {lam sach vung dem ban phim}  
    chon:=readkey; {tinh tam giac}  
    if ord(chon)=13 then
```

```

begin
    textcolor(blue); textbackground(14);
    window(1,4,80,25);
    tg;
    clrscr;
    goto h1
end
else
begin
    if ord(chon)=0 then chon:=readkey;
    if ord(chon)=79 then halt; {bam END de ket thuc}
    if ord(chon)=77 then w2;
end;

```

```

h2: w1;w2;w3;w4; w2;
while keypressed do
chon:=readkey;    {lam sach vung dem ban phim}
chon:=readkey;    {tinh chu nhat}
if ord(chon)=13 then
begin
    textcolor(blue); textbackground(14);
    window(1,4,80,25);
    cn;
    clrscr;
    goto h2;
end
else
Begin
    if ord(chon)=0 then
    begin
        chon:=readkey;
        if ord(chon)=79 then halt; {bam END de ket thuc}
        if ord(chon)=77 then w3; {chuyen sang tinh hinh tron}
        if ord(chon)=75 then goto h1; {quay ve tinh tam giac}
    end;
end;

```

```

h3: w1;w2;w3;w4;w3;
while keypressed do chon:=readkey;    {lam sach vung dem ban phim}
    chon:=readkey;    {tinh hinh tron}

```

```

if ord(chon)=13 then
  begin
    textcolor(blue); textbackground(14);
    window(1,4,80,25);
    tr;
    clrscr;
    goto h3;
  end
else
  Begin
  if ord(chon)=0 then
    begin
      chon:=readkey;
      if ord(chon)=79 then halt;    {bam END de ket thuc}
      if ord(chon)=77 then w4;    {sang chuc nang ket thuc }
      if ord(chon)=75 then goto h2; {quay ve tinh chu nhat}
    end;
  end;
  while keypressed do chon:=readkey;    {lam sach vung dem ban phim}
  chon:=readkey;
  if ord(chon)=13 then halt
  else
    if ord(chon)=0 then chon:=readkey;
    if ord(chon)=79 then halt;
    if ord(chon)=75 then goto h3;
  END.

```

8.4 Printer

Đơn vị chuẩn Printer có một phần khai báo máy in phục vụ việc in ấn dữ liệu ra giấy, máy in được cài đặt tên là LST.

Khi cần in ấn ta phải có lời gọi USES PRINTER; sau đó trong các lệnh Write hoặc writeln phải đưa tên máy in LST vào trước các tên biến cần in, ví dụ:

```
Writeln (1st, 'Ket qua tinh toan:', x1);
```

Dòng lệnh trên sẽ in ra giấy dòng chữ: "Ket qua tinh toan:" và giá trị biến của x1.

Bài tập ứng dụng chương 3

1. Lập chương trình tạo thực đơn hai mức, mức 1 theo chiều ngang , mức 2 theo chiều dọc theo mẫu dưới đây:

HINH PHANG	HINH KHONG GIAN
Tam giac	Hinh non
Tron	Hinh chop
Ket thuc	Ket thuc

Yêu cầu: khi con trỏ ở một trong hai chức năng HINH PHANG, HINH KHONG GIAN nếu bấm Enter thì xuất hiện thực đơn dọc phía dưới, còn nếu bấm phím dịch chuyển con trỏ thì sẽ chuyển con trỏ sang phải hoặc trái. Khi thực đơn dọc xuất hiện sẽ có con trỏ để dịch chuyển lên xuống, bấm Enter để thực hiện một trong các việc tính diện tích, chu vi (đối với hình phẳng) hoặc diện tích toàn phần và thể tích (đối với hình không gian).

2. Cho hai ma trận chữ nhật $A(m,n)$, $B(k,q)$ xây dựng chương trình kiểm tra xem có thể nhân hai ma trận hay không? Xây dựng một Unit tính tích hai ma trận.

3. Xây dựng một Unit với các thủ tục SXT (sắp xếp tăng) và SXG (sắp xếp giảm) để sắp xếp một dãy số.

4. Xây dựng một Unit lấy tên là HAMSOHOC trong đó có các hàm tính căn bậc n , lũy thừa bậc n của một số ($n > 2$).

5. Xây dựng một Unit nhằm đổi tên màu tiếng Anh sang tiếng Việt ví dụ RED - ĐỎ, BLUE - XANH DA TRỜI...

6. Xây dựng chương trình cho một dòng chữ chạy trên màn hình. Khi dòng chữ đang chạy có thể đổi hướng chạy bằng cách bấm các phím dịch chuyển con trỏ

7. Phát triển bài tập 6 bằng cách bổ xung tính năng tăng hoặc giảm tốc độ chạy khi bấm các phím PageUp hoặc PageDown

Chương IV

Con trỏ và cấu trúc động

Chương này đòi hỏi các kiến thức của môn Cấu trúc dữ liệu và giải thuật, đặc biệt là kiến thức về dữ liệu kiểu Cây. Do cách thức bố trí trong kế hoạch đào tạo môn này lại học song song với môn Lập trình nâng cao nên sẽ có một vài khó khăn khi trình bày cũng như khi nghe giảng. Trong chương này bạn đọc cần chú ý các vấn đề sau:

- Thế nào là kiểu dữ liệu con trỏ
- Sự khác nhau giữa kiểu dữ liệu con trỏ và biến con trỏ
- Sự phân vùng bộ nhớ cho biến con trỏ
- Cách thức mà hệ thống cấp phát bộ nhớ khi chương trình đang làm việc
- Thu hồi bộ nhớ dành cho từng biến và thu hồi hàng loạt
- Cây và cây nhị phân
- Bộ nhớ kiểu LIFO và FIFO và ứng dụng trong thiết kế cây nhị phân
- Con trỏ mảng và mảng con trỏ

1. Khái niệm

Khi khai báo một biến, dù là biến đơn hay biến thuộc kiểu dữ liệu có cấu trúc mặc nhiên chúng ta đã quy định độ lớn vùng nhớ dành cho biến.

Ví dụ

a: Real; biến a cần 6 byte

b: array[1..100] of Integer; biến mảng b cần 200 byte.

Việc khai báo như trên thường là phỏng đoán dung lượng cần thiết chứ không thật chính xác. Để tránh lỗi chúng ta thường khai báo dôi ra gây nên lãng phí bộ nhớ. Việc xác định địa chỉ lưu trữ biến và cấp phát bộ nhớ được thực hiện khi biên dịch, nghĩa là các địa chỉ này cũng như dung lượng bộ nhớ cần cấp phát đã được cố định trước khi thực hiện các thao tác khác. Các đại lượng này không thay đổi trong suốt quá trình thực hiện chương trình, nói cách khác đây là các đại lượng tĩnh.

Để tiết kiệm bộ nhớ, ngay khi chương trình đang làm việc người lập trình có thể yêu cầu cấp phát bộ nhớ cho các biến, điều này được gọi là *cấp phát bộ nhớ động*. Cấp phát bộ nhớ động được thực hiện thông qua *biến con trỏ*. Muốn có biến con trỏ chúng ta phải định nghĩa *kiểu con trỏ* trước.

2. Kiểu dữ liệu con trỏ - biến con trỏ

2.1 Con trỏ có định kiểu

Kiểu con trỏ là một kiểu dữ liệu đặc biệt dùng để biểu diễn các địa chỉ. Kiểu con trỏ do người lập trình định nghĩa theo cú pháp sau:

Type

Tên kiểu con trỏ = ^Kiểu dữ liệu;

Tên kiểu con trỏ tuân theo quy định đặt tên của Pascal, *Kiểu dữ liệu* của kiểu con trỏ là các kiểu dữ liệu đã định nghĩa trước trong pascal. Để một kiểu con trỏ có thể đại diện cho một biến nào đó thì *Kiểu dữ liệu* viết sau ký tự ^ sẽ phải giống như kiểu dữ liệu của biến đó, nói cách khác hai kiểu dữ liệu phải tương thích.

Ví dụ 4.1 khai báo kiểu con trỏ:

Type

Chu = string[20]; CT1 = ^Byte; CT2 = ^chu; CT3 = ^Nguoi;

Nguoi = record

Hoten:string[20];

Namsinh: 1900..2100;

End;

Ví dụ 4.1 định nghĩa ba kiểu con trỏ, riêng kiểu CT3 cách định nghĩa có vẻ hơi ngược là định nghĩa kiểu con trỏ trước, định nghĩa kiểu dữ liệu sau. Thật ra chúng ta cứ nên theo thói quen là định nghĩa kiểu dữ liệu trước rồi định nghĩa kiểu con trỏ sau, cách định nghĩa CT3 chẳng qua là muốn giới thiệu để chúng ta biết rằng Pascal cho phép làm ngược lại, tuy nhiên cần nhớ rằng nếu định nghĩa kiểu con trỏ trước thì ngay trong phần Type phải định nghĩa kiểu dữ liệu (không nhất thiết phải liền ngay sau định nghĩa kiểu con trỏ).

Cần chú ý rằng Pascal chỉ cho phép đưa trực tiếp vào định nghĩa kiểu con trỏ các kiểu dữ liệu đơn giản sau: **số nguyên, số thực, ký tự**. Các kiểu dữ liệu có cấu trúc muốn đưa vào con trỏ thì phải thông qua một tên kiểu khai báo trong phần Type

Cách định nghĩa hai kiểu con trỏ Hoten và Ds sau là sai:

Type

```
Hoten = ^String[20];
```

```
Ds = ^Array[1..10] of Byte;
```

Muốn sử dụng kiểu chuỗi và mảng cho kiểu con trỏ chúng ta phải định nghĩa như sau:

Type

```
S1 = string[20];
```

```
Hoten = ^S1;
```

```
a = array[1..10] of byte;
```

```
Ds = ^a;
```

2.2 Biến con trỏ

Biến con trỏ cũng như biến mảng, biến kiểu bản ghi hay kiểu tập hợp có thể khai báo thông qua kiểu con trỏ hoặc khai báo trực tiếp. Biến con trỏ có định kiểu sẽ trỏ đến một kiểu dữ liệu cụ thể.

Để thuận tiện từ nay chúng ta dùng thuật ngữ "**Con trỏ**" thay cho thuật ngữ "**Biến con trỏ**".

Ví dụ 4.2

Var

```
So: ^Integer;
```

```
Sinhvien: Ct3;
```

```
Hoten: Ct2;
```

```
Thutu, Mahoso: ^Word;
```

Trong ví dụ 4.2 chúng ta đã khai báo ba con trỏ So, Thutu, Mahoso theo kiểu trực tiếp, hai con trỏ Sinhvien và Hoten khai báo thông qua kiểu đã định nghĩa trong ví dụ 4.1. Con trỏ So trỏ tới kiểu dữ liệu số nguyên, con trỏ Sinhvien trỏ tới kiểu dữ liệu bản ghi còn con trỏ Hoten trỏ tới kiểu dữ liệu chuỗi.

Địa chỉ của các biến động và biến tĩnh sẽ được Pascal lưu trữ vào biến con trỏ điều này có nghĩa là *biến con trỏ không dùng để lưu trữ các giá trị của biến mà là địa chỉ của biến*. Dù kích thước vùng dữ liệu mà các biến con trỏ trỏ tới khác nhau thế nào thì kích thước của biến con trỏ cũng vẫn là 4 byte.

Các hàm và thủ tục xử lý biến con trỏ được Pascal lưu trữ trong Unit System.

Quy ước: Các biến con trỏ gọi là tương thích nếu chúng trỏ tới cùng một kiểu dữ liệu

2.3 Con trỏ không định kiểu

Con trỏ không định kiểu là kiểu con trỏ không quan tâm đến kiểu dữ liệu mà nó trỏ tới. Pascal dùng tên chuẩn Pointer để khai báo kiểu này.

Var

Tên biến:Pointer;

Con trỏ không định kiểu được coi là tương thích với mọi kiểu con trỏ.

Chú ý:

** Về bản chất tất cả con trỏ đều chứa địa chỉ nên chúng không có gì khác nhau song để tránh nhầm lẫn trong các quá trình xử lý Pascal chỉ coi các con trỏ cùng trỏ tới một kiểu dữ liệu là tương thích với nhau.*

2.4 Địa chỉ của một đối tượng

Đối tượng mà chúng ta đề cập trong mục này có thể là biến, hàm hay thủ tục. Khi biên dịch chương trình mỗi đối tượng được cấp phát một vùng nhớ, vùng nhớ này bao gồm một số ô nhớ liền kề nhau.

Địa chỉ một đối tượng trong bộ nhớ được xác định bởi địa chỉ của ô nhớ đầu tiên mà hệ thống dành cho đối tượng đó.

Bộ nhớ của các máy PC hiện nay là rất lớn và chúng được chia thành nhiều đoạn, mỗi đoạn có 65536 ô nhớ (2^{16} ô). Ô đầu tiên của mỗi đoạn có địa chỉ là 0 do đó ô cuối cùng có địa chỉ là 65535. Như vậy, để biết địa chỉ một ô nhớ cần biết ô nhớ đó thuộc đoạn nào và đó là ô nhớ số bao nhiêu trong đoạn đó.

Địa chỉ đoạn gọi là Segment và địa chỉ tương đối của ô nhớ trong đoạn gọi là Offset, mỗi giá trị này Pascal dùng 2 byte để lưu trữ nên một địa chỉ cần 4 byte, 2 byte thấp cho Offset và 2 byte cao cho segment.

Nếu ghi địa chỉ bằng các số nhị phân thì chúng ta phải dùng 32 chữ số 0 và 1 điều này khá là phiền phức do vậy người ta dùng hệ đếm cơ số 16. Cách ghi địa chỉ ô nhớ được quy ước như sau: địa chỉ đoạn viết trước, vị trí của ô trong đoạn viết sau, ký hiệu \$ được thêm vào trước các giá trị số để thể hiện rằng các số viết trong hệ 16.

Ví dụ: \$0101:\$FFFF

Ví dụ trên cho ta địa chỉ ô nhớ cuối cùng (ô thứ $ffff_{16} = 65535_{10}$) thuộc đoạn 257.

3. Các thủ tục và hàm tác động trên con trỏ

3.1 Gán giá trị ban đầu

Giả sử ct là một biến con trỏ đã được định nghĩa, để đảm bảo rằng ct chưa trỏ đến bất kỳ một đối tượng nào, nghĩa là ct là một con trỏ rỗng chúng ta phải gán cho ct giá trị NIL

ct := Nil;

3.2 Gán địa chỉ của một đối tượng cho con trỏ

Giả sử ct là một con trỏ và x là một đối tượng (biến, hàm, thủ tục), có ba cách gán địa chỉ của đối tượng x cho con trỏ ct:

a. ct := @x;

Trong phép gán trên toán tử @ tác động trên đối tượng x sẽ gán vào con trỏ ct địa chỉ kiểu Pointer của đối tượng đó.

b. **ct := Addr(x);**

Hàm Addr() cho địa chỉ của đối tượng x, địa chỉ này thuộc kiểu Pointer

c. **ct := Ptr(segment,offset) ;**

Hàm Ptr trong phép gán trên đòi hỏi các tham số segment và offset phải là giá trị kiểu Word viết trong hệ 16, ví dụ:

ct := Ptr(\$B800, \$0000); đưa con trỏ trỏ tới ô nhớ của vùng Video Ram

Nhận xét:

Hai phép gán @ và Addr() cùng trả về địa chỉ kiểu pointer nên chúng là tương đương.

3.3 Phép gán giữa hai con trỏ

Hai con trỏ tương thích (cùng kiểu) có thể gán giá trị cho nhau, khi đó chúng cùng trỏ tới một địa chỉ.

Ví dụ 4.3

Var

ct1: ^Float;

ct2: ^Byte;

ct3: Pointer;

x: string;

Ví dụ trên khai báo ba con trỏ thuộc ba kiểu khác nhau, ct1 là con trỏ thực, ct2 là con trỏ nguyên và ct3 là con trỏ không định kiểu, x là biến chuỗi. Khi đó các phép gán:

ct3:=@x;

ct2 := ct3;

là hợp lệ vì ct2 và ct3 là tương thích, chúng cùng trỏ đến địa chỉ của biến x.

Còn phép gán

ct1 := ct2;

là không hợp lệ vì hai con trỏ không tương thích.

3.4 Phép so sánh hai con trỏ

Chỉ tồn tại phép so sánh = (bằng nhau) và <> (khác nhau) giữa hai con trỏ nếu chúng tương thích. Kết quả so sánh là một giá trị Boolean nghĩa là True hoặc False.

*Hai con trỏ **tương thích** gọi là bằng nhau nếu chúng cùng trỏ tới một đối tượng, ngược lại gọi là khác nhau.*

Ví dụ 4.4

Program contro;

Uses crt;

Var

x,y:real; z:string;

ct1: ^integer; ct2: ^byte; ct3:pointer; ct4,ct5: ^real;

```

Begin
clrscr;
z:='Ha noi'; x:=5; y:=-123.45;
ct1:=@x; ct2:=@z; ct3:=@z; ct4:=@z; ct5:=@y;
Writeln(ct1=ct2);    { không tương thích, máy sẽ báo lỗi}
  Writeln(ct1=ct3); { false }
  Writeln(ct2=ct3); { true }
  writeln(ct4=ct5); { false }
readln;
end.

```

Ví dụ 4.4 khai báo năm con trỏ, ct1 và ct2 trỏ tới các kiểu nguyên khác nhau, ct3 là con trỏ không kiểu tức là tương thích với mọi con trỏ khác, hai con trỏ ct4 và ct5 là cùng kiểu. Các phép so sánh trong thân chương trình cho thấy một số điều cần chú ý :

- a. Phép so sánh ct1=ct2 là không hợp lệ vì hai con trỏ không tương thích
- b. Phép so sánh ct1 = ct3 cho kết quả False vì hai con trỏ tương thích nhưng trỏ tới các địa chỉ khác nhau
- c. Phép so sánh ct2 = ct3 cho kết quả True vì hai con trỏ là tương thích và cùng trỏ tới một địa chỉ.
- d. Phép so sánh ct4 = ct5 cho kết quả False vì hai con trỏ cùng kiểu nhưng trỏ tới các địa chỉ khác nhau.

4. Truy nhập dữ liệu

Khi con trỏ ct đang trỏ tới một vùng dữ liệu nào đó Pascal cho phép dùng ký hiệu ct[^] như là một biến để truy nhập vào vùng dữ liệu đó. Biến ct[^] mang trong nó dữ liệu của vùng mà con trỏ ct đang trỏ tới.

Như vậy chúng ta có thể truy nhập tới một biến, hàm hay thủ tục mà không cần biết tên các đối tượng này miễn là biết con trỏ đang trỏ vào chúng.

Ví dụ 4.5

```

Program control1;
Uses crt;
Type z1=string[3];
Var
  z:string; ct:^z1; i:byte;

```

```

Begin
clrscr;
z:='Ha noi'; ct:=@z;
writeln(ct^);

```

```

for i := 1 to length(z) do write(uppercase(ct^[i]));
readln;
end.

```

Chạy chương trình ta nhận được kết quả:

```

Ha noi
HA NOI

```

Điều này cho thấy rằng mọi xử lý trên biến z đều có thể xử lý trên biến ct[^] bởi vì biến con trỏ ct đang trỏ vào z.

Đến đây cần có sự phân biệt chính xác về biến con trỏ CT và biến CT[^]. Biến con trỏ CT mang trong nó địa chỉ của đối tượng mà nó trỏ tới, còn biến CT[^] lại chứa đựng dữ liệu trong vùng nhớ mà con trỏ CT đang trỏ tới.

Với con trỏ có kiểu tất cả các thao tác trên biến, hàm hay thủ tục mà con trỏ đang trỏ tới có thể thay thế bởi thao tác trên biến ct[^]. Kiểu của biến ct[^] chính là kiểu đã khai báo cho con trỏ ct chứ không phải là kiểu của đối tượng mà biến ct[^] đại diện.

Về điều này cần có một số giải thích cụ thể qua ví dụ sau:

Ví dụ 4.6

```

Program contro;
uses crt;
Type z1=string[3];
Var x,y:real; z:string;
    ct1:^byte; ct2:^integer; ct3:pointer; i:char;
    ct4:^real; ct5:^word; ct6:^z1; ct7:^longint;
Begin
  clrscr;
  z:='H';
  ct1:=@z; ct2:=@z; ct5:=@z; ct6:=@z; ct7:=@z;
  writeln(ct1^); writeln(ct6^); writeln(ct2^); writeln(ct5^); writeln(ct7^);
  readln;
End.

```

Chạy chương trình trên chúng ta nhận được kết quả:

```

1
H
18433
18433
18433

```

Thay lệnh gán z:='H'; bằng lệnh gán z:='Ha noi Viet nam': thì kết quả chạy chương trình sẽ là

```

15

```

Ha noi Viet nam

18447

18447

543246351

Trong ví dụ 4.6 tất cả con trỏ đều được gán địa chỉ của biến z, nhưng vì các con trỏ đại diện cho các kiểu dữ liệu khác nhau nên kết quả mà biến con trỏ trả về cũng khác nhau.

Biến $ct1^{\wedge}$ thuộc kiểu Byte nên nó cho ta dữ liệu trong Byte đầu tiên của vùng nhớ chứa z, đó chính là ô nhớ chứa độ dài của chuỗi z.

Biến $ct6^{\wedge}$ thuộc kiểu z1 tức là kiểu chuỗi nên kết quả mà nó trả về chính là chuỗi z (không phụ thuộc vào $CT6^{\wedge}$ khai báo dài bao nhiêu).

Các biến còn lại thuộc kiểu số nên kết quả trả về cũng là số, với trường hợp ký tự H các biến con trỏ $ct2^{\wedge}$, $ct5^{\wedge}$, $ct7^{\wedge}$ cho ta giá trị 18433. Nếu ký tự gán cho z là A thì giá trị trả về là 16641, là B thì giá trị này tăng thêm 256...

Với chuỗi "Ha noi Viet nam" sau H là 14 ký tự nữa cho nên giá trị trả về là

$18433 + 14 = 18447$.

Đến đây chúng ta có thể rút ra một số kết luận:

- *. Địa chỉ của một đối tượng có thể gán cho bất kỳ con trỏ nào.
- *. Kết quả mà biến ct^{\wedge} trả về thuộc kiểu dữ liệu của con trỏ chứ không thuộc kiểu dữ liệu của đối tượng.
- *. Muốn sử dụng biến ct^{\wedge} như một biến thông thường thay thế cho đối tượng thì biến con trỏ và đối tượng phải tương thích về kiểu (cùng một kiểu dữ liệu).
- *. Với con trỏ không định kiểu (Pointer) chúng ta không thể coi chúng là tương đương với các biến định kiểu thông thường, điều này có nghĩa là không thể sử dụng các thủ tục Write, Read hoặc phép gán cho biến ct^{\wedge} nếu ct là Pointer.

Ví dụ: trở lại ví dụ 4.5 các cặp thao tác mà chúng ta thực hiện sau đây là tương đương:

Thao tác trên biến

$z := 'Ha noi';$

$x := 5;$

$y := -123.45;$

Thao tác trên con trỏ

$ct2^{\wedge} := 'Ha noi';$

$ct1^{\wedge} := 5;$

$ct5^{\wedge} := -123.45;$

5. Mảng con trỏ và con trỏ kiểu mảng

Con trỏ là một kiểu dữ liệu cho nên biến con trỏ có thể là các thành phần của mảng, ngược lại mảng là một kiểu dữ liệu có cấu trúc nên con trỏ cũng có thể trỏ tới các biến mảng.

5.1 Con trỏ kiểu mảng

Khai báo:

Type m = array[1..5] of byte;

Var

ct1: m ;

Với cách khai báo trên đây ct1 là biến con trỏ kiểu mảng, khi đó biến ct1[^] sẽ gồm 5 phần tử, mỗi phần tử là một số kiểu Byte. Việc truy nhập vào biến ct1[^] thực chất là truy nhập vào từng phần tử, ví dụ:

```
Read(ct1^[i]); hoặc Write(ct1^[i]); với 1<=i<=5.
```

5.2 Mảng các con trỏ

Khai báo:

```
Var
    ct: array[1..10] of ^string;
    s1,s2: String;
Begin
    s1:='Ha noi Viet nam';
    s2:='Hppy New Year';
.....
```

Cách khai báo trên cho ta ct là mảng của 10 con trỏ, tất cả mười con trỏ này đều trỏ đến kiểu dữ liệu String. Mỗi con trỏ có thể trỏ đến một đối tượng khác nhau. Trong trường hợp này cách truy nhập dữ liệu cần phải thận trọng.

Nếu chúng ta chưa gán địa chỉ của bất kỳ đối tượng nào cho biến con trỏ mà chỉ thực hiện phép gán:

```
ct[i]^ := s1; với 1 <= i <= 10.
```

thì tất cả mười con trỏ đều trỏ tới biến s1.

Khi đó các lệnh

```
Write(ct[1]^); Write(ct[2]^); .... Write(ct[10]^);
```

 cho kết quả như nhau.

Trong trường hợp chúng ta gán dữ liệu từ một đối tượng cho nhiều biến con trỏ thì tất cả các con trỏ đều trỏ tới đối tượng được gán cuối cùng.

Nếu thực hiện phép gán

```
ct[1] := @s2;
```

nghĩa là gán địa chỉ của biến s2 vào con trỏ thứ nhất trong mảng thì chỉ có con trỏ ct[1] là trỏ tới biến s2, các con trỏ còn lại chưa trỏ vào đâu cả.

Xét ví dụ sau:

Ví dụ 4.7

```
Uses crt;
Type
    m = array[1..5] of byte;
Var
    i:byte; s1,s2:string;
    mct: array[1..10] of ^string; ctm: ^m;
Begin
    clrscr;
    for i:=1 to 5 do
        begin
```

```

ctm^[i]:=i;
write(ctm^[i], ' ');
end;
writeln;
writeln(ctm^[3]);
s1:='Ha noi Viet nam';
s2:='Happy New Year';
mct[10]^:='aaaaa';
mct[4]^:='bbbb';
mct[1]:=s1;
mct[2]:=s2;
writeln(mct[1]^);
writeln(mct[2]^);
writeln(mct[3]^);
writeln(mct[5]^);
readln;
end.

```

Chạy chương trình chúng ta nhận được kết quả:

```

1 2 3 4 5
3
Ha noi Viet nam
Happy New Year
bbbb
bbbb

```

Ví dụ 4.7 khai báo `ctm` là con trỏ kiểu mảng của các số nguyên, còn `mct` là mảng của các con trỏ kiểu chuỗi.

Với con trỏ kiểu mảng `ctm` chúng ta chỉ có một con trỏ, con trỏ này trỏ tới kiểu dữ liệu mảng `m` đã khai báo nên nó có 5 thành phần và chúng ta có thể truy nhập đến từng thành phần thông qua biến `ctm^[i]`.

Biến `mct` cho ta một biến mảng mỗi thành phần mảng là một con trỏ và tất cả các con trỏ này đều trỏ tới cùng một kiểu dữ liệu.

Phép gán:

```
mct[10]^:='aaaaa'; mct[4]^:='bbbb';
```

là gán dữ liệu trực tiếp vào hai biến `mct[10]^` và `mct[4]^`, sau hai phép gán này cả mười con trỏ đều trỏ tới chuỗi 'bbbb' chính vì vậy các lệnh:

```

writeln(mct[3]^);
writeln(mct[5]^);

```

đều cho kết quả là bbbbb.

Hai lệnh gán:

```
mct[1]:=@s1;
```

```
mct[2]:=@s2;
```

đã định lại hướng của con trỏ, con trỏ mct[1] trỏ tới biến s1, còn con trỏ mct[2] trỏ tới biến s2 vì vậy các lệnh:

```
writeln(mct[1]^);
```

```
writeln(mct[2]^);
```

cho kết quả là:

Ha noi Viet nam

Happy New Year

6. Cấp phát động

6.1 Quản lý vùng nhớ Heap

Với một biến con trỏ ct chúng ta có một biến ct[^] tương ứng, đây không phải là biến tĩnh vì không được khai báo ở phần Var, nhưng cũng chưa phải là biến động. Muốn ct là một biến động thực sự thì phải dùng kỹ thuật cấp phát bộ nhớ động.

Cấp phát bộ nhớ động là việc cấp phát bộ nhớ được thực hiện bởi câu lệnh trong thân chương trình chứ không phải bằng cách khai báo biến hoặc tham số. Khi một biến được cấp phát bộ nhớ động thì nó trở thành biến động (Dynamic Variable). Vùng nhớ dành cho cấp phát động bao giờ cũng là vùng nhớ tự do Heap.

Theo mặc định khi một biến động được hình thành thì địa chỉ của nó được lưu trong biến con trỏ tương ứng.

Trước khi nghiên cứu cách thức cấp phát động chúng ta cần biết hệ thống quản lý vùng nhớ tự do như thế nào.

Vùng nhớ thấp được dành cho hệ điều hành, tiếp đó là vùng lưu mã chương trình (Code) vùng lưu các biến toàn cục (Data), vùng lưu các biến cục bộ (Stack). Pascal dùng toàn bộ vùng nhớ còn lại của máy PC cho vùng nhớ tự do (Heap).

Vùng Stack được thiết kế phát triển theo chiều đi xuống, nghĩa là các địa chỉ cao được sử dụng trước, địa chỉ thấp được sử dụng sau.

Vùng Heap lại phát triển theo chiều đi lên, nghĩa là địa chỉ thấp được sử dụng trước, địa chỉ cao sử dụng sau.

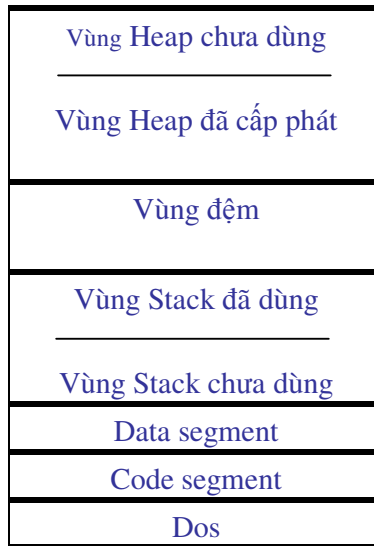
Toàn bộ vùng nhớ tự do Heap trong quá trình sử dụng bị chia thành nhiều khối với kích thước khác nhau. Do việc thu hồi vùng nhớ diễn ra thường xuyên trong chương trình nên các khối nhớ tự do còn lại có thể không nằm kề nhau.

Để quản lý Heap, Pascal có hai hàm là:

Hàm **MemAvail** cho biết tổng dung lượng còn được phép sử dụng trên Heap

Hàm **MaxAvail** cho biết dung lượng của khối lớn nhất còn tự do trên Heap.

Khi xin cấp phát động, điều quan trọng là cần biết kích thước khối nhớ lớn nhất còn tự do chứ không phải là tổng dung lượng tự do bởi vì mỗi đối tượng phải được lưu trữ trong một vùng nhớ liên tục.



Hình 4.1

Kích thước đối tượng dt được xác định bởi hàm `Sizeof(dt)`, do vậy nếu:

$MaxAvail < Sizeof(dt)$

thì không thể xin cấp phát vùng nhớ trên Heap cho dt được, mặc dù tổng dung lượng trên Heap có thể vẫn còn lớn hơn kích thước dt nhiều lần.

Để biết địa chỉ của Heap bắt đầu và kết thúc từ đâu có thể sử dụng các biến không kiểu đã thiết kế trong Pascal :

Biến **HeapOgr**: cho địa chỉ điểm bắt đầu của Heap dùng cho cấp phát động.

Lệnh `Write(Seg(heaporg^), ' : ', ofs(heaporg^));` sẽ hiện địa chỉ này lên màn hình dưới dạng `Segment:Ofset`.

Biến **HeapEnd**: Cho địa chỉ điểm cuối của Heap được sử dụng cho đối tượng.

Khi chương trình bắt đầu được tải vào bộ nhớ thì các giá trị trên cũng được hệ thống khởi gán và giữ nguyên không thay đổi

Biến **HeapPtr**: địa chỉ đỉnh Heap, tức là địa chỉ đáy của vùng nhớ tự do còn được phép sử dụng.

Lúc đầu giá trị `HeapPtr = Heaporg`, sau đó giá trị `HeapPtr` sẽ phát triển theo chiều hướng lên.

6.2 Thủ tục cấp phát bộ nhớ cho con trỏ định kiểu

Với một con trỏ định kiểu ct thủ tục cấp phát bộ nhớ động sẽ là :

`New(ct);`

Thủ tục `New(ct)` cùng một lúc thực hiện các công việc sau:

* Cấp phát một vùng nhớ trên Heap với kích thước bằng kích thước kiểu dữ liệu mà con trỏ trỏ tới.

* Tạo một biến động định kiểu ct[^] để có thể truy nhập vào vùng dữ liệu của đối tượng.

* Lưu địa chỉ của đối tượng vào con trỏ ct

Khi một biến động ct[^] đã hết giá trị sử dụng thì cần thu hồi vùng nhớ đã cấp phát cho nó để dùng vào việc khác. Thủ tục thu hồi là

Dispose(ct);

Sau khi thu hồi vùng nhớ mặc dù biến ct[^] vẫn còn tồn tại nhưng dữ liệu trong vùng nhớ đã dành cho ct sẽ không được bảo vệ, điều này có nghĩa là hệ thống có thể dùng vùng nhớ này vào việc khác. Ví dụ sau đây sẽ cho ta thấy rõ điều này.

Ví dụ 4.8

```
Program cp_dong;  
Uses crt;  
Type s=string[30];  
Var  
  ct:^s;  
Begin  
  clrscr;  
  new(ct);  
  ct^:='Ha noi Viet nam';  
  Writeln('Bien ct^ sau khi cap phat dong: ', ct^);  
  Dispose(ct);  
  Writeln('Bien ct^ sau thu tuc Dispose(ct) : ', ct^);  
  readln;  
End.
```

Chạy chương trình chúng ta nhận được kết quả:

Bien ct[^] sau khi cap phat dong: Ha noi Viet nam

Bien ct[^] sau thu tuc Dispose(ct) : Ha □ Viet nam

Có thể thấy rằng việc giải phóng vùng nhớ dành cho biến đã làm cho dữ liệu thay đổi mặc dù chúng ta chưa hề ra lệnh thay đổi nội dung vùng nhớ này.

6.3 Thủ tục cấp phát bộ nhớ cho con trỏ không định kiểu

Pascal có hai thủ tục cấp phát và thu hồi vùng nhớ cho con trỏ không định kiểu ct là:

Getmem(ct, n); cấp cho ct n Byte.

Freemem(ct, n); thu hồi n Byte vùng nhớ đã cấp cho ct.

Việc cấp phát không định kiểu thường được dùng trong trường hợp chưa biết trước kích thước của đối tượng, ví dụ để lưu một ảnh nằm trong khung chữ nhật tọa độ góc trên trái là x1, y1 và tọa độ góc dưới phải là x2, y2. Gọi n là kích thước ảnh tính bằng Byte, chúng ta xác định kích thước ảnh bằng hàm:

n := Imagesize(x1,y1,x2,y2);

Thủ tục Getmem(ct, n) sẽ cấp cho con trỏ không kiểu ct một vùng nhớ với kích thước đúng bằng kích thước ảnh, sau đó có thể lưu ảnh vào biến con trỏ thông qua thủ tục:

```
GetImage(x1,y1,x2,y2, ct^);
```

6.4 Thu hồi nhiều vùng nhớ

Để thu hồi một lúc nhiều vùng nhớ hoặc toàn bộ vùng Heap, chúng ta thực hiện các bước sau đây:

- * Khai báo một biến con trỏ ct không kiểu để lưu trữ địa chỉ ô nhớ bắt đầu thu hồi (tức là đánh dấu vị trí để sau này thu hồi các vùng nhớ từ vị trí này).

- * Đánh dấu vị trí sẽ bắt đầu thu hồi bởi thủ tục:

```
Mark(ct)
```

- * Thu hồi vùng nhớ (tức là huỷ bỏ toàn bộ các biến động đã được cấp phát trong vùng nhớ kể từ khi đánh dấu)

```
Release(ct);
```

6.5 Các ví dụ về cấp phát động

Ví dụ 4.9

Ví dụ 4.9 thiết kế chương trình con Diachi nhằm thông báo kích thước vùng nhớ Heap mà hệ thống cấp cho Pascal, kích thước của khối nhớ lớn nhất có trong Heap và địa chỉ đỉnh Heap trước và sau khi cấp phát động.

Con trỏ không kiểu bd (bắt đầu) dùng để đánh dấu vị trí bắt đầu cấp phát vùng nhớ cho các biến động ct1 và ct2. Sau này khi thu hồi vùng nhớ chương trình sẽ bắt đầu thu hồi từ đây.

```
Program Quan_ly_Heap;  
Uses crt;  
Var bd:pointer;  
    ct1,ct2:^string;  
Procedure diachi;  
Begin  
    writeln('Kich thuc toan bo vung nho Heap: ',memavail);  
    writeln('Kich thuc khoi lon nhat tren Heap: ',maxavail);  
    writeln('Dia chi bat dau vung Heap: ',seg(heaporg^),' : ',ofs(heaporg^));  
    writeln('Dia chi ket thuc vung Heap: ',seg(heapend^),' : ',ofs(heapend^));  
    writeln('Dia chi dinh Heap: ',seg(heapptr^),' : ',ofs(heapptr^));  
    writeln;writeln;  
end;  
Begin  
    clrscr;  
    writeln('Trang thai ban dau cua Heap');
```

```

diachi;
Mark(bd);
new(ct1);
ct1^:='aaaaaaaaaaaaa';
writeln;
writeln('Sau khi cap phat cho bien dong ct1^ ');
diachi;
New(ct2);
ct2^:='qqq';
writeln('Sau khi cap phat cho bien dong ct2^ ');
diachi;
release(bd);
writeln('Sau khi thu hoi toan bo bien dong ');
diachi;
readln;
End.

```

Ví dụ 4.10

Ví dụ 4.10 minh họa việc dùng con trỏ mảng để quản lý danh sách học sinh. Kiểu dữ liệu *Nguoi* bao gồm các trường *Stt* (số thứ tự), *Hoten* (họ và tên), *Gioi* (giới tính), *Tong* (tổng điểm), *Xeploai* (xếp loại). Con trỏ *Dslop* trỏ đến kiểu dữ liệu *ds*, vì *ds* là mảng của 100 phần tử kiểu *Nguoi* nên *dslop* là con trỏ mảng. Trong ví dụ chúng ta chỉ hình thành nên **một con trỏ** và con trỏ này có thể trỏ tới 100 phần tử của mảng. Thủ tục xin cấp phát động *New(dslop)* sẽ tạo nên một biến động kiểu mảng, phần tử thứ *i* của biến động này sẽ là $dslop^i$ và chúng ta có thể hình dung $dslop^i$ chính là bản ghi thứ *i* trong mảng *ds*. Ví dụ cũng nêu lên cách thức sử dụng toán tử *With ...* Do trong các bài toán có liên quan đến con trỏ.

Vì toán tử *With ...* Do chỉ có tác động đối với các biến kiểu *Record* nên chúng ta không thể đưa vào trong *With ...* Do các con trỏ mà chỉ có thể là các biến động.

```

Program con_tro_mang;
Uses crt;
Type nguoi=record
  stt:byte;
  Hoten:string[25];
  Gioi:char;
  tong:real;
  xeploai:string[5];
end;
ds=array[1..100] of nguoi;
Var
  dslop:^ds; i,n:byte;

```

```

Begin
  clrscr;
  write('cho biet so hoc sinh can nhap: ');
  Readln(n);
  new(dslop);
  For i:=1 to n do
  with dslop^[i] do
  Begin
    writeln('So thu tu: ',i); stt:=i;
    write('Nhap Ho ten: '); readln(hoten);
    write('Nhap gioi T/G : '); readln(gioi);
    write('Nhap tong diem: '); readln(tong);
    write('Nhap xep loai: '); readln(xeploai);
  End;
  writeln('          DANH SACH HOC SINH');
  writeln('So TT : Ho va ten   : Gioi : Tong diem : Xep loai ');
  For i:= 1 to n do
  With dslop^[i] do
    writeln(stt:3, hoten:15,' ',gioi:4,' ', Tong:5:2,' ', xeploai);
    Readln;
  End.

```

Thay vì sử dụng con trỏ mảng, chúng ta sử dụng mảng các con trỏ. Trong ví dụ 4.11 Dslop là mảng của 100 con trỏ, các con trỏ này trỏ tới kiểu dữ liệu Nguoi. Biến không kiểu BD dùng để đánh dấu vị trí bắt đầu cấp phát động.

Thủ tục New(dslop[i]) trong vòng lặp For i:= 1 to n cho phép tạo nên n biến động dslop[i]^, các biến động này được cấp phát vùng nhớ trên Heap.

Ví dụ 4.11

```

Program mang_con_tro;
uses crt;
Type nguoi=record
  stt:byte;
  Hoten:string[25];
  Gioi:char;
  tong:real;
  xeploai:string[5];
end;
Var
  bd:pointer;

```

```

    dslop:array[1..100] of ^nguoi; i,n:byte;
Begin
    clrscr;
    write('cho biet so hoc sinh can nhap: ');
    Readln(n);
    mark(bd);
    For i:= 1 to n do
    begin
        new(dslop[i]);
        with dslop[i]^ do
        begin
            writeln('So thu tu: ',i);stt:=i;
            write('Nhap Ho ten: '); readln(hoten);
            write('Nhap gioi T/G : '); readln(gioi);
            write('Nhap tong diem: '); readln(tong);
            write('Nhap xep loai: '); readln(xeploai);
        end; { with}
    end; {For}
    writeln('So TT : Ho va ten : Gioi : Tong diem : Xep loai ');
    For i:= 1 to n do
    with dslop[i]^ do
    writeln(stt:3, hoten:15,' ', gioi:4,' ', Tong:5:2,' ', xeploai);
    Readln;
    END.

```

7. Danh sách liên kết và hàng đợi

Với con trỏ kiểu mảng đã nêu việc lập trình tạo danh sách cũng như duyệt danh sách khá đơn giản, hạn chế của kiểu mảng là phải khai báo trước kích thước mảng do đó nhiều khi dẫn tới không tiết kiệm bộ nhớ. Sử dụng biến động chúng ta có thể khắc phục được nhược điểm này bằng cách cần đến đâu thì tạo biến động đến đó. Số biến động tạo ra chỉ bị hạn chế bởi bộ nhớ trong (Ram) của máy PC mà cụ thể là phụ thuộc vào kích thước vùng Heap.

Danh sách được hiểu là một tập hợp hữu hạn các phần tử liên kết với nhau, trường hợp tổng quát nhất mỗi phần tử là một bản ghi. Điều đặc biệt của mỗi bản ghi trong danh sách là ngoài các trường dữ liệu, còn một trường dùng để liên kết và trường này lại là một con trỏ. Con trỏ này có nhiệm vụ trỏ vào địa chỉ của bản ghi kế tiếp. Nếu bản ghi hiện thời là bản ghi cuối cùng thì con trỏ sẽ trỏ vào Nil.

Như vậy xuất phát từ bản ghi đầu, lần theo địa chỉ lưu ở trường con trỏ chúng ta có thể truy nhập vào bản ghi tiếp theo, quá trình sẽ tiếp diễn cho đến bản ghi cuối cùng.

Một danh sách chưa có phần tử nào được gọi là danh sách rỗng. Việc thêm một phần tử vào danh sách (nghĩa là tạo nên danh sách) có thể rơi vào một trong ba khả năng:

- a. Phần tử mới được thêm vào đầu danh sách
- b. Phần tử mới được nối vào cuối danh sách
- c. Phần tử mới được chèn vào một vị trí xác định.

Trường hợp a chúng ta có danh sách liên kết ngược (LIFO), còn trường hợp b chúng ta có danh sách liên kết thuận (FIFO) hay còn gọi là hàng đợi QUEUE.

7.1 Danh sách liên kết ngược

Là loại danh sách mà trường liên kết của phần tử tạo ra sau luôn trở vào phần tử tạo ra trước đó. Trường liên kết của phần tử tạo ra đầu tiên trở vào Nil. Điều này dẫn tới việc khi kết xuất thông tin ra chúng ta phải bắt đầu từ phần tử tạo ra cuối cùng vì chỉ có như vậy chúng ta mới biết địa chỉ của phần tử tạo ra trước đó. Nếu cố tình đi từ phần tử tạo ra đầu tiên thì chúng ta không thể biết phần tử tiếp theo là phần tử nào. Liên kết kiểu này gọi là kiểu LIFO (Last In - First Out) hay còn gọi là kiểu xếp chồng. Phần tử nhập vào cuối cùng sẽ được lấy ra đầu tiên. Danh sách tạo ra theo kiểu này được gọi là danh sách liên kết ngược.

Giả thiết rằng chúng ta cần xây dựng một danh sách học sinh với các trường dữ liệu là Mhs (mã hồ sơ), Hoten (Họ và tên), Diem (điểm tổng kết) , trường liên kết lấy tên là Tiejp (tiếp tục). Kiểu dữ liệu bản ghi với các trường nêu trên lấy tên là Nguoi. Để tạo ra danh sách học sinh chúng ta cần tạo ra một kiểu con trở DS trở vào kiểu dữ liệu Nguoi và trường liên kết Tiejp trong bản ghi Nguoi sẽ trở vào kiểu dữ liệu Ds.



Hình 4.2

Type

```

Ds = ^nguoi;
Nguoi = Record
Mhs:byte;
Hoten: string[20];
Diem:real;
Tiejp: Ds;
End;

```

Chú ý:

Với cách thức nói trên nếu chúng ta khai báo kiểu bản ghi Nguoi trước khi khai báo kiểu con trỏ Ds thì sẽ bị lỗi vì khi đó con trỏ Tiej sẽ trỏ vào một kiểu dữ liệu chưa được định nghĩa.

Sau khi khai báo kiểu dữ liệu cần khai báo biến con trỏ Dslop để lưu trữ dữ liệu nhập vào và biến Ctcuoi (con trỏ cuối) để trỏ vào phần tử cuối cùng.

Vấn đề là làm thế nào để con trỏ liên kết Tiej luôn trỏ vào phần tử tạo ra trước đó. Để làm việc này chúng ta tạo ra biến động Dslop để lưu trữ dữ liệu. Cứ mỗi bản ghi cần nhập vào thì tạo ra một biến động Dslop mới. Địa chỉ của biến động Dslop luôn được gán cho con trỏ cuối Ctcuoi. Dưới đây là đoạn mô phỏng chương trình tạo danh sách liên kết ngược:

```
Ctcuoi:=nil; {khởi tạo danh sách}
```

```
Bắt đầu lặp
```

```
New(dslop); {tạo biến động lưu trữ dữ liệu nhập vào}
```

```
Nhập dữ liệu; {nhập dữ liệu cho phần tử thứ i}
```

```
Tiej:=ctcuoi; {trường liên kết của phần tử thứ i trỏ vào địa chỉ  
của con trỏ cuối ctcuoi }
```

```
Ctcuoi :=dslop; {hướng con trỏ cuối vào bản ghi hiện thời}
```

```
Kết thúc lặp
```

Nhận xét:

* Vòng lặp thứ 1:

- Tạo biến động Dslop sau đó nhập dữ liệu vào phần tử đầu tiên.
- Câu lệnh Tiej:= ctcuoi; sẽ hướng con trỏ liên kết Tiej của bản ghi này trỏ vào Nil vì lúc này con trỏ cuối đang là Nil.
- Câu lệnh ctcuoi:=dslop; sẽ hướng con trỏ cuối đến phần tử thứ nhất.

* Vòng lặp thứ 2:

-Tạo biến động Dslop lần thứ hai, nhập dữ liệu cho phần tử thứ hai. Câu lệnh Tiej:=ctcuoi; sẽ hướng trường liên kết của phần tử thứ hai đến phần tử thứ nhất vì lúc này ctcuoi đang mang địa chỉ của phần tử thứ nhất.

- Câu lệnh ctcuoi:=dslop; sẽ chuyển hướng con trỏ cuối sang phần tử thứ hai.

- Có thể dễ dàng suy ra rằng nếu vòng lặp thực hiện lần thứ ba thì con trỏ liên kết của phần tử thứ ba sẽ trỏ vào phần tử thứ hai, còn con trỏ cuối sẽ trỏ vào phần tử thứ ba...

Quá trình sẽ lặp lại cho đến khi kết thúc.

Dưới đây là ví dụ xây dựng danh sách, chương trình con Hien_LIFO cho hiện dữ liệu lên màn hình theo chiều ngược, phần tử nhập sau hiện trước.

Ví dụ 4.12

```
Program danh_sach_lien_ket_nguoc;
```

```
Uses crt;
```

```
Type ds= ^nguoi;
```

```
nguoi=record
```

```
Mhs:byte; Hoten:string[25]; Diem:real; Tiej:ds; End;
```



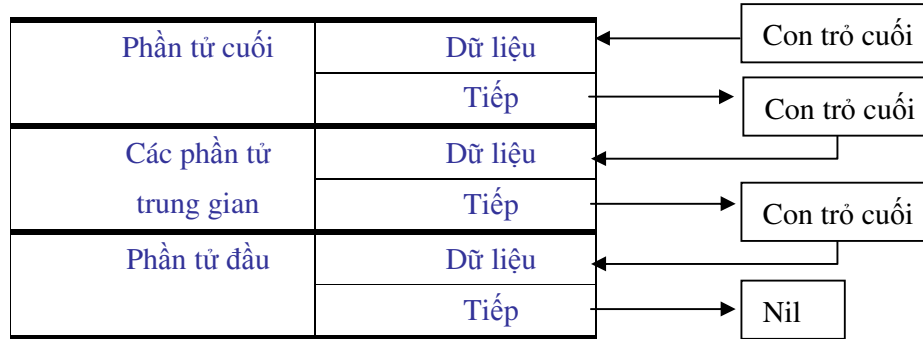
```

Var
    dslop, ctcuoi:ds; i,j:byte;
    lam:char;
Procedure Hien_LIFO; {Thu tuc hien du lieu tu cuoi ve dau}
var ct1:ds;
Begin
    clrscr;
    writeln('Du lieu da nhap - Hien tu cuoi ve dau');
    writeln;
    ct1:=ctcuoi;
    while ct1<>nil do
    with ct1^ do
    Begin
        Write(Mhs,' ', hoten);
        for j:=1 to (20-length(hoten)) do write(' ');
        writeln(diem:5:2);
        ct1:=tiiep;
    end;
End;

Begin {Than chuong trinh chinh}
    clrscr;
    ctcuoi:=nil; i:=0;
    Repeat
    New(dslop); i:=i+1;
    With dslop^ do
    Begin
        writeln('Ma ho so so: ',i); mhs:=i;
        write('Ho va ten: '); readln(hoten);
        write('Diem    : '); Readln(diem);
        tiiep:=ctcuoi;
        ctcuoi:=dslop;
        writeln('Nhap tiiep hay thoi? C/K '); lam:=readkey;
        writeln;
    end;
    Until lam in ['k','K'];
    Hien_lifo;
    Readln;
END.

```

Trong ví dụ trên thay vì trường *Tiếp* của phần tử hiện thời trở trực tiếp vào phần tử đứng trước chúng ta cho nó trở vào "con trở cuối" rồi "con trở cuối" sẽ trở vào phần tử kế tiếp. Mô hình mô tả quá trình nhập dữ liệu như sau:



Hình 4.3

7.2 Hàng đợi Queue - Danh sách liên kết thuận

Với loại danh sách mà phần tử nào nhập trước thì được lấy ra trước chúng ta gọi là danh sách liên kết thuận, nó cũng giống như hàng đợi người nào đến trước thì được gọi trước.

Để tạo ra hàng đợi ngoài *ctcuoi* chúng ta phải thêm vào con trở đầu (*ctdau*). Con trở cuối *ctcuoi* luôn trở vào phần tử cuối cùng, còn con trở đầu lại luôn trở vào phần tử đầu của danh sách. Dưới đây mô phỏng thuật toán chương trình xây dựng Queue.

```

Ctdau:=nil; {khởi tạo danh sách}
Bắt đầu lặp
    New(dslop); {tạo biến động lưu trữ dữ liệu }
    Nhập dữ liệu; {nhập dữ liệu cho phần tử thứ i, i=1,2... }
    Nếu Ctdau = Nil thì ctdau :=dslop;
    Còn nếu Ctdau <> Nil thì ctcuoi^.tiếp := dslop;
    Ctcuoi := dslop;
    Ctcuoi^.tiếp := Nil;
Kết thúc lặp
    
```

Chương trình mô phỏng trên cho ta kết quả sau:

*** Vòng lặp thứ nhất:**

- Nhập dữ liệu cho phần tử thứ nhất
- Ctdau trở vào phần tử thứ nhất
- Ctcuoi trở vào phần tử thứ nhất
- Trường liên kết *Tiếp* trở vào cuối danh sách (Nil), vì Ctcuoi đang trở vào phần tử thứ nhất nên trường *Tiếp* của Ctcuoi cũng chính là trường *Tiếp* của phần tử thứ nhất.

*** Vòng lặp thứ hai**

- Nhập dữ liệu cho phần tử thứ hai
 - Vì ctdau đang trỏ vào phần tử thứ nhất (ctdau<>Nil) nên chuyển hướng trường Tiej của ctcuoi đến phần tử thứ hai. Vì Ctcuoi đang trỏ vào phần tử thứ nhất nên điều này cũng có nghĩa là trường Tiej của phần tử thứ nhất trỏ vào phần tử thứ hai (không trỏ vào Nil nữa).
 - Xác nhận lại rằng ctcuoi trỏ vào phần tử thứ hai (ctdau vẫn trỏ vào phần tử thứ nhất).
 - Trường liên kết Tiej trỏ vào cuối danh sách (Nil), vì Ctcuoi đang trỏ vào phần tử thứ hai nên trường Tiej của Ctcuoi cũng chính là trường Tiej của phần tử thứ hai.
- * Vòng lặp tiếp theo sẽ tương tự như vòng lặp thứ hai
- Dưới đây là chương trình xây dựng hàng đợi, chương trình con Hien_FIFO cho hiện dữ liệu lên màn hình theo đúng thứ tự nhập vào.

Ví dụ 4.13

```

Program danh_sach_lien_ket_thuan;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
  Mhs: Byte; Hoten:string[25]; Diem:real; tiep:ds;
End;
Var
dslop, ctdau, ctcuoi:ds; bd:pointer;
lam:char; i,j:byte;

Procedure Hien_FIFO; {Hien du lieu tu dau xuong cuoi}
  Var ct1:ds;
Begin
  Clrscr;
  writeln('Du lieu da nhap - Hien tu dau xuong cuoi');
  writeln;
  ct1:=ctdau;
  while ct1<>nil do
    with ct1^ do
      Begin
        Write(Mhs,' ',hoten);
        for j:=1 to (20-length(hoten)) do write(' ');
        writeln(diem:5:2);
        ct1:=tiep;
      End;
  End;
End;

```

```

Begin {Than chuong trinh chinh}
clrscr;
mark(bd);
ctdau:=nil; i:=0;
Repeat
  New(dslop); {tạo biến động Dslop}
  i:=i+1;
  With dslop^ do
    Begin
      Writeln('Ma ho so so: ',i); mhs:=i;
      Write('Ho va ten: '); readln(hoten); {nhập dữ liệu}
      Write('Diem   : '); Readln(diem);
      if ctdau=nil then
        ctdau:=dslop {ctdau trở vào phần tử thứ 1 }
      else
        ctcuoi^.tiếp:=dslop; {Lưu trữ địa chỉ của phần tử hiện thời - kể từ phần tử
                               thứ hai vào trường liên kết Tiếp của Ctcuoi }
        ctcuoi:=dslop;      {ghi nhận lại con trỏ cuối, nghĩa là Tiếp đang trỏ
                               vào phần tử hiện thời}
        ctcuoi^.tiếp:=nil;  { trường liên kết Tiếp của con trỏ cuối trỏ vào Nil
                               - ket thuc danh sach}
    End;
  Writeln('Nhap tien hay thoi? C/K '); lam:=readkey;
  writeln;
End;
Until lam in ['k','K'];
Hien_FiFO;
Release(bd);
End.

```

7.3 Chèn thêm phần tử vào danh sách

Việc đầu tiên khi muốn chèn thêm phần tử vào danh sách là phải xác định được chính xác vị trí cần chèn, muốn vậy danh sách phải có một trường khoá, mỗi phần tử trong danh sách sẽ ứng với một và chỉ một giá trị của trường khoá. Ví dụ có thể lấy trường số thứ tự (STT) hoặc mã hồ sơ (MHS) làm trường khoá, không thể dùng trường Hoten để làm khoá vì trong danh sách có thể có nhiều người trùng Hoten.

Quá trình chèn một phần tử vào danh sách sẽ qua các bước:

- * Xác định vị trí chèn
- * Tạo một biến động (xem như một con trỏ nháp) và xin cấp phát vùng nhớ cho biến động để lưu dữ liệu sẽ chèn vào danh sách.
- * Chuyển trường Tiếp của phần tử hiện thời đến phần tử bổ xung
- * Chuyển trường Tiếp của phần tử bổ xung đến phần tử trước phần tử hiện thời.

```

New(ct1); {tạo phần tử nháp để tạm lưu dữ liệu}
With ct1^ do Nhập dữ liệu cho phần tử bổ xung
dslop:=ctcuoi; {hướng con trỏ đến phần tử cuối cùng trong danh sách}
While (dslop<>nil) and (dslop^.mhs <> n) do
  dslop:=dslop^.tiếp; {hướng con trỏ đến vị trí cần chèn}
  ct1^.tiếp:=dslop^.tiếp; {gán trường liên kết Tiếp của phần tử hiện thời cho Tiếp của
ct1 - nghĩa là chèn ct1 vào trước phần tử hiện thời}
  dslop^.tiếp:=ct1; {trường liên kết Tiếp của bản ghi hiện thời trỏ vào ct1 - nghĩa là lùi
phần tử hiện thời ra sau ct1 }

```

7.4 Xoá một phần tử khỏi danh sách

Quá trình xoá một phần tử trong danh sách không phải là quá trình làm rỗng ô nhớ chứa phần tử mà đơn giản chỉ là chuyển hướng trường liên kết không trỏ vào phần tử đó nữa. Nếu chúng ta xoá nhiều phần tử thì bộ nhớ sẽ bị phân mảnh ra nhiều đoạn ngắt quãng, trong trường hợp này cần bố trí lại các ô nhớ để một đối tượng được bố trí trên một số ô nhớ liên tục (xem ví dụ 4.14).

Ví dụ 4.14

```

Program danh_sach_lien_ket_nguoc;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
  Mhs:byte;
  Hoten:string[25];
  Diem:real;
  tiếp:ds;
End;
Var
dslop, ctcuoi:ds; i,j,n:byte;
lam:char;

Procedure Hien_LIFO; {Thu tuc hien du lieu tu cuoi ve dau}
var ct1:ds;
Begin
clrscr;
writeln('Du lieu da nhap - Hien tu cuoi ve dau');
writeln;
ct1:=ctcuoi;
while ct1<>nil do
  with ct1^ do

```

```

Begin
    Write(mhs, ' ', hoten);
    for j:=1 to (20-length(hoten)) do write(' ');
    writeln(diem:5:2);
    ct1:=tiep;
end;
Readln;
End;
{*****}
Procedure Chen;
Var n:byte; ct1:ds;
Begin
    Clrscr;
    writeln('Chen truoc ma ho so nao? '); Readln(n);
    New(ct1);
    With ct1^ do
    Begin
        Writeln('Ma ho so so: ',n-1);
        Write('Ho va ten: '); readln(hoten);
        Write('Diem    '); Readln(diem);
        End;
        dslop:=ctcuoi;
        While (dslop<>nil) and (dslop^.mhs <> n) do dslop:=dslop^.tiep;
        ct1^.tiep:=dslop^.tiep;
        dslop^.tiep:=ct1;
    End;
    {*****}
Procedure Xoa;
Var n:byte; ct1:ds;
Begin
    Write('Cho biet ma ho so can xoa '); readln(n);
    dslop:=ctcuoi;
    While (dslop<>nil) and (dslop^.mhs<>n) do
        Begin
            ct1:=dslop;    dslop:=dslop^.tiep;
        End;
        If dslop=ctcuoi then ctcuoi:=dslop^.tiep
        Else
            ct1^.tiep:=dslop^.tiep;
    End;

```

```

Procedure Nhap;
  Var lam:char;
Begin
  clrscr;
  ctcuoi:=nil;
  Repeat
    New(dslop); i:=i+1;
    With dslop^ do
      Begin
        Writeln('Ma ho so so: ',i); Mhs:=i;
        Write('Ho va ten: '); readln(hoten);
        Write('Diem    '); Readln(diem);
        tiep:=ctcuoi;
        ctcuoi:=dslop;
        Writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
        writeln;
      End;
    Until lam in ['k','K'];
  End;

Begin {Than chuong trinh chinh}
  Repeat
    clrscr;
    Writeln('1: Nhap du lieu');
    Writeln('2: Hien_lifo ');
    Writeln('3: Chen them phan tu');
    Writeln('4: Xoa phan tu ');
    Write('Xin moi chon cong viec! Bam 99 de ket thuc '); readln(n);
    Case n of
      1:Nhap;
      2:Hien_lifo;
      3:Chen;
      4:Xoa;
    End;
  Until n=99;
End.

```

Ví dụ 4.15

Danh sách liên kết thuận

```
Program danh_sach_lien_ket_thuan;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
mhs:byte;
  Hoten:string[25];
  Diem:real;
  tiep:ds;
End;
Var
dslop, ctdau, ctcuoi:ds;
lam:char; i,j,n:byte;

Procedure Hien_FIFO; {Hien du lieu tu dau xuong cuoi}
var ct1:ds;
Begin
  clrscr;
  ct1:=ctdau;
  writeln('Du lieu da nhap - Hien tu dau xuong cuoi');
  writeln;
  ct1:=ctdau;
  while ct1<>nil do
  with ct1^ do
  Begin
    Write(mhs,' ',hoten);
    for i:=1 to (20-length(hoten)) do write(' ');
    writeln(diem:5:2);
    ct1:=tiep;
  end;
  readln;
End;

{***** }
```



```

Procedure Nhap;
Begin
  clrscr;
  ctdau:=nil; j:=0;
Repeat
  New(dslop); j:=j+1;
  With dslop^ do
  Begin
    writeln('Ma ho so: ',j); mhs:=j;
    write('Ho va ten: '); readln(hoten);
    write('Diem   : '); Readln(diem);
    if ctdau=nil then
      ctdau:=dslop {ctdau tro vao phan tu 1 }
    else
      ctcuoi^.tiep:=dslop; {truong lien ket Tiep tro vao phan tu tiep theo }
      ctcuoi:=dslop; {ghi nhan lai con tro cuoi nghĩa là Tiep dang tro
        vao phan tu tiep theo}
      ctcuoi^.tiep:=nil;   {truong lien ket tro vao Nil - ket thuc danh sach}
    writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
    writeln;
  End;
  Until lam in ['k','K'];
End;

```

```
{*****}
```

```

Procedure Chen;
Var n:byte; ct1:ds;
Begin
  Clrscr;
  writeln('Chen truooc ma ho so nao? '); Readln(n);
  New(ct1);
  with ct1^ do
  Begin
    writeln('Ma ho so so: ',n-1);
    write('Ho va ten: '); readln(hoten);
    write('Diem   : '); Readln(diem);
  End;
  dslop:=ctdau;
  While (dslop<>nil) and (dslop^.mhs <> n) do dslop:=dslop^.tiep;

```

```

ct1^.tiep:=dslop^.tiep;
dslop^.tiep:=ct1;
End;

```

```

{*****}

```

```

Procedure Xoa;
Var n:byte; ct1:ds;
Begin
write('Cho biet ma ho so can xoa '); readln(n);
dslop:=ctdau;
while (dslop<>nil) and (dslop^.mhs<>n) do
Begin
ct1:=dslop;
dslop:=dslop^.tiep;
End;
If dslop=ctdau then ctdau:=dslop^.tiep
Else
ct1^.tiep:=dslop^.tiep;
End;

```

```

{*****}

```

```

Begin {Thân chương trình}
Repeat
clrscr;
writeln('1: Nhap du lieu');
writeln('2: Hien_Fifo ');
writeln('3: Chen them phan tu');
writeln('4: Xoa phan tu ');
write('Xin moi chon cong viec! Bam 99 de ket thuc '); readln(n);
Case n of
1:Nhap;
2:Hien_Fifo;
3:Chen;
4:Xoa;
End;
Until n=99;
END.

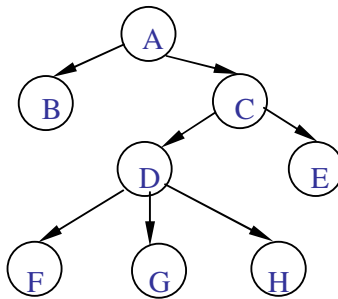
```

8. Cây nhị phân

Chúng ta đã làm quen với nhiều kiểu dữ liệu có cấu trúc như mảng, tệp, bản ghi, ... Dữ liệu kiểu cây cũng là một kiểu dữ liệu có cấu trúc được xây dựng trong Pascal.

8.1 Khái niệm Cây (Tree)

Cây gồm một tập hợp hữu hạn các phần tử gọi là nút (hay là đỉnh) và một tập hợp hữu hạn các cạnh có hướng nối từng cặp nút với nhau. Trong cây có một nút đặc biệt gọi là Gốc, giữa các nút có một quan hệ phân cấp gọi là quan hệ cha con.



Hình 4.4

- * Nút được gọi là Gốc khi từ nút đó chỉ có các cạnh có hướng đi khỏi nút.
- * Số nút con của một nút nào đó gọi là bậc tự do (Degree) hay còn gọi là cấp của nút đó. Trong hình vẽ trên nút A và C có bậc tự do bằng 2, còn nút D có bậc tự do bằng 3.
- * Nút có cấp bằng 0 (tức là không có nút con) gọi là nút lá, nút phân bổ giữa gốc và lá là nút nhánh.
- * Mức của một nút là một số xác định vị trí của nút trong cây. Gốc của một cây được coi là ở mức 1, mức của một nút con nào đó bằng mức của nút cha cộng với 1.
- * Chiều cao của cây bằng số mức lớn nhất của nút có trên cây.

Hình vẽ bên thể hiện một cây, nút A là gốc cây, các nút B, F, G, H, E là nút lá, còn các nút C, D là nút nhánh, chiều cao của cây là bằng 4 vì các nút F, G, H có mức là 4.

Cây nhị phân là trường hợp đặc biệt của cây mà bậc tự do của mỗi nút lớn nhất là bằng hai, nói cách khác mỗi nút cha chỉ có nhiều nhất là hai nút con.

Nếu chúng ta xây dựng các cây mà mỗi nút cha có 10 hoặc 16 nút con thì ta sẽ có cây thập phân hoặc cây thập lục phân.

2. Cây nhị phân

Như đã nêu, cây nhị phân là loại cây đặc biệt mà mỗi nút cha chỉ có nhiều nhất là 2 nút con, các nút con này được gọi là con bên trái (CBT) và con bên phải (CBP). Mô hình cây được vẽ từ trên xuống, gốc ở trên cùng, lá ở dưới. Dữ liệu trong mỗi nút có thể là số hoặc ký

tự và việc lưu trữ dữ liệu được quy định như sau: Nếu dữ liệu trong nút con lớn hơn dữ liệu trong nút cha thì lưu vào con bên phải, ngược lại lưu vào con bên trái.

Tùy thuộc vào dữ liệu lưu trữ tại các nút chúng ta có thể có một trong ba dạng cây sau đây:

a. Cây tự nhiên

Cây tự nhiên là loại cây mà cấp của các nút nằm trong miền (0..2)

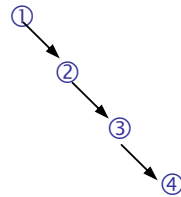
b. Cây đối xứng

Cây đối xứng là loại cây mà cấp của tất cả các nút đều bằng 2.

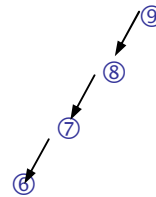
c. Cây một phía hay còn gọi là cây mọc lệch

Cây một phía là loại cây mà cấp của tất cả các nút đều bằng 1.

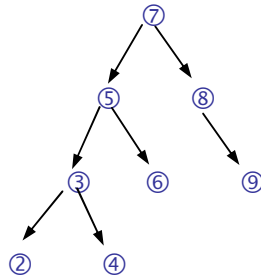
Hình 4.5 dưới đây mô tả các loại cây đã nêu:



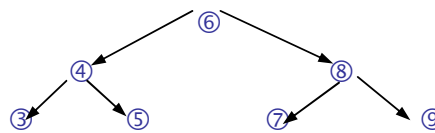
a. Cây lệch phải



b. Cây lệch trái



d. Cây tự nhiên



e. Cây đối xứng

Hình 4.5

Cây được vẽ theo chiều từ trên xuống dưới với nút gốc ở trên và nút lá ở dưới. Một nút mà từ đó có thể truy nhập đến các nút khác thì được gọi là nút cha của các nút đó, còn các nút được truy nhập gọi là nút con.

Cây nhị phân có thể cài đặt bằng cách dùng các cấu trúc liên kết đã nêu trên. Dữ liệu trên một nút được lưu trữ bằng bản ghi, tại mỗi nút có hai con trỏ, CTT và CTP. CTT dùng để trỏ đến con bên trái và CTP dùng để chỉ đến con bên phải. Ví dụ dưới đây nêu cách cài đặt cây nhị phân, dữ liệu tại mỗi nút là một ký tự của bảng mã ASCII. Cần chú ý rằng theo quy ước thì các ký tự có thứ tự là:

$a < b < c < \dots < z$ và $A < B < C < \dots < Z$

Type

ct=^Kytu;

Kytu = record

Nut : char;

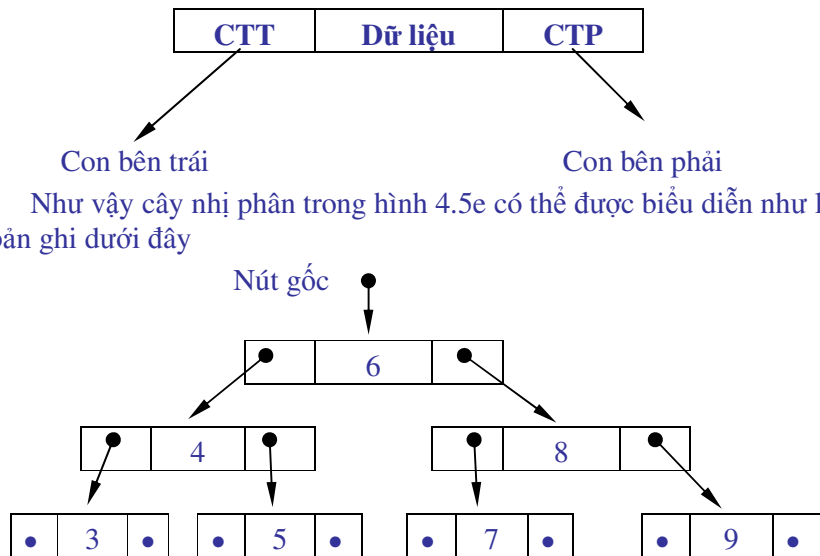
Ctt,Ctp : ct;

end;

Var chucai : char; tim : boolean;

goc, ct1 : ct; batdau : pointer;

Ví dụ trên đã khai báo kiểu dữ liệu con trỏ đặt tên là **Kytu**, đây là kiểu bản ghi với mục đích lưu trữ một ký tự của bảng mã, con trỏ CT dùng để trỏ vào kiểu dữ liệu Kytu. Bên trong kiểu Kytu có các con trỏ **Ctt** và **Ctp** dùng để trỏ vào các nút biểu diễn con bên trái và con bên phải của nút cha.

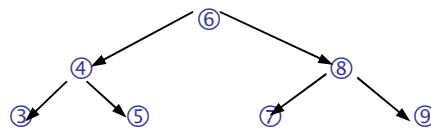


Hình 4.6

Trong hình 4.6 các nút chứa các giá trị 3, 5, 7, 9 là nút lá, chúng không có con bên trái hoặc con bên phải. Các con trở trái CTT và con trở phải CTP của chúng đều trở vào Nil

Một nút cha có thể có một hoặc hai cây con và bài toán xử lý cây nhị phân là thăm một nút, xử lý dữ liệu tại nút đó rồi chuyển sang thăm nút khác. Vấn đề phức tạp là ở chỗ mỗi nút đều có hai con trở do vậy khi di chuyển sang nút khác chúng ta chọn hướng đi nào, không những thế khi đến một nút chúng ta xử lý dữ liệu trước rồi mới đi thăm các nút con hay thăm nút con trước rồi xử lý dữ liệu tại nút cha sau? Tùy thuộc vào cách thức thăm và xử lý chúng ta sẽ có các kết quả khác nhau.

Dưới đây là một ví dụ minh họa.



Hình 4.7

Chúng ta gọi thao tác quét một cây nhị phân là việc thăm mỗi nút đúng một lần, không sót một nút nào và thông tin tại mỗi nút được xử lý cũng chỉ một lần.

Giả sử với cây nhị phân trên hình 4.7, chúng ta thực hiện quá trình quét như sau:

1. Thăm nút gốc và xử lý dữ liệu, cụ thể là hiện giá trị có tại nút
2. Quét cây con bên trái
3. Quét cây con bên phải

Với trình tự này kết quả nhận được sẽ là 6 4 3 5 8 7 9

Để tổng quát chúng ta ký hiệu các bước của tiến trình quét cây như sau:

G: xử lý dữ liệu tại nút (tức là gốc của các cây mẹ hoặc cây con)

T: quét cây con bên trái

P: quét cây con bên phải

Như vậy có tất cả 6 cách quét khác nhau:

TGP; GTP; TPG; GPT; PGT; PTG

Có thể nhận thấy rằng đối với việc quét một cây điều quan trọng không phải là xử lý dữ liệu tại các nút mà là việc xử lý ấy được thực hiện khi nào. Trên cùng một cây với mỗi trình tự thực hiện chúng ta nhận được một kết quả và các kết quả này là không giống nhau.

Trong 6 cách trên, ba cách đầu (cây con bên trái được quét trước cây con bên phải) là ba cách được quan tâm nhiều hơn cả. Căn cứ vào thời điểm xử lý dữ liệu tại nút (bước G), chúng ta đặt cho ba cách quét này các tên như sau:

TGP: quét trung tâm (Inorder)

GTP: quét trước (Preorder)

TPG: quét sau (Postorder)

Với cây hình 4.7 nếu sử dụng cách quét TGP chúng ta có kết quả như sau:

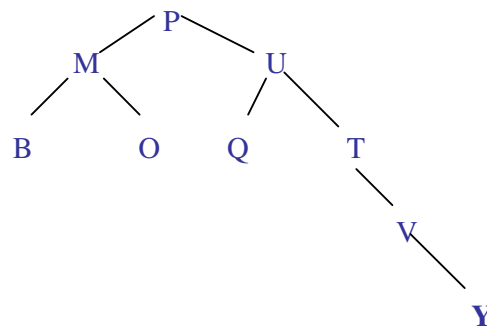
3 4 5 6 7 8 9

Sở dĩ có kết quả như trên là vì chúng ta đã quét theo nguyên tắc: quét cây con bên trái, xử lý dữ liệu ở gốc rồi sau đó quét cây con bên phải. Vì số 6 nằm ở gốc nên nó chưa được xử lý mà phải chuyển sang cây bên trái. Với cây con bên trái, số 4 nằm ở gốc nên lại phải theo nguyên tắc TGP nghĩa là sang bên trái tiếp. Số 3 nằm ở nút lá nên không còn đi đâu nữa kết quả là đầu ra ta có số 3. Quay về G ta có số 4 sau đó sang phải ta có số 5. Xong cây con bên trái ta quay về G và có số 6, tiếp tục với cây con bên phải ta có kết quả đã nêu.

Cách quét TGP sẽ cho ta kết quả là một dãy sắp xếp theo chiều tăng dần, điều này không những đúng với số mà cũng đúng với cây nhị phân chứa các ký tự tại mỗi nút, chúng ta sẽ nghiên cứu kỹ điều này qua ví dụ 4.16.

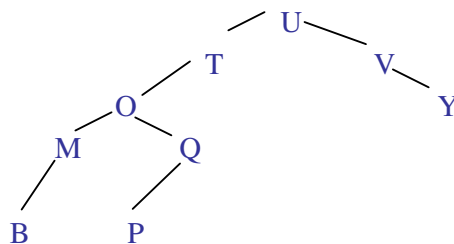
Khi xây dựng cây nhị phân, dữ liệu tại nút gốc mang một ý nghĩa quan trọng, nó sẽ quyết định dữ liệu nhập tiếp theo sẽ nằm ở cây con bên trái hay cây con bên phải.

Giả sử với một tập các ký tự : P, M, U, B, O, Q, T, V, Y nếu chúng ta tạo cây bằng cách đặt ký tự P vào nút gốc và nhập theo thứ tự trên thì cây sẽ có dạng hình 4.8



Hình 4.8

Còn nếu chúng ta đặt U vào gốc và nhập dữ liệu theo thứ tự U, T, V, O, M, Q, Y, B, P thì cây sẽ có dạng như trong hình 4.9



Hình 4.9

Ví dụ 4.16

```
Program Caynhiphan;
Uses crt;
Type
ct=^kytu;
kytu = record
    Nut:char;
    ctt,ctp:ct;
end;
Var chucai:char; goc:ct; i,j:byte;
    A:array[1..100] of char;
    ch:char;

Procedure Taocay (Var tg: ct);
{Chuong trinh con kiem tra xem ky tu dua vao da co trong cay chua
neu chua thi chen vao}
Begin
If tg=nil then
Begin
New(tg);
With tg^ do
Begin
Nut:= chucai;
ctt:=nil; ctp:=nil;
End;
End
Else
With tg^ do
Begin
if chucai<nut then taocay(ctt)
else
if chucai>nut then taocay(ctp)
else
Begin
textcolor(red);
Writeln('Ky tu ',chucai,' da co trong cay-Hay nhap tiep');
end;
End;
End;
```



```

Procedure quet_trung_tam(contro:ct);
Begin
if contro<>nil then
  Begin
  quet_trung_tam(contro^.ctt);
  write(contro^.nut, ' ');
  quet_trung_tam(contro^.ctp);
  end;
End;

```

```

Procedure quet_truoc(contro:ct);
Begin
if contro<>nil then
  Begin
  write(contro^.nut, ' ');
  quet_truoc(contro^.ctt);
  quet_truoc(contro^.ctp);
  end;
End;

```

```

Procedure quet_sau(contro:ct);
Begin
if contro<>nil then
  Begin
  quet_sau(contro^.ctt);
  quet_sau(contro^.ctp);
  write(contro^.nut, ' ');
  end;
End;

```

```

BEGIN
  Clrscr;
  writeln('Xay dung cay nhi phan cac ky tu');
  goc:=nil; i:=1; taocay(goc);
Repeat
  textcolor(14);
  write('Nhap ky tu vao nut - Bam so 9 de dung ');
  readln(chuca); chuca:=upcase(chuca);
  if (ord(chuca)>=65) and (ord(chuca)<=90) then
  Begin

```

```

    taocay(goc);
    a[i]:=chucac;
    i:=i+1;
    end;
Until (Ord(chucac)<65) or (ord(chucac)>90);
writeln;
textcolor(14);
writeln('Du lieu nhap ban dau');
for j:= 1 to i do write(a[j], ' ');
writeln;
writeln('Du lieu hien theo cach quet Trung tam TGP');
quet_trung_tam(goc);
writeln;
writeln('Du lieu hien theo cach quet Truoc GTP');
quet_truoc(goc);
writeln;
writeln('Du lieu hien theo cach quet Sau TPG');
quet_sau(goc);
readln;
END.

```

Chương trình con Taocay trong ví dụ 4.6 tạo nên một cây nhị phân mà dữ liệu tại các nút là các ký tự của bảng mã ASCII, khi bấm một phím bất kỳ trên bàn phím không phải là một trong các chữ cái chương trình tạo cây sẽ kết thúc. Các chương trình con Quet_trung_tam, Quet_truoc, Quet_sau thực hiện việc quét cây theo các trình tự TGP, GTP, TPG.

Nếu chúng ta nhập vào dãy ký tự U, T, V, O, M, Q, Y, V, U thì chương trình sẽ cho kết quả:

```

Du lieu nhap ban dau:
U T V O M Q Y V U

```

```

Du lieu hien theo cach quet Trung tam TGP
B M O P Q T U V Y

```

```

Du lieu hien theo cach quet Truoc GTP
U T O M B Q P V Y

```

```

Du lieu hien theo cach quet Sau TPG
B M P Q O T Y V U

```

Bài tập ứng dụng chương 4

1. Lập chương trình nhập vào mảng A n số thực, n nhập từ bàn phím. Nhập vào chuỗi S tối đa 30 ký tự. Sử dụng phương pháp cấp phát động cho các đối tượng A , S . Cho hiện lên màn hình cách thức bố trí bộ nhớ mà hệ thống dành cho mảng A và chuỗi S .

2. Sử dụng con trỏ mảng và mảng con trỏ để quản lý sách trong thư viện, giả thiết rằng mỗi cuốn sách gồm các thông tin: Tên sách, tên tác giả, số trang, giá tiền. Với cùng một số sách, bộ nhớ mà hệ thống sử dụng cho con trỏ mảng và mảng con trỏ có bằng nhau không?

3. Số liệu tuyển sinh bao gồm:

SBD, Họ và tên, phòng thi, điểm thi.

Sử dụng biến động nhập vào một số thí sinh, cho hiện danh sách thí sinh theo hai cách: hiện theo thứ tự nhập vào và hiện từ cuối lên đầu.

4. Tên người Việt là một chuỗi dài nhất là 7 ký tự. Nhập vào bộ nhớ n tên, sử dụng một trong các cách quét cây để hiện tên theo thứ tự tăng dần.

5. Tạo cây nhị phân mà mỗi nút chứa một số thực. Quét cây theo trình tự GTP, TGP, TPG. Tại mỗi nút yêu cầu xử lý dữ liệu là : hiện giá trị tổng các số trong nút cha và nút con.

6. Dãy số a_1, a_2, \dots, a_n là các số thực. Lập chương trình đưa dãy số trên vào cây nhị phân. Thiết kế 6 chương trình con tương ứng với 6 cách quét cây, yêu cầu xử lý tại mỗi nút là hiện trị tuyệt đối của số trong nút.

Chương 5

Giải thuật đệ quy

Nội dung của chương này đề cập đến những bài toán có tính đệ quy. Không phải bài toán nào cũng có tính đệ quy và không phải các bài toán có tính đệ quy thì đều phải giải bằng giải thuật đệ quy. Các vấn đề cần quan tâm trong chương này:

- Bài toán có tính đệ quy không
- Có cần dùng giải thuật đệ quy không
- Đệ quy có mang lại hiệu quả hơn các phương pháp thông thường hay không

1. Khái niệm đệ quy

Trong thân một chương trình con có thể đưa lời gọi tới chính chương trình con đó, tính chất này được gọi là tính "Đệ quy của chương trình con".

Trong toán học khái niệm giai thừa được định nghĩa như sau:

$$0! = 1$$

$$\text{Nếu } n > 0 \text{ thì } n! = 1 * 2 * 3 * \dots * n$$

Từ định nghĩa trên dễ dàng thấy rằng

$$n! = n * (n-1)!$$

$$(n-1)! = (n-1) * (n-2)!$$

.....

$$1! = 1 * 0! = 1$$

Cách thức lập luận như trên đưa chúng ta tới một nhận xét tổng quát là: lời giải của bài toán A có thể được thực hiện bởi lời giải của bài toán A' có dạng giống như A. Thật vậy, việc tính $n!$ có thể được thực hiện bởi việc tính $(n-1)!$.

Điều quan trọng để bài toán có lời giải là tuy A' giống như A song thực chất A' phải nhỏ hơn A và quá trình "thu nhỏ" phải có điểm dừng. Trong bài toán tính giai thừa từ chỗ cần tính giai thừa của n chúng ta đi tính giai thừa của $(n-1)$, để tính giai thừa của $(n-1)$ chúng ta đi tính giai thừa của $(n-2)$... kết thúc là giai thừa của 0.

Một đối tượng được gọi là đệ quy nếu nó được định nghĩa dưới dạng của chính nó.

Giải thuật của bài toán A được gọi là đệ quy nếu nó dẫn tới việc giải bài toán A' giống như A nhưng nhỏ hơn A và quá trình phải có điểm dừng.

Xét bài toán tính giai thừa với $n = 5$:

$5! = 5 * 4!$	Ô nhớ cuối	$1! = 1$
$4! = 4 * 3!$		$2! = 2 * 1!$
$3! = 3 * 2!$		$3! = 3 * 2!$
$2! = 2 * 1!$		$4! = 4 * 3!$
$1! = 1$	Ô nhớ đầu	$5! = 5 * 4!$

Hình 5.1

Như vậy khi biết $1!$ thì tính được $2!$, biết $2!$ thì tính được $3!$,...

Bằng cách sử dụng bộ nhớ ngăn xếp theo nguyên tắc LIFO (Last In - First Out) (Hình 1.3) những gì gửi vào cuối cùng thì được lấy ra trước tiên. Bóc ô nhớ trên đỉnh ta có $1! = 1$ và lộ ra ô tiếp theo $2! = 2 * 1!$. Vì $1!$ đã biết nên tính được $2! = 2$, bóc tiếp ô nhớ phía dưới ta có $3! = 3 * 2!$, vì $2!$ đã biết nên $3! = 3 * 2 = 6$ quá trình tiếp tục cho đến khi bóc ô nhớ dưới cùng và chúng ta nhận được $5! = 120$. Dưới đây là chương trình tính giai thừa

Ví dụ 5.1:

```
Program Giaithua;
Uses crt;
Var n: integer;
Function GT(m: integer): integer;      (* Hàm GT tính giai thừa của n*)
Begin
if m=0 then GT:=1 else GT:= m*GT(m-1); (*Gọi đệ qui của GT *)
End;
Begin
Clrscr;
write ('Tính giai thừa của n = '); Readln (n) (*Đọc giá trị n*)
write('Giai thừa của ',n, ' = ',GT(n)); (* Viết giá trị hàm GT *)
Repeat until keypressed;
End.
```

Ví dụ 5.1 có một số điều đáng lưu ý sau đây:

1. Hàm GT được xây dựng để tính giai thừa với tham số hình thức m, kiểu của m là kiểu Integer. Giá trị m sau này sẽ được thay thế bằng tham số thực n qua lời gọi GT(n) trong chương trình mẹ.

2. Khi định nghĩa kiểu của GT là Integer thì giá trị của n chỉ được chọn nhỏ hơn 8 vì $8! = 40320$ vượt quá giá trị cực đại của Integer (32767). Để có thể tính giai thừa với $n \geq 8$ ta phải định nghĩa function GT có kiểu Longint hoặc Real. Với kiểu Real lệnh viết giá trị của giai thừa phải là viết số thực với phần lẻ bằng 0, ví dụ:

```
Write (GT(n):12:0);
```

3. Sử dụng thuật toán đệ quy đồng nghĩa với việc phải xây dựng chương trình con, vì vậy nếu sử dụng các vòng lặp có thể giải được bài toán thì nên dùng vòng lặp. Trừ trường hợp bắt buộc phải giải bài toán không có tính lặp hoặc bài toán có khả năng truy hồi.

Với bài toán tính giai thừa có thể dùng vòng lặp và chúng ta thấy chương trình sẽ đơn giản hơn nhiều so với cách dùng tính đệ quy:

Ví dụ 5.2

```
Program Tinh_GT;
Uses crt;
Var i,n:byte; gt:longint;
Begin
Clrscr;
Write(' Nhập gia trị n '); Readln(n);
if (n = 0) or (n=1) then gt:=1;
if n > 1 then gt:=1;
For i:= 1 to n do gt:=gt*i;
Writeln('Giai thừa của ',n, ' = ',gt);
```

```
Readln;  
End.
```

Ví dụ 5.3: Lập chương trình tìm ước số chung lớn nhất của hai số nguyên n, m.

Ước số chung lớn nhất của hai số n và m tính theo công thức :

$$\text{USC}(n,m) = \begin{cases} n & \text{nếu } m = 0 \\ \text{USC}(m, n \bmod m) & \text{nếu } m \neq 0 \end{cases}$$

Ví dụ n= 4, m=8

$\text{USC}(4,8) = \text{USC}(8, 4) = \text{USC}(4,0) = 4$

Chương trình được xây dựng như sau:

```
Program timUSC ;  
Uses crt;  
Var  n,m : word;  Lam: Char;  
FUNCTION USC(a,b:word): word;  
Begin  
If b=0 then USC := a Else USC := USC(b,a mod b);  
End;  
BEGIN  
Repeat  
Clrscr;  
Write(' Hay cho hai so "n" va "m" can tim uoc so chung ');  
Readln(n,m);  
Writeln(' Uoc so chung lon nhat cua ',n,' va ',m,' la ',USC(n,m):5);  
Writeln;  
Write('Tim tiep hay thoi ? C/K '); read(lam);  
Until Uppcase(lam)='K';  
END.
```

Bài toán kinh điển ứng dụng giải thuật đệ quy là bài toán Tháp Hanoi. Nội dung bài toán như sau:

Có n chiếc đĩa tròn đường kính khác nhau, các đĩa có lỗ ở giữa. Người ta xếp lồng các đĩa này vào một cọc theo thứ tự trên nhỏ dưới to. Yêu cầu phải xếp các đĩa này sang cọc mới theo nguyên tắc:

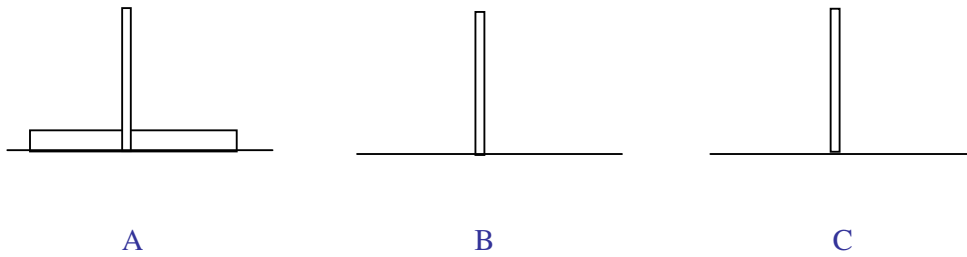
1. Mỗi lần chỉ được chuyển 1 đĩa
2. Không được để xảy ra tình trạng đĩa to ở trên, đĩa nhỏ ở dưới dù là tạm thời
3. Được phép dùng một cọc trung gian

Giả sử cọc đầu tiên là cọc A, cọc trung gian là B và cọc cần xếp đĩa sang là cọc C.
 Chúng ta sẽ xét một số trường hợp đơn giản để tìm quy luật (Hình 5.2)

* $n = 1$: cọc A chỉ có một đĩa

Chuyển đĩa từ cọc A sang cọc C

Kết thúc



Hình 5.2

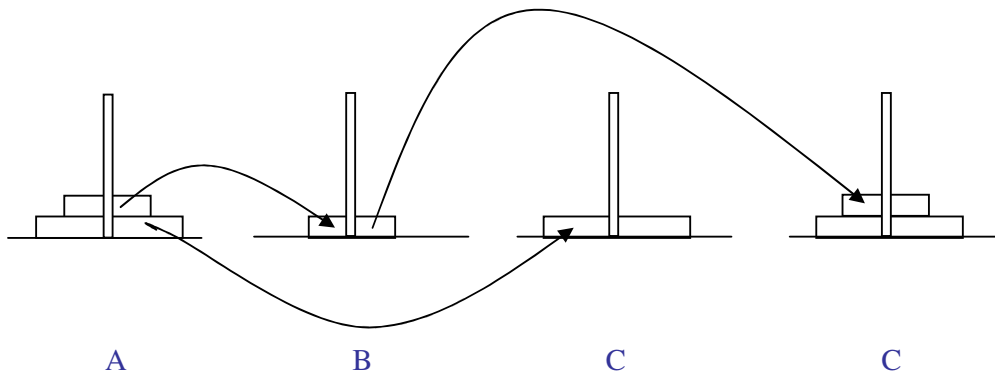
* $n = 2$: cọc A có 2 đĩa (Hình 5.3)

Chuyển đĩa trên sang B

Chuyển đĩa dưới sang C

Chuyển đĩa từ B sang C

Kết thúc



Hình 5.3

* $n = 3$: cọc A có 3 đĩa

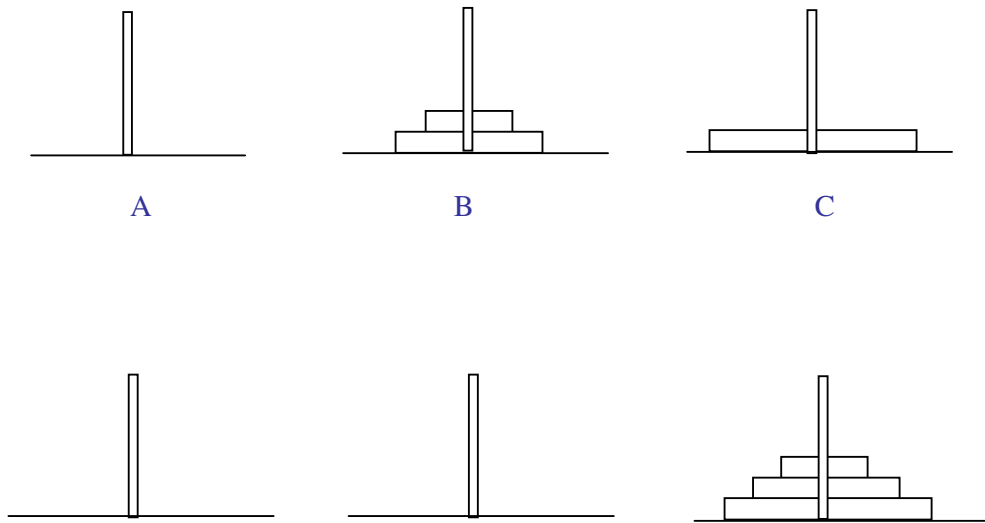
Với trường hợp 3 đĩa nếu chúng ta coi 2 đĩa trên là một đĩa thì bài toán quy về trường hợp $n = 2$, khi đó lời giải sẽ là: (Hình 5.4)

Chuyển 2 đĩa trên sang B

Chuyển đĩa dưới cùng sang C

Chuyển 2 đĩa từ B sang C

Kết thúc



Hình 5.4

Việc chuyển 2 đĩa từ A sang B chúng ta đã có lời giải (trường hợp $n = 2$)

Tóm lại với bài toán n đĩa chúng ta có lời giải tổng quát là:

- Chuyển $n-1$ đĩa trên cùng từ A sang B
- Chuyển đĩa dưới cùng từ A sang C
- Chuyển $n-1$ đĩa từ B sang C
- Kết thúc

Việc chuyển $n-1$ đĩa từ A sang B lại dẫn tới bài toán giống như chuyển n đĩa, song số lượng đĩa đã giảm đi 1, nghĩa là:

- Chuyển $n-2$ đĩa từ A sang B
- Chuyển đĩa thứ $n-1$ từ A sang C
- Chuyển $n-2$ đĩa từ B sang C
- Kết thúc

Quá trình sẽ dẫn tới lúc trên cọc A chỉ còn đĩa thứ n trên cọc B có $n-1$ đĩa và trên cọc C không có đĩa nào, đến đây ta chuyển đĩa thứ n từ A sang C. Bài toán sẽ được thu nhỏ lại với việc chuyển $n-2$ đĩa từ B sang A.

Số lần thực hiện các bước chuyển với $n = 3$ được cho trong bảng 5.1, ở đây chúng ta đã quy ước đĩa trên cùng là đĩa số 1, đĩa giữa là đĩa số 2 và đĩa dưới cùng là đĩa số 3. Dãy số trong các cọc luôn luôn sắp xếp theo chiều tăng dần chứng tỏ rằng đĩa nhỏ luôn ở trên, đĩa to luôn ở dưới.

Bảng 5.1

Bước	ý nghĩa	Cọc A	Cọc B	Cọc C
0		1,2,3		
1	Chuyển hai đĩa trên cùng từ A sang B	2,3		1
2		3	2	1
3		3	1,2	
4	Chuyển đĩa số 3 từ A sang C		1,2	3
5	Chuyển 2 đĩa từ B sang C	1	2	3
6		1		2,3
7				1,2,3

Các ví dụ trên cho thấy số lần chuyển tăng rất nhanh khi số đĩa tăng tuyến tính, với $n=1$ số lần chuyển là 1, với $n=2$ số lần chuyển là 3, với $n=3$ số lần chuyển là 7.

Bảng 5.2

Bước	ý nghĩa	Cọc A	Cọc B	Cọc C
0		1,2,3,4		
1	Chuyển 3 đĩa trên cùng từ cọc A sang cọc trung gian B	2,3,4	1	
2		3,4	1	2
3		3,4		1,2
4		4	3	1,2
5		1,4	3	2
6		1,4	2,3	
7		4	1,2,3	
8	Chuyển đĩa 4 từ A sang C		1,2,3	4
9	Chuyển 2 đĩa trên cùng từ B sang A		2,3	1,4
10		2	3	1,4
11		1,2	3	4
12	Chuyển đĩa 3 từ B sang C	1,2		3,4
13	Chuyển 2 đĩa 1 và 2 từ A sang C	2	1	3,4
14			1	2,3,4
15				1,2,3,4

Có thể dễ dàng tìm ra công thức truy hồi để tính số lần chuyển đĩa

Nếu số đĩa là n thì số lần chuyển là $2^n - 1$

Với $n=4$ số lần chuyển sẽ là 15 (bảng 5.2).

Nếu trên cọc có 32 đĩa thì số lần chuyển đã là hơn 4 tỷ, còn nếu số đĩa là 64 theo như bài toán cổ về tháp Hanoi thì số lần chuyển sẽ là: 36893488147000000000 lần

Giả sử chúng ta cho rằng mỗi lần chuyển một đĩa hết 1 giây thì để chuyển hết số đĩa nói trên từ cọc A sang cọc C chúng ta cần 1 169 884 834 700 năm. Đây là khoảng thời gian mà bản thân hệ mặt trời và dải ngân hà cũng khó có thể tồn tại.

2. Thiết kế giải thuật đệ quy - Khử đệ quy

Với một bài toán toán bất kỳ, để xem nó có tính đệ quy hay không chúng ta phải trả lời được các câu hỏi sau đây:

2.1 Bài toán đã cho có thể được định nghĩa dưới một dạng giống hệt như nó nhưng có kích thước nhỏ hơn hay không?

2.2 Quá trình thu nhỏ bài toán có dẫn tới một điểm dừng nào đó hay không? Nói cách khác bài toán thu nhỏ có dẫn tới một trường hợp đặc biệt mà ta gọi là trường hợp xuy biến nghĩa là biết lời giải hay không?

Với bài toán giai thừa trường hợp xuy biến là $0! = 1$

Với bài toán Tháp Hanoi trường hợp xuy biến là khi chỉ có 1 đĩa trên cọc A.

2.3 Khi xử dụng tính đệ quy mỗi lần thu nhỏ bài toán kích thước bài toán giảm đi như thế nào?

Các ví dụ về đệ quy có thể xem trong nhiều tài liệu về lập trình, vấn đề chúng ta quan tâm ở đây là khi một bài toán có thể giải quyết bằng cả đệ quy và cách thông thường thì nên dùng cách nào?

Với bài toán tính giai thừa sử dụng vòng lặp For... việc lập trình rất đơn giản, còn dùng đệ quy sẽ phức tạp hơn. Với bài toán tìm ước số chung lớn nhất rõ ràng là cách dùng đệ quy sẽ đơn giản hơn các cách khác. Với bài toán Tháp Hanoi nếu không dùng đệ quy chúng ta khó mà tìm ra một thuật giải nào trong sáng hơn. Nói như vậy cũng có nghĩa là không có một chuẩn mực nào chung cho mọi bài toán. Kinh nghiệm của các nhà lập trình đã chỉ ra rằng nếu một bài toán vừa có thể giải bằng đệ quy, vừa có thể giải bằng phương pháp lặp thông thường thì nên tránh dùng đệ quy.

Một bài toán có thể thay thế giải thuật đệ quy bằng các giải thuật không tự gọi đến chúng thì gọi là khử đệ quy.

Ví dụ với bài toán tính giai thừa chúng ta hoàn toàn có thể giải bằng cách dùng vòng lặp For . Bài toán trong trường hợp này đơn giản hơn cả về thuật toán lẫn kỹ thuật lập trình.

3. Hiệu lực của bài toán đệ quy

Như đã nêu đệ quy là một trong các công cụ để lập trình giải các bài toán. Cần khẳng định rằng đệ quy chỉ được dùng cho một số bài toán mà chúng ta tìm được giải thuật đệ quy. Trở lại ví dụ 4.16 về xây dựng cây nhị phân. Nếu cây mà mỗi nút cha có nhiều nhất hai nút con nhưng không quy định rõ con bên trái và con bên phải bố trí thế nào thì chúng ta không thể dùng được tính đệ quy. Trong trường hợp quy định rõ giá trị ở con bên trái luôn nhỏ hơn giá trị ở nút cha còn ở con bên phải là lớn hơn nút cha thì chúng ta thấy việc khảo sát một cây nhị phân tổng quát có thể xuy về việc khảo sát riêng từng cây con bên trái và cây con bên phải. Chương trình con Taocay trong ví dụ 4.16 đã sử dụng tính đệ quy để bổ xung dữ liệu vào các nút, với cách viết này chương trình Taocay không đơn giản hơn cách viết dùng vòng lặp. Tuy nhiên khi duyệt cây với các chương trình con Quet_trung_tam, Quet_truoc, Quet_sau

thì tính đệ quy đã làm cho chương trình trở nên trong sáng và rất dễ hình dung là cây được quét như thế nào.

Tính đệ quy về một khía cạnh nào đó có thể xem như tính quy nạp toán học, tuy nhiên cần phân biệt rằng đệ quy được thực hiện là nhờ các bộ nhớ kiểu LIFO hoặc FIFO còn quy nạp toán học thuần túy chỉ là lý thuyết. Việc sử dụng quy nạp toán học để chứng minh tính đúng đắn của giải thuật đệ quy đã được một vài tác giả thực hiện, điều này chỉ chứng minh tính đúng đắn của giải thuật đệ quy chứ không cho biết giải thuật ấy hiệu quả như thế nào. Vấn đề là người lập trình phải tự xác định xem thuật giải nào tiêu tốn ít công sức của người lập trình và tài nguyên trong máy nhất.

Bài tập chương 5

1. Dãy số Fibonacci a_1, a_2, \dots, a_n được định nghĩa như sau:

$$a_1 = a_2 = 1$$

$$a_3 = a_2 + a_1$$

.....

$$a_n = a_{n-1} + a_{n-2}$$

Lập chương trình với việc áp dụng tính đệ quy để xây dựng dãy số Fibonacci, giá trị n nhập từ bàn phím.

2. Lập chương trình giải bài toán Tháp Hanoi

3. Hàm $F(x,y)$ được định nghĩa như sau:

$$F(x,y) = \begin{cases} y+1 & \text{nếu } x = 0 \\ F(x-1,1) & \text{nếu } y = 0 \\ F(x-1, F(x, y-1)) & \text{với các trường hợp còn lại} \end{cases}$$

Lập chương trình có dùng đệ quy để tính hàm F với giá trị x,y nhập từ bàn phím.

4. S là một chuỗi ký tự có độ dài tối đa là 255. Lập chương trình nhập S từ bàn phím sau đó in ra chuỗi ngược của chuỗi ban đầu. Chương trình có dùng đệ quy

Ví dụ: S = 'Chúc mừng ngày 8/3'

In ra '3/8 yagn gnum cuhC'

5. Cho dãy số a_1, a_2, \dots, a_n . Gọi số hoán vị của dãy số là k.

Lập một chương trình hiện lên số hoán vị của dãy số và giá trị cụ thể của từng hoán vị

Ví dụ: dãy số 1 3 2

In ra: Số hoán vị = 6

Giá trị cụ thể 1 2 3; 1 3 2; 2 1 3; 2 3 1; 3 1 2; 3 2 1

Chương 6

Đồ hoạ

Đồ hoạ trong Pascal không phải là công cụ chuyên dùng để thiết kế hình ảnh. Tuy nhiên nếu biết tận dụng các công cụ sẵn có trong Unit Graph chúng ta có thể làm được nhiều việc, đặc biệt là vẽ đồ thị các hàm số. Trong chương này bạn đọc tiếp cận các khái niệm sau:

- Các thủ tục vẽ hình đơn giản
- Các thủ tục viết chữ trong chế độ đồ hoạ
- Các thủ tục tô màu
- Các phương pháp xử lý ảnh Bitmap
- Phương pháp vẽ đồ thị hàm số

1. Khái niệm chung

Màn hình máy vi tính có thể dùng ở một trong hai chế độ: chế độ TEXT - hiển thị văn bản và chế độ GRAPHIC - hiển thị đồ họa.

Trong chế độ TEXT màn hình thường được chia thành 25 dòng và 80 cột, nếu viết kín màn hình ta có thể viết được 2000 ký tự. Chúng ta có thể thay đổi độ phân giải để viết ra 25 dòng x 40 cột, 50 dòng x 80 cột hoặc 132 dòng x 43 cột.

Muốn vẽ hình, tô màu các hình ta phải chuyển sang chế độ đồ họa, trong chế độ này màn hình được xem là một ma trận điểm, tùy thuộc độ phân giải ta có thể có ma trận 640x480 điểm hoặc 1024x720 điểm.... Mỗi điểm trên màn hình được gọi là 1 Pixel tức là một phần tử ảnh (Picture Element), ta có thể hoàn toàn chủ động trong việc thay đổi màu sắc của từng điểm để tạo ra một bức tranh theo ý muốn. Vị trí của mỗi điểm trên màn hình được biểu diễn bởi hai tọa độ: Hoành độ và Tung độ. Góc tọa độ (0,0) là điểm ở góc trên bên trái màn hình.

Như đã nêu trong chương 1 phần cài đặt Pascal, muốn chuyển sang làm việc ở chế độ đồ họa, trong thư mục hiện hành (thư mục chứa chương trình Pascal) phải có các tệp GRAPH.TPU, *.BGI và *.CHR. Lời gọi đơn vị chương trình đồ họa phải đặt ở đầu chương trình ngay sau từ khoá PROGRAM như ví dụ 6.1.

Ví dụ 6.1

```
Program Ve_hinh;
```

```
Uses GRAPH;
```

```
.....
```

Trong phần thân chương trình cần phải đưa vào các thông báo về kiểu màn hình, chế độ đồ họa (MODE) tương ứng. Những người làm tin học ứng dụng thường không quan tâm lắm đến các thông số này do vậy dễ lúng túng khi cần khai báo. Để khắc phục nhược điểm đó trong Pascal đã thiết kế sẵn một thủ tục khởi tạo chế độ đồ họa là:

Initgraph(var GD,GM: Integer, DP:string[n]);

Khi gọi thủ tục này với các tham số hợp lệ Initgraph sẽ tự xác định kiểu màn hình và Mode đồ họa tối ưu .

Để gọi thủ tục Initgraph cần phải khai báo trước các tham số GD, GM thuộc kiểu Integer (Trong đó GD: Graph Driver - là một số nguyên xác định kiểu màn hình; GM: Graph Mode - cũng là một số nguyên xác định Mode đồ họa).

Nếu ngay sau từ khoá Begin của phần thân chương trình chúng ta khai báo

```
GD:= Detect;
```

thì Initgraph hiểu là nó phải tự đi xác định kiểu màn hình và Mode đồ họa sao cho đạt kết quả tối ưu. Nói chung trừ những trường hợp đặc biệt, chúng ta không nên tự xác định những thông số này làm gì.

Tham số DP (Driver Path) là đường dẫn tới thư mục chứa các tệp điều khiển kiểu màn hình đồ họa, thông thường Pascal được cài đặt trong trong đĩa cứng nên DP sẽ là 'C:\tp\bgi' nghĩa là ổ đĩa C, thư mục con BGI trong thư mục TP.

Ví dụ 6.2 là chương trình vẽ một đường tròn có tâm tại chính giữa màn hình và bán kính là 50 Pixel.

Ví dụ 6.2

```
Program Ve_hinh_tron;  
Uses graph;  
Var GD,DM: Integer;  
Begin  
GD:= detect;  
Initgraph(GD,GM,'c:\tp\bgi');  
If graphresult <> grok then halt(1);  
Circle(320,240,50);  
Readln; CloseGraph;  
End.
```

Dòng thứ 8 trong ví dụ 6.2

```
If graphresult <> grok then halt(1);
```

là câu lệnh kiểm tra giá trị của hàm graphresult (kết quả kiểm tra đồ họa), nếu hàm này nhận giá trị 0 thì đồ họa không có lỗi, chương trình tiếp tục làm việc, còn nếu giá trị của graphresult khác 0 chứng tỏ việc kiểm tra phát hiện ra lỗi, chương trình phải dừng lại.

Bảng 6.1 cho mã lỗi mà hàm graphresult trả về.

Thông thường lỗi xảy ra chủ yếu là do người sử dụng khai báo không đúng tham số GD, nghĩa là không chỉ ra cho Pascal biết các tệp điều khiển đồ họa nằm ở đâu, muốn sửa chữa chỉ cần khai báo lại tham số này theo đúng vị trí mà nó được cài đặt.

2. Một số thủ tục cơ bản để vẽ hình

2.1. MOVETO(x,y) : Di chuyển con trỏ đến tọa độ x,y (x là hoành độ, y là tung độ)
x,y là các giá trị kiểu Integer, với màn hình VGA thì $0 \leq x \leq 639$, $0 \leq y \leq 479$

2.2. LINETO(x,y) : Vẽ một đường thẳng từ vị trí con trỏ hiện thời tới tọa độ x,y kết thúc quá trình vẽ con trỏ nằm tại tọa độ mới.

Pascal có sẵn hai hàm để xác định tọa độ góc dưới bên phải màn hình đó là Getmaxx và Getmaxy. Để vẽ một đường chéo của màn hình từ góc trên bên trái xuống góc dưới bên phải ta có thể viết các lệnh

```
MOVETO(0,0);  
LINETO(Getmaxx,Getmaxy);
```

2.3. LINE(x1,y1,x2,y2) : Thủ tục này vẽ một đường thẳng từ tọa độ x1,y1 đến tọa độ x2,y2

2.4. LINEREL(dX,dY) : Vẽ đường thẳng từ vị trí hiện thời (toạ độ x,y) tới toạ độ x+dx, y+dy.

Bảng 6.1

Mã lỗi	Tên lỗi	Nghĩa của lỗi
0	Grok	Tốt, không có lỗi
1	GrnoInitgraph	Không tìm thấy đơn vị đồ hoạ
2	Grnotdetected	Không có phần cứng đồ hoạ
3	GrFilenotfound	Không tìm thấy các tệp điều khiển đồ hoạ
4	GrInvalidDriver	File điều khiển đồ hoạ (BGI) bị hỏng
5	GrNoloadMem	Không đủ bộ nhớ để nạp trình điều khiển đồ hoạ
6	GrNoscanMem	Không đủ bộ nhớ khi duyệt (kiểm tra)
7	grNoFloodMem	Không đủ bộ nhớ khi kết xuất (đưa ra)
8	grFontNotFound	Không tìm thấy Font (các tệp đuôi CHR)
9	grNoFontMem	Không đủ bộ nhớ để nạp Font
10	grInvalidMode	Sai mod đồ hoạ khi lựa chọn các điều khiển
11	grError	Lỗi đồ hoạ (lỗi tổng quát)
12	grIOerror	Có lỗi vào/ra đồ hoạ
13	grInvalidFont	Lỗi các tệp chứa Font
14	grInvalidFontNum	Lỗi số hiệu Font

2.5. CIRCLE(x,y,r) : vẽ đường tròn tâm tại toạ độ x,y bán kính bằng r Pixel

2.6. PUTPIXEL(x,y, n) : Thủ tục này sẽ vẽ một điểm sáng tại toạ độ x,y với màu là n. Giá trị n lấy trong khoảng 0-15 hoặc viết trực tiếp tên màu theo tiếng Anh.

2.7. RECTANGLE(x1,y1,x2,y2) : Vẽ hình chữ nhật toạ độ góc trên bên trái là x1,y1 , toạ độ góc dưới bên phải là x2,y2.

2.8. BAR(x1,y1,x2,y2) : Vẽ một hình chữ nhật góc trên bên trái có toạ độ x1,y1 góc dưới bên phải có toạ độ x2,y2. Khi dùng kết hợp với thủ tục số 9 sẽ đồng thời cho phép kẻ các vân hoa trên nền và tô màu cho nền.

2.9. SETFILLSTYLE(n1,n2) : Thủ tục định vân hoa và màu nền cho thủ tục BAR.

n1 là một giá trị nguyên với $0 \leq n1 \leq 11$: định kiểu vân hoa

n2 là số hiệu mã màu đã giới thiệu $0 \leq n2 \leq 15$

Trong chương trình khi muốn vẽ hình chữ nhật với màu và vân hoa thì cần đưa thủ tục SETFILLSTYLE(n1,n2) vào trước thủ tục BAR. Giá trị của màu và kiểu vân hoa sẽ được giữ cho đến khi ta định nghĩa lại.

3. Thiết lập màu đồ hoạ

Các thủ tục trình bày trong mục II không có tham số định màu kèm theo. Nếu muốn định màu cho nét vẽ chúng ta có thể dùng hai thủ tục sau đây :

3.1. SETCOLOR(n) : Định màu cho các nét vẽ (n là số hiệu màu)

3.2. SETBKCOLOR(n) : Định màu nền đằng sau nét vẽ

Pascal sử dụng một Byte để lưu trữ các thuộc tính màu, trong đó 4 bit thấp là màu chữ, 3 bit cao là màu nền, bit cao nhất dành cho thuộc tính nhấp nháy. Mã nhị phân 4 bit có thể biểu diễn các số từ 0 (số 0000) đến 15 (số 1111) do đó thuộc tính màu của chữ có 16 giá trị ứng với 16 màu, còn thuộc tính màu nền chỉ có 8 giá trị.

Chú ý:

* Giá trị n trong thủ tục 10 có thể chọn từ 0 đến 15, còn trong thủ tục 11 chỉ được chọn từ 0 đến 7.

* Thay vì chọn số hiệu màu n, Pascal cho phép viết tên hằng màu, như vậy hai lệnh sau đây là tương đương:

```
Textcolor(4);
```

```
Textcolor(Red);
```

Bảng giá trị n và màu tương ứng xem trong mục 8.3 chương III (Bảng 3.1)

Để tô màu cho một điểm chúng ta đưa ngay tham số màu vào thủ tục PutPixel, còn tô màu cho một hình khép kín thì dùng thủ tục SetFillstyle.

3.3. GetBkcolor: cho biết mã màu nền hiện dùng

4. Viết chữ trong chế độ đồ họa

Khi đã chuyển sang làm việc ở chế độ đồ họa ta không thể viết chữ bình thường như trong chế độ văn bản. Muốn viết chữ trên màn hình đồ họa chúng ta sử dụng một số thủ tục sau đây:

4.1. OUTTEXT(chuỗi) : Thủ tục này sẽ cho hiện chuỗi ký tự tại vị trí con trỏ hiện thời. Chuỗi có thể viết trực tiếp hoặc thông qua biến chuỗi như trong ví dụ 6.3 sau đây:

Ví dụ 6.3

```
Uses Graph;
```

```
Var
```

```
chuviet : string[30]
```

```
Begin
```

```
outtext('Cong hoa xa hoi chu nghĩa Viet nam');
```

```
chuviet:='Viet nam dan chu cong hoa';
```

```
outtext(chuviet);
```

```
End.
```

4.2. OUTTEXTXY(x,y,chuoi) : thủ tục này sẽ viết ra chuỗi ký tự tại tọa độ x,y.

4.3. SETTEXTSTYLE(Kiểu chữ, Chiều viết, Kích thước); Xác định kiểu chữ trong Mode đồ họa

Kiểu chữ là một tham số nguyên nhận giá trị trong khoảng 0-10

Chiều viết chỉ nhận 1 trong hai giá trị :

0: chữ nằm ngang; 1: chữ thẳng đứng

Kích thước Là hệ số phóng to chữ có thể chọn từ 0-10

Chú ý:

Các phiên bản Pascal cũ chỉ có 5 kiểu chữ đánh số từ 0 đến 4, phiên bản 7 bổ sung thêm các kiểu từ 5 đến 10 (xem bảng 6.2)

4.4. CLOSEGRAPH; *chấm dứt chế độ đồ họa trở về chế độ văn bản.*

Sau đó muốn quay lại chế độ đồ họa ta lại phải gọi lại INITGRAPH.

Trong một số trường hợp để chuyển nhanh giữa chế độ đồ họa và văn bản chúng ta có thể dùng hai thủ tục sau đây:

RESTORECRTMODE; ngừng chế độ đồ họa chuyển sang chế độ văn bản.

SETGRAPHMODE(n) ; Ngắt chế độ văn bản đã tạo ra bởi Restorecrtmode thiết lập trở lại chế độ đồ họa. Tham số n có thể lựa chọn trong khoảng 0-2. Ví dụ 6.4 dưới đây trình bày cách sử dụng các thủ tục này.

Giá trị	Tên Font	Tệp lưu trữ (đuôi CHR)
0	DefaultFont	Không có
1	TriplexFont	TRIP
2	SmallFont	LITT
3	SansSerifFont	SANS
4	GothicFont	GOTH
5	ScriptFont	SCRI
6	SimplexFont	SIMP
7	TriplexScriptFont	TSCR
8	ComplexFont	LCOM
9	EuropeanFont	EURO
10	BoldFont	BOLD

Bảng 6.2

Ví dụ 6.4

```
Program dohoa_text;  
uses crt,graph;  
var  
gd,gm:integer;  
begin  
gd:=detect;  
initgraph(gd,gm,'c:\tp\bgi');  
if graphresult<>grok then halt(1);  
moveto(0,0); setcolor(5);
```

```

lineto(300,300); delay(2500);
circle(400,300,100); delay(1500);
restorecrtmode;          (* Chuyển về chế độ văn bản *)
gotoxy(20,20);textcolor(9);
write('AAAAAAAAA');
readln;
setgraphmode(2); (* Trở về chế độ đồ hoạ với n=2 cho màn hình VEGA*)
setcolor(blue);
circle(100,100,50);
delay(2000);
restorecrtmode;          (* Chuyển sang chế độ văn bản lần thứ hai*)
textcolor(3);
gotoxy(20,0); write('NNNNNNNNNNNNNNNNNNNN');
readln;
closegraph;             (* Kết thúc chế độ đồ hoạ*)
End.

```

5. Các ví dụ

Việc sử dụng các thủ tục đồ hoạ không có gì phức tạp, với một chút cố gắng bạn có thể vẽ được những hình rất đẹp theo mong muốn. Trong phần này chúng ta sẽ sử dụng các thủ tục đã trình bày để vẽ một số hình.

5.1 Vẽ đồ thị hình Sin

Dưới đây là một chương trình vẽ đồ thị hình sin. Chạy chương trình ta sẽ thấy ba đường hình sin với các biên độ và màu sắc khác nhau.

Ví dụ 6.5

```

Program Do_thi_hinh_sin;
uses graph,crt;
const m=0.1;
Var t3,t4,t1,n,t2,gd,gm:integer;  t,x,y,z:real;
Begin
gd:=detect;
Initgraph(gd,gm,'C:\tp\bgi');
if graphresult<>gok then Halt(1);
x:=0;  t3:=100;  n:=0;  t2:=10;
while t2<=600 do
Begin
setcolor(green);

```

```

y:=sin(x);
t1:=round(y*50);
t3:=round(y*70);
t4:=round(y*100);
t1:=200-t1;
t3:=200-t3;
t4:=200+t4;
moveto(10,200);
lineto(620,200);
line(10,80,10,300);
settextstyle(3,0,3);
outtextxy(610,205,'x');
settextstyle(3,0,3);
outtextxy(15,75,'y');
settextstyle(4,0,3);
setcolor(5);
outtextxy(200,300,'do thi ham sin(x)');
setcolor(12);
moveto(10,200);
putpixel(t2,t1,11);
putpixel(t2,t3,14);
setcolor(red);
putpixel(t2,t4,random(14));
setcolor(12);
delay(5);
x:=x+0.07;
t2:=t2+1;
end;
repeat until keypressed;
t1:=1;
t2:=200;
while t1<=220 do
begin
line(340,240,round(sqrt(440*440-t1*t1)),t1);
t1:=t1+1;
delay(15);
end;
repeat until keypressed;
closegraph;
End.

```

5.2 Vẽ các hình không gian đơn giản

Chương trình sau đây sẽ vẽ một khung chữ nhật bao quanh toàn bộ màn hình, sau đó vẽ hai đoạn thẳng chia màn hình thành 4 phần. Trong góc phần tư thứ nhất vẽ một hình chóp, các nét nhìn thấy vẽ liền, nét khuất vẽ bằng nét đứt. Trong góc phần tư thứ hai vẽ một hình trụ. Xin mời các bạn nhập chương trình và sau đó vẽ thêm vào các góc phần tư còn lại các hình không gian khác, chẳng hạn hình hộp chữ nhật, hình nón...

Ví dụ 6.6

```
Program vehinh;  
Uses crt,graph;  
var i,j:word; gd,gm:integer;  
Begin  
clrscr;  
gd:=detect;  
initgraph(gd,gm,'C:\tp\bgi');  
if graphresult<>gok then halt;  
rectangle(0,0,639,479);  
moveto(320,0); lineto(320,480);  
moveto(0,240); lineto(639,240);  
outtextxy(30,90,'Hinh chop');  
setcolor(4);  
setlinestyle(0,1, normwidth);  
moveto(100,200);  
Linerel(120,0);  
Linerel(50,-40);  
setcolor(2);  
setlinestyle(3,1, normwidth);  
Linerel(-120,0);  
Linerel(-50,40);  
setcolor(4);  
setlinestyle(0,1, normwidth);  
Lineto(185,50);  
Lineto(220,200);  
moveto(270,160);  
Lineto(185,50);  
setcolor(2);  
setlinestyle(3,2, normwidth);  
Lineto(150,160);  
moveto(185,50);  
setcolor(14);
```

```

lineto(185,180);
circle(185,179,2);
moveto(100,200);
lineto(270,160);
moveto(150,160);
lineto(220,200);
ellipse(480,50,0,360,60,25);
ellipse(480,170,180,360,60,25);
setlinestyle(0,15, normwidth);
moveto(420,50); lineto(420,170);
moveto(540,50); lineto(540,170);
setlinestyle(2,15, normwidth);
setcolor(5);
ellipse(480,170,0,180,60,25);
setcolor(9);
outtextxy(330,90,'Hinh tru');
repeat until keypressed;
closegraph;
end.

```

5.3 Viết chữ trên màn hình

Sử dụng các thủ tục đã nêu chúng ta có thể viết chữ trên màn hình đồ họa, ví dụ sau đây sẽ viết lên màn hình dòng chữ "Chúc mừng nam moi" và cho dòng chữ này chạy từ đáy lên đỉnh màn hình. Việc cho một dòng chữ chạy được thực hiện bằng cách viết chữ tại một vị trí nào đó sau đó xoá chữ đi và viết lại chữ tại toạ độ khác. Tuy nhiên trong chế độ đồ họa chúng ta không dùng được lệnh xoá CLRSCR do đó có thể dùng cách viết lại dòng chữ với màu trùng màu nền.

Ví dụ 6.7

```

Program chucchay;
uses crt,graph;
var i,j,gd,gm:integer;
Begin
clrscr;
gd:=detect;
Initgraph(gd,gm,'C:\tp\bgi');
if graphresult<>grok then halt;
settextstyle(0,0,2);
For i:=439 downto 0 do
Begin
setbkcolor(3);

```

```

setcolor(magenta);
outtextxy(180,i,'Chuc mung nam moi');
delay(24);
setcolor(3);
outtextxy(180,i,'Chuc mung nam moi');
end;
closegraph;
end.

```

Chương trình dưới đây thiết kế một đồng hồ ba kim , tốc độ chạy của kim giây tùy thuộc vào lệnh DELAY(n) , nếu chọn DELAY(1000) thì cứ 1 giây kim giây chuyển một vị trí. Khi nhập chương trình vào máy cần lưu ý khai báo lại đường dẫn đến thư mục chứa các tệp *.BGI

Ví dụ 6.8

```

Program Ve_dong_ho;
uses crt,graph;
var
x,y, maxx,maxy, gd,gm,color,i,j,t:integer;
N:real;
LAM,TT:CHAR;
begin
gd:=detect;
initgraph(gd,gm,'c:\tp\BGI');
setcolor(5);
rectangle(30,20,610,450);
rectangle(31,21,609,449);
rectangle(32,22,608,448);
setfillstyle(9,2);
bar(33,23,607,447);
setcolor(red);
setbkcolor(red);
for i:=1 to 10 do circle(320,240,i);
setcolor(11);
setbkcolor(white);
for i:=11 to 80 do circle(320,240,i);
setcolor(14);
setbkcolor(white);
for i:=80 to 160 do circle(320,240,i);
setcolor(white);

```



```

for i:=160 to 200 do circle(320,240,i);
setcolor(11);
circle(320,240,79);
circle(320,240,80);
setcolor(4);
circle(320,240,159);
circle(320,240,160);
settextstyle(3,0,4);
outtextxy(310,40,'XII');
outtextxy(405,60,'I');
outtextxy(470,120,'II');
outtextxy(490,200,'III');
outtextxy(480,290,'IV');
outtextxy(410,370,'V');
outtextxy(310,400,'VI');
outtextxy(210,370,'VII');
outtextxy(135,290,'VIII');
outtextxy(130,210,'IX');
outtextxy(155,130,'X');
outtextxy(220,60,'XI');
setcolor(blue);
Settextstyle(4,0,5);
outtextxy(230,100,'DIAMON');
setcolor(random(14));
for i:=1 to 20 do
circle(320,360,i );
settextstyle(1,0,2);
setcolor(5);
outtextxy(200,450,'Copyright by Dr. Duong Xuan Thanh');
    for i:=1 to 20 do
        begin
            setcolor(random(14));
            circle(320,360,i );
            end;
    for i:=1 to 20 do
        begin
            setcolor(random(14));
            circle(320,360,i );
        end;
end;

```

```

for t:=0 to 12 do {----- Kim gio -----}
begin
setcolor(12);
moveto(320,240);
setlinestyle(0,0,3);
SetWriteMode(xorput);
linerel(round(110*cos((t*30-89)*pi/180)),round(110*sin((t*30-89)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-90)*pi/180)),round(110*sin((t*30-90)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-91)*pi/180)),round(110*sin((t*30-91)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-92)*pi/180)),round(110*sin((t*30-92)*pi/180)));
for i:=0 to 60 do { -----Kim phut -----}
begin
setcolor(12);
moveto(320,240);
setlinestyle(0,0,3);
SetWriteMode(xorput);
linerel(round(130*cos((i*6-89)*pi/180)),round(130*sin((i*6-89)*pi/180)));
moveto(320,240);
linerel(round(130*cos((i*6-90)*pi/180)),round(130*sin((i*6-90)*pi/180)));
moveto(320,240);
linerel(round(130*cos((i*6-91)*pi/180)),round(130*sin((i*6-91)*pi/180)));
(*-----Kim giay-----*)
for j:=0 to 360 do
begin
moveto(320,240);
setlinestyle(0,0,3);
SetWriteMode(XORPut);
setcolor(12);
linerel(round(150*cos((j-90)*pi/180)),round(150*sin((j-90)*pi/180)));
moveto(320,240);
linerel(round(150*cos((j-91)*pi/180)),round(150*sin((j-91)*pi/180)));
delay(1000);
moveto(320,240);
linerel(round(150*cos((j-90)*pi/180)),round(150*sin((j-90)*pi/180)));
moveto(320,240);
linerel(round(150*cos((j-91)*pi/180)),round(150*sin((j-91)*pi/180)));
end;

```

```

moveto(320,240);
linerel(round(130*cos((i*6-89)*pi/180)),round(130*sin((i*6-89)*pi/180)));
moveto(320,240);
linerel(round(130*cos((i*6-90)*pi/180)),round(130*sin((i*6-90)*pi/180)));
moveto(320,240);
linerel(round(130*cos((i*6-91)*pi/180)),round(130*sin((i*6-91)*pi/180)));
end;
moveto(320,240);
linerel(round(110*cos((t*30-89)*pi/180)),round(110*sin((t*30-89)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-90)*pi/180)),round(110*sin((t*30-90)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-91)*pi/180)),round(110*sin((t*30-91)*pi/180)));
moveto(320,240);
linerel(round(110*cos((t*30-92)*pi/180)),round(110*sin((t*30-92)*pi/180)));
end;
repeat until keypressed;
END.

```

6. Xử lý ảnh Bitmap

Mỗi hình vẽ trên một khu vực màn hình được coi như một ảnh và được gọi là Bitmap, các Bitmap được lưu trữ trong bộ nhớ bởi tập hợp các bit. Mỗi ảnh lại bao gồm nhiều phần tử ảnh (Pixel), mỗi Pixel được lưu trữ bởi 4 bit và việc xử lý ảnh thực chất là xử lý từng bit.

Pascal đã thiết kế sẵn một số thủ tục để lưu trữ ảnh hoặc để đưa ảnh ra màn hình:

6.1 Hàm xác định kích thước (dung lượng) ảnh

```
ImageSize(x1,y1,x2,y2);
```

Hàm này cho ta kích thước tính bằng Byte của một ảnh trong miền chữ nhật, góc trên bên trái x1,y1 và góc dưới bên phải là x2,y2.

6.2 Thủ tục gán kích thước vào biến

```
Getmem(biến, Imagesize(x1,y1,x2,y2));
```

Gán vào biến đã lựa chọn kích thước ảnh trong vùng chữ nhật góc trên trái x1,y1 góc dưới phải x2,y2, kích thước được tính bằng Byte

6.3 Thủ tục lưu trữ ảnh

```
GetImage(x1,y1,x2,y2, biến);
```

Cắt ảnh trong miền chữ nhật góc trên trái x1,y1 góc dưới phải x2,y2 vào một biến, đây là biến không định kiểu và phải có kích thước đủ lớn để lưu ảnh, trong nhiều ứng dụng đồ họa biến này thuộc kiểu con trỏ và được xác định kích thước thông qua thủ tục Getmem.

6.4 Thủ tục cho hiện ảnh

PutImage(x, y, biến, phương thức);

Cho hiện ảnh lưu trữ trong "biến" lên màn hình bắt đầu tại tọa độ x,y, "Phương thức" là một số giá trị mà Pascal đã thiết kế nhằm đưa ảnh Bitmap ra màn hình. Dưới đây là bảng các giá trị của "Phương thức"

Tên phương thức	Phép toán tác động
CopyPut hoặc NormalPut	MOV (màu ảnh đè lên trên màu nền)
XorPut	XOR
OrPut	OR
AndPut	AND
NotPut	NOT

Để hiểu rõ hơn về khái niệm "Phương thức" chúng ta cần lưu ý đến các phép toán mà mỗi phương thức xử dụng.

Chẳng hạn khi phương thức là XorPut thì có thể hiểu về ngữ nghĩa Put là đưa ảnh ra, Xor là phép toán trên Bit. Còn về phương diện kỹ thuật số thì XorPut được hiểu như sau:

Nếu điểm ảnh trong Bitmap có màu trắng (số hiệu màu là 15) và nền có màu xanh da trời (số hiệu màu là 1) thì khi đưa ra màn hình điểm ảnh này sẽ có màu vàng bởi vì $15_{10} = 1111_2$, $1_{10} = 0001_2$. Phép toán Xor được thực hiện trên từng bit

$$1111_2 \text{ xor } 0001_2 = 1110_2$$

Số $1110_2 = 14_{10}$ ứng với màu vàng.

Tương tự như vậy nếu chọn phương thức đưa ảnh ra màn hình là AndPut với màu ảnh là xanh da trời (0001), còn nền có màu đỏ (0100) thì:

$$0001_2 \text{ and } 0100_2 = 0000_2$$

nghĩa là ảnh sẽ có màu đen.

Thông thường chúng ta nên dùng "phương thức" là CopyPut nghĩa là màu ảnh sẽ hiện đè lên màu nền. Cách hiện màu này cho ta đúng màu sắc ảnh đã được lưu trữ trong Bitmap. Các phương thức còn lại mặc dù đưa được ảnh ra màn hình nhưng màu sắc không trung thực, không đúng với màu ảnh đã được lưu trữ trong Bitmap.

Xử dụng phương thức XorPut chúng ta có thể cho hiện ảnh và ẩn ảnh một cách dễ dàng. Để thấy rõ hơn chúng ta xét ví dụ sau:

Giả sử ảnh Bitmap được lưu trong một biến con trỏ P, ảnh có màu trắng và nền màn hình có màu đỏ. Thủ tục đưa ảnh ra màn hình bắt đầu từ vị trí tâm màn hình theo phương thức XorPut:

PutImage(320,240,P^,XorPut);

Với thủ tục trên ảnh đưa ra màn hình sẽ có màu xanh ngọc (số hiệu màu là 11) vì

$$1111_2 \text{ xor } 0100_2 = 1011_2 = 11_{10}$$

Nếu bây giờ chúng ta lại đưa tiếp ảnh đó ra màn hình tại cùng vị trí với ảnh đã có thì ảnh trên màn hình sẽ bị xoá, nói đúng hơn là ảnh đưa ra lần thứ hai sẽ có màu trùng với màu nền.

Trở lại ví dụ trên, ảnh trên màn hình đang có màu là 1011 còn ảnh trong Bitmap vẫn là màu 1111, khi đó màu ảnh trên màn hình sẽ là

$$1111_2 \text{ xor } 1011_2 = 0100_2 \quad (\text{nghĩa là màu đỏ trùng màu nền})$$

Từ đây chúng ta có thuật toán vẽ hình chuyển động như sau:

- 1- Vẽ hình tại toạ độ x,y theo phương thức XorPut
- 2- Lưu hình trên màn hình n miligiây
- 3- Xoá hình (bằng cách lặp lại bước 1)
- 4- Thay đổi toạ độ $x := x+dx; \quad y := y+dy;$
- 5- Quay lại bước 1

Chú ý: khi vẽ hình chuyển động cần phải tạo các các điểm dừng để khi không muốn chuyển động có thể ngắt ngay chương trình.

Dưới đây là chương trình vẽ một bầu trời sao, số ngôi sao trên bầu trời là 100 trong đó có 40 ngôi sao nhấp nháy, 20 ngôi sao nhấp nháy nhanh và 20 sao nhấp nháy chậm, các ngôi sao được phân bố ngẫu nhiên. Trong chương trình còn thiết kế một đĩa bay, đĩa bay có 25 kích thước khác nhau, mỗi hình vẽ đĩa bay được cất vào một biến con trỏ P, các biến con trỏ này nằm trong một mảng con trỏ. Các đĩa bay xuất hiện ngẫu nhiên trên bầu trời, chuyển động nhỏ dần và biến mất, sau đó lại hiện lên tại một toạ độ nào đó. Chương trình dừng lại khi bấm một phím bất kỳ.

Với các máy tính có dung lượng bộ nhớ lớn (128 hoặc 256 Mb) chúng ta có thể tăng số hình đĩa bay lên 50 hình khi đó đĩa bay sẽ chuyển động mịn màng hơn.

Ví dụ 6.9

```
Program Bautroi_diabay;  
Uses graph,crt;  
Var  
  i,j,gd,gm:integer; a,x,y,xx,yy:word;  
  p: array[1..30] of pointer;  
  m,n :array[1..40] of word;  
  
  Procedure bautroi(a:word);  
  var i,j:word; m,n :array[1..40] of word; ch:char;  
  
  Begin  
  for j:= 1 to 5 do  
    Begin  
    for i:= 1 to a do  
      begin
```

```

x:=random(639); y:=random(479);
if i<=40 then (*chon 40 ngoi sao nhap nhay*)
begin
m[i]:=x; n[i]:=y; (*ghi toa do sao nhap nhay vao mang*)
end;
putpixel(x,y,random(15));(*hien sao *)
end;
end;
end;

```

```

Procedure nhapnhay;
var i,j:word; k:byte;
Begin
for i:= 1 to 40 do (*chon 40 ngoi sao nhap nhay*)
begin
m[i]:=random(639);
n[i]:=random(479); (*ghi toa do sao nhap nhay vao mang*)
end;
setcolor(random(15));
for j:=0 to 3 do
For i:=1 to 20 do (*20 sao nhap nhay nhanh*)
begin
circle(m[i+j],n[i+j],1);
delay(5);
end;
for j:=0 to 3 do
For i:=21 to 40 do (*20 sao nhap nhay cham*)
begin
setcolor(random(4));
circle(m[i+j],n[i+j],1);
delay(10);
end;
End;

```

```

procedure vedia;
Var xx,yy,x,y,x1,y1,x2,y2:integer; p: array[1..30] of pointer;
r,i,r1,r2,r3,r4,r5: word;
Begin
setcolor(14);setbkcolor(0);
i:=1; x:= 50; y:= 50; {diem dau tien dia bay xuất hiện}

```

```

for r:= 25 downto 1 do {chọn 25 đĩa với các kích thước giảm dần}
Begin
r1:=round(r/2);
r2:=round(r/3);
r3:=round(r/5);
r4:=2*r2;
r5:=round(r/10);
Ellipse(x,y,0,360,r,r1);
Ellipse(x,y-r3,194,346,r,r2);
Line(x+r2,y-r2,x+r4,y-r4);
circle(x+r4,y-r4,r5);
line(x-r2,y-r2,x-r4,y-r4);
circle(x-r4,y-r4,r5);
x1:=x-r;
y1:=y-r4-r5;
x2:=x+r;
y2:=y+r1;
Getmem(p[i], imagesize(x1,y1,x2,y2));(*xac dinh kích thước ảnh*)
getimage(x1,y1,x2,y2,p[i]^); (*cat ảnh vào biến con trỏ*)
putimage(x1,y1,p[i]^,xorput); (*xoa ảnh*)
i:=i+1;
End;
i:=1;
While not keypressed do
Begin (*hien đĩa bay với các kích thước khác nhau*)
putimage(x,y,p[i]^,xorput);
delay(100);
xx:=x; yy:=y;
x:=x+random(25);
y:=y+random(25);
putimage(xx,yy,p[i]^,xorput); (*xoa đĩa bay*)
for j:= 1 to 40 do (*tao sự nhấp nháy của ngôi sao*)
begin
setcolor(random(15));
circle(m[j],n[j],1);
end;
i:=i+1;
if i>25 then {da hien het cac hình đĩa bay}
begin
x:=random(500);

```

```

y:=random(400);
i:=1;
end;
end;
end;

BEGIN {than chương trình chính}
gd:=detect; a:=100; {a là số ngôi sao trên bầu trời}
initgraph(gd,gm,'C:\tp\bgi');
if graphresult<>grok then halt;
bautroi(a); {hiện bầu trời sao}
while not keypressed do
begin
nhapnhay; {thiết kế các ngôi sao nhấp nháy}
vedia; {gọi địa bay}
end;
closegraph;
END.

```

7. Đồ thị hàm số

Giả thiết hàm $y = f(x)$ xác định và liên tục trên đoạn $[a,b]$, cần vẽ đồ thị hàm trong một khu vực nào đó trên màn hình. Hệ tọa độ quen thuộc mà chúng ta sử dụng là hệ xoy với gốc tọa độ nằm ở góc dưới bên trái, trục x hướng sang phải, trục y hướng lên trên.

Màn hình máy vi tính tùy theo kích thước và chế độ làm việc có thể là ma trận điểm từ 640x480 đến 1024x720. Gốc tọa độ của màn hình lại ở góc trên bên trái và trục x hướng theo chiều từ trên xuống dưới. Để thuận tiện cho việc vẽ và quan sát đồ thị chúng ta cần phải chuyển đổi từ tọa độ thực (như trong toán học) sang tọa độ màn hình.

Ký hiệu tọa độ thực của miền vẽ đồ thị điểm góc dưới trái là x_{min} , y_{min} và góc trên phải là x_{max} , y_{max} . Tọa độ màn hình góc trên trái là c_{min} , d_{min} , góc dưới phải là c_{max} , d_{max} .

Một điểm bất kỳ trên đồ thị có tọa độ thực là x , y và tọa độ màn hình là c , d . Có thể chứng minh rằng:

$$c = m \cdot (x - x_{min}) + c_{min}$$

$$d = n \cdot (y - y_{min}) + d_{min}$$

x_{max}, y_{max}



x_{min}, y_{min}

Trong đó:

$$p = (c_{\max} - c_{\min}) / (x_{\max} - x_{\min})$$

$$q = (d_{\min} - d_{\max}) / (y_{\max} - y_{\min})$$

Hiệu số $(d_{\min} - d_{\max})$ mang dấu âm cho biết chiều biến thiên của trục thẳng đứng giữa tọa độ màn hình và tọa độ thực là ngược nhau.

Khi vẽ đồ thị trên màn hình chúng ta sẽ dùng các công thức trên để chuyển đổi tọa độ tính được từ thực tế sang tọa độ màn hình.

Phương pháp vẽ đồ thị là xác định tọa độ từng điểm sau đó nối các điểm với nhau bằng các đoạn thẳng. Để đồ thị trơn tru thì số điểm phải thật nhiều, tuy nhiên chúng ta không thể tăng vô hạn số điểm. Giả sử chia đoạn $[a, b]$ thành n đoạn nhỏ, chúng ta sẽ có $n+1$ điểm đánh số từ 0 đến n . Khi đó chiều rộng mỗi đoạn sẽ là $h = (b - a) / n$.

Hoành độ của một điểm bất kỳ trong tọa độ thực:

$$x_i = a + h * i,$$

và tung độ tương ứng sẽ là $f(x_i)$.

Điều cần chú ý khi chọn các tọa độ thực và tọa độ màn hình là miền trên màn hình phải bao kín miền tọa độ thực. Muốn vậy chúng ta chọn:

$$x_{\min} = a, \quad x_{\max} = b, \quad y_{\min} = \min(y_i), \quad y_{\max} = \max(y_i).$$

Việc khảo sát hàm số $f(x)$ để tìm max và min theo kiểu toán học là không thực tế trong Pascal, do vậy có thể thay thế việc tìm $\min(y_i)$ và $\max(y_i)$ bằng việc lưu các giá trị $f(x_i)$ vào một mảng, sau đó tìm cực trị trong mảng.

Để chương trình có thể ứng dụng cho tất cả các hàm số chúng ta sẽ xây dựng một chương trình con lấy tên là HAM khai báo hàm và tính giá trị của hàm tại các tọa độ x_i . Một chương trình con lấy tên là TOADO để tính các hệ số m, n , và tung độ các điểm chia (tức là giá trị của hàm $f(x)$ tại các tọa độ x_i).

Ví dụ 6.10

```
Program vedothi;
```

```
Uses graph;
```

```
Const n=200;
```

```
cmin=3; dmin=3; cmax=200; dmax=140; {ve o goc phan tu thu nhat}
```

```
xmin=-5; xmax=5;
```

```
Var gd, gm: integer;
```

```
ymin, ymax, h, m, p, q: real;
```

```
y: Array[0..n] of real;
```

```
Function F(x: real) : real;
```

```
Begin
```

```
{f:=x*x;} 
```

```
F:=x*x*x-2*x;
```

```
End;
```

```
Procedure Toado;
```

```
Var i: integer; xi, yi: real;
```

```
Begin
```

```
h:=(xmax-xmin)/n;
```

```
For i:= 0 to n do
```

```
Begin
```

```

xi:=xmin+i*h;
yi:=f(xi);
y[i]:=f(xi);
End;
ymin:=y[0]; ymax:=-ymin;
For i:=1 to n do
Begin
If ymin>y[i] then ymin:=y[i];
If ymax<y[i] then ymax:=y[i];
End;
p:= (cmax-cmin)/(xmax-xmin);
q:= (dmin-dmax)/(ymax-ymin);
End;

Procedure Ve;
Var i,c,d:integer; xi:real;
Begin
c:=cmin;
d:= round(q*(y[0]-ymin)+dmax);
Moveto(c,d);
For i:=1 to n do
Begin
xi:=xmin+i*h;
c:=Round(p*(xi-xmin)+cmin);
d:=Round(q*(y[i]-ymin)+dmax);
setcolor(red);
Lineto(c,d) {noi cac diem do thi}
End;
setcolor(14);
moveto(cmin,dmin); Lineto(cmax,dmin);{ve khung do thi}
moveto(cmax,dmin); Lineto(cmax,dmax);
moveto(cmax,dmax); Lineto(cmin,dmax);
moveto(cmin,dmax); Lineto(cmin,dmin);
d:=Round(q*(-ymin)+dmax);
if (d>=dmin) and (d<=dmax) then
Begin
line(cmin,d,cmax,d);{truc hoanh}
line(cmax,d,cmax-6,d-3); {dau mui ten}
line(cmax,d,cmax-6,d+3);
end;
c:=Round(p*(-xmin)+cmin);
if (c>=cmin) and (c<=cmax) then
begin
Line(c,dmin,c,dmax); {truc tung}
Line(c,dmin,c-3,dmin+6); {dau mui ten}
Line(c,dmin,c+3,dmin+6);
end; End;

```

```
BEGIN
  gd:=detect;
  Initgraph(gd,gm,'c:\tp70\bgi');
  if graphresult<>grok then halt;
  toado;
  ve;
  readln;
  closegraph;
END.
```

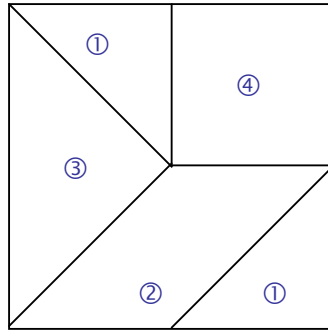
Với các hàm số cho theo tham số $x = x(t)$, $y = y(t)$, hoặc cho trong tọa độ cực $r = f(\varphi)$ phương pháp vẽ cũng tương tự, chúng tôi xin dành cho bạn đọc như là những bài tập ứng dụng.

Bài tập ứng dụng chương 6

1. Nhập và cho chạy thử chương trình trong ví dụ 6.9 sau đó thiết kế thêm một tên lửa. Cho tên lửa phóng từ góc dưới bên phải màn hình lên góc trên bên trái. Luồng khói phía đuôi dài bằng 3 lần chiều dài tên lửa.

2. Cho các ký tự của dòng chữ "Happy Birth Day" hiện ngẫu nhiên và nhấp nháy trên màn hình, tiếp đó cho chúng chuyển động đan xen và thu dần về tâm (320,240). Cuối cùng các ký tự lại từ tâm chạy ra trên một quỹ đạo tròn và đã sắp xếp thành dòng chữ hoàn chỉnh.

3. Chọn kích thước và thiết kế các hình đơn giản 1, 2, 3, 4 (xem hình vẽ), mỗi hình cắt vào một biến con trỏ. Cho hiện các hình tại các vị trí tùy ý sao cho hình vừa chuyển động vừa tự quay, cuối cùng các hình tự ghép với nhau thành một hình vuông ở giữa màn hình



4. Lập chương trình vẽ đồ thị hàm số cho dưới dạng tham số
 $x = x(t)$, $y = y(t)$

5. Lập chương trình vẽ đồ thị hàm số cho dưới dạng tọa độ cực
 $r = f(\varphi)$

6. Cho hàm $y = f(x)$ xác định và liên tục trong khoảng (a, b) . Lập chương trình nhập dạng một hàm cụ thể. Vẽ đồ thị hàm sau đó tính tích phân xác định

$$\int_a^b f(x) dx$$

Phụ lục 1

Bảng mã ASCII

Ngôn ngữ Pascal sử dụng các ký tự trong bảng mã ASCII để xây dựng nên các từ khoá, tên chuẩn, các hàm, thủ tục hoặc các lệnh có cấu trúc. Bảng mã ASCII gồm 256 ký tự được đánh số từ 0 đến 255 và được phân thành các nhóm ký tự sau:

1. Các ký tự điều khiển:

32 ký tự đầu tiên (từ 0 đến 31) là các ký tự điều khiển, các ký tự này không in được ra màn hình hoặc máy in, ví dụ: ký tự Enter (mã 13), ký tự ESC (mã 27).

2. Các ký tự đặc biệt:

Các ký tự có mã ASCII trong khoảng 32-47, 58-64, 91-96, và 123-127 là các ký tự đặc biệt như dấu chấm, dấu phẩy, ký tự @, ký tự Del,...

3. Nhóm chữ số:

Các ký tự có mã ASCII trong khoảng 48-57 là 10 chữ số thập phân 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

4. Nhóm chữ cái:

+ Các ký tự có mã ASCII trong khoảng 65-90 là các chữ cái in từ A đến Z

+ Các ký tự có mã ASCII trong khoảng 97-122 là các chữ cái nhỏ từ a đến z

Các chữ cái thường có mã ASCII lớn hơn 32 so với chữ in tương ứng. Ví dụ mã ASCII của a là 97 còn của A là 65.

5. Nhóm ký tự đồ hoạ

Các ký tự có mã ASCII trong khoảng 128-255 là các ký tự đồ hoạ.

Để hiện toàn bộ các ký tự của bảng mã lên màn hình (trừ 32 ký tự đầu) chúng ta có thể cho chạy chương trình sau:

```
Program Ma_ascii;  
  Var n : Byte;  
  Begin  
    For n:=32 to 255 do Write(chr(n),' ');  
  End.
```

Dưới đây là bảng mã ASCII và hình dạng của 128 ký tự đầu tiên.

Bảng mã ASCII

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
0	NUL	33	!	66	B
1	SOH	34	“	67	C
2	STX	35	#	68	D
3	ETX	36	\$	69	E
4	EOT	37	%	70	F
5	ENQ	38	&	71	G
6	ACK	39	,	72	H
7	BEL	40	(73	I
8	BS	41)	74	J
9	HT	42	*	75	K
10	LF	43	+	76	L
11	VT	44	,	77	M
12	FF	45	-	78	N
13	CR	46	.	79	O
14	SO	47	/	80	P
15	SI	48	0	81	Q
16	DLE	49	1	82	R
17	DC1	50	2	83	S
18	DC2	51	3	84	T
19	DC3	52	4	85	U
20	DC4	53	5	86	V
21	NAK	54	6	87	W
22	SYN	55	7	88	X
23	ETB	56	8	89	Y
24	CAN	57	9	90	Z
25	EM	58	:	91	[
26	SUB	59	;	92	\
27	ESC	60	<	93]
28	FS	61	=	94	^
29	GS	62	>	95	_
30	RS	63	?	96	‘
31	US	64	@	97	a
32	SPACE	65	A	98	b

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
99	c	109	m	119	w
100	d	110	n	120	x
101	e	111	o	121	y
102	f	112	p	122	z
103	g	113	q	123	{
104	h	114	r	124	
105	i	115	s	125	}
106	j	116	t	126	~
107	k	117	u	127	DEL
108	l	118	v		

Phụ lục 2

Tóm tắt các thủ tục và hàm của Turbo Pascal 7.0

Toàn bộ các thủ tục, hàm và định hướng biên dịch của Pascal có thể tìm hiểu trong phần Help trên thực đơn chính. Tổng cộng có 65 trang màn hình, mỗi trang được chia đôi và có 21 dòng tức là có khoảng 2500 mục cần nghiên cứu. Số lượng này ngay cả với những người chuyên nghiệp cũng không thể nắm bắt hết được. Trong phụ lục này chúng tôi chỉ chọn ra một số nhỏ những hàm và thủ tục thông dụng, khi cần biết ý nghĩa và cách dùng của một hàm hay thủ tục nào đó bạn đọc chỉ cần viết tên hàm hay thủ tục đó lên màn hình soạn thảo Pascal sau đó bấm tổ hợp phím Ctrl - F1, Pascal sẽ hiện lên các chỉ dẫn và ví dụ minh họa.

1. Hàm ABS

Cú pháp: Abs(r: Real) : Real; Abs(i: Integer) : Integer;

Công dụng: Cho giá trị tuyệt đối của đối số.

2. Hàm ADDR

Cú pháp: Addr(Var Variable) : Pointer;

Công dụng: Cho địa chỉ của biến.

3. Thủ tục APPEND

Cú pháp: Append(VAR F : Text);

Công dụng: Mở tệp văn bản để ghi bổ sung và định vị con trỏ tại cuối tệp.

4. Thủ tục ARC(Graph Unit)

Cú pháp: Arc(x,y:integer;gd,gc,R:word);

Công dụng: Vẽ cung tròn tâm(x,y) bán kính R, từ góc gd đến góc gc.

5. Hàm ARCTAN

Cú pháp: Arctan(r: Real) : Real;

Công dụng: Cho Arctangent của đối r.

6. Thủ tục ASSIGN

Cú pháp: Assign(var F : file; Name String);

Công dụng: Liên kết biến tệp F với tệp có tên chỉ định trong Name.

7. Thủ tục ASSIGNCRT(CRT unit)

Cú pháp: AssignCrt(Var F : text);

Công dụng: Thay việc gửi kết quả ra màn hình bằng việc lại đưa vào tệp F.

8. Thủ tục BAR(Graph unit)

Cú pháp: Bar(x1,y1,x2,y2:integer);

Công dụng: Vẽ vùng chữ nhật góc trên trái (x1,y1) góc dưới phải (x2,y2).

9. Thủ tục BARD3(Graph Unit)

Cú pháp: Bar3d(x1,y1,x2,y2:integer; Depth:word; Top:Boolean);

Công dụng: Vẽ hình hộp 3 chiều có tô màu. Chiều sâu là Depth Pixel (điểm ảnh). Nếu Top là True thì hộp có nắp, nếu Top là False thì hộp không nắp.

10. Thủ tục BLOCKREAD

Cú pháp: Blockread(Var F:file; Var B:type; N:Integer; <KQ:Word>);

Công dụng: Đọc N mẫu tin từ tệp không kiểu F vào bộ nhớ (qua biến B). KQ cho biết số mẫu tin thực sự đọc được.

11. Thủ tục BLOCKWRITE

Cú pháp: Blockwrite(Var F:file; Var B:type; N:Integer);

Công dụng: Ghi N mẫu tin từ vùng đệm B ra tệp không kiểu F.

12. Thủ tục CHDIR

Cú pháp: ChDir(S:String);

Công dụng: Chuyển đổi thư mục hiện thời sang thư mục có đường dẫn S.

13. Hàm CHR

Cú pháp: Chr(I: Integer);

Công dụng: Cho ký tự có mã ASCII bằng I.

14. Thủ tục CIRCLE (Graph Unit)

Cú pháp: Circle(x,y:Integer; R:word);

Công dụng: Vẽ đường tròn tâm(x,y) bán kính R.

15. Thủ tục CLEARDEVICE(Graph Unit)

Cú pháp: ClearDevice;

Công dụng: Xoá màn hình đồ hoạ.

16. Thủ tục CLEARVIEWPORT(Graph Unit)

Cú pháp: Clearviewport;

Công dụng: Xóa Viewport hiện hành.

17. Thủ tục CLOSE

Cú pháp: Close(Var F:File);

Công dụng: Đóng tệp.

18. Thủ tục CLOSEGRAPH(Graph Unit)

Cú pháp: CloseGraph;

Công dụng: Ngừng chế độ đồ họa, chuyển về chế độ Text. Thủ tục này cũng giải phóng vùng nhớ do hệ đồ họa sử dụng.

19. Thủ tục CLREOL(CRT Unit)

Cú pháp: Clreol;

Công dụng: Xóa ký tự từ vị trí con trỏ (cursor) đến cuối dòng hiện hành.

20. Thủ tục CLRSCR(CRT Unit)

Cú pháp: Clrscr;

Công dụng: Xóa màn hình trong chế độ văn bản và đưa con trỏ về vị trí(1,1).

21. Hàm CONCAT

Cú pháp: Concat(S1,S2,...,Sn:String);

Công dụng: Tạo chuỗi mới bằng cách nối các chuỗi S1,..., Sn. Nếu chiều dài của chuỗi kết quả lớn hơn 255 ký tự, Pascal sẽ cắt bỏ các ký tự từ 256 trở đi.

22. Hàm COPY

Cú pháp: Copy(S:String; P,L:Integer):String;

Công dụng: Trích từ chuỗi S một chuỗi con dài l ký tự bắt đầu từ ký tự thứ P.

23. Hàm COS

Cú pháp: Cos(X:Real): Real;

Công dụng: Cho cosine của góc X.

24. Hàm CSEG

Cú pháp: Cseg: word;

Công dụng: Cho địa chỉ của đoạn mã chương trình.

25. Thủ tục DEC

Cú pháp: Dec(Var X: SCalar; n:longInt);

Công dụng: Giảm biến X đi n. Nếu không có n thì X sẽ bị giảm đi 1.

26. Thủ tục DELAY(CRT Unit)

Cú pháp: Delay(s:word);

Công dụng: Tạm dừng chương trình trong s miligiây , s<=65535

27. Thủ tục DELETE

Cú pháp: Delet(S:String; P,L:integer);

Công dụng: Xoá L ký tự từ vị trí P ra khỏi chuỗi S

28. Thủ tục DELLINE(CRT Unit)

Cú pháp: Delline;

Công dụng: Xoá dòng chứa con trỏ. Các dòng phía dưới sẽ dồn lên phía trên.

29. Thủ tục DETECTGRAPH(CRT Unit)

Cú pháp: DetectGraph(Var GD,GM:Integer);

Công dụng: Phát hiện kiểu màn hình, mode đồ hoạ và gán cho các biến GD,GM.

30. Hàm DISKFREE(DOS Unit)

Cú pháp: DiskFree(Drive:word):longint;

Công dụng: Cho biết số byte còn trống trên ổ đĩa chỉ định trong Drive. Drive = 0 ứng với ổ ngầm định, Drive = 1 cho ổ A, 2 cho ổ B.

31. Hàm DISKSIZE(DOS Unit)

Cú pháp: Disksize(Drive:word):longint;

Công dụng: Cho biết dung lượng đĩa tính theo byte. Tham số Drive xác định ổ đĩa (Drive = 1 cho ổ A, 2 cho ổ B, 0 cho ổ ngầm định).

32. Thủ tục DISPOSE

Cú pháp: Dispose(P:Pointer);

Công dụng: Giải toả vùng heap cấp phát cho biến con trỏ P. Dispose dùng kèm với New.

33. Hàm DOSEXITCODE(DOS Unit)

Cú pháp: Dosexitcode:word;

Công dụng: Cho mã lỗi của tiến trình con (0 = chấm dứt bình thường, 1 = chấm dứt do CTRL - C, 2 = chấm dứt do lỗi thiết bị, 3 = chấm dứt do thủ tục KEEP).

34. Hàm DOSVERSION(DOS Unit)

Cú pháp: Dosversion : word;

Công dụng: Cho số hiệu phiên bản (Version) của hệ điều hành. Phần chính nằm ở byte cao, phần phụ nằm ở byte thấp.

35. Thủ tục DRAWPOLY(Graph Unit)

Cú pháp: Drawpoly(n:word; Var Polypoints);

Công dụng: Vẽ đa giác n đỉnh. Mảng Polypoints chứa tọa độ các đỉnh của đa giác.

36. Hàm ENVCOUNT(DOS Unit)

Cú pháp: EnvCount:integer;

Công dụng: Cho số lượng chuỗi được định nghĩa trong môi trường DOS

37. Hàm EOF

Cú pháp: Eof(F:File): Boolean;

Công dụng: Cho giá trị TRUE nếu con trỏ định vị ở cuối tệp, các trường hợp khác cho kết quả FALSE.

38. Hàm EOLN

Cú pháp: Eoln(F:File): Boolean;

Công dụng: Cho giá trị TRUE nếu con trỏ nằm cuối dòng(CR/LF) hoặc ở cuối tệp, cho FALSE trong trường hợp trái lại.

39. Thủ tục ERASE

Cú pháp: Erase(F:File);

Công dụng: Xoá tệp và các thông tin về tệp F khỏi đĩa.

40. Thủ tục EXEC

Cú pháp: Exec(Path,Cmdline:String);

Công dụng: Thực hiện tệp có tên nằm trong path với các tham biến dòng lệnh được định nghĩa trong Cmdline.

41. Thủ tục EXIT

Cú pháp: Exit;

Công dụng: Thoát khỏi khối hiện thời (khối có thể là chương trình con, cấu trúc lặp) nếu exit nằm trong chương trình chính thì thoát khỏi chương trình).

42. Hàm EXP

Cú pháp: EXP(X:Real):Real;

Công dụng: Lũ thừa e của đối số X.

43. Hàm FEXPAND(DOS Unit)

Cú pháp: Fexpand(P:PathStr):PathStr;

Công dụng: Cho đường dẫn của tệp P. Ví dụ tệp A1.pas hiện lưu trong thư mục c:\tp\bin. Trong phần khai báo chương trình có lệnh: Uses Dos; Khi đó lệnh Writeln(Fexpand('a1.pas')) cho kết quả là C:\TP\BIN\A1.PAS

44. Hàm FILEPOS

Cú pháp: FilePos(F:File):integer;

Công dụng: Cho số hiệu bản ghi của tệp F mà con trỏ đang trỏ tới.

45. Hàm FILESIZE

Cú pháp: FileSize(F:file):integer;

Công dụng: Cho số lượng bản ghi đang có trong tệp F.

46. Thủ tục FILLELIPES(Graph Unit)

Cú pháp: Fillellipse(x,y:integer; a,b:word);

Công dụng: Vẽ ellipse có tâm(x,y) với hai bán trục là a, b.

47. Thủ tục FILLPOLY(Graph Unit)

Cú pháp: Fillpoly(n:word; Var Polypoints);

Công dụng: Vẽ đa giác n đỉnh. Mảng Polypoints chứa tọa độ các đỉnh của đa giác.

48. Thủ tục FLOODFILL(Graph Unit)

Cú pháp: FloodFill(x,y:integer; border:word);

Công dụng: Tô màu một vùng khép kín trên màn hình đồ họa nếu tọa độ(x,y) nằm bên trong vùng. Nếu x,y nằm ngoài vùng thì miền bên ngoài được tô.

49. Thủ tục FLUSH

Cú pháp: Flush(Var F:Text);

Công dụng: Đẩy ra đĩa tất cả kết quả đang nằm trong vùng đệm cho tệp F.

50. Hàm FRAC

Cú pháp: Frac(x:Real):Real;

Công dụng: Cho phần lẻ thập phân của đối số x.

51. Thủ tục FREEMEM

Cú pháp: Freemem(Var F:pointer; I:Integer);

Công dụng: Giải phóng I byte của vùng heap đã cấp phát cho biến con trỏ F bằng thủ tục GetMem.

52. Thủ tục GETARCCOORDS(Graph Unit)

Cú pháp: Getarccoords(Var ArcCoords : ArcCooType);

Công dụng: Cho biết các tọa độ đã được lệnh Arc hay Ellipse trước đó sử dụng. Cấu trúc của ArcCoordsType như sau:

TYPE

ArcCoordstype = Recode

x, y, XStard, YStart, Xend, Yend:Integer;

End;

53. Thủ tục GETASPECTRATIO(Graph Unit)

Cú pháp:

GetAspectratio(Var Xasp, Yasp:word);

Công dụng: Cho độ phân giải của màn hình đồ hoạ (chứa trong các biến Xasp và Yasp).

54. Hàm GETBKCOLOR(Graph Unit)

Cú pháp: GetBKColor : word;

Công dụng: chọn màu nền , có thể thay thế số hiệu màu bởi tên hằng màu.

55. Hàm GETCOLOR(Graph Unit)

Cú pháp: GetColor:word;

Công dụng: Chọn màu nét vẽ, có thể thay thế số hiệu màu bởi tên hằng màu.

56. Thủ tục GETDATE(DOS Unit)

Cú pháp: GetDate(Var year, Month, Day, Dayofweek:word);

Công dụng: Cho ngày tháng hiện hành của hệ thống.

57. Thủ tục GETDEFAULTPALETTE(Graph Unit)

Cú pháp: Getdefaultpalette(Var Pal:PaletteType);

Công dụng: Cho trong Pal bảng màu Ngầm định cho trình điều khiển đồ hoạ hiện hành.

Cấu trúc của kiểu PaletteType như sau:

TYPE

PaletteType = Record

Size:Byte;

Colors: Array[0..Maxcolor] of Shortint;

End;

58. Thủ tục GETDIR

Cú pháp: GetDir(d: Byte; Var S: String);

Công dụng: Cho biết thư mục hiện hành trên ổ đĩa được chỉ định trong d. Tên thư mục được cho trong S (nếu d = 0 thì GetDir sẽ tìm trong ổ đĩa ngầm định, còn nếu d = 1, 2, 3 ứng với các ổ đĩa A, B, C).

59. Hàm GETDRIVERNAME(Graph Unit)

Cú pháp: GetDriverName : String;

Công dụng: Cho tên của trình điều khiển đồ hoạ hiện hành.

60. Thủ tục GETFATTR(DOS Unit)

Cú pháp: GetFAttr(Var F; Var Attr: word);

Công dụng: Cho biết thuộc tính của tệp F. Trước khi gọi thủ tục này tệp phải được gán thuộc tính nhưng không mở.

61. Thủ tục GETFILLPATTERN(Graph Unit)

Cú pháp: GetFillPattern(Var FP : FillpatternType);

Công dụng: Cho trong FP số hiệu của mẫu tô hiện hành (màu nền chỉ có 8 giá trị). Cấu trúc của FillPartternType là:

TYPE

FillPartternType = Array[1..8] of Byte

62. Thủ tục GETFILLSETTINGS(Graph Unit)

Cú pháp: GetFillSettings(Var FS : FillSettingsType);

Công dụng: Cho trong FS màu và mẫu tô hiện hành. Cấu trúc của FillSettingsType là:

TYPE

FillSettingsType = Record

Pattern : word; Color : word;

End;

63. Thủ tục GETFTIME(DOS Unit)

Cú pháp: GetFTime(Var F; Var Time : Longint);

Công dụng: cho biết ngày tháng và giờ mà tệp F được ghi.

64. Hàm GETGRAPHMODE(Graph Unit)

Cú pháp: GetGraphMode : Integer;

Công dụng: cho biết kiểu đồ hoạ hiện hành.

65. Thủ tục GETIMAGE(Graph Unit)

Cú pháp: Getimage(x1,y1,x2,y2 : integer; Var Bitmap);

Công dụng: lưu trữ phần hình chữ nhật (x1,y1,x2,y2) trên màn hình đồ hoạ vào BitMap.

66. Thủ tục GETLINESETTINGS(Graph Unit)

Cú pháp: GetlineSettings(Var LST : LineSettingsType);

Công dụng: Cho trong LST giá trị hiện hành của dạng đường , mẫu, bề dày. Cấu trúc của LineSettingsType như sau:

TYPE

LineSettingsType = Record

LineStyle , Pattern , Thickness : word;

End;

67. Hàm GetMaxcolor(Graph Unit)

Cú pháp: GetMaxcolor : word;

Công dụng: cho số hiệu cao nhất của mã màu trong bảng màu hiện hành.

68. Hàm GETMAXMODE(Graph Unit)

Cú pháp: GetMaxMode : word;

Công dụng: Hàm cho giá trị là một số kiểu word, số này ứng với kiểu đồ họa có độ phân giải cao nhất của màn hình đang dùng. Hầu hết các màn hình có kiểu đồ họa từ 0 đến Getmaxmode.

69. Hàm GETMAXX(Graph Unit)

Cú pháp: GetMaxX : integer;

Công dụng: cho hoành độ lớn nhất của màn hình đồ họa hiện hành.

70. Hàm GETMAXY(Graph Unit)

Cú pháp: GetMaxY : Integer;

Công dụng: cho tung độ lớn nhất của kiểu màn hình đồ họa hiện hành.

71. Thủ tục GETMEM

Cú pháp: GetMem(Var P : Pointer; I: integer);

Công dụng: Cấp phát cho biến con trỏ P một vùng nhớ có dung lượng I byte.

72. Hàm GETMODENAME(Graph Unit)

Cú pháp: GetModeName(ModeNumber : word) : String;

Công dụng: cho một chuỗi mô tả kiểu đồ họa ứng với ModeNumber.

73. Thủ tục GETMODERANGE(Graph Unit)

Cú pháp: GetModeRange(GraphDriver:Integer; Var Lomode, Himode:Integer);

Công dụng: Cho kiểu phân giải thấp nhất(LoMode) và cao nhất (HiMode) của trình điều khiển đồ họa ứng với GraphDriver.

74. Thủ tục GETPALETTE(Graph Unit)

Cú pháp: GetPalette(Var P : PaletteType);

Công dụng: trả về giá trị là một biến kiểu bản ghi gồm 2 trường trường size cho biết số lượng màu trong Palette, trường color cho biết mã màu hiện hành. PaletteType là kiểu Record được định nghĩa như sau:

```
TYPE
  PaleteType = Record
    Size : Byte;
    Colors : Array[0..15] of Shortint;
  End;
```


75. Hàm GETPALETTE_SIZE(Graph Unit)

Cú pháp: GetPaletteSize : word;

Công dụng: Cho số lượng sơ đồ màu lớn nhất mà kiểu màn hình đồ họa hiện hành có thể cung cấp. Ví dụ màn hình EGA hàm này cho kết quả là 16.

76. Hàm GETPIXEL(Graph Unit)

Cú pháp: GetPixel(x,y:Integer);

Công dụng: cho màu của điểm ảnh tại tọa độ(x,y).

77. Thủ tục GETTEXTSETTINGS(Graph Unit)

Cú pháp: GetTextSettings(Var TS : GettextsettingsType);

Công dụng: Cho biết các giá trị định dạng văn bản hiện hành. Các giá trị này lưu vào biến TS kiểu bản ghi. Cấu trúc của bản ghi là:

TYPE

 GetTextSettingsType = Record;

 Font, Direction, Charsize, Horiz, Vert:word;

 End;

78. Thủ tục GETTIME(DOS Unit)

Cú pháp: GetTime(Var Hour,Minute,Second,Sec100:word);

Công dụng: Cho giờ hệ thống.

79. Hàm GETX(Graph Unit)

Cú pháp: Getx : Integer;

Công dụng: Cho tọa hoành độ của con trỏ hiện hành.

80. Hàm GETY(Graph Unit)

Cú pháp: Gety : Integer;

Công dụng: Cho tung độ của con trỏ hiện hành.

81. Thủ tục GOTOXY(CRT Unit)

Cú pháp: Gotoxy(x,y: Integer);

Công dụng: Đưa Curson (con trỏ) đến tọa độ(x,y);

82. Hàm GRAPHERRORMSG(Graph Unit)

Cú pháp: GraphErrorMsg(Code : Integer):String;

Công dụng: Cho thông báo lỗi đồ họa ứng với mã lỗi trong Code.

83. Thủ tục GRAPHDEFAULTS(Graph Unit)

Cú pháp: GraphDefaults;

Công dụng: Trả về các giá trị ngầm định cho định dạng đồ hoạ.

84. Thủ tục GRAPHRESULT(Graph Unit)

Cú pháp: Graphresult : Integer;

Công dụng: Cho mã lỗi của thủ tục đồ hoạ sau cùng.

85. Thủ tục HALT

Cú pháp: Halt;

Công dụng: Dừng ngay chương trình.

86. Hàm HI

Cú pháp: Hi(I : Integer): Byte;

Công dụng: Cho Byte cao trong số nguyên I.

87. Thủ tục HIGHVIDEO(CRT Unit)

Cú pháp: HighVideo;

Công dụng: tăng độ sáng màn hình .

88. Hàm IMAGESIZE(Graph Unit)

Cú pháp: ImageSize(x1,y1,x2,y2 : Integer);

Công dụng: Cho dung lượng tính theo byte để lưu trữ một miền ảnh Bitmap trong vùng chữ nhật x1,y1,x2,y2.

89. Thủ tục INC

Cú pháp: Inc(Var x [,n : longint]);

Công dụng: Tăng giá trị x thêm n (x phải thuộc kiểu nguyên). Nếu bỏ qua n thì x sẽ tăng lên 1. Ví dụ:

```
Var x:integer;
```

```
Begin
```

```
X:=15; Inc(x); { x sẽ có giá trị là 16}
```

```
Inc(x,5);      { x sẽ có giá trị là 20}
```

```
End;
```

90. Thủ tục INITGRAPH(Graph Unit)

Cú pháp: InitGraph(Var GD, GM:Integer; Driverpath:string);

Công dụng: Khởi tạo môi trường đồ hoạ và đưa phần cứng vào mode đồ hoạ. GD là trình điều khiển đồ hoạ, GM là mode đồ hoạ. Nếu GD bằng Detect, thủ tục tự động kiểm tra phần cứng và chọn các thông số tối ưu. Thủ tục sẽ tìm các tệp BGI trong đường dẫn DriverPath.

91. Hàm INSERT

Cú pháp: Insert(Source:String; Var Target:String; Index:Integer);

Công dụng: Chèn chuỗi Source vào chuỗi Target ở vị trí Index.

92. Thủ tục InsLine(CRT Unit)

Cú pháp: InsLine;

Công dụng: chèn một dòng trống vào vị trí hiện hành của con trỏ trên màn hình.

93. Hàm INT

Cú pháp: Int(x:Real): Real;

Công dụng: Cho phần nguyên của x

94. Hàm INTR(DOS Unit)

Cú pháp: Intr(Func:Byte; Var Regs:Registers);

Công dụng: Gọi ngắt có số hiệu Func của Bios với các thanh ghi được định nghĩa qua Regs.

95. Hàm IORESULT(CRT Unit)

Cú pháp: IOResult : word;

Công dụng: Cho mã lỗi khi thực hiện thao tác vào/ra. Nếu IOResult khác không thì có nghĩa là đã xảy ra lỗi.

96. Thủ tục KEEP(DOS Unit)

Cú pháp: Keep(Exitcode:word);

Công dụng: Kết thúc chương trình nhưng vẫn giữ nó thường trú.

97. Hàm KEYPRESSED(CRT Unit)

Cú pháp: Keypressed : Boolean;

Công dụng: Cho giá trị TRUE khi có một phím được bấm.

98. Hàm LENGTH

Cú pháp: Leng(S:String):Integer;

Công dụng: Cho chiều dài của chuỗi S

99. Thủ tục LINE(Graph Unit)

Cú pháp: Line(x1,y1,x2,y2:Integer);

Công dụng: Vẽ đường thẳng từ (x1,y1) đến (x2,y2).

100. Thủ tục LINEREL(Graph Unit)

Cú pháp: Linerel(Dx,Dy:Integer);

Công dụng: Vẽ đường thẳng từ vị trí hiện tại của con trỏ (toạ độ x,y) đến điểm (x+Dx,y+Dy).

101. LINETO(Graph Unit)

Cú pháp: Lineto(x,y:Integer);

Công dụng: Vẽ đường thẳng từ vị trí hiện tại của con trỏ đến điểm(x,y).

102. Hàm LN

Cú pháp: Ln(Var x:Real):Real;

Công dụng: Cho logarit tự nhiên của x.

103. Thủ tục LOWVIDEO(CRT Unit)

Cú pháp: LowVideo;

Công dụng: giảm độ sáng màn hình.

104. Thủ tục MARK

Cú pháp: Mark(P:pointer);

Công dụng: đánh dấu địa chỉ đỉnh heap trong con trỏ P

105. Hàm MAXAVAIL

Cú pháp: MaxAvail : LongInt;

Công dụng: Cho kích thước khối lớn nhất của vùng nhớ chưa cấp phát của Heap.

106. Hàm MEMAVAIL

Cú pháp: MemAvail : ;

Công dụng: cho tổng dung lượng của các khối nhớ tự do trong Heap.

107. Thủ tục MKDIR

Cú pháp: MKDir(S:String);

Công dụng: Tạo thư mục mới với tên chỉ định trong S.

108. Thủ tục MOVE

Cú pháp: Move(Var V1,V2,n:Integer);

Công dụng: chép n byte từ biến V1 đến biến V2.

109. Thủ tục MOVEREL(Graph Unit)

Cú pháp: MoveRel(Dx,Dy:Integer);

Công dụng: chuyển con trỏ từ toạ độ hiện thời (x,y) đến toạ độ x+dx, y+dy.

110. Thủ tục MOVETO

Cú pháp: MoveTo(x,y:Integer);

Công dụng: định vị con trỏ tại toạ độ(x,y).

111. Thủ tục MSDOS(DOS Unit)

Cú pháp: MSDOS(Var Regs:Registers);

Công dụng: gọi phục vụ DOS với tập giá trị của các thanh ghi trong Regs.

112. Thủ tục NEW

Cú pháp: New(Var P:Pointer);

Công dụng: cấp phát vùng heap cho con trỏ P.

113. Thủ tục NORMVIDEO(CRT Unit)

Cú pháp: NormVideo;

Công dụng: khôi phục thuộc tính màn hình ngằm định.

114. Thủ tục NOSOUND(CRT Unit)

Cú pháp: NoSound;

Công dụng: tắt âm thanh phát ra loa.

115. Hàm ODD

Cú pháp: Odd(I:Integer): Boolean;

Công dụng: Cho True khi I là số lẻ và False khi I là số chẵn

116. Hàm OFS

Cú pháp: ofs(<Variable, Procedure, or Function>):Integer;

Công dụng: Cho Offset địa chỉ vùng nhớ của biến, thủ tục hay hàm.

117. Hàm ORD

Cú pháp: Ord(S:Scalar):Integer;

Công dụng: Cho số thứ tự của S (có kiểu vô hướng đếm được).

118. Thủ tục OUTTEXT(Graph Unit)

Cú pháp: Outtext(Textstring):String;

Công dụng: viết chuỗi Textstring trong chế độ đồ hoạ. Cần chọn trước kiểu chữ, kích thước, màu sắc và hướng viết.

119. Thủ tục OUTTEXTXY(Graph Unit)

Cú pháp: OuttextXY(x,y:Integer; TextString:String);

Công dụng: Hiển thị chuỗi TextString ở vị trí(x,y). Cần chọn trước kiểu chữ, kích thước, màu sắc và hướng viết.

120. Thủ tục OVRCLEARBUF(Overlay Unit)

Cú pháp: OvrClearBuf;

Công dụng: xoá sạch vùng đệm Overlay.

121. Hàm OVRGETBUF(Overlay Unit)

Cú pháp: OvrGetBuf : LongInt;

Công dụng: Cho kích thước của vùng đệm Overlay.

122. Thủ tục OVRINIT(Overlay Unit)

Cú pháp: OVRInit(FileName:String);

Công dụng: Khởi tạo quá trình quản lý Overlay và mở tệp Overlay được chỉ định trong FileName. Thủ tục này cần được gọi tới trước bất cứ thủ tục Overlay nào.

123. Thủ tục PACKTIME(DOS Unit)

Cú pháp: PackTime(Var DT: DateTime; Var Time:Longint);

Công dụng: Chuyển hoá thông tin ngày, tháng và giờ trong DT (kiểu bản ghi) thành số 4 Byte và gửi vào biến Time .

124. Thủ tục OVRINITEMS(Overlay Unit)

Cú pháp: OvrInitEMS(fileName:String);

Công dụng: Nạp tệp Overlay vào vùng nhớ mở rộng (EMS) nếu còn đủ .

125. Thủ tục OVRSETBUF(Overlay Unit)

Cú pháp: OvrsetBuf(BufSize:Integer);

Công dụng: Cấp phát Bufsize byte cho vùng đệm Overlay. BufSize phải nhỏ hơn vùng đệm Overlay ngầm định.

126. Hàm PARAMSTR

Cú pháp: ParamStr(I:word):String;

Công dụng: Cho tham số dòng lệnh thứ I. Paramstr(0) cho đường dẫn tên tệp chương trình hiện hành (c:\tp\bin\turbo.exe)

127. Hàm PI

Cú pháp: Pi:Real;

Công dụng: Cho hằng số Pi

128. Thủ tục PIESLICE(Graph Unit)

Cú pháp: Pieslice(x,y:Integer; a1,a2,R:word);

Công dụng: Vẽ hình quạt tâm(x,y) bán kính R, từ góc a1 đến góc a2.

129. Hàm POS

Cú pháp: Pos(SubS,S:String):Integer;

Công dụng: Cho vị trí bắt đầu của chuỗi con SubS trong S. Nếu SubS không nằm trong S thì hàm POS cho giá trị 0.

130. Hàm PRED

Cú pháp: Pred(S:Scalar):Integer;

Công dụng: Cho phần tử phía trước phần tử S (S phải thuộc kiểu vô hướng đếm được).
Pred(5) = 4.

131 Thủ tục PUTIMAGE(Graph Unit)

Cú pháp: PutImage(x,y:Integer; Var BitMap; BitBlt:word);

Công dụng: Hiện ảnh chứa trong BitMap bắt đầu tại điểm (x,y). BitBlt chỉ định phương thức hiện ảnh và có thể nhận giá trị kiểu số hoặc hằng ký tự như sau:

COPYPUT = 0; XORPUT = 1; ORPUT = 2; ANDPUT = 3; NOTPUT = 4;

132. Thủ tục PUTPIXEL(Graph Unit)

Cú pháp: PutPixel(x,y:Integer; Color:word);

Công dụng: Vẽ một điểm ảnh (Pixel) tại tọa độ (x,y) theo màu chỉ định bởi color.

133. Hàm RANDOM

Cú pháp: Random(I:word):word;

Công dụng: Cho một số ngẫu nhiên k với $0 \leq k < I$.

134. Hàm RAMDOMIZE

Cú pháp: Randomize;

Công dụng: Khởi tạo giá trị ban đầu của bộ tạo số ngẫu nhiên trong đồng hồ hệ thống. Giá trị này được sử dụng khi có lời gọi hàm Random. Nếu không có thủ tục Randomize thì giá trị mà hàm Random tạo ra sẽ giống nhau mỗi khi chương trình được chạy.

135. Thủ tục READ và READLN

Cú pháp: Read([Var F : File,] Parameters);

Readln([Var F : File,] Parameters);

Công dụng: Nhập số liệu từ thiết bị chuẩn (thường là bàn phím) vào biến Parameters hay từ tệp được chỉ định trong F. Readln chỉ dùng với tệp văn bản, sau khi nhập số liệu Readln đi chuyển con trỏ xuống dòng kế tiếp.

136. Hàm READKEY(CRT Unit)

Cú pháp: Readkey : Char;

Công dụng: Đọc một mã phím vào một biến. Với các phím mã phím gồm hai số, số đầu là 0 thì cần gọi lại hàm Readkey một lần nữa để lấy giá trị thứ hai của mã phím.

137. Thủ tục RECTANGLE

Cú pháp: Rectangle(x1,y1,x2,y2:Integer);

Công dụng: Vẽ hình chữ nhật có góc ở trên bên trái ở tọa độ(x1,y1) và có góc dưới bên phải ở tọa độ(x2,y2).

138. Thủ tục RELEASE

Cú pháp: Release(Var P:Pointer);

Công dụng: Giải toả vùng nhớ heap đã cấp phát kể từ lần ghi (Mark) gần nhất trước đó. P cất giữ địa chỉ của đỉnh heap.

139. Thủ tục RENAME

Cú pháp: Rename(Var F:File; S:String);

Công dụng: Đổi tên tệp F thành S.

140. Thủ tục RESET

Cú pháp: Reset(Var F:File [;I:Integer]);

Công dụng: Mở tệp F để đọc. Nếu tệp đó không kiểu, ta có thể chỉ định kích thước mẫu tin trong I.

141. Thủ tục RESTORECRTMODE(Graph Unit)

Cú pháp: RestoreCRTmode;

Công dụng: thoát khỏi kiểu màn hình đồ họa trở về kiểu văn bản.

142. Thủ tục REWRITE

Cú pháp: Rewrite(Var F:File [;I:Integer]);

Công dụng: Chuẩn bị tệp để ghi. Nếu tệp chưa tồn tại Pascal sẽ tạo ra tệp mới với tên tệp ghi trong F . Nếu đã có một tệp trên đĩa thì tệp cũ sẽ bị xoá và một tệp rỗng được thế chỗ. Với tệp không kiểu ta có thể định chiều dài của recor (bản ghi) trong I.

143. Thủ tục RMDIR(DOS Unit)

Cú pháp: RmDir(S:String);

Công dụng: xoá thư mục được có tên chỉ định trong S

144. Hàm ROUND

Cú pháp: Round(x;Real):LongInt;

Công dụng: Làm tròn x.

145. Thủ tục RUNERROR

Cú pháp: RunError [(ErrorCode:word)];

Công dụng: Dừng chương trình, phát sinh lỗi Run-Time với số hiệu lỗi Errorcode.

146. Thủ tục SECTOR(Graph Unit)

Cú pháp: Sector(x,y:Integer;g1,g2,a,b:word);

Công dụng: Vẽ và tô đầy cung elip có tâm (x,y), bán kính ngang là a, bán kính dọc là b, bắt đầu từ góc g1 đến góc g2.

147. Thủ tục SEEK

Cú pháp: Seek(Var F:File; P:Integer);

Công dụng: Di chuyển con trỏ tệp đến đầu của bản ghi thứ P trong tệp tin F.

148. Thủ tục SEEKEOF

Cú pháp: SeekEof(Var F:File):Boolean;

Công dụng: kiểm tra tình trạng kết thúc tệp (chỉ dùng cho tệp văn bản), nếu kết thúc tệp kết quả nhận được là True, ngược lại là False (xem thêm eof).

149. Hàm SEEKEOLN

Cú pháp: SeekEoln(Var F:File):Boolean;

Công dụng: kiểm tra tình trạng kết thúc dòng, SeekEoln tương tự như Eoln chỉ có khác biệt là nó nhảy qua khoảng trống, tab trước khi kiểm tra dấu kết thúc hết dòng.

150. Hàm SEG

Cú pháp: Seg(Var Variable):word;

Seg(<Procedure hoặc Function>):word;

Công dụng: Cho địa chỉ đoạn của biến, hàm hay thủ tục.

151. Thủ tục SETALLPALETTE(Graph Unit)

Cú pháp: SetAllPalette(Var Palette);

Công dụng: Thay đổi bảng màu theo thiết kế cho trong Palette.

152. Thủ tục SETBKCOLOR(Graph Unit)

Cú pháp: SetBKcolor(Color:word);

Công dụng: chọn màu nền ngầm định cho màn hình đồ hoạ.

153. Thủ tục SETCBREAK(DOS Unit)

Cú pháp: SetCBreak(Break: Boolean);

Công dụng: thiết lập trạng thái cho tổ hợp CTRL – BREAK. Khi Break = True, hệ thống sẽ ngừng chương trình mỗi khi Ctrl-Break được bấm. Nếu Break=False việc kiểm tra chỉ tiến hành khi vào ra (I/O) với bàn phím hoặc máy in.

154. Thủ tục SETCOLOR(Graph Unit)

Cú pháp: Setcolor(color:word);

Công dụng: chọn màu cho nét vẽ theo số hiệu màu color .

155. Thủ tục SETDATE(DOS Unit)

Cú pháp: Setdate(Year, Month, Day:word);

Công dụng: Cập nhật đồng hồ hệ thống theo ngày tháng năm truyền qua tham biến.

156. Thủ tục SETFILLPATTERN(Graph Unit)

Cú pháp: SetFillPattern(Patter:FillPatterType;color:word);

Công dụng: Thiết lập mẫu tô hình vẽ (do người dùng định nghĩa tự chọn)

157. Thủ tục SETFILLSTYLE(Graph Unit)

Cú pháp: SetFillStyle(Pattern:word;color:word);

Công dụng: ấn định mẫu tô và màu tô.

158. Thủ tục SETGRAPHMODE(Graph Unit)

Cú pháp: SetGraphMode(Mode:Integer);

Công dụng: ấn định kiểu đồ họa hiện hành.

159. Thủ tục SETLINESTYLE(Graph Unit)

Cú pháp: SetLineStyle(lineStyle:word; Pattern:word; thickness:word);

Công dụng: ấn định dạng, mẫu và bề dày của đường vẽ.

160. Thủ tục SETPALETTE(Graph Unit)

Cú pháp: SetPalette(m1:word; m2:shortInt);

Công dụng: đổi màu thứ m1 trong bảng màu thành màu m2. (0<=m1<=15)

Setpalette(0,red); đổi màu thứ nhất trong bảng màu thành màu đỏ

161. Thủ tục SETRGBPALETTE(Graph Unit)

Cú pháp: SetRGBPalette(m1,RedValue, Greenvalue,BlueValue:Integer);

Công dụng: thay đổi các thành phần tổ hợp màu, màu m1 trong bảng màu được thay bằng tổ hợp màu: đỏ,xanh,lục. Thủ tục này chỉ dùng với các thiết bị IBM 8514 và VGA

162. Thủ tục SETTEXTJUSTIFY(Graph Unit)

Cú pháp: Settextjustify(Horiz, Vert:word);

Công dụng: Xác định vị trí hiển thị văn bản trong màn hình đồ họa.

163. Thủ tục SETTEXTSTYLE(Graph Unit)

Cú pháp: SetTextStyle(Font:word; direction:word; Charsize:word);

Công dụng: Thiết lập Font chữ, chiều hiển thị và kích cỡ chữ.

164. Thủ tục SETTIME(DOS Unit)

Cú pháp: SetTime(H, M, S, S100:word);

Công dụng: đặt lại giờ hệ thống bao gồm giờ (H), phút (M), giây (S), phần trăm giây (S100).

165. Thủ tục SETVIEWPORT(Graph Unit)

Cú pháp: SetviewPort(x1,y1,x2,y2:Integer; Clip: Boolean);

Công dụng: Chọn hình chữ nhật(x1,y1,x2,y2) làm cửa sổ làm việc. Khi Clip = TRUE cho phép vẽ ra ngoài cửa cửa sổ.

166. Thủ tục SETWRITEMODE(Graph Unit)

Cú pháp: SetWriteMode(WriteMode;Integer);

Công dụng: Chọn một trong 2 kiểu đường: Copyput(0) hay Xorput (1).

167. Hàm SIN

Cú pháp: Sin(x:Real):Real;

Công dụng: Cho Sin(x)

168. Hàm SIZEOF

Cú pháp: Sizeof(Var Variable):word;

Công dụng: Cho kích thước(số byte) của biến hay kiểu dữ liệu.

169. Thủ tục SOUND(Graph Unit)

Cú pháp: Sound(Freq:word);

Công dụng: Phát âm thanh có tần số Freq.

170. Hàm SQR

Cú pháp: Sqr(x:Real):Real; Sqr(x:Integer):Integer;

Công dụng: Cho bình phương của x.

171. Hàm SQRT

Cú pháp: Sqrt(x:Real):Real;

Công dụng: Cho căn bậc hai của x

172. Thủ tục STR

Cú pháp: Str(I:Integer; [:length;] Var S:String);

Str(x:Real; [:length; [:Decimals;]] Var S:String);

Công dụng: Đổi số nguyên hay thực ra chuỗi và cất trong S; length là độ dài của chuỗi(số ký tự), Decimals là chữ số sau dấu chấm thập phân.

173. Hàm SUCC

Cú pháp: Suuc(S:Scalar):Integer;

Công dụng: Cho giá trị tiếp sau S.

174. Thủ tục TEXTBACKGROUND(CRT Unit)

Cú pháp: TextbackGround(color:Byte);

Công dụng: Thay đổi màu nền chữ viết bằng màu chỉ ra trong color.

175. Thủ tục TEXTCOLOR(CRT Unit)

Cú pháp: TextColor(Color: Byte)

Công dụng: Thay đổi màu ký tự bằng màu chỉ ra trong color.

176. Hàm TEXTHEIGHT(Graph Unit)

Cú pháp: TextHeight(TextString:String):word;

Công dụng: Cho chiều cao(số pixel) của TextString.

177. Thủ tục TEXTMODE(CRT Unit)

Cú pháp: TextMode(Mode:Word);

Công dụng: Chọn Mode văn bản.

178. Hàm TEXTWIDTH(Graph Unit)

Cú pháp: TextWidth(TextString: String):Word;

Công dụng: Cho chiều rộng(số pixel) của chuỗi TextString.

179. Hàm TRUNC

Cú pháp: Trunc(x:Real):integer;

Công dụng: Cho phần nguyên của x.

180. Thủ tục TRUNCATE

Cú pháp: Truncate(Var F);

Công dụng: Cắt dữ liệu khỏi tệp kể từ vị trí hiện thời của con trỏ.

181. Thủ tục UNPACKTIME(DOS Unit)

Cú pháp: UnPackTime(Time:Longint; Var DT:DateTime);

Công dụng: Giải mã biến Time (là một số kiểu Longint và chứa trong nó các đại lượng ngày, giờ), trả kết quả về DT. Cấu trúc của DateTime là:

TYPE

 DateTime = Record

 Year,Month,Day,Hour,Min,Sec:word;

 End;

182. Hàm UPCASE

Cú pháp: Uppcase(S:Char):Char;

Công dụng: Chuyển các ký tự trong S thành chữ in.

183. Thủ tục VAL

Cú pháp: Val(S:String;Var R:Real;Var Code:Integer);

Val(S:String;Var I:Real;Var Code:Integer);

Công dụng: Đổi chuỗi S thành số (R hoặc I). Nếu đổi chuỗi thành công biến Code nhận giá trị 0. Nếu không đổi được, code chứa số nguyên cho biết vị trí của ký tự trong chuỗi S đã gây lỗi.

184. Hàm WHEREX(CRT Unit)

Cú pháp: WhereX:Byte;

Công dụng: Cho biết cột đang chứa Con trỏ trong cửa sổ hiện hành.

185. Hàm WHEREY(CRT Unit)

Cú pháp: WhereY:Byte;

Công dụng: Cho biết dòng đang chứa Con trỏ trong cửa sổ hiện hành.

186. Thủ tục WINDOW(CRT Unit)

Cú pháp: Window(x1,y1,x2,y2:Byte);

Công dụng: định nghĩa một cửa sổ góc trái trên (x1,y1) và góc phải dưới là (x2,y2).

187. Thủ tục WRITE và WRITELN

Cú pháp: Write(Var F:File, Parameters); Writeln(Var F:File, Parameters);

Công dụng: Write sẽ ghi giá trị các tham số lên màn hình hoặc lên tệp. Writeln làm việc giống Write nhưng ghi thêm dấu hiệu hết dòng(CR/LF) vào cuối .

Phụ lục 3

Định hướng biên dịch (Compiler Directive)

Định hướng biên dịch trong ngôn ngữ Pascal bao gồm 52 kiểu và được quy định viết như sau:

a. Vị trí

Các định hướng biên dịch được đặt tại những vị trí thích hợp trong chương trình tùy thuộc vào các yêu cầu cụ thể của bài toán.

b. Tất cả định hướng biên dịch đều đặt trong cặp ký tự " { " và " } "

Mỗi định hướng đều có giá trị ngầm định (by default) và trình biên dịch sẽ sử dụng giá trị này nếu trong chương trình không có những định hướng biên dịch khác. Để tìm hiểu nội dung của toàn bộ các định hướng biên dịch trong Pascal xin mở Help từ trang màn hình thứ 1 đến trang màn hình thứ 2. Dưới đây là một số định hướng biên dịch thông dụng:

1. Ghép tệp

Cú pháp: {\$I filename}

Định hướng cho trình biên dịch gộp tên tệp chỉ định bởi FileName vào lúc dịch. Kết quả là tệp được chèn vào văn bản cần dịch ngay phía sau dẫn hướng này. Phần mở rộng ngầm định cho filename là .PAS. Nếu trong filename không chỉ định thư mục, thì trình biên dịch sẽ tìm kiếm tệp đó trong thư mục hiện hành.

2. Kiểm tra vào ra (Input/Output)

Cú pháp: {\$I+} hoặc {\$I-}

Ngầm định: {\$I-}

Không tự động sinh mã kiểm tra kết quả lệnh khi có lệnh gọi đến các thủ tục vào ra. Khi dùng {\$I-}, chương trình sẽ không dừng ngay cả khi có lỗi vào ra (mã lỗi khác 0). Muốn kiểm tra lỗi cần dùng hàm IOResult. Nếu thủ tục vào ra có lỗi và định hướng là {\$I+} thì chương trình sẽ chấm dứt và đưa ra thông báo lỗi.

3. Link Object File

Cú pháp: {\$L filename}

Yêu cầu trình biên dịch liên kết tệp chỉ định bởi Filename với chương trình hay unit cần dịch. Phần mở rộng ngầm định của filename là .OBJ. Nếu trong filename không chỉ rõ đường dẫn thì trình biên dịch sẽ tìm tệp đó trong thư mục hiện hành.

4. Xử lý số(Numeric Processing)

Cú pháp: {\$N+} hoặc {\$N-}

Ngầm định: {\$N-}

Định hướng việc phát sinh mã dấu phẩy động với số thực trong Pascal. Với {N-} việc sinh mã để thực hiện tính toán dấu phẩy động do các chương trình trong thư viện Run-Time đảm nhận. Với {N+} sẽ dùng bộ xử lý số 8087 để sinh mã.

5. Overlay code generation.

Cú pháp: {O+} hoặc {O-}

Ngầm định: {O-}

Khi chọn {O+} sẽ cho phép unit là Overlay. Trình tạo mã các xử lý đặc biệt để truyền tham số cho các thủ tục hay hàm có Overlay.

6. Kiểm tra phạm vi (Range)

Cú pháp: {R+} hoặc {R-}

Ngầm định: {R-} Định hướng này cho không cho phát sinh mã kiểm tra phạm vi. Với {R+} tất cả biểu thức chỉ số mảng và chuỗi đều được kiểm tra xem có nằm trong phạm vi đã định nghĩa hay không. Tất cả các phép gán vào biến kiểu vô hướng và đoạn con cũng đều được kiểm tra phạm vi. Nếu có lỗi, chương trình kết thúc và hiện thông báo lỗi Run-time. Định hướng này sẽ làm cho chương trình có kích thước lớn và chạy chậm, chỉ nên dùng khi chạy thử chương trình. Sau khi chương trình đã chạy tốt thì nên bỏ.

7. Kiểm tra tràn Stack

Cú pháp: {S+} hoặc {S-}

Ngầm định: {S-}

Định hướng này không chế việc sinh mã kiểm tra tràn Stack.

Với {S+} trình biên dịch sinh mã ở phần đầu của từng thủ tục hay hàm để kiểm tra có đủ chiều dài Stack cho các biến cục bộ và vùng nhớ khác. Khi không đủ chiều dài Stack thì lệnh gọi đến thủ tục hay hàm được biên dịch với {S+} sẽ chấm dứt chương trình và xuất hiện thông báo lỗi Run-time.

Với {S-} lệnh gọi như thế sẽ đưa đến ngừng hệ thống.

8. {U-} hoặc {U+}

Ngầm định {U-}

Định hướng này không cho phép dùng tổ hợp phím Ctrl - C để ngắt chương trình. Với {U+} người sử dụng có thể dùng chương trình bất kỳ lúc nào bằng cách bấm tổ hợp Ctrl - C.

Phụ lục 4

Thông báo lỗi

Để tham khảo các thông báo lỗi, từ Text Editor (màn hình soạn thảo chương trình) hãy gõ từ Error rồi bấm Ctrl-F1, chúng ta sẽ truy nhập vào mục Error - Message. Trong mục này Pascal trình bày ba nhóm lỗi:

Compile error messages gồm các lỗi từ số 1 đến số 99 (liên tục)

Compile error messages gồm các lỗi từ số 100 đến số 170 (liên tục)

Run - Time error messages gồm các lỗi từ số 1 đến số 216 (không liên tục)

Các lỗi Run-time và Compile có một số lỗi cùng số hiệu nhưng ý nghĩa khác nhau. Do số lượng lỗi khá nhiều nên chúng tôi bỏ bớt một số lỗi ít gặp.

1. Lỗi biên dịch

Số hiệu lỗi	Thông báo	Giải thích ý nghĩa
1	Out of memory	Bộ nhớ không đủ khi biên dịch
2	Identifler expected	Một tên được dùng không phù hợp. Có thể bạn đang khai báo lại một danh từ riêng, từ khoá của Turbo Pascal.
3	Unknown identifier	Tên (hoặc biến) chưa được khai báo, hoặc không được biết đến ở trong khối hiện thời. Có thể đó là một tên thiếu hoặc nhầm một vài kí tự so với tên đã được khai báo.
4	Duplicate identifier	Sử dụng hai tên trùng nhau trong cùng một khối. Có thể là tên chương trình trùng tên biến.
5	Syntax error	Lỗi chính tả trong văn bản nguồn. Có thể là do bỏ quên dấu nháy trên 'abcde'.
6	Error in real constant	Lỗi khi biểu diễn các số thực, cách viết sai có thể là: .98 (không có chữ số 0 ở đầu).
7	Error in integer constant	Lỗi biểu diễn số nguyên
8	String constant	Thường là do quên đánh dấu kết thúc hoặc mở đầu một xâu kí tự, hoặc do vị trí dấu phẩy và dấu nháy không đúng.
10	Unexpected end of file	Kết thúc file không hợp lệ * File nguồn kết thúc trước lệnh end của phần lệnh chính, hoặc lệnh begin và end không đi cặp với nhau. * Phần chú thích đóng mở không đúng
11	Line too long	Dòng lệnh dài quá 127 kí tự
12	Type identifier expected	Biến (tên) không phù hợp về kiểu mà đang ra nó phải có.
13	Too many open files	Mở quá nhiều file hoặc trong file CONFIG.SYS không có khai báo FILES=n (n là số lượng File sẽ mở) hoặc giá trị n quá nhỏ, hãy tăng giá trị này lên khoảng từ 15 đến 20.

14	Invalid file name	Tên file không đúng hoặc chỉ định một đường dẫn không tồn tại.
15	File not found	Không tìm thấy file trong thư mục hiện hành hoặc trong các đường dẫn đã chỉ định để tìm kiếm từng loại file.
16	Disk full	Đĩa đã đầy, không còn chỗ. Hãy xoá bỏ một vài file, nếu là đĩa mềm hay dùng đĩa khác.
17	Invalid compiler directive	Ký tự định hướng dịch không đúng hoặc chưa được biết, một trong các tham số dẫn hướng có thể lỗi.
18	Too many files	Quá nhiều file (của chương trình hay unit) được đưa vào dịch. Cần giảm số File đi.
19	Undefined type in pointer definition	Một kiểu được xem như là con trỏ trỏ đến kiểu đã được khai báo trước, song nó chưa hề được khai báo.
20	Variable identifier expected	Biến (tên) không thể hiện một biến mà đáng ra nó phải có.
21	Error in type	Lỗi về kiểu dữ liệu
22	Structure too large	Cỡ lớn nhất cho phép của một kiểu cấu trúc là 65535 byte.
23	Set base type out of range	Kiểu cơ bản của tập hợp phải là một tập con 0..255 hoặc là kiểu liệt kê có không nhiều hơn 256 giá trị.
24	File components may not be files or object	Cấu trúc file of file hoặc file of object là không cho phép. Kiểu File và kiểu cấu trúc không thể chứa một kiểu object hoặc kiểu file.
25	Invalid String leng	Độ dài của một chuỗi phải trong khoảng 1 - 255
26	Type mismatch	Lỗi này xảy ra trong các trường hợp sau: * Giá trị gán cho biến không tương thích về kiểu. * Tham số thực sự và tham số hình thức không tương thích về kiểu * Kiểu của biểu thức không tương thích với kiểu chỉ số khi làm chỉ số mảng. * Không tương thích giữa các kiểu của các toán hạng trong một biểu thức.
27	Invalid subrange base type	Kiểu cơ sở của kiểu khoảng con không hợp lệ.
28	Lower bound greater than upper bound	Khai báo kiểu miền con cận dưới cao hơn cận trên, hoặc giá trị đầu và cuối trong vòng For không phù hợp
29	Ordinal type expected	Kiểu thực, kiểu chuỗi, kiểu cấu trúc, kiểu con trỏ không được cho phép ở đây. Các kiểu này không phải là các kiểu đếm được.
30	Integer constant expected	Không viết đúng hằng số nguyên.
31	Constant expected	Hằng viết không đúng.
32	Integer or real constant expected	Viết không đúng hằng thực hoặc hằng nguyên.

33	Pointer type identifier expected	Biến (tên) không thể hiện một kiểu con trỏ mà đáng ra nó có.
34	Invalid function result type	Kiểu kết quả của hàm không hợp lệ. Các kiểu kết quả của hàm trả về hợp lệ là các kiểu đơn giản, kiểu chuỗi và kiểu con trỏ.
35	Label identifier expected	Tên nhãn không đúng.
36	BEGIN expected	Thiếu từ khoá Begin hoặc
37	END expected	Thiếu từ khoá End
38	Integer expression expected	Biểu thức kiểu nguyên không đúng.
39	Ordinal expression expected	Biểu thức phía trước phải là biểu thức đếm được.
40	Boolean expression expected	Biểu thức phía trước phải là kiểu logic.
41	Operand types do not match operator	Toán tử sử dụng trong các biểu thức không phù hợp với toán hạng, ví dụ $A \bmod 2$
42	Error in expression	Lỗi trong một thao tác. Ví dụ quên viết một toán tử giữa hai toán hạng.
43	Legal assignment	* Không thể gán giá trị cho file và biến không kiểu. * Tên hàm chỉ có thể gán giá trị trong thân của hàm.
44	File identifier expected	Biến (tên) không thể hiện là một trong bản ghi tương ứng hoặc trong biến object.
45	Object file too large	Pascal không thể liên kết với các file .OBJ lớn hơn 64k.
46	Undefined external	Thủ tục hoặc hàm external không đặt định nghĩa PUBLIC trong file object. Có thể bạn đã chỉ ra tất cả các file object trong dẫn hướng biên dịch $\{ \$L filename \}$, cần kiểm tra chính tả trong tên thủ tục hay hàm trong file ASM.
47	Invalid object file record	File .OBJ chứa một bản ghi không hợp lệ. Cần đảm bảo rằng file này thật sự phải là một file .OBJ.
48	Code segment too large	Bộ nhớ dành cho chương trình hay unit là 65520 byte (64KB). Nếu chương trình lớn hãy chuyển vài thủ tục hay hàm vào trong một unit. Nếu bạn đang biên dịch unit hãy chia nhỏ nó thành một hay nhiều unit.
49	Data segment too large	Kích thước to nhất của một đoạn dữ liệu chương trình là 65520 byte (64KB), kể cả dữ liệu được khai báo trong các unit được dùng.
50	Do expected	Thiếu từ khoá DO hoặc để không đúng chỗ.

51	Invalid PUBLIC definition	Định nghĩa PUBLIC không đúng. * Hai hay nhiều chỉ dẫn PUBLIC trong ngôn ngữ assembly định nghĩa cùng một tên. * File .OBJ định nghĩa ký hiệu PUBLIC không thuộc về đoạn CODE
52	Invalid EXTRN definition	Định nghĩa EXTRN không đúng. * Biến (tên) được chuyển tới hướng dẫn EXTRN trong ngôn ngữ assembly. Nhưng nó không được khai báo trong chương trình Pascal hay unit, không có cả phần giao diện của bất cứ unit được dùng nào.
53	Too many EXTRN definitions	Turbo Pascal không thể quản lý file .OBJ với nhiều hơn 256 khai báo EXTRN.
54	OF expected	Thiếu từ khoá OF hoặc từ OF không xuất hiện nơi nó cần có.
55	INTERFACE expected	Thiếu danh từ riêng Interface.
56	Invalid relocatable reference	Chỉ dẫn tái định vị bị sai
57	Then expected	Thiếu từ THEN hoặc nó được đặt không đúng chỗ
58	To or Downto expected	Từ khoá To hoặc Downto đặt không đúng vị trí.
59	Undefined forward	* Thủ tục hoặc hàm được khai báo trong phần interface của một unit, nhưng chưa định nghĩa trong phần implementation. * Thủ tục hoặc hàm được khai báo với từ khoá forward nhưng định nghĩa của nó không có.
61	Invalid typecast	Kích thước của biến và kiểu nhận khác nhau, hoặc một biểu thức được đưa vào nơi chỉ cho phép biến có mặt.
62	Division by zero	Phép chia cho số 0.
63	Invalid file type	Kiểu file không đúng. Ví dụ: readln không thể dùng với file có kiểu hoặc seek với file text.
64	Cannot Read and Write variables of this type	Kiểu biến không thể áp dụng trong các lệnh Read hoặc Write
65	Pointer variable expected	Chưa khai báo biến kiểu con trỏ.
66	String variable expected	Chưa khai báo biến kiểu chuỗi.
67	String expression expected	Biểu thức phải thuộc kiểu chuỗi.
68	Circular unit reference	Lỗi vì các Unit tham chiếu vòng tròn (a gọi tới b, b gọi tới c, c gọi tới a)
69	Unit name mismatch	Tên của unit tìm thấy trong file .TPU không phù hợp với tên chỉ ra trong mệnh đề uses.

70	Unit version mismatch	Một hoặc nhiều unit được dùng bởi unit này đã được thay đổi từ khi unit được dịch. Dùng Compiler/Make or Compile/Built trong IDE và lựa chọn/M hoặc /B trong trình biên dịch theo dòng lệnh để tự động dịch lại các unit cần thiết.
71	Internal stack overflow	Tràn stack , dung lượng hiện có của Stack không đủ để nạp những gì chương trình yêu cầu
72	Unit file format error	File đơn vị chương trình .TPU bị lỗi. Có thể file được tạo nên bởi một version cũ của pascal, cần tạo lại file mới bằng cách dịch lại file nguồn.
73	IMPLEMENTATION expected	Từ khoá implementation không xuất hiện nơi cần. Có thể là do gộp phần imlementation của một thủ tục, một hàm hoặc một hành vi vào trong phần interface của unit.
74	Constan and case types do not match	Kiểu của biến chọn trong cấu trúc case không tương thích với giá trị của các nhãn
75	Record or object variable expected	Biến phải là kiểu dữ liệu bản ghi hoặc kiểu object.
76	Constant out of range	Lỗi có thể là: * Đánh chỉ số mảng với một giá trị vượt ra ngoài phạm vi. * Gán một giá trị vượt phạm vi cho một biến. * Cho một hằng vượt phạm vi làm tham số cho một biến hoặc hàm.
77	File variable expected	Biến phía trước phải thuộc kiểu file.
78	Pointer expression expected	Biểu thức phía trước phải thuộc kiểu con trỏ.
79	Integer or real expected	Biểu thức phải là kiểu nguyên hay kiểu thực
80	Label not within current block	Nhãn không liên kết với Block hiện tại. Lệnh goto nhảy tới một nhãn nằm ngoài khối hiện tại.
81	Label already defined	Nhãn đã được sử dụng
82	Undefined label in preceding statement part	Nhãn được khai báo và chỉ dẫn trong phần lệnh trước nhưng chưa được định nghĩa
83	Invalid @ argument	Đối số (Argument) hợp lệ cho toán tử @ là một biến, tên thủ tục hoặc hàm.
84	UNIT expected	Thiếu từ khoá unit
85	“;” expected	Thiếu dấu ;
86	“:” expected	Thiếu dấu :
87	“,” expected	Thiếu dấu ,
88	“(” expected	Thiếu dấu ngoặc đơn mở
89)” expected	Thiếu dấu ngoặc đơn đóng
90	“=” expected	Thiếu dấu bằng
91	“:=” expected	Thiếu toán tử gán

92	“[” or “(.” expected	Thiếu dấu móc trái
93	“]” or “.)” expected	Thiếu dấu móc phải
94	“.” Expected	Thiếu dấu chấm .
95	“.” expected	Thiếu dấu miền con .
96	Too many variables	Quá nhiều biến. (tổng cộng các biến toàn cục khai báo trong chương trình hay unit không được vượt hơn 64k. Tổng cộng các biến địa phương khai báo trong chương trình con hay unit cũng không được vượt hơn 64k).
97	Invalid FOR control variable	Biến điều khiển trong câu lệnh FOR phải thuộc kiểu đơn giản đếm được và phải được khai báo trước.
98	Integer variable expected	Biến phía trước phải là kiểu nguyên
99	File types are not allowed here.	Kiểu file không cho phép ở đây (hàng định kiểu không thể là kiểu file.)
100	String length mismatch	Độ dài của một chuỗi kí tự không phù hợp với số thành phần trong mảng ký tự.
101	Invalid ordering of fields	Trường của hằng bản ghi hay hằng object phải viết trong trật tự của sự khai báo.
102	String constant expected	Chuỗi hằng không xuất hiện ở nơi cần có.
103	Integer or real variable expected	Biến phải thuộc kiểu nguyên hay kiểu thực.
104	Ordinal variable expected	Biến phía trước phải là kiểu đếm được.
105	INLINE error	Toán tử < không cho phép cùng các tham chiếu tái định vị tới biến , các tham chiếu như vậy luôn luôn có kích thước word.
106	Character expression expected	Biểu thức phía trước phải là kiểu ký tự.
107	Too many relocation items	Kích thước của bảng tái định vị file kiểu .EXE vượt hơn 64K, lỗi này xuất hiện là do chương trình quá to nên chương trình linker của Turbo Pascal không quản lý được. Nó cũng có thể quá to để DOS thi hành.
108	Overflow in arithmetic operation	Kết quả của các phép toán số học không nằm trong phạm vi số nguyên dài (-2147483648 .. 21474783647). Hiệu chỉnh phép toán hoặc dùng giá trị kiểu số thực thay thế giá trị kiểu nguyên.
109	No enclosing FOR, WHILE, or REPEAT statement	Thủ tục chuẩn Break và continue không thể dùng ngoài lệnh for, while hoặc repeat.
112	CASE constant out of range	Trong cấu trúc CASE các hằng là số nguyên phải trong phạm vi -32768..32767.
113	Error in statement	Ký hiệu này không thể bắt đầu một câu lệnh.

114	Cannot call an interrupt procedure	Bạn không thể trực tiếp gọi một thủ tục ngắt.
117	Target address not found	Lệnh Searess Find Error trong IDE hoặc lựa chọn /F trong phiên bản dịch dòng lệnh không phát hiện được vị trí câu lệnh tương ứng với địa chỉ chỉ ra.
118	Include files are not allowed here	Mọi phân lệnh phải hoàn toàn nằm trong một file.
121	Invalid qualifier	Lỗi thuộc một trong các trường hợp sau: * Đánh chỉ số một biến không phải là phần tử mảng. * Chỉ ra một trường trong một biến không phải là bản ghi. * Lấy địa chỉ trả về của một biến không phải là con trỏ.
122	Invalid variable reference	Chương trình tham chiếu tới biến, nhưng biến lại không phải là một vị trí bộ nhớ (vì biến phải được lưu trong những ô nhớ có địa chỉ). Thường là chương trình đang cố gắng thay đổi một tham số const hoặc đang gọi một hàm con trỏ nhưng quên lấy địa chỉ trả về của kết quả.
123	Too many symbols	Chương trình hoặc unit khai báo vượt quá 64K ký hiệu. Nếu định hướng dịch là {\$D+}, hãy thử đặt nó off, nghĩa là {\$D-}, (điều này sẽ ngăn trở bạn tìm lỗi run – time trong khối này). Hãy thử chuyển một số khai báo (hàng, biến, kiểu) tới một unit riêng rẽ.
124	Statement part too large	Pascal giới hạn dung lượng của phân lệnh khoảng 24K. Gặp lỗi này, hãy cắt các đoạn và đưa vào một hay nhiều thủ tục hơn.
126	Files must be var parameters	Kiểu file không thể đưa làm tham trị trong chương trình con, kiểu File chỉ có thể là tham biến nghĩa là phải khai báo với từ khoá Var .
127	Too many conditional symbols	Không đủ chỗ để khai báo ký hiệu điều kiện thêm nữa. Cố gắng bỏ đi vài ký hiệu hoặc rút ngắn vài tên tượng trưng.
129	ENDIF directive missing	Lỗi khi biên dịch các cấu trúc có điều kiện. Số lượng của các từ khoá mở {\$Ifxxx} và các từ khoá đóng {\$ENDIF} trong file nguồn phải bằng nhau.
130	Error in initial conditional defines	Các ký hiệu điều kiện ban đầu chỉ định trong Option/Compiler/Condition Defines (trong môi trường khép kín IDE) hoặc trong dịch hướng biên dịch /D (với trình biên dịch dòng) không hợp lệ.
131	Header does not match previous definition	Các khai báo ở phần đầu (chương trình) không phù hợp với khai báo thủ tục hoặc hàm chỉ ra trong phần interface hoặc trong phần khai báo forward
133	Cannot evaluate this expression	Không thể ước lượng (định giá) biểu thức. Có thể đây là một phép tính không hợp lệ. Ví dụ bạn đang cố gắng dùng hàm Sin trong khai báo const.
134	Expression incorrectly terminated	Biểu thức kết thúc không hợp lệ. Ví dụ cuối biểu thức phải là toán hạng chứ không phải là toán tử.
137	Structured variables are not allowed here	Cấu trúc biến không cho phép thực hiện các thao tác mà chương trình đòi hỏi. Ví dụ, bạn đang cố gắng làm phép nhân 2 bản ghi.

138	Cannot evaluate without system unit	Thư viện TURBO.TPL phải bao gồm System Unit để chương trình gỡ rối có thể tính toán các biểu thức.
140	Invalid floating – point operation	Một thao tác trên hai giá trị thực đã tạo nên sự tràn hoặc một phép chia cho số 0.
141	Cannot compile overlays to memory	Chương trình dùng overlays phải ghi kết quả dịch trên đĩa chứ không thể gửi vào bộ nhớ
142	Pointer or procedural variable expected	Hàm chuẩn Assigned yêu cầu đối số là một biến con trỏ hoặc kiểu thủ tục.
143	Invalid procedure or function reference	Lỗi dùng tên một thủ tục trong một biểu thức tính toán. (Nếu bạn đang chuẩn bị gán một thủ tục hoặc hàm cho một biến thủ tục, nó phải được dịch trong trạng thái {\$F+} và không thể khai báo với inline hay interrupt.)
144	Cannot overlay this unit	Bạn đang cố gắng overlay một unit mà nó không được dịch với trạng thái {\$O+}.
146	File access denied	Không thể mở file hay tạo ra file mới. Lỗi thường xảy ra khi trình biên dịch đang cố gắng viết vào một file chỉ đọc (Read only file).
147	Object type expected	Biến (tên) không thể hiện một kiểu object.
148	Local object types are not allowed	Kiểu object chỉ có thể định nghĩa ở tầm xa nhất của một chương trình hay unit. Định nghĩa kiểu object trong thủ tục hay hàm là không được phép.
149	VIRTUAL expected	Từ dành riêng virtual bị thiếu.
150	Method identifier expected	Biến (tên) không thể hiện một hành vi.
151	Virtual constructors are allowed	Hành vi constructor phải là tĩnh.
152	Constructor identifier expected	Biến (tên) không thể hiện một constructor.
153	Destructor identifier expected	Biến (tên) không thể hiện một destructor.
154	Fail only allowed within constructors	Thủ tục chuẩn Fail chỉ được dùng bên trong một constructor.
155	Invalid combination of opcode and operands	Mã assembler không chấp nhận liên kết các toán hạng. Các nguyên nhân có thể: * Quá ít hay quá nhiều các toán hạng đối với lệnh assembler này, ví dụ INC AX, BX hay Mov AX . * Số lượng các toán hạng là phù hợp, nhưng kiểu hoặc thứ tự không phù hợp với mã; ví dụ DEC1, MOV AX, CL hoặc MOV 1, AX .

161	Code generation error	Phần lệnh phía trước chứa một lệnh LOOPNE , LOOPE , LOOP , hoặc JCXZ song các lệnh nhảy này không thể với tới nhãn đích.
162	ASM expected	Bạn đang cố gắng biên dịch một hàm hoặc thủ tục assembler có sẵn bên trong và chúng chứa một câu lệnh begin...end thay vì dùng asm...end .

2. Lỗi liên quan đến hệ điều hành DOS (Dos errors)

1	Invalid function number	Số hiệu của một hàm (DOS) không tồn tại
2	File Not Found	Không tìm thấy file (có thể là do các thủ tục <i>Reset</i> , <i>Append</i> , <i>Rename</i> , <i>Rewrite</i> , <i>Erase</i> báo lại khi tên File không hợp lệ hoặc tên gán cho biến file không xác định một file tồn tại.)
3	Path not found	Đường dẫn không tìm thấy, lỗi có thể là do: Các thủ tục <i>Reset</i> , <i>Append</i> , <i>Rename</i> , <i>Rewrite</i> , <i>ChDir</i> , <i>Mkdir</i> , <i>Rmdir</i> hoặc <i>Erase</i> gặp tên gán cho tên biến file không hợp lệ hoặc đường dẫn tới một thư mục con không tồn tại.
4	Too many open files	Do các thủ tục <i>Reset</i> , <i>Rewrite</i> hoặc <i>Append</i> báo lại nếu chương trình có quá nhiều file được mở. DOS không cho phép mở quá 15 file cho một ứng dụng. Nếu gặp lỗi này khi mở ít hơn 15 file thì có thể là trong file CONIG.SYS không chứa khai báo <i>FILES=n</i> hoặc giá trị n quá nhỏ hãy tăng n lên
5	File access denied	Từ chối truy cập file, lỗi có thể xảy ra khi các thủ tục: * làm việc với file nhưng lại bị chỉ định tới tên thư mục * làm việc với các file đã gán thuộc tính chỉ đọc. * Làm việc với các file chưa được mở
6	Invalid file handle	Lỗi này được báo lại nếu một thẻ file không hợp lệ được chuyển tới một lời gọi hệ thống DOS, nguyên nhân có thể là do file bị lỗi.
12	Invalid file access code	Thủ tục <i>Reset</i> hoặc <i>Append</i> gặp phải một file có kiểu hoặc không có kiểu mà giá trị <i>FileMode</i> không hợp lệ.
15	Invalid drive number	Do các thủ tục <i>GetDir</i> hoặc <i>ChDir</i> báo lại nếu mã số ổ đĩa không hợp lệ.
16	Cannot remove current directory	Không thể chuyển thư mục đang làm việc (bởi thủ tục <i>Rmdir</i>)
17	Cannot rename across drives	Không đổi được tên vì hai tên không có cùng đường dẫn
18	No more files	Do biến <i>DosError</i> trong unit <i>Dos</i> và <i>WinDos</i> báo lại khi một lời gọi tới <i>FindFirst</i> hoặc <i>FindNext</i> không tìm thấy file thích hợp với tên file và tập các thuộc tính chỉ ra.

3. Lỗi vào ra (Input/Output error)

Lỗi vào ra thường xuất hiện khi biên dịch chương trình, nếu định hướng dịch là $\{I+\}$ (ngầm định) thì chương trình sẽ dừng ngay khi gặp lỗi. Nếu ở đầu chương trình chúng ta quy định $\{I-\}$ thì chương trình tiếp tục thi hành, và lỗi được thông báo bởi hàm *IOResult*. Dưới đây là những lỗi hay gặp:

100	Disk read error	Lỗi đọc dữ liệu, thường là do thủ tục Read thông báo khi trên một file có kiểu nếu bạn đang cố gắng đọc quá vị trí cuối file.
101	Disk write error	Lỗi khi thực hiện các thủ tục Close, Write, WriteIn hoặc Flush khi đĩa đã đầy
102	File not assigned	File không thể tạo ra, lỗi này xuất hiện khi thực hiện các thủ tục Reset, Rewrite, Append, Rename và Erase nhưng biến file chưa được gán tên qua thủ tục Assign.
103	File not open	File chưa được mở khi thực hiện các thủ tục Close, Read, Write, Seek, Eof, FilePos, FileSize, Flush, BlockRead hoặc BlockWrite .
104	File not open for input	File (văn bản) chưa được mở để đọc khi thực hiện các thủ tục Read, Readln, Eof, Eoln, SeekEof hoặc SeekEoln .
105	File not open for output	File chưa được mở để ghi khi thực hiện các thủ tục Write hoặc WriteIn .
106	Invalid numeric format	Trị số đọc từ file văn bản không đúng với dạng số hợp lệ Khi thực hiện các thủ tục Read hoặc Readln .

4. Các lỗi liên quan đến phần cứng

Các lỗi liên quan đến phần cứng có thể tham khảo thêm trong các tài liệu về hệ điều hành MS-DOS

150	Disk is write protected	Đĩa (mềm) bị khoá chống ghi. Hãy bỏ các lẫy chống ghi trên đĩa ra.
151	Unknown Unit	Tên đơn vị chương trình chưa được biết đến
152	Drive not ready	Ổ đĩa không sẵn sàng. Có thể do chưa đưa đĩa vào, có thể mất nguồn điện nuôi.
153	Unknown command	Không có lệnh này. Chắc bị gõ nhầm một vài ký tự.
154	CRC error in data	Lỗi vùng dữ liệu
155	Bad drive request structure length	
156	Disk seek error	Lỗi khi truy nhập đĩa
157	Unknown media type	Kiểu thiết bị không rõ
158	Sector not found	Cung từ không tìm thấy. Có lẽ bị hỏng do nhiều nguyên nhân: mốc, xước hoặc bị bụi bẩn...
159	Printer out of paper	Máy in không có giấy
160	Device write fault	Việc ghi vào thiết bị không thực hiện được
161	Device read fault	Việc đọc bị lỗi
162	Hardware failure	Phần cứng bị hỏng

5. Lỗi trình biên dịch (run-time error)

Khi trình biên dịch gặp các lỗi này chương trình sẽ bị dừng ngay lập tức.

200	Division by zero	Chương trình cố gắng chia một số cho 0 trong phép toán mod hoặc div .
201	Range check error	Nếu định hướng biên dịch là {R+} thì lỗi sẽ được thông báo khi một trong các tình huống sau xảy ra: * Chỉ số của một mảng vượt ra ngoài phạm vi đã khai báo * Gán một giá trị vượt ra ngoài phạm vi cho phép của một biến. * Giá trị gán cho tham số của một hàm hay một biến vượt ra ngoài phạm vi cho phép
202	Stack overflow error	Stack bị tràn. Nếu định hướng biên dịch là {S+} thì lỗi sẽ được thông báo khi không có đủ chỗ trống trên stack để bố trí bộ nhớ cục bộ của các chương trình con. Có thể thay đổi Stack bằng cách dùng định hướng biên dịch \$M. * Lỗi này cũng có thể do vòng lặp vô tận gây ra, hoặc do một thủ tục viết trong ngôn ngữ assembler không duy trì tình trạng stack.
203	Heap overflow error	Lỗi này do các thủ tục New hoặc GetMem báo lại nếu không còn đủ chỗ trống trong heap để cấp phát một vùng nhớ theo yêu cầu.
204	Invalid pointer operation	Lỗi này do các thủ tục Dispose hoặc FreeMem gây ra nếu như con trỏ là Nil hoặc trỏ tới một vị trí nằm ngoài heap.
205	Floating point overflow	Phép tính với dấu chấm động tạo ra kết quả là một số quá lớn .
207	Invalid floating point operation	Một tham số thực trong các hàm Trunc hoặc Round không thể chuyển đổi thành một số nguyên trong phạm vi số nguyên dài LongInt (-2147483648 - 2147483647). * Tham số của hàm Sqrt mang giá trị âm. * Tham số của hàm Ln là zero hoặc âm. * Sự tràn 8087 đã xảy ra. (xem trong chương 14. “Using the 80x87” trong cuốn Language guide.)
208	Overlay manager not installed	Trình quản lý overlay chưa được cài đặt. Lỗi có thể là do không gọi OverInit hoặc lời gọi OverInit không chạy.
209	Overlay file read error	Lỗi xảy ra khi trình quản lý overlay cố gắng đọc một overlay từ file overlay.

Mục lục

Lời mở đầu	2
Chương 1: Chương trình con - Thủ tục và hàm	2
1. Khái niệm về chương trình con	4
2. Tham số trong chương trình con	5
3. Truyền tham số cho chương trình con	8
4. Biến toàn cục và biến địa phương	9
5. Cách thức bố trí bộ nhớ	11
6. Tính đệ quy của chương trình con	12
7. Lời gọi chương trình con	13
8. Khai báo trước bằng Forward	16
Bài tập ứng dụng chương 1	19
Chương 2: Các kiểu dữ liệu có cấu trúc	20
1. Dữ liệu kiểu bản ghi	21
2. Dữ liệu kiểu tệp	37
3. Dữ liệu kiểu tập hợp	58
Bài tập ứng dụng chương 2	67
Chương 3: Đơn vị chương trình và thư viện chuẩn	69
1. Khái niệm đơn vị chương trình	70
2. Thư viện chuẩn	70
3. Các Unit khác	71
4. Xây dựng các Unit	72
5. Tham chiếu đến các Unit	78
6. Trình tiện ích Tpmover	81
8. Một số Unit chuẩn	83
Bài tập ứng dụng chương 3	94
Chương 4: Con trỏ và cấu trúc động	95
1. Khái niệm	96
2. Kiểu dữ liệu con trỏ - Biến con trỏ	96
3. Các thủ tục và hàm tác động trên con trỏ	92
4. Truy nhập dữ liệu	94
5. Mảng con trỏ và con trỏ mảng	96
6. Cấp phát động	99
7. Danh sách liên kết và hàng đợi	105
8. Cây nhị phân	117
Bài tập ứng dụng chương 4	
Chương 5: Giải thuật đệ quy	126
1. Khái niệm đệ quy	126
2. Thiết kế giải thuật đệ quy - Khử đệ quy	130
3. Hiệu lực bài toán đệ quy	130
Bài tập ứng dụng chương 5	

Chương 6: Đồ hoạ	132
1. Khái niệm chung	132
2. Một số thủ tục cơ bản để vẽ hình	134
3. Thiết lập màu đồ hoạ	134
4. Viết chữ trong chế độ đồ hoạ	135
5. Các ví dụ	137
6. Xử lý ảnh Bitmap	144
7. Đồ thị hàm số	149
Bài tập ứng dụng chương 6	152
Phụ lục 1	
Bảng mã ASCII	161
Phụ lục 2	
Tóm tắt các thủ tục và hàm của Turbo Pascal 7.0	166
Phụ lục 3	
Định hướng biên dịch	191
Phụ lục 4	
Thông báo lỗi	180
1. Lỗi biên dịch	180
2. Lỗi liên quan đến hệ điều hành	188
3. Lỗi vào ra	188
4. Các lỗi liên quan đến phần cứng	189
5. Lỗi trình biên dịch	190
Mục lục	205
Tài liệu tham khảo	207

Tài liệu tham khảo

1. Dương Xuân Thành, Tin học đại cương, Nxb Thống kê 2003
2. Dương Xuân Thành, Giáo trình ngôn ngữ lập trình Pascal, Nxb Thống kê 2004
3. Đỗ Xuân Lôi, Cấu trúc dữ liệu và giải thuật, Nxb Khoa học và Kỹ thuật - 1998
4. Nguyễn Tô Thành, Lập trình nâng cao trên ngôn ngữ Pascal, Nxb Đại học quốc gia 2001
5. Larry Nyhoff - Sanford Leeostima, Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu, Nxb Đà Nẵng 1998
6. Quách Tuấn Ngọc, Bài tập ngôn ngữ lập trình Pascal, Nxb Thống kê 2001

Tài liệu kỹ thuật lập trình nâng cao Ma trận

MA TRẬN

Bai 71-79

```

#include <iostream>
#include <time.h>
#include <iomanip>
#define MAX 100
using namespace std;

//Nhap mang 2 chieu tu ban phim
void NhapMaTranTuBanPhim(int a[][MAX], int &d, int &c)
{
    do{
        cout<<"Nhap so dong (tu 1 den "<<MAX<<")= ";
        cin>>d;
    }while(d<1 || d>MAX);

    do{
        cout<<"Nhap so cot (tu 1 den "<<MAX<<")= ";
        cin>>c;
    }while(c<1 || c>MAX);

    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
        {
            cout<<"Nhap phan tu  ["<<i<<"]["<<j<<"]= ";
            cin>>a[i][j];
        }
}

//Bai 71: Nhap mang hai chieu ngau nhien
void NhapMaTranNgauNhien(int a[][MAX], int &d, int &c)
{
    srand((unsigned int) time (NULL));
    do{
        cout<<"Nhap so dong (tu 1 den "<<MAX<<")= ";
        cin>>d;
    }while(d<1 || d>MAX);

    do{
        cout<<"Nhap so cot (tu 1 den "<<MAX<<")= ";
        cin>>c;
    }while(c<1 || c>MAX);

    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
        {

```



```

        a[i][j]=rand() %21;
    }
    cout<<"Da tao xong ma tran";

}
//Ham xuất
void XuatMaTran(int a[][MAX], int d, int c)
{
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
        {
            cout<<setw(4)<<a[i][j];
        }
        cout<<endl;
    }
}
Bai 72:tao ma tran a cac so nguyen gom 9 dong 14 cot.Trong do
phan tu a[i][j]=i*j
#include <iostream>
using namespace std;
#include <time.h>
#include <iomanip>
#define MAX 20
void mtsonguyen(int a[][MAX], int &d, int &c)
{
    d=9;c=14;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            a[i][j]=i*j;
}
void xuatmt(int a[][MAX], int d, int n)
{
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
void main()
{
    int a[MAX][MAX], d, c;
    mtsonguyen(a,d,c);
    xuatmt(a,d,c);
}

```

```

//Bai 73: Nhap ma tran cac so nguyen duong(nhap sai bao loi
khong chon hap)
#include<iostream>
using namespace std;
#include<time.h>
#include<iomanip>
#define MAX 20
void nhapmt(int a[][MAX], int &d, int &c)
{
    do
    {
        cout<<"Nhap so dong: "; cin>>d;
    }while(d<1||d>MAX);
    do
    {
        cout<<"Nhap so cot: "; cin>>c;
    }while(c<1||c>MAX);
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            {
                do
                {
                    cout<<"a["<<i<<"] ["<<j<<"]=";
                    cin>>a[i][j];
                    if(a[i][j]<0)
                        exit(1);
                }while(a[i][j]<0);
            }
}
void xuatmt(int a[][MAX], int d, int c)
{
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
void main()
{
    int a[MAX][MAX], d, c;
    nhapmt(a,d,c);
    cout<<"\nMa tran vua nhap\n";
    xuatmt(a,d,c);
}
//bai 74: In cac phan tu co ki so tan cung la 5
void tancungla5(int a[][MAX], int dong, int cot)

```

```
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<cot; j++)
            if(a[i][j] % 10 ==5)
                cout<<setw(4)<<a[i][j];
            else
                cout<<setw(4)<<"*";
        cout<<endl;
    }
}
//bai 75: In cac phan tu nam tren hai duong cheo
void Xuat2duongcheo(int a[][MAX],int dong, int cot)
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<dong; j++)
            if(i==j||i+j== dong - 1)
                cout<<setw(4)<<a[i][j];
            else
                cout<<setw(4)<<" ";
        cout<<endl;
    }
}
//bai 76: In cac phan tu nam phia tren duong cheo phu cua ma
tran cac so nguyen
void XuatTrenDuongCheoPhu(int a[][MAX],int dong, int cot)
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<dong; j++)
            if(i+j < dong -1)
                cout<<setw(4)<<a[i][j];
            else
                cout<<setw(4)<<" ";
        cout<<endl<<endl;
    }
}
//bai 77: In cac phan tu nam phia duoi duong cheo phu cua ma
tran vuong cac so nguyen
void XuatDuoiDuongCheoPhu(int a[][MAX],int dong, int cot)
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<dong; j++)
            if(i+j > dong -1)
```

```
        cout<<setw(4)<<a[i][j];
    else
        cout<<setw(4)<<" ";
    cout<<endl<<endl;
}
}
//bai 78: In cac phan tu nam phia tren duong cheo chinh cua ma
tran vuong cac so nguyen
void XuatTrenDuongCheoChinh(int a[][MAX], int dong, int cot)
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<dong; j++)
            if(i<j)
                cout<<setw(4)<<a[i][j];
            else
                cout<<setw(4)<<" ";
        cout<<endl<<endl;
    }
}
//bai 79: In cac phan tu nam phia duoi duong cheo chinh cua ma
tran vuong cac so nguyen
void XuatDuoiDuongCheoChinh(int a[][MAX], int dong, int cot)
{
    for(int i=0; i<dong; i++)
    {
        for(int j=0; j<cot; j++)
            if(i>j)
                cout<<setw(4)<<a[i][j];
            else
                cout<<setw(4)<<" ";
        cout<<endl<<endl;
    }
}

void main()
{
    int a[MAX][MAX], dong, cot;
    char chon;
    while(1)
    {
        cout<<"Chuong trinh mang hai chieu:\n\n";
        cout<<"a.Nhap ma tran tu ban phim\n";
        cout<<"b.Nhap ma tran ngau nhien\n";
        cout<<"c.Xuat ma tran\n";
        cout<<"d.Xuat nhung phan tu co ky so tan cung la 5\n";
        cout<<"e.Xuat cac phan tu nam tren hai duong cheo\n";
    }
}
```

```

cout<<"f.Xuat phan tu tren duong cheo phu\n";
cout<<"g.Xuat phan tu duoi duong cheo phu\n";
cout<<"h.Xuat phen tu tren duong cheo chinh\n";
cout<<"i.Xuat phen tu tren duong cheo chinh\n";
cout<<"z.Thoat\n";
cout<<"Moi ban chon muc: ";
cin>>chon;
switch(chon)
{
case 'a':nhapmang2chieubanphim(a,dong,cot);
break;
case 'b':NhapMaTranNN(a,dong,cot);
break;
case 'c':xuatmang2chieu(a,dong,cot);
break;
case 'd':tancungla5(a,dong,cot);
case 'e':Xuat2duongcheo(a,dong,cot);
case 'f':XuatTrenDuongCheoPhu(a,dong,cot);
case 'g':XuatDuoiDuongCheoPhu(a,dong,cot);
case 'h':XuatTrenDuongCheoChinh(a,dong,cot);
case 'i':XuatDuoiDuongCheoChinh(a,dong,cot);
case 'z':exit(1);
cout<<"\n\n\n\n\n\n";
}
}
}

```

Bai 80-86

```

#include <iostream>
using namespace std;
#define MAX 100
#include <iomanip>
#include <time.h>

void NhapMaTran(int a[][MAX],int &d, int &c)
{
do
{
cout<<"Nhap so dong (tu 1 den "<<MAX<<" ) : ";
cin>>d;
}while((d<1) || (d>MAX));
do
{
cout<<"Nhap so cot (tu 1 den "<<MAX<<" ) : ";
cin>>c;
}while((c<1) || (c>MAX));
for(int i=0; i<d; i++)

```

```

        for(int j=0; j<c; j++)
        {
            cout<<"Nhap ptu ["<<i<<"] ["<<j<<"] =";
            cin>>a[i][j];
        }
    }
void NhapMaTranNN(int a[][MAX],int &d,int &c)
{
    srand((unsigned)time(NULL));
    do
    {
        cout<<"Nhap so dong (tu 1 den "<<MAX<< " ) : ";
        cin>>d;
    }while((d<1) || (d>MAX));
    do
    {
        cout<<"Nhap so cot (tu 1 den "<<MAX<< " ) : ";
        cin>>c;
    }while((c<1) || (c>MAX));
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
            a[i][j]=rand()%100;
    cout<<"\nDa tao xong ma tran";
}
void XuatMaTran(int a[][MAX],int d,int c)
{
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
/*-----*/
//bai 80: Tinh tong cac phan tu tren cung mot dong
int TongDongK(int a[][MAX],int c,int k)
{
    int S=0;
    for(int j=0; j<c; j++)
        S=S+a[k][j];
    return S;
}
//bai 81: Tinh tong cac phan tu tren cung mot cot
int TongCotK(int a[][MAX],int d,int k)
{
    int S=0;
    for(int i=0; i<d; i++)

```

```
        S=S+a[i][k];
    return S;
}
//bai82: Tinh tong cac ptu chan trong ma tran
int TongChan(int a[][MAX],int d,int c)
{
    int S=0;
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
            {
                if(a[i][j]%2==0)
                    S=S+a[i][j];
            }
    return S;
}
//bai 83: Tinh tong cac phan tu thuoc duong cheo chinh cua ma
tran vuong
int TongDuongCheoChinh(int a[][MAX],int d)
{
    int S=0;
    for(int i=0; i<d; i++)
        {
            for(int j=0; j<c; j++)
                if(i==j)
                    S=S+a[i][j];
        }
    return S;
}
//bai 84: Tinh tong cac phan tu la so nguyen to co trong ma tran
bool LaNguyenTo(int k)
{
    if(k<2)
        return false;
    for(int i=2;i<k;i++)
        if(k%i==0)
            return false;
    return true;
}
int TongNguyenTo(int a[][MAX],int d, int c)
{
    int S1=0;
    for(int i=0;i<d;i++)
        {
            for(int j=0;j<c;j++)
                if(LaNguyenTo(a[i][j])==true)
                    S1=S1+a[i][j];
        }
}
```

```

    return S1;

}
//bai 85: Tinh tong cac so hoan thien trong ma tran so
nguyen(giai rieng)
//bai 86: Tinh tong cac phan tu lon nhat tren moi dong
int TongMaxDong(int a[][MAX], int d, int c)
{
    int s=0, maxdong=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if(a[i][j]>a[i][maxdong])
                maxdong=j;
        s= s+ a[i][maxdong];
    }
    return s;
}

void main()
{
    int a[MAX][MAX],d,c;
    char chon;
    while(1)
    {
        cout<<"Chuong trinh mang 2 chieu\n";
        cout<<"a.Nhap ma tran ngau nhien\n";
        cout<<"b.Nhap ma tran tu ban phim\n";
        cout<<"c.Xuat ma tran\n";
        cout<<"d.Tong ptu tren cung dong(80)\n";
        cout<<"e.Tong ptu tren cung cot(81)\n";
        cout<<"f.Tong cac ptu chan(82)\n";
        cout<<"g.Tong duong cheo chinh(83)\n";
        cout<<"h.Tong cac phan tu la so nguyen to:(84)\n";
        cout<<"i.Tong cac gia tri lon nhat tren moi
dong: (86)\n";

        cout<<"z.Thoat\n";
        cout<<"Moi ban chon muc :";
        cin>>chon;
        switch(chon)
        {
            case 'a': NhapMaTranNN(a,d,c); break;
            case 'b': NhapMaTran(a,d,c); break;
            case 'c': XuatMaTran(a,d,c); break;
            case 'd':
            {

```



```

        int k;
        cout<<"Nhap dong can tinh tong : ";
        cin>>k;
        cout<<"Tong cac ptu trendong
"<<k<<"la:"<<TongDongK(a,c,k);
        cout<<endl;
    } break;
    case 'e':
    {
        int k;
        cout<<"Nhap cot can tinh tong : ";
        cin>>k;
        cout<<"Tong cac ptu tren cot
"<<k<<"la:"<<TongCotK(a,d,k)<<endl;
        cout<<endl;
    } break;
    case 'f':
    {
        cout<<"Tong cac ptu chan la :
"<<TongChan(a,d,c)<<endl;
        cout<<endl;
    } break;
    case 'g':
    {
        cout<<"Tong duong cheo chinh
la:"<<TongDuongCheoChinh(a,d)<<endl;
        cout<<endl;
    } break;
    case 'h':
    {
        cout<<"tong cac phan tu la so nguyen
to:\n"<<TongNguyenTo(a,c,d)<<endl;
    } break;
    case 'i':
    {
        cout<<"Tong cac gia tri lon nhat tren moi
dong\n"<<TongMaxDong(a,d,c)<<endl;
    }

    case 'z': exit(1);
    }
    cout<<"\n\n\n\n";
}
}

```

Bai 85:Tinh tong cac so hoan thien trong ma tran cac so nguyen

```

bool LaSoHoanThien(int k)
{

```

```

    int tong=0;
    for(int i=1; i<k; i++)
        if(k%i==0)
            tong=tong + i;
    if(tong == k)
        return true;
    return false;
}
int TongSoHoanThien(int a[][MAX], int d, int c )
{
    int tong=0;
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(LaSoHoanThien(a[i][j])== 1)
                tong = tong + a[i][j];
    }
    return tong;
}

```

Bai 86: Tinh tong cac gia tri lon nhat tren moi dong

//Tong cac phan tu lon nhat tren moi dong

```

int TongPTLNDong(int a[][MAX], int d, int c)
{
    int s=0;
    int maxdong=0;
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(a[i][j]>a[i][maxdong])
                maxdong=j;
        s=s + a[i][maxdong];
    }
    return s;
}

```

Bai 87: Tinh gia tri trung binh cua cac phan tu nho nhat tren moi cot

```

float AverageItemsMinColumn(int a[][MAX], int &d)
{
    int mincot=0;
    float s=0;
    int i,j;
    for( i=0; i<d; i++)
    {
        for(j=0; j<d; j++)
        {
            if(a[i][j]<a[mincot][j])
                mincot=i;
        }
    }
}

```

```

        s=s + a[mincot][j];
    }
    return (float)s/d;
}

```

Bai 88: Tinh tong cac gia tri nho nhat thuc tung duong cheo song song voi duong cheo chinh

```

void InpuMatrixRandom(int A[][MAX], int &n)
{
    cout<<"Enter n: ";
    cin>>n;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            A[i][j]=rand()%20;
}
void OuputMatrixRandom(int A[][MAX], int n)
{
    cout<<"All element random in matrix:\n";
    cout<<"\n";
    cout<<"\n";
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            cout<<"\t"<<A[i][j];
        cout<<"\n";
        cout<<"\n";
        cout<<"\n";
    }
}
//Tinh tong cac gia tri nho nhat thuc tung duong cheo song song
voi duong cheo chinh
int TongDC_min1(int a[][MAX], int &d, int &k)
{
    int min, tong=0;
    for(int i=0; i<d ;i++)
    {
        k=i;
        min=a[k][0];
        for(int j=1; j<=d-1-i; j++)
        {
            if(a[k+1][j]<min)
                min=a[k+1][j];
            k++;
        }
        tong+=min;
    }
}

```

```

        return tong;
    }
int TongDC_min2(int a[][MAX], int &d, int &k)
{
    int min, tong=0;
    for(int j=d-1; j>0; j--)
    {
        min=a[0][j];
        for(int i=1; i<=d-1-j; i++)
        {
            if(a[i][k+1]<min)
                min=a[i][k+1];
            k++;
        }
        tong+=min;
    }
    return tong;
}

float TongDC_NhoNhat(int a[][MAX], int &d, int &k)
{
    return TongDC_min1(a,d,k)+TongDC_min2(a,d,k);
}

```

Bai 89: Tim duong cheo co tong lon nhat thuoc tung duong cheo song song voi duong cheo phu.

Bai 90: Tim vi tri phan tu lon nhat trong ma tran cac so nguyen

```

void TimVTPhanTuMax(int a[][MAX],int d,int c, int &vtd, int
&vtc)
{
    vtd=0;
    vtc=0;
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(a[i][j]>a[vtd][vtc])
            {
                vtd = i;
                vtc = j;
            }
    }
}
//bai 91: Tim vi tri phan tu nho nhat trong ma tran cac so
nguyen
void NhoNhat(int a[][MAX],int d, int c,int &vtd,int &vtc)

```

```

{
    vtd=vtc=0;
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(a[i][j]<a[vtd][vtc])
            {
                vtd=i;
                vtc=j;
            }
    }
}
void main()
{
    int a[MAX][MAX],d,c;
    int vtd,vtc;
    TimVTPhanTuMax(a,d,c,vtd,vtc);
    cout<<"Phan tu lon nhat la : "<<a[vtd][vtc]<<endl;
    cout<<"Nam o vi tri a["<<vtd<<","<<vtc<<"]";
}
//bai 92: Tim vi tri phan tu chan cuoi trong ma tran cac so
nguyen
void ChanCuoi(int a[][MAX],int d, int c,int &vtd,int &vtc)
{
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(a[i][j]%2==0)
            {
                vtd=i;
                vtc=j;
            }
    }
}

//bai 93: Tim phan tu am le lon nhat trong ma tran(sai)
bool AmLeLonNhat(int a[][MAX],int d, int c,int &vtd,int &vtc)
{
    vtd=vtc=0;
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            if(a[i][j]<0 && a[i][j]%2!=0 &&
a[i][j]>a[vtd][vtc])
            {

```

```

        vtd=i;
        vtc=j;
        return true;
    }
    return false;
}

}
94. Viết hàm tìm phần tử chẵn dương và nhỏ nhất trong ma trận.
#include<iostream>
using namespace std;
#include<time.h>
#include<iomanip>
#define MAX 10
void mtnn(int a[][MAX], int &d, int &c)
{
    d=1+rand()%MAX;
    c=1+rand()%MAX;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            a[i][j]=rand()%100;
    cout<<"\nDa tao xong mang ngau nhien\n";
}
void xuatmt(int a[][MAX], int d, int c)
{
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
int chandungdau(int a[][MAX], int d, int c,int &n)
{
    for(int i=0;i<d*c;i++)
        if(a[i/n][i%n]>0 && a[i/n][i%n]%2==0)
            return i;
    return -1;
}
void minchandung(int a[][MAX], int d, int c, int &n)
{
    if(chandungdau(a,d,c,n)!=-1)
    {
        int minc=chandungdau(a,d,c,n)%n;
        int mind=chandungdau(a,d,c,n)/n;
        for(int i=0; i<d;i++)

```

```

        for(int j=0;j<c;j++)
            if(a[i][j]<a[mind][minc] && a[i][j]%2==0)
            {
                minc=j;
                mind=i;
            }
        cout<<a[mind][minc];
    }
    else
        cout<<"Khong tim thay";
}
void main()
{
    int a[MAX][MAX], d, c;
    srand((unsigned) time (NULL));
    mtnn(a,d,c);
    xuatmt(a,d,c);
    minchanduong(a,d,c,n);
}

void main()
{
    int a[MAX][MAX],d,c;
    char chon;
    while(1)
    {
        cout<<"CHUONG TRINH MANG HAI CHIEU\n\n\n";
        cout<<"a.Nhap mang tran ngau nhien\n";
        cout<<"b.Nhap ma tran tu ban phim\n";
        cout<<"c.Xuat ma tran\n";
        cout<<"d.Tim ptu lon nhat\n";
        cout<<"e.Tim ptu nho nhat\n";
        cout<<"f.Tim ptu chan cuoi\n";
        cout<<"g.Tim ptu am le lon nhat\n";
        cout<<"h.Tim phan tu chan duong nho nhat\n";
        cout<<"z.Thoat\n";
        cout<<"Moi ban chon muc :";
        cin>>chon;
        switch(chon)
        {
            case 'a': NhapMaTranNN(a,d,c); break;
            case 'b': NhapMaTran(a,d,c); break;
            case 'c': XuatMaTran(a,d,c); break;
            case 'd':
            {
                int vtd,vtc;
                LonNhat(a,d,c,vtd,vtc);
            }
        }
    }
}

```

```

        cout<<"Ptu lon nhat la : "<<a[vtd][vtc];
    }
    break;
case 'e':
{
    int vtd,vtc;
    NhoNhat(a,d,c,vtd,vtc);
    cout<<"Ptu nho nhat la : "<<a[vtd][vtc];
}
    break;
case 'f':
{
    int vtd,vtc;
    ChanCuoi(a,d,c,vtd,vtc);
    cout<<"Ptu chan cuoi la : "<<a[vtd][vtc];
}
    break;
case 'g':
{
    int vtd,vtc;
    bool kq=AmLeLonNhat(a,d,c,vtd,vtc);
    if(kq==true)
        cout<<"Ptu am le lon nhat la :
"<<a[vtd][vtc];
    else
        cout<<"Khong tim thay";
    cout<<endl;
}
    break;
case 'h':
{
    cout<<"Phan tu chan duong nho nhat trong ma tran
la: "<<ChanDuong_NhoNhat(a,d,c);
    cout<<endl;
}
    break;
case 'z': exit(1);
}
cout<<"\n\n\n\n";
}
}

```

Bai 95:Tim so hoan thien dau tien trong ma tran so nguyen

```

#include <iostream>
#include <time.h>
using namespace std;
#define MAX 10

```



```

void InpuMAtrixRandom(int A[][MAX], int &n)
{
    cout<<"nhap n: ";
    cin>>n;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            A[i][j]=rand()%51;
}
void OuputMAtrixRandom(int A[][MAX], int n)
{
    cout<<"All element random in matrix:\n";
    cout<<"\n";
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
}
//ham kiem tra so hoan thien
bool Test_sht(int k)
{
    int s=0;
    for(int i=1; i<k; i++)
        if(k%i==0)
            s+=i;
    if(s==k)
        return 1;
    return 0;
}
//ham tim so hoan thien dau tien trong ma tran
int SearchFirst_sht(int A[][MAX], int n, int &v, int &d)
{
    for(int i=0; i<n ;i++)
        for(int j=0; j<n; j++)
            if(Test_sht(A[i][j]) == 1)
            {
                v=i;
                d=j;
                return A[i][j];
            }
    return 0;
}
}
Cach 2:
bool ktht(int n)
{

```

```

    int s=0;
    for(int i=1;i<n;i++)
        if(n%i==0)
            s=s+i;
    if(s==n)
        return true;
    else
        return false;
}
int htdau(int a[][MAX], int m, int n)
{
    for(int i=0;i<m*n;i++)
        if(ktht(a[i/n][i%n]))
            return i;
    return -1;
}
void main()
{
    int A[MAX][MAX], n, d, v;
    InpuMAtrixRandom(A,n);
    cout<<"\n";
    OuputMAtrixRandom(A,n);
    int kq=SearchFirst_sht(A,n,v,d);
    if(kq==0)
        cout<<"\nHasn't element SHT in matrix\n";
    else
        cout<<"\nElement first SHT int matrix is "<<kq<<" and
have location is ("<<v<<","<<d<<")\n";
    cout<<"\n";
}

```

Bai 96: Tim so hoan thien lon nhat trong ma tran so nguyen(ok)

```

#include <iostream>
#include <iomanip>
#include <time.h>
#define MAX 100
using namespace std;

void NhapMaTran(int a[][MAX],int &d, int &c)
{
    do
    {
        cout<<"Nhap so d (tu 1 den "<<MAX<<" ) : ";
        cin>>d;
    }while((d<1) || (d>MAX));
    do
    {
        cout<<"Nhap so c (tu 1 den "<<MAX<<" ) : ";

```

```

        cin>>c;
    }while((c<1) || (c>MAX));
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
        {
            cout<<"Nhap ptu ["<<i<<"] ["<<j<<"] =";
            cin>>a[i][j];
        }
}
void NhapMaTranNN(int a[][MAX],int &d,int &c)
{
    srand((unsigned)time(NULL));
    do
    {
        cout<<"Nhap so d (tu 1 den "<<MAX<< " ) : ";
        cin>>d;
    }while((d<1) || (d>MAX));
    do
    {
        cout<<"Nhap so c (tu 1 den "<<MAX<< " ) : ";
        cin>>c;
    }while((c<1) || (c>MAX));
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
            a[i][j]=rand()%100;
    cout<<"\nDa tao xong ma tran";
}
void XuatMaTran(int a[][MAX],int d,int c)
{
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}

int sohoanthien(int n)
{
    int tong=0;
    for (int i=1 ; i<n; i++)
        if(n%i==0)
            tong = tong + i;
    if (tong == n)
        return 1;
    return 0;
}

```

```
int SoHoanThienDauTien(int a[][100], int d, int c)
{
    for (int i=0; i<d; i++)
    {
        for (int j=0; j<c; j++)
        {
            if (sohoanthien(a[i][j])==1)
            {
                return a[i][j];
            }
        }
    }
    return -1;
}

int GiaTriCanTim(int a[][100], int d, int c)
{
    int SoHoanThienMax = SoHoanThienDauTien(a,c,d);
    if (SoHoanThienMax == -1)
    {
        return -1;
    }
    for (int i=0; i<d;i++)
    {
        for (int j=0; j<c; j++)
        {
            if (sohoanthien(a[i][j])==1)
            {
                SoHoanThienMax =
                (SoHoanThienMax>a[i][j])?SoHoanThienMax:a[i][j];
            }
        }
    }
    return SoHoanThienMax;
}

void main()
{
    int a[MAX][MAX],d,c;
    char chon;
    while(1)
    {
        cout<<"Chuong trinh mang 2 chieu\n";
        cout<<"a.Nhap ma tran ngau nhien\n";
        cout<<"b.Nhap ma tran tu ban phim\n";
        cout<<"c.Xuat ma tran\n";
        cout<<"d.Tim so hoan thien lon nhat\n";
        cout<<"z.Thoat\n";
    }
}
```

```

        cout<<"Moi ban chon muc :";
        cin>>chon;
        switch(chon)
        {
        case 'a': NhapMaTranNN(a,d,c); break;
        case 'b': NhapMaTran(a,d,c); break;
        case 'c': XuatMaTran(a,d,c); break;
        case 'd':
            {
                cout<<"So hoan thien lon nhat
la:\n"<<GiaTriCanTim(a,d,c)<<endl;
                break;
            }
        case 'z':exit(1);
        }
        cout<<"\n\n\n\n";
    }
}

```

Bai 97: Tim vi tri phan tu nguyen to cuoi cung trong ma tran so nguyen

```

#include <iostream>
using namespace std;
#define MAX 100
#include <iomanip>
#include <time.h>
void InpuMatrixRandom(int A[][MAX], int &n)
{
    cout<<"nhap n: ";
    cin>>n;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            A[i][j]=rand()%51;
}
void OuputMatrixRandom(int A[][MAX], int n)
{
    cout<<"All element random in matrix:\n";
    cout<<"\n";
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
}
//ham kiem tra so hoan thien
bool Test_sht(int k)

```

```

{
    int s=0;
    for(int i=1; i<k; i++)
        if(k%i==0)
            s+=i;
    if(s==k)
        return 1;
    return 0;
}
//ham kiem tra so nguyen to
bool Test_snt(int k)
{
    int d=0;
    for(int i=1; i<=k; i++)
        if(k%i==0)
            d++;
    if(d==2)
        return 1;
    return 0;
}
//ham tim so nguyen to cuoi cung trong ma tran
int SearchLast_snt(int A[][MAX], int n, int &v, int &d)
{
    for(int i=n-1; i>=0 ;i--)
        for(int j=n-1; j>=0; j--)
            if(Test_snt(A[i][j]) == 1)
            {
                v=i;
                d=j;
                return A[i][j];
            }
    return 0;
}
Cach 2:
bool ktnt(int n)
{
    if(n<2)
        return false;
    for(int i=2;i<n;i++)
        if(n%i==0)
            return false;
    return true;
}
int ngtocuoi(int a[][MAX], int d, int c)
{
    for(int i=d-1;i>=0;i--)
        for(int j=c-1;j>=0;j--)

```

```

        if (ktnt(a[i][j])==true)
            return a[i][j];
    return -1;
}

void main()
{
    int A[MAX][MAX], n, v, d;
    InpuMatrixRandom(A,n);
    cout<<"\n";
    OuputMatrixRandom(A,n);
    int kq=SearchLast_snt(A,n,v,d);
    if(kq==0)
        cout<<"\nHaven't element SNT in matrix\n";
    else
        cout<<"\nElement last SNT in matrix is "<<kq<<" and
have location is ("<<v<<","<<d<<")\n";
    cout<<"\n";
}

```

Bai 98:Tim phan tu lon nhat thuoc duong cheo chinh cua ma tran vuong

```

#include <iostream>
#include <time.h>
using namespace std;
#define MAX 10
void InpuMatrixRandom(int A[][MAX], int &n)
{
    cout<<"nhap n: ";
    cin>>n;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            A[i][j]=rand()%51;
}
void OuputMatrixRandom(int A[][MAX], int n)
{
    cout<<"All element random in matrix:\n";
    cout<<"\n";
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
}
int SearchMax(int A[][MAX], int n)
{

```

```

    int max=A[0][0];
    for(int i=1; i<n; i++)
        if(A[i][i]>max)
            max=A[i][i];
    return max;
}
void main()
{
    int A[MAX][MAX], n, d, c;
    InpuMAtrixRandom(A,n);
    cout<<"\n";
    OuputMAtrixRandom(A,n);
    cout<<"\nElement max in nmatrix is "<<SearchMax(A,n)<<"\n";
    cout<<"\n";
}

```

Bai 99:Tim trong hai ma tran cac so nguyen nhung phan tu giong nhau

```

#define MAX 100
void nhap(int &n, int &m)
{
    cout<<"Nhap n: "; cin>>n;
    cout<<"Nhap m: "; cin>>m;
}
void input(int A[], int n)
{
    for(int i=0; i<n ;i++)
    {
        cout<<"A["<<i<<"]=" ";
        cin>>A[i];
    }
}
void output(int A[], int n)
{
    for(int i=0; i<n; i++)
        cout<<A[i]<<" ";
}
void input1(int B[], int m)
{
    for(int i=0; i<m ;i++)
    {
        cout<<"B["<<i<<"]=" ";
        cin>>B[i];
    }
}
void output1(int B[], int m)
{

```



```

        for(int i=0; i<m; i++)
            cout<<B[i]<<" ";
    }
void Tim(int A[], int B[], int m, int n)
{
    for(int i=0; i<n ;i++)
        for(int j=0; j<m; j++)
            if(A[i]==B[j])
                cout<<A[i]<<" ";
}
void main()
{
    int A[MAX], B[MAX], n, m;
    nhap(n,m);
    input(A,n);
    cout<<"\nCac phan tu trong mang A: ";
    output(A,n);
    cout<<"\n";
    input1(B,m);
    cout<<"\nCac phan tu trong mang B: ";
    output1(B,m);
    cout<<"\n";
    cout<<"\nCac phan tu giong nhau la: ";
    Tim(A,B,m,n);
    cout<<"\n";
}

```

Bai 100:

Bai 101: Viết hàm tìm và liệt kê những phần tử cực đại trong ma trận (một phần tử được coi là cực đại khi nó lớn hơn các phần tử xung quanh nó - trái, phải, trên và dưới).

```

#define MAX 10
void InpuMAtrixRandom(int A[][MAX], int &n)
{
    cout<<"Enter n: ";
    cin>>n;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            A[i][j]=rand()%51;
}
void OuputMAtrixRandom(int A[][MAX], int n)
{
    cout<<"All element random in matrix:\n";
}

```

```
    cout<<"\n";
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
}
//ham kiem tra so hoan thien
bool Test_sht(int k)
{
    int s=0;
    for(int i=1; i<k; i++)
        if(k%i==0)
            s+=i;
    if(s==k)
        return 1;
    return 0;
}
//ham kiem tra so nguyen to
bool Test_snt(int k)
{
    int d=0;
    for(int i=1; i<=k; i++)
        if(k%i==0)
            d++;
    if(d==2)
        return 1;
    return 0;
}
//ham tim cot co tong nho nhat trong ma tran
int SearchSumMin(int A[][MAX], int n, int &d, int &c)
{
    int s1=0, s2=0;;
    for(int i=0; i<n ;i++)
        for(int j=0; j<n; j++)
            s1+=A[i][j];
    int min=s1;
    for(int i=1; i<n; i++)
    {
        for(int j=0; j<n; j++)
            s2+=A[i][j];
        if(s2<min)
        {
            d=i;
            c=j;
            min=s2;
        }
    }
}
```

```

    }
    }
    return min;
}
void main()
{
    int A[MAX][MAX], n, d, c;
    InpuMAtRixRandom(A,n);
    cout<<"\n";
    OuputMAtRixRandom(A,n);
    cout<<"\nColumn have sum min in matrix is
("<<d<<","<<c<<")\n";
    cout<<"\n";
}

```

Bai 102: Viet ham tim cot co tong nho nhat trong ma tran so nguyen

Bai 103-Bai 109

```

#include <iostream>
#include <iomanip>
#include <time.h>
#include <windows.h>
#include <conio.h>
#define MAX 10
using namespace std;
void NhapMaTran(int a[][MAX], int &d, int &c)
{
    do{
        cout<<"Nhap so song (tu 1 den "<<MAX<<)"=";
        cin>>d;
    }while(d<1||d>MAX);

    do{
        cout<<"Nhap so cot (tu 1 den "<<MAX<<)"=";
        cin>>c;
    }while(c<1||c>MAX);
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
        {
            cout<<"Nhap phan tu thu ["<<i<<"]["<<j<<"]=" ";
            cin>>a[i][j];
        }
}
void NhapMaTranNgauNhien(int a[][MAX], int &d, int &c)
{
    srand((unsigned) time (NULL));
}

```

```
do{
    cout<<"Nhap so song (tu 1 den "<<MAX<<")=";
    cin>>d;
}while(d<1||d>MAX);

do{
    cout<<"Nhap so cot (tu 1 den "<<MAX<<")=";
    cin>>c;
}while(c<1||c>MAX);
for(int i=0;i<d;i++)
{
    for(int j=0;j<c;j++)
        a[i][j]=rand()%20;
}
cout<<"Da tao xong ma tran";
cout<<endl;
}
void Xuat(int a[][MAX], int &d, int &c)
{
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
//bai 103 dem cac gia tri le trong ma tran cac so nguyen
int DemGTLe(int a[][MAX], int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if(a[i][j]%2!=0)
                count++;
    }
    return count;
}
//bai 104 dem cac gia tri nho hon x trong ma tran cac so nguyen
int DemGTNhoHonX(int a[][MAX], int d, int c)
{
    int x;
    cout<<"Moi ban nhap X ";cin>>x;
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
```

```

        if(a[i][j]<x)
            count ++;
    }
    return count;
}
//bai 105 dem cac phan tu la so nguyen to trong ma tran
bool KT_SNTTo(int k)
{
    if(k<2)
        return false;
    for(int i=2;i<k;i++)
        if(k%i==0)
            return false;
    return true;
}
int DemPT_NguyenTo(int a[][MAX], int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if(KT_SNTTo(a[i][j])==true)
                count ++;
    }
    return count;
}
//bai 106 dem cac gia tri cuc dai trong ma tran cac so nguyen
int Dem_PT_CucDai(int a[][MAX], int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if(a[i][j]>a[i][j-1]
                && a[i][j]>a[i][j+1]
                && a[i][j]>a[i-1][j]
                && a[i+1][j])
                count ++;
    }
    return count;
}
//bai 107 dem cac gia tri cuc tieu trong ma tran cac so nguyen
int Dem_PT_CucTieu(int a[][MAX], int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)

```

```

    {
        for(int j=0;j<c;j++)
            if(a[i][j]<a[i][j-1]
                && a[i][j]<a[i][j+1]
                && a[i][j]<a[i-1][j]
                && a[i+1][j])
                count ++;
    }
    return count;
}
//dem cac cuc tri trong ma tran cac so nguyen
int Dem_Cuc_Tri(int a[][MAX], int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if((a[i][j]>a[i][j-1]
                && a[i][j]>a[i][j+1]
                && a[i][j]>a[i-1][j]
                && a[i+1][j])
                ||a[i][j]<a[i][j-1]
                && a[i][j]<a[i][j+1]
                && a[i][j]<a[i-1][j]
                && a[i+1][j])
                count ++;
    }
    return count;
}
bool KiemTra_So_Hoan_thien(int k)
{
    int tong=0;
    for(int i=1;i<k;i++)
        if(k%i==0)
            tong=tong+i;
    if(tong==k)
        return true;
    return false;
}
int DemSoHoanThien(int a[][MAX],int d, int c)
{
    int count=0;
    for(int i=0;i<d;i++)
    {
        for(int j=0;j<c;j++)
            if(KiemTra_So_Hoan_thien(a[i][j])==true)
                count++;
    }
}

```

```

    }
    return count;
}
int in_menu()
{
    int cv;
    cout<<"\n\n";
    cout<<"-----CHUONG TRINH MANG HAI CHIEU-----\n\n";
    cout<<"1.Nhap ma tran tu ban phim\n";
    cout<<"2.Nhap ma tran ngau nhien\n";
    cout<<"3.Xuat\n";
    cout<<"4.Dem gia tri le\n";
    cout<<"5.Dem cac gia tri nho hon x\n";
    cout<<"6.Dem cac phan tu la so nguyen to\n";
    cout<<"7.Dem phan tu cuc dai\n";
    cout<<"8.Dem phan tu cuc tieu\n";
    cout<<"9.Dem cuc tri\n";
    cout<<"10.Dem cac so hoan thien trong ma tran\n";
    cout<<"0.End programme!\n";
    do{
        cout<<"Hay chon cong viec: ";
        cin>>cv;

        }while(cv<0||cv>11);
    return cv;
}
void main()
{
    int a[MAX][MAX], d, c;
    int cv;

    do
    {
        cv=in_menu();
        system("cls");
        switch(cv)
        {
            case 1:NhapMaTran(a,d,c);break;
            case 2:NhapMaTranNgauNhien(a,d,c);break;
            case 3:Xuat(a,d,c);break;
            case 4:
                {
                    cout<<"Cac gia tri le trong ma trn la:
"<<DemGTLe(a,d,c)<<endl;
                    }break;
            case 5:
                {

```

```

        cout<<"Cac phan tu nho hon X trong ma tran
la: "<<DemGTNhoHonX(a,d,c)<<endl;
    }
    case 6:
    {
        cout<<"Cac phan tu la so nguyen to trong ma
tran la: "<<DemPT_NguyenTo(a,d,c)<<endl;
    }break;
    case 7:
    {
        cout<<"Cac phan tu cuc dai trong ma tran la:
"<<Dem_PT_CucDai(a,d,c)<<endl;
    }break;
    case 8:
    {
        cout<<"Cac phan tu cuc tieu trong ma tran
la: "<<Dem_PT_CucTieu(a,d,c)<<endl;
    }break;
    case 9:
    {
        cout<<"Cac cuc tri trong ma tran la:
"<<Dem_Cuc_Tri(a,d,c)<<endl;
    }break;
    case 10:
    {
        cout<<"cac so hoan thien trong ma tran la:
"<<DemSoHoanThien(a,d,c)<<endl;
    }break;
    case 0: cout<<"Exit Programme";break;
    cout<<"\n\n";
    }
    _getch();
}while(cv!=0);

```

```

}
Bai 110-Bai 113
#include <iostream>
#include <iomanip>
#include <time.h>
#define MAX 100
using namespace std;

void NhapMaTran(int a[][MAX],int &d, int &c)
{
    do
    {
        cout<<"Nhap so dong (tu 1 den "<<MAX<<" ) : ";

```



```

        cin>>d;
    }while((d<1) || (d>MAX));
    do
    {
        cout<<"Nhap so cot (tu 1 den "<<MAX<<" ) : ";
        cin>>c;
    }while((c<1) || (c>MAX));
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
        {
            cout<<"Nhap ptu ["<<i<<"]["<<j<<"] =";
            cin>>a[i][j];
        }
    }
void NhapMaTranNN(int a[][MAX],int &d,int &c)
{
    srand((unsigned)time(NULL));
    do
    {
        cout<<"Nhap so dong (tu 1 den "<<MAX<<" ) : ";
        cin>>d;
    }while((d<1) || (d>MAX));
    do
    {
        cout<<"Nhap so cot (tu 1 den "<<MAX<<" ) : ";
        cin>>c;
    }while((c<1) || (c>MAX));
    for(int i=0; i<d; i++)
        for(int j=0; j<c; j++)
            a[i][j]=rand()%100;
    cout<<"\nDa tao xong ma tran";
}
void XuatMaTran(int a[][MAX],int d,int c)
{
    for(int i=0; i<d; i++)
    {
        for(int j=0; j<c; j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl;
    }
}
bai 110
void SapXepTang(int a[], int n)
{
    for(int i=0; i<n; i++)
        for(int j= i +1; j<n; j++)
            if(a[i]>a[j])

```

```
        {
            int temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }
}
void SapxepMaTranTang(int a[][MAX], int d, int c)
{
    int b[100], l=0;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            b[l++]=a[i][j];
    SapXepTang(b,l);
    l=0;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            a[i][j]=b[l++];
}
//bai 111
void SapXepGiam(int a[], int n)
{
    for(int i=0; i<n; i++)
        for(int j= i +1; j<n; j++)
            if(a[i]<a[j])
            {
                int temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
}
void SapxepMaTranGiam(int a[][MAX], int d, int c)
{
    int b[100], l=0;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            b[l++]=a[i][j];
    SapXepGiam(b,l);
    l=0;
    for(int i=0;i<d;i++)
        for(int j=0;j<c;j++)
            a[i][j]=b[l++];
}
}
```

bai 112: Sap xep tat ca ca dong deu tang dan

```
void SapXepDongTangDan(int a[][MAX], int d, int c)
{
    int i,j;
    for(int i=0;i<d;i++)
    for(int j=0; j<c-1; j++)
    {
        for(int k=j+1; k<c; k++)
        {
            if(a[i][j]>a[i][k])
            {
                int tam=a[i][j];
                a[i][j]=a[i][k];
                a[i][k]=tam;
            }
        }
    }
}
```

bai 113: sap xep tat ca ca cot giam dan

```
void SapXepCotGiamDan(int a[][MAX], int d, int c)
{
    for(int i=0; i<d-1; i++)
        for(int j=0;j<c;j++)
        {
            for(int k=i+1; k<d; k++)
            {
                if(a[i][j]<a[k][j])
                {
                    int tam=a[i][j];
                    a[i][j]=a[k][j];
                    a[k][j]=tam;
                }
            }
        }
}
```

bai 114:

```
void SapXep(int a[][100],int d, int c)
{
    int s=0;
    for(int dc=1;dc<d-1;dc++)
    {
        int b[100], c[100],i=0,j=dc,l=0;
        while(i<(d-dc) && (j<d))
```

```

        {
            a[i][j]=b[l];
            c[l++]=a[j++][i++];
        }
        SapXep(b,l);
    }

void main()
{
    int a[MAX][MAX],d,c;
    char chon;
    while(1)
    {
        cout<<"CHUONG TRINH MANG HAI CHIEU\n\n\n";
        cout<<"a.Nhap mang tran ngau nhien\n";
        cout<<"b.Nhap ma tran tu ban phim\n";
        cout<<"c.Xuat ma tran\n";
        cout<<"d.Sap xep ma tran tang:\n";
        cout<<"e.Sap xep ma tran giam:\n";
        cout<<"f.Sap xep dong tang dan\n";
        cout<<"g.Sap xep cot giam dan\n";
        cout<<"h.Tong gia tri nho nhat\n";
        cout<<"z.Thoat\n";
        cout<<"Moi ban chon muc :";
        cin>>chon;
        switch(chon)
        {
            case 'a': NhapMaTranNN(a,d,c); break;
            case 'b': NhapMaTran(a,d,c); break;
            case 'c': XuatMaTran(a,d,c); break;
            case 'd': SapxepMaTranTang(a,d,c); break;
            case 'e': SapxepMaTranGiam(a,d,c); break;
            case 'f': SapXepDongTangDan(a,d,c); break;
            case 'g': SapXepCotGiamDan(a,d,c); break;
            case 'h':

                {
                    cout<<"tong nho nhat
la:"<<TongMinDCChinh(a,d);

                }
                break;
            case 'z': exit(1);
        }
        cout<<"\n\n\n\n";
    }
}

```

Bài 117: (OK)

```
int SumMax(int a[][MAX], int d, int c, int &k)
{
    int s=0, s1=0, max;
    for(int j=0; j<c; j++)
        s+=a[0][j];
    max=s;
    k=0;
    for(int i=1; i<d ;i++)
    {
        for(int j=0; j<c; j++)
            s1+=a[i][j];
        if(s1>max)
        {
            max=s1;
            k=i;
        }
    }
    return k;
}
```

```
void XoaLineSumMax(int a[][MAX], int &d, int c, int &k)
{
    for(int i=SumMax(a,d,c,k); i<d-1; i++)
        for(int j=0; j<c; j++)
        {
            if(TongPTLNDong(a,d,c)==1)
                a[i][j]=a[i+1][j];
        }
        d--;
}
```

Bài 118 (OK)

```
int SumMax(int a[][MAX], int d, int c, int &k)
{
    int s=0, s1=0, max;
    for(int j=0; j<c; j++)
        s+=a[0][j];
    max=s;
    k=0;
    for(int i=1; i<d ;i++)
    {
        for(int j=0; j<c; j++)
            s1+=a[i][j];
        if(s1>max)
```

```

        {
            max=s1;
            k=i;
        }
    }
    return k;
}
int SumMin(int a[][MAX], int d, int c, int &k)
{
    int s=0, s1=0, min;
    for(int j=0; j<c; j++)
        s+=a[0][j];
    min=s;
    k=0;
    for(int i=1; i<d ;i++)
    {
        for(int j=0; j<c; j++)
            s1+=a[i][j];
        if(s1<min)
        {
            min=s1;
            k=i;
        }
    }
    return k;
}

void HoanVi(int &a, int &b)
{
    int tam=a;
    a=b;
    b=tam;
}
/*void DoiCho(int A[][MAX], int d, int c) (Việt viết)
{
    for(int j=0; j<c; j++)
        HoanVi(a[LineMax(A,n,m)][j],A[LineMin(A,n,m)][j]);
}*/
void DoiCho(int a[][MAX], int d, int c, int &k)//(Giang viết)
{
    for(int j=0; j<c; j++)
        HoanVi(a[SumMax(a,d,c,k)][j],a[SumMin(a,d,c,k)][j]);
}

```

Bai 119
#define MAX 10
void InpuMatrixRandom(int A[][MAX], int &n, int &m)
{

```

    cout<<"nhap n: ";
    cin>>n;
    cout<<"\nNhap m: ";
    cin>>m;
    srand((unsigned)time(NULL));
    for(int i=0; i<n; i++)
        for(int j=0; j<m; j++)
            A[i][j]=rand()%51;
}
void OuputMAtrixRandom(int A[][MAX], int n, int m)
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
            cout<<A[i][j]<<"\t";
        cout<<"\n";
    }
}
void ThayYBangX(int A[][MAX], int n, int m, int x, int y)
{
    for(int i=0; i<n; i++)
        for(int j=0; j<m; j++)
            if(A[i][j]==x)
                A[i][j]=y;
}
void main()
{
    int A[MAX][MAX], n, m, x, y;
    InpuMAtrixRandom(A,n,m);
    OuputMAtrixRandom(A,n,m);
    cout<<"\nNhap phan tu x: ";
    cin>>x;
    cout<<"\nNhap phan tu y: ";
    cin>>y;
    cout<<"\n";
    ThayYBangX(A,n,m,x,y);
    OuputMAtrixRandom(A,n,m);
    cout<<"\n";
}

```

1) Nhập dãy số nguyên A có n phần tử ($0 < n \leq 20$). Tách A thành 2 dãy: dãy B chứa các số lẻ, dãy C chứa các số chẵn. Xuất các dãy lên màn hình

2) Nhập dãy số nguyên A có n phần tử ($0 < n \leq 20$). Cài đặt các chương trình con để thực hiện các công việc:

a) Sắp xếp các phần tử vị trí lẻ tăng dần, các phần tử vị trí chẵn giảm dần

b) Sắp các phần tử dương về đầu dãy có thứ tự giảm dần, các phần tử âm cuối dãy có thứ tự tăng dần

c) Sắp các số nguyên tố về đầu dãy có thứ tự tăng dần, các phần tử còn lại có thứ tự

giảm dần.

3) Nhập 2 dãy số nguyên b tăng dần và c giảm dần. Trộn 2 dãy b,c thành dãy a tăng dần (Không cho phép sắp xếp trên dãy a)

4) cho ma trận số nguyên cấp $n \times m$. cài đặt các hàm thực hiện các thao tác trên ma trận.

a) nhập ma trận

b) in ma trận

c) tìm phần tử nhỏ nhất

d) tìm phần tử lớn nhất(chỉ ra vị trí)

e) tìm dòng có tổng lớn nhất(nếu nhiều dòng có tổng bằng nhau thì cho biết dòng đầu tiên)

5) nhập vào ma trận vuông số thực cấp n. kiểm tra các tính chất sau đây của ma trận

a) ma trận tam giác trên

b) ma trận tam giác dưới

c) ma trận đơn vị

d) ma trận đối xứng qua đường chính

e) đối xứng tâm

6) nhập vào ma trận số thực cấp $n \times m$. cài đặt các hàm thực hiện các thao tác trên ma trận

a) xóa dòng k (k nhập từ bàn phím)

b) xóa cột l (l nhập từ bàn phím)

c) xóa dòng và cột chứa phần tử nhỏ nhất

d) nhập số nguyên dương k ($k < n$) và dãy có n số thực, chèn dãy đó vào dòng k

e) nhập số nguyên dương K ($k < m$) và dãy có m số thực, chèn dãy đó vào cột k

f) hoán vị các dòng sao cho tổng giá trị của từng dòng giảm từ trên xuống

7) nhập ma trận ký tự, vuông cấp n. thực hiện các thao tác:

a) đổi các ký tự anphabe thành ký tự in hoa

b) cho biết dòng nào có nhiều nguyên âm nhất (nếu có 2 dòng thì in dòng phía trên)

Kĩ thuật tính toán

320 Tổng các số dương trong ma trận thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=432>

321 Tích các số lẻ trong ma trận nguyên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=433>

322 Tổng giá trị trên 1 dòng ma trận thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=434>

323 Tích giá trị dương trên 1 cột ma trận thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=435>

324 Tổng giá trị dương trên 1 dòng ma trận thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=436>

325 Tích các chẵn trên 1 cột ma trận nguyên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=437>

- 326 Trung bình cộng các số dương ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=438>
- 327 Tổng các giá trị nằm trên biên của ma trận
<http://taiachau.com/4rum/viewtopic.php?f=88&t=439>
- 328 Trung bình nhân các số dương ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=442>
- 329 Biến đổi ma trận bằng cách thay các giá trị âm bằng giá trị tuyệt đối của nó
<http://taiachau.com/4rum/viewtopic.php?f=88&t=443>
- 330 Biến đổi ma trận thực bằng cách thay các giá trị bằng giá trị nguyên gần nó nhất
<http://taiachau.com/4rum/viewtopic.php?f=88&t=444>
- 331 Tổng các giá trị trên 1 dòng ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=445>
- 332 Tổng giá trị lẻ trên 1 cột ma trận nguyên
<http://taiachau.com/4rum/viewtopic.php?f=88&t=446>
- 333 Tổng các số hoàn thiện trong ma trận số nguyên
<http://taiachau.com/4rum/viewtopic.php?f=88&t=447>

Kĩ thuật đếm

- 334 Viết hàm đếm số lượng số dương trong ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=448>
- 335 Đếm số lượng số ng tố trong ma trận nguyên
<http://taiachau.com/4rum/viewtopic.php?f=88&t=449>
- 336 Đếm tần suất xuất hiện của 1 giá trị x trong ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=465>
- 337 Đếm số chữ số trong ma trận số nguyên dương
<http://taiachau.com/4rum/viewtopic.php?f=88&t=466>
- 338 Đếm số lượng số dương trên 1 hàng ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=467>
- 339 Đếm số lượng số hoàn thiện trên 1 hàng ma trận nguyên
<http://taiachau.com/4rum/viewtopic.php?f=88&t=468>
- 340 Đếm số lượng âm trên 1 cột ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=469>
- 341 Đếm số lượng dương trên biên của ma trận thực
<http://taiachau.com/4rum/viewtopic.php?f=88&t=470>
- 342 Đếm số lượng ptử cực đại ma trận thực. 1 ptử cực đại là khi nó lớn hơn các ptử xung quanh
<http://taiachau.com/4rum/viewtopic.php?f=88&t=471>
- 343 Đếm số lượng ptử cực trị ma trận thực. 1 ptử cực trị khi nó lớn hơn hoặc nhỏ hơn ptử xung quanh

<http://taiachau.com/4rum/viewtopic.php?f=88&t=472>

344 Đếm số lượng giá trị có trong ma trận thực. Lưu ý: nếu có k ptử ($k \geq 1$) trong ma trận bằng nhau thì ta chỉ tính là 1

<http://taiachau.com/4rum/viewtopic.php?f=88&t=473>

345 Tính tổng các ptử cực trị trong ma trận thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=474>

346 Đếm số lượng giá trị "hoàng hậu" trên ma trận. 1 ptử hoàng hậu khi nó lớn nhất trên dòng, cột và 2 đường chéo

<http://taiachau.com/4rum/viewtopic.php?f=88&t=475>

347 Đếm số lượng "yên ngựa". 1 ptử yên ngựa khi nó lớn nhất trên dòng, nhỏ nhất trên cột

<http://taiachau.com/4rum/viewtopic.php?f=88&t=476>

Kĩ thuật Đăt Cờ Hiêu

348 Ktra ma trận có số dương ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=477>

349 Ktra ma trận có số hoàn thiện ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=478>

350 Ktra ma trận có số lẻ ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=479>

351 Ktra ma trận có toàn dương ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=480>

352 Ktra 1 hàng ma trận có tăng dần hay ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=481>

353 Ktra 1 cột ma trận có giảm dần ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=482>

354 Ktra các gtrị trong ma trận có giảm dần theo cột và dòng hay ko

<http://taiachau.com/4rum/viewtopic.php?f=88&t=483>

355 Liệt kê các dòng toàn âm trong ma trận các số thực

<http://taiachau.com/4rum/viewtopic.php?f=88&t=484>

356 Liệt kê chỉ số các dòng chứa toàn giá trị chẵn

<http://taiachau.com/4rum/viewtopic.php?f=88&t=485>

357 Liệt kê dòng chứa số nguyên tố

<http://taiachau.com/4rum/viewtopic.php?f=88&t=486>

358 liệt kê các dòng có chứa giá trị chẵn

<http://taiachau.com/4rum/viewtopic.php?f=88&t=487>

359 Liệt kê các dòng có chứa giá trị âm

<http://taiachau.com/4rum/viewtopic.php?f=88&t=488>

360 Liệt kê các cột chứa số chính phương

<http://taiachau.com/4rum/viewtopic.php?f=88&t=489>

361 Liệt kê các dòng thỏa: có giá trị âm, dương và 0 (ptử trung hòa)

<http://taiachau.com/4rum/viewtopic.php?f=88&t=491>

362 Liệt kê các dòng giảm dần

<http://taiachau.com/4rum/viewtopic.php?f=88&t=492>

363 Liệt kê các cột tăng dần

<http://taiachau.com/4rum/viewtopic.php?f=88&t=493>

364 Cho 2 ma trận A,B. Ktra ma trận A có là ma trận con của B ko?

<http://taiachau.com/4rum/viewtopic.php?f=88&t=494>

365 Đếm số lần xuất hiện của ma trận A trong ma trận B?

<http://taiachau.com/4rum/viewtopic.php?f=88&t=495>

Kĩ thuật đặt lính canh

366 Tìm số chẵn đầu tiên trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=496>

367 Tìm max trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=497>

368 Tìm giá trị lớn thứ 2 trong ma trận (max2)

<http://taiachau.com/4rum/viewtopic.php?f=88&t=498>

369 Tìm số dương đầu tiên trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=499>

370 Tìm giá trị âm lớn nhất trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=500>

371 Liệt kê các dòng có chứa max

<http://taiachau.com/4rum/viewtopic.php?f=88&t=625>

372 Tìm giá trị lớn nhất trên 1 dòng

<http://taiachau.com/4rum/viewtopic.php?f=88&t=626>

373 Tìm giá trị nhỏ nhất trên 1 cột

<http://taiachau.com/4rum/viewtopic.php?f=88&t=627>

374 Tìm số nguyên tố đầu tiên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=628>

375 Tìm số chẵn lớn nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=629>

376 Tìm số dương nhỏ nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=630>

377 Tìm số nguyên tố lớn nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=631>

378 Tìm 1 chữ số xuất hiện nhiều nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=632>

379 Đếm số lượng min

<http://taiachau.com/4rum/viewtopic.php?f=88&t=633>

380 Đếm số lượng chẵn nhỏ nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=634>

381 Tìm giá trị xuất hiện nhiều nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=643>

382 Tìm số chính phương lớn nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=644>

383 Tìm số hoàn thiện nhỏ nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=645>

384 Tìm các chữ số xuất hiện nhiều nhất trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=646>

385 Liệt kê các dòng có tổng lớn nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=686>

386 Liệt kê các cột có tổng nhỏ nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=687>

387 Liệt kê các dòng có nhiều số chẵn nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=688>

388 Liệt kê các dòng có nhiều số nguyên tố nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=689>

389 Liệt kê các dòng có nhiều số hoàn thiện nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=690>

390 * Liệt kê các cột có nhiều chữ số nhất

<http://taiachau.com/4rum/viewtopic.php?f=88&t=691>

391 * Tìm ma trận con có tổng lớn nhất

Kĩ thuật xử lý ma trận

392 Hoán vị hai dòng ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=694>

393 Hoán vị hai cột ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=695>

394 Dịch xuống xoay vòng các hàng trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=696>

395 Dịch lên xoay vòng các hàng trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=697>

396 Dịch trái xoay vòng các cột trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=698>

397 Dịch phải xoay vòng các cột trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=699>

398 Dịch phải xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=700>

399 Dịch trái xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=701>

400 Xóa 1 dòng trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=702>

401 Xóa 1 cột trong ma trận

<http://taiachau.com/4rum/viewtopic.php?f=88&t=734>

402 Xoay ma trận 1 góc 90 độ

<http://taiachau.com/4rum/viewtopic.php?f=88&t=735>

403 Xoay ma trận 1 góc 180 độ

<http://taiachau.com/4rum/viewtopic.php?f=88&t=736>

404 Xoay ma trận 1 góc 270 độ

<http://taiachau.com/4rum/viewtopic.php?f=88&t=737>

405 Chiếu gương ma trận theo trục dọc

<http://taiachau.com/4rum/viewtopic.php?f=88&t=738>

406 Chiếu gương ma trận theo trục ngang

<http://taiachau.com/4rum/viewtopic.php?f=88&t=739>

Kĩ thuật sắp xếp

407 Sắp xếp ptử trên 1 dòng tăng dần từ trái sang phải

<http://taiachau.com/4rum/viewtopic.php?f=88&t=740>

408 Sắp xếp ptử trên 1 dòng giảm dần từ trái sang phải

<http://taiachau.com/4rum/viewtopic.php?f=88&t=741>

409 Sắp xếp ptử trên 1 cột tăng dần từ trên xuống dưới

<http://taiachau.com/4rum/viewtopic.php?f=88&t=742>

410 Sắp xếp ptử trên 1 cột giảm dần từ trên xuống dưới

<http://taiachau.com/4rum/viewtopic.php?f=88&t=743>

411 Xuất giá trị chẵn trong ma trận theo thứ tự giảm dần

412 Xuất các số nguyên tố theo thứ tự tăng dần

413 Sắp xếp theo yêu cầu: Dòng có chỉ số chẵn tăng dần, lẻ giảm dần

<http://taiachau.com/4rum/viewtopic.php?f=88&t=804>

414 Sắp xếp theo yêu cầu: Cột có chỉ số chẵn giảm từ trên xuống, lẻ tăng từ trên xuống

<http://taiachau.com/4rum/viewtopic.php?f=88&t=805>

415 Sắp xếp ptử tăng dần theo hàng và cột: Dùng 2 phương pháp: sử dụng mảng phụ và ko dùng mảng phụ

<http://taiachau.com/4rum/viewtopic.php?f=88&t=806>

416 Sắp xếp ptử dương tăng dần theo cột và dòng. Dùng 2 phương pháp: Sử dụng

mảng phụ và ko sử dụng mảng phụ (*như bài 415 thêm điều kiện xét ptử dương*)
 417 Sắp xếp ptử chẵn giảm dần theo cột và dòng. Dùng 2 phương pháp: Sử dụng mảng phụ và ko sử dụng mảng phụ (*như bài 415 thêm điều kiện xét ptử chẵn và đổi dấu > thành <*)

418 Sắp xếp âm tăng dần, dương giảm dần, 0 giữ nguyên

<http://taiachau.com/4rum/viewtopic.php?f=88&t=807>

419 Sắp xếp chẵn tăng, lẻ giảm (*như bài 418 chỉ sửa lại điều kiện*)

420 Sắp xếp các giá trị nằm trên biên ma trận tăng dần

<http://taiachau.com/4rum/viewtopic.php?f=88&t=808>

Giả sử người dùng muốn xóa dòng 2 thì họ nhập vào 2 . Nhưng trong ma trận sẽ hiểu là xóa dòng 1.

Dùng 2 dòng lặp for để duyệt ma trận thì nếu i lớn hơn dòng 1 (dòng muốn xóa) thì giá trị của dòng 1 sẽ được thay thế bởi dòng 2 và dòng 2 được thay thế bằng dòng 3

Dịch trái 1 cột của ma trận theo cơ chế vòng tròn

vd:xét ma trận 3x5

1 3 3 4 5

5 3 7 8 9

0 2 3 7 2

```
for(int i=0;i<n;i++)
    int temp = a[i][0];
    for (int j=0;j<m-1;j++)
        a[i][j] = a[i][j+1]
    a[i][m-1]= temp
```

Cho ma trận vuông cấp n có $2n-1$ đường chéo song song với đường chéo phụ được đặt tên là 0 đến $2*n-2$. Hãy cho biết đường chéo nào có tổng lớn nhất ?

```
#include<stdio.h>
int main()
{
    int n,tong,max,A[10][10],i,j,k;
    printf("nhap vao cap cua ma tran:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
        printf("nhap vaoA[%d][%d]:",i,j);
        scanf("%d",&A[i][j]);
    }
    printf("ma tran vua nhap vao la:\n");
    for(i=0;i<n;i++,printf("\n"))
    for(j=0;j<n;j++)
    printf("%4d",A[i][j]);
    max=A[0][0];
    for(i=0;i<n;i++)
    {
        tong=0;
        for(j=0,k=i;j<=i;j++,k--)
            tong+=A[k][j];
        if(tong>max)max=tong;
    }
    for(j=0;j<n;j++)
    {
        tong=0;
        for(i=n-1,k=j;i>=j;i--,k++)
            tong+=A[i][k];
        if(tong>max)max=tong;
    }
    printf("tong lon nhat la:%d\n",max);
    return 0;
}
```

Giáo trình

Lập trình nâng cao



Chương 1 : GIỚI THIỆU LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

I. LỊCH SỬ PHÁT TRIỂN CỦA LẬP TRÌNH :

1. Lập trình tuyến tính :

- Việc lập trình cho các máy tính đầu tiên phải viết theo ngôn ngữ máy trong hệ nhị phân nên mất nhiều thời gian khi chạy và thử nghiệm chương trình để gỡ rối.
- Khả năng sử dụng lại các đoạn chương trình: không có
- Khi các khả năng của máy tính (MT) tăng: Lập trình phát triển từ đơn giản đến phức tạp hơn.
- Các tiện nghi cần thiết cho việc sử dụng lại chương trình gốc ban đầu hầu như không có trong các ngôn ngữ lập trình tuyến tính (LTTT) ban đầu. Khi cần làm công việc này người ta phải sao chép lại các chương trình gốc, dẫn đến chương trình dài ra. Nên việc bảo dưỡng, sửa chữa khó, rất mất thời gian
- Dữ liệu: Toàn cục, không có tính che dấu dữ liệu nên rất khó kiểm soát

2. Lập trình có cấu trúc :

- Phân mảnh vấn đề lớn thành các vấn đề con độc lập. Từ những vấn đề con này xây dựng thành thủ tục và hàm
- Dữ liệu truyền giữa các thủ tục thông qua đối số, ngoài ra nó có các dữ liệu riêng mà các thủ tục bên ngoài phạm vi của nó không thể thâm nhập tới được

3. Sự trừu tượng hoá chức năng :

- Trong một chương trình (CT) có cấu trúc chỉ cần biết thủ tục hay hàm đã cho làm được công việc cụ thể gì là đủ, còn làm thế nào mà công việc đó lại thực hiện được thì không quan trọng. Một khi thủ tục còn được tin cậy thì nó có thể dùng mà không cần biết là nó đã phải làm gì để hoàn thành đúng chức năng. Điều này được gọi là sự trừu tượng hoá chức năng (functional abstraction), đây là nền tảng của lập trình có cấu trúc

4. Lập trình hướng đối tượng (Object Oriented Programming):

- Lập trình hướng đối tượng (LTHĐT) là xây dựng trên nền tảng của lập trình có cấu trúc với sự trừu tượng hoá dữ liệu.
- Sự trừu tượng dữ liệu (data abstraction) tác động trên các dữ liệu cũng tương tự như sự trừu tượng hoá chức năng đã làm trên chức năng. Khi sự trừu tượng hoá dữ liệu xảy ra, các cấu trúc dữ liệu và các phần tử có thể được sử dụng mà không cần để ý tới các chi tiết cụ thể mà người ta xây dựng.
- Sự thay đổi căn bản là ở chỗ: 1 chương trình hướng đối tượng (HĐT) được thiết kế xoay quanh dữ liệu mà ta làm việc trên nó hơn là bản thân chức năng chương trình.

- LTHĐT sẽ liên kết cấu trúc dữ liệu (CTDL) với các thao tác theo cách mà chúng ta thường nghĩ về thế giới xung quanh đó là: gắn 1 hành động cụ thể với 1 loại đối tượng nào đó.

VD :

- Ô tô thì có bánh xe, di chuyển được và hướng chúng thay đổi bằng cách quay tay lái.
- Tương tự ta biết cây là 1 loại thực vật thân gỗ và có lá
- Ô tô không phải là cây, cây không phải là ô tô, vậy ta có thể kết luận rằng điều thực hiện được với ô tô thì không thể thực hiện được với cây. Thật hoang tưởng khi đi lái cây hoặc tưới nước cho ô tô để nó lớn lên

- LTHĐT cho phép sử dụng các quá trình suy nghĩ về thế giới quan vào dữ liệu

VD :

- Một mẫu tin thì có thể đọc ra, thay đổi và lưu trữ; còn một số phức thì có thể dùng trong các phép toán. Tuy vậy không thể viết một số phức vào tập tin làm một mẫu tin nhân sự được, và ngược lại không thể cộng hai mẫu tin nhân sự lại với nhau. Một chương trình LTHĐT sẽ xác định đặc điểm và hành vi cụ thể của các kiểu dữ liệu. Điều đó cho phép chúng ta xác định 1 cách chính xác chúng ta có thể có được những gì ở các kiểu dữ liệu khác nhau.
- Chúng ta cũng luôn luôn liên hệ các khái niệm mới với các khái niệm đã tồn tại và lại có khả năng suy luận dựa trên sự liên hệ giữa các sự vật . LTHĐT cũng làm việc theo cách tương tự, cho phép ta xây dựng CTDL mới dựa trên những CTDL đang có mang theo những tính năng của cấu trúc nền mà chúng dựa trên đó, trong khi vẫn thêm vào những tính năng mới – tính thừa kế (inheritance)

II. NHỮNG THUẬT NGỮ CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG :

1. Lớp (class), đối tượng (object), phương thức (method) :

- LTHĐT cho phép tổ chức dữ liệu theo 1 cách tương tự như các nhà sinh học tổ chức các loài thực vật khác nhau. Theo cách nói của LTHĐT thì mỗi 1 loài thực vật đó sẽ được gọi là 1 lớp-class
- Một lớp là 1 bảng mẫu mô tả các thông tin CTDL lẫn các công việc cụ thể của các phần tử dữ liệu
 - Mô tả
 - Chỉ ra nó làm được cái gì ? – Hành vi trên nó
- Một phần tử mà được khai báo thuộc 1 lớp gọi là 1 đối tượng-Object Các hàm được định nghĩa hợp lệ trên 1 lớp gọi là phương thức-Method và chúng là các hàm duy nhất có thể xử lý dữ liệu của các đối tượng của lớp đó
- Các CTDL dùng để mô tả 1 lớp thì gọi là các thuộc tính-Properties

VD : `class complex`
`{`

```

int real, imag // properties;
void cong (complex c); // 1
void tru (complex c); // 2
void nhan (complex c); // 3
void chia (complex c); // 4
};
// 1,2,3,4: method
complex a,b; // a,b: object
    
```

- Mỗi một đối tượng thì có riêng 1 bản sao các phần tử dữ liệu của lớp

```

a:real
  imag
b:real
  imag
    
```

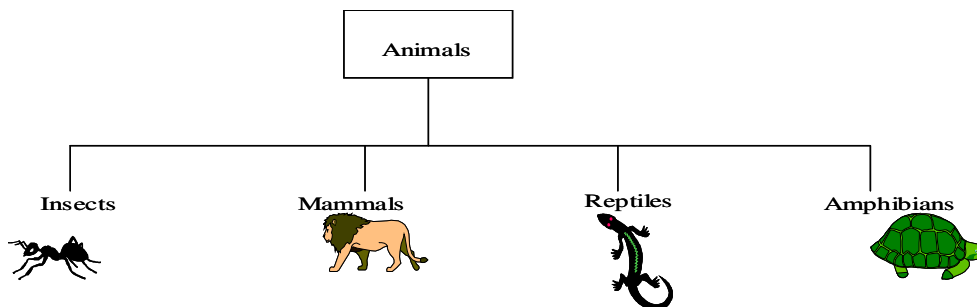
- Các phương thức định nghĩa trong 1 lớp thì có thể gọi bởi bất kỳ 1 đối tượng nào. Điều này gọi là gọi thông điệp cho đối tượng. Các thông điệp thì chỉ phụ thuộc vào đối tượng nhận, nghĩa là đối tượng nào nhận thông điệp thì mới phải làm theo thông điệp đó.

2. Lớp cơ sở (base class), lớp dẫn xuất (derived class) :

- Không giống như các kiểu dữ liệu chuẩn sẵn, các lớp có thể sử dụng các lớp khác làm các viên gạch xây dựng cho mình.
- Một lớp thì có thể dùng để xây dựng 1 lớp mới. Lớp ban đầu thì được gọi là lớp cơ sở-base class. Còn lớp mới gọi là lớp dẫn xuất-derived class.

3. Tính kế thừa (Inheritance) :

- Đó là khả năng cho phép sử dụng lại lớp đã có sẵn để xây dựng lớp mới như trên đã đề cập.
- **Vd:** Từ lớp Animals có thể xây dựng các lớp dẫn xuất (hay còn gọi là lớp con-subclass) từ nó.
- Lớp dẫn xuất thì được **kế thừa** tất cả các thuộc tính-properties và phương thức-method của lớp cơ sở, ngoài ra có thể có các thuộc tính mới và phương thức mới của riêng nó.



- Các lớp Insects, Mammals, Reptiles, Amphibians là những lớp dẫn xuất từ lớp Animals chúng đều có chung thuộc tính được thừa kế từ lớp Animals là có hai mắt, di chuyển được ...
- Nhưng ngoài ra chúng vẫn có những thuộc tính riêng, chẳng hạn Mammals thì chỉ sống ở trên cạn còn Amphibians thì vẫn có thể sống được ở cả trên cạn lẫn dưới nước

Chương 2 : GIỚI THIỆU VỀ NHỮNG ĐIỂM MỚI CỦA C++

I. C++ LÀ GÌ ?

- Là ngôn ngữ được xây dựng từ ngôn ngữ C
- Có tất cả các tính năng của C
- “Tôn trọng” cú pháp của C
- Có bổ sung và cải tiến

II. SỰ KHÁC NHAU GIỮA C VÀ C++

1. Ép kiểu :

- Trong C : (tenkieu)bien
- Trong C++: tenkieu(bien)
- **Vd** Tính công thức $S = 2/1 + 3/2 + \dots + (n+1)/n$
với n là số nguyên dương nhập từ bàn phím, có thể viết:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    printf("\n So phan tu cua day N= ");
    scanf("%d",&n);
    float s=0.0;
    for (int i=1; i<=n; ++i)
        s += float(i+1)/float(i); //Ep kieu theo C++
    printf("S=%0.2f ",s);
    getch();
}
```

2. Ghi chú :

- Trong C : /*...*/
- Trong C++: /*...*/: ghi chú nhiều dòng
// : ghi chú đến cuối một dòng

3. Khai báo :

- Trong C++ : Chỉ cần khai báo trước khi sử dụng (mọi nơi)

4. Hằng

- Trong C : #define N 100; (Không biết N là kiểu gì)
- Trong C++: const int N=100; (Cho biết kiểu của N, đây gọi là một hằng có kiểu)

5. Toán tử phạm vi ::

- Trong C++ :: (bốn dấu chấm)

Vd: int a=2;
void main()
{

```
int a=3;
printf(" a ngoai:%d", ::a); (a=2)
printf(" a trong:%d", a); (a=3)
}
```

6. Vào ra trong C++

a. In dữ liệu ra màn hình :

Hàm printf()

Toán tử xuất: cout << bt <<...<< bt;
//đưa giá trị các bt ra màn hình

b. Nhập dữ liệu từ bàn phím :

Hàm scanf()

Toán tử nhập: cin >>biến >>...>>biến

Vd : Nhập 1 dãy không quá n ký tự và chứa vào mảng h (kiểu char) :

```
cin.get(h,n);
```

* **Chú ý 1:** 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

Khắc phục: dùng cin.ignore(1);

Mục đích : để bỏ qua 1 ký tự chuyển dòng

* **Chú ý 2:** Để sử dụng các phương thức nói trên cần khai báo tệp tiêu đề:

```
#include <iostream.h>
```

7. Định dạng khi in ra màn hình :

■ Quy định độ rộng tối thiểu là w vị trí cho giá trị (nguyên,thực,chuỗi) được in trong toán tử xuất ta dùng hàm:

```
setw(w)
```

- Hàm này cần đặt trong toán tử xuất
- Chỉ có hiệu lực cho 1 giá trị được in gần nhất
- Các giá trị in tiếp theo có độ rộng tối thiểu mặc định là 0

Vd: cout << setw(3) << "AB" << "CD"

Sẽ in ra 5 ký tự gồm một dấu cách và 4 chữ cái A,B,C,D.

- Hàm trên nằm trong thư viện
#include <iomanip.h>

8. Kiểu liệt kê :

Kiểu liệt kê (enum) :

■ Tên viết sau từ khoá enum được xem là kiểu liệt kê và có thể dùng để khai báo,

Vd:

```
enum MAU {xanh, do, tim, vang}; //Đn kiểu MAU
```

```
MAU m, dsm[10]; //Khai báo các biến, mảng kiểu MAU
```

■ Các giá trị kiểu liệt kê là các số nguyên. Do đó có thể thực hiện các phép tính trên các giá trị này, có thể in, có thể gán giá trị này cho biến nguyên.

Vd: MAU m1, m2;
int n1, n2;

```
m1= tim;
m2= vang;
n1 = m1; // n1= 2
n2 = m1+ m2; // n2= 5
printf ("\n %d", m2); //in ra số 3
```

■ Không thể gán trực tiếp 1 giá trị nguyên cho 1 biến enum mà phải dùng phép ép kiểu,

Vd: m1= 2; // lỗi
 m1= MAU(2); // đúng

9. Cấp phát và giải phóng bộ nhớ :

■ Trong C:

```
p=(int) malloc(sizeof(int)); cấp phát 1 vùng nhớ
p=(int) calloc(n, sizeof(int)); cấp phát n vùng nhớ
free(p); giải phóng bộ nhớ
```

■ Trong C++:

```
p= new int; cấp phát 1 vùng nhớ
p= new int[n]; cấp phát n vùng nhớ
delete p; Giải phóng
```

10. Đối kiểu tham chiếu :

■ Trong C để nhận kết quả của hàm cần dùng đối con trỏ. Làm cho việc xây dựng cũng như sử dụng hàm khá phiền phức.

■ Trong C++ đưa vào đối kiểu tham chiếu (giống Pascal) dùng để chứa kết quả của hàm. Việc tạo lập và sử dụng đơn giản hơn.

Vd :

Trong C :

```
void swapint(int *a, int *b)
{
    int temp= *a;
    *a      = *b;
    *b      = temp;
}
```

Gọi hàm: swapint(&x, &y)

Trong C++:

```
void swapint (int &a, int &b)
{
    int temp  = a;
    a         = b;
    b         = temp;
}
```

Gọi hàm: swapint (x, y) //không cần toán tử &

■ Vậy một biến tham chiếu thì được xác định bằng toán tử & dùng trước tên biến giống như toán tử * dùng trước con trỏ

- Điểm khác nhau giữa 1 con trỏ chỉ đến 1 biến và 1 biến tham chiếu đến nó là:

- Đối với con trỏ thì phải dùng phép toán lấy địa chỉ
- Đối với biến tham chiếu thì không cần.

11. Đối có giá trị mặc định :

- Trong nhiều trường hợp người dùng viết 1 lời gọi hàm nhưng còn chưa biết nên chọn giá trị nào cho các đối. Để khắc phục khó khăn này, C++ đưa ra giải pháp đối có giá trị mặc định. Khi xây dựng hàm, ta gán giá trị mặc định cho một số đối. Người dùng nếu không cung cấp giá trị cho các đối này, thì hàm sẽ dùng giá trị mặc định.

Vd1:

```
void delay ( int loops = 1000)
{
    "Lệnh..."
};
Gọi: delay( ); // loops = 1000;
     delay(2000); // loops = 2000;
```

Vd2:

```
void test ( int a=100, int b=10; int c=1)
{
    "Lệnh..."
};
test( ); // lấy các giá trị mặc định
test(1,1,10); // a=1, b=1, c=10
test(10); // a=10, b,c mặc định
test( ,10, ); // b=10, a,c mặc định
test(20,10); // a=20, b=10, c mặc định
```

12. Hàm Inline :

- Khi chương trình biên dịch nhìn thấy một lời gọi hàm, nó thường nhảy đến hàm đó. Tại vị trí cuối của hàm nó sẽ quay về lại lệnh theo sau lời gọi hàm.

- Có thể lưu trong không gian bộ nhớ nhưng lại tốn thêm thời gian.

- Đối với những hàm ngắn khoảng một đến hai dòng lệnh nên sử dụng inline

- Một hàm inline được viết như một hàm bình thường trong file nguồn nhưng biên dịch vào trong mã inline thay vì vào trong một hàm.

```
inline float converter(float dollars);
```

- Hàm inline được trình bày như một thực thể riêng biệt trong file nguồn nhưng khi chương trình được biên dịch, thân của hàm thật sự được chèn vào trong chương trình bất cứ ở đâu một lời gọi hàm xảy ra.

Cách viết hàm inline :

Cách 1:

```
inline[kiểu trả về] <tên hàm> (<đối số>);
[kiểu trả về ] <tên hàm> (<đối số>)
```



```
{
    // các câu lệnh
    [return<biểu thức hoặc hằng>]
}
```

Cách 2:

```
inline[kiểu trả về] <tên hàm> (<đôi số>)
{
    // các câu lệnh
    [return<biểu thức hoặc hằng>]
}
```

*** Hàm Inline : những điểm cần lưu ý**

- Từ khoá inline phải xuất hiện trước các lời gọi hàm thì trình biên dịch mới biết cần xử lý hàm theo kiểu inline.

Vd:

```
#include<stdio.h>
int cong(int a, int b);
void int main()
{
    int tong=cong(1,2);
    print("%i\n", tong);
}
inline int cong(int a, int b) // tính tổng hai số
{
    return a+b;
}
```

Chương trình biên dịch sẽ thông báo lỗi vì ta đã khai báo và sử dụng `cong()` như một hàm thực nhưng lại định nghĩa inline.

- Bởi vì mã của hàm inline phải được biên dịch trước trước khi nó được chen vào chương trình, cho nên ta luôn phải định nghĩa hàm inline trước khi tham khảo tới chúng.
- Từ khoá inline thật sự chỉ là một gợi ý cho chương trình biên dịch chứ không phải là một lệnh bắt buộc. Thỉnh thoảng chương trình biên dịch (CTBD) sẽ bỏ qua inline và biên dịch hàm như một hàm bình thường.
 - Chẳng hạn nếu có quá nhiều hàm inline, CTBD sẽ không chấp nhận inline nữa vì thiếu bộ nhớ, hoặc khi các hàm inline quá dài.

13. Hàm chồng (Function Overloading) :

- Sử dụng để định nghĩa một tập hợp những hàm được cho cùng tên và cơ bản cùng thực hiện những thao tác như nhau, nhưng sử dụng danh sách đối số khác nhau.

```
void display(); // hàm display
void display(const char*);
void display(int one, int two);
void display(float number);
```

- Chương trình biên dịch sử dụng ngữ cảnh để xác định định nghĩa nào của một hàm được chồng được gọi: tùy thuộc vào số và kiểu của những đối số được cung cấp trong lời gọi.
- Chỉ những hàm mà cơ bản thực hiện cùng một tác vụ, trên những tập hợp dữ liệu khác nhau mới được chồng.

Ưu điểm:

- Rút ra việc sử dụng cùng một tác vụ cho những hàm có tên khác nhau.
- Giúp dễ hiểu và gỡ rối mã dễ dàng.
- Duy trì mã dễ dàng hơn.

Overloading với những kiểu dữ liệu khác nhau

- Chương trình biên dịch có thể phân biệt những hàm chồng có cùng số lượng đối số nhưng khác kiểu.

```
int square(int);
float square(float);
double square(double);
```

Overloading với số lượng đối số khác nhau

```
int square(int); //khai báo hàm
int square(int, int, int);
int asq = square(a) //gọi hàm
int bsq = square(x, y, z)
```

- Gọi hàm phải tương thích trên đối số, ngược lại, nếu không có hàm nào có sự tương thích đó thì chương trình biên dịch sẽ thông báo lỗi.
- Chú ý rằng chương trình biên dịch giải quyết overloading tùy thuộc vào trật tự mà trong đó hàm được khai báo.
- Kiểu trả về của hàm không cần xem xét.

Hàm chồng: Luật phạm vi

- Nguyên lý của overloading được chấp nhận chỉ trong cùng phạm vi với lời khai báo hàm

```
class first{
public:
void display();
};
class second{
public:
void display();
};
void main()
{
first obj1;
second obj2;
obj1.display();
//không có hàm chồng xảy ra
obj2.display();
}
```

- Phạm vi được giới hạn nghiêm ngặt đối với những lớp mà trong đó chúng được khai báo.

14. Chồng toán tử (Operator Overloading) :

- Việc dùng các phép toán thay cho 1 lời gọi hàm rõ ràng làm chương trình ngắn gọn và sáng sủa hơn nhiều.

Vd: Để thực hiện phép cộng 2 ma trận ta viết:

`C = A + B`; rất gần với toán học

- C++ cho phép dùng các phép toán chuẩn để đặt tên cho các hàm (gọi là định nghĩa chồng toán tử).

- Đó là khả năng kết hợp một toán tử-operator đã tồn tại với một hàm thành phần và sử dụng nó với những đối tượng của lớp chứa nó như là những toán hạng-operands.

- Những biểu thức với những operators như +, -, >, +=, ==, ...có thể chỉ được áp dụng trên kiểu dữ liệu chuẩn như int và float

- Theo trên ta có chồng toán tử cho phép thực hiện những câu lệnh như :

```
if (obj1>obj2){ . . . }
```

với obj1 và obj2 là những đối tượng của một lớp.

- Thao tác của việc so sánh những đối tượng có thể được định nghĩa trong một hàm thành phần và kết hợp với toán tử so sánh-comparison operator.

- Chương trình biên dịch có thể phân biệt giữa những toán tử chồng bằng việc kiểm tra kiểu dữ liệu của những toán tử của nó.

- Chồng toán tử là một hình thức của đa hình.

Ưu điểm

- Giúp cho chương trình dễ dàng để đọc và gỡ rối.
- Dễ dàng để hiểu rằng hai đối tượng được cộng lại với nhau và kết quả được gán cho đối tượng thứ ba, nếu ta sử dụng cú pháp:

```
obj3 = obj1 + obj2;
```

thay vì,

```
obj3.addobjects(obj1,obj2);
```

Định nghĩa các toán tử chồng

- Để định nghĩa toán tử chồng người ta dùng cách viết như sau:

```
kiểu trả về operator <op> (<danhsách các đối>)
```

```
{
    //các dòng lệnh của thân hàm toán tử
    [ return <biểu thức hoặc hằng> ]
}
```

<op> tên các phép toán

<ds các đối> chính là các biến có kiểu giá trị tương ứng với kiểu dữ liệu mới, nó đại diện cho các toán hạng của toán tử định nghĩa

Vd:

```
PS operator +(PS ps1, PS ps2); // đn chồng phép + hai phân số
```

```
PS operator -(PS ps1, PS ps2); // đn chồng phép - hai phân số
```

Cả 2 phép toán trên đều là phép toán 2 toán hạng

Ps1 đóng vai trò là toán hạng 1, ps2 là toán hạng 2

- Với các phép toán << và >> để truy xuất 1 kiểu dữ liệu không chuẩn, định nghĩa chồng được viết theo kiểu sau:

```
ostream& operator <<(ostream& os,<tên lớp> <tên đối tượng>);
```

```
istream& operator >>(istream& is,<tên lớp> <tên đối tượng>);
```

<tên lớp> chính là kiểu dữ liệu mới được định nghĩa, nó có thể là tên lớp (class) sẽ được nói đến trong chương sau hoặc tên 1 cấu trúc (struct). Theo cách thức này đối tượng sẽ được đưa đến luồng ra (lớp ostream) hoặc đưa đến luồng vào (lớp istream)

Khi dùng hàm toán tử có 2 cách gọi :

C1: Dùng như 1 hàm thông thường, tức là gọi nó thông qua tên hàm.

Vd:

```
PS ps1, ps2, ps3, ps4, tg1, tg2;
ps1=PS_set(3,4); ps2=PS_set(1,2);
ps3=PS_set(5,7); ps4=PS_set(1,7);
tg1 = operator+(ps3,ps4); // tính tổng 2 phân số
tg2 = operator-(ps1,ps2); // tính hiệu 2 phân số
```

C2: dùng như phép toán số học chuẩn

Vd:

```
PS ps1, ps2, ps3, ps4, tg1, tg2;
ps1= PS_set(3,4); ps2 = PS_set(1,2);
ps3= PS_set(5,7); ps4 = PS_set(1,7);
tg1 = ps3+ps4;
tg2 = ps1-ps2;
```

Nhận xét:

Ta thấy dùng như cách 2 sẽ tự nhiên và đơn giản hơn cách 1 bởi vì có thể kết hợp nhiều phép toán để viết các biểu thức phức tạp.

BÀI TẬP CHƯƠNG 2

- Bt1:** Cho một dãy điểm trên mặt phẳng, biết tọa độ (x,y) của từng điểm. Viết chương trình tìm một cặp điểm xa nhau nhất.
- Bt2:** Cho biết họ tên, điểm toán, lý, hoá của N thí sinh. In ra danh sách thí sinh trúng tuyển khi biết điểm chuẩn. Yêu cầu in theo trật tự tăng dần của tổng điểm.
- Bt3:** Cho một ma trận số thực cấp $m \times n$; viết chương trình tìm phần tử lớn nhất và phần tử nhỏ nhất trên từng hàng của dãy số.
- Bt4:** Viết chương trình hiện xâu ký tự bất kỳ trên màn hình đồ hoạ, tại vị trí (x,y) và có màu m
- Bt5:** Cho n hình chữ nhật, biết độ dài hai cạnh của từng hình. Viết chương trình tính và in ra chu vi và diện tích của các hình.
- Bt6:** Cho biết họ tên, tuổi, mức lương và thu nhập của N công nhân. Viết chương trình in ra danh sách công nhân bao gồm: số thứ tự, họ tên, tuổi, mức lương và thu nhập. Yêu cầu in theo trật tự tăng dần của tuổi.
- Bt7:** Tìm số lớn nhất của một dãy số nguyên, dãy số thực.

Chương 3: CÁC LỚP (Classes)

I. ĐỊNH NGHĨA LỚP (CLASS) :

1. Khái niệm về class :

- Lớp (class) là khái niệm trung tâm của LTHĐT
- Lớp là sự mở rộng khái niệm cấu trúc (struct) của C và bản ghi (record) của Pascal
- Ngoài các thành phần dữ liệu như cấu trúc, lớp còn chứa các thành phần hàm, còn gọi là phương thức (method) hay hàm thành viên (member function).
- Cũng giống như cấu trúc, lớp có thể xem như 1 kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và được dùng để khai báo các biến, mảng đối tượng
- Trong phần này sẽ xây dựng các định nghĩa về lớp, phương thức, phạm vi truy nhập, sử dụng các thành phần của lớp, lời gọi các phương thức

2. Định nghĩa lớp (class) :

- Lớp (class) là nhóm của những đối tượng (objects) có cùng chung thuộc tính (properties) và có những mối quan hệ chung
- Thuật ngữ class là được viết tắt từ nhóm từ “class of objects”

Vd: class of persons

- Lớp được định nghĩa như sau:

```
class tenlop
{
    // Khai báo các thành phần dữ liệu; (properties)
    // Khai báo các phương thức (methods)
};
```

- Minh họa đối tượng và lớp :

3. Đối tượng (Object) :

- Thể hiện một thực thể trong thế giới thực
- Một khái niệm với những định nghĩa bên ngoài mà liên quan đến vấn đề chúng ta đang đối mặt.
- Đối tượng phục vụ hai mục đích:

- Chúng giúp dễ hiểu được thế giới thực
- Chúng cung cấp một nền tảng thiết thực cho sự ứng dụng của máy tính
- Mỗi đối tượng có những properties hoặc những nét đặc trưng riêng của nó mô tả những gì nó có hoặc nó làm.
 - Xe cộ trong một ứng dụng giám sát giao thông
 - mouse và keyboard
 - Một mẫu tin nhân sự
 - Bảng điểm liên quan đến kỳ thi
 - Những số phức
- Minh họa : Những đối tượng khác nhau

4. Thuộc tính (Property) :

- Một đặc trưng được đòi hỏi của một đối tượng hoặc một thực thể khi được trình bày trong một lớp được gọi là một thuộc tính.
- Thuộc tính của 1 lớp có thể là các biến, mảng, con trỏ kiểu chuẩn (int, float, long...) hoặc kiểu ngoài chuẩn đã được định nghĩa trước (struct, union, class,...)
- Thuộc tính của 1 lớp không thể là kiểu của chính lớp đó nhưng được phép là kiểu con trỏ của lớp này:

Vd:

```
class A
{
    A    x; // Không được vì x có kiểu lớp A
    A    *px; // Được vì px là con trỏ kiểu lớp A
};
```

5. Phương thức (Method) :

- Một hành động được đòi hỏi của một đối tượng hay một thực thể khi được trình bày trong một lớp được gọi là một phương thức hay hàm thành phần.
 - Trong một lớp polygon: "draw", "erase" và "move" là những ví dụ của những phương thức—là một phần của lớp.
- Đối tượng là một “hộp đen” nhận và gửi thông điệp.
- Hộp đen thực chất chứa mã (dãy những câu lệnh của máy tính) và dữ liệu (thông tin mà những câu lệnh thực hiện trên nó).

- Thông tin được truyền đi và nhận lại từ mỗi khu vực hoặc bởi những bản ghi nhớ giữa các khu vực với nhau hoặc bởi những câu lệnh bằng lời là những thông điệp giữa những đối tượng.
- Những thông điệp này có thể được biên dịch bởi những lời gọi hàm trong chương trình

- Phương thức có thể được xây dựng từ bên ngoài hoặc bên trong định nghĩa lớp. Thông thường phương thức ngắn viết trong, phương thức dài viết ngoài.
- Giá trị trả về của phương thức có thể có kiểu bất kỳ (chuẩn hoặc ngoài chuẩn).
- Trong thân phương thức của 1 lớp có thể sử dụng:
 - Các thuộc tính của lớp
 - Các phương thức của lớp
 - Các hàm xây dựng trong chương trình

Vd: `class complex`

```

{
    private:
        float real, imag;
    public:
        void set (float i=0.0, float j=0.0
        complex &operator + (complex x);
        complex &operator - (complex x);
        void in ( )
        {
            printf ("%f \t%f", real, imag);
        }
};

```

// Định nghĩa các method

```

    complex complex:: &operator + (complex x)
// complex thứ 2 để chỉ rõ phép toán này thuộc lớp complex trong trường hợp có nhiều lớp.

```

```

{
    real = real+x.real;
    imag = imag+x.imag;
}

```

```

void main( )

```



```

{
    complex a, b, c, d;
    a.set(1,2);
    b.set(3,4);
    c = a + b;
    d = a - b;
    c.in ( );
    d.in ( );
};

```

*** Chú ý:**

■ Định nghĩa phương thức trong lớp:

```

Kiểu tenpt (ds đối)
{
    // Lệnh
}

```

■ Định nghĩa phương thức ngoài lớp

```

Kiểu tenlop:: tenpt (ds đối)
{
    // Lệnh
}

```

■ Nếu phương thức không có đối ta vẫn phải giữ nguyên dấu ().

Vd: Xây dựng lớp số nguyên:

```

class dayso
{
    private:
        int *p, n;
    public:
        void nhap( );
        int max( );
        int min( );
        void sapxep( );
        void in( );
};

void dayso::nhap( )
{
    printf ("Nhap so pt:"); scanf ("%d",&n);
    p= new int[n];
    printf ("Nhap cac phan tu");
    For (int *pa= p; pa<= p+n; pa++)
        scanf ("%d", pa);
}

int dayso::max
{
    int m= *p;
    For (int *pa= p; pa<p+n; pa++)
        if (*pa>m) m= *pa;
    return m;
}

void dayso::sapxep( )

```

```

    {
        int *px,*py,tg;
        for (px= p; px<p+n-1; px++)
            for (py= px; py<p+n; py++)
                if (*py>*px)
                    {
                        tg= *py;
                        *py= *px;
                        *px= tg;
                    }
    }
void dayso::in( )
{
    int *px;
    for (px= p; px< p+n; px++)
        printf ("%d \t", *px);
}
void main( )
{
    dayso a;
    clrscr( );
    a.nhap( );
    printf ("so lon nhat: %d",a.max( ));
    printf ("day ban dau");
    a.in ( );
    a.sapxep( );
    printf ("day sau khi sap xep:");
    a.in ( );
    getch ( );
}

```

Vd: Nhập danh sách học sinh, nhập điểm chuẩn, và in ra số lượng học sinh đậu

Dữ liệu hs: + Họ tên
 + Tuổi
 + Điểm

Phương thức: + Nhập
 + getD (trả lại điểm)
 + In

```

class hocsinh
{
    private:
        char ht[25];
        int diem;
        int tuoi;
    public:
        void nhap( );
        int getd
            {
                return diem;
            }
}

```

```

        void in( );
    };

void hocsinh::nhap( )
{
    int t;
    fflush(stdin); puts("Ho ten:");
    gets(ht);
    printf("tuoi:"); scanf("%d",&t);
    tuoi= t;
    printf("diem:"); scanf("%d",&t);
    diem= t;
}
void hocsinh::in( )
{
    printf("\n%s\t%d\t%d",ht,tuoi, diem);
}

void main( )
{
    hocsinh ds[100];
    int i,j,n;
    printf("\n so hoc sinh");
    scanf("%d", &n);
    printf("Nhap hoc sinh");
    for(i=0; i<n;i++)
        ds[i].nhap( );
    int dc;
    printf(" Diem chuan:");
    scanf("%d", &dc);
    puts("ds hoc sinh trung tuyen");
    for(i=0;i<n;i++)
        if(ds[i].getd( )>dc)
            ds[i].in( );
    getch( );
}

```

6. Con trỏ this và đối tượng không tường minh (object implicit) :

- Từ khoá this cho địa chỉ của đối tượng mà được sử dụng để gọi phương thức.
- Bất cứ khi nào một phương thức được gọi, chương trình biên dịch sẽ gán địa chỉ của đối tượng mà gọi phương thức đó cho con trỏ this.
- Con trỏ this có thể được sử dụng giống như bất cứ một con trỏ nào khác đến một đối tượng.
- Có thể được sử dụng để truy xuất những thành phần của đối tượng nó chỉ đến bằng cách sử dụng dấu mũi tên.

```

    this->age = 5;
    this->getdata();

```

```

class Person

```

```

{
    private:
        int age;
    public:
        void display();
};
void Person :: display()
{
    age = 25;
    cout<<age;
}
void main()
{ Person Jack;
  Jack.display();
}

```

- Ta nhận thấy trong `display()` thuộc tính thành phần `age` được sử dụng một cách đơn độc không gắn với một đối tượng nào.
- Điều này có vẻ như mâu thuẫn với các quy tắc lấy thành phần.
- Thực tế thì theo trên ta đã nói, khi gọi một phương thức, C++ tự động phát sinh con trỏ `this` trong phương thức.
- Các thuộc tính trong phương thức nếu không gắn liền với một đối tượng cụ thể nào thì được hiểu là thuộc một đối tượng do con trỏ `this` trỏ tới.
- Đối tượng được gửi đầu tiên tới phương thức do con trỏ `this` chỉ đến được gọi là đối tượng không tường minh hay đối tượng ẩn (object implicit).

▪ Vậy `display()` có thể viết lại như sau:

```

class Person
{
    private:
        int age;
    public:
        void display();
};
void Person :: display()
{ this->age = 25;      // age=25
  cout<<this->age;    // cout<<age
}
void main()
{
    Person Jack;
    Jack.display();
}

```

*** Tham số ứng với đối con trỏ `this` :**

Xét lời gọi tới `display()`

```

Person Jack;
Jack.display();

```

- Trong trường hợp này tham số truyền cho con trỏ `this` chính là địa chỉ của `Jack`:

`this = &Jack`

Do đó:

`this->age` chính là `Jack.age`

Như vậy câu lệnh

`Jack.display()`; sẽ xuất ra dữ liệu của thuộc tính của đối tượng `Jack`

Vậy ta có thể rút ra kết luận:

- Tham số truyền cho đối con trở `this` chính là địa chỉ của đối tượng đi kèm với phương thức trong lời gọi phương thức.

7. Toán tử phạm vi (Scope resolution operator) :

- Hàm có thể được định nghĩa bên ngoài lớp sử dụng một toán tử phạm vi `::` (hai dấu hai chấm).

- Cú pháp chung:

`Kieu_tra_ve tenlop ::ham_thanh_phan(danh_sach_doi_so)`

- Kiểu của những đối số của hàm thành phần phải phù hợp chính xác với kiểu được khai báo trong lớp chỉ định.

- Quan trọng cho việc định nghĩa những hàm thành phần bên ngoài sự khai báo lớp.

- Toán tử ở bên trái của `::` phải là tên của lớp.

- Có sự tồn tại của toán tử phạm vi mới nhận dạng được hàm như là một thành viên của một lớp cụ thể.

- Cũng được sử dụng để đề cập đến tên của một biến toàn cục (global variable) trong trường hợp có một biến toàn cục và một biến cục bộ cùng tên.

- Cú pháp thường sử dụng: `::biến_toàn_cục`

- Trong việc đặt tên biến.

- Nếu hai biến có những mục đích khác nhau, tên của chúng nên đặt khác nhau.

Vd: `int a=2;`

```
void main( )
{
    int a=3;
    printf(" a ngoai:%d", ::a); (a=2)
    printf(" a trong:%d", a); (a=3)
}
```

- `::` còn được dùng trong các định nghĩa hàm ngoài của các phương thức của các lớp

- Nó còn được dùng để phân biệt các thành phần trùng tên nhau của các lớp cơ sở khác nhau.

8. Phạm vi lớp (class scope) :

- Phạm vi:

- Là phần chương trình mà trong đó ta có thể truy xuất đến một tên biến nào đó

- Hai kiểu phạm vi thường dùng trong C:

- Global: biến được khai báo ở bên ngoài một hàm, có phạm vi toàn cục, có thể được truy xuất từ bất kỳ chỗ nào trong chương trình

- Local: biến được khai báo bên trong một hàm hoặc một khối lệnh, có thể được truy xuất bên trong hàm hoặc khối lệnh đó, biến cục bộ có phạm vi địa phương
- Phạm vi lớp:
 - Để kiểm soát cách truy xuất tới các thành phần dữ liệu cũng như method của class.
 - Tất cả các thành phần của một class được coi là thuộc về phạm vi của class đó.
 - ☞ Một thành phần bất kỳ nào đó của một class có thể tham khảo đến bất kỳ một thành phần nào khác của cùng class đó
 - ☞ Đây chính là một phần của khái niệm bao bọc (encapsulation).

9. Encapsulation:

- Là việc truy xuất đến một đối tượng chỉ thông qua những thông điệp của nó, trong khi vẫn giữ cho những chi tiết **private** được gọi dưới dạng thông tin ẩn.
- Một lớp có nhiều thuộc tính và phương thức. Người sử dụng không cần thiết phải truy xuất đến tất cả chúng.
- Tất cả sự liên lạc đến đối tượng được thực hiện thông qua thông điệp.
- Nếu một lớp dẫn xuất có thể truy xuất trực tiếp đến các thành phần **private** của lớp cơ sở thì những lớp dẫn xuất từ nó sau đó cũng có thể truy xuất đến những thành phần này. Khả năng này sẽ được lan truyền trên khắp các lớp dẫn xuất trên cây thừa kế có thứ bậc. Điều này vi phạm tính bao bọc dữ liệu của lớp cơ sở về khả năng ẩn thông tin.
- Như vậy dữ liệu sẽ được tổ chức sao cho các đối tượng ở lớp khác không thể truy nhập được mà chỉ cho phép các hàm trong cùng lớp hoặc trong những lớp có quan hệ kế thừa với nhau được quyền truy nhập.
- Nguyên tắc bao bọc dữ liệu để ngăn cấm sự truy nhập trực tiếp trong lập trình còn được gọi là sự che dấu thông tin.

Vd:

```
class complex
{
    private:
        double real;
        double imag;
    public:
        ...
};

complex cong(complex c1, complex c2) // hàm tự do
{
    complex tg ;
    tg.real=c1.real+c2.real;
    tg.imag=c1.imag+c2.imag;
    return tg;
}
```

thì khi thực hiện, CTBD sẽ đưa ra thông báo lỗi sau:

```
'complex::real' is not accessible
'complex::imag' is not accessible
```

- Để sửa chữa ta phải khai báo lại hai thuộc tính `real` và `imag` có đặc tính truy xuất là `public`.
- Bởi vì hàm `cong(complex c1, complex c2)` là một hàm tự do nằm ngoài phạm vi lớp nên trong phạm vi hàm không thể truy nhập trực tiếp được đến các thuộc tính `private` của lớp `complex`.
- Khi chúng ta bao bọc đúng cách, chúng ta sẽ đạt được 2 mục đích sau:
 - Xây dựng được một bức tường không thể thâm nhập được để bảo vệ những đoạn mã tránh những hư hại ngẫu nhiên do những lỗi nhỏ mà vô tình chúng ta mắc phải.
 - Chúng ta cũng có thể cô lập những lỗi đến những phần nhỏ của mã khiến cho chúng dễ dàng hơn để tìm kiếm và sửa chữa.
- Mục đích chính của việc này là phải bao bọc cấu trúc dữ liệu và chức năng của một lớp lại, để việc truy xuất đến cấu trúc dữ liệu của các lớp từ bên ngoài lớp phải bị giới hạn hoặc trở nên không cần thiết nữa hoặc không thể thực hiện được.

10. Từ khoá xác định thuộc tính truy xuất (access specifier): **PRIVATE:** & **PUBLIC :**

- Khi khai báo các thành phần của 1 lớp (phương thức & thuộc tính) thì có thể sử dụng các từ khoá **PRIVATE:** và **PUBLIC:** để chỉ rõ phạm vi sử dụng của các thành phần
- Mặc định là **PRIVATE:**
- Thành phần được khai báo là **PRIVATE:** sử dụng trong chính nó
- Thành phần được khai báo là **PUBLIC:** sử dụng mọi nơi (cả bên trong và bên ngoài lớp), có thể được truy xuất bởi bất cứ một hàm nào bên ngoài lớp.
- Đối với 1 lớp thì thuộc tính thường được khai báo là **PRIVATE** để đảm bảo tính che dấu của dữ liệu.
- Các phương thức thường được khai báo là **PUBLIC** để có thể được sử dụng ở bất cứ mọi nơi trong chương trình.
- Dữ liệu `private` không có sẵn bên ngoài lớp đối với bất cứ chương trình nào
- Dữ liệu `private` của những lớp khác thì ẩn và không có sẵn trong phạm vi truy xuất của những hàm của lớp này.

11. Hàm bạn (Friend Function) :

Hàm :

- Một lời khai báo hàm cho:
 - Tên của hàm
 - Kiểu của giá trị trả về (nếu có) bởi hàm
 - Số lượng và kiểu của đối số được cung cấp trong lời gọi của hàm
- Một lời khai báo hàm có thể hoặc không chứa tên đối số
- Có thể gọi một hàm mà không chỉ rõ tất cả những đối số của nó.

Hàm với đối số mặc định :

- Lời khai báo hàm phải cung cấp giá trị mặc định cho những đối số mà không được chỉ định rõ.
- Bất cứ khi nào có một lời gọi đến hàm mà không xác định rõ đối số, chương trình sẽ tự động gán giá trị bởi những tham số từ sự khai báo mặc định.

```
void func(int = 1, int = 3, char = '*'); //khai báo
//nguyên mẫu
```

hoặc

```
void func(int num1, int num2 = 3, char ch = '*');
```

Đối số với giá trị mặc định (default values)

- Chỉ có giá trị cuối mới có thể được mặc định.

```
void func(int num1=2, int num2, char ch='+'); //lỗi
```

- Giá trị mặc định phải có kiểu chính xác, nếu không chương trình biên dịch sẽ phát sinh lỗi.
- Giá trị mặc định có thể được cho trong nguyên mẫu hàm hoặc trong phần đầu của định nghĩa hàm nhưng không ở trong cả hai.
- Giá trị mặc định thường được cho trong khai báo nguyên mẫu hơn là trong định nghĩa hàm .

Ta có những lời gọi sau cho hàm `func` được khai báo ở trên:


```
func(2,13, '+');
func(1);
//giá trị mặc định cho đối số thứ hai và thứ ba
func(2,25); //giá trị mặc định cho đối số thứ ba
func(); //giá trị mặc định cho tất cả ba đối
func(2,, '+'); //không hợp lệ
```

- Nếu ta bỏ sót bất cứ một đối số nào trong khoảng giữa chương trình biên dịch sẽ không biết ta đang đề cập đến cái gì và sẽ phát sinh ra lỗi.

Ưu điểm :

- Đối số mặc định sẽ có ích nếu ta muốn sử dụng những đối, mà hầu như luôn luôn không thay đổi giá trị trong một hàm.
- Cũng có ích khi sau khi một chương trình được viết, người lập trình quyết định tăng khả năng của một hàm bằng cách thêm vào một đối số.
- Những lời gọi function đã tồn tại có thể tiếp tục sử dụng số lượng đối số cũ, trong khi những lời gọi hàm mới có thể sử dụng nhiều hơn

Hàm bạn (friend function) :

- Giá trị của dữ liệu private không thể được đọc hoặc viết bởi những hàm không phải là thành viên.
- Chúng ta cần một phương tiện để cho phép một hàm truy xuất đến những phần private của một lớp mà không đòi hỏi nó là thành viên.
- Một hàm không phải là thành viên được cho phép truy xuất đến những phần private của một lớp được gọi là bạn của lớp đó.
- Một hàm được tạo ra với vai trò là bạn của một lớp bởi một lời khai báo friend trong lớp đó:

```
class person{
public:
    void getdata();
    friend void display(person abc);
};
void display(person abc) //friend function
//không có toán tử ::
    {//... một vài mã...}
```

- Từ khoá friend không được lặp lại trong định nghĩa hàm.
- Nếu cùng một hàm cần để truy xuất đến những đối tượng từ những lớp khác nhau thì việc biến nó thành bạn của những lớp khác nhau là cách hữu hiệu nhất.

```
class Teacher; //khai báo trước
class Student
{
private:
    int st_data;
public:
    void getstuddata();
    friend void display(Student abc, Teacher xyz);
};
class Teacher
{
```

```
private:
    int th_data;
public:
    void getteachdata();
    friend void display(Student abc, Teacher xyz);
};
void display(Student abc, Teacher xyz)
{ //... một vài mã ... }
```

■ Khai báo trước (forward declaration): Một class không thể được đề cập đến cho đến khi nó đã được khai báo. Vì vậy class Teacher phải được khai báo trước class Student.

■ Những đặc trưng của hàm bạn:

- Một hàm bạn không có gì đặc biệt ngoại trừ quyền truy xuất đến thành phần private của một class.
- Hàm bạn không có con trỏ this.
- Khai báo friend có thể được đặt hoặc là trong phần private hoặc trong public của một lớp.
- Định nghĩa một hàm bạn không đòi hỏi tên lớp cùng với toán tử phạm vi (::) ở trước nó.

Ưu điểm :

- Hàm bạn cung cấp một mức độ của sự tự do trong những tùy chọn thiết kế giao diện.
- Hàm thành phần và hàm bạn có những đặc quyền như nhau.
- Điểm khác nhau chính là một hàm bạn được gọi dưới dạng `func(object)`, trong khi một hàm thành phần được gọi dưới dạng `object.func()`.

Vd: C1: Dùng hàm thành phần :

```
_class complex
{
private:
    float real, imag;
public:
    complex cong(complex u2)
    {
        complex u;
        u.real = this->real + u2.real;
        u.imag = this->imag + u2.imag;
        return u;
    }
};
```

Cách dùng:

```
complex u, u1, u2;
u = u1.cong(u2);
```

C2: Dùng hàm bạn :

```
class complex
{
private:
    float real, imag;
```

```

public:
    friend complex cong(complex u1, complex u2)
    {
        complex u;
        u.real = u1.real + u2.real;
        u.imag = u1.imag + u2.imag;
        return u;
    }
};

```

Cách dùng:

```

complex u, u1, u2;
u = cong(u1, u2);

```

12. Lớp bạn (Friend classes) :

- Khai báo một hàm thành phần đơn, một vài hàm thành phần hoặc toàn bộ lớp với tư cách là bạn của lớp khác.

- Vd của một hàm đơn với tư cách là bạn

```

class beta; //khai báo trước
class alpha{
private:
    int a_data;
public:
    alpha(){a_data = 10;}
    void display(beta);
};
class beta{
private:
    int b_data;
public:
    beta(){b_data = 20;}
    friend void alpha::display(beta bb);
};
void alpha::display(beta bb)
{ cout<<"\n data of beta ="<<bb.b_data;
cout<<"\n data of alpha ="<<a_data; }
void main(){
    alpha a1;
    beta b1;
    a1.display(b1);
}

```

- Khi tất cả hoặc hầu hết những hàm của một lớp cụ thể phải truy xuất đến một lớp khác, ta có thể cho lớp đó có một đặc quyền là lớp bạn.

```

class beta;
class alpha{
private:
    int data;
public:
    friend class beta;//beta là một lớp bạn
};
class beta{

```

```
public:
    void display(alpha d) //có thể truy xuất alpha
        {cout<<d.data;}
    void get_data(alpha d)//có thể truy xuất alpha
        {int x = d.data;}
};
```

- Tất cả những hàm thành phần của lớp beta có thể truy xuất đến những thành phần dữ liệu private của alpha.
- Tuy nhiên, những hàm thành phần public của lớp alpha không thể truy xuất đến những thành phần private của lớp beta.

13. Phương thức toán tử :

- Chứa những lệnh thật sự để chõng một toán tử. Toán tử được chõng sẽ theo sau từ khoá "operator".
- Các phương thức cũng giống như các phương thức thông thường (Cách xây dựng). Tuy nhiên nó chỉ khác trong cách đặt tên, đó là:

```
kieu_tra_ve operator op(danh sach doi);
{
    < các lệnh bên trong thân phương thức>;
    [return<biểu thức>; ]
};
```

với op là dấu của phép toán được chõng

- Giống các phương thức thông thường, phương thức toán tử có đối số đầu tiên là con trỏ this
- Dấu phép toán: (+,-,*,...)
- Đối với các toán tử 1 ngôi: Ta dùng con trỏ this làm đối.
- Đối với các toán tử nhiều ngôi: Đối thứ nhất dùng con trỏ this, các đối sau phải khai báo tường minh.

Vd: + Khai báo toán tử một ngôi

```
class complex
{
    private:
        double real;
        double imag;
    public:
        void set(double r, double i=0.0)
            { real = r; imag = i; };
        complex operator-();
};
complex complex::operator-( )
{
    complex u;
    u.real = -this->real;
    u.imag = -this->imag;
    return u;
}
```

Cách dùng:

```
complex u, v;
v.set(3,-2)
```

```

    u = -v;
+ Khai báo toán tử hai ngôi:
class complex
{
    private:
        double real;
        double imag;
    public:
        void set(double r, double i=0.0)
            { real = r; imag = i; };
        complex operator+(complex x);
};
complex complex::operator+(complex x)
{
    complex u;
    u.real = this->real + x.real;
    u.imag = this->imag + x.imag;
    return u;
}

```

Cách dùng:

```

complex p, q, r;
p.set(3, -2);
q.set(1, -2);

```

r = p + q;

14. Phương thức kiểu inline

- Cũng như các hàm khác, các hàm thành phần cũng có thể viết theo kiểu inline.
- Có hai kiểu viết để tạo hàm thành phần là inline:
 - Đặt inline vào **định nghĩa** hàm thành phần nằm bên trong định nghĩa lớp.
 - Đặt inline vào **định nghĩa** hàm thành phần nằm bên ngoài định nghĩa lớp.

Vd: Xây dựng lại lớp complex ở trên theo kiểu dùng hàm thành phần

void set(double r, double i=0.0) là inline.

C1: Đặt inline trong lớp complex

```

class complex
{
    private:
        double real;
        double imag;
    public:
        inline void set(double r, double i=0.0)
            {
                real = r;
                imag = i;
            };
        complex operator+(complex x);
};

```

C2: Đặt inline ngoài lớp complex

```
class complex
{
    private:
        double real;
        double imag;
    public:
        void set(double r, double i=0.0)
        complex operator+(complex x);
};
// định nghĩa phương thức kiểu inline
inline void complex::set(double r, double i)
{
    real = r; imag = i;
};
```

II. CẤU TỬ (CONSTRUCTOR), HUỖ TỬ (DESTRUCTOR) VÀ CÁC VẤN ĐỀ LIÊN QUAN

1. Cấu tử (constructor) :

- Cấu tử là một hàm thành phần đặc biệt cho sự khởi tạo tự động của một đối tượng.
- Có cùng tên với lớp chứa nó
- Có thể khai báo và định nghĩa cấu tử trong phạm vi lớp, hoặc có thể khai báo chúng trong phạm vi lớp và định nghĩa chúng bên ngoài như bất cứ một hàm thành phần nào khác

```
class username
{
    public:
        username(); //constructor
};
username::username() { }
```

- Cấu tử cũng được gọi khi một đối tượng tạm thời hoặc cục bộ của lớp được tạo ra
- Công dụng:
 - Cấu tử là 1 phương thức của lớp nhưng khá đặc biệt dùng để tạo dựng 1 đối tượng mới.
 - Chương trình dịch sau khi cấp phát bộ nhớ cho đối tượng sẽ gọi đến cấu tử
 - Cấu tử thường dùng để khởi tạo và gán giá trị cho các thuộc tính và có thể thực hiện 1 số công việc khác nhằm chuẩn bị cho 1 đối tượng mới.

* Cách viết cấu tử :

- Cấu tử khác các phương thức thông thường ở những điểm sau:
 - Tên cấu tử trùng tên lớp.
 - Cấu tử không có giá trị trả về
 - Không khai báo kiểu cho cấu tử
 - Cấu tử phải là public
- Sự giống nhau giữa cấu tử và các phương thức thông thường:
 - Có thể xây dựng cấu tử bên trong và bên ngoài lớp.
 - Cấu tử có thể có hoặc không đối (default constructor)
 - Cấu tử có thể có đối ngầm định
 - Trong 1 lớp thì có thể có nhiều cấu tử (chúng cùng tên nhưng khác nhau ở bộ đối)

Vd:

```
class date
{
    private:
        int month, day, year;
    public:
        date() //cấu tử không đối
```

```

    {day=1; month=1; year=2003;}
    date(int x) //chỉ có day được xác định
    {day=x; month=1; year=2003;}
    date(int x, int y, int z) //day month year
    {day=x; month=y; year=z;}
};

```

Vd: Xây dựng lớp DIEM. Trong lớp DIEM có cấu tử DIEM (int, int, int) . Như vậy lớp DIEM được viết như sau:

```

class DIEM
{
    private:
        int x,y;//tung độ, hoành độ của điểm đồ hoạ
        int m; // màu của điểm
    public:
        // định nghĩa cấu tử không đổi
        // gán x=0, y=0 và m=1
        DIEM( )
        {
            x=0; y=0; m=1;
        }
        // khai báo cấu tử có đổi và đổi mặc định,
        // nó được định nghĩa bên ngoài định nghĩa lớp
        DIEM (int x1, int y1, int m1=13);
        // các phương thức khác của lớp
};
// định nghĩa cấu tử có đổi
DIEM::DIEM(int x1, int y1, int m=13)
{
    x=x1; y=y1; m=m1;
}
...

```

2. Cấu tử sao chép (copy constructor) :

Xét dòng khai báo sau:

```

int a;
int b=a;

```

dòng lệnh thứ 2: khai báo 1 biến b và gán cho nó giá trị của biến a.

Tương tự:

```

complex c1(9,3); // giả sử lớp complex có constructor 2 tham số
complex c2=c1;

```

Với hai dòng lệnh này hai đối tượng cũ c1 và mới c2 có cùng một nội dung

- Khi một đối tượng được tạo ra thì một cấu tử của lớp tương ứng sẽ được gọi.
- Cấu tử được gọi khi khai báo và khởi tạo nội dung một đối tượng mới thông qua một đối tượng khác gọi là cấu tử sao chép
- Nhiệm vụ của một cấu tử sao chép là tạo ra 1 đối tượng mới giống hệt đối tượng đang có.

- Ban đầu ta nhận thấy cấu tử sao chép giống phép gán.
- Thực chất chúng có giống nhau không?
 - Phép gán thực hiện việc sao chép nội dung từ đối tượng này sang đối tượng khác, vậy **cả hai đối tượng trong phép gán đều đã tồn tại**
 - Cấu tử sao chép đồng thời thực hiện hai nhiệm vụ:
 - Tạo đối tượng mới
 - Sao chép nội dung từ đối tượng đã có sang đối tượng mới

Vd: phép gán

```
complex c1(9,3); //giả sử lớp complex có constructor
                //hai tham số
complex c2;     //giả sử lớp complex có constructor
                //không đối
c2=c1;
```

- Hai đối tượng c1,c2 được khai báo ở dòng lệnh 1 và 2 có giá trị các thành phần:


```
        c1.real=9; c1.imag=3;
        c2.real=0; c2.imag=0;
```
- Khi thực hiện phép gán (dòng lệnh 3) các đối tượng c1,c2 đã tồn tại
- Phép gán chỉ có tác dụng làm cho các thành phần của c2 có giá trị bằng giá trị của các thành phần c1:
- Vậy khi đó:


```
        c2.real=9; c2.imag=3.
```
- Trong khi đó nếu dùng cấu tử sao chép ta có thể thay 3 dòng lệnh này bằng 2 dòng lệnh sau:


```
        complex c1(9,3);
        complex c2=c1; hoặc bằng:
        complex c1(9,3);
        complex c2(c1);
```
- **Tóm lại:** Ta dùng cấu tử sao chép trong các trường hợp:
 - Cần khởi tạo đối tượng mới có nội dung tương tự như đối tượng đã có.
 - Khi cần truyền 1 đối tượng cho hàm theo kiểu tham trị
 - Hàm cần trả về một đối tượng nhằm tạo ra một đối tượng giống hệt một đối tượng cùng lớp đã có trước đó.
- **Một lớp điển hình gồm có:**
 - Khi một lớp X có một thành phần dữ liệu kiểu con trỏ, lớp này nên có: cấu tử, hàm toán tử gán, cấu tử sao chép and hủy tử.

```
class X
{
    X(some_value);           //constructor
```

```
X(const X&); //copy constructor
X& operator=(const X&); //assignment
~X(); //destructor
};
```

3. Cấu tử sao chép mặc định (default copy constructor) :

* Cấu tử sao chép mặc định :

- Tương tự như cấu tử mặc định, cấu tử sao chép mặc định được tự động phát sinh nếu chúng ta không định nghĩa. Nó được sử dụng để copy nội dung của một đối tượng đến một đối tượng mới trong suốt quá trình xây dựng đối tượng mới đó.
- Điều này nhằm bảo đảm tính đúng đắn của chương trình trong những trường hợp cần đến cấu tử sao chép.
- Vậy khi ta khai báo các đối tượng của lớp có ít nhất 2 hàm default có thể được gọi:
 - Cấu tử mặc định (default constructor)
 - Cấu tử sao chép mặc định (default copy constructor)

* Công dụng của một cấu tử sao chép mặc định :

- Tạo đối tượng mới
- Gán giá trị của các thành phần trong đối tượng cũ cho các thành phần trong đối tượng mới.

Vd: Xây dựng lớp complex không có cấu tử sao chép. Trong chương trình kiểm tra, máy sẽ tự động dùng cấu tử sao chép mặc định để tạo ra đối tượng mới giống như đối tượng đã có trước đó.

```
#include<conio.h>
#include<iostream.h>
class complex
{
private:
double real;
double imag;
public:
// cấu tử với đối mặc định
complex(double r1=0,double i1=0)
{
real = r1;
imag= i1;
}
void in( )
{
cout<<"\n Phan thuc:"<< real<<" Phan
ao:"<<imag;
}
};
void main( ) // kiểm tra cấu tử sao chép mặc định
{
```

```

clrscr( );
complex c1(9,3);
complex c2 = c1;
c1.in( );
c2.in( );
cout<<"\n Dung cach 2 \n";
complex c3(c1);
c1.in( ); c3.in( );
getch( );
}

```

4. Cấu tử sao chép tường minh :

* Cấu tử sao chép tường minh :

■ Định nghĩa 1 cấu tử sao chép tường minh

Dạng 1:

```

tenlop (tenlop &<tên đối tượng cũ của lớp>)
{
    // các dòng lệnh của hàm nhằm tạo lập đối tượng
    mới this và gán giá trị của các thành phần dữ liệu
    của đối tượng mới bằng giá trị của các thành phần dữ
    liệu của đối tượng cũ
};

```

Dạng 2:

```

tenlop (const tenlop &<tên đối tượng cũ của lớp>)
{
    //các dòng lệnh của hàm nhằm tạo lập đối
    tượng mới this và gán giá trị của các thành phần dữ
    liệu của đối tượng mới bằng giá trị của các thành
    phần dữ liệu của đối tượng cũ
};

```

trong đó, từ khoá const nhằm ngăn cấm mọi thay đổi nội dung của tham số truyền cho hàm.

Chương trình sau sẽ bổ sung thêm cấu tử sao chép vào lớp complex.

```

#include<conio.h>
#include<iostream.h>
class complex
{
private:
    double real;
    double imag;
public:
    // cau tu voi doi mac dinh.
    complex(double r1=0, double i1=0)
    {
        real = r1;
        imag = i1;
    };
};

```

```

        complex(const complex &c)// cau tu sao chep
        {
            this->real=c.real;
            this->imag=c.imag;
            getch( );
        };
    void in( )
        { cout<<"\n Phan thuc:"<< real<<" Phan
ao:"<<imag;
        }
};
void main()//kt cau tu sao chep
{
    clrscr( );
    complex c1(9,3);
    complex c2=c1;
    c1.in( ); c2.in( );
    cout<<"\n Dung cach 2:\n";
    complex c3(c1);
    c1.in( ); c3.in( );
    getch( );
}

```

- Trong chương trình này ta đã xây dựng cấu tử sao chép theo dạng 2.
- Chương trình còn minh họa cách dùng cấu tử sao chép theo cả hai dạng.
- Từ hai chương trình xây dựng lớp complex như trên ta nhận thấy đối với trường hợp này không cần xây dựng cấu tử sao chép tường minh mà chỉ cần dùng cấu tử sao chép mặc định là đủ.
- **Vậy tại sao và khi nào cần xây dựng một cấu tử sao chép tường minh**

- Đối với các lớp không có các thành phần dữ liệu kiểu con trỏ hoặc kiểu tham chiếu thì chỉ cần dùng cấu tử sao chép mặc định là đủ.
- Khi các class có các thành phần dữ liệu kiểu con trỏ hoặc tham chiếu thì cấu tử sao chép mặc định chưa đủ đáp ứng các yêu cầu vì nó sẽ gây ra việc sử dụng chung một số vùng nhớ của đối tượng mới tạo và đối tượng cũ.(quá trình copy byte by byte sẽ copy một con trỏ từ đối tượng này đến đối tượng khác và chúng cùng chỉ đến một địa chỉ trong bộ nhớ)
- Chính điều này dẫn đến sự nhập nhằng dữ liệu giữa các đối tượng mới và đối tượng cũ, gây ra các lỗi dữ liệu không lường trước được khi xử lý.

5. Huỷ tử (Destructor)

* **Công dụng:**

- Huỷ tử là hàm thành phần của lớp, được gọi một cách tự động khi một đối tượng bị huỷ bỏ
- Có cùng tên với lớp
- Huỷ tử được gọi trong các trường hợp sau:
 - Khi thoát khỏi hàm hoặc phương thức, lúc đó cần phải giải phóng các biến, mảng cục bộ...
 - Khi thực hiện các hàm toán tử, vd: +,-,... hoặc hàm giải phóng bộ nhớ như delete, free...

*** Cách viết huỷ tử :**

- Giống như cấu tử, huỷ tử không có kiểu và không có giá trị trả về.
- Huỷ tử không có đối số
- Huỷ tử cũng là thành phần public của class.
- Huỷ tử được viết theo mẫu sau:

```
~tenlop( )
```

Vd: Huỷ tử của lớp số phức complex có thể được viết như sau:

```
class complex
{
public:
    ~complex( ) // Huỷ tử
    {
        real=0;
        imag=0;
    }
    ...
};
```

- Công dụng phổ biến nhất của huỷ tử là để huỷ việc cấp phát bộ nhớ mà đã được cấp phát cho đối tượng bởi cấu tử sử dụng toán tử new.

```
class String
{
private:
    char *str;
public:
    String(char *s) //cấu tử
    {
        int length = strlen(s);
        str = new char[length+1];
        strcpy(str, s);
    }
    ~String() //huỷ tử
    {delete[] str;}
};
```

6. Cấu tử, huỷ tử mặc định (default constructor, destructor) :

*** Cấu tử mặc định :**

- Nếu lớp không định nghĩa cấu tử thì chương trình dịch sẽ cung cấp 1 cấu tử mặc định không đối và chỉ đơn thuần gán 0 vào tất cả các byte của các biến thể hiện của các đối tượng.
- Trong thực tế ít có chương trình nào sử dụng cấu tử mặc định vì việc khởi động bằng 0 như vậy thường chưa đủ để chuẩn bị 1 đối tượng làm việc.
- Nếu lớp có ít nhất 1 cấu tử thì cấu tử mặc định sẽ không được phát sinh nữa. Khi đó mọi câu lệnh xây dựng đối tượng mới đều gọi đến cấu tử của lớp. Nếu không tìm thấy cấu tử cần gọi thì chương trình sẽ báo lỗi.
- Khi xây dựng cấu tử thì có thể dùng chúng trong khai báo để tạo 1 đối tượng đồng thời gán cho các thuộc tính của đối tượng các giá trị. Dựa vào các tham số trong khi khai báo trình biên dịch sẽ biết gọi đến cấu tử nào.

Vd: `class complex`

```

{
    private:
        float real, imag;
    public:
        void in( )
        {
            cout<<"phan thuc:"<<real<<"phan
ao:"<<imag;
        }
        complex(float i, float j)
        {
            real=i; imag=j;
        }
};

void main( )
{
    complex a(1,2); //gọi cấu tử có đối: hợp lệ
    a.in( ); getch( );
    complex c;      //không hợp lệ
    c.in( ); getch( );
}

```

Dòng lệnh

```
complex a(1,2);
```

sẽ gọi cấu tử có đối để gán các thành phần dữ liệu của đối tượng khai báo a giá trị:

```
a.real = 1 và a.imag = 2
```

Còn dòng lệnh

```
complex c
```

sẽ gọi 1 cấu tử không đối để khai báo đối tượng c, nhưng vì trong class chưa có cấu tử không đối nên máy sẽ báo lỗi.

- Ta phải xây dựng 1 cấu tử nữa (cấu tử không đối)

```

complex ( )
{
    real=imag=0;
}

```

- Trong các trường hợp để khai báo mà không khởi tạo giá trị hoặc khai báo và khởi tạo giá trị đều có thể sử dụng được thì người ta cần xây dựng ít nhất 2 cấu tử
 - Một cấu tử không đối
 - Một cấu tử có đối

***Chú ý:**

- Nếu muốn viết 1 cấu tử duy nhất thì danh sách các đối đều phải đặt giá trị mặc định.

Vd: `complex(float i=0, float j=0)`

- Như vậy chương trình trên sẽ có 2 cách khắc phục:
 - Cách 1: Dùng cấu tử có đối mặc định.
 - Cách 2: Xây dựng thêm cấu tử không đối.

Chương trình sẽ được sửa như sau:

```

// sử dụng cấu tử có đối mặc định
#include <iostream.h>
#include <conio.h>
class complex
{
private:
    float real, imag;
public:
    void in( )
    {
        cout<<"phan thuc:"<<real<<"phan ao:"<<imag;
    }
    complex(float i=1, float j=0)//cấu tử voi
        //doi mac dinh.
    {
        real=i;
        imag=j;
    }
};

void main( )
{
    clrscr( ); cout<<"dung cấu tử co doi \n";
    complex a(1,2); // goi cấu tử co doi
    a.in( );
    cout<<"\n dung cấu tử co doi mac dinh \n";
    getch( );
    complex c; // goi cấu tử co doi mac dinh
    c.in( );
    getch( ); clrscr( );
}

```

```
cout<<"\n dung câu tử co doi mac dinh \n";
c=complex(9,9); // goi câu tử co doi
c.in( ); getch( );
}
```

***Huỷ tử mặc định (default destructor) :**

- Nếu trong lớp không có định nghĩa huỷ tử thì huỷ tử mặc định (không làm gì cả) được phát sinh.
- Đối với nhiều lớp không dùng biến con trỏ và tham chiếu thì dùng huỷ tử mặc định là đủ và không cần dùng huỷ tử tường minh.

7. Toán tử gán (Assignment operator)

***Toán tử gán mặc định (default assignment operator) :**

- Toán tử gán (cho lớp) là 1 trường hợp đặc biệt so với các toán tử khác.
- Nếu trong lớp chưa có định nghĩa một phương thức toán tử gán thì trình biên dịch sẽ phát sinh 1 toán tử gán mặc định để thực hiện lệnh gán 2 đối tượng của 1 lớp

Vd: Với lớp complex ta có thể viết :

```
complex c1(3,4),c2;// khai báo hai đối tượng số
// phức c1, c2
```

```
c2 = c1;
```

- Thực chất của phép toán này là sao chép các giá trị của các thành phần dữ liệu từ đối tượng c1 sang các thành phần dữ liệu tương ứng của c2, tức là:

```
c2.real = c1.real = 3;
```

```
c2.imag = c1.imag = 4;
```

- Trong đa số các trường hợp khi lớp không có các thành phần dữ liệu động (con trỏ hay tham chiếu) thì toán tử gán mặc định là đủ.
- Trong trường hợp ngược lại thì toán tử gán mặc định là không thích hợp. Đối với các lớp này, khi thực hiện phép gán các đối tượng, việc sao chép sẽ không liên quan đến các vùng dữ liệu động (đó là sự sao chép bề mặt).
- Nói cách khác, sau khi bị gán các đối tượng gán và bị gán sẽ có chung vùng dữ liệu động chứa thông tin. Do vậy sẽ gây ra các phiền toái khi quản lý thông tin và tính “riêng tư” của dữ liệu các đối tượng đã bị vi phạm.

Vd: Có 2 đối tượng của lớp dt1,dt2 của lớp DT. Cả 2 đối tượng này đều có thành phần dữ liệu tĩnh là n và 1 con trỏ a chỉ đến vùng nhớ cấp phát động để chứa dữ liệu về hệ số của đa thức.

Nếu thực hiện phép gán hai đối tượng trên qua các dòng lệnh:

```
DT dt1, dt2; // khai báo 2 đối tượng dt1, dt2
dt1.nhap( ); // nhập dữ liệu cho đối tượng dt1(giả
              // sử dt1.n=5)
dt2=dt1;     // thực hiện gán
```

- Thực tế, phép gán trên đã được thực hiện qua các dòng lệnh:
dt2.n = dt1.n và dt2.a = dt1.a;
- Do vậy con trỏ dt2.a và dt1.a đều cùng chỉ đến 1 vùng nhớ động.
- Điều này dẫn tới việc nếu thay đổi các thông tin động trên dt1 thì cũng làm thay đổi các thông tin động trên dt2 và ngược lại.
- Đây là điều không muốn vì nó làm mất tính riêng tư của các đối tượng.
- Chính vì vậy ta cần xây dựng toán tử gán tường minh khi lớp có các thành phần dữ liệu động.

***Cách viết toán tử gán :**

- Toán tử gán cũng là 1 phương thức của lớp cho nên đối thứ nhất của toán tử là con trỏ this biểu thị đối tượng đích và dùng một đối thứ hai tường minh để biểu thị đối tượng nguồn.
- Vì trong thân của toán tử gán phải làm việc trực tiếp với đối tượng nguồn nên kiểu đối tượng minh nhất thiết phải là kiểu tham chiếu.
- Toán tử gán có thể có hoặc không có giá trị trả về.
 - Nếu không có giá trị trả về (void) thì khi viết chương trình không được phép viết câu lệnh gán liên tiếp nhiều đối tượng như:


```
dt1=dt2=dt3;
```
 - Nếu toán tử gán trả về tham chiếu (&) của đối tượng nguồn thì có thể dùng toán tử gán để gán liên tiếp nhiều đối tượng.
- Cách định nghĩa chung toán tử gán được viết theo mẫu:


```
[void] [const tenlop &] operator= (const tenlop &
đôi_tượng_nguồn)
{
    // lệnh cấp phát bộ nhớ động cho
```

```
// các thành phần động của đối
// tượng đích và các lệnh gán giá trị
// cho các thành phần
[return<biểu thức hoặc hằng>; ]
};
```

Vd: Xây dựng toán tử gán cho lớp DT theo hai cách:

```
C1: void DT::operator=(const DT &dt)
    { n=dt.n; this->a=new double[n+1];
      if (dt.a==NULL) a=NULL;
      else for(int i=0;i<=n;i++)
        a[i]=dt.a[i];
    }
```

Với toán tử gán này chỉ cho phép gán đối tượng nguồn cho một đối tượng đích. Như vậy câu lệnh gán sau là sai:

```
DT dt1,dt2,dt3; // khai báo
dt1.nhap( ); // nhập thông tin cho dt1
dt2=dt3=dt1; // lệnh gán sai
```

C2: Để có thể gán liên tiếp các đối tượng trên, toán tử gán của lớp DT phải được xây dựng lại như sau:

```
const DT &DT::operator=(const DT &dt)
{
    n=dt.n; this->a=new double[n+1];
    if (dt.a==NULL) a=NULL;
    else for(int i=0; i<=n; i++)
        a[i] = dt.a[i];
    return dt;
};
```

Với cách viết này, có thể viết 1 câu lệnh để gán đối tượng nguồn cho nhiều đối tượng đích.

Vd: Các câu lệnh sau là đúng:

```
DT dt1,dt2,dt3,dt4; // khai báo
dt1.nhap( ); // nhập thông tin cho dt1
dt2=dt3=dt4=dt1; // lệnh gán đúng
```

Vd:

- + Xây dựng lớp HOCSINH để quản lý học sinh thi.
- + Thông tin vào gồm có: tên, điểm toán, điểm lý, điểm hoá
- + Thông tin ra là tổng điểm

Xây dựng bài toán như sau:

+ Dữ liệu:

*ht là biến kiểu con trỏ kí tự để lưu trữ họ tên của thí sinh
toan, ly, tong là các biến thực để lưu trữ điểm toán, điểm lý và tổng điểm của thí sinh.

+ Phương thức:

```
HOCSINH( ); // hàm tạo không đối của lớp HOCSINH.
```

```
const HOCSINH &operator=(const HOCSINH &hs); // hàm
định nghĩa toán tử gán.
```

```

~ HOCSINH( ); // là hàm huỷ của lớp.
void nhap( ); // là phương thức để nhập thông tin cho học sinh
void in( ); // để đưa các thông tin của học sinh ra màn hình.
double getd( ); // để lấy tổng điểm của thí sinh (vì các hàm không
thể truy nhập vào các thành phần riêng của lớp).
void hoanvi(HOCSINH &hs); // là phương thức để chuyển đổi giá
trị của hai đối tượng trong lớp HOCSINH cho nhau.

```

Chương trình có nội dung:

- + Nhập danh sách hs gồm các thông tin: họ tên học sinh, điểm toán, điểm lý.
- + Sắp xếp hs theo trật tự tổng điểm giảm dần
- + In ds học sinh theo trật tự mới

```

#include <iostream.h>
#include <conio.h>
#include <string.h>
class HOCSINH
{
private:
char *ht;
double toan, ly, tong;
public:
HOCSINH( ) // hàm tạo không đổi
{
ht = new char[25];
toan = 0; ly = 0; tong = 0;
}

~ HOCSINH( )
{
delete ht;
}
const HOCSINH &operator=(const HOCSINH &hs)
{
this->tong = hs.tong;
this->toan=hs.toan;
this->ly = hs.ly;
strcpy(this->ht,hs.ht);
}
return *this;

void nhap( );
void in( )
{
cout<<"\n"<<ht<<"\t"<<toan<<"\t"<<ly<<"\t"
<<tong;
}
}

```

```

    }
    void hoanvi(HOCSINH &hs)
    {
        HOCSINH tg = *this;
        *this = hs;
        hs = tg;
    }

    double getd( )
    {
        return tong;
    }
};

void HOCSINH:: nhap( )
{
    cout<<"\n hoc sinh:"<<i;
    cout<<" Ho ten:"; cin>> ht;
    cout<<" diem toan:"; cin>> toan;
    cout<<" diem ly:"; cin>> ly;
    tong = toan + ly;
};

void main( )
{
    HOCSINH *ds; int n; clrscr( );
    cout<<"\n Vao so hoc sinh:"; cin>>n;
    ds = new HOCSINH[n];
    for (int i = 0; i<n; i++)
        ds[i].nhap( ); clrscr( );
    cout<<" ds hoc sinh thi:";
    for (i = 0; i<n; i++)
    {
        ds[i].in( );
        getch( );
        clrscr( );
        cout<<"\n ds hs theo thu tu sx giam
dan cua diem:";
        for (i=0; i<n; i++)
            for (int j = i+1; j<n; j++)
                if (ds[i].getd( )<ds[j].getd( ))
                    ds[i].hoanvi(ds[j]);

        clrscr( );
        cout<<"\n Ket qua:";
        for(i=0; i<n; i++)
            {ds[i].in( );
            }
        getch( ); delete ds;
    }
}

```

8. Hàm chuyển đổi (Conversion Functions)

- Những hàm thành phần sử dụng để chuyển đổi những đối tượng đến hoặc từ những kiểu dữ liệu cơ bản và cho những sự chuyển đổi giữa những đối tượng của những lớp khác nhau.

```
class Converter{
private:
    int feet;
    float inches;
public:
    Converter() // cấu tử không đổi
        { feet = 0; inches =0.0;}
    Converter(float metres) //cấu tử với
        //một đổi
        {
            float f;
            f= 3.28 * metres;
            feet = int(f);
            inches = 12 *(f-feet);
        }
};
void main()
{
    Converter d1 = 1.55; //dùng cấu tử thứ hai
    Converter d2;        //dùng cấu tử thứ nhất
    d2 = 2.0;            //dùng cấu tử thứ hai
}
```

* Kiểu cơ bản (basic) đến kiểu người sử dụng định nghĩa (user-Defined) :

- Khai báo của đối tượng d1 sử dụng cấu tử thứ hai và gán giá trị 1.55.
 - Nó trình bày một sự chuyển đổi của một hằng kiểu float đến một đối tượng của lớp Converter.

float đến đối tượng d2.

- Chương trình biên dịch đầu tiên sẽ kiểm tra một hàm toán tử cho toán tử gán.
 - Nếu toán tử gán không được chùng, thì nó sẽ sử dụng cấu tử để thực hiện việc chuyển đổi.
 - Nếu cấu tử cũng không được định nghĩa chương trình biên dịch sẽ phát sinh lỗi.

* User-Defined đến Basic :

- Chương trình biên dịch phải được chỉ thị rõ ràng nếu một đối tượng phải được chuyển đổi thành kiểu dữ liệu cơ bản.
- Những chỉ thị này sẽ được mã hoá trong một hàm chuyển đổi và được định nghĩa như một thành phần của lớp đó.

```
operator float()
{
```

```
float f;
f = inches/12;
f+= float(feet);
return (f/3.28);
}
```

Có thể được sử dụng như sau, $m = d2$; hoặc sử dụng một dòng lệnh rõ ràng, $m = \text{float}(d1)$;

- Hàm chuyển đổi tiến hành chông toán tử ép kiểu.
- Hàm chuyển đổi chứa từ khoá operator và thay vì một dấu phép toán (operator symbol) nó sẽ chứa kiểu dữ liệu.
- Hàm chuyển đổi không định nghĩa kiểu trả về (return type) cũng không nên có bất cứ một đối số nào.

*** Chuyển đổi giữa những đối tượng :**

- Hàm chuyển đổi phải được chỉ định bởi vì chương trình biên dịch không biết gì về kiểu người sử dụng định nghĩa (user-defined).
- Có thể là một hàm thành phần của lớp nguồn (nằm ở phía bên phải của toán tử gán)
- Hoặc nó có thể là một hàm thành phần của lớp đích (nằm ở phía bên trái của toán tử gán).

```
objectA = objectB;
```

objectA: đối tượng của lớp đích

objectB: đối tượng của lớp nguồn.

- Chuyển đổi của những đối tượng của hai lớp khác nhau có thể thực hiện được với:
 - Cấu tử một đối số (one_argument constructor) được định nghĩa trong lớp đích.
 - Hoặc một hàm chuyển đổi (conversion function) được định nghĩa trong lớp nguồn.

Vd:

```
class LFeet{
private:
    int feet;
    float inches;
public:
    LFeet(){feet = 0; inches =0.0;}//Cấu tử 1
    LFeet(int ft,float in)           //Cấu tử 2
    {
        feet = ft;
        inches = in;
    }
};

class LMetres{
private:
    float metres;
public:
    LMetres(){metres = 0.0;}//Cấu tử 1
```

```

        LMetres(float m)           //Cấu tử 2
            {metres = m;}
};
void main()
{
    LMetres dm1 = 1.0;
    LFeet df1;
    df1 = dm1;
}

```

Hàm chuyển đổi (conversion function) trong lớp nguồn :

- Hàm chuyển đổi trong lớp nguồn để chuyển đổi chiều dài từ lớp nguồn LMetres đến lớp đích Lfeet sẽ như sau:

```

operator LFeet() //hàm chuyển đổi
{
    float ffeet, inc;
    int ifeet;
    ffeet = 3.28*metres;
    ifeet = int(ffee);
    inc = 12*(ffee - ifeet);
    return LFeet(ifeet, inc);
}

```

- `df1 = dm1;` gọi một hàm chuyển đổi ẩn.

Cấu tử trong lớp đích :

```

LFeet::LFeet(LMetres dm) //cấu tử
{
    float ffeet;
    ffeet = 3.28*dm.GetMetres();
    feet = int(ffee);
    inches = 12*(ffee - feet);
}

```

- Cũng cần phải định nghĩa hàm thành phần `GetMetres()` trong lớp nguồn Lmetres:

```

float LMetres::GetMetres()
{ return metres; }

```

Bảng cho chuyển đổi kiểu

Kiểu của Conversion	Hàm trong lớp đích	Hàm trong lớp nguồn
basic đến class	Cấu tử	
class đến basic		Hàm chuyển đổi
class đến class	Cấu tử	Hàm chuyển đổi

9. Cấu tử và đối tượng thành phần

* Lớp bao và lớp thành phần

- Nếu một lớp có các thuộc tính là các đối tượng của các lớp khác thì được gọi là lớp bao.
- Còn các lớp có các đối tượng là thuộc tính của lớp bao thì được gọi là lớp thành phần.

```
class A;
{
    private:
        int a1, a2;
        // các thành phần dữ liệu khác
    public:
        // các hàm thành phần lớp A
}
class B
{
    private:
        int b1, b2;
        A    b3; // thành phần dữ liệu của B là
                // đối tượng của A
        // các thành phần dữ liệu khác
    public:
        // các hàm thành phần lớp B
};
```

Vậy trong vd, ta có

B: là lớp bao

A : là lớp thành phần của B

* Cấu tử của lớp bao :

- Trong lớp bao, các method không thể truy cập được đến các thuộc tính của các đối tượng của lớp thành phần.
- Do đó ta phải sử dụng các cấu tử của các lớp thành phần để khởi gán cho các đối tượng thành phần của lớp bao khi xây dựng cấu tử lớp bao.
- Cách dùng cấu tử của lớp thành phần khởi gán cho đối tượng thành phần của lớp bao được viết theo mẫu sau:
tên đối tượng(<danh sách giá trị>)
- Mẫu này được viết bên ngoài thân cấu tử của lớp bao trên dòng đầu tiên. Cụ thể hơn cấu tử lớp bao được định nghĩa theo dạng:

```
tenlop(ds đối):tênđoitài(tênđoitài(ds giá trị),
...
tênđoitài(tênđoitài(ds giá trị))
{
    //các câu lệnh khởi gán các thành phần
    //dữ liệu riêng không phải là đối tượng thành
phần
};
```

* Chú ý:

- Tên đối tượng i chính là tên của đối tượng thành phần trong lớp bao
- Dấu ngoặc đơn sau tên đối tượng luôn phải có, ngay cả khi danh sách giá trị bên trong là rỗng
- Các đối tượng thành phần nếu muốn khởi tạo bằng cấu tử không đối có thể bỏ qua, không cần phải liệt kê trong cấu tử. Nói cách khác, các đối tượng không được liệt kê trên dòng đầu của cấu tử của lớp bao đều được khởi gán bằng cấu tử không đối của lớp thành phần.
- Danh sách giá trị lấy từ danh sách đối. Dựa vào danh sách giá trị, CTBD sẽ biết cần dùng hàm tạo nào để khởi gán cho đối tượng. Nếu ds giá trị là rỗng thì hàm tạo không đối sẽ được sử dụng.

Vd:

```
class A
{
    private:
        int a, b;
    public:
        A( ) // cấu tử không đối
        {
            a=0; b=0;
        }

        A(int a1, int b1) // cấu tử có đối
        {
            a=a1; b=b1;
        }

        ...
};
class B
{
    private:
        double x, y, z;
    public:
        B( ) // cấu tử không đối
        {
            x=y=z=0;
        }

        B(double x1, double y1, double z1)
        // cấu tử có đối
        {
            x=x1; y=y1; z=z1;
        }

        ...
};
class C
{
```

```

private:
    int m, n;
    A    u, v;
    B    p, r;
public:
    C(int m1, int n1, int a1, int b1, double x1,
double y1, double z1): u( ), v(a1, b1),
    r(x1, y1, z1)
        {
            m=m1;
            n=n1;
        }
};

```

■ Đoạn chương trình trên mô tả việc xây dựng cấu tử của lớp bao C dựa trên cấu tử của các lớp thành phần A và B.

■ Trong cấu tử lớp C các đối tượng thành phần được khởi gán như sau:

u: được khởi gán bằng cấu tử không đối của lớp A

v: được khởi gán bằng cấu tử hai đối của lớp A

r: được khởi gán bằng cấu tử 3 đối của lớp B

p: không có mặt được khởi gán bằng cấu tử không đối của lớpB

*** Sử dụng các phương thức của lớp thành phần :**

■ Trong lớp bao sẽ có thể sử dụng bất kỳ một phương thức nào của đối tượng thành phần theo cách:

tênđoitài tượng.tênpt()

Vd:

```

class A
{
private:
    int a;
public:
    A( )
    {
        a=0
    }
    A(int i)
    {
        a=i;
    }
    void in( )
    {
        cout<<"a"<<a
    }
};
class B
{

```

```
private:
    int b;
    A x;
public:
    B( ): x( )
        { b=0; }

    B(int i, int j): x(i)
        {
            b=j;
        }
    void in( )
        {
            x.in( );
            cout<<"b="<<b;
        }
};

void main( )
{
    B z(10, 20);
    z.in( );
    getch( );
}
```

III. ĐỐI TƯỢNG VÀ HÀM

1. Static member :

- Trong 1 lớp có thể chứa các thành phần tĩnh (static)
- Để khai báo các thành phần tĩnh này chỉ cần thêm vào dòng khai báo của chúng từ khoá đầu là static.
- Có hai loại thành phần tĩnh :
 - Dữ liệu tĩnh (Static Data Members)
 - Phương thức tĩnh (Static Member Functions)

Thành phần dữ liệu tĩnh (static data members) :

- Có ích khi mọi đối tượng của lớp cùng chia sẻ một mục thông tin chung
- Nếu một thành phần dữ liệu của một lớp được định nghĩa là static, thì dù cho có bao nhiêu đối tượng đi nữa, chúng ta cũng chỉ cần tạo ra mỗi một thành phần dữ liệu đó cho lớp.
- static: cấp phát 1 vùng nhớ cố định.
- Chỉ hiện hữu trong phạm vi của lớp, nhưng thời gian sống của nó xuyên suốt toàn bộ chương trình.

```
static kieu_du_lieu bien;
```

Ví dụ :

```
class race_cars
{
private:
    static int count;
    int car_number;
    char name[30];
public:
    race_cars(){count++;} //câu tử để tăng biến count
    ~race_cars(){count--;} //huỷ tử để giảm count
};
int race_cars::count;
```

- Thành phần dữ liệu tĩnh nên được khởi tạo trước khi hàm main() bắt đầu.

Quy cách khai báo thành phần dữ liệu tĩnh :

```
class <tên lớp>
{
    [private/ public]:
        static <kieu_ du_lieu> <tên thành phần dl tinh>;
        ...
    [public/ private]:
        <cac phuong thuc>
};
```

Cách truy xuất thành phần dữ liệu tĩnh :

- **C1:** Tương tự như kiểu dữ liệu thông thường: truy xuất thông qua tên đối tượng và tên thành phần dữ liệu, chúng được kết hợp với nhau bởi dấu . hoặc dấu → phụ thuộc vào từng loại đối tượng.
- **C2:** thông qua tên lớp và tên thành phần dữ liệu tĩnh, chúng được nối với nhau thông qua phép toán phạm vi :: theo kiểu:
`<tênlớp>::<tên của thành phần dữ liệu tĩnh>`
- Cách này xem là tốt hơn vì các thành phần dữ liệu tĩnh không gắn với 1 đối tượng cụ thể nào. Do đó C2 trực quan hơn C1.
- Thành phần dữ liệu tĩnh giống như những thành phần dữ liệu khác.
- Nếu một thành phần tĩnh được khai báo là thuộc tính private trong một class, thì những hàm không phải là thành viên không thể truy xuất được nó.
- Nếu nó được khai báo như là public, thì bất cứ thành phần nào của lớp cũng có thể truy xuất được.
- Thành phần tĩnh có thể trở thành một dữ liệu toàn cục (global) cho lớp.

Vd1:

```
class A
{
    private:
        int x;
        static int ts;
        ...
};
A u, v;
```

Khi đó:

```
u.x;      v.x → ở 2 vùng nhớ khác nhau
          → u.x ≠ v.x
u.ts;     v.ts → cùng 1 vùng nhớ
          → u.ts = v.ts
```

→ Các thành phần tĩnh là chung cho cả lớp

Vd2: Xây dựng lớp NGUOI như sau:

```
class NGUOI
{
    private:
        ...
        float tn; //thu nhập từng người
        static int tongsn; //tổng số người có trong
                          //lớp (số đối tượng)

        static float tongtn; //tổng thu nhập của
                          //các đối tượng
        ...
    public:
        ...
};
```

NGUOI nguoi1, nguoi2;

Khi đó:

- nguoi1.tn và nguoi2.tn sẽ chiếm 2 vùng nhớ khác nhau
- nguoi1.tongtn và nguoi2.tongtn chỉ là 1 giá trị, chúng chứa chung trong 1 vùng nhớ của class.
- Thành phần tĩnh tongtn tồn tại ngay cả khi các đối tượng nguoi1 và nguoi2 chưa khai báo.
- Việc truy xuất tới hai thành phần dữ liệu tn và tongtn cũng khác nhau

Vd: - Nếu viết nguoi1.tn, nguoi2.tn thì sẽ truy xuất tới thành phần tn của nguoi1 và tới thành phần tn của nguoi2 (truy xuất tới hai vùng nhớ của hai đối tượng)

- Nếu viết NGUOI::tongtn, nguoi1.tongtn hoặc nguoi2.tongtn thì sẽ truy xuất tới cùng 1 vùng nhớ là tongtn của lớp, chúng có tác dụng như nhau và có thể thay thế nhau.

Khởi tạo giá trị cho thành phần tĩnh:

```
<kieu_du_lieu> <tenlop>::<tên thành phần dữ liệu tĩnh>;
```

hoặc

```
<kieu_du_lieu> <tenlop>::<tên thành phần dữ liệu static> = <gia tri>;
```

Vd:

```
float NGUOI::tongtn; // khởi gán cho tongtn giá trị 0
float NGUOI::tongtn=10; // khởi gán cho tongtn giá
// trị 10
```

```
int NGUOI::tongs; // khởi gán cho tongs giá trị 0
int NGUOI::tongs=1; // khởi gán cho tongs giá trị 1
```

Chú ý: Khi khởi gán giá trị thì thành phần tĩnh chưa tồn tại. Vd sau sẽ minh họa điều này:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
class NGUOI
{
private:
char maso[9], hoten[25];
float tn;
static int tongs;
static float tongtn;
public:
static void in( )
{
cout<<"\n tong so nguoi:"<<tongs<<
```

```
"tong thu nhap:"<<tongtn;
    }
};
```

```
void main( )
{
    NGUOI::in( ); getch( );
}
```

- Các thành phần dữ liệu tĩnh `tongsn` và `tongtn` chưa khởi gán nên chưa tồn tại. Vì vậy các câu lệnh trong hàm tĩnh `in()` là không logic.

Khi dịch chương trình, CTBD sẽ đưa ra các thông báo lỗi:

```
undefined symbol NGUOI::tongsn in module...
undefined symbol NGUOI::tongtn in module...
```

Khắc phục:

- Đưa thêm vào chương trình các dòng lệnh khởi gán giá trị cho các biến `static` `tongsn` và `tongtn`. Chương trình được viết lại như sau:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
class NGUOI
{
    private:
        char maso[9], hoten[25];
        float tn;
        static int tongsn;
        static float tongtn;

    public:
        static void in( )
        {
            cout<<"\n tong so nguoi:"<<tongsn<<
            "tong thu nhap:"<<tongtn;
        }
};

int NGUOI::tongsn=0;
float NGUOI::tongtn=0;
void main( )
    clrscr( );
    {
        NGUOI::in( ); getch( );
    }
```

Sử dụng thành phần dữ liệu tĩnh khi nào:

- Với những thuộc tính của lớp có các thể hiện chung trên mọi đối tượng của lớp thì người ta chọn chúng là các thuộc tính static.

Vd: Trong một gia đình,

+ Mỗi người có riêng mã số, họ tên, thu nhập..., đó chính là các thể hiện riêng của mỗi người và chúng sẽ là những thành phần dữ liệu thông thường của lớp NGUOI.

+ Thuộc tính tổng số người và tổng thu nhập của gia đình là các dữ liệu chung cho cả lớp nên chúng được chọn là thuộc tính static trong lớp NGUOI.

Bài tập:

Thêm mới một hoá đơn, xoá, sửa. Ta luôn luôn biết được tổng số hoá đơn và tổng số tiền. Vậy 2 thành phần dữ liệu này, ta xây dựng là thành phần dữ liệu tĩnh.

```
class HD
{
    private:
        char *tenhang;
        float sotien;
        static int tshd;
        static double tstien;
    public:
        HD (char *tenhang1=NULL, float st=0.0)
        {
            strcpy(tenhang, tenhang1);
            ++tshd; // tăng lên 1
            tstien+ =sotien; // cộng dồn thêm
                        // tổng số tiền
        }
        ~HD( )
        {
            --tshd;
            tstien- =sotien;
        }
        void sua( )
        {
            cout<<"tenhang:"<<tenhang;
            cout<<"sotien:"<<sotien;
            tstien- =sotien;// trừ đi số tiền cũ
                        // của hoá đơn
            cout<<sua so tien thanh:";
            cin>>sotien;
            tstien+ =sotien;//cộng số tiền vừa
sửa
                                                //vào tổng số tiền
        }
        void in( )
        {
            cout<<"Tong so hoa don:"<<tshd;
```



```

        cout<<"Tong so tien:"<<tstien;
    }
int HD::tshd=0; double HD::tstien=0.0;
void main( )
{
    HD h1, h2, h3;
    h1= new HD("Rau",10);
    h2= new HD("Thit",20);
    h3= new HD("Cachua",40);
    h2.in( );
    getch( );
    delete h2;
    h3.in( );
    getch( );
    h1.sua( );
    h3.in( );
    getch( );
}

```

Hàm thành phần tĩnh (static member functions) :

- Một hàm thành phần tĩnh có thể thực hiện chỉ trên thành phần dữ liệu tĩnh của lớp.
- Hoạt động với phạm vi toàn cục đối với tất cả các thành phần của lớp của nó mà không ảnh hưởng đến phần còn lại của chương trình.
- Nó không có một con trỏ this.
- Có thể sử dụng một hàm thành phần tĩnh để truy vấn một đối tượng để tìm ra nó là đối tượng nào.
- Có ích trong việc gỡ rối một chương trình giữa những tình huống khác nhau

Vd: Nếu ta dùng câu lệnh:

```
NGUOI nguoi1, nguoi2;
```

xét trên lớp NGUOI đã xây dựng trong mục trước thì để gọi hàm thành phần tĩnh in() ta có thể dùng một trong các lệnh sau:

```

nguoi1.in( );
nguoi2.in( );
NGUOI::in( );

```

Vd:

```

class alpha
{
    private:
        static int count;//thành phần dữ liệu tĩnh
    public:
        alpha(){count++;};//câu tử tăng biến count
        static void display_count() //hàm thành phần
tĩnh
        { cout<<count; }

};
void main()

```

```

{
    alpha::display_count(); //trước khi bất cứ một
                            //đối tượng nào được tạo

alpha obj1, obj2, obj3;
alpha::display_count();    //sau khi 3 đối tượng
                            //được tạo
}

```

- Thậm chí khi không có đối tượng nào được tạo chúng ta vẫn có thể trực tiếp gọi thành phần tĩnh sử dụng tên của lớp và toán tử phạm vi như sau:

```
alpha::display_count();
```

Các tính chất của hàm thành phần tĩnh :

- Hàm thành phần tĩnh cũng giống như các method thông thường ở chỗ trong thân của hàm thành phần tĩnh có thể truy cập tới các thành phần dữ liệu của lớp với điều kiện các thành phần này không phải là của đối tượng chủ thể this.
- Hàm thành phần tĩnh là chung cho cả lớp, nó không lệ thuộc vào 1 đối tượng cụ thể nào, nó tồn tại ngay cả khi lớp chưa có đối tượng nào. Do đó, các phương thức tĩnh có thể được gọi cho dù các đối tượng của lớp đó có khởi động hay không (được định nghĩa hay chưa được định nghĩa).
- Một hàm thành phần tĩnh sẽ không hề liên hệ với một đối tượng implicit nào xác định. Vì không đòi hỏi một đối tượng implicit nào, nên 1 hàm thành phần được khai báo là static sẽ không có con trỏ this.
- Chính vì điều này nên không thể dùng hàm thành phần tĩnh để truy xuất tới dữ liệu của đối tượng chủ thể (đối tượng this) trong lời gọi hàm thành phần tĩnh (trừ thành phần dữ liệu static).

Vd: Nếu ta cho thêm vào trong phương thức tĩnh `in()` của lớp `NGUOI` các dòng lệnh để in ra các thuộc tính riêng (`maso`, `hoten` và `tn`) của đối tượng chủ thể `this` thì sẽ có chương trình sau:

```

#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<string.h>
#include<stdlib.h>
class NGUOI
{
    private:
        char maso[9], hoten[25];
        float tn;
        static int tongsn;
        static float tongtn;
    public:
        static void in( )
        {

```

```

        cout<<"\n ma so:"<<maso<<"ho
ten:"<<hoten<<"thu nhap:"<<tn;
        cout<<"\n tong so nguoi:"<<tongsn<<
"tong thu nhap:"<<tongtn;
    }
};
int NGUOI::tongsn=0;
float NGUOI::tongtn=0;
void main( )
    clrscr( );
    {
        NGUOI::in( ); getch( );
    }

```

- Nếu chạy chương trình này thì chương trình dịch của máy sẽ thông báo lỗi:

member maso cannot be used without an object
 member hoten cannot be used without an object
 member tn cannot be used without an object

- Như vậy, phương thức không thể truy xuất tới các thành phần dữ liệu của đối tượng chủ thể this.

2. Cấp phát bộ nhớ động (dynamic memory)

- Một đối tượng được tạo khi sự định nghĩa nó xuất hiện trong chương trình và nó sẽ bị huỷ khi tên của nó ra khỏi phạm vi hoặc giới hạn của chương trình.
- Có ích để tạo ra một đối tượng mới mà sẽ chỉ tồn tại lâu như nó cần.
- new tạo ra những đối tượng như vậy và toán tử delete có thể được sử dụng để phá huỷ chúng sau đó.

* New :

- Toán tử new được sử dụng để tạo một không gian nhớ cho một đối tượng của một class.
- Cú pháp chung của toán tử new là:

```
kieu_dl bien_con_tro = new kieu_dl;
```

Ví dụ,

```

int *p;           // con trỏ chỉ đến kiểu integer
float *f;        // con trỏ chỉ đến kiểu float
p = new int;     // cấp phát bộ nhớ cho integer
f = new float;   // cấp phát bộ nhớ cho float

```

- Nếu gọi đến new thành công, nó sẽ trả về một con trỏ đến không gian mà đã được cấp phát.
- Trả về 0 nếu không gian không có sẵn hoặc nếu có một vài lỗi được phát hiện.

Vd:

```

Student *stu_ptr;
//con trỏ đến một đối tượng của kiểu Student
stu_ptr = new Student;

```

Toán tử `new` tương tự như hàm `malloc()` được sử dụng trong C.

*** Delete :**

- Đối tượng được tạo bởi `new` tồn tại cho đến khi nó bị huỷ hoàn toàn bởi `delete`.

```
delete pointer_variable;
```

- Ví dụ về `new` và `delete`.

```
int *ptr;
ptr = new int;
*ptr = 12;
cout << *ptr;
delete ptr;
```

- Luôn giữ thói quen tốt là hãy `delete` bộ nhớ khi ta đã hoàn thành công việc với nó.
- Hãy lưu ý rằng chúng ta không được sử dụng những con trỏ chỉ đến những bộ nhớ mà đã bị xoá.

3. Cấp phát mảng :

- Cấp phát khối gồm những mảng có chiều dài khác nhau ta cũng sử dụng một kỹ thuật tương tự.

```
int *ptr;
ptr = new int[100];
delete [] ptr;
```

- Bất cứ khi nào ta cấp phát một mảng những đối tượng sử dụng `new`, Ta phải sử dụng `[]` trong dòng `delete`.
- CTBD sẽ báo lỗi nếu ta `delete` một biến được `malloc`'ed và nó cũng sẽ báo lỗi nếu ta `free` một biến được cấp phát bởi `new`.

4. Biến, mảng đối tượng

- Một lớp được xem như một kiểu dữ liệu(hay là 1 kiểu đối tượng) do đó có thể khai báo biến, mảng theo các kiểu sau:

```
tenlop tenbien(ds doi so);
tenlop tenmang[kichco];
```

Vd: Sử dụng lớp `DIEM` có thể khai báo các biến, mảng `DIEM` như sau:

```
DIEM d[21]; // Khai báo mảng d gồm 21 phần tử
DIEM d1, d2, d3; // Khai báo 3 đối tượng d1, d2, d3
```

- Sau khi khai báo thì 1 biến hoặc 1 mảng (đối tượng) đều được cấp phát 1 vùng nhớ riêng để chứa các thuộc tính riêng của chúng nhưng sẽ không có vùng nhớ riêng để chứa các phương thức. Các phương thức sẽ được sử dụng chung cho tất cả các đối tượng cùng lớp.

Trong vd trên :

```
sizeof(d1) = sizeof(d2) = sizeof(d3) = 3 * sizeof(int) = 6
sizeof(d) = 21 * 6 = 126
```

Để truy nhập đến thuộc tính của đối tượng thì ta viết:

```
tenbien.tenthuoc_tinh
tenmang[chiso].tenthuoc_tinh
(tenđoitượng.tenthuoc_tinh)
```

Vd: Với vd trên, mỗi đối tượng d1,d2,d3 và mỗi phần tử d[i] đều có 3 thuộc tính là x,y,z. Để truy nhập đến thuộc tính ta viết như sau:

```
d1.x // thuộc tính x của obj d1
d2.x // thuộc tính x của obj d2
d3.y // thuộc tính y của obj d3
d[3].z // thuộc tính z của phần tử d[3]
d1.x = 99 // gán 99 cho d1.x
d2.y = d1.x // gán d1.x cho d2.y.
```

Để gọi đến phương thức thì phải dùng :

```
tênbiến.tenpt( )
mang[chỉ số].tenpt( );
```

Vd:

```
d1.nhap( ); // nhập số liệu vào các thành
            // phần d1.x,d1.y,d1.z
d[3].nhap( ); //nhập số liệu vào các thành
            // phần d[3].x, d[3].y, d[3].z
```

Vd: Sử dụng lớp DIEM để nhập 3 điểm, hiện rồi ẩn các điểm vừa nhập.

Sử dụng hàm dohoa() để khởi động đồ hoạ

```
#include<conio.h>
#include<iostream.h>
#include<graphics.h>
class DIEM
{
private:
    int x,y,z;
public:
    void nhap( );
    void an( );
    {
        putpixel(x,y,getbkcolor( ));
    }
    void hien( );
};

void DIEM::nhap( )
{
    cout<<"\n Nhập hoành do(cot) va tung
do(hang) cua diem:";
    cin>>x>>y;
    cout<<"\n Nhập ma mau cua diem:";
    cin>>z;
}

void DIEM::hien( )
{
    int mauht;
    mauht= getcolor( );
```

```

        putpixel(x,y,z);
        setcolor(mauht);
    }
void dohoa( )
{
    int mh, mode;
    mh=mode=0;
    initgraph(&mh, &mode, ' ');
}
void main( )
{
    DIEM d1, d2, d3;
    d1.nhap( );
    d2.nhap( );
    d3.nhap( );
    dohoa( );
    setbkcolor(BLACK);
    d1.hien( );
    d2.hien( );
    d3.hien( );
    getch( );
    d1.an( );
    d2.an( );
    d3.an( );
    getch( );
    closegraph( );
}

```

5. Con trỏ đối tượng

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

```
tenlop *ten_con_trỏ;
```

Vd: dùng lớp DIEM để 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

```

- Ta 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 1 đối tượng và chứa địa
              // chỉ của nó vào p3

```

- Để sử dụng thuộc tính của đối tượng thông qua con trỏ, ta viết như sau:

```
ten_contro→tenthuoctinh;
```

- Sử dụng phương thức:

```
ten_contro→tenpt( );
```

Con trỏ, đối tượng và mảng:

```
A    d[20];
A    *p;
p = d // p trỏ tới đầu mảng d
```

Chú ý:

Nếu con trỏ đối tượng chứa địa chỉ đầu của mảng (trỏ vào đầu mảng đối tượng) thì có thể dùng tên con trỏ thay cho tên mảng.

(dùng p[i].y thay cho d[i])

Theo trên ta thấy:

```
p1→x và d2.x là như nhau
p2[i].y và d[i].y là như nhau
```

Tóm tắt:

Ta có quy tắc sau

- Quy tắc sử dụng thuộc tính: sử dụng thuộc tính của đối tượng ta phải dùng phép . hoặc phép →. Trong chương trình không cho phép viết tên thuộc tính 1 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_contrỏ→tên_thuộc_tính
tên_mảngđoìitượng[chisố].tên_thuộc_tính
tên_contrỏ[chisố].tên_thuộc_tính
```

Để truy nhập đến các thành phần của đối tượng thì ta phải dùng tên biến hoặc mảng

```
tênbiên/tênmảng.tênthànhphần
tênthànhphần có thể là thuộc tính hoặc phương thức...
con trỏ→tênthànhphần
```

Vd: Chương trình dưới đây sử dụng lớp DIEM để nhập 1 dãy điểm, hiển thị và ẩn các điểm vừa nhập. Chương trình dùng 1 con trỏ kiểu DIEM và toán tử new để tạo ra 1 dãy đối tượng

```
#include<conio.h>
#include<iostream.h>
#include<graphics.h>
Class DIEM
{
private:
    int x,y,z;
public:
    void nhap( );
    void an( );
    {
        putpixel(x,y,getbkcolor( ));
    }
    void hien( );
```

```

};
void DIEM::nhap( )
{
    cout<<"\n nhap hoành do va tung do cua
diem:";
    cin>>x>>y;
    cout<<"\n nhap ma mau cua diem:";
    cin>>z;
}
void DIEM::hien( )
{
    int mauht;
    mauht = getcolor( );
    putpixel(x,y,z);
    setcolor(mauht);
}
void dohoa( )
{
    int mh, mode;
    mh = mode = 0;
    initgraph(&mh, &mode, "");
}
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].nhap( );
    dohoa( );
    for(i=1; i<=n; ++i)
        p[i].hien( );
    getch( );
    for(i=1; i<=n; ++i)
        p[i].an( );
    getch( );
    closegraph( );
}

```

6. Đối tượng hằng (Const object)

Giống như các phân tử dữ liệu khác, một đối tượng có thể được khai báo là hằng bằng cách dùng từ khoá const.

Vd:

```

class DIEM
{

```



```

private:
    int x, y;
    int m;
public:
    DIEM()
    {
        x=y=m=0;
    }

    DIEM(int x1, int y1, int m1=15)
    {
        x=x1; y=y1; m=m1;
    }

    ...
};
const DIEM d= DIEM(200,100); //Khai báo đối tượng
hằng

```

- Khi khai báo cần sử dụng các câu từ để khởi gán giá trị cho đối tượng hằng

- Giá trị khởi tạo có thể là hằng, biến, biểu thức, hàm.

- Các phương thức có thể sử dụng cho các đối tượng hằng là cấu tử và huỷ tử. Về lý thuyết, các đối tượng hằng không thể bị thay đổi, mà chỉ được tạo ra hoặc huỷ bỏ đi.

- Khi dùng một method cho đối tượng hằng thì CTBD sẽ cảnh báo:

Non-const function called for const object

- Tuy nhiên khi thực hiện chương trình nội dung các đối tượng hằng vẫn bị thay đổi.

Vd: Chương trình sau sẽ minh hoạ điều này

Phương thức toán tử ++ vẫn có thể làm thay đổi đối tượng hằng

```

#include<conio.h>
#include<iostream.h>
#include<string.h>
#include<math.h>
class PS
{
private:
    int t, m;
public:
    PS()
    {
        t=m=0;
    }
    PS(int t1, int m1)
    {
        t=t1; m=m1;
    }
    PS operator++()

```

```

        {
            t +=m;
            return *this;
        }

void in()
{
    cout<<"\nPS="<<t<<"/"<<m;
}
void nhap()
{
    cout<<"\n Nhap tu va mau:";
    cin>>t>>m;
}
};

void main()
{
    int t1=-3, m1=5;
    const PS p=PS(abs(t1)+2, m1+2); // Khai bao
                                    // đối tượng hàng

    clrscr();
    p.in();
    ++p;
    p.in();
    getch();
}

```

BÀI TẬP CHƯƠNG 3

Bt1: Hãy xác định lỗi, nếu có, trong những lời khai báo dưới đây:

```
int fn1(int, int = 0, char* = 0);
```

```
int fn2(int = 0, int = 0, char*);
```

```
int fn3(int = 0, int, char* = 0);
```

Bt2: Xây dựng một lớp gồm các đối tượng là các lý lịch. Viết chương trình để nhập và in hai lý lịch (hai đối tượng của lớp LY_LICH là nhanvien1, nhanvien2)

Bt3: Viết chương trình nhập ba điểm bất kỳ (toạ độ ba điểm), vẽ tam giác tạo thành bởi 3 điểm vừa nhập và tính chu vi của tam giác này.

Bt4: Nhập một dãy hình chữ nhật, tìm hình chữ nhật có diện tích lớn nhất và hình chữ nhật có chu vi lớn nhất.

- Nhập một dãy hình chữ nhật
- Tìm và in hình chữ nhật có diện tích lớn nhất và hình chữ nhật có chu vi lớn nhất.

Bt5: Nhập dãy điểm bất kỳ, tìm tam giác có diện tích lớn nhất trong các tam giác tạo ra bởi các đỉnh là các điểm vừa nhập.

- Nhập dãy các điểm
- Tìm và in tam giác có diện tích lớn nhất

Bt6: Viết chương trình quản lý nhân sự của các khu phố trong thành phố, gồm các chức năng:

- Nhập dữ liệu cho các thành viên của gia đình trong khu phố
- Sửa đổi thông tin của một thành viên của gia đình trong khu phố
- In gia đình có người có tuổi cao nhất và tuổi thấp nhất trong từng khu phố

Bt7: Cho một danh sách cán bộ gồm các thông tin: mã số, họ tên và thu nhập. Viết chương trình in ra danh sách cán bộ với các thông tin: mã số, họ tên, thu nhập và ở cuối danh sách in ra tổng số cán bộ hiện có cùng tổng lương của họ. Chương trình chính gồm các chức năng cơ bản:

- Nhập thông tin của từng cán bộ trong danh sách cán bộ
- In danh sách cán bộ cùng thông tin về tổng số người và tổng thu nhập.

Bt8: Xây dựng lớp ma trận và lớp vectơ. Trong các lớp có xây dựng phương thức để thực hiện phép toán nhân ma trận với vectơ (sử dụng cấu tử, huỷ tử).

Bt9: Xây dựng lớp HINH_TRON với các thuộc tính r (bán kính), m (màu hình tròn), x, y (vị trí hiển thị của hình tròn trên màn hình), *pr (trỏ tới vùng nhớ chứa ảnh hình tròn), hien (=1: nếu là trạng thái hiện, =0: nếu là trạng thái ẩn) Viết chương trình tạo ra các chuyển động lên, xuống của các hình tròn.

Bt10: Sử dụng lại đề bài bt4 nhưng giải quyết bằng cách dùng phương thức tĩnh.

Chương 4 :

SỰ KẾ THỪA (Inheritance)

I. KHÁI NIỆM :

Thừa kế là một trong bốn nguyên tắc cơ sở của LTHĐT. Đặc biệt đây là cơ sở cho việc nâng cao khả năng sử dụng lại các bộ phận của chương trình, nhờ đó có thể phát triển, bổ sung để nâng cấp các chương trình và các lớp đã được sử dụng trước đó.

II. KẾ THỪA ĐƠN (SINGLE INHERITANCE)

1. Kế thừa đơn :

- Kế thừa đơn là quá trình của việc tạo ra những lớp mới từ một lớp cơ sở (base class) đã tồn tại.

Vd : Chúng ta hãy xem xét một chương trình mà trong đó chúng ta tiến hành làm việc với những người đang phục vụ trong một tổ chức.

- Mỗi một lớp con (subclass) được xem như được dẫn xuất từ lớp Employee. Lớp Employee được gọi là lớp cơ sở (base class) và những lớp mới được tạo ra được gọi là lớp dẫn xuất (derived class).

- Trong một lớp có thứ bậc, những lớp dẫn xuất thừa kế những phương thức và biến của lớp cơ sở.
- Chúng cũng có thể có những thuộc tính và phương thức của riêng chúng.

Thuận lợi

- Thuận lợi quan trọng nhất: Khả năng sử dụng lại của mã .
 - Một lớp cơ sở được xây dựng một lần, nó có thể thích ứng để làm việc trong những tình huống khác nhau.
- Kết quả của khả năng sử dụng lại mã là sự phát triển của thư viện lớp (class libraries).
 - Một thư viện lớp bao gồm dữ liệu và phương thức được bao bọc trong một lớp.
 - Dẫn xuất một lớp từ một lớp đã tồn tại cho phép định nghĩa lại hàm thành phần của lớp cơ sở và cũng được thêm vào những thành phần mới cho lớp dẫn xuất.
 - Lớp cơ sở duy trì không thay đổi trong cả quá trình.

2. Lớp cơ sở và lớp dẫn xuất :

- Sự dẫn xuất (derivation) sẽ được biểu diễn bằng dấu mũi tên từ lớp dẫn xuất đến lớp cơ sở.
- Mũi tên chỉ về hướng lớp cơ sở được hiểu rằng lớp dẫn xuất tham khảo đến hàm và dữ liệu trong lớp cơ sở, trong khi lớp cơ sở không truy xuất đến lớp dẫn xuất.
- Khai báo một lớp dẫn xuất đơn giản tương tự như bất cứ một lớp bình thường nào.
- Chúng ta cũng phải cho biết tên của lớp cơ sở. Vd,

```
class Manager : public Employee
```

- Bất cứ lớp nào cũng có thể được sử dụng như một lớp cơ sở.
- Một lớp cơ sở có thể được chia làm hai loại :
 - cơ sở trực tiếp (direct base)
 - cơ sở gián tiếp (indirect base)

Lớp cơ sở trực tiếp và không trực tiếp (direct và indirect base) :

- Một lớp cơ sở được gọi là trực tiếp nếu nó được đề cập trong danh sách cơ bản.

Vd :

```
class A
{ };
```

```
class B : public A
{ }; // lớp A là một lớp trực tiếp.
```

- Một lớp không trực tiếp có thể được viết như sau :

```
class A
{ };
class B : public A
{ };
class C : public B
{ }; //Có thể được mở rộng đến một số lượng cấp độ tùy ý.
```

Lớp dẫn xuất (derived class) :

- Định nghĩa lớp dẫn xuất :
 - Trong định nghĩa lớp dẫn xuất phải liệt kê được tên của tất cả các lớp cơ sở trực tiếp của nó và định nghĩa này được viết theo mẫu sau :

```
class <ten_lop_dan_xuat>:[public/ protected/ private]
<ten_lop_co_so_1>
[, [public/ protected/ private] <ten_lop_co_so_2>, ...]
{
    // các thuộc tính riêng của lớp dẫn xuất,
    // đó là các thành phần dữ liệu mới.
    // các phương thức, những hàm mới của
    // lớp dẫn xuất, hoặc hàm, phương thức
    // được định nghĩa lại
};
```

- Từ khoá [private/ public] dùng để xác định kiểu thừa kế là private hay public.
- Nếu bỏ qua không dùng các từ khoá này thì máy ngầm hiểu là kiểu thừa kế private.

Khả năng truy xuất (Accessibility) :

- Được thể hiện khi một hàm thành phần hoặc thành phần dữ liệu của một lớp cơ sở có thể được sử dụng bởi những đối tượng của lớp dẫn xuất.
 - Những thành phần của lớp luôn có thể được truy xuất bởi những hàm thành phần trong phạm vi lớp của chúng, bất kể những thành phần đó là private hay public.
 - Những đối tượng được định nghĩa bên ngoài lớp có thể truy xuất đến những thành phần của lớp chỉ nếu những thành phần đó là public.

Vd : nếu emp1 là một đối tượng của lớp Employee, và display() là một hàm thành phần của Employee, thì trong main() dòng lệnh emp1.display(); là hợp lệ nếu display() là public.

- đối tượng emp1 không thể truy xuất đến những thành phần private của lớp Employee.

3. Tham khảo và truy xuất những thành phần của lớp cơ sở :

- Với thừa kế:
 - Những thành phần của lớp dẫn xuất có thể truy xuất những thành phần của lớp cơ sở nếu những thành phần đó là public.
 - Những thành phần của lớp dẫn xuất không thể truy xuất đến những thành phần private của lớp cơ sở.

Kiểu của Inheritance

- Một lớp dẫn xuất có thể được khai báo với một trong các chỉ định sau: public, private và protected.

Thừa kế public và thừa kế private:

- Trong kiểu thừa kế private, tất cả các thành phần public của lớp cơ sở được thừa kế trong lớp dẫn xuất và trở thành các thành phần private của lớp dẫn xuất.
- Nghĩa là những đối tượng của lớp dẫn xuất trong main() thậm chí không thể truy cập được đến những hàm thành phần public của lớp cơ sở.
- Trong kiểu thừa kế public, các thành phần public của lớp cơ sở được thừa kế trong lớp dẫn xuất và trở thành các thành phần public của lớp dẫn xuất. Như vậy những đối tượng của lớp dẫn xuất có thể truy xuất đến những hàm thành phần public của lớp cơ sở.
- Những hàm trong lớp dẫn xuất có thể truy xuất đến những thành phần protected và public trong lớp cơ sở.
- Đối tượng của lớp dẫn xuất nằm bên ngoài lớp hoặc trong main() không thể truy xuất đến những thành phần private hay protected của lớp cơ sở.

Sự khác nhau giữa lớp dẫn xuất public và lớp dẫn xuất private .

- Đối tượng của lớp B, là lớp được dẫn xuất public từ A có thể truy xuất đến những thành phần public của lớp cơ sở.
- Tuy nhiên, đối tượng của lớp C, là lớp được dẫn xuất private từ A, không thể truy xuất đến bất kỳ thành phần nào của lớp cơ sở.
- Function trong một lớp dẫn xuất protected có thể truy xuất đến những thành phần protected và public của lớp cơ sở. Tuy nhiên, đối tượng của lớp dẫn xuất (trong main() hay bên ngoài lớp) không thể truy xuất bất kỳ thành phần nào của lớp cơ sở.

Base class members	Public Inheritance	Private Inheritance	Protected Inheritance
Public	Public	Private	Protected
Protected	Protected	Private	Protected
Private	Not accessible	Not accessible	Not accessible

Vd :

```
class A{ //lớp cơ sở
    private: int privA;
    protected: int protA;
    public: int pubA;
```

```

};
class B : public A //lớp dẫn xuất public
{
    public:
        void fn(){
            int a;
            a = privA; //error:not accessible
            a = protA; //valid
            a = pubA; //valid
        }
};
class C : private A{ //lớp dẫn xuất private
    public:
        void fn()
        { int a;
          a = privA; //error:not accessible
          a = protA; //valid
          a = pubA; //valid
        }
};
void main()
{
    int m;
    B obj1; //đối tượng của lớp dẫn xuất public
    m = obj1.privA; //error:not accessible
    m = obj1.protA; //error:not accessible
    m = obj1.pubA; //valid: B là dẫn xuất
                //public từ lớp A
    C obj2; //đối tượng của lớp dẫn xuất private
    m = obj2.privA; //error:not accessible
    m = obj2.protA; //error:not accessible
    m = obj2.pubA; //error:not accessible:
                //C là dẫn xuất private từ lớp A
}

```

☞ Nếu không cho một chỉ định truy xuất nào khi tạo một class, private sẽ được mặc định.

Thừa kế các thành phần dữ liệu :

- Các thuộc tính của lớp cơ sở được thừa kế trong lớp dẫn xuất. Như vậy tập thuộc tính của lớp dẫn xuất sẽ bao gồm hai loại :
 - Tất cả các thuộc tính của lớp cơ sở
 - Các thuộc tính mới được khai báo trong định nghĩa lớp dẫn xuất
- Tuy nhiên trong phạm vi lớp dẫn xuất, tất cả các thành phần dữ liệu thừa kế và mới đều có thể được truy cập trực tiếp, ngoại trừ thành phần dữ liệu private của lớp cơ sở .
- Để truy cập tới các thành phần dữ liệu private của 1 lớp cơ sở thì phải dùng các phương thức của chính lớp cơ sở đó. Mọi phương pháp

truy cập trực tiếp đến các thành phần private của lớp cơ sở từ lớp dẫn xuất đều không được phép.

4. Sự quan hệ giữa các thành phần trùng tên trong lớp cơ sở và lớp dẫn xuất

- Khi xây dựng lớp dẫn xuất, cho phép đặt trùng tên thuộc tính trong các lớp dẫn xuất và lớp cơ sở. Hai thành phần trùng tên này có thể cùng kiểu hay khác kiểu.
- Như vậy bên trong một đối tượng của lớp dẫn xuất có hai thành phần khác nhau có cùng tên, nhưng trong phạm vi lớp dẫn xuất tên chung đó nhằm chỉ định thành phần được khai báo lại trong lớp dẫn xuất.
- Khi muốn chỉ định thành phần cùng tên trong lớp cơ sở thì phải sử dụng tên lớp cơ sở và toán tử phạm vi "::" để đặt trước tên thành phần đó.

Vd : Nếu định nghĩa lớp dẫn xuất C như sau :

```
class A
{
    private:
        int a1;
    protected:
        int a2;
    public:
        int a3;
        ...
};

class B
{
    private:
        double a1, a2, x;
    public:
        ...
};

class C: public A, B
{
    private:
        char *a1;
        double a2, y;
    public:
        ...
};
```

- Lớp C sẽ chứa các nhóm thuộc tính sau :
 - Nhóm 1: Các thuộc tính mới
 - *a1 là thuộc tính kiểu con trỏ ký tự, a2 và y là hai thuộc tính kiểu double, a2 là thuộc tính được định nghĩa lại trong lớp dẫn xuất.
 - Nhóm 2: Các thuộc tính thừa kế
 - a1, a2, a3: là các thuộc tính kiểu int được thừa kế từ lớp A

a1,a2, x: là các thuộc tính kiểu double được thừa kế từ lớp B.

- Trong các phương thức của lớp C (phạm vi lớp C) chỉ cho phép truy cập trực tiếp tới các thuộc tính khai báo trong C (*a1, a2, y) và các thuộc tính protected hoặc public của các lớp cơ sở được thừa kế (A::a2, A::a3)

Thừa kế phương thức

- Một lớp dẫn xuất gồm có các phương thức sau :
 - Các phương thức public của lớp cơ sở sẽ được thừa kế trong lớp dẫn xuất, ngoại trừ các hàm đặc biệt: cấu tử, hủy tử và toán tử gán của lớp cơ sở .
 - Các phương thức mới được xây dựng trong lớp dẫn xuất.
- Trong các phương thức mới có thể truy cập trực tiếp tới các thuộc tính public, protected của lớp cơ sở và các thuộc tính mới của lớp dẫn xuất. Ngoài ra, trong các phương thức này cũng cho phép sử dụng các phương thức public của các lớp cơ sở

Những hàm thành phần trùng tên

- Hàm thành phần trong một lớp dẫn xuất có thể có cùng tên như những hàm thành phần trong lớp cơ sở.
- Khi một hàm được gọi với đối tượng của lớp cơ sở, thì hàm từ lớp cơ sở sẽ được gọi.
- Khi ta sử dụng tên của đối tượng của lớp dẫn xuất, thì hàm từ lớp dẫn xuất được gọi.
- Nếu một hàm thành phần từ lớp dẫn xuất muốn gọi một hàm có cùng tên của lớp cơ sở, nó cũng phải sử dụng toán tử phạm vi::
- Một hàm trong lớp cơ sở có thể được gọi bởi những đối tượng của lớp cơ sở cũng như là của lớp dẫn xuất.
- Nếu một hàm tồn tại trong lớp dẫn xuất nhưng không ở trong lớp cơ sở, nó có thể được gọi chỉ với những đối tượng của lớp dẫn xuất.

```
class Base{
    protected:
        int ss;
    public:
        int func()
            {return ss;}
};
class Derived: public Base{
    public:
        int func()
            {return Base::func();}
};
void main()
{
    Base b1;           //đối tượng của lớp cơ sở
    b1.func();        //gọi hàm func của lớp cơ sở
    Derived a1;      //đối tượng của lớp dẫn xuất
}
```

```
a1.func(); //gọi hàm func của lớp dẫn xuất
}
```

- Đây còn được gọi là sự định nghĩa lại hàm thành phần.
- Khái niệm định nghĩa lại chỉ được xét tới khi ta nói đến sự thừa kế.
 - Hàm định nghĩa lại và hàm bị định nghĩa lại giống hệt nhau về tên, tham số và giá trị trả về.
 - Chúng chỉ khác nhau về vị trí: một hàm đặt trong lớp dẫn xuất và hàm kia thì có mặt trong lớp cơ sở.
- Định nghĩa lại hàm thành phần chính là cơ sở cho việc cài đặt tính đa hình (polymorphism) của các phương thức của đối tượng .

5. Chỉ định truy nhập Protected

- Từ khoá protected tương tự private xét về mặt phạm vi và truy nhập.
 - Thành phần protected có thể được truy xuất chỉ bởi những thành phần của lớp đó.
 - Thành phần protected không thể được truy xuất bởi những đối tượng hay hàm từ bên ngoài lớp, chẳng hạn hàm main().
 - Sự khác nhau giữa private và protected chỉ xuất hiện trong lớp dẫn xuất.

Truy nhập những thành phần của lớp cơ sở

- Những thành phần của lớp dẫn xuất có thể truy xuất đến những thành phần public và protected; chúng không thể truy xuất đến thành phần private của lớp cơ sở.
 - Nó tuân theo khái niệm của object-oriented về vấn đề ẩn thông tin.
- Cho phép điều khiển sự truy xuất bằng việc cung cấp thêm một vài thành phần protected.
- Thừa kế không được thực hiện trong trường hợp ngược lại.

Luật lệ truy nhập đối với những thành phần của lớp cơ sở

Chỉ định truy nhập	Khả năng truy nhập từ chính lớp đó	Khả năng truy nhập từ lớp dẫn xuất	Khả năng truy nhập từ đối tượng bên ngoài lớp
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

Vd :

```
class Employee{ //lớp cơ sở
    private:
        int privA;
    protected:
        int protA;
    public:
        int pubA;
};
```

```
//lớp dẫn xuất
class Manager : public Employee{
public:
    void fn()
    {
        int a;
        a = privA; //error:not accessible
        a = protA; //valid
        a = pubA; //valid
    }
};
void main()
{
    Employee emp; //đối tượng của lớp cơ sở
    emp.privA = 1; //error:not accessible
    emp.protA = 1; //error:not accessible
    emp.pubA = 1; //valid
    Manager mgr; //đối tượng của lớp dẫn xuất
    mgr.privA = 1; //error:not accessible
    mgr.protA = 1; //error:not accessible
    mgr.pubA = 1; //valid
}
```

Tóm tắt việc tham khảo và truy nhập đến các thành phần của lớp cơ sở và các kiểu dẫn xuất

- Phạm vi truy nhập:
 - Mặc dù lớp dẫn xuất được kế thừa tất cả các thành phần của lớp cơ sở, nhưng trong phạm vi lớp dẫn xuất không thể truy nhập được tới tất cả các thành phần này.
 - Đó là do có 1 cơ chế kiểm soát nghiêm ngặt để tránh việc người ta có thể dùng dẫn xuất 1 lớp mới chỉ nhằm để can thiệp vào tính riêng tư của các thành phần của 1 lớp nào đó.
 - Giải pháp thường dùng là sử dụng các phương thức của các lớp cơ sở để truy cập tới các thuộc tính mang tính riêng tư của chính lớp cơ sở đó.
 - Phạm vi truy nhập tới các thuộc tính của lớp cơ sở từ bên trong lớp dẫn xuất có thể tóm tắt như sau:
 - ♣ Các thành phần private của lớp cơ sở không thể truy nhập được trong lớp dẫn xuất .
 - ♣ Các thành phần public của lớp cơ sở có thể bị truy nhập từ bất kỳ vị trí nào trong chương trình. Do vậy trong phạm vi lớp dẫn xuất các thành phần này cũng có thể được truy nhập trực tiếp.
 - ♣ Các thành phần protected của lớp cơ sở cũng có thể được truy nhập trực tiếp trong lớp dẫn xuất. Phạm vi truy nhập thành phần protected sẽ rộng hơn các thành phần private, nhưng lại hẹp hơn thành phần public.

a. Dẫn xuất public:

- Với kiểu dẫn xuất này, các hàm thành phần, các hàm bạn và các đối tượng của lớp dẫn xuất không thể truy cập được tới các thành phần private của lớp cơ sở.
- Các thành phần protected trong lớp cơ sở sẽ trở thành các thành phần protected trong lớp dẫn xuất .
- Các thành phần public trong lớp cơ sở sẽ trở thành các thành phần public trong lớp dẫn xuất.

b. Dẫn xuất protected:

- Trong kiểu dẫn xuất này, các thành phần public, protected trong lớp cơ sở tương ứng sẽ trở thành các thành phần protected trong lớp dẫn xuất .
- Thành phần private của lớp cơ sở vẫn là thành phần private trong lớp dẫn xuất.

c. Dẫn xuất private:

- Các thành phần public của lớp cơ sở không thể truy nhập được từ các đối tượng của lớp dẫn xuất , nghĩa là chúng trở thành các thành phần private trong lớp dẫn xuất .
- Các thành phần protected trong lớp cơ sở cũng sẽ trở thành các thành phần private trong lớp dẫn xuất .
- Dẫn xuất private thường dùng trong các tình huống đặc biệt khi lớp dẫn xuất không khai báo thêm các thành phần hàm mới mà chỉ định nghĩa lại các phương thức đã có trong lớp cơ sở.

Vd : Ta có sơ đồ thừa kế sau

- Giả sử lớp cơ sở A có các thành phần :
 - Thuộc tính private a1.
 - Thuộc tính protected a2.
 - Thuộc tính public a3.
- Nếu lớp B1 dẫn xuất public từ lớp cơ sở A thì A::a1 vẫn là private trong B1 song nó không thể bị truy nhập trực tiếp từ trong lớp B1 hoặc ngoài lớp B1, A::a2 trở thành protected trong B1 và A::a3 trở thành public trong lớp B1. Cả hai thuộc tính A::a2, A::a3 đều có thể bị truy nhập trực tiếp trong lớp B1.
- Do đó nếu xây dựng lớp C1 dẫn xuất private từ lớp B1 thì trong C1 có thể truy nhập tới các thuộc tính A::a2 và A::a3 của lớp A.

- Tương tự, trong lớp C2 cũng có thể truy nhập trực tiếp tới các thuộc tính A::a2 và A::a3 nhưng bên ngoài lớp C2 cả hai thuộc tính này đều không cho phép truy nhập.
- Lớp B3 được dẫn xuất private từ A, do đó cả hai thuộc tính A::a2 và A::a3 đều trở thành private trong B3 – chúng có thể bị truy nhập trực tiếp trong B3. Lớp C3 là lớp dẫn xuất private từ B3, trong lớp C3 sẽ không cho phép truy nhập trực tiếp hai thuộc tính A::a2 và A::a3 vì hai thuộc tính này đã trở thành private trong B3.

6. Cấu tử và huỷ tử của lớp dẫn xuất

Xây dựng cấu tử của lớp dẫn xuất:

- Ta đã biết rằng lớp dẫn xuất không được thừa kế các cấu tử, huỷ tử và toán tử gán của lớp cơ sở, do đó ta phải tự xây dựng các function trên cho lớp dẫn xuất.
- Các cấu tử của lớp dẫn xuất khi xây dựng phải chứa đựng các thông tin làm đối số cho các cấu tử của lớp cơ sở.
- Khi ta khai báo một đối tượng của lớp dẫn xuất, với dòng lệnh `Derived obj;` nó sẽ khiến cho cấu tử của lớp cơ sở sẽ được gọi trước và sau đó là cấu tử của lớp dẫn xuất
- Cấu tử của lớp cơ sở được cho sau cấu tử của lớp dẫn xuất, phân cách bởi dấu : như sau :

```
Derived() : Base() {}
```

- Có thể chọn lựa rõ ràng cấu tử nào của lớp cơ sở nên được gọi trong suốt lời gọi của cấu tử của một lớp dẫn xuất.
 - `Derived obj1(20);` sử dụng cấu tử một đối số trong `Derived`. Cấu tử này cũng gọi cấu tử tương ứng trong lớp cơ sở.
 - `Derived(int c) : Base(c);` //đối c được truyền //cho Base
- Có thể sử dụng cấu tử của lớp cơ sở để xây dựng cấu tử của lớp dẫn xuất.

Cách xây dựng như sau:

- Constructor cần có các đối để khởi gán cho tất cả các thuộc tính của lớp (kể cả của lớp cơ sở)
- Các thuộc tính thì được chia làm 3 nhóm ứng với 3 cách khởi gán:
 - + Nhóm 1: Các thành phần kiểu đối tượng.
 - Trong lớp dẫn xuất không được phép truy nhập tới các thuộc tính của các đối tượng này, vì vậy để khởi gán cho các đối tượng thành phần thì phải dùng cấu tử của các lớp thành phần.

Cách viết :

```
tên_đối_tượng(<danh_sach_đối>)
```

+ Nhóm 2: Tập các thuộc tính được kế thừa từ lớp cơ sở.

- Với các thuộc tính này, để khởi gán các giá trị ban đầu cho chúng người ta phải sử dụng cấu tử của lớp cơ sở.

- Cách thức này cũng giống như khởi gán cho các đối tượng thành phần trong cấu tử lớp bao, nó chỉ khác ở chỗ:

- Để khởi gán cho đối tượng thành phần ta dùng tên đối tượng thành phần
- Còn để khởi gán cho các thuộc tính thừa kế từ lớp cơ sở ta dùng tên lớp cơ sở :

Tên đối tượng thành phần(<ds giá trị của đối>)

Tên lớp cơ sở(<ds giá trị của đối>)

- Danh sách giá trị phải tương ứng với các đối trong cấu tử của lớp thành phần hoặc của lớp cơ sở và chúng được lấy ra từ danh sách các đối của cấu tử của lớp dẫn xuất

+ Nhóm 3: Tập các thuộc tính mới khai báo trong lớp dẫn xuất .

- Trong lớp dẫn xuất tập các thuộc tính này hoàn toàn có thể được truy nhập trực tiếp tới.

- Vì vậy chúng sẽ được khởi gán bằng các câu lệnh gán trực tiếp viết trong thân cấu tử của lớp dẫn xuất .

Tóm lại, cấu tử của lớp dẫn xuất sẽ được viết theo mẫu:

```
<tênlớp dẫn xuất>(<danhsach đối>):
//bắt đầu khởi gán các thuộc tính của lớp cơ sở
[cấu tử của lớp cơ sở 1(<ds giá trị>)]
[,cấu tử của lớp cơ sở 2(<ds giá trị>)]
...
[,cấu tử của lớp cơ sở i(<ds giá trị>)]
//bắt đầu khởi gán các thuộc tính của các đối
//tượng thành phần

[, Tên đối tượng thành phần 1(<ds giá trị>)]
[, Tên đối tượng thành phần 2(<ds giá trị>)]
...
[, Tên đối tượng thành phần i(<ds giá trị>)]
// bắt đầu khởi gán các thuộc tính mới của
// lớp dẫn xuất
{
[Tên thuộc tính mới 1=<biểu thức hoặc giá trị>;]
[Tên thuộc tính mới 2=<biểu thức hoặc giá trị>;]
...
[Tên thuộc tính mới i=<biểu thức hoặc giá trị>;]
};
```

- Khi gọi một cấu tử của lớp dẫn xuất trình tự thực hiện sẽ là: hàm thiết lập cho lớp cơ sở, rồi đến hàm gán giá trị cho các thuộc tính của đối tượng thành phần, sau cùng là gán giá trị cho các thuộc tính mới của lớp dẫn xuất.

Chú ý : Nếu muốn dùng hàm thiết lập ngầm định của lớp cơ sở thì có thể không cần mô tả nó trong định nghĩa của hàm thiết lập lớp dẫn xuất.

Hủy tử (destructor)

- Khi một đối tượng của lớp dẫn xuất được giải phóng (bị huỷ) thì các đối tượng thành phần và các đối tượng thừa kế từ các lớp cơ sở cũng được giải phóng theo.
- Như vậy các huỷ tử tương ứng của các lớp thành phần và lớp cơ sở sẽ tự động được gọi đến.
- Huỷ tử được gọi trong trật tự ngược lại với cấu tử.
- Huỷ tử sẽ được gọi cho lớp dẫn xuất trước, sau đó cho lớp cơ sở.
 - Nếu cấu tử của lớp dẫn xuất không làm bất cứ cái gì hoặc không có thành phần nào được thêm vào trong lớp dẫn xuất, thì huỷ tử cho lớp dẫn xuất có thể là một hàm rỗng.
- Ví dụ về xây dựng cấu tử và huỷ tử trong lớp dẫn xuất :

- Xây dựng các lớp :

MONHOC: tênmôn học
số tiết

NGUOI : tên
nsinh

GIAOVIEN dẫn xuất từ lớp NGUOI, và là lớp bao của lớp MONHOC.

- Yêu cầu :

- Dùng cấu tử có đối để tạo ra con trỏ đối tượng g.
- In ra các thông tin của đối tượng này.

```
#include<iostream.h>
#include<conio.h>
class MONHOC
{
private:
char *tenmon;
int sotiet;
public:
MONHOC(char *tm1,int st1) //cau tu co doi
{
int n=strlen(tm1);
tenmon=new char[n+1];
strcpy(tenmon,tm1);
sotiet=st1;
getch( );
}
~MONHOC( )
{
delete tenmon;
sotiet=0;
cout"\n Da huy MONHOC";
}
void in( )
{
cout<<tenmon<<" So tiet:"<<sotiet;
```



```

    }
};

class NGUOI
{
    private:
        char *ten;
        int nsinh;
    public:
        NGUOI(char *ten1, int nsinh1)
        {
            int n=strlen(ten1);
            ten =new char[n+1];
            strcpy(ten,ten1);
            nsinh=nsinh1;
            cout<<"\n Da dung cau tu lop NGUOI";
            getch( );
        }
// huy tu
~NGUOI( )
{
    delete ten;
    nsinh=0;
    cout<<"\n Da huy NGUOI";
}
void in( )
{
    cout<<ten<<"Nam sinh:"<<nsinh;
}
};

class GIAOVIEN:public NGUOI
{
    private:
        char *bomon;
        MONHOC mh;
    public:
        GIAOVIEN(char *ten1,int nsinh1,char
        *tenmon1,int sotiet1,char *bomon1):
        NGUOI(ten1,nsinh1),mh(tenmon1,sotiet1)
        {
            int n=strlen(bomon1);
            bomon=new char[n+1];
            strcpy(bomon,bomon1);
        }

// Huy tu

```

```

~GIAOVIEN( )
    {
        delete bomon;
    }
void in( )
    {
        cout<<" Giao vien:";
        NGUOI::in( ); // trong lớp dẫn xuất gọi
                       // phương thức của lớp cơ sở
                       // như sau: Tenlopcs::tenpt
        cout<<" Cong tac tai bo mon:"<<bomon;
        cout<<" Day mon:";
        mh.in( ); // gọi phương thức của đối tượng
                  // thành phần mh, ta dùng:
                  // tendoituongthanhphan.tenpt
    }
};
void main( )
    {
        clrscr( );
        GIAOVIEN *g;
        g= new GIAOVIEN("Trần Uyên Trang", 1979,
            "OOP", 60, "ToanTin");
        g->in( );
        getch( );
    }

```

Bài toán: Xây dựng lớp CONGCHUC để quản lý cán bộ của các cơ quan. Lớp CONGCHUC sẽ được dẫn xuất public từ lớp cơ sở COQUAN và có một đối tượng thành phần kiểu lớp NGUOI.

Quan hệ giữa lớp CONGCHUC, lớp COQUAN và lớp NGUOI được mô tả theo sơ đồ quan hệ sau:

Vd :

Lớp NGUOI bao gồm các thành phần:

♣ Dữ liệu :

- char *maso, *hoten, *noisinh; // là các biến con trỏ kí tự để trỏ tới vùng nhớ lưu trữ các thông tin: mã số, họ tên và nơi sinh của mỗi người.

- int ngay, thang, nam; // là các biến nguyên chứa ngày, tháng và năm sinh

- char *noiohientai; // là biến con trỏ kí tự để trỏ tới vùng nhớ lưu trữ nơi ở hiện tại.

Vd :

♣ Phương thức :

- NGUOI() là cấu tử không đối

- NGUOI(char *ms, char *ht, char *ns, char *noio, int ngay1=12, int thang1=12, int nam1=2003) là cấu tử có đối của lớp NGUOI

- ~NGUOI () là huỷ tử

♣ Các hàm bạn để định nghĩa toán tử nhập xuất lớp NGUOI:

```
friend ostream& operator>>(ostream& is, NGUOI
&nguoì);
friend ostream& operator<<(ostream& os, const NGUOI
&nguoì)
```

Lớp COQUAN gồm các thành phần:

♣ Dữ liệu:

- char *ma, *ten; // là hai biến con trỏ kiểu ký tự để chỉ tới vùng nhớ sẽ chứa mã cơ quan và tên cơ quan.
 - char *diachi; // là biến con trỏ kí tự để chỉ đến vùng nhớ lưu địa chỉ cơ quan
 - longint dienthoai; // là biến số nguyên để lưu giữ điện thoại cơ quan

♣ Phương thức :

- COQUAN () là cấu tử không đối
 - COQUAN(char *ma1, char *ten1, char *dc1, long dt1) là cấu tử có đối của lớp COQUAN
 - ~COQUAN () là huỷ tử
 - COQUAN *laydccq (); // là phương thức lấy địa chỉ của đối tượng ẩn this của lớp COQUAN.

♣ Hai hàm bạn để định nghĩa toán tử nhập xuất lớp COQUAN

```
friend ostream& operator>>(ostream& is, COQUAN &cq)
friend ostream& operator<<(ostream& os, const
COQUAN &cq)
```

lớp CONGCHUC là lớp được kế thừa public các thành phần của lớp COQUAN và là lớp bao của lớp NGUOI, nó được bổ sung thêm các thành phần:

♣ Dữ liệu :

- NGUOI nguoi; // là biến kiểu NGUOI để chứa thông tin chung về một đối tượng công chức, đây là đối tượng thành phần.
 - int nambienche; // là biến nguyên để lưu năm biên chế của công chức
 - float mucluong; // là biến thực để lưu trữ mức lương của công chức
 - char *ghichu; // là biến con trỏ kiểu ký tự để lưu trữ các thông tin đặc biệt khác của công chức.

♣ Phương thức :

- ~CONGCHUC (); // là huỷ tử
 - CONGCHUC (); // là cấu tử không đối của CONGCHUC
 - CONGCHUC(char *ms, char *ht, char *ns, char *noio, int ngay1, int thang1, int nam1, char *ma1, char *ten1, char *dc1, long dt1, int nbcl, float ml1); // là cấu tử có đối của lớp CONGCHUC

♣ Hai hàm bạn để định nghĩa toán tử nhập xuất lớp CONGCHUC

```
friend ostream &operator>>(ostream& is, CONGCHUC
&cc)
friend ostream &operator>>(ostream& os, CONGCHUC
&cc)
```

Nội dung chương trình:

- Dùng cấu tử không đối để khai báo một biến đối tượng cc1 của lớp CONGCHUC; sau đó dùng toán tử nhập xuất để nhập và in ra thông tin của biến đối tượng cc1 vừa tạo.
- Dùng cấu tử có đối để tạo ra con trỏ đối tượng cc2 và biến đối tượng cc3 của lớp CONGCHUC, sau đó dùng toán tử xuất để in ra các thông tin của các đối tượng này.
- Sử dụng toán tử delete để xoá con trỏ đối tượng cc2, nhằm xem tác dụng của các hủy tử trong các lớp.

Chương trình như sau:

```
// lớp dẫn xuất có thành phần đối tượng
// thừa kế đơn
// xây dựng cấu tử, hủy tử, toán tử nhập xuất
// sử dụng phép lấy địa chỉ của đối tượng implicit
của lớp cơ sở
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<stdlib.h>
#include<string.h>
class NGUOI
{
private:
char *maso, *hoten, *noisinh;
int ngay, thang, nam;
char *noiohientai;
public:

    NGUOI( ) // cấu tử không có đối số
    {
        ngay=0;thang=0;nam=0;
        hoten=new char[25];maso=new char[9];
        noisinh=new char[45];
        noiohientai=new char[45];
        cout<<"\n Da dung cấu tử không doi
NGUOI"; getch( );
    };
    // cấu tử với đối số mặc định
    NGUOI(char *ms, char *ht, char *ns, char
*noio, int ngay1=12, int thang1=12, int nam1=2003)
    { hoten=new char[25];
maso=new char[9];
noisinh=new char[45];
```

```

        noiohientai=new char[45];
        ngay=ngay1;thang=thang1;nam=nam1;
        strcpy(hoten,ht); strcpy(maso,ms);
        strcpy(noisinh,ns);
        strcpy(noiohientai,noio);
        cout<<"\n Da dung cấu tử co doi
NGUOI";getch( );
    };
    ~NGUOI( ) // hủy tử
    {
        ngay=0;thang=0;nam=0;
        delete hoten;delete maso;
        delete noisinh;
        delete noiohientai;
        cout<<"\n Da xoa nguoi"; getch( );
    };
    // toan tu nhap xuat
    friend istream& operator>>(istream& is, NGUOI
&nguoi);
    friend ostream &operator<<(ostream& os, const
NGUOI &nguoi)
    {
        cout<<"\n Ma so:"<<nguoi.maso;
        cout<<"\n Ho ten:"<<nguoi.hoten;
        cout<<"\n Sinh ngay:"<<nguoi.ngay;
        cout<<" thang:"<<nguoi.thang<<" nam:"
<<nguoi.nam;
        cout<<"\n Noi sinh:"<<nguoi.noisinh;
        cout<<" Cho o hien tai:"<<nguoi.noiohientai;
        return os;
    }
};

```

Vd:

```

    istream& operator>>(istream& is, NGUOI &nguoi);
    {
        cout<<"HO SO NHAN SU";
        cout<<"\n An 1 phim bat ky de nhap!!!";
        cout<<"\n Ma so:"; cin.get(nguoi.maso,9);
        cout<<" Ho ten:"; cin.get(nguoi.hoten,25);
        cout<<" Sinh ngay:"; cin>>nguoi.ngay;
        cout<<" thang:"; cin>>nguoi.thang;
        cout<<" nam:"; cin>>nguoi.nam;
        cout<<" Noi sinh:";
        cin.get(nguoi.noisinh,25);
        cout<<" Noi o hien tai:";
        cin.get(nguoi.noiohientai,25);
        return is;
    }

```

```
};
```

Vd:

```
class COQUAN
{
private:
    char *ma, *ten; // ma va ten co quan
    char *diachi;
    long dienthoai;
public:
    COQUAN( ) // cấu tử không có doi
    {
        ma=new char[9]; ten=new char[25];
        diachi=new char[45]; dienthoai=655328;
        cout<<"\n Da dung cấu tử không doi
COQUAN";
        getch( );
    }
}
```

Vd:

```
COQUAN(char *ma1, char *ten1, char *dc1,
long dt1) // cấu tử có doi
{
    ma=new char[9]; ten=new char[25];
    diachi=new char[45]; strcpy(ma,ma1);
    strcpy(ten,ten1); strcpy(diachi,dc1);
    dienthoai=dt1;
    cout<<"\n Da dung cấu tử có doi COQUAN";
    getch( );
}
~COQUAN( ) // hủy tử
{
    delete ma; delete ten;
    delete diachi; dienthoai=0;
    cout<<"\n Da xoa COQUAN"; getch( );
}
```

Vd:

```
// phương thức để lấy địa chỉ của đối tượng implicit
// của lớp
COQUAN *laydccq( )
{
    return this;
}
// xây dựng toán tử nhập xuất
friend ostream& operator>>(ostream& is, COQUAN &cq)
{
    cout<<" Ma có quan:";cin.get(cq.ma,9);
    cout<<" Ten có quan:";cin.get(cq.ten,25);
    cout<<" Địa chỉ:"; cin.get(cq.diachi,25);
```

```

        cout<<" Dien thoai:"; cin>>cq.dienthoai;
        return is;
    }

```

Vd:

```

friend ostream &operator<<(ostream& os, const COQUAN
&cq)
    {
        cout<<" Ma co quan:"<<cq.ma;
        cout<<" Ten co quan:"<<cq.ten;
        cout<<" Dia chi:"<<cq.diachi;
        cout<<" Dien thoai:"<<cq.dienthoai;
        return os;
    }
};
class CONGCHUC:public COQUAN
{
    private:
        NGUOI  nguai;
        int nambienche;
        float mucluong;
        char *ghichu;

    public:
        CONGCHUC( ):nguai( ), COQUAN( )
            {
                nambienche=0; mucluong=0;
                ghichu=new char[60];
                cout<<"\n Da dung câu tử không doi
CONGCHUC";
                getch( );
            };
        // câu tử co doi.
        CONGCHUC(char *ms, char *ht, char *ns, char
*noio, int ngay1, int thang1, int nam1, char *ma1,
char ten1, char *dc1, long dt1, int nbcl, float ml1,
char *ghichu1):
        nguai(ms,ht,ns,noio,ngay1,thang1,nam1),
        COQUAN(ma1,ten1,dc1,dt1)

        {
            ghichu=new char[60];
            nambienche=nbcl;
            mucluong=ml1;
            strcpy(ghichu,ghichu1);
            cout<<"\n Dung câu tử co doi CONGCHUC";
            getch( );
        };
};

```

```

~CONGCHUC// huỷ tử

{
    nambienche=0;
    mucluong=0;
    delete ghichu;
    cout<<"\n Da xoa CONGCHUC";getch( );
};
// xay dung toan tu xuat nhap
friend istream &operator>>(istream& is, CONGCHUC
&cc)
{
    cin>>cc.nguoi;COQUAN *cq;
    cq=cc.laydccq( ); // lay dia chi đối tượng
                        // lớp cơ sở
    cin>> *cq;
    cout<<" Nam bienche:";cin>>cc.nambienche;
    cout<<" Mucluong:";cin>>cc.mucluong;
    cout<<" Ghichu:";cin.get(cc.ghichu,60);
    return is;
}

friend ostream &operator<<(ostream &os, CONGCHUC
&cc)
{
    cout<<"HO SO CAN BO";
    cout<<cc.nguoi;
    COQUAN *cq;
    cq=cc.laydccq( ); //Lay dc đối tượng lớp cơ
sở
    cout<< *cq;
    cout<<"\n Nam bien che:"<<cc.nambienche;
    cout<<"\n Mucluong:"<<cc.mucluong;
    cout<<"\n Chu y:"<<cc.ghichu;
    return os;
};
};
void main( )
{
    clrscr( ); cout<<"\n Test 1: Kiem tra cau tu";
    CONGCHUC ccl;
    cout<<"\n Vao cong chuc 1:"; cin>>ccl;
    clrscr( );
    cout<<"\n Ho so ly lich cong chuc ccl\n";
    cout<<ccl;
    getch( ); clrscr( );
}

```



```
cout<<"\n Test 2: Kiem tra cấu tử co doi voi
con tro doi tuong cc2";
getch( );
```

Vd :

```
CONGCHUC *cc2;
    cc2=new CONGCHUC("A9","Tran Uyen Trang", "Da
Nang", "Da Nang", 21, 7, 1979,"SP","DHSP","Ton Duc
Thang", 841323, 2002, 390, "Tin hoc");
    cout<<"\n Con tro doi tuong cc2="<< *cc2;
    getch( ); clrscr( );
    cout<<"\n Test 3: Cau tu co doi voi bien doi
tuong";
    CONGCHUC cc3("A8"," Le Bao Tran","Tam Ky","Da
Nang",11,2,1979,"SP","DHSP","Ton Duc Thang",
841323,2002,360,"Tin hoc");
    cout<<"\n Bien doi tuong cc3="<<cc3;
    getch( );
    cout<<"\n Thu ham huy"; delete cc2;
    getch( );
};
```

Chương trình ngoài việc minh họa cách xây dựng các cấu tử, hủy tử của lớp dẫn xuất còn đề cập đến các vấn đề:

- Cách xây dựng và sử dụng phương thức lấy địa chỉ của đối tượng ẩn this của một lớp
- Cách xây dựng các toán tử nhập xuất cho một lớp dẫn xuất.

III. ĐA THỪA KẾ (MULTIPLE INHERITANCE)

Nhắc lại về Thừa kế

- Thừa kế cho phép ta định nghĩa 1 lớp mới, gọi là lớp dẫn xuất, từ các lớp đã có, gọi là lớp cơ sở
 - Một lớp có thể là lớp cơ sở cho nhiều lớp (single inheritance).
 - Và một lớp thì có thể được dẫn xuất từ nhiều lớp cơ sở (multiple inheritance).
 - Một lớp dẫn xuất sẽ trở thành một lớp cơ sở nếu nó lại được dùng để xây dựng các lớp dẫn xuất mới. Sự thừa kế kiểu này được gọi là sự thừa kế nhiều mức.

1. Đa thừa kế (Multiple Inheritance) :

- Đa thừa kế là quá trình của việc tạo ra một lớp mới từ nhiều hơn một lớp cơ sở.
 - Lớp dẫn xuất thừa kế những thuộc tính của hai hoặc nhiều lớp cơ sở.
- Đa thừa kế có thể phối hợp những hành vi của nhiều lớp cơ sở trên một lớp đơn lẻ.
- Một đa thừa kế có thứ bậc thể hiện một sự phối hợp của những lớp cơ sở của nó.

- Cú pháp cho đa thừa kế tương tự như đối với thừa kế đơn.

```
class Teacher
{ };
class Student
{ };
class Teach_asst:public Teacher,public Student
    ■ Tên của cả hai lớp cơ sở được cung cấp ngăn cách bởi dấu phẩy.
    ■ Luật lệ của thừa kế và truy xuất cho đa thừa kế đều giống như thừa kế đơn.
```

2. Xây dựng lớp có nhiều lớp cơ sở

- Giả sử ta đã xây dựng lớp A và lớp B. Để xây dựng lớp C dẫn xuất từ A và B ta sẽ viết theo mẫu:

```
class C: public A, B
{
    private:
        // khai báo các thành phần mới
    public:
        // khai báo các phương thức mới
};
```

- Trong vd ta có:
 - C thừa kế public từ A
 - C thừa kế private từ B

- Giả sử ta đã định nghĩa các lớp A, B1 và B2. Để xây dựng lớp C dẫn xuất trực tiếp từ hai lớp B1 và B2, ta viết như sau:

```
class C: public B1, public B2
```

```

{
    private:
        // các thuộc tính riêng của C
    public:
        // các phương thức riêng của C
}

```

- Với định nghĩa này, lớp C được thừa kế public tất cả các thành phần có trong các lớp B1 và B2. Vì lớp B1 được dẫn xuất từ lớp A, nên lớp B1 được thừa kế các thành phần có trong A, do đó lớp C cũng được thừa kế các thành phần của A.

Vd2: Sử dụng sơ đồ dẫn xuất trên, nếu ta định nghĩa lớp C theo kiểu:

```

class C: public B1, B2
{
    private:
        // các thuộc tính riêng của C
    public:
        // các phương thức riêng của C
}

```

thì lớp C sẽ thừa kế theo kiểu public các thành phần lớp B1 và theo kiểu private các thành phần lớp B2.

3. Thừa kế nhiều mức, nhiều lớp và sự trùng tên

Sự thừa kế nhiều mức, nhiều lớp:

- Như đã nói ở bài trước, bất kỳ một lớp nào đã được xây dựng đều có thể đóng vai trò là một lớp cơ sở để xây dựng một lớp khác gọi là lớp dẫn xuất. Một lớp có thể dùng làm cơ sở để xây dựng nhiều lớp dẫn xuất khác nhau.
- Đến lượt mình, một lớp dẫn xuất lại có thể trở thành 1 lớp cơ sở mới nếu nó lại được dùng để xây dựng các lớp dẫn xuất mới khác.
- Quá trình này có thể được tiếp nối không hạn chế. Sự thừa kế kiểu này được gọi là sự thừa kế nhiều mức.
- Ngoài ra, một lớp có thể được dẫn xuất từ nhiều lớp cơ sở. Sự thừa kế này được gọi là sự thừa kế nhiều lớp (multiple inheritance).
 - Trong sơ đồ, các lớp A là các lớp cơ sở để xây dựng các lớp dẫn xuất B, các lớp B lại đóng vai trò là cơ sở để xây dựng các lớp dẫn xuất C...
- Trong dẫn xuất nhiều mức-nhiều lớp, các lớp dẫn xuất sẽ được thừa kế tất cả các thành phần dữ liệu và phương thức của các lớp cơ sở, kể cả các thành phần mà lớp cơ sở được thừa kế trước đó.
- Nói cách khác lớp dẫn xuất sẽ được thừa kế tất cả các thành phần của các lớp có quan hệ với nó ở các mức trước.

Vd:

```

class A1
{
    protected:
        int a1;
    public:

```

```

        void set(int i)
            {
                a1=i;
            }
};
class A2
{
    private:
        int a2;
    public:
        void set(int i)
            {a2=i;}
        int get( )
            {
                return a2;
            }
};
class B: public A1, private A2
{
    public:
        void print( )
            {
                cout<<"\n Thanh phan a1="<<a1;
                cout<<"\n Thanh phan a2="<<get();
            }
};

```

Sự trùng tên và nguyên tắc sử dụng các thành phần trùng tên trong lớp dẫn xuất:

- Ở ví dụ trước ta thấy có sự trùng tên của các thành phần trong các lớp cơ sở A1 và A2: hàm set () có trong lớp A1, lại được định nghĩa trùng tên trong lớp A2.
- Thực tế, trong thừa kế nhiều mức cho phép định nghĩa lại các thành phần (dữ liệu, phương thức), tức là các thành phần của các lớp có thể được định nghĩa trùng tên. Định nghĩa trùng tên trong thừa kế nhiều mức được xây dựng dựa trên các nguyên tắc:
 - Tên các thành phần trong cùng 1 lớp không được trùng lặp
 - Tên các lớp có liên quan tới lớp dẫn xuất không được trùng lặp
 - Các thành phần trên các lớp khác nhau có quyền được trùng lặp
- Sự trùng tên này gây ra những điểm mơ hồ khi sử dụng.

Vd: Với các lớp đã cho trong vd trên, nếu xét dòng lệnh:

```

B b1;
b1.set(5);

```

thì hàm set () nào sẽ được gọi?

- Ta có hai hàm `set ()` khác nhau từ hai lớp cơ sở `A1, A2`.
 - `A1 :: set` : là thành phần public của lớp cơ sở public → nó trở thành một thành phần public của lớp B.
 - `A2 :: set` : là thành phần public của lớp cơ sở private → nó trở thành một thành phần private của lớp B.
- Do đó ta có thể suy luận theo logic chương trình biên dịch sẽ chọn `set ()` của lớp `A1` (bản public).
- Thực ra, sẽ có lỗi báo rằng sự tham khảo đến `set ()` là chưa rõ ràng. CTBD sẽ không thể xác định kiểu truy xuất nào để áp dụng cho hàm, cho đến khi nó biết được cụ thể hàm nào đang giao tiếp với nó.
- Vậy dùng lệnh `b1.set ()` là sai.
- Để chỉ định hàm `set` nào được gọi, ta phải dùng lệnh gọi như sau:

```
b1.A1::set(5);
```

hoặc `b1.A2::set(5); // not accessible`

- Để gọi hàm `set` ta phải chỉ rõ hàm đó thuộc lớp cơ sở nào thông qua toán tử phạm vi.
- Ta nhận thấy lần gọi thứ hai là sai vì `A2 :: set` không phải là thành phần public của lớp B. Nên không thể truy xuất được từ đối tượng của lớp B (bên ngoài lớp B).
- Nhìn chung, việc sử dụng các thành phần trong lớp dẫn xuất dựa trên các quy tắc sau:

- *Quy tắc 1:* Với các thành phần được khai báo mới trong lớp dẫn xuất, khi sử dụng chỉ cần dùng tên thành phần là đủ:

```
tên_đối_tượng.tên_thành_phần
```

- *Quy tắc 2:* Với các thành phần mà lớp dẫn xuất được thừa kế từ lớp cơ sở, khi sử dụng có thể dùng một trong hai cách sau:

Cách 1: Dùng cả tên lớp và tên thành phần, chúng được nối với nhau bởi toán tử phạm vi.

```
tên_đối_tượng.tênlớp::tên_thànhphần
```

- Khi đó, CTBD sẽ dễ dàng nhận biết thành phần cần dùng thuộc lớp nào.

Cách 2: Không dùng tên lớp chỉ dùng tên thành phần

- Với kiểu dùng này, CTBD sẽ phải tự phán đoán để nhận biết thành phần sử dụng thuộc lớp nào.

- Cách phán đoán như sau:

- Trước tiên xem thành phần đang xét có phải là thành phần mới của lớp dẫn xuất hay không (quy tắc 1).
- Nếu không phải là thành phần của lớp dẫn xuất thì tiếp tục xét các lớp cơ sở theo trình tự: các lớp có quan hệ gần với lớp dẫn xuất xét trước, các lớp có quan hệ xa hơn xét sau.
- Trong trường hợp nếu thành phần đang xét có mặt đồng thời trong nhiều lớp cơ sở có cùng một đẳng cấp quan hệ

với lớp dẫn xuất thì CTBD sẽ bắt buộc đưa ra thông báo lỗi vì nó không thể xác định được thành phần đang xét thuộc lớp nào.

- Trong trường hợp này, khi cần sử dụng các thành phần, bắt buộc phải dùng theo cách 1 có kèm theo tên lớp .

Vd2:

```
class A
{
    public:
        void display();
};
class B
{
    public:
        void display();
};
class C: public A, public B
{};

void main()
{
    C obj;
    obj.display(); // ambiguos : không thể biên
dịch được
}
```

- Chương trình biên dịch sẽ không thể hiểu được hàm nào được sử dụng nếu hai lớp cơ sở có cùng tên hàm hoặc tên dữ liệu thành phần.
- Theo trên để truy xuất đến hàm hoặc thành phần dữ liệu chính xác, Chúng ta cần phải sử dụng toán tử phạm vi và tên class
obj.A::display();
obj.B::display();
- Sử dụng như vậy sẽ giúp cho người lập trình tránh được những xung đột và mơ hồ.
- Ngoài ra cũng có thể được giải quyết bằng cách định nghĩa một hàm mới display() trong lớp dẫn xuất
void C::display()
{
 A::display();
 B::display();
}
- Chương trình biên dịch sẽ có thể phát hiện ra những xung đột và giải quyết nó.

Vd: Xây dựng lớp dẫn xuất B trên hai lớp cơ sở A1 và A2. Các class A1, A2 và B có sơ đồ thừa kế:

- Chương trình chính thực hiện các công việc:
 - Khai báo một đối tượng của class B
 - Nhập thông tin cho đối tượng vừa khai báo của class B (gọi đến phương thức `nhap()` của class B).
 - In ra nội dung thành phần `ghichu` của class B
 - Thực hiện phép gọi đến phương thức `xuat()` của lớp dẫn xuất B.
 - Thực hiện các phép gọi phương thức `xuat()` của hai lớp cơ sở A1 và A2.

```
// thừa kế nhiều mức, trùng tên + quy tắc sử dụng
#include<iostream.h>
#include<conio.h>
class A1
{
    private:
        int a11;
    public:
        char *ghichu;
        void nhap( )
        {
            ghichu=" Day la ghi chu lop co so A1";
            cout<<"\n Vao gia tri lop co so A1,a11?";
            cin>>a11;
        }
        void xuat( )
        {
            cout<<ghichu;
            cout<<" a11:"<<a11;
        }
};
class A2
{
    private:
        int a21;
    public:
        char *ghichu;
        void nhap( )
        {
            ghichu=" Day la ghi chu lop co so A2";
            cout<<"\n Vao gia tri lop co so A2,a21?";
            cin>>a21;
        }
        void xuat( )
        {
            cout<<ghichu;
            cout<<" a21:"<<a21;
```

```

    }
};
class B:public A1, public A2
{
private:
    int b11;
public:
    char *ghichu;
    void nhap( )
    {
        ghichu=" Day la ghi chu lop dan xuat B";
        A1::nhap( ); A2::nhap( );
        cout<<"\n Vao gia tri lop dan xuat B,
b11?";
        cin>>b11;
    void xuat( )
    {
        cout<<ghichu;
        cout<<A1::ghichu;
        cout<<A2::ghichu;
        A1::xuat( ); A2::xuat( );
        cout<<" Day la ghi chu lop dan xuat B";
        cout<<"\n b11:"<<b11;
    }
};
void main( )
{
    B b; clrscr( ); b.nhap( ); clrscr( );
    cout<<"\n Se in ghi chu lop B";
    cout<<b.ghichu; getch( );
    cout<<"\n Goi phuong thuc xuat( ) lop B\n";
    b.xuat( );
    getch( );
    cout<<"\n Goi phuong thuc xuat( ) lop A1\n";
    b.A1::xuat( ); getch( );
    cout<<"\n Goi phuong thuc xuat( ) lop A2\n";
    b.A2::xuat( ); getch( );
}

```

■ Chương trình đã minh họa các vấn đề:

- Cách sử dụng các thành phần trùng tên (ghichu, nhap(), xuat()) trong thừa kế nhiều mức.
- Cách xây dựng phương thức của lớp dẫn xuất (nhap, xuat) dựa trên các phương thức của các lớp cơ sở (A1::nhap, A1::xuat, A2::nhap, A2::xuat). Việc sử dụng các phương thức của lớp cơ sở trong lớp dẫn xuất cũng là một phương pháp để truy xuất đến các thành phần private của lớp cơ sở từ lớp dẫn xuất.

4. Đa thừa kế : Lớp cơ sở chung (Common base class)

- Như đã nói ở phần trước, ta có rất nhiều kiểu thừa kế.
- Sẽ xảy ra khả năng coi một lớp như là một lớp cơ sở hai lần.
- Cả Teacher và Student chứa một phiên bản của những thành phần của lớp Person.
 - Khi Teaching assistant được dẫn xuất từ Teacher và Student, nó chứa hai bản copy của những thành phần của lớp Person - một từ Teacher và một từ Student.
 - Điều này sẽ phát sinh ra một mơ hồ giữa những thành phần dữ liệu của lớp cơ sở.
- Một vấn đề khác là việc khai báo một đối tượng của lớp Teaching assistant sẽ gọi cấu tử của lớp Person hai lần.

5. Cấu tử và huỷ tử trong Đa thừa kế

- Trong kế thừa đa mức cũng dùng cấu tử của lớp cơ sở để xây dựng cấu tử của lớp dẫn xuất, tuy nhiên chỉ dùng cấu tử của hai mức kề nhau giống như trong kế thừa đơn.

```
class Teacher
{
    private:
        int x;
    public:
        Teacher() // cấu tử
            {x=0;}
        Teacher(int s)
            {x=s;}
};

class Student
{
    private:
        int y;
    public:
        Student() // cấu tử
            {y=0;}
        Student(int a)
            {y=a;}
};

class Teach_asst: public Teacher, public Student
{
    private:
        int z;
    public:
        Teach_asst():Teacher(),Student()//cấu tử
            {z=0;}
        Teach_asst(int s, int a, int b):Teacher(s),
Student(a)
```

```

        { z=b; }
    };

```

- Cấu tử phải được định nghĩa để khởi tạo những thành phần dữ liệu của tất cả các lớp
- Tên của cấu tử lớp cơ sở đi sau dấu hai chấm : và được ngăn cách nhau bởi dấu phẩy ,

```
Teach_asst() : Teacher(), Student()
```

- Nếu cấu tử có đối số thì:

```

Teach_asst(int s, đối số cho lớp Teacher
           int a, đối số cho lớp Student
           int b): đối số cho Teach_asst
                Teacher(s),   gọi cấu tử Teacher
                Student(a)   gọi cấu tử Student
                {z=b;}       khởi đặt thành phần

```

dữ liệu

Trật tự chung của việc gọi cấu tử :

- Lớp cơ sở theo thứ tự xuất hiện của chúng trong danh sách của những lớp cơ sở : Teacher, Student.
- Nếu như có những đối tượng thành phần trong lớp , chúng sẽ được khởi tạo tiếp theo, theo trật tự chúng xuất hiện trong khai báo của lớp dẫn xuất .
- Chính đối tượng đó, sử dụng mã trong cấu tử của nó.

Trật tự chung của việc gọi huỷ tử :

- Huỷ tử được gọi theo trật tự ngược lại.
- Huỷ tử của lớp đó được gọi trước tiên, sau đó đến huỷ tử của các đối tượng thành phần, cuối cùng là đến lớp cơ sở.

Bài toán: Xây dựng các lớp A, B, C, D. Trong các lớp có xây dựng cấu tử , huỷ tử ... và giả sử giữa chúng có quan hệ thừa kế sau:

- Trong sơ đồ, lớp C dẫn xuất từ B, lớp D dẫn xuất từ lớp C và có thành phần là đối tượng kiểu A (hay A là lớp thành phần của D)
- Mục đích chương trình:
 - Minh hoạ cách xây dựng cấu tử, huỷ tử trên thừa kế đa mức.
 - Trình tự thực hiện các công việc khi gọi cấu tử và huỷ tử của lớp dẫn xuất .

- Cách sử dụng các thành phần trùng tên.

- Chương trình chính thực hiện các công việc:
 - Khai báo một con trỏ đối tượng kiểu D (*d)

- Cấp phát vùng nhớ và gán giá trị cho con trỏ đối tượng d (dùng hàm tạo có đối của lớp dẫn xuất D...)
- Gọi phương thức hien() của lớp D để hiện thông tin của đối tượng mà con trỏ đối tượng d chỉ tới.
- Gọi phương thức hien() của lớp B và lớp C. Phương thức này sẽ hiện thông tin của các đối tượng cơ sở tạo nên đối tượng dẫn xuất của lớp D.
- Thực hiện xoá đối tượng d, khi đó CTBD sẽ gọi tuần tự các phương thức huỷ của lớp cơ sở và lớp thành phần.

```
// thừa kế nhiều mức. Xây dựng cấu tử, huỷ tử
#include<conio.h>
#include<string.h>
#include<iostream.h>
class A
{
private:
    int a;
    char *sa;
public:
    A( )
    {
        a=0; sa=NULL;
        cout<<"\n Da dung cau tu lop thanh
phan A"; getch( );
    }
    A(int a1, char *sa1)
    {
        a=a1; strcpy(sa,sa1);
        cout<<"\n Da dung cau tu co doi lop
thanh phan A";
        getch( );
    }
    ~A( )
    {
        a=0;
        delete sa;
        cout<<"\n Da dung huy tu lop thanh
phan A";
        getch( );
    }
    void xuat( )
    {
        cout<<"\n Gia tri a cua lop thanh phan
A="<<a;
        cout<<"\n Ghi chu lop thanh phan
A:"<<sa;
    }
};
```

```

class B
{
    private:
        int b;
        char *sb;
    public:
        B( )
        {
            b=0;
            sb=NULL;
            getch( );
        }
        B(int b1, char *sb1)
        {
            b=b1;
            strcpy(sb, sb1);
            getch( );
        }
        ~B( )
        {
            b=0;
            delete sb;
            getch( );
        }
        void xuat( )
        {
            cout<<"\n Day la gia tri b cua lop
co so B="<<b;
            cout<<"\n Ghi chu lop co so
B:"<<sb;
        }
};
class C:public B
{
    private:
        int c;
        char *sc;
    public:
        C( ):B( )
        {
            c=0;
            sc=NULL;
            getch( );
        }
        C(int c1, char *sc1, int b1, char *sb1):B(b1, sb1)
        {

```

```

        c=c1;
        strcpy(sc,sc1);
        getch( );
    }
    ~C( )
    {
        c=0;
        delete sc;
        getch( );
    }
    void xuất( )
    {
        B::xuất( );
        cout<<"\n Gia tri c cua lop dan
xuất C="<<c;
        cout<<"\n Ghi chu lop dan xuất
C:"<<sc;
    }
};
class D:public C
{
    private:
        int d;
        char *sd;
        A dt1;

    public:
        D( ):C( ),dt1( )
        {
            d=0;sd=NULL;
            getch( );
        }
        D(int d1,char *sd1,int c1,char *sc1, intb1,
char *sb1, int a1, char *sa1):
        C(c1,sc1,b1,sb1),dt1(a1,sa1)
        {
            d=d1;
            strcpy(sd,sd1);
            getch( );
        }

    ~D( )
    {
        d=0;
        delete sd;
        getch( );
    }
};

```

```

        void xuất( )
        {
            dt1.xuat( );
            C::xuat( );
            cout<<"\n Gia tri d cua lop
bao+dan xuat="<<d;
            cout<<"\n Ghi chu lop bao+dan
xuat:"<<sd;
        }
    };
void main( )
{
    clrscr( );
    D h(3,"D",2,"C",1,"B",0,"A");
    cout<<" Cac gia tri trong B:";
    h.B::xuat( );
    cout<<" Cac gia tri trong B va C:";
    h.C::xuat( );
    cout<<" Cac gia tri trong A, B, C, D:";
    h.xuat( );
}

```

6. Lớp cơ sở ảo (Virtual Base class)

- Những thứ bậc của đa thừa kế có thể phức tạp và dẫn đến tình huống mà trong đó một lớp dẫn xuất thừa kế nhiều lần từ cùng một lớp cơ sở không trực tiếp (indirect class)

```

class cho
{ protected:
    int x;
};
class choxu: public cho
{};
class chotrang: public cho
{};
class choxutrang: public choxu, public chotrang
{
    public:
        int display()

```

```
        {return x;}
};
```

- Khi lớp choxu và chotrang được dẫn xuất từ cho, mỗi lớp sẽ thừa kế một phiên bản của cho. Phiên bản này được gọi là một đối tượng con (subobject)
- Mỗi đối tượng con chứa phiên bản riêng của nó của x từ lớp cho
- Khi lớp choxutrang chỉ đến x thì chương trình biên dịch không biết được phiên bản nào đang được truy xuất và vì vậy lỗi xảy ra
- Để tránh tình trạng có hai phiên bản của một lớp cơ sở, chúng ta sử dụng một lớp cơ sở virtual
- Lớp cơ sở virtual được xây dựng bằng cách thêm vào từ khoá virtual khi liệt kê lớp cơ sở trong lời khai báo lớp dẫn xuất.

```
lớp choxu: virtual public cho
{ };
```

```
lớp chotrang: virtual public cho
{ };
```

- Khai báo như vậy giúp tránh được những sự lặp lại không cần thiết của cùng một thành phần dữ liệu trong đa thừa kế có thứ bậc.
- Lớp cơ sở được thể hiện dưới dạng sau :

Cấu tử và Huỷ tử

- Sự hiện diện của những lớp cơ sở ảo làm thay đổi trật tự gọi lớp cơ sở
 - Một lớp cơ sở ảo được khởi tạo trước bất cứ một lớp cơ sở không ảo nào
 - Nếu có từ hai lớp cơ sở ảo trở lên thì trật tự của sự khởi tạo sẽ được xác định bởi vị trí của chúng trong đoạn thừa kế từ trên xuống dưới, từ trái sang phải
 - Cách gọi huỷ tử cũng theo quy tắc trên, nhưng trong trật tự ngược lại

IV. ĐA HÌNH (POLYMORPHISM)

1. Thừa kế trong lớp dẫn xuất và kết buộc tĩnh (static binding)

Thừa kế trong lớp dẫn xuất:

- Lớp dẫn xuất được thừa kế tất cả các thành phần dữ liệu và phương thức của lớp cơ sở

- Ngoài ra lớp dẫn xuất và lớp cơ sở còn có các sự tương thích sau:
 - Sự tương thích giữa đối tượng thuộc lớp cơ sở và đối tượng thuộc lớp dẫn xuất :
 - Có thể gán một đối tượng của lớp dẫn xuất cho một đối tượng của lớp cơ sở .
 - Kết quả, đối tượng lớp cơ sở chỉ chứa các phần dữ liệu tương ứng với các thuộc tính thừa kế của đối tượng dẫn xuất.
 - Trong trường hợp này chương trình biên dịch đã sử dụng phép chuyển kiểu ngầm định để chuyển đối tượng từ kiểu dẫn xuất sang kiểu cơ sở.
 - Không cho phép gán một đối tượng lớp cơ sở cho một đối tượng lớp dẫn xuất nếu chưa định nghĩa phép chuyển kiểu tường minh.

Vd: Ta có sơ đồ thừa kế sau:

- Lớp DV (động vật) là lớp cơ sở chung, nó chứa các thành phần dữ liệu và các phương thức chung cho các loại động vật được dẫn xuất từ nó. Các lớp dẫn xuất gồm có:
 - Lớp HC (lớp hai chân)
 - Lớp BC (lớp bốn chân)
 - Lớp CM (lớp con mèo, dẫn xuất trực tiếp từ lớp cơ sở BC)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class DV
{
protected:
    char *ten;
public:
    DV( )
    {
        ten=NULL;
    }
    DV(char *ten1)
    {
        ten=strdup(ten1);
    }
}
```



```

        ~DV( )
        {
            delete ten;
        }
        void xuất( )
        {
            cout<<"\n Dong vat co ten
la:"<<ten;
        }
    };
class BC:public DV
{
    public:
        BC( ):DV( )
        {}
        BC(char *ten1):DV(ten1)
        {
        }
        void xuất( )
        {
            DV::xuất( );
            cout<<"\n Loai bon chan";
        }
};
class CM:public BC
{
    public:
        CM( ):BC( )
        {
        }
        CM(char *ten1):BC(ten1)
        {
        }
        void xuất( )
        {
            BC::xuất( );
            cout<<"\n La con meo";
        }
};

void main( )
{
    CM c("Win");
    BC b("Bon chan khong ten");
    DV d("Dong vat khong ten");
    c.xuất( ); b.xuất( ); d.xuất( );
}

```

```

    getch( );
    d=c; // gan doi tuong c của lớp dx cho doi
        // tuong d của lớp cơ sở
    c.xuat( ); b.xuat( ); d.xuat( );
    getch( );
    b=c; // gan doi tuong c của lớp dan xuất cho
        // doi tuong b của lớp cơ sở
    c.xuat( ); b.xuat( ); d.xuat( );
    getch( );
}

```

Con trỏ lớp dẫn xuất _con trỏ lớp cơ sở:

- Sự tương thích giữa con trỏ lớp dẫn xuất và con trỏ lớp cơ sở:
 - Một con trỏ đối tượng thuộc lớp cơ sở có thể trỏ tới một đối tượng của lớp dẫn xuất.
 - Ngược lại một con trỏ lớp dẫn xuất không thể trỏ tới một đối tượng của lớp cơ sở nếu chưa định nghĩa phép chuyển kiểu tường minh
- Điều này cũng có các ý nghĩa sau:
 - Con trỏ lớp cơ sở có thể dùng để chứa địa chỉ các đối tượng của lớp dẫn xuất.
 - Có thể gán 1 con trỏ đối tượng của lớp dẫn xuất cho một con trỏ đối tượng của lớp cơ sở.

Vd: thay đoạn chương trình chính của vd trước bằng đoạn chương trình sau:

```

void main( )
{
    DV d("Dong vat khong ten");
    clrscr( );
    BC b("Bon chan khong ten");
    CM c("Win");
    DV *tg1, *tg2, *tg3, *tg;
    tg1=&c; tg2=&b;
    tg1->xuat( ); tg2->xuat( );
    getch( );
    CM *c1=new CM("Lu");
    tg=c1; tg->xuat( );
    getch( );
}

```

- Các dòng lệnh gán địa chỉ đối tượng lớp dẫn xuất cho con trỏ lớp cơ sở:

```

    tg1=&c;
    tg2=&b;

```

Và dòng lệnh gán con trỏ lớp dẫn xuất cho con trỏ lớp cơ sở:

```

    tg=c1;

```

là hợp lệ

Nếu ta dùng:

```

    c1=&d;

```

`c1=tg;`
thì CTBD sẽ thông báo lỗi

Tham chiếu lớp dẫn xuất _tham chiếu lớp cơ sở:

- Sự tương thích giữa tham chiếu lớp dẫn xuất và tham chiếu lớp cơ sở :
 - Tương tự như sự tương thích giữa con trỏ lớp dẫn xuất và con trỏ lớp cơ sở, biến tham chiếu của lớp cơ sở cũng có thể tham chiếu đến các đối tượng của lớp dẫn xuất nhưng không có điều ngược lại.

Vd: Có thể viết đoạn chương trình sau:

```
void main( )
{
    DV d("Dong vat khong ten");
    clrscr( );
    CM c("Win");
    DV &tg1=c; // dung bien tham chieu tg1
    tg1.xuat( ); getch( );
}
```

dòng lệnh: `DV &tg1=c;` là hợp lệ

nhưng nếu ta khai báo:

```
CM &c1=d;
```

thì CTBD sẽ thông báo lỗi.

2. Kết buộc tĩnh (Static binding)

- Khi 1 lớp dẫn xuất được thừa kế từ một lớp khác, nó sẽ thừa hưởng các thành phần dữ liệu và phương thức của lớp đó (lớp cơ sở). Đây gọi là Thừa kế đơn (single inheritance).
- Song có thể 1 lớp được thừa kế từ nhiều lớp khác trên cây thứ bậc của lớp. (như sơ đồ của vd trên). Đây chính là thừa kế bội (multiple inheritance).
- Thừa kế bội thường gây ra những khó khăn và sự mập mờ (ambiguity) cho OOP, thường xuất phát từ vấn đề nên quyết định lớp cha nào là lớp sẽ được dùng khi có 1 thông điệp gửi cho đối tượng.
- Sự mập mờ này càng gia tăng khi một lớp có nhiều lớp cha.

Vd: Lớp CM được thừa kế từ các lớp BC và lớp DV.

- Cả 3 lớp này đều có các phương thức `xuat()` của riêng nó. Vậy làm sao để CTBD biết phương thức `xuat()` nào sẽ được dùng để in ra các tham số của đối tượng thuộc lớp CM.
- Như vậy, khi thiết kế CTBD của một ngôn ngữ OOP, người ta phải thiết kế cả một hệ thống thực thi để giải quyết các vấn đề còn mập mờ trong thừa kế bội.
- Hệ thống thực thi này thường giải quyết các vấn đề còn mập mờ theo hai hình thức:
 - Kết buộc tĩnh (static binding)
 - Kết buộc động (dynamic binding)

- “Kết buộc” thể hiện sự kết nối giữa loại phương thức được thực thi và kiểu đối tượng tiếp nhận thông điệp.
- Nếu phương thức sử dụng được kết buộc với đối tượng nào thì nó chỉ làm thay đổi thông tin trên đối tượng đó
- Kết buộc tĩnh là kiểu kết buộc được xác định ngay trong quá trình biên dịch chương trình.
- Trong hình thức kết buộc tĩnh, CTBD xác định phương thức sẽ được sử dụng ứng với đối tượng nhận thông điệp dựa trên các quy tắc sau:

qt1: Nếu phương thức được gọi có kèm theo tên lớp nào thì phương thức của lớp đó sẽ được thực hiện.

- Thường sử dụng cách gọi này để gọi phương thức của lớp cơ sở trong lớp dẫn xuất.

- Muốn gọi phương thức lớp cơ sở ở phía ngoài lớp dẫn xuất thì trong lời gọi phương thức phải có thêm tên đối tượng cần tác động

qt2: Nếu phương thức được gọi kèm với đối tượng thuộc lớp nào thì phương thức của lớp đó sẽ được thực hiện.

qt3: Nếu phương thức được gọi kèm với con trỏ đối tượng kiểu lớp nào thì phương thức của lớp đó sẽ được thực hiện, bất kể con trỏ chứa địa chỉ của đối tượng lớp nào

qt4: nếu trong trường hợp phương thức cần sử dụng không có trong các quy tắc trên thì CTBD phải tự phán đoán để biết phương thức cần gọi thuộc lớp nào.

- Cách phán đoán: so sánh tuần tự phương thức cần gọi có trùng tên với phương thức của các lớp cơ sở theo trình tự: các lớp có quan hệ gần với lớp dẫn xuất xét trước, các lớp ở xa lớp dẫn xuất xét sau.

Vd: Với bài toán Động vật, ta thay đoạn chương trình chính:

```
void main( )
{
    DV *d1, *d2, *d3=new DV("Dong vat khong ten");
    clrscr( ); CM c("Win");
    CM c1("Lu");
    CM *c2=new CM("Nu");
    c.BC::xuat( );// goi phuong thuc kieu qt1
    c.xuat( ); // goi phuong thuc kieu qt2
    c2->xuat( ); // goi phuong thuc kieu qt3
    getch( );
    d1=&c; d1->xuat( );cout<<"\n";
    d2=c2; d2->xuat( ); cout<<"\n";
    d3=&c1; d3->xuat( );
    cout<<"\n"; getch( );
}
```

Với dòng lệnh:

```
d1=&c; d1->xuat( );
```

phương thức `xuat()` của lớp `DV` được dùng kết buộc với đối tượng `c` thuộc lớp `CM` (`d1` là con trỏ đối tượng kiểu `DV` trỏ tới đối tượng `c`)

`d2=c2; d2->xuat();`

phương thức `xuat()` của lớp `DV` được dùng kết buộc với đối tượng do con trỏ `c2` thuộc lớp `CM` trỏ tới.

- Như vậy, nếu kèm con trỏ đối tượng lớp cơ sở (`DV *d1, *d2, *d3`) trong lời gọi phương thức `xuat()`, dù con trỏ được gán địa chỉ của bất kỳ loại đối tượng thuộc lớp dẫn xuất nào thì CTBD vẫn kết buộc cố định phương thức `xuat()` của lớp cơ sở với đối tượng được con trỏ đó chỉ tới.

- Nói cách khác, trong kết buộc tĩnh khi gọi 1 phương thức có kèm theo con trỏ, nếu con trỏ thuộc kiểu lớp nào thì phương thức thuộc lớp đó sẽ được sử dụng.

3. Hàm ảo (Virtual Function)

- Một hàm ảo là một hàm không thật sự tồn tại nhưng ảnh hưởng đến một số phần của chương trình

- Xét một chương trình để vẽ những hình thể shapes khác nhau như hình tam giác (`triangle`), hình tròn (`circle`), hình vuông (`square`), hình `ellipse`... Mỗi lớp này có một hàm thành phần là `draw()`

```
class Shapes
{
public:
    void draw() //hàm trong lớp cơ sở
        {cout<<"Draw Base\n"; }
};
class circles: public Shapes
{
private:
    int rad;
public:
    void draw()//định nghĩa lại trong lớp dẫn xuất
        {cout<<"Draw circle";}
};
class square: public Shapes
{
private:
    int l;
public:
    void draw() //định nghĩa lại trong lớp dẫn xuất
        {cout<<"Draw square";}
};
void main()
{
    circle c;
```

```

square s;
Shape *ptr;
ptr = &c;
ptr->draw();
ptr = &s;
ptr->draw();
}

```

■ Địa chỉ của đối tượng lớp dẫn xuất được gán bởi con trỏ của lớp cơ sở
`ptr = &c;` // `c` là đối tượng của lớp dẫn xuất `circles`

■ Khi ta gọi hàm `draw()`

`ptr->draw();` ta muốn hàm `draw()` của lớp dẫn xuất `circle` sẽ được gọi

■ Nhưng kết quả là

Draw Base

Draw Base

■ Chúng ta muốn sử dụng lời gọi hàm để vẽ một hình vuông hoặc bất cứ một đối tượng nào khác tùy thuộc vào đối tượng mà `ptr` chỉ đến

- Những hàm `draw()` khác nhau nên được thực thi bởi cùng một lời gọi hàm
- Tất cả những lớp khác nhau của `Shapes` phải được dẫn xuất từ một lớp cơ sở `Shapes` duy nhất
- Hàm `draw()` phải được khai báo là ảo trong lớp cơ sở
- Sau đó nó có thể được định nghĩa lại trong mỗi lớp dẫn xuất

```
class Shapes
```

```
{
```

```
    public:
```

```
        virtual void draw() //hàm virtual
```

```
                                //trong lớp cơ sở
```

```
        {cout<<"Draw Base\n"; }
```

```
};
```

■ Một hàm ảo cho phép lớp dẫn xuất thay thế sự thực thi được cung cấp bởi lớp cơ sở

■ Chương trình biên dịch đảm bảo chắc chắn rằng sự thay thế luôn luôn được gọi bất cứ khi nào đối tượng trong lời gọi thật sự là của lớp dẫn xuất.

■ Với hàm ảo, kết quả là:

```
Draw circle
```

```
Draw square
```

■ Cùng một lời gọi hàm :

```
ptr->draw();
```

 thực hiện những hàm khác nhau tùy

thuộc vào nội dung của `ptr`

Chú ý

■ Hàm `Virtual` nên được khai báo trong lớp cơ sở

■ Kiểu trả về của một hàm thành phần phải theo sau từ khoá `virtual`

- Nếu thứ bậc của những lớp dẫn xuất được sử dụng, hàm virtual phải được khai báo ở mức độ cao nhất
- Một hàm virtual phải được định nghĩa cho lớp mà trong đó nó được khai báo lần đầu tiên.
- Hàm được khai báo lại trong lớp dẫn xuất phải có cùng những tham số (cùng số lượng và cùng kiểu dữ liệu)_ CTBD sẽ hiểu là ta muốn tiến hành chồng hàm virtual.

4. Phương thức ảo (Virtual) và kết buộc động (dynamic binding)

Phương thức Virtual:

- Nhằm giải quyết các mập mờ khi sử dụng các phương thức trong thừa kế bội
- Chỉ nên cài đặt với các phương thức trùng dòng tiêu đề (trùng tên, trùng kiểu dữ liệu và trùng các đối) của nhiều lớp có quan hệ thừa kế
- Có thể được sử dụng trong cả hai hình thức kết buộc: kết buộc tĩnh và kết buộc động. Hình thức được chọn để dùng khi sử dụng phương thức tùy thuộc vào từng hoàn cảnh cụ thể.
- Để định nghĩa các phương thức là các phương thức ảo, theo trên ta có thể dùng một trong hai cách sau:
 - **Cách 1:** Thêm từ khoá virtual vào đầu dòng khai báo của phương thức bên trong định nghĩa lớp cơ sở.
 - **Cách 2:** Thêm từ khoá virtual vào các dòng khai báo của các phương thức bên trong định nghĩa của tất cả các lớp cơ sở và lớp dẫn xuất.

Vd: Trong bài toán class DONGVAT, ta sẽ định nghĩa tất cả các phương thức `xuat()` là phương thức ảo.

C1:

```
class DV
{
protected:
    char *ten;
public:
    DV( )
    {
        ten=NULL;
    }
    DV(char *ten1)
    {
        ten=strdup(ten1);
    }
    ~DV( )
    {
        delete ten;
    }
    virtual void xuất( )
    {
```

```

        cout<<"\n Day la loai dong vat
co ten la:"<<ten;
    }
};
class BC:public DV
{
    public:
        BC( ):DV( )
        {}
        BC(char *ten1):DV(ten1)
        {}
        void xuat( )
        {
            DV::xuat( );
            cout<<"\n Loai bon chan";
        }
};
class CM:public BC
{
    public:
        CM( ):BC( )
        {}
        CM(char *ten1):BC(ten1)
        {}
        void xuat( )
        {
            BC::xuat( );
            cout<<"\n La con meo";
        }
};
C2:
class DV
{
    protected:
        char *ten;
    public:
        DV( )
        {ten=NULL;}
        DV(char *ten1)
        {ten=strdup(ten1);}
        ~DV( )
        {delete ten;}
        virtual void xuat( )
        {
            cout<<"\n Day la loai dong vat
            co ten la:"<<ten;
        }
}

```



```

};
class BC:public DV
{
public:
    BC( ):DV( )
        {
        }
    BC(char *ten1):DV(ten1)
        {
        }
    virtual void xuat( )
        {
            DV::xuat( );
            cout<<"\n Loai bon chan";
        }
};
class CM:public BC
{
public:
    CM( ):BC( )
        {
        }
    CM(char *ten1):BC(ten1)
        {
        }
    virtual void xuat( )
        {
            BC::xuat( );
            cout<<"\n La con meo";
        }
};

```

Chú ý: Từ khoá `virtual` không cho phép đặt ngoài định nghĩa lớp.

```

class DV
{
protected:
    char *ten;
public:
    DV( )
        {ten=NULL;}

    DV(char *ten1)
        {ten=strdup(ten1);}
    ~DV( )
        {delete ten; }
    virtual void xuat( );
};

```

```

        virtual void DV::xuat( )
            {cout<<"\n Day la loai dong vat co
ten:"<<ten;}

```

Trường hợp này chương trình sẽ thông báo lỗi.

Sửa lại như sau:

```

class DV
{
    protected:
        char *ten;
    public:
        DV( )
        {
            ten=NULL;
        }
        DV(char *ten1)
        {
            ten=strdup(ten1);
        }
        ~DV( )
        {
            delete ten;
        }
        virtual void xuat( );
};

void DV::xuat( )
{
    cout<<"\n Day la loai dong vat co
ten:"<<ten;
}

```

Kết buộc động (dynamic binding) và quy tắc gọi phương thức Virtual :

- Kết buộc động là kiểu kết buộc chưa được xác định rõ ràng trong quá trình biên dịch.
- Nghĩa là chương trình chưa xác định được phương thức thuộc lớp nào sẽ được gọi cho tới khi nó thực sự được nhận một đối tượng cụ thể. Địa chỉ của mã trong lời gọi phương thức được xác định tại thời điểm cuối cùng có thể, dựa trên kiểu động của đối tượng tại thời gian chạy
- Khả năng trì hoãn thực hiện cho tới khi thực thi mới quyết định đối tượng thuộc về lớp nào và phương thức thuộc lớp nào sẽ được gọi để tham khảo tới đối tượng đó thể hiện các đặc tính cơ bản trong kết buộc động.
- Phương thức ảo được dùng trong cả hai hình thức kết buộc: kết buộc tĩnh và kết buộc động. Khi thực hiện chương trình nguồn, CTBD sẽ xác định phương thức ảo nào được sử dụng ứng với đối tượng nhận thông điệp.
- Việc lựa chọn phương thức kết buộc với các đối tượng nhận thông điệp sẽ dựa trên các quy tắc:

qt1: + Nếu phương thức ảo được gọi có kèm theo tên lớp nào thì phương thức của lớp đó sẽ được thực hiện

+ Còn nếu muốn gọi phương thức ảo của lớp cơ sở ở phía ngoài lớp dẫn xuất thì trong lời gọi phương thức phải có thêm tên đối tượng cần tác động).

+ Cách gọi phương thức ảo theo kiểu này được thực hiện trong kết buộc tĩnh.

qt2: + Nếu phương thức ảo được gọi kèm với đối tượng thuộc lớp nào thì phương thức của lớp đó sẽ được thực hiện.

+ Đây cũng là trường hợp phương thức được sử dụng theo kết buộc tĩnh.

qt3: + Nếu phương thức được gọi kèm với con trỏ đối tượng kiểu lớp cơ sở.

+ Nếu con trỏ đối tượng lớp cơ sở chứa địa chỉ hoặc trỏ đến đối tượng thuộc lớp nào thì phương thức của lớp đó sẽ được thực hiện.

+ Trong trường hợp này kết buộc sẽ được thiết lập theo hình thức kết buộc động.

- Điều đó có nghĩa là chỉ khi chạy chương trình, con trỏ đối tượng thuộc lớp cơ sở mới nhận được địa chỉ của đối tượng nhận thông điệp, khi đó CTBD và thông dịch mới xác định được phương thức thuộc lớp nào cùng đối tượng thuộc lớp nào sẽ được sử dụng

qt4: + Nếu trong trường hợp phương thức ảo cần sử dụng không có trong lớp dẫn xuất được xác định theo các quy tắc trên thì CTBD phải tự phán đoán để biết phương thức cần gọi thuộc lớp nào.

Cách phán đoán:

+ So sánh tuần tự phương thức cần gọi có trùng tên với phương thức của các lớp cơ sở theo trình tự: các lớp có quan hệ gần với lớp dẫn xuất xét trước, các lớp ở xa lớp dẫn xuất xét sau.

```
class DV
{
    protected:
        char *ten;
    public:
        DV( )
            {ten=NULL; }
        DV(char *ten1)
            {ten=strdup(ten1); }
        ~DV( )
            {delete ten; }
        virtual void xuat( )
            {
                cout<<"\n Day la loai dong vat
co ten la:"<<ten;
            }
};
class BC:public DV
{
```

```

        public:
        BC( ):DV( )
            {}
        BC(char *ten1):DV(ten1)
            {}
        virtual void xuat( )
            {
                DV::xuat( );
                cout<<"\n Loai bon chan";
            }
};
class CM:public BC
{
    public:
        CM( ):BC( )
            {}
        CM(char *ten1):BC(ten1)
            {}
        virtual void xuat( )
            {
                BC::xuat( );
                cout<<"\n La con meo";
            }
};

    void hien(DV *d)
        {
            d->xuat( );
        }

void main( )
{
    clrscr( );
    DV *d3, *d1, *d2=new DV("Dong vat khong ten");
    BC b1("Bon chan khong ten");
    CM m1("Win");
    d2->xuat( );
    d3=&m1; d3->xuat( );
    getch( ); cout<<"\n";
    char loai; loai='D';
    cout<<"Chon loai: Dv,Bc,Cm (D,B,M)?"";
    cin>>loai; loai=toupper(loai);
    DV *d;
    switch (loai)
        {
            case 'D':d=d2; break;
            case 'B':d=&b1; break;
            case 'M':d=&m1; break;
            default:cout<<"\n Vao sai loai";
        }
}

```

```

        getch( ); exit(0);
    }
    d->xuat( ); // hoac co the thay bang hien(d), la
               // ket buoc dong theo qt3
    getch( );
};

```

Khi biên dịch chương trình, với các dòng lệnh:

```
d2->xuat( );
```

phương thức `xuat()` của lớp DV kết buộc với đối tượng của lớp DV được chỉ bởi con trỏ đối tượng `d2` kiểu lớp DV. Kết buộc động này được xác định theo **qt3**.

```
d3=&m1; d3->xuat( );
```

phương thức `xuat()` của lớp CM kết buộc với đối tượng `m1` thuộc lớp CM theo **qt3**. Đây là kết buộc muện.

5. Quá trình gọi phương thức Virtual và các đặc điểm của nó

- CTBD đã đưa phương thức ảo vào làm việc bằng cách tạo ra mỗi một thành phần dữ liệu ẩn cho mỗi một hàm ảo của lớp.
- Những thành phần dữ liệu ẩn này là những con trỏ chỉ đến hàm ảo đúng ở lớp thực sự của đối tượng.
- Khi sử dụng phương thức ảo để xây dựng các chương trình ta cần chú ý đến các đặc điểm cơ bản sau của nó:
 - Không nhất thiết phải ghi rõ từ khoá `virtual` khi định nghĩa một phương thức ảo trong các lớp dẫn xuất.
 - Phương thức ảo của một lớp có thể được khai báo là phương thức bạn của lớp khác.
 - Các định nghĩa của các phương thức ảo trong lớp cơ sở và lớp dẫn xuất cần phải thống nhất. Có nghĩa là tất cả phương thức ảo phải được định nghĩa có cùng kiểu trả về, tên và danh sách đối.
 - Phương thức ảo cũng được thừa kế trong lớp dẫn xuất. Điều này có nghĩa là không nhất thiết phải định nghĩa lại một phương thức ảo được thừa kế từ lớp cơ sở.

6. Tính đa hình (Polymorphism)

- Trong tiếng Hy Lạp, “poly” có nghĩa là nhiều (many), morphism có nghĩa là hình thức (form)
- Đó là quá trình của việc định nghĩa một số đối tượng của những lớp khác nhau trong một nhóm và sử dụng những lời gọi hàm khác nhau để thực hiện những thao tác của những đối tượng đó
- Điều đó có nghĩa là thực hiện những bước quy trình khác nhau bởi những hàm có cùng những thông điệp.

Vd: Xét lớp `mammals`: Những `mammals` khác nhau sẽ có những sự đáp lại khác nhau đến hàm `eat()`

- Đối tượng sẽ thực hiện hoạt động thích hợp nhất đối với nó
- Những đối tượng là đa hình nếu chúng có một vài điểm tương đồng nhưng vẫn có một chút gì đó khác nhau

- Sự dễ dàng gọi một hàm thích hợp từ bất cứ một lớp nào trong những lớp đã cho, sử dụng con trỏ của lớp cơ sở là một hình thức của đa hình.

Vd: Sử dụng những virtual function

Hàm ảo thuần túy (Pure Virtual function) :

- Một vài lớp như lớp Shapes thể hiện những khái niệm trừu tượng cho những đối tượng không thể tồn tại
- Không thể cung cấp những khái niệm cụ thể cho những hàm ảo của nó mà thật sự sẽ tạo ra một đối tượng Shape
- Thay đổi bằng cách: Khai báo hàm ảo của lớp Shapes như là một hàm ảo thuần túy
- Một hàm ảo thuần túy chỉ có một khai báo hàm
 - Nó được khai báo bằng cách gán giá trị 0 cho hàm
`virtual void ten_ham() = 0;`
- Mỗi lớp dẫn xuất phải chứa một định nghĩa hàm cho mỗi hàm ảo thuần túy được thừa kế từ lớp cơ sở nếu nó phải được sử dụng để tạo một đối tượng
- Không thể tạo ra một đối tượng của lớp chứa một hoặc nhiều hàm ảo thuần túy bởi vì sẽ không có gì để trả lời khi một lời gọi hàm được gửi đến một phương thức ảo thuần túy .
- Vậy hàm ảo thuần túy là một hàm ảo mà nội dung của nó không có gì

7. Lớp cơ sở trừu tượng (Abstract classes)

- Một lớp chứa một hoặc nhiều hàm ảo thuần túy không thể sử dụng để định nghĩa một đối tượng_ gọi là lớp trừu tượng.
- Không có một đối tượng nào của một lớp trừu tượng có thể được tạo ra. Bởi vì nó chỉ được dùng để định nghĩa một số khái niệm tổng quát, chung cho các lớp khác.
- Một lớp trừu tượng chỉ có thể được sử dụng như một lớp cơ sở cho các lớp khác

```
class Shapes
{
public:
    virtual void draw() = 0;
    // Pure virtual function
    virtual void move(int) = 0;
    // Pure virtual function
};
```

```
class circle: public Shapes
{
private:
    int rad;
public:
    circle(int x);
```

```

        void draw();
        void move(int) { }
};

```

- Nếu một lớp thừa kế một lớp trừu tượng mà không cung cấp định nghĩa cho hàm ảo thuần túy thì nó cũng sẽ trở thành một lớp trừu tượng và không thể được sử dụng để định nghĩa một đối tượng
- Theo ý nghĩa về hướng đối tượng, ta vẫn có thể có một lớp trừu tượng mà không nhất thiết phải chứa những hàm ảo thuần túy.
- Cách dễ dàng để nhận biết một lớp trừu tượng là xem có dùng lớp đó để khai báo đối tượng hay không?_ Nếu không thì đó là lớp cơ sở trừu tượng.
- Sử dụng quan trọng của một lớp trừu tượng là để cung cấp một giao diện mà không phơi bày bất cứ một chi tiết bổ sung nào.

Hủy tử ảo (Virtual destructors)

- Hủy tử của lớp dẫn xuất không được gọi để giải phóng không gian nhớ được cấp phát bởi cấu tử của lớp dẫn xuất
- Bởi vì hủy tử là không ảo và thông điệp sẽ không đến được hủy tử dưới kết buộc động.
- Tốt hơn là nên có một hủy tử ảo để giải phóng không gian nhớ một cách hữu hiệu dưới kết buộc động.

```

class A
{
private:
    char *aptr;
public:
    A( ) // cấu tử không thể ảo
    {
        aptr= new char[9];
    }
    virtual void f( ); // hàm thành phần ảo
    virtual ~A( )      // hủy tử ảo
        {delete[ ] aptr;}
};

class B: public A
{
private:
    char *bptr;
public:
    B( )
        {bptr= new char[99]; }
    ~B( )
        {delete[ ] bptr; }
};

void main()
{
    A *ptr = new B;
}

```

```
delete ptr;
}
```

- Hàm ảo thừa nhận mã kết hợp với lớp của đối tượng hơn là với lớp của con trở hoặc tham chiếu
 - Khi chúng ta viết `delete ptr` và lớp cơ sở có một hủy tử ảo, thì hủy tử mà nhận lời gọi là một hàm kết hợp với kiểu của đối tượng `*ptr` hơn là một hàm kết hợp với kiểu của con trở
- Khi một thể hiện của lớp dẫn xuất chứa một thể hiện của lớp cơ sở thì việc gọi những hủy tử của cả hai lớp là cần thiết để chắc rằng tất cả những không gian trên heap đã được giải phóng.

8. Chiến lược sử dụng Đa hình (Polymorphism)

- Đa hình cho phép xét các vấn đề khác nhau, các đối tượng khác nhau, các phương pháp khác nhau, các cách giải quyết khác nhau theo cùng một lược đồ chung.
- Các bước áp dụng đa hình có thể tổng kết lại như sau:
 - Xây dựng lớp cơ sở trừu tượng bao gồm những thuộc tính chung nhất của các thực thể cần quản lý
 - + Đưa vào các phương thức Virtual dùng để xây dựng các nhóm phương thức Virtual cho các lớp dẫn xuất sau này
 - + Mỗi nhóm phương thức Virtual sẽ thực hiện một chức năng nào đó trên các lớp
 - Xây dựng các lớp dẫn xuất bắt đầu từ lớp cơ sở Virtual
 - + Các lớp dẫn xuất sẽ mô tả các đối tượng cụ thể cần quản lý
 - + Số mức dẫn xuất không hạn chế
 - Xây dựng các phương thức Virtual trong các lớp dẫn xuất
 - + Các phương thức này tạo thành các nhóm phương thức Virtual trong sơ đồ các lớp có quan hệ thừa kế
 - Xây dựng lớp quản lý các đối tượng
 - + Dữ liệu của lớp này là một dãy các con trở của lớp cơ sở trừu tượng ban đầu
 - + Các con trở này có thể chứa địa chỉ đối tượng của các lớp dẫn xuất. Do đó có thể sử dụng chúng để thực hiện các thao tác trên các đối tượng của bất kỳ lớp dẫn xuất nào.

Vd: Giả sử có 20 ô, mỗi ô có thể nuôi một con chó hoặc một con mèo.

- Xây dựng chương trình gồm các chức năng:
 - Nhập 1 con vật mới mua vào ô rỗng đầu tiên
 - Xuất 1 con vật
 - Thống kê các con vật đang nuôi trong 20 ô
- Tổ chức chương trình như sau:
 - Trước tiên định nghĩa lớp CONVAT là lớp cơ sở Virtual.
 - + Lớp này có một thuộc tính là tên con vật và một phương thức virtual dùng để xưng tên.

CONVAT

- + Hai lớp CONMEO và CONCHO được dẫn xuất từ lớp CONVAT
- + Xây dựng lớp DSCONVAT dùng để quản lý chung cả mèo và chó

- Lớp này có ba thuộc tính: số con vật max (chính bằng số ô), số con vật đang nuôi, và mảng con trở kiểu CONVAT
- Mỗi phần tử mảng sẽ chứa địa chỉ của một đối tượng kiểu CONMEO hoặc CONCHO.

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<ctype.h>
#include<string.h>
Class CONVAT
{
    protected:
        char *ten;
    public:
        CONVAT()
        {
            ten=NULL;
        }

        CONVAT(char *ten1)
        {
            ten=strdup(ten1);
        }
        virtual void xungten()
        {}
};
class CONMEO:public CONVAT
{
    public:
        CONMEO():CONVAT()
        {
        }

        CONMEO(char *ten1):CONVAT(ten1)
        {
        }
        virtual void xungten()
        {
            cout<<"\n toi la chu meo:"<<ten;
        }
};
class CONCHO:public CONVAT
```

```

    {
        public:
            CONCHO():CONVAT()
            {
            }
            CONCHO(char *ten1):CONVAT(ten1)
            {
            }
            virtual void xungten()
            {
                cout<<"\nToi la chu cho:"<<ten;
            }
    };
class DSCONVAT
    {
        private:
            int maxsoconvat;
            int soconvat;
            CONVAT **h;
        public:
            DSCONVAT(int max);
            ~DSCONVAT();
            int nhap(CONVAT *c);
            CONVAT *xuat(int n);
            void thongke();
    };
DSCONVAT::DSCONVAT(int max)
    {
        maxsoconvat=max;
        soconvat=0;
        h=new CONVAT*[max];
        for(int i=0; i<max; ++i)
            h[i]=NULL;
    }
DSCONVAT::~~DSCONVAT()
    {
        maxsoconvat=0;
        soconvat =0;
        delete h;
    }
int DSCONVAT::nhap(CONVAT *c)
    {
        if(soconvat==maxsoconvat)
            return 0;
        int i=0;
        while(h[i]!=NULL) ++i;
        h[i]=c;
    }

```

```

        soconvat++;
        return (i+1);
    }
CONVAT *DSCONVAT::xuat(int n)
{
    if(n<1||n>maxsoconvat)
        return NULL;
    --n;
    if(h[n])
    {
        CONVAT *c=h[n];
        h[n]=NULL;
        soconvat--;
        return c;
    }
    else
        return NULL;
}
void DSCONVAT::thongke()
{
    if(soconvat)
    {
        cout<<"\n";
        for(int i=0; i<maxsoconvat; ++i)
            if (h[i])
                h[i]->xungten();
    }
}

CONCHO c1("MUC");
CONCHO c2("VEN");
CONCHO c3("LAI");
CONCHO c4("XU");
CONCHO c5("BONG");
CONMEO m1("MUOP");
CONMEO m2("TRANG");
CONMEO m3("VANG");
CONMEO m4("TAMTHE");
CONMEO m5("DEN");
void main()
{
    DSCONVAT d(20);
    clrscr();
    d.nhap(&c1);
    int im2=d.nhap(&m2);
    d.nhap(&c3);
    d.nhap(&m1);
}

```

```

        int ic4=d.nhap(&c4);
        d.nhap(&c5);
        d.nhap(&m5);
        d.nhap(&c2);
        d.nhap(&m3);
        d.thongke();
        d.xuat(im2);
        d.xuat(ic4);
        d.thongke();
        getch();
    }

```

Chú ý :

- Theo quan điểm chung ta coi lớp CONVAT là lớp cơ sở trừu tượng.
- Nhưng theo quan điểm của C++ thì lớp này chưa phải là lớp cơ sở trừu tượng, vì trong nó chưa có các hàm ảo thuần túy.
- xungten là một hàm ảo, được định nghĩa đầy đủ. Do vậy khai báo:

CONVAT cv("Con vat bon chan"); vẫn được C++ chấp nhận

- Nếu định nghĩa lại xungten như sau:

virtual void xungten()=0; // hàm ảo thuần túy

- Khi đó C++ sẽ coi CONVAT là lớp trừu tượng. Vậy câu lệnh CONVAT cv("Con vat bon chan"); sẽ phát sinh lỗi:
Cannot create instance of abstract class 'CONVAT'

BÀI TẬP CHƯƠNG 4

- Bt1:** Viết chương trình xây dựng lớp HINH_TRON dẫn xuất từ lớp DIEM.
- + Định nghĩa lớp DIEM có: các thuộc tính x, y; hai hàm tạo; phương thức in() để in tọa độ
 - + Xây dựng lớp HINH_TRON dẫn xuất từ lớp DIEM thêm vào thuộc tính r; hai hàm tạo; phương thức getR() trả về bán kính của hình tròn
 - + Trong hàm main():
 - Khai báo đối tượng h kiểu HINH_TRON
 - Sử dụng phương thức in() đối với h (sự thừa kế) để xuất ra tọa độ tâm hình tròn
 - Sử dụng phương thức getR() đối với h.
 - + Nội dung chương trình chính là định nghĩa một hình tròn sau đó in ra bán kính và tọa độ tâm hình tròn vừa khai báo
- Bt2:** Xác định lỗi trong đoạn chương trình được viết theo sơ đồ sau:

```
#include<iostream.h>
#include<conio.h>
class window
{
    public:int w;
};
class border:public window
{
    public:int b;
};
class menu:public window
{
    public:int m;
};
class border_and_menu:public border, public menu
{
    public:int bm;
};
void main()
{
```

```
border_and_menu  bm1;
bm1.bm = 3;
bm1.m   = 3; bm1.b = 3;
bm1.w   = 3;
cout<<bm1.bm; getch();
};
```

Bt3: Chỉ ra lỗi, nếu có, trong đoạn chương trình sau

```
class Alpha
{
    public:
        virtual fn() {}
};
class Beta:public Alpha
{
    private:
        fn() {}
};
void main()
{
    Beta b;
    Alpha* aa = &b;
    Beta* cc = &b;
    aa->fn();
    cc->fn();
}
```

Bt4: Kiểm tra những lớp được cho dưới đây:

```
class Base
{
    public:
        virtual void fn1();
        virtual void fn2();
        void func();
};
class Derived:public Base
{
    public:
        void fn1();
        void fn2(int);
        void func();
};
```

Giả sử trong hàm main() ta có:

```
Derived d ;
Base* ptr = &d ;
```

hàm nào được gọi trong mỗi dòng lệnh sau:

```
ptr->fn1();
ptr->fn2();
ptr->func();
```

Chương 5 : INPUT/OUTPUT

I. DÒNG VÀO, DÒNG RA :

1. Các lớp stream :

- Có 4 lớp quan trọng :
 - Lớp cơ sở ios
 - 2 lớp istream và ostream dẫn xuất từ ios
 - Lớp iostream được dẫn xuất từ hai lớp istream và ostream

Ta có sơ đồ:

- iostream.h định nghĩa ostream class và istream class. Dòng output chuẩn sử dụng đối tượng “cout” của ostream class. Class này thực hiện overload toán tử (<<). istream class khai báo một đối tượng “cin”, thực hiện thao tác input. Nó tiến hành overload toán tử (>>) cho việc nhập liệu.

2. Đưa vào và lấy ra theo dòng vào/ra :

Dòng cin và toán tử nhập :

- Dòng cin là một đối tượng kiểu istream đã định nghĩa trong C++
- Là dòng vào chuẩn gắn với bàn phím
- Các thao tác nhập trên dòng cin đồng nghĩa với nhập dữ liệu từ bàn phím
- Với cin ta có thể sử dụng toán tử nhập >> và các phương thức nhập của các lớp ios và istream
- Cách dùng toán tử nhập để đọc dữ liệu từ dòng cin:
`cin >> thamsol >> thamsol2 >> ...>> thamsok;`
- Trong đó thamso có thể là: biến hoặc phần tử mảng (nguyên, thực, ký tự), hoặc có thể là con trỏ ký tự

Cách thức nhập :

- Bỏ qua các ký tự trắng đứng trước và sau đó đọc vào các ký tự tương ứng với kiểu yêu cầu

Vd:

```
int m;
float y;
cin>>m>>y;
```

Nếu gõ: <s><s>456<s><s>4.5<Enter>

Thì kết quả nhập là: m=456 ; y=4.5

Ký tự Enter vẫn còn lại trên dòng nhập

Dòng Cout và toán tử xuất :

- Dòng cout là một đối tượng kiểu ostream đã định nghĩa trong C++.
- Đó là dòng xuất chuẩn gắn với màn hình
- Với cout ta có thể sử dụng toán tử xuất << và các phương thức xuất của lớp ios và ostream.
- Cách dùng toán tử xuất :

```
cout << thamsol << thamsol2 <<...<<thamsok ;
```

Chú ý :

- Toán tử xuất được định nghĩa chồng với toán tử dịch trái và chúng cùng có mức độ ưu tiên
- Toán tử xuất có thứ tự ưu tiên lớn hơn các toán tử trong biểu thức điều kiện. vì vậy nếu ta thực hiện như sau :

```
int a=5, b=10 ;
cout<< "\nMax=" <<a>b ?a:b;
```

 thì trình biên dịch sẽ báo lỗi

Khắc phục: ta dùng dấu ngoặc tròn để bao biểu thức điều kiện lại

3. Định dạng và các bộ phận xử lý định dạng :

Nội dung định dạng giá trị xuất:

Nội dung định dạng là xác định các thông số :

- Độ rộng quy định
- Độ chính xác
- Ký tự độn
- Các thông số khác

+ **Độ rộng thực tế của giá trị xuất:** C++ sẽ biến đổi giá trị cần xuất thành một chuỗi ký tự rồi đưa chuỗi này ra màn hình. Ta gọi số ký tự của chuỗi này là độ rộng thực tế của giá trị xuất.

Vd:

```
int n=4567, m=-23;
float x=-3,1416;
char ht[]="Truong Tri Lam";
```

thì: độ rộng thực tế của n là 4, m là 3, x là 7, ht là 14.

+ **Độ rộng quy định :** là số vị trí tối thiểu trên màn hình dành để in giá trị. Theo mặc định, độ rộng quy định bằng 0. Ta có thể dùng phương thức `cout.width()` để thiết lập độ rộng này.

Vd : `cout.width(8) ;` sẽ thiết lập độ rộng quy định là 8.

+ Mối quan hệ giữa độ rộng thực tế và độ rộng quy định :

- Nếu độ rộng thực tế lớn hơn hoặc bằng độ rộng quy định thì số vị trí trên màn hình chứa giá trị xuất sẽ bằng độ rộng thực tế.
- Nếu độ rộng thực tế nhỏ hơn độ rộng quy định thì số vị trí trên màn hình chứa giá trị xuất sẽ bằng độ rộng quy định. Khi đó sẽ có một số vị trí dư thừa. Các vị trí dư thừa sẽ được độn bằng khoảng trống.

+ Xác định ký tự độn :

- Ký tự độn mặc định là dấu cách. Tuy nhiên có thể dùng phương thức `cout.fill()` để chọn một ký tự độn khác.

```
Vd: int n=123 ;           // Độ rộng thực tế là 3
      cout.fill('*') ;   // Ký tự độn là *
      cout.width(5) ;    // Độ rộng quy định là 5
      cout<<n ;
```

Thì kết quả in ra là : **123

+ Độ chính xác : là số vị trí dành cho phần phân (khi in số thực). Độ chính xác mặc định là 6. Tuy nhiên có thể dùng phương thức `cout.precision()` để chọn độ chính xác.

```
Vd: float x=34.455 ;     // Độ rộng thực tế là 6
      cout.precision(2) ; // Độ chính xác 2
      cout.width(7) ;     // Độ rộng quy ước 8
      cout.fill('0') ;    // Ký tự độn là số 0
      cout<<x;
```

thì kết quả in ra là : 0034.46

Các phương thức định dạng :

- `int cout.width()` : cho biết độ rộng quy định hiện tại
- `int cout.width(int n)` : thiết lập độ rộng quy định mới là n và trả về độ rộng quy định trước đó.
- `int cout.precision()` : cho biết độ chính xác hiện tại
- `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.
- `char cout.fill()` : cho biết ký tự độn hiện tại đang được áp dụng
- `char cout.fill(char ch)` : quy định ký tự độn mới sẽ được dùng là ch và cho biết ký tự độn đang được 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.

Cờ định dạng :

- Khái niệm chung về cờ:
 - Mỗi cờ chứa trong một bit. Cờ có hai trạng thái: On/Off tương ứng với 2 giá trị 1 và 0
 - Các cờ có thể chứa trong một biến kiểu long. Trong tệp <iostream.h> đã định nghĩa các cờ sau:

```
ios::left
ios::dec
ios::fixed
ios::uppercase
ios::right
ios::oct
ios::scientific
```

ios::showpoint

ios::internal

ios::hex

ios::showpos

ios::showbase

- Công dụng của các cờ: Có thể chia các cờ thành các nhóm:

- Nhóm 1 gồm các cờ định vị: ios::left, ios::right,

ios::internal

ios::left: Khi bật cờ này giá trị in ra nằm ở bên trái vùng quy định, các ký tự độn nằm sau:

- **Vd:** 35***

ios::right: bật cờ này giá trị in ra nằm bên phải vùng quy định, các ký tự độn nằm trước

- **Vd:** ** -89

ios::internal: tác dụng giống cờ ios::right chỉ khác là

dấu (nếu có) in đầu tiên,

- **Vd:** -**89

- Nhóm 2 gồm các cờ định dạng số nguyên : ios::dec,

ios::oct, ios::hex

ios::dec bật (mặc định) : Số nguyên được in dưới dạng cơ số 10

ios::oct bật : Số nguyên được in dưới dạng cơ số 8

ios::hex bật : Số nguyên được in dưới dạng cơ số 16

- Nhóm 3 gồm các cờ định dạng số thực : ios::fixed,

ios::scientific, ios::showpoint

- Mặc định: Cờ ios::fixed bật và cờ ios::showpoint tắt

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

- **Vd:** 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 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.

- **Vd:** 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 ios::showpoint tắt thì số thực in ra dưới dạng mũ. Số chữ số phần phân của phần định trị được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

- **Vd:** Nếu độ chính xác n=4 thì:

Số thực -87.1500 được in: -8.715e+01
 Số thực 23.45425 được in: 2.3454e+01
 Số thực 678.0 được in: 6.78e+02

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

- **Vd:** Nếu độ chính xác n=4 thì:
 - Số thực -87.1500 được in: -8.7150e+01
 - Số thực 23.45425 được in: 2.3454e+01
 - Số thực 678.0 được in: 6.7800e+01
- Nhóm 4 gồm các hiển thị: ios::showpos, ios::showbase, ios::uppercase
 - Nếu ios::showpos tắt (mặc định) thì dấu cộng không được in trước số dương.
 - Nếu ios::showpos bật thì dấu cộng được in trước số dương.
 - Nếu 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.
- **Vd:** Nếu a=40 thì:
 - Dạng in hệ 8 là: 050
 - Dạng in hệ 16 là 0x28
 - Nếu ios::showbase tắt thì không in 0 trước số nguyên hệ 8 và không in 0x trước số nguyên hệ 16.
- **Vd:** Nếu a=40 thì:
 - Dạng in hệ 8 là: 50
 - Dạng in hệ 16 là 28
 - Nếu ios::uppercase bật thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ hoa.
 - Nếu ios::uppercase tắt thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ thường.

Các bộ phận định dạng (định nghĩa trong <iostream.h>)

- dec // như cờ ios::dec
- oct // như cờ ios::oct
- hex // như cờ ios::hex
- endl // xuất ký tự '\n'
- flush// đẩy dữ liệu ra thiết bị xuất

Chúng có tác dụng như cờ định dạng nhưng được viết nổi đuôi trong toán tử xuất nên tiện sử dụng hơn.

Vd:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    cout.setf(ios::showbase)
```

```
cout<<"ABC"<<endl<<hex<<40<<" "<<41;
getch();
}
```

Chương trình sẽ in 2 dòng sau ra màn hình:

```
ABC
0x28 0x29
```

Các hàm định dạng (định nghĩa trong <iomanip.h>)

Các hàm định dạng gồm:

- setw(int n) // như cout.width(int n)
- setprecision(int n) // như cout.precision(int n)
- setfill(char ch) // như cout.fill(char ch)
- setiosflags(long l) // như cout.setf(long f)
- resetiosflags(long l) // như cout.unsetf(long f)

Các hàm định dạng có tác dụng như các phương thức định dạng nhưng được viết nối đuôi trong toán tử xuất nên tiện sử dụng hơn.

4. Một vài hàm liên quan đến dòng vào, ra :

ostream và istream functions

- Ngoài "cout" và "cin" chúng ta hãy xem xét một vài input/output functions khác cũng được định nghĩa trong istream và ostream classes.

* Ngoài trừ hàm toán tử chòng<< () , ostream classes còn cung cấp những functions **put()** và **write()** cho output.

- function put () chỉ in ra những ký tự (character). Nó có thể được gọi như sau:

```
cout.put('Y');
```

- function này cũng trả về một tham chiếu đến đối tượng cout và vì vậy có thể được sử dụng như sau:

```
cout.put('Y').put('E').put('S').put('\n');
```

- function write() in ra chuỗi (string) và có dạng như sau
ostream &write(char*, int);

+ Đối số đầu tiên cho địa chỉ của chuỗi ký tự được in, đối thứ hai chỉ định số lượng ký tự được in ra. Function này không nhận ký tự null. Nếu đối thứ hai nhiều hơn chiều dài của chuỗi, thì nó sẽ in ra thêm một vài ký tự không được định nghĩa.

- Một function **flush()** có thể được sử dụng để làm đầy bộ đệm output. Một function khác **endl()** in một dòng ký tự mới. Chúng có thể được sử dụng như sau:

```
cout<<"Input a newline"<<endl;
cout<<"Output to be flushed"<<flush;
```

- Những functions này có thể được sử dụng với toán tử () và đối tượng cout, bởi vì ostream class có một overloaded function cho chúng. Chúng có thể được sử dụng trong dạng sau :

```
flush(cout);
endl(cout);
```

- `istream` class cũng định nghĩa function `get()` và `getline()` cho input. Một vài sự thực thi trong C++ không hỗ trợ `get()` và `getline()`.

- `getline()` và overloaded function `get()` chấp nhận kiểu chuỗi. Khuôn mẫu function của chúng như sau:

```
istream &get(char* s, int size, char ch = '\n');
istream &getline(char* s, int size, char ch = '\n');
```

Sự khác nhau giữa `get()` và `getline()` là:

- `get()` để lại một dòng ký tự mới trong input stream, trong khi `getline()` rút ra và loại bỏ nó.
- Ta cũng có những functions khác trong `istream` để phục vụ cho việc input, như `read()`, `peek()`, `gcount()`, `putback()`, và `ignore()`.
- function `gcount()` thông báo số lượng ký tự được đọc bởi thao tác nhập cuối cùng.

Vd: Trong chương trình này chúng ta sẽ sử dụng một vài functions phục vụ cho input như `get()`, `getline()` và `read()`. Và một vài output functions như `write()` và `put()`.

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{
    clrscr();
    int i;
    char str[25]="PYRAMID";
    int len = strlen(str);
    for(i=1;i<=len;++i) //write
        cout.write(str,i)<<endl;
    for(i=0;i<=len;++i) //put
    {
        cout.put(str[i]);
        cout.put(' ');
    }
    char ch;
    cout<<"\nEnter a short sentence and press
ENTER>:";
    cin.get(ch); //get
    while(ch!='\n')
    {
        cout.put(ch);
        cin.get(ch);
    }
    char* next = new char[80];
    cout<<"\nEnter another string:";
    cin.getline(next,25,'\n');
```

```

    cout<<next;
    delete next;
    cout<<"\nEnter a word (at least 5 characters):";
    char* str2 = new char[7];
    cin.read(str2,6); //read
    cout<<"The word is = ";
    cout.write(str2,5);
    cout<<endl;
    delete str2;
}

```

Ta sẽ nhìn thấy dòng output sau đây và sẽ được nhắc nhở để enter một vài chuỗi như được cho dưới đây:

OUTPUT:

```

P
PY
PYR
PYRA
PYRAM
PYRAMI
PYRAMID
P Y R A M I D

```

Enter a short sentence and press <ENTER>: This is a test

This is a test

Enter another string: What should I do?

What should I do?

Enter a word (at least 5 characters): Finish

Finis

- Trong chương trình này chúng ta đã sử dụng một function `write()` để output một chuỗi. Đối số thứ hai của `write()` chỉ định số lượng của ký tự được in ra. Sử dụng vòng lặp `for` chúng ta sẽ chỉ định được số lượng của ký tự được in ra.
- Tiếp đến ta sử dụng function `put()` để in cũng chuỗi "PYRAMID"- mỗi lần một ký tự được chia cách bởi một khoảng trống. Function này chỉ in những ký tự.
- Chúng ta cũng đã sử dụng function `get()` để đọc một ký tự. Bằng việc sử dụng `cin.get(ch)` chúng ta có thể đọc mỗi lần một ký tự từ dòng input. Trong vòng lặp `while` chúng ta kiểm tra xem ký tự được đọc vào có phải là một newline character hay không. Nếu không, ký tự đó sẽ được hiển thị trên dòng output sử dụng `put()` và những ký tự khác được chấp nhận.
- Function tiếp theo được sử dụng là `getline()`. Nó được dùng để đọc một chuỗi. Đối số thứ hai của `getline()` chỉ định số ký tự tối đa được đọc (cộng thêm một cho null character). Đối số thứ ba là ký tự đầu cuối. Nếu nó được bỏ qua, function sẽ đọc lên đến số lượng ký tự tối đa được chỉ định, mà là 25 trong trường hợp

này, hoặc chỉ là một newline character- ký tự chuyển dòng, bất cứ cái nào đến trước. Bởi vì chúng ta cấp phát bộ nhớ cho chuỗi tiếp theo sử dụng toán tử new, nên delete được sử dụng để giải phóng bộ nhớ.

- Trong function `read()`, đối số thứ hai chỉ định số lượng ký tự được đọc từ dòng input và lưu trữ trong đối số thứ nhất. Ở đây ta đã sử dụng `read()` để chấp nhận 6 ký tự từ dòng input và lưu trữ chúng trong `str2`.
- Nếu đối số thứ hai trong function `write()` được cho nhiều hơn chiều dài của chuỗi, thì nó sẽ in ra một vài ký tự không được định nghĩa. Nếu đối số thứ hai nhỏ hơn chiều dài của chuỗi thì function sẽ xuất ra số lượng ký tự chỉ bằng như đã quy định trong đối số. Dòng cuối cùng của output chỉ cho ra Finis mặc dù chúng ta đã gõ vào nguyên từ Finish. Trong trường hợp này function `write()` chỉ định rằng chỉ 5 ký tự được in ra.

5. Xử lý lỗi :

- Khi làm việc với tệp thường phát sinh lỗi. Thế nên trong lớp ios của C++ có nhiều phương thức phát hiện lỗi
- `int eof();` // trả về giá trị khác 0 nếu con trỏ đã ở cuối tệp mà lại thực hiện một lệnh đọc dữ liệu, ngược lại trả về 0
- `int fail();` // trả về giá trị khác 0 nếu lần nhập, xuất cuối cùng có lỗi, ngược lại trả về 0.
- `int bad();` // trả về giá trị khác 0 khi một phép nhập, xuất không hợp lệ hoặc chưa phát hiện được khi có lỗi, ngược lại trả về 0
- `int good();` // trả về giá trị khác 0 nếu không có lỗi nào xảy ra, ngược lại thì bằng 0
- `void clear();` // dùng để tắt tất cả các bit lỗi

II. VÀO RA NHỊ PHÂN

1. Chọn kiểu nhập, xuất nhị phân :

- Kiểu nhập xuất mặc định là văn bản. Để chọn kiểu nhị phân thì trong tham số mode cần chứa giá trị: `ios::binary`

Vd: Để mở tệp “DSSV.DL” đọc ghi theo kiểu nhị phân và gắn tệp với dòng nhập/xuất fs, ta dùng câu lệnh sau:

```
fstream fs(“DSSV.DL”, ios::in | ios::out | ios::binary);
```

2. Đọc ghi ký tự :

- Để ghi một ký tự lên tệp có thể dùng phương thức:
`ostream & put(char);`
- Để đọc một ký tự từ tệp có thể dùng phương thức:
`istream & get(char &);`

3. Đọc ghi một dãy ký tự theo kiểu nhị phân :

- Phương thức:
`ostream & write(char *buf, int n);`

sẽ xuất n ký tự chứa trong buf ra dòng xuất

- Phương thức:

`istream & read(char *buf, int n);`

sẽ nhập n ký tự từ dòng nhập và chứa vào buf

- Phương thức:

`int gcount`

cho biết số ký tự thực sự đọc được trong phương thức read

Chương 6 : CÁC LỚP TIỆN DỤNG

I. KHÁI NIỆM CÁC LỚP TIỆN DỤNG :

- Các lớp tiện dụng được dùng rất nhiều trong OOP.
- Các lớp tiện dụng dùng để định nghĩa các đối tượng có khả năng chứa đựng các đối tượng khác, như các cây và các cây nhị phân
- Ít có một chương trình phức tạp nào mà không có ít nhất một cấu trúc cây hoặc xâu
- Vì C++ thiếu một bộ các các lớp lưu trữ chuẩn nên đưa thêm class tiện dụng vào sẽ rất hữu ích.

II. CÁC KHÁI NIỆM THIẾT KẾ CHUNG

Có những vấn đề nảy sinh khi xây dựng các CTDL tổng quát trong C++, đó là:

- Các lớp C++ không phải tất cả đều được dẫn xuất từ một lớp gốc đơn độc như với các ngôn ngữ như Smalltalk, do đó không thể có một phương tiện chung để gộp lại được tất cả các kiểu dữ liệu.
- Vấn đề này phát sinh khi cần xây dựng một class tiện dụng tổng quát, sao cho một chương trình stack hoặc queue có thể được hoán đổi cho nhau. Như vậy các lớp tiện dụng phải có tính đa hình (polymorphism). Điều này có nghĩa là các hàm virtual phải có một kiểu dữ liệu đồng nhất cho mọi lớp mà trong đó nó định nghĩa. Do đó, ta cần phải tìm một số phương thức có thể tham khảo các đối tượng của tất cả các kiểu theo cùng một lối chung.
- Giải pháp tốt nhất là dùng các **pointer void**.
- Một pointer void có thể chỉ tới mọi thứ, bất kể nó là kiểu bên trong hay là một đối tượng của một class nào đó. Do vậy các lớp tiện dụng đưa ra ở đây sẽ chỉ lưu trữ các pointer chỉ đến các đối tượng thực sự.
- Điều này giúp chúng ta không cần phải cấp phát không gian lưu trữ bản sao chép đối tượng và các thành phần dữ liệu, nhưng nó cũng lưu ý ta phải cẩn thận, không nên lưu trữ bất kỳ một pointer nào vào một tiện dụng mà vùng nhớ của pointer đó lại có thể bị huỷ trước khi bản thân tiện dụng này bị xoá
 - Ta không thể thực hiện việc kiểm tra kiểu của các đối tượng được gán vào hoặc lấy ra từ các pointer void.

Class Container

- Là lớp cơ sở dùng để tạo ra mọi lớp tiện dụng khác
- Có tệp tiêu đề là `CONTAINER.HPP` để chứa các định nghĩa và có nội dung sau:

```
#if !defined(_CONTAINER_HPP)
#define _CONTAINER_HPP1
Class container
{
```

```
protected:
    unsigned long count;// số phần tử trong thùng
                        // chứa
    void(*errhandler)();// pointer chỉ đến hàm xử
                        // lý ngoài lề

public:
    container();//cấu tử
    container(const container &c);//copy cấu tử
    void operator=(const container &c);// assignment
                        // oprator
    unsigned long getcount();// trả về số phần tử có
                        // trong vật chứa
    // định nghĩa hàm xử lý lỗi
    void seterrorhandler(void(*userhanler)());
    // bốn pure virtual phương thức
    virtual int store(void *item)=0;// cất1phần tử
                        // vào thùng
    virtual void examine()=0; // xem giá trị 1 phần
tử
    virtual void retrieve()=0;// lấy một phần tử ra
    virtual void empty()=0; // làm cho thùng rỗng;
};
#endif
```

Tệp tin Container.cpp, thể hiện phần cài đặt của class Container

```
// phan cai dat
// xu li loi mac dinh
#include <Container.hpp>
#include <iostream.h>
void defaulthandler();
void defaulthandler()
{
    cout<<"\n Container error.memory allocation
failure";
}
container::container()
{
    count=0; errhandler=defaulthandler;
}
container::container(const container &c)
{
    count=c.count; errhandler=c.errhandler;
}
// assignment operator
void container::operator=(const container &c)
{
    count=c.count; errhandler=c.errhandler;
```

```

    }
    // cho biet so phan tu trong thung
    unsigned long container::getcount()
    {
        return count;
    }
    // dinh nghia xu ly loi
    void container::seterrorhandler(void
    (*userhandler)())
    {
        errhandler=userhandler;
    }

```

- Class Container định nghĩa hai thành phần dữ liệu :
 - Biến đếm số phần tử đang được cất trong Container
 - Một pointer chỉ tới một hàm được gọi khi xảy ra lỗi trong Container
- Cấu tử cho Container không có tham số. Chỉ gán 0 vào phần tử dữ liệu count
- Assignment operator: để chép các giá trị của count và Errhandler từ obj nguồn vào obj đích
- Class Container không cần có huỷ tử vì nó không cần phải dọn dẹp gì
- Mục đích của cấu tử và assignment operator cho Container:
 - Cung cấp một cách giao tiếp phục vụ yêu cầu chung cho tất cả các lớp tiện dụng.

Vd:

- Mọi obj của các class tiện dụng đều phải gán 0 cho thành phần dữ liệu count, thừa hưởng từ Container.
 - Thay vì mỗi class lưu trữ đều phải thực hiện cùng một phép gán như vậy trong các cấu tử của chúng, tốt hơn là ta thực hiện điều đó trong cấu tử của lớp cơ sở
- Trong class Container có bốn pure virtual phương thức:
 - Store: cất tham số pointer của nó vào trong Container
 - Examine và Retrieve: đều trả về pointer của phần tử hiện thời trong một Container. Chúng chỉ khác nhau ở chỗ Retrieve sẽ xoá mọi sự tham khảo đến phần tử đó ra khỏi danh sách.
 - Empty: loại bỏ tất cả các pointer phần tử ra khỏi một Container
- Những phương thức này là pure virtual vì chúng còn phụ thuộc vào kiểu tiện dụng mà trên đó chúng được sử dụng.
 - Store của Stack làm việc rất khác với Store của Queue.
- Container được định nghĩa như một abstract class để chứa các khả năng chung cho một nhóm các lớp. Vì nó không cài đặt bốn phương thức nên nó định nghĩa chúng như các pure virtual phương thức.

- Getcount và seterrorhandler dùng để trả về giá trị của biến count và gán một pointer cho một function mà nó sẽ được dùng để thông báo lỗi.

- Nếu seterrorhandler không hề được gọi, default function thông báo lỗi sẽ được dùng.

III. DANH SÁCH KẾT NỐI (SINGLY LINKED-LIST)

- Singly linked-list là một trong những khối được xây dựng rất căn bản trong các chương trình
- Singly linked-list class (s_list) là lớp cơ sở cho các xâu đơn khác và nó dẫn xuất từ class Container(abstract class)
- Singly linked-list class là một abstract class có tệp tin tiêu đề

S_list.hpp:

```
#if !defined(_S_LIST_HPP)
#define _S_LIST_HPP1
#include<Container.hpp>
class s_list: virtual public container
{
protected:
    // cấu trúc một nút trong xâu
    struct listnode
    {
        void *dataptr;
        listnode *next;
    };
    listnode *head; // pointer chỉ đến nút đầu
    listnode *tail; // pointer chỉ đến nút cuối
private:
    void copy(const s_list &s1); // phương thức sao
chép
public:
    s_list(); // cấu tử
    s_list(const s_list &s1); // copy cấu tử
    ~s_list(); // huỷ tử
    void operator=(const s_list &s1);
    // 4 virtual phương thức
    virtual int store(void *item)=0; // cat mot phan
// tu vao thung
    virtual void *examine()=0; // xem gia tri mot phan
// tu
    virtual void *retrieve()=0; // lay ra mot phan tu
    virtual void empty(); // lam cho thung rong
};
#endif
```

- Cấu trúc Listnode định nghĩa dạng của một mắt xích trong một xâu.

- Mặc dù `Listnode` được định nghĩa bên trong định nghĩa `s-list`, nhưng thực ra nó là một cấu trúc có phạm vi toàn cục.
- Bất kỳ một định nghĩa kiểu nào nằm trong định nghĩa của các class (unions hoặc struct) cũng đều dùng chung được, bất kể chúng được định nghĩa dưới chỉ định truy xuất nào.
- `Head` và `tail` là các pointer chỉ tới các thành phần đầu và cuối của xâu
- Function `Copy` sao chép xâu các pointer chỉ tới `listnode` từ một đối tượng `s_list` này sang một đối tượng khác. Nó sẽ được dùng chung cho cả copy cấu tử lẫn assignment operator. Điều này giúp chương trình đơn giản hơn
- Vì ta không muốn copy được gọi trực tiếp ở bên ngoài phạm vi class nên ta cho nó được bảo vệ bằng cách chỉ định kiểu truy xuất là `private`.

Cài đặt class `S_list` :

- Cài đặt các phương thức của class `s_list` được thông qua tệp tin `S_list.cpp` có nội dung sau:

```
#include<S_list.hpp>
void s_list::copy(const s_list &s1)//phương thức sao
chép
{ head=NULL, tail=NULL;
  listnode *temp=s1.head;

  while (temp!=NULL)
  {
    if(head==NULL)
    {
      head=new listnode;
      if(head==NULL) errhandler();
      tail=head;
    }
    else
    {
      tail->next=new listnode;
      if (tail->next==NULL) errhandler();
      tail=tail->next;
    }
    tail->dataptr=temp->dataptr;
    tail->next=NULL;
    temp=temp->next;
  }
}
s_list::s_list():container() // cấu tử
{
  head=NULL; tail=NULL;
}
```

```

s_list::s_list(const s_list &s1):container(s1)
// copy cấu tử
{ copy(s1);
}
s_list::~~s_list()// huỷ tử
{
    this->empty();
}
void s_list::operator=(const s_list &s1) //assignment
operator
{
    this->empty();
    count=s1.count;
    copy(s1);
}
void s_list::empty()
{
    listnode *q,*p;
    p=head;
    head=NULL; tail=NULL;
    while (p!=NULL)
    {
        q=p; p=p->next;
        delete q;
    }
}

```

- S_list cũng là một abstract lớp cơ sở, nhưng nó vẫn cung cấp những bản cài đặt cho các phương thức tiện ích có thể được dùng cho các class dẫn xuất từ nó.

IV. NGĂN XẾP (STACK) VÀ HÀNG ĐỢI (QUEUE)

- Stack và queue là hai dạng tương tự với Linked-list.
- Trong Stack: các phần tử được cất theo nguyên tắc: Last in first out
- Trong Queue: ngược lại các phần tử được cất theo: First in first out
- Sự khác nhau giữa Stack và Queue: cách cất các phần tử
 - Stack: các phần tử mới được cất vào đầu xâu
 - Queue: các phần tử mới được cất vào cuối xâu

Stack

- Được xây dựng như một lớp dẫn xuất từ s_list class, có tệp tin tiêu đề stack.hpp được viết như sau:

```

#if !defined(_STACK_HPP)
#define _STACK_HPP1
#include <S_list.hpp>
class stack: public s_list
{

```

```

public:
    stack(); // cấu tử
    stack(const stack &st); // copy cấu tử
void operator=(const stack &st); // assignment operator
virtual int store(void *item); // cất một pt vào thùng
virtual void *examine(); // xem giá trị một phần tử
virtual void *retrieve(); // lấy ra một pt
};
#endif

```

- class stack không định nghĩa một thành phần dữ liệu nào, mọi thành phần cần thiết đều được thừa hưởng từ s_list class_ class này được thừa hưởng từ class Container.
- Việc cài đặt các function và phương thức của class Stack được thể hiện trong tệp gốc Stack.cpp

```

#include<stack.hpp>
stack::stack():s_list() // cấu tử
{
}
stack::stack(const stack &st):s_list(st)
// copy cấu tử
{
}
void stack::operator=(const stack &st)
// assignment operator
{
    this->s_list::operator=(st);
}
int stack::store(void *item) // cất một phần tử vào
//thùng
{
    // đưa vào đầu danh sách
    listnode *p;
    p=new listnode;
    if(p==NULL) return 1;
    count++;
    p->dataptr=item; p->next=head;
    head=p; return 0;
}
// xem giá trị một phần tử
void *stack::examine()
{if (count==0) return NULL;
else
    return head->dataptr;
}
// lấy một phần tử ra và hủy bỏ phần tử trên đỉnh
// ngăn xếp
void *stack::retrieve()

```

```

{
    if(count==0) return NULL;
    else
    {
        listnode *p; void *value;
        value=head->dataptr;
        p=head;
        head=p->next;
        delete p;
        count--;
        return value;
    }
}

```

- Ta nhận thấy các cấu tử của lớp Stack cũng rất đơn giản, chúng chỉ đơn thuần gửi các thông số tới cho các hàm tạo của class S_list.
- Toán tử gán của Stack cho ta thấy cách gọi một phương thức toán tử được kế thừa sẽ như thế nào. Đây cũng là một ví dụ mà ở đó ta nhận thấy vai trò quan trọng của con trỏ this.
- Store dùng để đặt phần tử mới vào đầu của xâu
- Retrieve và Examine để trả lại phần tử pointer đang giữ ở đầu của xâu.

Queue

- Sự khác nhau giữa Stack và Queue: cách lưu trữ các phần tử, nên class queue có thể được dẫn xuất từ stack
- Có tệp tiêu đề queue .hpp có nội dung:

```

#ifndef _Queue_HPP
#define _Queue_HPP
#include <Stack.hpp>
class queue: public stack
{
public:
    queue(); // cấu tử
    queue(const queue &q); // copy cấu tử

    void operator=(const queue &q); //assignment
    // operator
    virtual int store(void *item); //cất pt vào thùng
};
#endif

```

- Queue class chỉ định nghĩa những thứ cần thiết: cấu tử, assignment operator và phương thức Store của riêng nó để đặt các phần tử mới ở cuối xâu.
- Ta biết các lớp dẫn xuấtes không được thừa hưởng các cấu tử, assignment operator trong lớp cơ sởes

- Do đó Queue phải định nghĩa riêng cho nó các phương thức này.

Chúng được cài đặt trong Queue.cpp:

```
#include<queue.hpp>
queue::queue():stack() // cấu tử
{
}
queue::queue(const queue &q):stack(q)// copy
// cấu tử
{
}
void queue::operator=(const queue &q) //assignment
// operator
{
    this->stack::operator=(q); // dùng stack
// assignment operator
}
int queue::store(void *item)//cat 1phan tu vao thung
{
    // dat vao cuoi xau
    listnode *q;
    q=new listnode;
    if (q==NULL) return 1;
// bo sung
    q->next=NULL; q->dataptr=item;
    if (count==0)
    {
        head=q; tail=q;
    }
    else
    {
        tail->next=q;
        tail=q;
    }
    count++; return 0;
}
```

V. DANH SÁCH KẾT NỐI KÉP (DOUBLE LINKED-LIST)

- Trong một singly linked-list, mỗi phần tử của danh sách giữ một con trỏ (*next) chỉ tới phần tử đi sau nó.
- Trong một Double linked-list, mỗi phần tử trong danh sách sẽ giữ thêm một con trỏ (*prev) chỉ đến phần tử đi trước nó.
- Một singly linked-list xử lý nhanh hơn một Double linked-list, nhưng những Double linked-list thì có thể được dùng cho những công việc phức tạp hơn
- Sau đây là định nghĩa một Double linked-list:

```
#if !defined(_DBLLIST_HPP)
#define _DBLLIST_HPP1
```

```
#include<Container.hpp>
Class DoublyLinkedList:public Container
{
    protected:
        // cau truc mot nut
        struct DListNode
        {
            void *DataPtr;
            DListNode *Prev;
            DListNode *Next;
        };
        // pointer chi den nut dau va nut cuoi cua xau
        DListNode *Head;
        DListNode *Tail;
        // phuong thuc copy
        void Copy(const DoublyLinkedList &d1);
    public:
        DoublyLinkedList(); // cấu tử
        DoublyLinkedList(const DoublyLinkedList &d1);
                            // copy cấu tử
        void operator=(const DoublyLinkedList &d1);
                            // assignment operator
        ~DoublyLinkedList(); // huỷ tử
        virtual int Store(void *item)=0; // lưu một phần tử
        virtual void *Examine()=0; // xem một phần tử
        virtual void *retrieve()=0; // lấy ra một phần tử
        virtual void Empty(); // hủy bỏ tất cả các phần tử của danh sách
};
#endif
```

- Kiểu của phương thức và thành phần dữ liệu được định nghĩa trong Double linked-list tương ứng với những định nghĩa đã có trong S_list:
- Giống như S_list, nó cũng là một lớp trừu tượng.
- Nó sẽ cung cấp một cơ sở cho các lớp Double linked-list khác.
- Nó được cài đặt như sau:

```
#include<Dbllist.hpp>
#include<stddef.h>
void DoublyLinkedList::Copy(const DoublyLinkedList
&d1)
{
    Head=NULL;
    Tail=NULL;
    DListNode *temp=d1.Head;
```

```

while (temp!=NULL)
{
    if(Tail==NULL)
    {
        Tail=new DListNode;
        if (Tail==NULL) ErrHandler();
        Head=Tail;
        Tail->Next=NULL;
        Tail->Prev=NULL;
        Tail->DataPtr=temp->DataPtr;
    }
    else
    {
        Tail->Next=new DListNode;
        if(Tail->Next==NULL) ErrHandler();
        Tail->Next->Next=NULL;
        Tail->Prev->Prev=Tail;
        Tail->Next->DataPtr=temp->DataPtr;
        Tail=tail->Next;
    }
    temp=temp->Next;
}

DoublyLinkedList::DoublyLinkedList():Container()
// cấu tử
    {Head=NULL; Tail=NULL; }
DoublyLinkedList::DoublyLinkedList(const
DoublyLinkedList &d1) // copy cấu tử
    {Copy(d1); }
void DoublyLinkedList::operator=(const
DoublyLinkedList &d1) // assignment operator
    {
        this->Empty();
        count=d1.count;
        Copy(d1);
    }
DoublyLinkedList::~~DoublyLinkedList()//huỷ tử
    {this->Empty(); }
// huy bo tat ca cac phan tu cua xau
void DoublyLinkedList::Empty()
    {
        DListNode *temp, *hold;
        temp=head;
        while (temp!=NULL)
        {
            hold=temp->Next;

```

```

        delete temp;
        temp=hold;
    }
}

```

VI. LỚP WORKLIST

- Được dẫn xuất từ DoublyLinkedList class
- Là kiểu Linked_list rất linh hoạt
- Thường được dùng trong xây dựng các chương trình OO.
- Định nghĩa WorkList:

```

#if !defined(_WORKLIST_HPP)
#define _WORKLIST_HPP1
#include <Dbllist.hpp>
class WorkList:public DoublyLinkedList
{
protected:
    DlistNode *Current;// chi nut hien tai cua danh
                        // sach
public:
    WorkList();// cấu tử
    WorkList(const WorkList &w1);//copy cấu tử
    void operator=(const WorkList &w1);//assignment
                        //operator

    virtual int Store(void *item);// luu mot phan tu
    virtual void *Examine();// xem mot phan tu
    virtual void *Retrieve();// doc va huy bo mot
                        // phan tu
    virtual int Delete(void *item);// xoa mot phan
                        // tu khoi ds
    virtual void GotoHead();// dua ve dau ds
    virtual void Goto Tail();// dua ve cuoi ds
    virtual void GoNext();// di den phan tu ke
    virtual void GoPrev();
};
#endif

```

- Điều do class WorkList minh hoạ tốt nhất ở đây là khả năng mà một lớp dẫn xuất có thể bổ sung cho một lớp cơ sở.
- WorkList đưa ra một phần tử dữ liệu và năm phương thức_ thêm vào những gì mà nó đã thừa hưởng từ DoublyLinkedList
- Khác với những kiểu list khác đã nêu ra, bất kỳ pointer nào cất trong WorkList cũng đều có thể chọn ra để xem xét.
- Phương pháp cài đặt các phương thức của WorkList như sau:

```

#include<WorkList.hpp>
#include<stddef.h>
WorkList::WorkList():DoublyLinkedList()// cấu tử
{

```

```

        current=NULL;
    }
WorkList::WorkList(const WorkList &w1):
DoublyLinkedList(w1)// copy cấu tử
    {
    }
void WorkList::operator=(const WorkList &w1)//
assignment operator
    {
        DoublyLinkedList::operator=(w1);
        current=head;
    }
int WorkList::Store(void *item)// lưu 1 phần tử
    {
        DListNode *new_item;
        new_item=new DListNode;
        if (item==NULL) return 1;
        new_item->Prev=NULL;
        new_item->Next=NULL;
        new_item->DataPtr=item;
        if (Head=NULL)
            {
                Head=new_item;
                Tail=new_item;
                Current=new_item;
            }
        else
            {
                Head->Prev=new_item;
                new_item->Next=Head;
                Head=new_item;
            }
        ++Count;
        return 0;
    }
void *WorkList::Examine()// xem 1 phần tử
    {
        if (Current==NULL)
            return NULL;
        else
            return Current->DataPtr;
    }
void *WorkList::Retrieve()// đọc & hủy bỏ 1 phần tử
    {
        void *value=Examine();
        Delete(value);
        return value;
    }

```

```

    }
int WorkList::Delete(void *item)// xoa 1 phan tu khoi
xau
    {
        DListNode *temp;
        temp=head;
while (temp!=NULL)
    {
        if (temp->DataPtr==item)
            {
                if(temp==Current)
                    Current=temp->Next;
                if (temp->Prev==NULL)
                    {
                        Head=temp->Next;
                    if (Head!=NULL)
                        Head->Prev=NULL;
                    if (temp->Next==NULL)
                        Tail=NULL;
                    }
                else
                    {
                        temp->Prev->Next=temp->Next;
                        if (temp->Next==NULL)
                            Tail=temp->Prev;
                        else
                            temp->Next->Prev=temp->Prev;
                    }
                delete temp;
                ~Count;
                return 0;
            }
        temp=temp->Next;
    }
return 1;
}
void WorkList::GotoHead() // dua ve dau xau
    {
        Current=Head;
    }
void WorkList::Gototail() // dua ve cuoi xau
    {
        Current=tail;
    }
void WorkList::GoNext() // den phan tu ke tiep
    {
        if (Current!=NULL)

```

```

        Current=Current->Next;
    }
void WorkList::GoPrev() // den phan tu truoc do
{
    if (Current!=NULL)
        Current=Current->Prev;
}

```

- Các pointer chỉ đến các phần tử được cất ở cuối xâu bằng Store
- Delete tìm trong các pointer được lưu trữ một pointer nào đó khớp với pointer được gửi tới ở tham số. Nếu tìm thấy phần tử tương ứng pointer đó và nút của nó sẽ bị huỷ khỏi xâu.
- Pointer Current chỉ đến phần tử hiện hành đang được chọn.
- Examine và Retrieve trả về giá trị của phần tử pointer được cất trong nút được chỉ bởi Current.
- Retrieve xoá đi tất cả những liên hệ đến phần tử đang được chọn hiện thời ra khỏi WorkList.

MỤC LỤC

Chương 1 :	1
GIỚI THIỆU LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	1
I. LỊCH SỬ PHÁT TRIỂN CỦA LẬP TRÌNH :	1
1. Lập trình tuyến tính :	1
2. Lập trình có cấu trúc :	1
3. Sự trừu tượng hoá chức năng :	1
4. Lập trình hướng đối tượng (Object Oriented Programming):	1
II. NHỮNG THUẬT NGỮ CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG :	2
1. Lớp (class), đối tượng (object), phương thức (method) :	2
2. Lớp cơ sở (base class), lớp dẫn xuất (derived class) :	3
3. Tính kế thừa (Inheritance) :	3
4. Tính đa hình (Polymorphism) :	4
Chương 2 :	5
GIỚI THIỆU VỀ NHỮNG ĐIỂM MỚI CỦA C++	5
I. C++ LÀ GÌ ?	5
II. SỰ KHÁC NHAU GIỮA C VÀ C++	5
1. Ép kiểu :	5
2. Ghi chú :	5
3. Khai báo :	5
4. Hằng	5
5. Toán tử phạm vi ::	5
6. Vào ra trong C++	6
a. In dữ liệu ra màn hình :	6
b. Nhập dữ liệu từ bàn phím :	6
7. Định dạng khi in ra màn hình :	6
8. Kiểu liệt kê :	6
9. Cấp phát và giải phóng bộ nhớ :	7
10. Đối kiểu tham chiếu :	7
11. Đối có giá trị mặc định :	8
12. Hàm Inline :	8
13. Hàm chồng (Function Overloading) :	9
14. Chồng toán tử (Operator Overloading) :	11
BÀI TẬP CHƯƠNG 2	13
Chương 3: CÁC LỚP (Classes)	14
I. ĐỊNH NGHĨA LỚP (CLASS) :	14
1. Khái niệm về class :	14
2. Định nghĩa lớp (class) :	14
3. Đối tượng (Object) :	14
4. Thuộc tính (Property) :	15
5. Phương thức (Method) :	15
6. Con trỏ this và đối tượng không tường minh (object implicit) :	19
7. Toán tử phạm vi (Scope resolution operator) :	21
8. Phạm vi lớp (class scope) :	21
9. Encapsulation:	22
10. Từ khoá xác định thuộc tính truy xuất (access specifier): PRIVATE: & PUBLIC :	23
11. Hàm bạn (Friend Function) :	24
12. Lớp bạn (Friend classes) :	27
13. Phương thức toán tử :	28
14. Phương thức kiểu inline.....	29

II. CẤU TỬ (CONSTRUCTOR), HUỖ TỬ (DESTRUCTOR) VÀ CÁC VẤN ĐỀ LIÊN QUAN	31
1. Cấu tử (constructor) :.....	31
2. Cấu tử sao chép (copy constructor) :.....	32
3. Cấu tử sao chép mặc định (default copy constructor) :.....	34
4. Cấu tử sao chép tường minh :.....	35
5. Huỷ tử (Destructor)	36
6. Cấu tử, huỷ tử mặc định (default constructor, destructor) :.....	37
7. Toán tử gán (Assignment operator)	40
8. Hàm chuyển đổi (Conversion Functions).....	45
9. Cấu tử và đối tượng thành phần.....	47
III. ĐỐI TƯỢNG VÀ HÀM	52
1. Static member :.....	52
2. Cấp phát bộ nhớ động (dynamic memory)	59
3. Cấp phát mảng :.....	60
4. Biến, mảng đối tượng.....	60
5. Con trỏ đối tượng.....	62
6. Đối tượng hằng (Const object).....	64
BÀI TẬP CHƯƠNG 3	67
<i>Chương 4</i> :.....	68
SỰ KẾ THỪA (Inheritance)	68
I. KHÁI NIỆM :.....	68
II. KẾ THỪA ĐƠN (SINGLE INHERITANCE)	68
1. Kế thừa đơn :.....	68
2. Lớp cơ sở và lớp dẫn xuất :.....	69
3. Tham khảo và truy xuất những thành phần của lớp cơ sở :.....	70
4. Sự quan hệ giữa các thành phần trùng tên trong lớp cơ sở và lớp dẫn xuất	73
5. Chỉ định truy nhập Protected	75
6. Cấu tử và huỷ tử của lớp dẫn xuất	78
III. ĐA THỪA KẾ (MULTIPLE INHERITANCE)	89
1. Đa thừa kế (Multiple Inheritance) :.....	89
2. Xây dựng lớp có nhiều lớp cơ sở.....	90
3. Thừa kế nhiều mức, nhiều lớp và sự trùng tên.....	91
4. Đa thừa kế : Lớp cơ sở chung (Common base class).....	97
5. Cấu tử và huỷ tử trong Đa thừa kế.....	97
6. Lớp cơ sở ảo (Virtual Base class).....	102
IV. ĐA HÌNH (POLYMORPHISM)	103
1. Thừa kế trong lớp dẫn xuất và kết buộc tĩnh (static binding).....	103
2. Kết buộc tĩnh (Static binding).....	107
3. Hàm ảo (Virtual Function).....	109
4. Phương thức ảo (Virtual) và kết buộc động (dynamic binding)	111
5. Quá trình gọi phương thức Virtual và các đặc điểm của nó	117
6. Tính đa hình (Polymorphism).....	117
7. Lớp cơ sở trừu tượng (Abstract classes)	118
8. Chiến lược sử dụng Đa hình (Polymorphism)	120
BÀI TẬP CHƯƠNG 4	125
<i>Chương 5</i> :.....	127
INPUT/OUTPUT	127
I. DÒNG VÀO, DÒNG RA :.....	127
1. Các lớp stream :.....	127
2. Đưa vào và lấy ra theo dòng vào/ra :.....	127
3. Định dạng và các bộ phận xử lý định dạng :.....	128

4.	Một vài hàm liên quan đến dòng vào, ra :	132
5.	Xử lý lỗi :	135
II.	VÀO RA NHỊ PHÂN	135
1.	Chọn kiểu nhập, xuất nhị phân :	135
2.	Đọc ghi ký tự :	135
3.	Đọc ghi một dãy ký tự theo kiểu nhị phân :	135
Chương 6 :		137
CÁC LỚP TIỆN DỤNG		137
I.	KHÁI NIỆM CÁC LỚP TIỆN DỤNG :	137
II.	CÁC KHÁI NIỆM THIẾT KẾ CHUNG	137
III.	DANH SÁCH KẾT NỐI (SINGLY LINKED-LIST)	140
IV.	NGĂN XẾP (STACK) VÀ HÀNG ĐỢI (QUEUE)	142
V.	DANH SÁCH KẾT NỐI KÉP (DOUBLE LINKED-LIST)	145
VI.	LỚP WORKLIST	148