

Giới thiệu

Tin học là một ngành khoa học mũi nhọn phát triển hết sức nhanh chóng trong vài chục năm lại đây và ngày càng mở rộng lĩnh vực nghiên cứu, ứng dụng trong mọi mặt của đời sống xã hội.

Ngôn ngữ lập trình là một loại công cụ giúp con người thể hiện các vấn đề của thực tế lên máy tính một cách hữu hiệu. Với sự phát triển của tin học, các ngôn ngữ lập trình cũng dần tiến hoá để đáp ứng các thách thức mới của thực tế.

Khoảng cuối những năm 1960 đầu 1970 xuất hiện nhu cầu cần có các ngôn ngữ bậc cao để hỗ trợ cho những nhà tin học trong việc xây dựng các phần mềm hệ thống, hệ điều hành. Ngôn ngữ C ra đời từ đó, nó đã được phát triển tại phòng thí nghiệm Bell. Đến năm 1978, giáo trình " Ngôn ngữ lập trình C " do chính các tác giả của ngôn ngữ là Dennis Ritchie và B.W. Kernighan viết, đã được xuất bản và phổ biến rộng rãi.

C là ngôn ngữ lập trình vạn năng. Ngoài việc C được dùng để viết hệ điều hành UNIX, người ta nhanh chóng nhận ra sức mạnh của C trong việc xử lý cho các vấn đề hiện đại của tin học. C không gắn với bất kỳ một hệ điều hành hay máy nào, và mặc dầu nó đã được gọi là " ngôn ngữ lập trình hệ thống" vì nó được dùng cho việc viết hệ điều hành, nó cũng tiện lợi cho cả việc viết các chương trình xử lý số, xử lý văn bản và cơ sở dữ liệu.

Và bây giờ chúng ta đi tìm hiểu thế giới của ngôn ngữ C từ những khái niệm ban đầu cơ bản nhất.

Hà nội tháng 11 năm 1997

Nguy?n H?u Tu?n

Chương 1

các khái niệm cơ bản

1.1. Tập ký tự dùng trong ngôn ngữ C :

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

26 chữ cái hoa : A B C .. Z

26 chữ cái thường : a b c .. z

10 chữ số : 0 1 2 .. 9

Các ký hiệu toán học : + - * / = ()

Ký tự gạch nối : _

Các ký tự khác : . , ; [] { } ! \ & % # \$...

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

Chú ý :

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

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

1.2. Từ khoá :

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

asm	break	case	cdecl
char	const	continue	default
do	double	else	enum
extern	far	float	for
goto	huge	if	int
interrupt	long	near	pascal

register	return	short	signed
sizeof	static	struct	switch
typedef	union	unsigned	void
volatile	while		

ý nghĩa và cách sử dụng của mỗi từ khoá sẽ được đề cập sau này, ở đây ta cần chú ý :

- Không được dùng các từ khoá để đặt tên cho các hằng, biến, mảng, hàm ...
- Từ khoá phải được viết bằng chữ thường, ví dụ : viết từ khoá khai báo kiểu nguyên là int chứ không phải là INT.

1.3. Tên :

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

Tên được đặt theo qui tắc sau :

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

Ví dụ :

Các tên đúng :

a_1 delta x1 _step GAMA

Các tên sai :

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

Chú ý :

Trong TURBO C, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. trong C, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho

hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

1.4. Kiểu dữ liệu :

Trong C sử dụng các các kiểu dữ liệu sau :

1.4.1. Kiểu ký tự (char) :

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

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

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

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

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

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

Khi đó thực chất :

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

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

Phân loại ký tự :

Có thể chia 256 ký tự làm ba nhóm :

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

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

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

1.4.2. Kiểu nguyên :

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

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

Chú ý :

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

1.4.3. Kiểu dấu phẩy động :

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

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

Giải thích :

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

1.5. Định nghĩa kiểu bằng TYPEDEF :

1.5.1. Công dụng :

Từ khoá typedef dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ được dùng để khai báo dữ liệu sau này. Nên chọn tên kiểu ngắn và gọn để dễ nhớ. Chỉ cần thêm từ khoá typedef vào trước một khai báo ta sẽ nhận được một tên kiểu dữ liệu và có thể dùng tên này để khai báo các biến, mảng, cấu trúc, vv...

1.5.2. Cách viết :

Viết từ khoá typedef, sau đó kiểu dữ liệu (một trong các kiểu trên), rồi đến tên của kiểu.
Ví dụ câu lệnh :

```
typedef int nguyen;
```

sẽ đặt tên một kiểu int là nguyen. Sau này ta có thể dùng kiểu nguyen để khai báo các biến, các mảng int như ví dụ sau ;

```
nguyen x,y,a[10],b[20][30];
```

Tương tự cho các câu lệnh :

```
typedef float mt50[50];
```

Đặt tên một kiểu mảng thực một chiều có 50 phần tử tên là mt50.

```
typedef int m_20_30[20][30];
```

Đặt tên một kiểu mảng thực hai chiều có 20x30 phần tử tên là m_20_30.

Sau này ta sẽ dùng các kiểu trên khai báo :

```
mt50 a,b;
```

```
m_20_30 x,y;
```

1.6. Hằng :

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán.

1.6.1. Tên hằng :

Nguyên tắc đặt tên hằng ta đã xem xét trong mục 1.3.

Để đặt tên một hằng, ta dùng dòng lệnh sau :

```
#define tên_hằng giá_trị
```

Ví dụ :

```
#define MAX 1000
```

Lúc này, tất cả các tên MAX trong chương trình xuất hiện sau này đều được thay bằng 1000. Vì vậy, ta thường gọi MAX là tên hằng, nó biểu diễn số 1000.

Một ví dụ khác :

```
#define pi 3.141593
```

Đặt tên cho một hằng float là pi có giá trị là 3.141593.

1.6.2. Các loại hằng :

1.6.2.1. Hằng int :

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

Ví dụ :

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

Chú ý :

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

1.6.2.2. Hằng long :

Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647.

Hằng long được viết theo cách :

```
1234L hoặc 1234l
```

(thêm L hoặc l vào đuôi)

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

Ví dụ :

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

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

1.6.2.3. Hằng int hệ 8 :

Hằng int hệ 8 được viết theo cách `0c1c2c3...` ở đây ci là một số nguyên dương trong khoảng từ 1 đến 7. Hằng int hệ 8 luôn luôn nhận giá trị dương.

Ví dụ :

`#define h8 0345` Định nghĩa hằng int hệ 8 có giá trị là
 $3*8*8+4*8+5=229$

1.6.2.4. Hằng int hệ 16 :

Trong hệ này ta sử dụng 16 ký tự : 0,1,..,9,A,B,C,D,E,F.

Cách viết	Giá trị
a hoặc A	10
b hoặc B	11
c hoặc C	12
d hoặc D	13
e hoặc E	14
f hoặc F	15

Hằng số hệ 16 có dạng `0xc1c2c3...` hoặc `0Xc1c2c3...` ở đây ci là một số trong hệ 16.

Ví dụ :

```
#define h16 0xa5
#define h16 0xA5
#define h16 0Xa5
#define h16 0XA5
```

Cho ta các hằng số h16 trong hệ 16 có giá trị như nhau. Giá trị của chúng trong hệ 10 là :

$$10*16+5=165.$$

1.6.2.5. Hằng ký tự :

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

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

'9'-'0'=57-48=9

Ví dụ :

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

Hằng ký tự còn có thể được viết theo cách sau :

'\c1c2c3'

trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

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

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

Chú ý :

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

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

1.6.2.5. Hằng xâu ký tự :

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

Ví dụ :

#define xau1 "Ha noi"

```
#define xau2 "My name is Giang"
```

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

Chú ý :

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

1.7. Biến :

Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng. Việc khai báo biến được thực hiện theo mẫu sau :

Kiểu dữ liệu của biến tên biến ;

Ví dụ :

int a,b,c;	Khai báo ba biến int là a,b,c
long dai,mn;	Khai báo hai biến long là dai và mn
char kt1,kt2;	Khai báo hai biến ký tự là kt1 và kt2
float x,y	Khai báo hai biến float là x và y
double canh1, canh2;	Khai báo hai biến double là canh1 và canh2

Biến kiểu int chỉ nhận được các giá trị kiểu int. Các biến khác cũng có ý nghĩa tương tự. Các biến kiểu char chỉ chứa được một ký tự. Để lưu trữ được một xâu ký tự cần sử dụng một mảng kiểu char.

Vị trí của khai báo biến :

Các khai báo cần phải được đặt ngay sau dấu { đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác. Sau đây là một ví dụ về khai báo biến sai :

(Khái niệm về hàm và cấu trúc chương trình sẽ nghiên cứu sau này)

```
main()  
{
```

```

int a,b,c;
a=2;
int d; /* Vị trí của khai báo sai */
.....
}

```

Khởi đầu cho biến :

Nếu trong khai báo ngay sau tên biến ta đặt dấu = và một giá trị nào đó thì đây chính là cách vừa khai báo vừa khởi đầu cho biến.

Ví dụ :

```

int a,b=20,c,d=40;
float e=-55.2,x=27.23,y,z,t=18.98;

```

Việc khởi đầu và việc khai báo biến rồi gán giá trị cho nó sau này là hoàn toàn tương đương.

Lấy địa chỉ của biến :

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ của biến sẽ được sử dụng trong một số hàm ta sẽ nghiên cứu sau này (ví dụ như hàm scanf).

Để lấy địa chỉ của một biến ta sử dụng phép toán :

& tên biến

1.8 Mảng :

Mỗi biến chỉ có thể biểu diễn một giá trị. Để biểu diễn một dãy số hay một bảng số ta có thể dùng nhiều biến nhưng cách này không thuận lợi. Trong trường hợp này ta có khái niệm về mảng. Khái niệm về mảng trong ngôn ngữ C cũng giống như khái niệm về ma trận trong đại số tuyến tính.

Mảng có thể được hiểu là một tập hợp nhiều phần tử có cùng một kiểu giá trị và chung một tên. Mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu biến thì có bấy nhiêu kiểu mảng. Mảng cần được khai báo để định rõ :

Loại mảng : int, float, double...

Tên mảng.

Số chiều và kích thước mỗi chiều.

Khái niệm về kiểu mảng và tên mảng cũng giống như khái niệm về kiểu biến và tên biến. Ta sẽ giải thích khái niệm về số chiều và kích thước mỗi chiều thông qua các ví dụ cụ thể dưới đây.

Các khai báo :

```
int a[10],b[4][2];
```

```
float x[5],y[3][3];
```

sẽ xác định 4 mảng và ý nghĩa của chúng như sau :

Thứ tự	Tên mảng	Kiểu mảng	Số chiều	Kích thước	Các phần tử
1	A	Int	1	10	a[0],a[1],a[2]...a[9]
2	B	Int	2	4x2	b[0][0], b[0][1] b[1][0], b[1][1] b[2][0], b[2][1] b[3][0], b[3][1]
3	X	Float	1	5	x[0],x[1],x[2]...x[4]
4	Y	Float	2	3x3	y[0][0], y[0][1], y[0][2] y[1][0], y[1][1], y[1][2] y[2][0], y[2][1], y[2][2]

Chú ý :

Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác, các phần tử của mảng có địa chỉ liên tiếp nhau.

Trong bộ nhớ, các phần tử của mảng hai chiều được sắp xếp theo hàng.

Chỉ số mảng :

Một phần tử cụ thể của mảng được xác định nhờ các chỉ số của nó. Chỉ số của mảng phải có giá trị int không vượt quá kích thước tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử z,b,x,y đã được khai báo như trên, và giả sử i,j là các biến nguyên trong đó i=2, j=1. Khi đó :

```
a[j+i-1]    là    a[2]
b[j+i][2-i] là    b[3][0]
y[i][j]     là    y[2][1]
```

Chú ý :

Mảng có bao nhiêu chiều thì ta phải viết nó có bấy nhiêu chỉ số. Vì thế nếu ta viết như sau sẽ là sai : `y[i]` (Vì `y` là mảng 2 chiều) vv..

Biểu thức dùng làm chỉ số có thể thực. Khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng.

Ví dụ :

`a[2.5]` là `a[2]`

`b[1.9]` là `a[1]`

* Khi chỉ số vượt ra ngoài kích thước mảng, máy sẽ vẫn không báo lỗi, nhưng nó sẽ truy cập đến một vùng nhớ bên ngoài mảng và có thể làm rối loạn chương trình.

Lấy địa chỉ một phần tử của mảng :

Có một vài hạn chế trên các mảng hai chiều. Chẳng hạn có thể lấy địa chỉ của các phần tử của mảng một chiều, nhưng nói chung không cho phép lấy địa chỉ của phần tử của mảng hai chiều. Như vậy máy sẽ chấp nhận phép tính : `&a[i]` nhưng không chấp nhận phép tính `&y[i][j]`.

Địa chỉ đầu của một mảng :

Tên mảng biểu thị địa chỉ đầu của mảng. Như vậy ta có thể dùng `a` thay cho `&a[0]`.

Khởi đầu cho biến mảng :

Các biến mảng khai báo bên trong thân của một hàm (kể cả hàm `main()`) gọi là biến mảng cục bộ.

Muốn khởi đầu cho một mảng cục bộ ta sử dụng toán tử gán trong thân hàm.

Các biến mảng khai báo bên ngoài thân của một hàm gọi là biến mảng ngoài.

Để khởi đầu cho biến mảng ngoài ta áp dụng các qui tắc sau :

Các biến mảng ngoài có thể khởi đầu (một lần) vào lúc dịch chương trình bằng cách sử dụng các biểu thức hằng. Nếu không được khởi đầu máy sẽ gán cho chúng giá trị 0.

Ví dụ :

....

```
float y[6]={3.2,0,5.1,23,0,42};
```

```
int z[3][2]={  
            {25,31},  
            {12,13},  
            {45,15}  
            }
```

```
....
```

```
main()
```

```
{
```

```
....
```

```
}
```

Khi khởi đầu mảng ngoài có thể không cần chỉ ra kích thước (số phần tử) của nó. Khi đó, máy sẽ dành cho mảng một khoảng nhớ đủ để thu nhận danh sách giá trị khởi đầu.

Ví dụ :

```
....
```

```
float a[]={0,5.1,23,0,42};
```

```
int m[][3]={  
            {25,31,4},  
            {12,13,89},  
            {45,15,22}  
            };
```

Khi chỉ ra kích thước của mảng, thì kích thước này cần không nhỏ hơn kích thước của bộ khởi đầu.

Ví dụ :

```
....
```

```
float m[6]={0,5.1,23,0};
```

```
int z[6][3]={  
            {25,31,3},  
            {12,13,22},  
            {45,15,11}  
            };
```

....

Đối với mảng hai chiều, có thể khởi đầu với số giá trị khởi đầu của mỗi hàng có thể khác nhau :

Ví dụ :

....

```
float z[][3]={
    {31.5},
    {12,13},
    {-45.76}
};
int z[13][2]={
    {31.11},
    {12},
    {45.14,15.09}
};
```

Khởi đầu của một mảng char có thể là

Một danh sách các hằng ký tự.

Một hằng xâu ký tự.

Ví dụ :

```
char ten[]={ 'h','a','g' }
char ho[]='tran'
char dem[10] ="van"
```

Chương 2

Các lệnh vào ra

Chương này giới thiệu thư viện vào/ra chuẩn là một tập các hàm được thiết kế để cung cấp hệ thống vào/ra chuẩn cho các chương trình C. Chúng ta sẽ không mô tả toàn bộ thư viện vào ra ở đây mà chỉ quan tâm nhiều hơn đến việc nêu ra những điều cơ bản nhất để viết chương trình C tương tác với môi trường và hệ điều hành.

2.1. Thâm nhập vào thư viện chuẩn :

Mỗi tệp gốc có tham trở tới hàm thư viện chuẩn đều phải chứa dòng :

```
#include <conio.h> cho các hàm getch(), putch(), clrscr(), gotoxy() ...
```

```
#include <stdio.h> cho các hàm khác như gets(), fflush(), fwrite(), scanf()...
```

ở gần chỗ bắt đầu chương trình. Tệp stdio.h định nghĩa các macro và biến cùng các hàm dùng trong thư viện vào/ra. Dùng dấu ngoặc < và > thay cho các dấu nháy thông thường để chỉ thị cho trình biên dịch tìm kiếm tệp trong danh mục chứa thông tin tiêu đề chuẩn.

2.2. Các hàm vào ra chuẩn - getchar() và putchar() - getch() và putch() :

2.2.1. Hàm getchar () :

Cơ chế vào đơn giản nhất là đọc từng ký tự từ thiết bị vào chuẩn, nói chung là bàn phím và màn hình của người sử dụng, bằng hàm getchar().

Cách dùng :

Dùng câu lệnh sau :

```
biến = getchar();
```

Công dụng :

Nhận một ký tự vào từ bàn phím và không đưa ra màn hình. Hàm sẽ trả về ký tự nhận được và lưu vào biến.

Ví dụ :

```
int c;  
c = getchar()
```


2.2.2. Hàm putchar () :

Để đưa một ký tự ra thiết bị ra chuẩn, nói chung là màn hình, ta sử dụng hàm putchar()

Cách dùng :

Dùng câu lệnh sau :

```
putchar(ch);
```

Công dụng :

Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị với màu trắng.

Ví dụ :

```
int c;  
c = getchar();  
putchar(c);
```

2.2.3. Hàm getch() :

Hàm nhận một ký tự từ bộ đệm bàn phím, không cho hiện lên màn hình.

Cách dùng :

Dùng câu lệnh sau :

```
getch();
```

Công dụng :

Nếu có sẵn ký tự trong bộ đệm bàn phím thì hàm sẽ nhận một ký tự trong đó.

Nếu bộ đệm rỗng, máy sẽ tạm dừng. Khi gõ một ký tự thì hàm nhận ngay ký tự đó (không cần bấm thêm phím Enter như trong các hàm nhập khác). Ký tự vừa gõ không hiện lên màn hình.

Nếu dùng :

```
biến=getch();
```

Thì biến sẽ chứa ký tự đọc vào.

Ví dụ :

```
c = getch();
```

2..2.4. Hàm putchar() :

Cách dùng :

Dùng câu lệnh sau :

```
putchar(ch);
```

Công dụng :

Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị theo màu xác định trong hàm textcolor.

Hàm cũng trả về ký tự được hiển thị.

2.3. Đưa kết quả lên màn hình - hàm printf :

Cách dùng :

```
printf(điều khiển, đối số 1, đối số 2, ...);
```

Hàm printf chuyển, tạo khuôn dạng và in các đối của nó ra thiết bị ra chuẩn dưới sự điều khiển của xâu *điều khiển*. Xâu *điều khiển* chứa hai kiểu đối tượng : các ký tự thông thường, chúng sẽ được đưa ra trực tiếp thiết bị ra, và các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng và in đối tiếp sau của printf.

Chuỗi điều khiển có thể có các ký tự điều khiển :

\n sang dòng mới

\f sang trang mới

\b lùi lại một bước

\t dấu tab

Dạng tổng quát của đặc tả :

```
%[-][fw][.pp]ký tự chuyển dạng
```

Mỗi đặc tả chuyển dạng đều được đưa vào bằng ký tự % và kết thúc bởi một ký tự chuyển dạng. Giữa % và ký tự chuyển dạng có thể có :

Dấu trừ :

Khi không có dấu trừ thì kết quả ra được dồn về bên phải nếu độ dài thực tế của kết quả ra nhỏ hơn độ rộng tối thiểu fw dành cho nó. Các vị trí dư thừa sẽ được lấp đầy bằng các khoảng trống. Riêng đối với các trường số, nếu dãy số fw bắt đầu bằng số 0 thì các vị trí dư thừa bên trái sẽ được lấp đầy bằng các số 0.

Khi có dấu trừ thì kết quả được dồn về bên trái và các vị trí dư thừa về bên phải (nếu có) luôn được lấp đầy bằng các khoảng trống.

fw :

Khi fw lớn hơn độ dài thực tế của kết quả ra thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số 0 và nội dung của kết quả ra sẽ được đẩy về bên phải hoặc bên trái.

Khi không có fw hoặc fw nhỏ hơn hay bằng độ dài thực tế của kết quả ra thì độ rộng trên thiết bị ra dành cho kết quả sẽ bằng chính độ dài của nó.

Tại vị trí của fw ta có thể đặt dấu *, khi đó fw được xác định bởi giá trị nguyên của đối tượng ứng.

Ví dụ :

Kết quả ra	fw	Dấu -	Kết quả đưa ra
-2503	8	có	-2503
-2503	08	có	-2503
-2503	8	không	-2503
-2503	08	không	000-2503
"abcdef"	8	không	abcdef
"abcdef"	08	có	abcdef
"abcdef"	08	không	abcdef

pp :

Tham số pp chỉ được sử dụng khi đối tượng ứng là một chuỗi ký tự hoặc một giá trị kiểu float hay double.

Trong trường hợp đối tượng ứng có giá trị kiểu float hay double thì pp là độ chính xác của trường ra. Nói một cách cụ thể hơn giá trị in ra sẽ có pp chữ số sau số thập phân.

Khi vắng mặt pp thì độ chính xác sẽ được xem là 6.

Khi đối là xâu ký tự :

Nếu pp nhỏ hơn độ dài của xâu thì chỉ pp ký tự đầu tiên của xâu được in ra. Nếu không có pp hoặc nếu pp lớn hơn hay bằng độ dài của xâu thì cả xâu ký tự sẽ được in ra.

Ví dụ :

Kết quả ra	fw	pp	Dấu -	Kết quả đưa ra	Độ dài trường ra
-435.645	10	2	có	-435.65	7
-435.645	10	0	có	-436	4
-435.645	8	vắng	có	-435.645000	11
"alphabeta"	8	3	vắng	alp	3
"alphabeta"	vắng	vắng	vắng	alphabeta	9
"alpha"	8	6	có	alpha	5

Các ký tự chuyển dạng và ý nghĩa của nó :

Ký tự chuyển dạng là một hoặc một dãy ký hiệu xác định quy tắc chuyển dạng và dạng in ra của đối tương ứng. Như vậy sẽ có tình trạng cùng một số sẽ được in ra theo các dạng khác nhau. Cần phải sử dụng các ký tự chuyển dạng theo đúng qui tắc định sẵn. Bảng sau cho các thông tin về các ký tự chuyển dạng.

Ký tự chuyển dạng	ý nghĩa
d	Đối được chuyển sang số nguyên hệ thập phân
o	Đối được chuyển sang hệ tám không dấu (không có số 0 đứng trước)
x	Đối được chuyển sang hệ mười sáu không dấu (không có 0x đứng trước)
u	Đối được chuyển sang hệ thập phân không dấu
c	Đối được coi là một ký tự riêng biệt
s	Đối là xâu ký tự, các ký tự trong xâu được in cho tới khi gặp ký tự không hoặc cho tới khi đủ số lượng ký tự được xác định bởi các đặc tả về độ chính xác pp.
e	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m.n.nE[+ hoặc -] với độ dài của xâu chứa n là pp.
f	Đối được xem là float hoặc double và được chuyển sang dạng thập

phân có dạng [-]m..m.n..n với độ dài của xâu chứa n là pp. Độ chính xác mặc định là 6. Lưu ý rằng độ chính xác không xác định ra số các chữ số có nghĩa phải in theo khuôn dạng f.

g Dùng %e hoặc %f, tùy theo loại nào ngắn hơn, không in các số 0 vô nghĩa.

Chú ý :

Mọi dãy ký tự không bắt đầu bằng % hoặc không kết thúc bằng ký tự chuyển dạng đều được xem là ký tự hiển thị.

Để hiển thị các ký tự đặc biệt :

Cách viết	Hiển thị
\'	'
\"	"
\\	\

Các ví dụ :

```
1 printf("\" Nang suat tang : %d % \" \n\\d\"",30,-50);      "Nang suat tang ; 30 %"
                                \d=-50
2 n=8
float x=25.5, y=-47.335
printf("\n%f\n%*.2f",x,n,y);      25.500000
                                -47.34
Lệnh này tương đương với
printf("\n%f\n%8.2f",x,n,y);
Vì n=8 tương ứng với vị trí *
```

2.4. Vào số liệu từ bàn phím - hàm scanf :

Hàm scanf là hàm đọc thông tin từ thiết bị vào chuẩn (bàn phím), chuyển dịch chúng (thành số nguyên, số thực, ký tự vv..) rồi lưu trữ nó vào bộ nhớ theo các địa chỉ xác định.

Cách dùng :

```
scanf(điều khiển,đôi 1, đôi 2, ...);
```

Xâu *điều khiển* chứa các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng biến tiếp sau của scanf.

Đặc tả có thể viết một cách tổng quát như sau :

`%[*][d...d]` ký tự chuyển dạng

Việc có mặt của dấu * nói lên rằng trường vào vẫn được dò đọc bình thường, nhưng giá trị của nó bị bỏ qua (không được lưu vào bộ nhớ). Như vậy đặc tả chứa dấu * sẽ không có đối tượng tương ứng.

`d...d` là một dãy số xác định chiều dài cực đại của trường vào, ý nghĩa của nó được giải thích như sau :

Nếu tham số `d...d` vắng mặt hoặc nếu giá trị của nó lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ trường vào sẽ được đọc, nội dung của nó được dịch và được gán cho địa chỉ tương ứng (nếu không có dấu *).

Nếu giá trị của `d...d` nhỏ hơn độ dài của trường vào thì chỉ phần đầu của trường có kích cỡ bằng `d...d` được đọc và gán cho địa chỉ của biến tương ứng. Phần còn lại của trường sẽ được xem xét bởi các đặc tả và đối tượng tương ứng tiếp theo.

Ví dụ :

```
int a;  
float x,y;  
char ch[6],ct[6]  
scanf("%f%5f%3d%3s%s",&x&y&a&ch&ct0);
```

Với dòng vào : 54.32e-1 25 12452348a

Kết quả là lệnh scanf sẽ gán

5.432 cho x

25.0 cho y

124 cho a

xâu "523" và dấu kết thúc \0 cho ch

xâu "48a" và dấu kết thúc \0 cho ct

Ký tự chuyển dạng :

Ký tự chuyển dạng xác định cách thức dò đọc các ký tự trên dòng vào cũng như cách chuyển dịch thông tin đọc được trước khi gán nó cho các địa chỉ tương ứng.

Cách dò đọc thứ nhất là đọc theo trường vào, khi đó các khoảng trắng bị bỏ qua. Cách này áp dụng cho hầu hết các trường hợp.

Cách dò đọc thứ hai là đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét bình đẳng như các ký tự khác. Phương pháp này chỉ xảy ra khi ta sử dụng một trong ba ký tự chuyển dạng sau : C, [dãy ký tự], [^dãy ký tự]

Các ký tự chuyển dạng và ý nghĩa của nó :

- c Vào một ký tự, đối tượng ứng là con trỏ ký tự. Có xét ký tự khoảng trắng
- d Vào một giá trị kiểu int, đối tượng ứng là con trỏ kiểu int. Trường phải vào là số nguyên
- ld Vào một giá trị kiểu long, đối tượng ứng là con trỏ kiểu long. Trường phải vào là số nguyên
- o Vào một giá trị kiểu int hệ 8, đối tượng ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 8
- lo Vào một giá trị kiểu long hệ 8, đối tượng ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 8
- x Vào một giá trị kiểu int hệ 16, đối tượng ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 16
- lx Vào một giá trị kiểu long hệ 16, đối tượng ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 16
- f hay e Vào một giá trị kiểu float, đối tượng ứng là con trỏ float, trường vào phải là số dấu phẩy động
- lf hay le Vào một giá trị kiểu double, đối tượng ứng là con trỏ double, trường vào phải là số dấu phẩy động
- s Vào một giá trị kiểu char, đối tượng ứng là con trỏ kiểu char, trường vào phải là dãy ký tự bất kỳ không chứa các dấu cách và các dấu xuống dòng

[Dãy ký tự], [^Dãy ký tự] Các ký tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một ký tự không thuộc tập các ký tự đặt trong[]. Đối tượng ứng là con trỏ kiểu char. Trường vào là dãy ký tự bất kỳ (khoảng trắng được xem như một ký tự).

Ví dụ :

```
int a,b;  
char ch[10], ck[10];  
scanf("%d%[0123456789]%^[0123456789]%3d",&a,ch,ck,&b);
```

Với dòng vào :

```
35 13145 xyz 584235
```

Sẽ gán :

```
35 cho a
```

```
xâu "13145" cho ch
```

```
xâu "xyz" cho ck
```

```
584 cho b
```

Chú ý :

Xét đoạn chương trình dùng để nhập (từ bàn phím) ba giá trị nguyên rồi gán cho ba biến a,b,c như sau :

```
int a,b,c;  
scanf("%d%d%d",&a,&b,&c);
```

Để vào số liệu ta có thể thao tác theo nhiều cách khác nhau:

Cách 1 :

Đưa ba số vào cùng một dòng, các số phân cách nhau bằng dấu cách hoặc dấu tab.

Cách 2 :

Đưa ba số vào ba dòng khác nhau.

Cách 3 :

Hai số đầu cùng một dòng (cách nhau bởi dấu cách hoặc tab), số thứ ba trên dòng tiếp theo.

Cách 4 :

Số thứ nhất trên một dòng, hai số sau cùng một dòng tiếp theo (cách nhau bởi dấu cách hoặc tab), số thứ ba trên dòng tiếp theo.

Khi vào sai sẽ báo lỗi và nhảy về chương trình chứa lời gọi nó.

2.5. Đưa kết quả ra máy in :

Để đưa kết quả ra máy in ta dùng hàm chuẩn fprintf có dạng sau :

```
fprintf(stdprn, điều khiển, biến 1, biến 2,...);
```

Tham số stdprn xác định thiết bị đưa ra là máy in.

Điều khiển có dạng đặc tả như lệnh printf.

Dùng giống như lệnh printf, chỉ khác là in ra máy in.

Ví dụ :

Đoạn chương trình in ma trận A, cỡ 8x6. Mỗi hàng của ma trận được in trên một dòng :

```
float a[8][6];
int i,j;
fprintf(stdprn, "\n%20c MA TRAN A\n\n", ' ');
for (i=0;i<8;++i)
    { for (j=0;j<6;++j)
        fprintf(stdprn, "%10.2f", a[i][j]);
        fprintf(stdprn, "\n");
    }
```

Chương 3

Biểu thức

Toán hạng có thể xem là một đại lượng có một giá trị nào đó. Toán hạng bao gồm hằng, biến, phần tử mảng và hàm.

Biểu thức lập nên từ các toán hạng và các phép tính để tạo nên những giá trị mới. Biểu thức dùng để diễn đạt một công thức, một qui trình tính toán, vì vậy nó là một thành phần không thể thiếu trong chương trình.

3.1. Biểu thức :

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

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

Biểu thức gán.

Biểu thức điều kiện .

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

Biểu thức thường được dùng trong :

Vế phải của câu lệnh gán.

Làm tham số thực sự của hàm.

Làm chỉ số.

Trong các toán tử của các cấu trúc điều khiển.

Tới đây, ta đã có hai khái niệm chính tạo nên biểu thức đó là toán hạng và phép toán. Toán hạng gồm : hằng, biến, phần tử mảng và hàm trước đây ta đã xét. Dưới đây ta sẽ nói đến các phép toán. Hàm sẽ được đề cập trong chương 6.

3.2. Lệnh gán và biểu thức:

Biểu thức gán là biểu thức có dạng :

$$v=e$$

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

$$v=e;$$

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác.

Ví dụ như khi ta viết

$$a=b=5;$$

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

Hoàn toàn tương tự như :

$$a=b=c=d=6; \text{ gán } 6 \text{ cho cả } a, b, c \text{ và } d$$

Ví dụ :

$$z=(y=2)*(x=6); \{ \text{ ở đây } * \text{ là phép toán nhân } \}$$

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

3.3. Các phép toán số học :

Các phép toán hai ngôi số học là

Phép toán	ý nghĩa	Ví dụ
+	Phép cộng	$a+b$
-	Phép trừ	$a-b$
*	Phép nhân	$a*b$
/	Phép chia	a/b
		(Chia số nguyên sẽ chặt phần thập phân)
%	Phép lấy phần dư	$a\%b$

(Cho phần dư của phép chia a cho b)

Có phép toán một ngôi - ví dụ $-(a+b)$ sẽ đảo giá trị của phép cộng $(a+b)$.

Ví dụ :

$$11/3=3$$

$$11\%3=2$$

$$-(2+6)=-8$$

Các phép toán $+$ và $-$ có cùng thứ tự ưu tiên, có thứ tự ưu tiên nhỏ hơn các phép $*$, $/$, $\%$ và cả ba phép này lại có thứ tự ưu tiên nhỏ hơn phép trừ một ngôi.

Các phép toán số học được thực hiện từ trái sang phải. Số ưu tiên và khả năng kết hợp của phép toán được chỉ ra trong một mục sau này

3.4. Các phép toán quan hệ và logic :

Phép toán quan hệ và logic cho ta giá trị đúng (1) hoặc giá trị sai (0). Nói cách khác, khi các điều kiện nêu ra là đúng thì ta nhận được giá trị 1, trái lại ta nhận giá trị 0.

Các phép toán quan hệ là :

Phép toán	ý nghĩa	Ví dụ
>	So sánh lớn hơn	a>b 4>5 có giá trị 0
>=	So sánh lớn hơn hoặc bằng	a>=b 6>=2 có giá trị 1
<	So sánh nhỏ hơn	a<b 6<=7 có giá trị 1
<=	So sánh nhỏ hơn hoặc bằng	a<=b 8<=5 có giá trị 0
==	So sánh bằng nhau	a==b 6==6 có giá trị 1
!=	So sánh khác nhau	a!=b 9!=9 có giá trị 0

Bốn phép toán đầu có cùng số ưu tiên, hai phép sau có cùng số thứ tự ưu tiên nhưng thấp hơn số thứ tự của bốn phép đầu.

Các phép toán quan hệ có số thứ tự ưu tiên thấp hơn so với các phép toán số học, cho nên biểu thức :

$$i < n - 1$$

được hiểu là $i < (n - 1)$.

Các phép toán logic :

Trong C sử dụng ba phép toán logic :

Phép phủ định một ngôi !

a	!a
khác 0	0
bằng 0	1

Phép và (AND) &&

Phép hoặc (OR) ||

a	b	a&&b	a b
khác 0	khác 0	1	1
khác 0	bằng 0	0	1
bằng 0	khác 0	0	1
bằng 0	bằng 0	0	0

Các phép quan hệ có số ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và ||, vì vậy biểu thức như :

$(a < b) \&\& (c > d)$

có thể viết lại thành :

$a < b \&\& c > d$

Chú ý :

Cả a và b có thể là nguyên hoặc thực.

3.5. Phép toán tăng giảm :

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng là ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm -- thì sẽ trừ toán hạng đi 1.

Ví dụ :

n=5

++n Cho ta n=6

--n Cho ta n=4

Ta có thể viết phép toán ++ và -- trước hoặc sau toán hạng như sau : ++n, n++, --n, n--.

Sự khác nhau của ++n và n++ ở chỗ : trong phép n++ thì tăng sau khi giá trị của nó đã được sử dụng, còn trong phép ++n thì n được tăng trước khi sử dụng. Sự khác nhau giữa n-- và --n cũng như vậy.

Ví dụ :

n=5

x=++n Cho ta x=6 và n=6

x=n++ Cho ta x=5 và n=6

3.6. Thứ tự ưu tiên các phép toán :

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

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

TT	Phép toán	Trình tự kết hợp
1	() [] ->	Trái qua phải
2	! ~ & * - ++ -- (type) sizeof	Phải qua trái
3	* (phép nhân) / %	Trái qua phải
4	+ -	Trái qua phải
5	<<>>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	&	Trái qua phải
9	^	Trái qua phải
10		Trái qua phải
11	&&	Trái qua phải
12		Trái qua phải
13	?:	Phải qua trái
14	= += -= *= /= %= <<= >>= &= ^= =	Phải qua trái
15	,	Trái qua phải

Chú thích :

Các phép toán tên một dòng có cùng thứ tự ưu tiên, các phép toán ở hàng trên có số ưu tiên cao hơn các số ở hàng dưới.

Đối với các phép toán cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại được chỉ ra trong cột *trình tự kết hợp*.

Ví dụ :

--px(--px) (Phải qua trái)

8/4*6=(8/4)*6 (Trái qua phải)

Nên dùng các dấu ngoặc tròn để viết biểu thức một cách chính xác.

Các phép toán lạ :

Dòng 1

[] Dùng để biểu diễn phần tử mảng, ví dụ : a[i][j]

. Dùng để biểu diễn thành phần cấu trúc, ví dụ : ht.ten

-> Dùng để biểu diễn thành phần cấu trúc thông qua con trỏ

Dòng 2

* Dùng để khai báo con trỏ, ví dụ : int *a

& Phép toán lấy địa chỉ, ví dụ : &x

(type) là phép chuyển đổi kiểu, ví dụ : (float)(x+y)

Dòng 15

Toán tử , thường dùng để viết một dãy biểu thức trong toán tử for.

3.7. Chuyển đổi kiểu giá trị :

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

Khi gán biểu thức gồm các toán hạng khác kiểu.

Khi gán một giá trị kiểu này cho một biến (hoặc phần tử mảng) kiểu khác. Điều này xảy ra trong toán tử gán, trong việc truyền giá trị các tham số thực sự cho các đối.

Ngoài ra, ta có thể chuyển từ một kiểu giá trị sang một kiểu bất kỳ mà ta muốn bằng phép chuyển sau :

(type) biểu thức

Ví dụ :

(float) (a+b)

Chuyển đổi kiểu trong biểu thức :

Khi hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán. Kết quả thu được là một giá trị kiểu cao hơn.

Chẳng hạn :

Giữa int và long thì int chuyển thành long.

Giữa int và float thì int chuyển thành float.

Giữa float và double thì float chuyển thành double.

Ví dụ :

1.5*(11/3)=4.5

$$1.5 * 11/3 = 5.5$$

$$(11/3) * 1.5 = 4.5$$

Chuyển đổi kiểu thông qua phép gán :

Giá trị của vế phải được chuyển sang kiểu vế trái đó là kiểu của kết quả. Kiểu int có thể được chuyển thành float. Kiểu float có thể chuyển thành int do chặt đi phần thập phân. Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int bằng cách cắt bỏ một vài chữ số.

Ví dụ :

int n;

n=15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức :

Theo cách này, kiểu của biểu thức được đổi thành kiểu type theo nguyên tắc trên.

Ví dụ :

Phép toán : (int)a

cho một giá trị kiểu int. Nếu a là float thì ở đây có sự chuyển đổi từ float sang int. Chú ý rằng bản thân kiểu của a vẫn không bị thay đổi. Nói cách khác, a vẫn có kiểu float nhưng (int)a có kiểu int.

Đối với hàm toán học của thư viện chuẩn, thì giá trị của đối và giá trị của hàm đều có kiểu double, vì vậy để tính căn bậc hai của một biến nguyên n ta phải dùng phép ép kiểu để chuyển kiểu int sang double như sau :

sqrt((double)n)

Phép ép kiểu có cùng số ưu tiên như các toán tử một ngôi.

Chú ý :

Muốn có giá trị chính xác trong phép chia hai số nguyên cần dùng phép ép kiểu :

((float)a)/b

Để đổi giá trị thực r sang nguyên, ta dùng :

(int)(r+0.5)

Chú ý thứ tự ưu tiên :

(int)1.4*10=1*10=10

`(int)(1.4*10)=(int)14.0=14`

Chương 4

Cấu trúc cơ bản của chương trình

4.1. Lời chú thích :

Các lời bình luận, các lời giải thích có thể đưa vào ở bất kỳ chỗ nào của chương trình để cho chương trình dễ hiểu, dễ đọc hơn mà không làm ảnh hưởng đến các phần khác. Lời giải thích được đặt giữa hai dấu `/*` và `*/`.

Trong một chương trình cần (và luôn luôn cần) viết thêm những lời giải thích để chương trình thêm rõ ràng, thêm dễ hiểu.

Ví dụ :

```
#include "stdio.h"
#include "string.h"
#include "alloc.h"
#include "process.h"
int main()
{
    char *str;
    /* Cấp phát bộ nhớ cho xâu ký tự */
    if ((str = malloc(10)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1); /* Kết thúc chương trình nếu thiếu bộ nhớ */
    }
    /* copy "Hello" vào xâu */
    strcpy(str, "Hello");
    /* Hiện thị xâu */
    printf("String is %s\n", str);

    /* Giải phóng bộ nhớ */
    free(str);
    return 0;
}
```

4.2. Lệnh và khối lệnh :

4.2.1. Lệnh :

Một biểu thức kiểu như $x=0$ hoặc $++i$ hoặc `scanf(...)` trở thành câu lệnh khi có đi kèm theo dấu ;

Ví dụ :

```
x=0;
++i;
scanf(...);
```

Trong chương trình C, dấu ; là dấu hiệu kết thúc câu lệnh.

4.2.2. Khối lệnh :

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

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

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

Khai báo ở đầu khối lệnh :

Các khai báo biến và mảng chẳng những có thể đặt ở đầu của một hàm mà còn có thể viết ở đầu khối lệnh :

```
{
    int a,b,c[50];
    float x,y,z,t[20][30];
    a==b==3;
    x=5.5; y=a*x;
    z=b*x;
    printf("\n y= %8.2f\n z=%8.2f",y,z);
}
```

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

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

Khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại trong thời gian máy làm việc bên trong khối lệnh và chúng lập tức biến mất ngay sau khi máy ra khỏi khối lệnh. Vậy :

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

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

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

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

Ví dụ :

Xét đoạn chương trình sau :

```
{
    int a=5,b=2;
    {
        int a=4;
        b=a+b;
        printf("\n a trong =%3d b=%3d",a,b);
    }
    printf("\n a ngoai =%3d b=%3d",a,b);
}
```

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

a trong =4 b=6

a ngoai =5 b=6

Do tính chất biến a trong và ngoài khối lệnh.

4.3. Cấu trúc cơ bản của chương trình :

Cấu trúc chương trình và hàm là một trong các vấn đề quan trọng của C. Về hàm ta sẽ có một chương nói tỉ mỉ về nó. ở đây ta chỉ đưa ra một số qui tắc chung :

Hàm là một đơn vị độc lập của chương trình. Tính độc lập của hàm thể hiện ở hai điểm :

Không cho phép xây dựng một hàm bên trong các hàm khác.

Mỗi hàm có các biến, mảng .. riêng của nó và chúng chỉ được sử dụng nội bộ bên trong hàm. Nói cách khác hàm là đơn vị có tính chất khép kín.

Một chương trình bao gồm một hoặc nhiều hàm. Hàm main() là thành phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện các câu lệnh đầu tiên của hàm main() và kết thúc khi gặp dấu } cuối cùng của hàm này. Khi chương trình làm việc, máy có thể chạy từ hàm này sang hàm khác.

Các chương trình C được tổ chức theo mẫu :

```
.....  
hàm 1  
.....  
hàm 2  
.....  
.....  
hàm n
```

Bên ngoài các hàm ở các vị trí (.....) là chỗ đặt : các toán tử #include ... (dùng để khai báo sử dụng các hàm chuẩn), toán tử #define ... (dùng để định nghĩa các hằng), định nghĩa kiểu dữ liệu bằng typedef, khai báo các biến ngoài, mảng ngoài....

Việc truyền dữ liệu và kết quả từ hàm này sang hàm khác được thực hiện theo một trong hai cách :

Sử dụng đối của hàm.

Sử dụng biến ngoài, mảng ngoài ...

Vậy nói tóm lại cấu trúc cơ bản của chương trình như sau :

- Các #include
- Các #define

- Khai báo các đối tượng dữ liệu ngoài (biến, mảng, cấu trúc vv..).
- Khai báo nguyên mẫu các hàm.
- Hàm main().
- Định nghĩa các hàm (hàm main có thể đặt sau hoặc xen vào giữa các hàm khác).

Ví dụ :

Chương trình tính x lũy thừa y rồi in ra máy in kết quả :

```
#include "stdio.h"
#include "math.h"
main()
{
    double x,y,z;
    printf("\n Nhap x va y");
    scanf("%lf%lf",&x,&y);
    z=pow(x,y); /* hàm lấy lũy thừa y lũy thừa x */
    fprintf(stdprn, "\n x= %8.2lf \n y=%8.2lf \n z=%8.2lf",x,y,z);
}
```

4.4. Một số qui tắc cần nhớ khi viết chương trình :

Qui tắc đầu tiên cần nhớ là :

Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu ;

Qui tắc thứ hai là :

Các lời giải thích cần được đặt giữa các dấu / và */ và có thể được viết*

Trên một dòng

Trên nhiều dòng

Trên phần còn lại của dòng

Qui tắc thứ ba là :

Trong chương trình, khi ta sử dụng các hàm chuẩn, ví dụ như printf(), getch() ,... mà các hàm này lại chứa trong file stdio.h trong thư mục của C, vì vậy ở đầu chương trình ta phải khai báo sử dụng ;

```
#include "stdio.h "
```

Qui tắc thứ tư là :

Một chương trình có thể chỉ có một hàm chính (hàm main()) hoặc có thể có thêm vài hàm khác.

Chương 5

Cấu trúc điều khiển

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các cấu trúc điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể nhảy thực hiện một câu lệnh khác ở một vị trí trước hoặc sau câu lệnh hiện thời.

Xét về mặt công dụng, có thể chia các cấu trúc điều khiển thành các nhóm chính :

Nhảy không có điều kiện.

Rẽ nhánh.

Tổ chức chu trình.

Ngoài ra còn một số toán tử khác có chức năng bỏ trợ như break, continue.

5.1. Cấu trúc có điều kiện :

5.1.1. Lệnh if-else :

Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tùy thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau :

if (biểu thức) khối lệnh 1; /* Dạng một */	if (biểu thức) khối lệnh 1; else khối lệnh 2 ; /* Dạng hai */
--	---

Hoạt động của biểu thức dạng 1 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu

biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ :

Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình.
Chương trình có thể viết bằng cả hai cách trên như sau :

```
#include "stdio.h"
main()
{
    float a,b,max;
    printf("\n Cho a=");
    scanf("%f",&a);
    printf("\n Cho b=");
    scanf("%f",&b);
    max=a;
    if (b>max) max=b;
    printf(" \n Max của hai số a=%8.2f và b=%8.2f là Max=%8.2f",a,b,max);
}
```

```
#include "stdio.h"
main()
{
    float a,b,max;
    printf("\n Cho a=");
    scanf("%f",&a);
    printf("\n Cho b=");
    scanf("%f",&b);
    if (a>b) max=a;
    else max=b;
    printf(" \n Max của hai số a=%8.2f và b=%8.2f là Max=%8.2f",a,b,max);
}
```

Sự lồng nhau của các toán tử if :

C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else.

Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất. Chẳng hạn như đoạn chương trình ví dụ sau :

```
if ( n>0 )      /* if thứ nhất*/
    if ( a>b )  /* if thứ hai*/
        z=a;
    else
        z=b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với :

```
if ( n>0 )      /* if thứ nhất*/
{
    if ( a>b )  /* if thứ hai*/
        z=a;
    else
        z=b;
}
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau :

```
if ( n>0 )      /* if thứ nhất*/
{
    if ( a>b )  /* if thứ hai*/
        z=a;
}
else
    z=b;
```

5.1.2. Lệnh else-if :

Khi muốn thực hiện một trong n quyết định ta có thể sử dụng cấu trúc sau :

```

if ( biểu thức 1 )
khởi lệnh 1;
else if ( biểu thức 2 )
khởi lệnh 2;
.....
else if ( biểu thức n-1 )
khởi lệnh n-1;
else
khởi lệnh n;

```

Trong cấu trúc này, máy sẽ đi kiểm tra từ biểu thức 1 trở đi đến khi gặp biểu thức nào có giá trị khác 0.

Nếu biểu thức thứ i (1,2, ...n-1) có giá trị khác 0, máy sẽ thực hiện khối lệnh i , rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Nếu trong cả $n-1$ biểu thức không có biểu thức nào khác 0, thì máy sẽ thực hiện khối lệnh n rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Ví dụ :

Chương trình giải phương trình bậc hai.

```

#include "stdio.h"
main()
{
float a,b,c,d,x1,x2;
printf("\n Nhap a, b, c:");
scanf("%f%f%f",&a&b&c);
d=b*b-4*a*c;
if (d<0.0)
printf("\n Phuong trinh vo nghiem ");
else if (d==0.0)
printf("\n Phuong trinh co nghiem kep x1,2=%8.2f",-b/(2*a));
else
{
printf("\n Phuong trinh co hai nghiem ");
printf("\n x1=%8.2f",(-b+sqrt(d))/(2*a));

```

```
printf("\n x2=%8.2f",(-b-sqrt(d))/(2*a));
}
```

5.2. Lệnh nhảy không điều kiện - toán tử goto :

Nhãn có cùng dạng như tên biến và có dấu : đứng ở phía sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình.

Ví dụ :

```
ts : s=s++;
```

thì ở đây **ts** là nhãn của câu lệnh gán s=s++.

Toán tử goto có dạng :

```
goto nhãn;
```

Khi gặp toán tử này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khoá goto.

Khi dùng toán tử goto cần chú ý :

Câu lệnh goto và nhãn cần nằm trong một hàm, có nghĩa là toán tử goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân một hàm và không thể dùng để nhảy từ một hàm này sang một hàm khác.

Không cho phép dùng toán tử goto để nhảy từ ngoài vào trong một khối lệnh. Tuy nhiên việc nhảy từ trong một khối lệnh ra ngoài là hoàn toàn hợp lệ. Ví dụ như đoạn chương trình sau là sai.

```
goto n1;

.....
{
    .....
    n1: printf("\n Gia tri cua N la: ");
    .....
}
```

Ví dụ :

Tính tổng s=1+2+3+....+10

```
#include "stdio.h"
```

```
main()
```

```

{
    int s,i;
    i=s=0;
    tong:
    ++i;
    s=s+i;
    if (i<10) goto tong;
    printf("\n tong s=%d",s);
}

```

5.3. Cấu trúc rẽ nhánh - toán tử switch:

Là cấu trúc tạo nhiều nhánh đặc biệt. Nó căn cứ vào giá trị một biểu thức nguyên để để chọn một trong nhiều cách nhảy.

Cấu trúc tổng quát của nó là :

switch (biểu thức nguyên)

```

{
    case n1
        khối lệnh 1
    case n2
        khối lệnh 2
    .....
    case nk
        khối lệnh k
    [ default
        khối lệnh k+1 ]
}

```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau.

Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch.

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

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

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

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

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

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

Chú ý :

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

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

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

Ví dụ :

Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch :

```
#include "stdio.h"
main()
{
    int diem;
    tt: printf("\nVao du lieu :");
    printf("\n Diem =");
    scanf("%d",&diem);
    switch (diem)
    {
        case 0:
        case 1:
        case 2:
        case 3:printf("Kem\n");break;
        case 4:printf("Yeu\n");break;
```

```

        case 5:
        case 6:printf("Trung binh\n");break;
        case 7:
        case 8:printf("Kha\n");break;
        case 9:
        case 10:printf("Gioi\n");break;
        default:printf("Vao sai\n");
    }
    printf("Tiep tuc 1, dung 0 :")
    scanf("%d",&diem);
    if (diem==1) goto tt;
    getch();
    return;
}

```

5.4. Cấu trúc lặp :

5.4.1. Cấu trúc lặp với toán tử while và for :

5.4.1.1. Cấu trúc lặp với toán tử while :

Toán tử while dùng để xây dựng chu trình lặp dạng :

```

while ( biểu thức )
    Lệnh hoặc khối lệnh;

```

Như vậy toán tử while gồm một biểu thức và thân chu trình. Thân chu trình có thể là một lệnh hoặc một khối lệnh.

Hoạt động của chu trình như sau :

Máy xác định giá trị của biểu thức, tùy thuộc giá trị của nó máy sẽ chọn cách thực hiện như sau :

Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi chu trình và chuyển tới thực hiện câu lệnh tiếp sau chu trình trong chương trình.

Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh hoặc khối lệnh trong thân của while. Khi máy thực hiện xong khối lệnh này nó lại thực hiện xác định lại giá trị biểu thức rồi làm tiếp các bước như trên.

Chú ý :

Trong các dấu ngoặc () sau while chẳng những có thể đặt một biểu thức mà còn có thể đặt một dãy biểu thức phân cách nhau bởi dấu phẩy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

Bên trong thân của một toán tử while lại có thể sử dụng các toán tử while khác. bằng cách đó ta đi xây dựng được các chu trình lồng nhau.

Khi gặp câu lệnh break trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

Trong thân while có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân while để ra khỏi một hàm nào đó.

Ví dụ :

Chương trình tính tích vô hướng của hai véc tơ x và y :

Cách 1 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
{
    float s=0;
    int i=-1;
    while (++i<4)
        s+=x[i]*y[i];
    printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
}
```

Cách 2 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
{
    float s=0;
    int i=0;
```



```

while (1)
{
    s+=x[i]*y[i];
    if (++i>=4) goto kt;
}
kt:printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
}

```

Cách 3 :

```

#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
{
    float s=0;
    int i=0;
    while ( s+=x[i]*y[i], ++i<=3 );
    printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
}

```

5.4.1.2. Cấu trúc lặp với toán tử for :

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau :

```

for ( biểu thức 1; biểu thức 2; biểu thức 3)
    Lệnh hoặc khối lệnh ;

```

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ; .

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for :

Toán tử for hoạt động theo các bước sau :

Xác định biểu thức 1

Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh :

Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for.

Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Chú ý :

Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu ;. Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phẩy. Khi đó các biểu thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một vị trí mong muốn bất kỳ.

Ví dụ 1:

Nhập một dãy số rồi đảo ngược thứ tự của nó.

Cách 1:

```
#include "stdio.h"
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
    int i,j;
```

```

float c;
for (i=0,j=n-1;i<j;++i,--j)
{
    c=x[i];x[i]=x[j];x[j]=c;
}
fprintf(stdprn,“\n Day so dao la \n\n”);
for (i=0;i<n;++i)
fprintf(stdprn,“%8.2f”,x[i]);
}

```

Cách 2 :

```

#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
    int i,j;
    float c;
    for (i=0,j=n-1;i<j;c=x[i],x[i]=x[j],x[j]=c,++i,--j)
    fprintf(stdprn,“\n Day so dao la \n\n”);
    for (i=0;+i<n;)
    fprintf(stdprn,“%8.2f”,x[i]);
}

```

Cách 3 :

```

#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
    int i=0,j=n-1;
    float c;
    for ( ; ; )

```

```

    {
        c=x[i];x[i]=x[j];x[j]=c;
        if (++i>--j) break;
    }
    fprintf(stdprn, "\n Day so dao la \n\n");
    for (i=-1;i++<n-1; fprintf(stdprn, "%8.2f",x[i]));
}

```

Ví dụ 2:

Tính tích hai ma trận $m \times n$ và $n \times p$.

```

#include "stdio.h"
float x[3][2],y[2][4],z[3][4],c;
main()
{
    int i,j;
    printf("\n nhap gia tri cho ma tran X ");
    for (i=0;i<=2;++i)
    for (j=0;j<=1;++j)
    {
        printf("\n x[%d][%d]=",i,j);
        scanf("%f",&c);
        x[i][j]=c;
    }
    printf("\n nhap gia tri cho ma tran Y ");
    for (i=0;i<=1;++i)
    for (j=0;j<=3;++j)
    {
        printf("\n y[%d][%d]=",i,j);
        scanf("%f",&c);
        y[i][j]=c;
    }
    for (i=0;i<=3;++i)
    for (j=0;j<=4;++j)

```

```
z[i][j]
}
```

5.4.2. Chu trình do-while

Khác với các toán tử while và for, việc kiểm tra điều kiện kết thúc đặt ở đầu chu trình, trong chu trình do while việc kiểm tra điều kiện kết thúc đặt cuối chu trình. Như vậy thân của chu trình bao giờ cũng được thực hiện ít nhất một lần.

Chu trình do while có dạng sau :

do

Lệnh hoặc khối lệnh;

while (biểu thức);

Lệnh hoặc khối lệnh là thân của chu trình có thể là một lệnh riêng lẻ hoặc là một khối lệnh.

Hoạt động của chu trình như sau :

Máy thực hiện các lệnh trong thân chu trình.

Khi thực hiện xong tất cả các lệnh trong thân của chu trình, máy sẽ xác định giá trị của biểu thức sau từ khoá while rồi quyết định thực hiện như sau :

Nếu biểu thức đúng (khác 0) máy sẽ thực hiện lặp lại khối lệnh của chu trình lần thứ hai rồi thực hiện kiểm tra lại biểu thức như trên.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc chu trình và chuyển tới thực hiện lệnh đứng sau toán tử while.

Chú ý :

Những điều lưu ý với toán tử while ở trên hoàn toàn đúng với do while.

Ví dụ :

Đoạn chương trình xác định phần tử âm đầu tiên trong các phần tử của mảng x.

```
#include "stdio.h"
```

```
float x[5],c;
```

```
main()
```

```
{
```

```
int i=0;
```

```
printf("\n nhap gia tri cho ma tran x ");
```

```

for (i=0;i<=4;++i)
{
    printf("\n x[%d]=",i);
    scanf("%f",&c);
    y[i]=c;
}
do
    ++i;
while (x[i]>=0 && i<=4);
if (i<=4)
    printf("\n Phan tu am dau tien = x[%d]=%8.2f",i,x[i]);
else
    printf("\n Mang khong có phan tu am ");
}

```

5.5. Câu lệnh break :

Câu lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình bên trong nhất chứa nó không cần điều kiện gì. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ví dụ :

Biết số nguyên dương n sẽ là số nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn bậc hai của n. Viết đoạn chương trình đọc vào số nguyên dương n, xem n có là số nguyên tố.

```

#include "stdio.h"
#include "math.h"
unsigned int n;
main()
{
    int i,nt=1;
    printf("\n cho n=");
    scanf("%d",&n);
    for (i=2;i<=sqrt(n);++i)

```

```

    if ((n % i) == 0)
    {
        nt=0;
        break;
    }
    if (nt)
        printf("\n %d la so nguyen to",n);
    else
        printf("\n %d khong la so nguyen to",n);
}

```

5.6. Câu lệnh continue :

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra, còn trong for điều khiển được chuyển về bước khởi đầu lại (tức là bước : tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình).

Chú ý :

Lệnh continue chỉ áp dụng cho chu trình chứ không áp dụng cho switch.

Ví dụ :

Viết chương trình để từ một nhập một ma trận a sau đó :

Tính tổng các phần tử dương của a.

Xác định số phần tử dương của a.

Tìm cực đại trong các phần tử dương của a.

```

#include "stdio.h"
float a[3][4];
main()
{
    int i,j,soptd=0;
    float tongduong=0,cucdai=0,phu;
    for (i=0;i<3;++i)
        for (j=0;j<4;++j)

```

```

{
    printf("\n a[%d][%d]=",i,j );
    scanf("%f",&phu);
    a[i][j]=phu;
    if (a[i][j]<=0) continue;
    tongduong+=a[i][j];
    if (cucdai<a[i][j]) cucdai=a[i][j];
    ++soptd;
}
printf("\n So phan tu duong la : %d",soptd);
printf("\n Tong cac phan tu duong la : %8.2f",tongduong);
printf("\n Cuc dai phan tu duong la : %8.2f",cucdai);
}

```


Chương 6

Hàm

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó có một hàm chính (hàm main()). Hàm chia các bài toán lớn thành các công việc nhỏ hơn, giúp thực hiện những công việc lặp lại nào đó một cách nhanh chóng mà không phải viết lại đoạn chương trình. Thứ tự các hàm trong chương trình là bất kỳ, song chương trình bao giờ cũng đi thực hiện từ hàm main().

6.1. Cơ sở :

Hàm có thể xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không có phép xây dựng một hàm bên trong các hàm khác.

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau :

```
type tên hàm ( khai báo các đối )
{
    Khai báo các biến cục bộ
    Các câu lệnh
    [return[biểu thức];]
}
```

Dòng tiêu đề :

Trong dòng đầu tiên của hàm chứa các thông tin về : kiểu hàm, tên hàm, kiểu và tên mỗi đối.

Ví dụ :

```
float max3s(float a, float b, float c)
```

khai báo các đối có dạng :

Kiểu đối 1 tên đối 1, kiểu đối 2 tên đối 2, ..., kiểu đối n tên đối n

Thân hàm :

Sau dòng tiêu đề là thân hàm. Thân hàm là nội dung chính của hàm bắt đầu và kết thúc bằng các dấu { }.

Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đã đề ra cho hàm.

Thân hàm có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở các chỗ khác nhau, và cũng có thể không sử dụng câu lệnh này.

Dạng tổng quát của nó là :

```
return [biểu thức];
```

Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

Ví dụ :

Xét bài toán : Tìm giá trị lớn nhất của ba số mà giá trị của chúng được đưa vào bàn phím.

Xây dựng chương trình và tổ chức thành hai hàm : Hàm main() và hàm max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số đọc vào, giả sử là a,b,c. Nhiệm vụ của hàm main() là đọc ba giá trị vào từ bàn phím, rồi dùng hàm max3s để tính như trên, rồi đưa kết quả ra màn hình.

Chương trình được viết như sau :

```
#include "stdio.h"
```

```
float max3s(float a,float b,float c ); /* Nguyên mẫu hàm*/
```

```
main()
```

```
{
```

```
    float x,y,z;
```

```
    printf("\n Vao ba so x,y,z:");
```

```
    scanf("%f%f%f",&x&y&z);
```

```
    printf("\n Max cua ba so x=%8.2f y=%8.2f z=%8.2f la : %8.2f",
```

```
    x,y,z,max3s(x,y,z));
```

```
}
```

```
/* Kết thúc hàm main*/
```

```
float max3s(float a,float b,float c)
```

```
{
```

```
    float max;
```

```
    max=a;
```

```
    if (max<b) max=b;
```

```
    if (max<c) max=c;
    return(max);
} /* Kết thúc hàm max3s*/
```

Quy tắc hoạt động của hàm :

Một cách tổng quát lời gọi hàm có dạng sau :

tên hàm ([Danh sách các tham số thực])

Số các tham số thực thế thay vào trong danh sách các đối phải bằng số tham số hình thức và lần lượt chúng có kiểu tương ứng với nhau.

Khi gặp một lời gọi hàm thì nó sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau :

Cấp phát bộ nhớ cho các biến cục bộ.

Gán giá trị của các tham số thực cho các đối tương ứng.

Thực hiện các câu lệnh trong thân hàm.

Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối, biến cục bộ và ra khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

Các tham số thực, các đối và biến cục bộ :

Do đối và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối và biến cục bộ cần có tên khác nhau.

Đối và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được xét đến và bị xoá khi ra khỏi hàm nên ta không thể mang giá trị của đối ra khỏi hàm.

Đối và biến cục bộ có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gán cho các đối (trong ví dụ trên hàm max3s, các tham số thực là x,y,z, các đối tương ứng là a,b,c). Như vậy các đối chính là các bản sao của các tham số thực. Hàm chỉ làm việc trên các đối.

Các đối có thể bị biến đổi trong thân hàm, còn các tham số thực thì không bị thay đổi.

Chú ý :

Khi hàm khai báo không có kiểu ở trước nó thì nó được mặc định là kiểu int.

Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nói chung nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm hay tự động việc chuyển dạng.

Nguyên mẫu của hàm thực chất là dòng đầu tiên của hàm thêm vào dấu ;. Tuy nhiên trong nguyên mẫu có thể bỏ tên các đối.

Hàm thường có một vài đối. Ví dụ như hàm max3s có ba đối là a,b,c. cả ba đối này đều có giá trị float. Tuy nhiên, cũng có hàm không đối như hàm main.

Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị các đối.

6.2. Hàm không cho các giá trị :

Các hàm không cho giá trị giống như thủ tục (procedure) trong ngôn ngữ lập trình PASCAL. Trong trường hợp này, kiểu của nó là void.

Ví dụ hàm tìm giá trị max trong ba số là max3s ở trên có thể được viết thành thủ tục hiển thị số cực đại trong ba số như sau :

```
void htmax3s(float a, float b, float c)
{
    float max;
    max=a;
    if (max<b) max=b;
    if (max<c) max=c;
}
```

Lúc này, trong hàm main ta gọi hàm htmax3s bằng câu lệnh :

```
htmax3s(x,y,z);
```

6.3. Hàm đệ qui :

6.3.3. Mở đầu :

C không những cho phép từ hàm này gọi tới hàm khác, mà nó còn cho phép từ một điểm trong thân của một hàm gọi tới chính hàm đó. Hàm như vậy gọi là hàm đệ qui.

Khi hàm gọi đệ qui đến chính nó, thì mỗi lần gọi máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với tập các biến cục bộ đã được tạo ra trong các lần gọi trước.

Để minh họa chi tiết những điều trên, ta xét một ví dụ về tính giai thừa của số nguyên dương n. Khi không dùng phương pháp đệ qui hàm có thể được viết như sau :

```

long int gt(int n) /* Tính n! với n>=0*/
{
    long int gtpHu=1;
    int i;
    for (i=1;i<=n;++i)
        gtpHu*=i;
    return s;
}

```

Ta nhận thấy rằng $n!$ có thể tính theo công thức truy hồi sau :

$n!=1$	nếu $n=0$
$n!=n*(n-1)!$	nếu $n>0$

Hàm tính $n!$ theo phương pháp đệ qui có thể được viết như sau :

```

long int gtdq(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return(n*gtdq(n-1));
}

```

Ta đi giải thích hoạt động của hàm đệ qui khi sử dụng trong hàm main dưới đây :

```
#include "stdio.h"
```

```
main()
```

```

{
    printf("\n 3!=%d",gtdq(3));
}

```

Lần gọi đầu tiên tới hàm gtdq được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến tự động của hàm gtdq. Tập này chỉ gồm các đối n. Ta gọi đối n được tạo ra lần thứ nhất là n thứ nhất. Giá trị của tham số thực (số 3) được gán cho n thứ nhất. Lúc này biến n trong thân hàm được xem là n thứ nhất. Do n thứ nhất có giá trị bằng 3 nên điều kiện trong toán tử if là sai và do đó máy sẽ lựa chọn câu lệnh else. Theo câu lệnh này, máy sẽ tính giá trị biểu thức :

$$n * gtdq(n-1) (*)$$

Để tính biểu thức trên, máy cần gọi chính hàm gtdq vì thế lần gọi thứ hai sẽ thực hiện. Máy sẽ tạo ra đối n mới, ta gọi đó là n thứ hai. Giá trị của n-1 ở đây lại là đối của hàm, được truyền cho hàm và hiểu là n thứ hai, do vậy n thứ hai có giá trị là 2. Bây giờ, do n thứ hai vẫn chưa thoả mãn điều kiện if nên máy lại tiếp tục tính biểu thức :

n*gtdq(n-1) (**)

Biểu thức trên lại gọi hàm gtdq lần thứ ba. Máy lại tạo ra đối n lần thứ ba và ở đây n thứ ba có giá trị bằng 1. Đối n=1 thứ ba lại được truyền cho hàm, lúc này điều kiện trong lệnh if được thoả mãn, máy đi thực hiện câu lệnh :

return 1=gtdq(1) (***)

Bắt đầu từ đây, máy sẽ thực hiện ba lần ra khỏi hàm gtdq. Lần ra khỏi hàm thứ nhất ứng với lần vào thứ ba. Kết quả là đối n thứ ba được giải phóng, hàm gtdq(1) cho giá trị là 1 và máy trở về xét giá trị biểu thức

n*gtdq(1) đây là kết quả của (**)

ở đây, n là n thứ hai và có giá trị bằng 2. Theo câu lệnh return, máy sẽ thực hiện lần ra khỏi hàm lần thứ hai, đối n thứ hai sẽ được giải phóng, kết quả là biểu thức trong (**) có giá trị là 2.1. Sau đó máy trở về biểu thức (*) lúc này là :

n*gtdq(2)=n*2*1

n lại hiểu là thứ nhất, nó có giá trị bằng 3, do vậy giá trị của biểu thức trong (*) là 3.2.1=6. Chính giá trị này được sử dụng trong câu lệnh printf của hàm main() nên kết quả in ra trên màn hình là :

3!=6

Chú ý :

Hàm đệ qui so với hàm có thể dùng vòng lặp thì đơn giản hơn, tuy nhiên với máy tính khi dùng hàm đệ qui sẽ dùng nhiều bộ nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy khi gặp một bài toán mà có thể có cách giải lặp (không dùng đệ qui) thì ta nên dùng cách lặp này. Song vẫn tồn tại những bài toán chỉ có thể giải bằng đệ qui.

6.3.2. Các bài toán có thể dùng đệ qui :

Phương pháp đệ qui thường áp dụng cho các bài toán phụ thuộc tham số có hai đặc điểm sau :

Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Người ta thường gọi là trường hợp suy biến.

Trong trường hợp tổng quát, bài toán có thể qui về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Sau một số hữu hạn bước biến đổi đệ qui nó sẽ dẫn tới trường hợp suy biến.

Bài toán tính n giai thừa nêu trên thể hiện rõ nét đặc điểm này.

6.3.3. Cách xây dựng hàm đệ qui :

Hàm đệ qui thường được xây dựng theo thuật toán sau :

```
if ( trường hợp suy biến)
{
    Trình bày cách giải bài toán khi suy biến
}
else /* Trường hợp tổng quát */
{
    Gọi đệ qui tới hàm ( đang viết ) với các giá
    trị khác của tham số
}
```

6.3.4. Các ví dụ về dùng hàm đệ qui :

Ví dụ 1 :

Bài toán dùng đệ qui tìm USCLN của hai số nguyên dương a và b .

Trong trường hợp suy biến, khi $a=b$ thì USCLN của a và b chính là giá trị của chúng.

Trong trường hợp chung :

$uscln(a,b)=uscln(a-b,b)$ nếu $a>b$

$uscln(a,b)=uscln(a,b-a)$ nếu $a<b$

Ta có thể viết chương trình như sau :

```
#include "stdio.h"
```

```
int uscln(int a,int b ); /* Nguyên mẫu hàm*/
```

```
main()
```

```
{    int m,n;
    printf("\n Nhập các giá trị của a và b :");
    scanf("%d%d",&m,&n);
    printf("\n USCLN của a=%d và b=%d là :%d",m,m,uscln(m,n))
```

```

    }
int uscln(int a,int b)
    {
        if (a==b)
            return a;
        else
            if (a>b)
                return uscln(a-b,b);

            else
                return uscln(a,b-a);
    }

```

Ví dụ 2 :

Chương trình đọc vào một số rồi in nó ra dưới dạng các ký tự liên tiếp.

```

#include "stdio.h"
#include "conio.h"
void prind(int n);
main()
{
    int a;
    clrscr();
    printf("n=");
    scanf("%d",&a);
    prind(a);
    getch();
}
void prind(int n)
{
    int i;
    if (n<0)
    { putchar('-');
      n=-n;

```



```

}
if ((i=n/10)!=0)
prind(i);
putchar(n%10+'0');
}

```

6.4. Bộ tiền xử lý C :

C đưa ra một số cách mở rộng ngôn ngữ bằng các bộ tiền xử lý macro đơn giản. Có hai cách mở rộng chính là #define mà ta đã học và khả năng bao hàm nội dung của các file khác vào file đang được dịch.

Bao hàm file :

Để dễ dàng xử lý một tập các #define và khai báo (trong các đối tượng khác), C đưa ra cách bao hàm các file khác vào file đang dịch có dạng :

```
#include "tên file"
```

Dòng khai báo trên sẽ được thay thế bởi nội dung của file có tên là tên file. Thông thường có vài dòng như vậy xuất hiện tại đầu mỗi file gốc để gọi vào các câu lệnh #define chung và các khai báo cho các biến ngoài. Các #include được phép lồng nhau. Thường thì các #include được dùng nhiều trong các chương trình lớn, nó đảm bảo rằng mọi file gốc đều được cung cấp cùng các định nghĩa và khai báo biến, do vậy tránh được các lỗi khó chịu do việc thiếu các khai báo định nghĩa. Tất nhiên khi thay đổi file được bao hàm vào thì mọi file phụ thuộc vào nó đều phải dịch lại.

Phép thế MACRO :

Định nghĩa có dạng :

```
#define biểu thức 1 [ biểu thức 2 ]
```

sẽ gọi tới một macro để thay thế biểu thức 2 (nếu có) cho biểu thức 1.

Ví dụ :

```
#define YES 1
```

Macro thay biến YES bởi giá trị 1 có nghĩa là hễ có chỗ nào trong chương trình có xuất hiện biến YES thì nó sẽ được thay bởi giá trị 1.

Phạm vi cho tên được định nghĩa bởi #define là từ điểm định nghĩa đến cuối file gốc. Có thể định nghĩa lại tên và một định nghĩa có thể sử dụng các định nghĩa khác trước đó. Phép thế

không thực hiện cho các dấu nháy, ví dụ như YES là tên được định nghĩa thì không có việc thay thế nào được thực hiện trong đoạn lệnh có "YES".

Vì việc thiết lập #define là một bước chuẩn bị chứ không phải là một phần của chương trình biên dịch nên có rất ít hạn chế về văn phạm về việc phải định nghĩa cái gì. Chẳng hạn như những người lập trình ưa thích PASCAL có thể định nghĩa :

```
#define then
#define begin {
#define end; }
```

sau đó viết đoạn chương trình :

```
if (i>0) then
begin
    a=i;
    .....
end;
```

Ta cũng có thể định nghĩa các macro có đối, do vậy văn bản thay thế sẽ phụ thuộc vào cách gọi tới macro.

Ví dụ :

Định nghĩa macro gọi max như sau :

```
#define max(a,b) ((a)>(b) ?(a):(b))
```

Việc sử dụng :

```
x=max(p+q,r+s);
```

tương đương với :

```
x=((p+q)>(r+s) ? (p+q):(r+s));
```

Như vậy ta có thể có hàm tính cực đại viết trên một dòng. Chừng nào các đối còn giữ được tính nhất quán thì macro này vẫn có giá trị với mọi kiểu dữ liệu, không cần phải có các loại hàm max khác cho các kiểu dữ liệu khác nhưng vẫn phải có đối cho các hàm.

Tất nhiên nếu ta kiểm tra lại việc mở rộng của hàm max trên, ta sẽ thấy rằng nó có thể gây ra số bẫy. Biểu thức đã được tính lại hai lần và điều này là không tốt nếu nó gây ra hiệu quả phụ kiểu như các lời gọi hàm và toán tử tăng. Cần phải thận trọng dùng thêm dấu ngoặc để đảm bảo trật tự tính toán. Tuy vậy, macro vẫn rất có giá trị.

Chú ý :

Không được viết dấu cách giữa tên macro với dấu mở ngoặc bao quanh danh sách đối.

Ví dụ :

Xét chương trình sau :

```
main()
{
    int x,y,z;
    x=5;
    y=10*5;
    z=x+y;
    z=x+y+6;
    z=5*x+y;
    z=5*(x+y);
    z=5*((x)+(y));
    printf("Z=%d",z);
    getch();
    return;
}
```

Chương trình sử dụng MACRO sẽ như sau :

```
#define BEGIN {
#define END }
#define INTEGER int
#define NB 10
#define LIMIT NB*5
#define SUMXY x+y
#define SUM1 (x+y)
#define SUM2 ((x)+(y))
main()
    BEGIN
        INTEGER x,y,z;
        x=5;
        y=LIMIT;
```

```
z=SUMXY;  
z=5*SUMXY;  
z=5*SUM1;  
z=5*SUM2;  
printf("\n Z=%d",z);  
getch();  
return;
```

END

Chương 7

Con trỏ

Con trỏ là biến chứa địa chỉ của một biến khác. Con trỏ được sử dụng rất nhiều trong C, một phần là do chúng đôi khi là cách duy nhất để biểu diễn tính toán, và phần nữa do chúng thường làm cho chương trình ngắn gọn và có hiệu quả hơn các cách khác.

Con trỏ đã từng bị coi như có hại chẳng kém gì lệnh goto do cách sử dụng chúng đã tạo ra các chương trình khó hiểu. Điều này chắc chắn là đúng khi người ta sử dụng chúng một cách lộn xộn và do đó tạo ra các con trỏ trỏ đến đâu đó không biết trước được.

7.1. Con trỏ và địa chỉ :

Vì con trỏ chứa địa chỉ của đối tượng nên nó có thể xâm nhập vào đối tượng gián tiếp qua con trỏ. Giả sử x là một biến kiểu int, và giả sử px là con trỏ được tạo ra theo một cách nào đó.

Phép toán một ngôi & sẽ cho địa chỉ của đối tượng, nên câu lệnh :

```
px=&x;
```

sẽ gán địa chỉ của biến x cho trỏ px, và px bây giờ được gọi là " trỏ tới biến x ". Phép toán & chỉ áp dụng được cho các biến và phần tử mảng, kết cấu kiểu &(x+1) và &3 là không hợp lệ. Lấy địa chỉ của biến register cũng là sai.

Phép toán một ngôi * coi là toán hạng của nó là đại chỉ cần xét và thâm nhập tới địa chỉ đó để lấy ra nội dung. Nếu biến y có kiểu int thì lệnh :

```
y=*px;
```

sẽ gán giá trị của biến mà trỏ px trỏ tới. Vậy dãy lệnh :

```
px=&x;
```

```
y=*px;
```

sẽ gán giá trị của x cho y như trong lệnh :

```
y=x;
```

Các khai báo cho các biến con trỏ có dạng :

```
tên kiểu *tên con trỏ
```

Ví dụ :

Như trong ví dụ trên, ta khai báo con trỏ px kiểu int :

```
int *px;
```

Trong khai báo trên ta đã ngụ ý nói rằng đó là một cách tượng trưng, rằng tổ hợp `*px` có kiểu `int`, tức là nếu `px` xuất hiện trong ngữ cảnh `*px` thì nó cũng tương đương với biến có kiểu `int`.

Con trỏ có thể xuất hiện trong các biểu thức. Chẳng hạn, nếu `px` trỏ tới số nguyên `x` thì `*px` có thể xuất hiện trong bất kỳ ngữ cảnh nào mà `x` có thể xuất hiện.

Ví dụ :

Lệnh `y=*px+1;`

sẽ đặt `y` lớn hơn `x` một đơn vị.

Lệnh `printf("%d",*px);`

sẽ in ra giá trị hiện tại của `x`

Lệnh :

```
d=sqrt((double) *px);
```

sẽ gán cho biến `d` căn bậc hai của `x`, giá trị này bị buộc phải chuyển sang `double` trước khi được chuyển cho `sqrt` (cách dùng hàm `sqrt`).

Trong các biểu thức kiểu như :

```
y=*px+1;
```

phép toán một ngôi `*` và `&` có mức ưu tiên cao hơn các phép toán số học, cho nên biểu thức này lấy bất kỳ giá trị nào mà `px` trỏ tới, cộng với 1 rồi gán cho `y`.

Con trỏ cũng có thể xuất hiện bên vế trái của phép gán. Nếu `px` trỏ tới `x` thì sau lệnh :

```
*px=0;
```

`x` sẽ có giá trị bằng 0. Cũng tương tự các lệnh:

```
*px+=1;
```

```
(*px)++;
```

sẽ tăng giá trị của `x` lên 1 đơn vị.

Các dấu ngoặc đơn ở câu lệnh cuối là cần thiết , nếu không thì biểu thức sẽ tăng `px` thay cho tăng ở biến mà nó trỏ tới vì phép toán một ngôi như `*` và `++` được tính từ phải sang trái.

Cuối cùng, vì con trỏ là biến nên ta có thao tác chúng như đối với các biến khác. Nếu `py` cũng là con trỏ `int` thì lệnh :

```
py=px;
```

sẽ sao nội dung của `px` vào `py`, nghĩa là làm cho `py` trỏ tới nơi mà `px` trỏ.

7.2. Con trỏ và mảng một chiều :

Trong C có mối quan hệ chặt chẽ giữa con trỏ và mảng : các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ.

7.2.1. Phép toán lấy địa chỉ :

Phép toán này chỉ áp dụng cho các phần tử của mảng một chiều. Giả sử ta có khai báo :

```
double b[20];
```

Khi đó phép toán :

```
&b[9]
```

sẽ cho địa chỉ của phần tử b[9].

7.2.2. Tên mảng là một hằng địa chỉ :

Khi khai báo :

```
float a[10];
```

máy sẽ bố trí bộ nhớ cho mảng a mười khoảng nhớ liên tiếp, mỗi khoảng nhớ là 4 byte. Như vậy, nếu biết địa chỉ của một phần tử nào đó của mảng a, thì ta có thể dễ dàng suy ra địa chỉ của các phần tử khác của mảng.

Với C ta có :

a tương đương với `&a[0]`

a+i tương đương với `&a[i]`

*(a+i) tương đương với `a[i]`

7.2.3. Con trỏ trỏ tới các phần tử của mảng một chiều :

Khi con trỏ pa trỏ tới phần tử a[k] thì :

pa+i trỏ tới phần tử thứ i sau a[k], có nghĩa là nó trỏ tới a[k+i].

pa-i trỏ tới phần tử thứ i trước a[k], có nghĩa là nó trỏ tới a[k-i].

*(pa+i) tương đương với pa[i].

Như vậy, sau hai câu lệnh :

```
float a[20],*p;
```

```
p=a;
```

thì bốn cách viết sau có tác dụng như nhau :

```
a[i]    *(a+i)    p[i]    *(p+i)
```

Ví dụ :

Vào số liệu của các phần tử của một mảng và tính tổng của chúng :

Cách 1:

```
#include "stdio.h"
main()
{
    float a[4], tong;
    int i;
    for (i=0; i<4; ++i)
    {
        printf("\n a[%d]=", i);
        scanf("%f", &a[i]);
    }
    tong=0;
    for (i=0; i<4; ++i)
        tong+=a[i];
    printf("\n Tong cac phan tu mang la :%8.2f", tong);
}
```

Cách 2 :

```
#include "stdio.h"
main()
{
    float a[4], tong, *troa;
    int i;
    troa=a;
    for (i=0; i<4; ++i)
    {
        printf("\n a[%d]=", i);
        scanf("%f", &troa[i]);
    }
    tong=0;
    for (i=0; i<4; ++i)
```



```

        tong+=troa[i];
    printf("\n Tong cac phan tu mang la :%8.2f ",tong);
}

```

Cách 3 :

```

#include "stdio.h"
main()
{
    float a[4],tong,*troa;
    int i;
    troa=a;
    for (i=0;i<4;++i)
    {
        printf("\n a[%d]=",i);
        scanf("%f",troa+i);
    }
    tong=0;
    for (i=0;i<4;++i)
        tong+=*(troa+i);
    printf("\n Tong cac phan tu mang la :%8.2f ",tong);
}

```

Chú ý :

Mảng một chiều và con trỏ tương ứng phải cùng kiểu.

7.2.4. Mảng, con trỏ và chuỗi ký tự :

Như ta đã biết trước đây, chuỗi ký tự là một dãy ký tự đặt trong hai dấu nháy kép, ví dụ như :

```
"Viet nam"
```

Khi gặp một chuỗi ký tự, máy sẽ cấp phát một khoảng nhớ cho một mảng kiểu char đủ lớn để chứa các ký tự của chuỗi và chứa thêm ký tự '\0' là ký tự dùng làm ký tự kết thúc của một chuỗi ký tự. Mỗi ký tự của chuỗi được chứa trong một phần tử của mảng.

Cũng giống như tên mảng, xâu ký tự là một hàng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo biến **xau** như một con trỏ kiểu char :

```
char *xau;
```

thì phép gán :

```
xau="Ha noi"
```

là hoàn toàn có nghĩa. Sau khi thực hiện câu lệnh này trong con trỏ **xau** sẽ có địa chỉ đầu của mảng (kiểu char) đang chứa xâu ký tự bên phải. Khi đó các câu lệnh :

```
puts("Ha noi");
```

```
puts(xau);
```

sẽ có cùng một tác dụng là cho hiện lên màn hình dòng chữ **Ha noi**.

Mảng kiểu char thường dùng để chứa một dãy ký tự đọc vào bộ nhớ. Ví dụ, để nạp từ bàn phím tên của một người ta dùng một mảng kiểu char với độ dài 25, ta sử dụng các câu lệnh sau :

```
char ten[25];
```

```
printf("\n Ho ten :");
```

```
gets(ten);
```

Bây giờ ta xem giữa mảng kiểu char và con trỏ kiểu char có những gì giống và khác nhau. Để thấy được sự khác nhau của chúng, ta đưa ra sự so sánh sau :

```
char *xau, ten[15];
```

```
ten="Ha noi"
```

```
gets(xau);
```

Các câu lệnh trên là không hợp lệ. Câu lệnh thứ hai sai ở chỗ : ten là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác. Câu lệnh thứ ba không thực hiện được, mục đích của câu lệnh là đọc từ bàn phím một dãy ký tự và lưu vào một vùng nhớ mà con trỏ **xau** trỏ tới. Song nội dung của con trỏ **xau** còn chưa xác định. Nếu trỏ **xau** đã trỏ tới một vùng nhớ nào đó thì câu lệnh này hoàn toàn có ý nghĩa. Chẳng hạn như sau khi thực hiện câu lệnh :

```
xau=ten;
```

thì cách viết :

```
gets(ten) ; và gets(xau);
```

đều có tác dụng như nhau.

7.3. Con trỏ và mảng nhiều chiều :

Việc sử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều. Không phải mọi qui tắc đúng với mảng một chiều đều có thể áp dụng cho mảng nhiều chiều.

7.3.1. Phép lấy địa chỉ :

Phép lấy địa chỉ đối với các phần tử mảng hai chiều chỉ có thể áp dụng khi các phần tử mảng hai chiều có kiểu nguyên, còn lại thì phép lấy địa chỉ cho các phần tử mảng nhiều chiều là không thực hiện được. Ví dụ như ta có thể lấy địa chỉ `&a[1][2]` khi `a` là mảng nguyên.

Thuật đọc từ bàn phím phần tử mảng hai chiều dùng lệnh `scanf` :

Chương trình đọc vào số liệu cho một ma trận hai chiều sẽ được thực hiện thông qua việc đọc vào một biến trung gian, đọc một giá trị và chứa tạm vào một biến trung gian sau đó ta gán biến cho phần tử mảng:

```
#include "stdio.h"
main()
{
    float a[2][3], tg;
    int i,j;
    for (i=0;i<2;++i)
        for (j=0;j<3;++j)
        {
            printf("\n a[%d][%d]=",i,j);
            scanf("%8.2f",&tg);
            a[i][j]=tg;
        }
}
```

7.3.2. Phép cộng địa chỉ trong mảng hai chiều:

Giả sử ta có mảng hai chiều `a[2][3]` có 6 phần tử ứng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự sau :

Phần tử	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>
Địa chỉ	1	2	3	4	5	6

Tên mảng `a` biểu thị địa chỉ đầu tiên của mảng. Phép cộng địa chỉ ở đây được thực hiện như sau :

C coi mảng hai chiều là mảng (một chiều) của mảng, như vậy khai báo

```
float a[2][3];
```

thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của mảng).

Vì vậy :

a trở phần tử thứ nhất của mảng : phần tử a[0][0]

a+1 trở phần tử đầu hàng thứ hai của mảng : phần tử a[1][0]

.....

7.3.3. Con trỏ và mảng hai chiều :

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta có thể dùng con trỏ như minh

hoạ ở ví dụ sau :

```
float *pa,a[2][3];
```

```
pa=(float*)a;
```

lúc đó :

pa trở tới a[0][0]

pa+1 trở tới a[0][1]

pa+2 trở tới a[0][2]

pa+3 trở tới a[1][0]

pa+4 trở tới a[1][1]

pa+5 trở tới a[1][2]

Ví dụ :

Dùng con trỏ để vào số liệu cho mảng hai chiều.

Cách 1 :

```
#include "stdio.h"
```

```
main()
```

```
{
```

```
    float a[2][3],*pa;
```

```
    int i;
```

```
    pa=(float*)a;
```

```
    for (i=0;i<6;++i)
```

```
        scanf("%f",pa+i);
```

```
}
```

Cách 2 :

```
#include "stdio.h"
main()
{
    float a[2][3],*pa;
    int i;
    for (i=0;i<6;++i)
        scanf("%f", (float*)a+i);
}
```

7.4. Kiểu con trỏ, kiểu địa chỉ, các phép toán trên con trỏ :

7.4.1. Kiểu con trỏ và kiểu địa chỉ :

Con trỏ dùng để lưu địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán địa chỉ cho con trỏ chỉ có thể thực hiện được khi kiểu địa chỉ phù hợp với kiểu con trỏ.

Ví dụ theo khai báo :

```
float a[20][30],*pa,(*pm)[30];
```

thì :

pa là con trỏ float

pm là con trỏ kiểu float [30]

a là địa chỉ kiểu float [30]

Vì thế phép gán :

```
pa=a;
```

là không hợp lệ. Nhưng phép gán :

```
pm=a;
```

7.4.2. Các phép toán trên con trỏ:

Có 4 phép toán liên quan đến con trỏ và đại chỉ là :

Phép gán.

Phép tăng giảm địa chỉ.

Phép truy cập bộ nhớ.

Phép so sánh.

Phép gán :

Phép gán chỉ thực hiện với các con trỏ cùng kiểu. Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu như ví dụ sau :

```
int x;  
char *pc;  
pc=(char*)&x;
```

Phép tăng giảm địa chỉ :

Để minh họa chi tiết cho phép toán này, ta xét ví dụ sau :

Các câu lệnh :

```
float x[30],*px;  
px=&x[10];
```

cho con trỏ px là con trỏ float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ 4 byte, nên các phép tăng giảm địa chỉ được thực hiện trên 4 byte. Vì thế :

```
px+i trỏ tới phần tử x[10+i]  
px-i trỏ tới phần tử x[10-i]
```

Xét ví dụ khác :

Giả sử ta khai báo :

```
float b[40][50];
```

Khai báo trên cho ta một mảng b gồm các dòng 50 phần tử thực. Kiểu địa chỉ của b là $50*4=200$ byte.

Do vậy :

```
b trỏ tới đầu dòng thứ nhất ( phần tử b[0][0]).  
b+1 trỏ tới đầu dòng thứ hai ( phần tử b[1][0]).  
.....  
b+i trỏ tới đầu dòng thứ i ( phần tử b[i][0]).
```

Phép truy cập bộ nhớ :

Con trỏ float truy nhập tới 4 byte, con trỏ int truy nhập 2 byte, con trỏ char truy nhập 1 byte. Giả sử ta có cá khai báo :

```
float *pf;  
int *pi;
```

```
char *pc;
```

Khi đó :

Nếu trỏ pi trỏ đến byte thứ 100 thì *pf biểu thị vùng nhớ 4 byte liên tiếp từ byte 100 đến 103.

Nếu trỏ pi trỏ đến byte thứ 100 thì *pi biểu thị vùng nhớ 2 byte liên tiếp từ byte 100 đến 101.

Nếu trỏ pc trỏ đến byte thứ 100 thì *pc biểu thị vùng nhớ 1 byte chính là byte 100.

Phép so sánh :

Cho phép so sánh các con trỏ cùng kiểu, ví dụ nếu p1 và p2 là các con trỏ cùng kiểu thì nếu :

$p1 < p2$ nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới.

$p1 = p2$ nếu địa chỉ p1 trỏ tới cũng là địa chỉ p2 trỏ tới.

$p1 > p2$ nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

Ví dụ :

Ví dụ 1 :

Đoạn chương trình tính tổng các số thực dùng phép so sánh con trỏ :

```
float a[100],*p,*pcuoi,tong=0.0;
int n;
pcuoi=a+n-1; /* Địa chỉ cuối dãy*/
for (p=a;p<=pcuoi;++p)
    s+=*p;
```

Ví dụ 2 :

Dùng con trỏ char để tách các byte của một biến nguyên, ta làm như sau :

Giả sử ta có lệnh :

```
unsigned int n=0xABCD; /* Số nguyên hệ 16*/
char *pc;
pc=(char*)&n;
```

Khi đó :

```
*pc=0xAB (byte thứ nhất của n)
```

```
*pc+1=0xCD (byte thứ hai của n)
```

7.4.3. Con trỏ kiểu void :

Con trỏ kiểu void được khai báo như sau :

```
void *tên_con_trỏ;
```

Đây là con trỏ đặc biệt, con trỏ không kiểu, nó có thể nhận bất kỳ kiểu nào. Chẳng hạn câu lệnh sau là hợp lệ :

```
void *pa;
```

```
float a[20][30];
```

```
pa=a;
```

Con trỏ void thường dùng làm đối để nhận bất kỳ địa chỉ kiểu nào từ tham số thực. Trong thân hàm phải dùng phép chuyển đổi kiểu để chuyển sang dạng địa chỉ cần sử lý.

Chú ý :

Các phép toán tăng giảm địa chỉ, so sánh và truy cập bộ nhớ không dùng được trên con trỏ void.

Ví dụ :

Viết hàm thực hiện công ma trận :

```
void congmt(void *a,void *b,void *c,int N,int N, int m);
```

```
{
```

```
    float *pa,*pb,*pc;
```

```
    int i,j;
```

```
    pa=(float*)a;
```

```
    pb=(float*)b;
```

```
    pc=(float*)c;
```

```
    for (i=1;i<m;++i)
```

```
        for (j=1;j<m;++j)
```

```
            *(pc+i*N+j)=*(pa+i*N+j)+*(pb+i*N+j);
```


}

Vì đối là con trỏ void nên nó có thể nhận được địa chỉ của các ma trận trong lời gọi hàm. Tuy nhiên ta không thể sử dụng trực tiếp các đối con trỏ void trong thân hàm mà phải chuyển kiểu của chúng sang thành float.

7.5. Mảng con trỏ :

Mảng con trỏ là sự mở rộng khái niệm con trỏ. Mảng con trỏ là một mảng mà mỗi phần tử của nó chứa được một địa chỉ nào đó. Cũng giống như con trỏ, mảng con trỏ có nhiều kiểu : Mỗi phần tử của mảng con trỏ kiểu int sẽ chứa được các địa chỉ kiểu int. Tương tự cho các mảng con trỏ của các kiểu khác.

Mảng con trỏ được khai báo theo mẫu :

Kiểu *Tên_mảng_con_trỏ[N];

Trong đó **Kiểu** có thể là int, float, double, char ... còn **Tên_mảng_con_trỏ** là tên của mảng, N là một hằng số nguyên xác định độ lớn của mảng.

Khi gặp khai báo trên, máy sẽ cấp phát N khoảng nhớ liên tiếp cho N phần tử của mảng **Tên_mảng_con_trỏ**.

Ví dụ :

Lệnh :

```
double *pa[100];
```

Khai báo một mảng con trỏ kiểu double gồm 100 phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double.

Chú ý :

Bản thân các mảng con trỏ không dùng để lưu trữ số liệu. Tuy nhiên mảng con trỏ cho phép sử dụng các mảng khác để lưu trữ số liệu một cách có hiệu quả hơn theo cách : chia mảng thành các phần và ghi nhớ địa chỉ đầu của mỗi phần vào một phần tử của mảng con trỏ.

Trước khi sử dụng một mảng con trỏ ta cần gán cho mỗi phần tử của nó một giá trị. Giá trị này phải là giá trị của một biến hoặc một phần tử mảng. Các phần tử của mảng con trỏ kiểu char có thể được khởi đầu bằng các chuỗi ký tự.

Ví dụ :

Xét một tổ lao động có 10 người, mã của mỗi người chính là số thứ tự. Ta lập một hàm để khi biết mã số của nhân viên thì xác định được họ tên của nhân viên đó.

```
#include "stdio.h"
#include "ctype.h"
void tim(int code);
main()
{
    int i;
    tt:printf("\n Tim nguoi co so TT la :");
    scanf("%d",&i);
    tim(i);
    printf("Co tiep tục nữa không C/K : ");
    if (toupper(getch())='C')
        goto tt;
}
void tim(int code);
{
    static char *list[]= {
        "Khong co so thu tu nay "
        " Nguyen Van Toan"
        "Huynh Tuan Nghia"
        "Le Hong Son"
        "Tran Quang Tung"
        "Chu Thanh Tu"
        "Mac Thi Nga"
        "Hoang Hung"
        "Pham Trong Ha"
        "Vu Trung Duc"
        "Mai Trong Quat"
    };
    printf("\n\n Ma so : %d",code);
    printf(" : %s",());
}
}
```

7.6. Con trỏ tới hàm :

7.6.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm :

Ta sẽ trình bày quy tắc khai báo thông qua các ví dụ :

Ví dụ 1:

Câu lệnh :

```
float (*f)(float),(*mf[50])(int);
```

Để khai báo :

- f là con trỏ hàm kiểu float có đối là float
- mf là mảng con trỏ hàm kiểu float có đối kiểu int (có 50 phần tử)

Ví dụ 2:

Câu lệnh :

```
double (*g)(int, double),(*mg[30])(double, float);
```

Để khai báo :

- g là con trỏ hàm kiểu double có các đối kiểu int và double
- mg là mảng con trỏ hàm kiểu double có các đối kiểu double và float (có 30 phần tử)

7.6.2. Tác dụng của con trỏ hàm :

Con trỏ hàm dùng để chứa địa chỉ của hàm. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có ý nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm.

Ví dụ 1:

```
#include "stdio.h"
```

```
double fmax(double x, double y) /* Tính max x,y */
```

```
{
```

```
    return(x>y ? x:y);
```

```
}
```

```

double (*pf)(double,double)=fmax; /*Khai báo và gán tên hàm cho con trỏ hàm */
main() /* Sử dụng con trỏ hàm*/
{
    printf("\n max=%f",pf(5.0,9.6));
}

```

Ví dụ 2:

```

#include "stdio.h"
double fmax(double x, double y ) /* Tính max x,y */
{
    return(x>y ? x:y);
}
double (*pf)(double,double); /* Khai báo con trỏ hàm*/

main() /* Sử dụng con trỏ hàm*/
{
    pf=fmax;
    printf("\n max=%f",pf(5.0,9.6));

}

```

7.6.3. Đối của con trỏ hàm :

C cho phép thiết kế các hàm mà tham số thực trong lời gọi tới nó lại là tên của một hàm khác. Khi đó tham số hình thức tương ứng phải là một con trỏ hàm.

Cách dùng con trỏ hàm trong thân hàm :

Nếu đối được khai báo :

```
double (*f)(double, int);
```

thì trong thân hàm ta có thể dùng các cách viết sau để xác định giá trị của hàm (do con trỏ f trỏ tới) :

```
f(x,m) hoặc (f)(x,m) hoặc (*f)(x,m)
```

ở đây x là biến kiểu double còn m là biến kiểu int.

Ví dụ :

Dùng mảng con trỏ để lập bảng giá trị cho các hàm : $x*x$, $\sin(x)$, $\cos(x)$, $\exp(x)$ và \sqrt{x} . Biến x chạy từ 1.0 đến 10.0 theo bước 0.5

```
#include "stdio.h"
```

```
#include "math.h"
```

```
double bp(double x) /* Hàm tính x*x */
```

```
{  
    return x*x;  
}
```

```
main()
```

```
{  
    int i,j;  
    double x=1.0;  
    typedef double (*ham)(double);  
    ham f[6]; /* Khai báo mảng con trỏ hàm */  
    /* Có thể khai báo như sau double (*f[6])(double)*/  
    f[1]=bp; f[2]=sin; f[3]=cos; f[4]=exp; f[5]=sqrt;  
    /* Gán tên hàm cho các phần tử mảng con trỏ hàm */  
    while (x<=10.0) /* Lập bảng giá trị */  
    {  
        printf("\n");  
        for (j=1;j<=5;++j)  
            printf("%10.2f ",f[j](x));  
        x+=0.5;  
    }  
}
```

Chương 8

Cấu trúc

Cấu trúc là tập hợp của một hoặc nhiều biến, chúng có thể khác kiểu nhau, được nhóm lại dưới một cái tên duy nhất để tiện sử lý. Cấu trúc còn gọi là bản ghi trong một số ngôn ngữ khác, chẳng hạn như PASCAL.

Cấu trúc giúp cho việc tổ chức các dữ liệu phức tạp, đặc biệt trong những chương trình lớn vì trong nhiều tình huống chúng cho phép nhóm các biến có liên quan lại để xử lý như một đơn vị thay vì các thực thể tách biệt.

Một ví dụ được đề cập nhiều đến là cấu trúc phiếu ghi lương, trong đó mỗi nhân viên được mô tả bởi một tập các thuộc tính chẳng hạn như : tên, địa chỉ, lương, phụ cấp vv.. một số trong các thuộc tính này lại có thể là cấu trúc bởi trong nó có thể chứa nhiều thành phần : Tên (Họ, đệm, tên), Địa chỉ (Phố, số nhà) vv.

Trong chương này chúng ta sẽ minh hoạ cách sử dụng của các cấu trúc trong chương trình.

8.1. Kiểu cấu trúc :

Khi xây dựng cấu trúc, ta cần mô tả kiểu của nó. Điều này cũng tương tự như việc phải thiết kế ra một kiểu nhà trước khi ta đi xây dựng những căn nhà thực sự ở các địa điểm khác nhau. Công việc định nghĩa một kiểu cấu trúc bao gồm việc nêu ra tên của kiểu cấu trúc và các thành phần của nó theo mẫu sau :

```
struct tên_kiểu_cấu_trúc
{
    Khai báo các thành phần của cấu trúc    (1)
};
```

Trong đó :

- struct là từ khoá
- tên_kiểu_cấu_trúc là một tên bất kỳ do người lập trình tự đặt theo qui tắc đặt tên nêu ra trong chương 1.

Thành phần của cấu trúc có thể là : biến, mảng, cấu trúc khác đã được định nghĩa trước đó vv..

Ví dụ :

Ví dụ 1:

Đoạn chương trình :

```
struct ngay {
    int ngaythu;
    char thang[12];
    int nam;
};
```

mô tả một kiểu cấu trúc có tên là **ngay** gồm có ba thành phần : Biến nguyên **ngaythu**, mảng **thang**, và biến nguyên **nam**.

Ví dụ 2:

Đoạn chương trình :

```
struct nhancong
{
    char ten[15];
    char diachi[20];
    double bacluong;
    struc ngay ngaysinh;
    struc ngay ngaybatdaucongtac;
};
```

tạo ra kiểu cấu trúc có tên là **nhancong** gồm có năm thành phần. Ba thành phần đầu không có gì cần nói thêm. Chỉ có hai thành phần còn lại là các cấu trúc **ngaysinh** và **ngaybatdaucongtac** được xây dựng theo cấu trúc **ngay** được định nghĩa trong ví dụ 1.

Định nghĩa cấu trúc bằng typedef :

Có thể dùng toán tử typedef để định nghĩa các kiểu cấu trúc **ngay** và **nhancong** ở trên như sau :

```
typedef struct
{
    int ngaythu;
    char thang[12];
```

```

int nam;

    } ngay;
typedef struct
    {

        char ten[15];
        char diachi[20]
        double bacluong;
        struc ngay ngaysinh;
        struc ngay ngaybatdaucongtac;
    } nhancong;

```

8.2. Khai báo theo một kiểu cấu trúc đã định nghĩa :

Xây dựng những cấu trúc thực sự theo các kiểu đã khai báo trước đó. Vấn đề này hoàn toàn giống như việc khai báo các biến và các mảng. Giả sử ta đã có các kiểu cấu trúc **ngay** và **nhancong** như trong mục trên. Khi đó ta khai báo :

Ví dụ 1 :

```
struct ngay ngaydi, ngayden;
```

sẽ cho ta hai cấu trúc với tên là **ngaydi** và **ngayden**. Cả hai cấu trúc đều được xây dựng theo cấu trúc kiểu **ngay**.

Ví dụ 2 :

```
struct nhancong nhom1, nhom2;
```

sẽ cho ta hai cấu trúc với tên là **nhom1** và **nhom2**. Cả hai cấu trúc đều được xây dựng theo cấu trúc kiểu **nhancong**.

Như vậy, một cách tổng quát, việc khai báo cấu trúc được thực hiện theo mẫu sau :

Cách 1 :

```
struct tên_kiểu_cấu_trúc_đã_khai_báo danh_sách_tên_các_cấu_trúc; (2)
```


Chú ý :

Các biến cấu trúc được khai báo theo mẫu trên sẽ được cấp phát bộ nhớ một cách đầy đủ cho tất cả các thành phần của nó.

Việc khai báo có thể thực hiện đồng thời với việc định nghĩa kiểu cấu trúc. Muốn vậy, chỉ cần đặt danh sách tên biến cấu trúc cần khai báo sau dấu } của (*) như trên .

Nói cách khác, để vừa khai báo kiểu vừa khai báo biến ta dùng cách sau :

Cách 2 :

```
struct tên_kiểu_cấu_trúc
{
    Các thành phần của cấu trúc (3)
} danh_sách_tên_các_cấu_trúc;
```

Ví dụ :

Ví dụ 1 :

```
struct ngay
{
    int ngaythu;
    char thang[12];
    int nam;
} ngaydi, ngayden;
```

Ví dụ 2 :

```
struct nhancong
{
    char ten[15];
    char diachi[20];
    double bacluong;
    struc ngay ngaysinh;
    struc ngay ngaybatdaucongtao;
```

```
} nhóm1,nhóm2;
```

Khi vừa định nghĩa kiểu cấu trúc vừa khai báo cấu trúc như trong ví dụ trên, ta không thể không cần đến tên kiểu cấu trúc. Nói cách khác cấu trúc có thể được khai báo theo cách sau :

```
struct  
{  
    Các thành phần của cấu trúc (4)  
} danh_sách_tên_các_cấu_trúc;
```

Ví dụ :

```
struct  
{  
    int ngaythu;  
    char thang[12];  
    int nam;  
} ngaydi,ngayden;
```

Sự khác nhau của các cách khai báo cấu trúc trong (3) và (4) là ở chỗ : Với (3) ta vừa khai báo được một kiểu cấu trúc vừa khai báo được các cấu trúc, và có thể dùng kiểu cấu trúc này để khai báo cho các cấu trúc khác như trong (2), còn (4) chỉ khai báo được các cấu trúc.

Chú ý :

Nếu dùng từ khoá typedef để định nghĩa kiểu cấu trúc như trong mục 8.1 thì khi khai báo các cấu trúc mới ta không cần dùng từ khoá struct, chỉ cần dùng tên kiểu.

Ví dụ như kiểu cấu trúc **ngay** được khai báo bằng typedef trong 8.1 thì khi khai báo các cấu trúc mới là **ngaydi** và **ngayden** có cùng kiểu **ngay** ta dùng dòng lệnh sau :

```
ngay ngaydi,ngayden;
```

8.3. Truy nhập đến các thành phần cấu trúc :

Ta đã khá quen với việc sử dụng các biến, các phần tử của mảng và tên mảng trong các câu lệnh. Trên đây ta cũng đã đề cập đến các thành phần của cấu trúc là biến và mảng. Việc xử lý một cấu trúc bao giờ cũng phải được thực hiện thông qua các thành phần của nó.

Để truy cập đến một thành phần cơ bản (là biến hoặc mảng) của một cấu trúc ta sử dụng một trong các cách viết sau :

```
tên_cấu_trúc.tên_thành_phần  
tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần  
tên_cấu_trúc.tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần  
.....
```

Cách viết thứ nhất như trên được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc. Ví dụ như biến **ngaythu**, biến **nam** và mảng **thang** là các thành phần trực tiếp của các cấu trúc **ngaydi**, **ngayden**. Các biến **bacluong**, các mảng **ten**, **diachi** là các thành phần trực tiếp của các cấu trúc **nhancong**.

Các cách viết còn lại như trên được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc mà bản thân cấu trúc này lại là thành phần của các cấu trúc lớn hơn.

Ví dụ :

Ta xét phép toán trên các thành phần của cấu trúc **nhom1**, **nhom2** :

Câu lệnh :

```
printf("%s",nhom1.ten);
```

sẽ đưa lên màn hình tên của **nhom1**.

Câu lệnh :

```
tongluong=nhom1.bacluong+nhom2.bacluong;
```

sẽ gán tổng lương của **nhom1** và **nhom2** rồi gán cho biến **tongluong**.

Câu lệnh :

```
printf("%d",nhom1.ngaysinh.ten);
```

sẽ đưa lên màn hình ngày sinh của **nhom1**.

Câu lệnh :

```
printf("%d",nhom1.ngaybatdaucongtac.nam);
```

sẽ đưa lên màn hình ngày bắt đầu công tác của **nhom1**.

Chú ý :

- Có thể sử dụng phép toán lấy địa chỉ đối với các thành phần cấu trúc để nhập số liệu trực tiếp vào các thành phần cấu trúc. Ví dụ như ta viết :

```
scanf("%d",&nhom1.ngaybatdaucongtac.nam);
```

Nhưng đối với các thành phần không nguyên, việc làm trên có thể dẫn đến treo máy. Vì thế nên nhập số liệu vào một biến trung gian sau đó mới gán cho thành phần của cấu trúc.

Cách làm như sau :

```
int year;
scanf("%d",&year);
nhom1.ngaybatdaucongtac.nam=year;
```

- Để tránh dài dòng khi làm việc với các thành phần cấu trúc ta có thể dùng lệnh #define. Ví dụ trong câu lệnh scanf ở ví dụ trên, ta có thể viết như sau :

```
#define p nhom1.ngaybatdaucongtac
.....
scanf("%d",&p.nam);
```

Ví dụ :

Giả sử ta lập trình quản lý thông tin cán bộ. Giả sử mỗi dữ liệu của một cán bộ gồm :

- Ngày tháng năm sinh.
- Ngày tháng năm vào cơ quan.
- Bậc lương.

Yêu cầu viết một chương trình để :

- Xây dựng cấu trúc cơ sở dữ liệu cho cán bộ.
- Vào số liệu của một cán bộ.
- Đưa số liệu đó ra máy in.

Chương trình được viết như sau :

```
#include "stdio.h"
typedef struct
{
    int ngay;
    char thang[10];
```

```

        int nam;
    } date;
typedef struct
    {
        date ngaysinh;
        date ngayvaocq;
        float luong;
    } canbo;
main()
{
    canbo p;
    printf("\n Sinh ngay : ");
    scanf("%d",&p.ngaysinh.ngay);
    printf("\n Thang : ");
    scanf("%d",&p.ngaysinh.thang);
    printf("\n Nam : ");
    scanf("%d",&p.ngaysinh.nam);
    printf("\n Vao co quan ngay : ");
    scanf("%d",&p.ngayvaocq.ngay);
    printf("\n Thang : ");
    scanf("%d",&p.ngayvaocq.thang);
    printf("\n Nam : ");
    scanf("%d",&p.ngayvaocq.nam);
    printf("\n Luong : ");
    scanf("%d",&p.luong);
    fprintf(stdprn, "\n Ngay sinh:%d%s%d",p.ngaysinh.ngay,p.ngaysinh.thang,
    p.ngaysinh.nam);
    fprintf(stdprn, "\n Ngay vao co quan:%d%s%d",p.ngayvaocq.ngay,
    p.ngayvaocq.thang,p.ngayvaocq.nam);
    fprintf(stdprn, "\n Luong : %8.2f",p.luong);
}

```

8.4. Mảng cấu trúc :

Như đã đề cập ở các chương trước, khi sử dụng một kiểu giá trị (ví dụ như kiểu int) ta có thể khai báo các biến và các mảng kiểu đó. Ví dụ như khai báo :

```
int a,b,c[10];
```

cho ta hai biến nguyên là a,b và một mảng nguyên c có 10 phần tử.

Hoàn toàn tương tự như vậy : ta có thể sử dụng một kiểu cấu trúc đã mô tả để khai báo các cấu trúc và mảng cấu trúc.

Cách khai báo mảng cấu trúc :

```
struct tên_kiểu_cấu_trúc_đã_định_nghĩa tên_mảng_cấu_trúc[số phần tử của mảng];
```

Ví dụ :

Ví dụ 1 :

Giả sử kiểu cấu trúc **canbo** đã được định nghĩa như mục trên. Khi đó dòng khai báo :

```
struct canbo cb1,cb2,nhom1[10],nhom2[7];
```

sẽ cho :

Hai biến cấu trúc cb1 và cb2.

Hai mảng cấu trúc nhom1 có 10 phần tử và nhom2 có 7 phần tử và mỗi phần tử của hai nhóm này có kiểu **canbo**.

Ví dụ 2 :

Đoạn chương trình sau sẽ tính tổng lương cho các phần tử nhóm 1:

```
double tongluong=0;
for (i=0;i<10;++i)
    tongluong+=nhom1[i].luong;
```

Chú ý :

Không cho phép sử dụng phép toán lấy địa chỉ đối với các thành phần của mảng cấu trúc khác kiểu nguyên. Chẳng hạn không cho phép sử dụng câu lệnh sau :

```
scanf("%f",&nhom1[5].luong);
```

Trong trường hợp này ta dùng biến trung gian.

8.5. Khởi đầu một cấu trúc :

Có thể khởi đầu cho một cấu trúc ngoài, cấu trúc tĩnh, mảng cấu trúc ngoài và mảng cấu trúc tĩnh

8.6. Phép gán cấu trúc :

Có thể thực hiện phép gán trên các biến và phần tử mảng cấu trúc cùng kiểu như sau :

- Gán hai biến cấu trúc cho nhau
- Gán biến cấu trúc cho phần tử mảng cấu trúc
- Gán phần tử mảng cấu trúc cho biến cấu trúc
- Gán hai phần tử mảng cấu trúc cho nhau

Mỗi một phép gán trên tương đương với một dãy phép gán các thành phần tương ứng.

Ví dụ :

Đoạn chương trình sau minh họa cách dùng phép gán cấu trúc để sắp xếp n thí sinh theo thứ tự giảm của tổng điểm :

```
struct thisinh
{
    char ht[25];
    float td;
} tg,ts[100];
for (i=1;i<=n-1;++i)
for (j=1;j<=n;+j)
    if (ts[i].td<ts[j].td)
    {
        tg=ts[i];
        ts[i]=ts[j];
        ts[j]=tg;
    }
```

8.7. Con trỏ cấu trúc và địa chỉ cấu trúc :

8.7.1. Con trỏ và địa chỉ :

Ta xét ví dụ sau :

```
struct ngay
```

```

    {
        int ngaythu;
        char thang[10];
        int nam;
    };
struct nhancong
    {
        char ten[20];
        char diachi[25];
        double bacluong;
        struct ngay ngaysinh;
    };

```

Nếu khai báo :

```
struct nhancong *p,*p1,*p2,nc1,nc2,ds[100];
```

ta có :

- p, p1, p2 là con trỏ cấu trúc
- nc1, nc2 là các biến cấu trúc
- ds là mảng cấu trúc

Con trỏ cấu trúc dùng để lưu trữ địa chỉ của biến cấu trúc và mảng cấu trúc.

Ví dụ :

```

p1=&nc1;    /* Gửi địa chỉ nc1 vào p1 */
p2=&ds[4];  /* Gửi địa chỉ ds[4] vào p2 */
p=ds;      /* Gửi địa chỉ ds[0] vào p */

```

8.7.2. Truy nhập qua con trỏ:

Có thể truy nhập đến các thành phần thông qua con trỏ theo một trong hai cách sau :

Cách một :

Tên_con_trỏ->Tên_thành_phần

Cách hai :

(*Tên_con_trỏ).Tên_thành_phần

Ví dụ :

```
nc1.ngaysinh.nam
p1-> ngaysinh.nam
ds[4].ngaysinh.thang
(*p2). ngaysinh.thang
```

8.7.3. Phép gán qua con trỏ:

Giả sử ta gán :

```
p1=&nc1;
p2=&ds[4];
```

Khi đó có thể dùng :

```
*p1 thay cho nc1
*p2 thay cho ds[4]
```

Tức là viết:

```
ds[5]=nc1;
ds[4]=nc2;
```

Tương đương với :

```
ds[5]=*p1;
*p2=nc2;
```

8.7.4. Phép cộng địa chỉ :

Sau các phép gán :

```
p=ds;
p2=&ds[4];
```

thì p trỏ tới ds[[0]] và p2 trỏ tới ds[4]. Ta có thể dùng các phép cộng, trừ địa chỉ để làm cho p và p2 trỏ tới các thành phần bất kỳ nào khác.

Ví dụ :

Sau các lệnh :

```
p=p+10;
p2=p2-4;
```

thì p trỏ tới ds[10] còn p2 trỏ tới ds[0]

8.7.5. Con trỏ và mảng :

Giả sử con trỏ p trỏ tới đầu mảng ds, khi đó :

- Ta có thể truy nhập tới các thành phần cấu trúc bằng các cách sau :

+ ds[i].thành_phần ds[i].ngaysinh.nam
+ p[i].thành_phần p[i].ngaysinh.nam
+ (p+i)->thành_phần (p+i)->ngaysinh.nam

- Khi ta sử dụng cả cấu trúc thì các cách viết sau là tương đương :

ds[i] p[i] *(p+i)

8.8. Cấu trúc tự trỏ và danh sách liên kết :

Khi ta lập một chương trình quản lý mà bản thân số biến (cấu trúc) chưa được biết trước, nếu ta sử dụng mảng (cấp phát bộ nhớ tĩnh) thì ta phải sử dụng số các phần tử là tối đa. Như vậy sẽ có rất nhiều vùng nhớ được cấp phát mà không bao giờ dùng đến. Lúc đó ta có cách để cấp phát bộ nhớ động. Số vùng nhớ cấp ra đủ số biến cần dùng.

Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa gọi là cấu trúc tự trỏ.

Ví dụ :

Các cách để định nghĩa cấu trúc tự trỏ person:

Cách 1 :

```
typedef struct pp
{
    char ht[20];
    char qq[25];
    int tuoi;
    struct pp *tiếp;
} person;
```

Cách 2 :

```
typedef struct pp person
```

```

struct pp
{
    char ht[20];
    char qq[25];
    int tuoi;
    person *tiep;
};

```

Cách 3 :

```

struct pp
{
    char ht[20];
    char qq[25];
    int tuoi;
    struct pp *tiep;
};

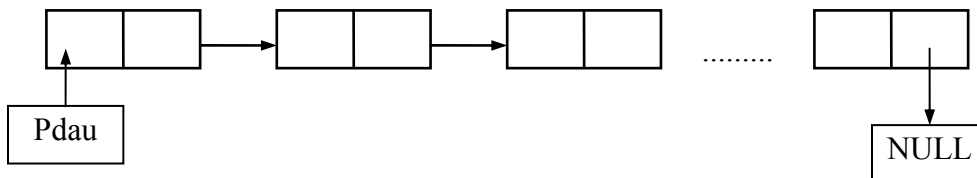
```

typedef pp person;

Cấu trúc tự trỏ được dùng để xây dựng danh sách liên kết (móc nối), đó là một nhóm các cấu trúc có tính chất sau : (Móc nối theo chiều thuận).

- Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó.
- Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp sau của danh sách.
- Cấu trúc cuối chứa hằng NULL.

Ví dụ :



Với danh sách này, ta có thể lần lượt từ cấu trúc đầu đến cấu trúc cuối theo chiều từ trên xuống dưới.

Nhóm cấu trúc móc nối theo chiều ngược có tính chất sau :

- Biết địa chỉ cấu trúc cuối.
- Trong mỗi cấu trúc (trừ cấu trúc đầu) đều chứa địa chỉ của cấu trúc trước.

- Cấu trúc đầu chứa hằng NULL.

Với danh sách này, ta có thể lần lượt từ cấu trúc cuối lên cấu trúc đầu theo chiều từ dưới lên trên. Ngoài ra, ta có thể xây dựng các danh sách mà mỗi phần tử chứa hai địa chỉ của cấu trúc trước và cấu trúc sau. Với loại danh sách này, ta có thể truy nhập theo cả hai chiều trên.

Khi làm việc với danh sách móc nối, ta thường phải tiến hành các công việc sau sau :

(Giả sử ta có con trỏ **p**, trỏ **pdau** chỉ cấu trúc đầu của danh sách, con trỏ **tiếp** là thành phần con trỏ của cấu trúc)

Tạo danh sách mới :

- Cấp phát bộ nhớ cho một cấu trúc
- Nhập một biến cấu trúc vào vùng nhớ vừa cấp
- Gán địa chỉ của cấu trúc sau cho thành phần con trỏ của cấu trúc trước

Duyệt qua tất cả các phần tử của danh sách :

- Đưa trỏ p về trỏ cùng cấu trúc với pdau bằng lệnh :
 $p=pdau$
- Để chuyển tiếp đến người tiếp theo ta dùng lệnh :
 $p=p->tiếp$
- Dấu hiệu để biết đang xét cấu trúc cuối cùng của danh sách là :

$p->tiếp==NULL$

Loại một cấu trúc ra khỏi danh sách :

- Lưu trữ địa chỉ của cấu trúc cần loại vào một con trỏ (Để giải phóng bộ nhớ của cấu trúc này)
- Sửa để cấu trúc trước đó có địa chỉ của cấu trúc cần loại
- Giải phóng bộ nhớ cấu trúc cần loại

Bổ xung hoặc chèn một cấu trúc vào danh sách:

- Cấp phát bộ nhớ và nhập bổ xung
- Sửa thành phần con trỏ trong các cấu trúc có liên quan để đảm bảo mỗi cấu trúc chứa địa chỉ của cấu trúc tiếp theo

Hàm cấp phát bộ nhớ :

```
void *malloc(kichthuc_t kichthuc);
```

Hàm lấy trong thư viện alloc.h hoặc stdlib.h.

kichthuc tính bằng số by te. Hàm sẽ đưa con trỏ về vị trí ô nhớ vừa được cấp hoặc về NULL nếu không đủ bộ nhớ cần thiết. Nếu **kichthuc** == 0 thì nó trả về NULL.

Ví dụ :

```
#include "stdio.h"
#include "string.h"
#include "alloc.h"
#include "process.h"
int main()
{
    char *str;
    /* Cấp phát bộ nhớ cho xâu ký tự */
    if ((str = malloc(10)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1); /* Kết thúc chương trình nếu thiếu bộ nhớ */
    }
    /* copy "Hello" vào xâu */
    strcpy(str, "Hello");
    /* Hiện thị xâu */
    printf("String is %s\n", str);
    /* Giải phóng bộ nhớ */
    free(str);
    return 0;
}
```

Ví dụ :

Tạo một danh sách liên kết. Các biến cấu trúc gồm các trường : Họ tên, Quê quán, tuổi, và một trường con trỏ là Tiếp.

Móc nối theo chiều thuận (Vào trước ra trước FIFO first in first out):

```
#include "stdio.h"
```

```

#include "alloc.h"
#include "conio.h"
#include "string.h"
typedef struct pp
    {
        char ht[25];
        char qq[20];
        int tuoi;
        struct pp *tiep;
    } nhansu;
main()
    {
        char tt;
        nhansu *pdau,*pcuoi,*p;
        char tam[10];
        clrscr();
        pdau=NULL;
        do
            {
                p=(nhansu*)malloc(sizeof(nhansu));
                printf("\n Ho ten : ");
                gets(p->ht);
                printf(" Que quan : ");
                gets(p->qq);
                printf(" Tuoi: ");
                gets(tam);
                p->tuoi=atoi(tam);
                if (pdau==NULL)
                    {
                        pdau=p;
                        pcuoi=p;
                        p->tiep=NULL;
                    }
            }
    }

```

```

        else
            {
                pcuoi->tiemp=p;
            pcuoi=p;
            p->tiemp=NULL;
            }
        printf("\nBam phim bat ky de tiep tuc, ESC de dung");
        tt=getch();
    } while(tt!=27) ;
    /* Đưa danh sách liên kết ra màn hình, trở pđau tro */
    printf("\n Danh sach nhu sau :\n");
    p=pđau;
    while (p!=NULL)
        {
            printf("\n Ho ten: %25s Que : %20s Tuoi :
            %d",(*p).ht,(*p).qq,(*p).tuoi);
            p=p->tiemp;
        }
    getch();
}

```

Móc nối theo chiều ngược (Vào sau ra trước LIFO last in first out):

```

#include "stdio.h"
#include "alloc.h"
#include "conio.h"
#include "string.h"
typedef struct pp
    {
        char ht[25];
        char qq[20];
        int tuoi;
        struct pp *tiemp;
    } nhansu;
main()

```

```

{
    char tt;
    nhansu *pdau,*pcuoi,*p;
    char tam[10];
    clrscr();
    pdau=NULL;
    do
        {
p=(nhansu*)malloc(sizeof(nhansu));
printf("\n Ho ten : ");
    gets(p->ht);
printf(" Que quan : ");
    gets(p->qq);
printf(" Tuoi: ");
    gets(tam);
    p->tuoi=atoi(tam);
if (pdau==NULL)
    {
    pdau=p;
    pcuoi=p;
        p->tiemp=NULL;
        }
    else
        {
        p->tiemp=pcuoi;
    pcuoi=p;
        }
printf("\nBam phim bat ky de tiep tuc, ESC de dung");
    tt=getch();
} while(tt!=27) ;
    /* Đưa danh sách liên kết ra màn hình, trở pdau trở */
printf("\n Danh sach nhu sau :\n");
p=pcuoi;

```



```
while (p!=NULL)
    {
        printf("\n Ho ten: %25s Que : %20s Tuoi :
%d",(*p).ht,(*p).qq,(*p).tuoi);
        p=p->tiep;
    }
getch();
}
```

Chương 9

tập tin - file

9.1. Khái niệm về tập tin :

Tập tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau và có cùng một kiểu được nhóm lại với nhau thành một dãy. Chúng thường được chứa trong một thiết bị nhớ ngoài của máy tính (đĩa mềm, đĩa cứng...) dưới một cái tên nào đó.

Tên tiếng Anh của tệp là **file**, nó được dùng để chỉ ra một hộp đựng các phiếu hay thẻ ghi của thư viện. Một hình ảnh rõ nét giúp ta hình dung ra tệp là tủ phiếu của thư viện. Một hộp có nhiều phiếu giống nhau về hình thức và tổ chức, song lại khác nhau về nội dung. ở đây, tủ phiếu là tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng hoặc một đĩa mềm đóng vai trò chiếc tủ (để chứa nhiều tệp).

Tệp được chứa trong bộ nhớ ngoài, điều đó có nghĩa là tệp được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện. Chính vì lý do trên, chỉ những dữ liệu nào cần lưu trữ (như hồ sơ chẳng hạn) thì ta nên dùng đến tệp.

Tệp là một kiểu dữ liệu có cấu trúc. Định nghĩa tệp có phần nào giống mảng ở chỗ chúng đều là tập hợp của các phần tử dữ liệu cùng kiểu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong định nghĩa.

Trong C, các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia làm hai nhóm : nhóm 1 và nhóm 2. Các hàm cấp 1 là các hàm nhập / xuất hệ thống, chúng thực hiện việc đọc ghi như DOS. Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1 nên trong các chương trình viết trong C, các hàm cấp 2 hay được sử dụng hơn.

Một tệp tin dù được xây dựng bằng cách nào đi nữa cũng chỉ đơn giản là một dãy các byte ghi trên đĩa (có giá trị từ 0 đến 255). Số byte của dãy chính là độ dài của tệp.

Có hai kiểu nhập xuất dữ liệu lên tệp : Nhập xuất nhị phân và nhập xuất văn bản.

Nhập xuất nhị phân :

- Dữ liệu ghi lên tệp theo các byte nhị phân như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị biến đổi.
- Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (được định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác 0.

Nhập xuất văn bản:

- Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.
- Mã chuyển dòng :
 Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF
 Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF

Mã kết thúc tệp :

Trong khi đọc, nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (bằng -1) và hàm feof(fp) cho giá trị khác 0 (bằng 1).

9.2. Khai báo sử dụng tệp - một số hàm thường dùng khi thao tác trên tệp :

9.2.1. Khai báo sử dụng tệp :

Để khai báo sử dụng tệp, ta dùng lệnh sau :

```
FILE biến_con_trò_tệp;
```

Trong đó biến_con_trò_tệp có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phẩy (dấu ,).

Ví dụ :

```
FILE *vb, *np; /* Khai báo hai biến con trò tệp */
```

9.2.2. Mở tệp - hàm fopen :

Cấu trúc ngữ pháp của hàm :

```
FILE *fopen(const char *tên_tệp, const char *kiểu);
```

Nguyên hàm trong : stdio.h .

Trong đó :

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Công dụng :

Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp hai sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng sau chỉ ra các giá trị của kiểu :

Tên kiểu	ý nghĩa
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w" "wt"	Mở một tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a" "at"	Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"ab"	Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w+" "w+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

Chú ý :

Trong các kiểu đọc ghi, ta nên làm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc ngược lại. Ta sẽ đề cập đến các hàm với tính năng xoá sau này.

Ví dụ :

```
f=fopen("TEPNP","wb");
```

9.2.3. Đóng tệp - hàm fclose :

Cấu trúc ngữ pháp của hàm :

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

Nguyên hàm trong : stdio.h .

Trong đó :

fp là con trỏ ứng với tệp cần đóng.

Công dụng :

Hàm dùng để đóng tệp khi kết thúc các thao tác trên nó. Khi đóng tệp, máy thực hiện các công việc sau :

- Khi đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
- Khi đang đọc dữ liệu thì máy sẽ xoá vùng đệm
- Giải phóng biến trỏ tệp.
- Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

```
fclose(f);
```

9.2.4. Đóng tất cả các tệp đang mở- hàm fcloseall :

Cấu trúc ngữ pháp của hàm :

```
int fcloseall(void);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Hàm dùng để đóng tất cả các tệp đang mở . Nếu lệnh thành công, hàm sẽ cho giá trị bằng số là số tệp được đóng, trái lại nó cho hàm EOF.

Ví dụ :

```
fcloseall();
```

9.2.5. Làm sạch vùng đệm - hàm fflush :

Cấu trúc ngữ pháp của hàm :

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

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tệp fp. Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

```
fflush(f);
```

9.2.6. Làm sạch vùng đệm của các tệp đang mở - hàm fflushall :

Cấu trúc ngữ pháp của hàm :

```
int fflushall(void);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tất cả các tệp đang mở. Nếu lệnh thành công, hàm sẽ cho giá trị bằng số các tệp đang mở, trái lại nó cho hàm EOF.

Ví dụ :

```
fflushall();
```

9.2.7. Kiểm tra lỗi file - hàm ferror :

Cấu trúc ngữ pháp của hàm :

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

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra lỗi khi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không có lỗi, trái lại hàm cho giá trị khác 0.

9.2.8. Kiểm tra cuối tệp - hàm feof :**Cấu trúc ngữ pháp của hàm :**

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

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

9.2.9. Truy nhập ngẫu nhiên - các hàm di chuyển con trỏ chỉ vị :**9.2.7.1. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind :****Cấu trúc ngữ pháp :**

```
void rewind(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu.

Ví dụ :

```
rewind(f);
```

9.2.9.2. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek :**Cấu trúc ngữ pháp :**

```
int fseek(FILE *fp, long sb, int xp);
```

Nguyên hàm trong : `stdio.h` .

Trong đó

`fp` là con trỏ tệp.

`sb` là số byte cần di chuyển.

`xp` cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.

`xp` có thể nhận các giá trị sau :

`xp=SEEK_SET` hay 0 : Xuất phát từ đầu tệp.

`xp=SEEK_CUR` hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

`xp=SEEK_END` hay 2 : Xuất phát từ cuối tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp `fp` về vị trí xác định bởi `xp` qua một số byte xác định bằng giá trị tuyệt đối của `sb`. Chiều di chuyển là về cuối tệp nếu `sb` dương, trái lại nó sẽ di chuyển về đầu tệp. Khi thành công, hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

Chú ý :

Không nên dùng `fseek` trên tệp tin văn bản, do sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

Ví dụ :

```
fseek(stream, SEEK_SET, 0);
```

9.2.9.3. Vị trí hiện tại của con trỏ chỉ vị - Hàm `ftell` :

Cấu trúc ngữ pháp :

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

Nguyên hàm trong : `stdio.h` .

Trong đó

`fp` là con trỏ tệp.

Công dụng :

Hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp **fp**) khi thành công.
Số thứ tự tính từ 0. Trái lại hàm cho giá trị -1L.

Ví dụ :

Sau lệnh `fseek(fp,0,SEEK_END);`

`ftell(fp)` cho giá trị 3.

Sau lệnh `fseek(fp,-1,SEEK_END);`

`ftell(fp)` cho giá trị 2.

9.2.10. Ghi các mẫu tin lên tệp - hàm `fwrite` :

Cấu trúc ngữ pháp của hàm :

```
int fwrite(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : `stdio.h` .

Trong đó :

ptr là con trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm ghi `n` mẫu tin kích thước **size** byte từ vùng nhớ **ptr** lên tệp **fp**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự ghi được.

Ví dụ :

```
#include "stdio.h"
```

```
struct mystruct
```

```
{
```

```
int i;
```

```
char ch;
```

```
};
```

```
main()
```

```
{
```

```
FILE *stream;
```

```

struct mystruct s;
stream = fopen("TEST.TXT", "wb") /* Mở tệp TEST.TXT */
s.i = 0;
s.ch = 'A';
fwrite(&s, sizeof(s), 1, stream); /* Viết cấu trúc vào tệp */
fclose(stream); /* Đóng tệp */
return 0;
}

```

9.2.11. Đọc các mẫu tin từ tệp - hàm fread :

Cấu trúc ngữ pháp của hàm :

```
int fread(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : **stdio.h** .

Trong đó :

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm đọc n mẫu tin kích thước **size** byte từ tệp **fp** lên lên vùng nhớ **ptr**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ :

```

#include "string.h"
#include "stdio.h"
main()
{
FILE *stream;
char msg[] = "Kiểm tra";
char buf[20];
stream = fopen("DUMMY.FIL", "w+");

```

```

/* Viết vài dữ liệu lên tệp */
fwrite(msg, strlen(msg)+1, 1, stream);
/* Tìm điểm đầu của file */
fseek(stream, SEEK_SET, 0);
/* Đọc số liệu và hiển thị */
fread(buf, strlen(msg)+1, 1, stream);
printf("%s\n", buf);
fclose(stream);
return 0;
}

```

9.2.10. Nhập xuất ký tự :

9.2.10.1. Các hàm putc và fputc :

Cấu trúc ngữ pháp :

```

int putc(int ch, FILE *fp);
int fputc(int ch, FILE *fp);

```

Nguyên hàm trong : `stdio.h` .

Trong đó :

ch là một giá trị nguyên
fp là một con trỏ tệp.

Công dụng :

Hàm ghi lên tệp fp một ký tự có mã bằng
`m=ch % 256.`

ch được xem là một giá trị nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại cho EOF

Ví dụ :

```

#include "stdio.h"
main()
{
char msg[] = "Hello world\n";

```

```

int i = 0;
while (msg[i])
    putchar(msg[i++], stdout); /* stdout thiết bị ra chuẩn - Màn hình*/
return 0;
}

```

9.2.12.2. Các hàm getc và fgetc :

Cấu trúc ngữ pháp :

```

int getc(FILE *fp);
int fgetc(FILE *fp);

```

Nguyên hàm trong : stdio.h .

Trong đó :

fp là một con trỏ tệp.

Công dụng :

Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ :

```

#include "string.h"
#include "stdio.h"
#include "conio.h"
main()
{
    FILE *stream;
    char string[] = "Kiem tra";
    char ch;
    /* Mở tệp để cập nhật*/
    stream = fopen("DUMMY.FIL", "w+");
    /*Viết một xâu ký tự vào tệp */

```

```

fwrite(string, strlen(string), 1, stream);
/* Tìm vị trí đầu của tệp */
fseek(stream, 0, SEEK_SET);
do
{
    /* Đọc một ký tự từ tệp */
    ch = fgetc(stream);
    /* Hiện thị ký tự */
    putchar(ch);
} while (ch != EOF);
fclose(stream);
return 0;
}

```

9.2.13. Xoá tệp - hàm unlink:

Cấu trúc ngữ pháp :

```
int unlink(const char *tên_tệp)
```

Nguyên hàm trong : dos.h, io.h, stdio.h .

Trong đó

tên_tệp là tên của tệp cần xoá.

Công dụng :

Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ :

```

#include <stdio.h>
#include <io.h>
int main(void)
{
    FILE *fp = fopen("junk.jnk","w");
    int status;
    fprintf(fp,"junk");
    status = access("junk.jnk",0);
}

```

```
if (status == 0)
    printf("Tập tồn tại\n");
else
    printf("Tập không tồn tại\n");
fclose(fp);
unlink("junk.jnk");
status = access("junk.jnk",0);
if (status == 0)
    printf("Tập tồn tại\n");
else
    printf("Tập không tồn tại\n");
return 0;
}
```

Chương 10

Đồ hoạ

Chương này sẽ giới thiệu các hàm và thủ tục để khởi động hệ đồ hoạ, vẽ các đường và hình cơ bản như hình tròn, cung elip, hình quạt, đường gãy khúc, đa giác, đường thẳng, hình chữ nhật, hình hộp chữ nhật....

Các hàm và thủ tục đồ hoạ được khai báo trong file graphics.h.

10.1. Khởi động đồ hoạ :

Mục đích của việc khởi động hệ thống đồ hoạ là xác định thiết bị đồ hoạ (màn hình) và mode đồ hoạ sẽ sử dụng trong chương trình. Để làm công việc này, ta có hàm sau :

```
void initgraph(int *graphdriver,int graphmode,char *driverpath);
```

Trong đó :

- driverpath là xâu ký tự chỉ đường dẫn đến thư mục chứa các tập tin điều khiển đồ hoạ.
- graphdriver cho biết màn hình đồ hoạ sử dụng trong chương trình.
- graphmode cho biết mode đồ hoạ sử dụng trong chương trình.

Bảng dưới đây cho các giá trị khả dĩ của graphdriver và graphmode :

graphdriver	graphmode	Độ phân giải
DETECT (0)		
CGA (1)	CGAC0 (0)	320x200
	CGAC1 (1)	320x200
	CGAC2 (2)	320x200
	CGAC3 (3)	320x200
	CGAHi (4)	640x200
MCGA (2)	MCGA0 (0)	320x200
	MCGA1 (1)	320x200
	MCGA2 (2)	320x200
	MCGA3 (3)	320x200
	MCGAMed (4)	640x200
	MCGAHi (5)	640x480
EGA (3)	EGAL0 (0)	640x200

	EGAHi (1)	640x350
EGA64 (4)	EGA64LO (0)	640x200
	EGA64Hi (1)	640x350
EGAMONO (5)	EGAMONOH (0)	640x350
VGA (9)	VGALO (0)	640x200
	VGAMED (1)	640x350
	VGAHI (2)	640x480
HERCMONO (7)	HERCMONOH	720x348
ATT400 (8)	ATT400C0 (0)	320x200
	ATT400C1 (1)	320x200
	ATT400C2 (2)	320x200
	ATT400C3 (3)	320x200
	ATT400MED (4)	640x400
	ATT400HI (5)	640x400
PC3270 (10)	PC3270HI (0)	720x350
IBM8514 (6)	PC3270LO (0)	640x480 256 màu
	PC3270HI (1)	1024x768 256 màu

Chú ý :

- Bảng trên cho ta các hằng và giá trị của chúng mà các biến graphdriver và graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0 vv...

Khi lập trình ta có thể thay thế vào vị trí tương ứng của chúng trong hàm tên hằng hoặc giá trị của hằng đó.

Ví dụ :

Giả sử máy tính có màn hình VGA, các tập tin đồ họa chứa trong thư mục C:\TC \BGI, khi đó ta khởi động hệ thống đồ họa như sau :

```
#include "graphics.h"
main()
{
    int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
    initgraph(&mh,&mode,"C:\\TC\\BGI");
```



```
/* Vì kí tự \ trong C là kí tự đặc biệt nên ta phải gấp đôi nó */
```

```
}
```

- Bảng trên còn cho thấy độ phân giải còn phụ thuộc cả vào màn hình và mode. Ví dụ như trong màn hình EGA nếu dùng EGALo thì độ phân giải là 640x200 (Hàm getmaxx() cho giá trị cực đại của số điểm theo chiều ngang của màn hình. Với màn hình EGA trên : 639, Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình. Với màn hình EGA trên : 199).
- Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver bằng DETECT hay giá trị 0. Khi đó, kết quả của initgraph sẽ là :

Kiểu màn hình đang sử dụng được phát hiện, giá trị của nó được gán cho biến graphdriver.

Mode đồ hoạ ở độ phân giải cao nhất ứng với màn hình đang sử dụng cũng được phát hiện và trị số của nó được gán cho biến graphmode.

Như vậy dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ hoạ với màn hình hiện có theo mode có độ phân giải cao nhất mà còn giúp ta xác định kiểu màn hình đang sử dụng.

Ví dụ :

Chương trình dưới đây xác định kiểu màn hình đang sử dụng :

```
#include "graphics.h"  
#include "stdio.h"  
main()  
{  
    int mh=0, mode;  
    initgraph(&mh,&mode,"C:\\TC\\BGI");  
    printf("\n Gia tri so cua man hinh la : %d",mh);  
    printf("\n Gia tri so mode do hoa la : %d",mode);  
    closegraph();  
}
```

- Nếu chuỗi dùng để xác định driverpath là chuỗi rỗng thì chương trình dịch sẽ tìm kiếm các file điều khiển đồ hoạ trên thư mục chủ (Thư mục hiện thời).

10.2. Các hàm đồ hoạ :

10.2.1. Mẫu và màu :

- **Đặt màu nền :**

Để đặt màu cho nền ta dùng thủ tục sau :

```
void setbkcolor(int màu);
```

- **Đặt màu đường vẽ :**

Để đặt màu vẽ đường ta dùng thủ tục sau :

```
void setcolor(int màu);
```

- **Đặt mẫu (kiểu) tô và màu tô :**

Để đặt mẫu (kiểu) tô và màu tô ta dùng thủ tục sau :

```
void setfillstyle(int mẫu, int màu);
```

Trong cả ba trường hợp **màu** xác định mã của màu.

Các giá trị khả dĩ của **màu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của màu

Tên hằng	Giá trị số	Màu hiển thị
BLACK	0	Đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIGHTGRAY	7	Xám nhạt
DARKGRAY	8	Xám đậm
LIGHTBLUE	9	Xanh xa trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	16	Trắng

Các giá trị khả dĩ của **mẫu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của mẫu

Tên hằng	Giá trị số	Kiểu mẫu tô
EMPTY_FILL	0	Tô bằng màu nền
SOLID_FILL	1	Tô bằng đường liền nét
LINE_FILL	2	Tô bằng đường -----
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

Chọn giải màu :

Để thay đổi giải màu đã được định nghĩa trong bảng trên, ta sử dụng hàm :

```
void setpalette(int số_thứ_tự_màu, int màu );
```

Ví dụ :

Câu lệnh :

```
setpalette(0,lightcyan);
```

biến màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng.

- **Lấy giải màu hiện thời :**

+ Hàm getcolor trả về màu đã xác định bằng thủ tục setcolor ngay trước nó.

+ Hàm getbkcolor trả về màu đã xác định bằng hàm setbkcolor ngay trước nó.

10.2.2. Vẽ và tô màu :

Có thể chia các đường và hình thành bốn nhóm chính :

- Cung tròn và hình tròn.
- Đường gấp khúc và đa giác.
- Đường thẳng.

- Hình chữ nhật.

10.2.2.1. Cung tròn và đường tròn :

Nhóm này bao gồm : Cung tròn, đường tròn, cung elip và hình quạt.

- **Cung tròn :**

Để vẽ một cung tròn ta dùng hàm :

```
void arc(int x, int y, int gd, int gc, int r);
```

Trong đó :

(x,y) là toạ độ tâm cung tròn.

gd là góc đầu cung tròn(0 đến 360 độ).

gc là góc cuối cung tròn (gd đến 360 độ).

r là bán kính cung tròn .

Ví dụ :

Vẽ một cung tròn có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán kính 30.

```
arc(100,50,0,180,30);
```

- **Đường tròn :**

Để vẽ đường tròn ta dùng hàm :

```
void circle(int x, int y, int r);
```

Trong đó :

(x,y) là toạ độ tâm cung tròn.

r là bán kính đường tròn.

Ví dụ :

Vẽ một đường tròn có tâm tại (100,50) và bán kính 30.

```
circle(100,50,30);
```

- **Cung elip**

Để vẽ một cung elip ta dùng hàm :

```
void ellipse(int x, int y, int gd, int gc, int xr, int yr);
```

Trong đó :

(x,y) là toạ độ tâm cung elip.
gd là góc đầu cung tròn(0 đến 360 độ).
gc là góc cuối cung tròn (gd đến 360 độ).
xr là bán trục nằm ngang.
yr là bán trục thẳng đứng.

Ví dụ :

Vẽ một cung elip có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán trục ngang 30, bán trục đứng là 20.

```
ellipse(100,50,0,180,30,20);
```

• **Hình quạt :**

Để vẽ và tô màu một hình quạt ta dùng hàm :

```
void pieslice(int x, int y, int gd, int gc, int r);
```

Trong đó :

(x,y) là toạ độ tâm hình quạt.
gd là góc đầu hình quạt (0 đến 360 độ).
gc là góc cuối hình quạt (gd đến 360 độ).
r là bán kính hình quạt .

Ví dụ :

Chương trình dưới đây sẽ vẽ một cung tròn ở góc phần tư thứ nhất, một cung elip ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
# include "graphics.h"  
#include "stdio.h"  
#include "conio.h"  
main()  
{  
    int md=0,mode;  
    initgraph(&md,&mode,"C:\\TC\\BGI");  
    setbkcolor(BLUE);  
    setcolor(YELLOW);  
    setfillstyle(SOLID_FILL,RED);;
```

```

arc(160,50,0,90,45);
circle(160,150,45);
pieslice(480,150,90,360,45);
getch();
closegraph();
}

```

10.2.3. Vẽ đường gấp khúc và đa giác :

- **Vẽ đường gấp khúc :**

Muốn vẽ đường gấp khúc đi qua n điểm : (x_1, y_1) , (x_2, y_2) , ..., (x_n, y_n) thì trước hết ta phải gán các tọa độ (x_i, y_i) cho một mảng a kiểu int nào đó theo nguyên tắc sau :

```

Tọa độ x1 gán cho a[0]
Tọa độ y1 gán cho a[1]
Tọa độ x2 gán cho a[2]
Tọa độ y2 gán cho a[3]
....
Tọa độ xn gán cho a[2n-2]
Tọa độ yn gán cho a[2n-1]

```

Sau đó gọi hàm :

```
drawpoly(n,a);
```

Nếu điểm cuối cùng (x_n, y_n) trùng với điểm đầu (x_1, y_1) thì ta nhận được một đường gấp khúc khép kín.

- **Tô màu đa giác :**

Giả sử ta có a là mảng đã đề cập đến trong mục trên, khi đó ta gọi hàm :

```
fillpoly(n,a);
```

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm (x_1, y_1) , (x_2, y_2) , ..., (x_n, y_n)

Ví dụ :

Vẽ một đường gấp khúc và hai đường tam giác.

```
#include "graphics.h"
```

```
#include "stdio.h"
```

```
#include "conio.h"
```

```

int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
{
    int md=0,mode;
    initgraph(&md,&mode,"C:\\TC\\BGI");
    setbkcolor(CYAN);
    setcolor(YELLOW);
    setfillstyle(SOLID_FILL,MAGENTA);
    drawpoly(3,poly1);
    fillpoly(3,poly2);
    fillpoly(4,poly3);
    getch();
    closegraph();
}

```

- **Vẽ đường thẳng :**

Để vẽ đường thẳng nối hai điểm bất kỳ có tọa độ (x_1,y_1) và (x_2,y_2) ta sử dụng hàm sau :

```
void line(int x1, int y1, int x2, int y2);
```

Con chạy đồ họa giữ nguyên vị trí.

Để vẽ đường thẳng nối từ điểm con chạy đồ họa đến một điểm bất kỳ có tọa độ (x,y) ta sử dụng hàm sau :

```
void lineto(int x, int y);
```

Con chạy sẽ chuyển đến vị trí (x,y) .

Để vẽ một đường thẳng từ vị trí con chạy hiện tại (giả sử là điểm x,y) đến điểm có tọa độ $(x+dx,y+dy)$ ta sử dụng hàm sau :

```
void linerel(int dx, int dy);
```

Con chạy sẽ chuyển đến vị trí $(x+dx,y+dy)$.

- **Di chuyển con chạy đồ họa :**

Để di chuyển con chạy đến vị trí (x,y) , ta sử dụng hàm sau :

```
void moveto(int x, int y);
```

- **Chọn kiểu đường :**

Hàm void setlinestyle(int kiểu_đường, int mẫu, int độ_dày);
 tác động đến nét vẽ của các thủ tục vẽ đường line, lineto, linerel, circle, rectangle (hàm vẽ hình chữ nhật, ta sẽ học trong phần vẽ miền ở dưới).

Hàm này sẽ cho phép ta xác định ba yếu tố khi vẽ đường thẳng, đó là : Kiểu đường, bề dày và mẫu tự tạo.

Dạng đường do tham số **kiểu_đường** xác định. Bảng dưới đây cho các giá trị khả dĩ của **kiểu_đường** :

Tên hằng	Giá trị số	Kiểu đường
SOLID_LINE	0	Nét liền
DOTTED_LINE	1	Nét chấm
CENTER_LINE	2	Nét chấm gạch
DASHED_LINE	3	Nét gạch
USERBIT_LINE	4	Mẫu tự tạo

Bề dày của đường vẽ do tham số **độ_dày** xác định, bảng dưới đây cho các giá trị khả dĩ của **độ_dày** :

Tên hằng	Giá trị số	Bề dày
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Mẫu tự tạo : Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số **mẫu**. Ví dụ ta xét đoạn chương trình :

```
int pattern = 0x1010;
setlinestyle(USERBIT_LINE,pattern,NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010, trong hệ 2 là :

```
0001 0000 0001 0000
```

Bit 1 sẽ cho điểm sáng, bit 0 sẽ làm tắt điểm ảnh.

Ví dụ :

Chương trình vẽ một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau :

(20,20),(620,20),(620,180),(20,180) và (320,100)

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
{
    int mh=0, mode;
    initgraph(&mh,&mode,"C:\\TC\\BGI");
    setbkcolor(BLUE);
    setcolor(YELLOW);
    setlinestyle(SOLID-LINE,0,THICK_WIDTH);
    moveto(320,100); /* con chạy ở vị trí ( 320,100 ) */
    line(20,20,620,20); /* con chạy vẫn ở vị trí ( 320,100 ) */
    linerel(-300,80);
    lineto(620,180);
    lineto(620,20);
    getch();
    closegraph();
}
```

10.2.4. Vẽ điểm, miền :

- **Vẽ điểm :**

Hàm :

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo màu xác định bởi **color**.

Hàm :

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu màu của điểm ảnh ở vị trí (x,y).

Chú ý :

Nếu điểm này chưa được tô màu bởi các hàm vẽ hoặc hàm putpixel (mà chỉ mới được tạo màu nền bởi setbkcolor) thì hàm cho giá trị 0.

- **Tô miền :**

Để tô màu cho một miền nào đó trên màn hình, ta dùng hàm sau :

```
void floodfill(int x, int y, int border);
```

ở đây :

(x,y) là toạ độ của một điểm nào đó gọi là điểm gieo.

Tham số border chứa mã của màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x,y,border và trạng thái màn hình.

+ Khi trên màn hình có một đường cong khép kín hoặc đường gấp khúc khép kín mà mã màu của nó bằng giá trị của border thì :

- Nếu điểm gieo (x,y) nằm trong miền này thì miền giới hạn phía trong đường sẽ được tô màu.

- Nếu điểm gieo (x,y) nằm ngoài miền này thì miền phía ngoài đường sẽ được tô màu.

+ Trong trường hợp khi trên màn hình không có đường cong nào như trên thì cả màn hình sẽ được tô màu.

Ví dụ :

Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Toạ độ (x,y) của điểm gieo được nạp từ bàn phím. Tùy thuộc giá trị cụ thể của x,y chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include "graphics.h"
```

```
#include "stdio.h"
```

```
main()
```

```
{
```

```
    int mh=mode=0, x, y;
```

```
    printf("\nVao toa do x,y:");
```

```
    scanf("%d%d",&x,&y);
```

```
    initgraph(&mh,&mode,"");
```

```
    if (graphresult != grOk) exit(1);
```

```
    setbkcolor(BLUE);
```

```
    setcolor(RED);
```

```

setfillstyle(11,YELLOW);
circle(320,100,50);
moveto(1,150);
floodfill(x,y,RED);
closegraph();
}

```

10.2.5. Hình chữ nhật :

- Hàm :

```
void rectangle(int x1, int y1, int x2, int y2);
```

sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hình chữ nhật là (x2,y2).

- Hàm :

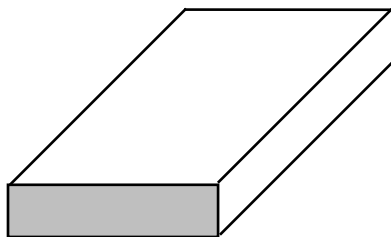
```
void bar(int x1, int y1, int x2, int y2);
```

sẽ vẽ và tô màu một hình chữ nhật. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hình chữ nhật là (x2,y2).

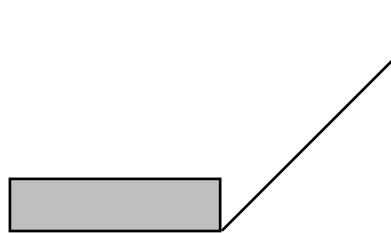
- Hàm :

```
void bar3d(int x1, int y1, int x2, int y2, int depth, int top);
```

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các toạ độ (x1,y1), (x2,y2). Hình chữ nhật này được tô màu thông qua hàm setfillstyle . Tham số **depth** xác định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số **top** có thể nhận các giá trị 1 hay 0 và khối 3 chiều tương ứng sẽ có nắp hoặc không.



top=1



top=0

Ví dụ :

Chương trình dưới đây tạo nên một hình chữ nhật, một khối hình chữ nhật và một hình hộp có nắp :

```
#include "graphics.h"
main()
{
    int mh=mode=0;
    initgraph(&mh,&mode,"");
    if (graphresult != grOk) exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL,YELLOW);
    rectangle(5,5,300,160);
    bar(3,175,300,340);
    bar3d(320,100,500,340,100,1);
    closegraph();
}
```

10.2.6. Cửa sổ (Viewport) :

- **Thiết lập viewport :**

Viewport là một vùng chữ nhật trên màn hình đồ hoạ. Để thiết lập viewport ta dùng hàm :

```
void setviewport(int x1, int y1, int x2, int y2, int clip);
```

trong đó (x1,y1) là toạ độ góc trên bên trái, (x2,y2) là toạ độ góc dưới bên phải. Bốn giá trị này vì thế phải thoả mãn :

$$0 \leq x1 \leq x2$$

$$0 \leq y1 \leq y2$$

Tham số clip có thể nhận một trong hai giá trị :

clip=1 không cho phép vẽ ra ngoài viewport.

clip=0 cho phép vẽ ra ngoài viewport.

Ví dụ :

```
setviewport(100,50,200,150,1);
```

Lập nên một vùng viewport hình chữ nhật có toạ độ góc trái cao là (100,50) và toạ độ góc phải thấp là (200,150) (là toạ độ trước khi đặt viewport).

Chú ý :

Sau khi lập viewport, ta có hệ tọa độ mới mà góc trên bên trái sẽ có tọa độ (0,0).

- **Nhận diện viewport hiện hành :**

Để nhận viewport hiện thời ta dùng hàm :

```
void getviewsetting(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau :

```
struct viewporttype
{
    int left,top,right,bottom;
    int clip;
};
```

- **Xóa viewport :**

Sử dụng hàm :

```
void clearviewport(void);
```

- **Xoá màn hình, đưa con chạy về tọa độ (0,0) của màn hình :**

Sử dụng hàm :

```
void cleardevice(void);
```

- **Tọa độ âm dương :**

Nhờ sử dụng viewport có thể viết các chương trình đồ họa theo tọa độ âm dương. Muốn vậy ta thiết lập viewport và cho clip bằng 0 để có thể vẽ ra ngoài giới hạn của viewport.

Sau đây là đoạn chương trình thực hiện công việc trên :

```
int xc,yc;
xc=getmaxx()/2;
yc=getmaxy()/2;
setviewport(xc,yc,getmaxx(),getmaxy(),0);
```

Như thế, màn hình sẽ được chia làm bốn phần với tọa độ âm dương như sau :

Phần tư trái trên : x âm, y âm.

x : từ $-\text{getmaxx}()/2$ đến 0.

y : từ $-\text{getmaxy}()/2$ đến 0.

Phần tư trái dưới : x âm, y dương.

x : từ $-\text{getmaxx}()/2$ đến 0.

y : từ 0 đến $\text{getmaxy}()/2$.

Phần tư phải trên : x dương, y âm.

x : từ 0 đến $\text{getmaxx}()/2$.

y : từ $-\text{getmaxy}()/2$ đến 0.

Phần tư phải dưới : x dương, y dương.

x : từ 0 đến $\text{getmaxx}()/2$.

y : từ 0 đến $\text{getmaxy}()/2$.

Ví dụ :

Chương trình vẽ đồ thị hàm sin x trong hệ trục tọa độ âm dương. Hoành độ x lấy các giá trị từ -4π đến 4π . Trong chương trình có sử dụng hai hàm mới là `settextjustify` và `outtextxy` ta sẽ đề cập ngay trong phần sau.

```
#include "graphics.h"
```

```
#include "conio.h"
```

```
#include "math.h"
```

```
#define TYLEX 20
```

```
#define TYLEY 60
```

```
main()
```

```
{
```

```
    int mh=mode=DETECT;
```

```
    int x,y,i;
```

```
    initgraph(mh,mode,"");
```

```
    if (graphresult!=grOK ) exit(1);
```

```
    setviewport(getmaxx()/2,getmaxy()/2,getmaxx(),getmaxy(),0);
```

```
    setbkcolor(BLUE);
```

```
    setcolor(YELLOW);
```

```
    line(-getmaxx()/2,0,getmaxx()/2,0);
```

```
    line(0,-getmaxy()/2,0,getmaxy()/2,0);
```

```
    settextjustify(1,1);
```

```

setcolor(WHITE);
outtextxy(0,0,"(0,0)");
for (i=-400;i<=400;++i)
    {
        x=floor(2*M_PI*i*TYLEX/200);
        y=floor(sin(2*M_PI*i/200)*TYLEY);
        putpixel(x,y,WHITE);
    }
getch();
closegraph();
}

```

10.3. Xử lý văn bản trên màn hình đồ họa :

- **Hiển thị văn bản trên màn hình đồ họa :**

Hàm :

```
void outtext(char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí con trỏ đồ họa hiện thời.

Hàm :

```
void outtextxy(int x, int y,char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí (x,y).

Ví dụ :

Hai cách viết dưới đây :

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
outtext(" Say HELLO");
```

cho cùng kết quả.

- **Sử dụng các Fonts chữ :**

Các Fonts chữ nằm trong các tập tin *.CHR trên đĩa. Các Fonts này cho các kích thước và kiểu chữ khác nhau, chúng sẽ được hiển thị lên màn hình bằng các hàm outtext và outtextxy. Để chọn và nạp Fonts ta dùng hàm :

```
void settextstyle(int font, int direction, int charsize);
```

Tham số font để chọn kiểu chữ và nhận một trong các hằng sau :

```
DEFAULT_FONT=0
```

```
TRIPLEX_FONT=1
```

```
SMALL_FONT=2
```

```
SANS_SERIF_FONT=3
```

```
GOTHIC_FONT=4
```

Tham số direction để chọn hướng chữ và nhận một trong các hằng sau :

```
HORIZ_DIR=0    văn bản hiển thị theo hướng nằm ngang từ trái qua phải.
```

```
VERT_DIR=1     văn bản hiển thị theo hướng thẳng đứng từ dưới lên trên.
```

Tham số charsize là hệ số phóng to của ký tự và có giá trị trong khoảng từ 1 đến 10.

Khi charsize=1, font hiển thị trong hình chữ nhật 8*8 pixel.

Khi charsize=2 font hiển thị trong hình chữ nhật 16*16 pixel.

.....

Khi charsize=10, font hiển thị trong hình chữ nhật 80*80 pixel.

Các giá trị do settextstyle lập ra sẽ giữ nguyên tới khi gọi một settextstyle mới.

Ví dụ :

Các dòng lệnh :

```
settextstyle(3,VERT_DIR,2);
```

```
outtextxy(30,30,"GODS TRUST YOU");
```

sẽ hiển thị tại vị trí (30,30) dòng chữ GODS TRUST YOU theo chiều từ dưới lên trên, font chữ chọn là SANS_SERIF_FONT và cỡ chữ là 2.

• Đạt vị trí hiển thị của các xâu ký tự cho bởi outtext và outtextxy :

Hàm settextjustify cho phép chỉ định ra nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy và của outtextxy theo quan hệ với tọa độ (x,y);

Hàm này có dạng sau :

```
void settextjustify(int horiz, int vert);
```


Tham số horiz có thể là một trong các hằng số sau :

LEFT_TEXT=0 (Văn bản xuất hiện bên phải con chạy).
CENTER_TEXT (Chỉnh tâm văn bản theo vị trí con chạy).
RIGHT_TEXT (Văn bản xuất hiện bên trái con chạy).

Tham số vert có thể là một trong các hằng số sau :

BOTTOM_TEXT=0 (Văn bản xuất hiện phía trên con chạy).
CENTER_TEXT=1 (Chỉnh tâm văn bản theo vị trí con chạy).
TOP_TEXT=2 (Văn bản xuất hiện phía dưới con chạy).

Ví dụ :

```
settextjustify(1,1);  
outtextxy(100,100,"ABC");
```

sẽ cho dòng chữ ABC trong đó điểm (100,100) sẽ nằm dưới chữ B.

- **Bề rộng và chiều cao văn bản :**

Chiều cao :

Hàm :

```
textheight(char *s);
```

cho chiều cao (tính bằng pixel) của chuỗi do con trỏ s trỏ tới.

Ví dụ 1 :

Với font bit map và hệ số phóng đại là 1 thì `textheight("A")` ch giá trị là 8.

Ví dụ 2 :

```
#include "stdio.h"
```

```
#include "graphics.h"
```

```
main()
```

```
{
```

```
    int mh=mode=DETECT, y,size;  
    initgraph(mh,mode,"C:\\TC\\BGI");  
    y=10;  
    settextjustify(0,0);  
    for (size=1;size<5;++size)
```

```

        {
            settextstyle(0,0,size);
            outtextxy(0,y,"SACRIFICE");
            y+=textheight("SACRIFICE")+10;
        }
    getch();
    closegraph();
}

```

Bề rộng :

Hàm :

```
textwidth(char *s);
```

cho bề rộng chuỗi (tính theo pixel) mà con trỏ s trỏ tới dựa trên chiều dài chuỗi, kích thước font chữ, hệ số phóng đại.

Mục lục

Giới thiệu

Chương 1

Các khái niệm cơ bản

- 1.1. Tập ký tự dùng trong ngôn ngữ C
- 1.2. Từ khoá
- 1.3. Tên
- 1.4. Kiểu dữ liệu
 - 1.4.1. Kiểu ký tự (char)
 - 1.4.2. Kiểu nguyên
 - 1.4.3. Kiểu dấu phẩy động
- 1.5. Định nghĩa kiểu bằng TYPEDEF
 - 1.5.1. Công dụng
 - 1.5.2. Cách viết
- 1.6. Hằng
 - 1.6.1. Tên hằng
 - 1.6.2. Các loại hằng
 - 1.6.2.1. Hằng int
 - 1.6.2.2. Hằng long
 - 1.6.2.3. Hằng int hệ 8
 - 1.6.2.4. Hằng int hệ 16
 - 1.6.2.5. Hằng ký tự
 - 1.6.2.5. Hằng xâu ký tự
- 1.7. Biến
- 1.8. Mảng

Chương 2

Các lệnh vào ra

- 2.1. Thâm nhập vào thư viện chuẩn
- 2.2. Các hàm vào ra chuẩn - getchar() và putchar()
 - 2.2.1. Hàm getchar()
 - 2.2.2. Hàm putchar()
 - 2.2.3. Hàm getch()

- 2.2.4. Hàm putchar()
- 2.3. Đưa kết quả lên màn hình - hàm printf
- 2.4. Vào số liệu từ bàn phím - hàm scanf
- 2.5. Đưa kết quả ra máy in

Chương 3

Biểu thức

- 3.1. Biểu thức
- 3.2. Lệnh gán và biểu thức
- 3.3. Các phép toán số học
- 3.4. Các phép toán quan hệ và logic
- 3.5. Phép toán tăng giảm
- 3.6. Thứ tự ưu tiên các phép toán
- 3.7. Chuyển đổi kiểu giá trị

Chương 4

Cấu trúc cơ bản của chương trình

- 4.1. Lời chú thích
- 4.2. Lệnh và khối lệnh
 - 4.2.1. Lệnh
 - 4.2.2. Khối lệnh
- 4.3. Cấu trúc cơ bản của chương trình
- 4.4. Một số quy tắc cần nhớ khi viết chương trình

Chương 5

Cấu trúc điều khiển

- 5.1. Cấu trúc có điều kiện
 - 5.1.1. Lệnh if-else
 - 5.1.2. Lệnh else-if
- 5.2. Lệnh nhảy không điều kiện - toán tử goto
- 5.3. Cấu trúc rẽ nhánh - toán tử switch
- 5.4. Cấu trúc lặp
 - 5.4.1. Cấu trúc lặp với toán tử while và for

5.4.1.1. Cấu trúc lặp với toán tử while

5.4.1.2. Cấu trúc lặp với toán tử for :

5.4.2. Chu trình do-while

5.5. Câu lệnh break

5.6. Câu lệnh continue

Chương 6

Hàm

6.1. Cơ sở

6.2. Hàm không cho các giá trị

6.3. Hàm đệ qui

6.3.3. Mở đầu

6.3.2. Các bài toán có thể dùng đệ qui

6.3.3. Cách xây dựng hàm đệ qui

6.3.4. Các ví dụ về dùng hàm đệ qui

6.4. Bộ tiền sử lý C

Chương 7

Con trỏ

7.1. Con trỏ và địa chỉ

7.2. Con trỏ và mảng một chiều

7.2.1. Phép toán lấy địa chỉ

7.2.2. Tên mảng là một hằng địa chỉ

7.2.3. Con trỏ trỏ tới các phần tử của mảng một chiều

7.2.4. Mảng, con trỏ và xâu ký tự

7.3. Con trỏ và mảng nhiều chiều

7.3.1. Phép lấy địa chỉ

7.3.2. Phép cộng địa chỉ trong mảng hai chiều

7.3.3. Con trỏ và mảng hai chiều

7.4. Kiểu con trỏ kiểu địa chỉ, các phép toán trên con trỏ

7.4.1. Kiểu con trỏ và kiểu địa chỉ

7.4.2. Các phép toán trên con trỏ

7.4.3. Con trỏ kiểu void

- 7.5. Mảng con trỏ
- 7.6. Con trỏ tới hàm
 - 7.6.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm
 - 7.6.2. Tác dụng của con trỏ hàm
 - 7.6.3. Đối của con trỏ hàm

Chương 8

Cấu trúc

- 8.1. Kiểu cấu trúc
- 8.2. Khai báo theo một kiểu cấu trúc đã định nghĩa
- 8.3. Truy nhập đến các thành phần cấu trúc
- 8.4. Mảng cấu trúc
- 8.5. Khởi đầu một cấu trúc
- 8.6. Phép gán cấu trúc
- 8.7. Con trỏ cấu trúc và địa chỉ cấu trúc
 - 8.7.1. Con trỏ và địa chỉ
 - 8.7.2. Truy nhập qua con trỏ
 - 8.7.3. Phép gán qua con trỏ
 - 8.7.4. Phép cộng địa chỉ
 - 8.7.5. Con trỏ và mảng
- 8.8. Cấu trúc tự trỏ và danh sách liên kết

Chương 9

Tập tin - file

- 9.1. Khái niệm về tệp tin
- 9.2. Khai báo sử dụng tệp - một số hàm thường dùng khi thao tác trên tệp
 - 9.2.1. Khai báo sử dụng tệp
 - 9.2.2. Mở tệp - hàm fopen
 - 9.2.3. Đóng tệp - hàm fclose
 - 9.2.4. Đóng tất cả các tệp đang mở- hàm fcloseall
 - 9.2.5. Làm sạch vùng đệm - hàm fflush
 - 9.2.6. Làm sạch vùng đệm của các tệp đang mở - hàm fflushall
 - 9.2.7. Kiểm tra lỗi file - hàm ferror

- 9.2.8. Kiểm tra cuối tệp - hàm feof
- 9.2.9. Truy nhập ngẫu nhiên - các hàm di chuyển con trỏ chỉ vị
 - 9.2.9.1. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind
 - 9.2.9.2. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek
 - 9.2.9.3. Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell
- 9.2.10. Ghi các mẫu tin lên tệp - hàm fwrite
- 9.2.11. Đọc các mẫu tin từ tệp - hàm fread
- 9.2.12. Nhập xuất ký tự
 - 9.2.12.1. Các hàm putc và fputc
 - 9.2.12.2. Các hàm getc và fgetc
- 9.2.13. Xoá tệp - hàm unlink

Chương 10

Đồ hoạ

- 10.1. Khởi động đồ hoạ
- 10.2. Các hàm đồ hoạ
 - 10.2.1. Mẫu và màu
 - 10.2.2. Vẽ và tô màu
 - 10.2.3. Vẽ đường gấp khúc và đa giác
 - 10.2.4. Vẽ điểm, miền
 - 10.2.5. Hình chữ nhật
 - 10.2.6. Cửa sổ (Viewport)
- 10.3. Sử lý văn bản trên màn hình đồ hoạ

Bài tập.

Phần thứ nhất : *Nhóm các bài tập về tính toán, hàm và chu trình .*

Bài tập 1 :

Viết chương trình hiển thị tháp Pascal :

tài liệu tham khảo

1. Các tài liệu tiếng Việt :

- 1.1. Ngô Trung Việt - Ngôn ngữ lập trình C và C++ - Bài giảng- Bài tập - Lời giải mẫu
NXB giao thông vận tải 1995
- 1.2. Viện tin học - Ngôn ngữ lập trình C
Hà nội 1990
- 1.3. Lê Văn Doanh - 101 thuật toán và chương trình bằng ngôn ngữ C

2. Các tài liệu tiếng Anh :

- 2.1. B. Kernighan and D. Ritchie - The C programming language
Prentice Hall 1989
- 2.2. Programmer's guide Borland C++ Version 4.0
Borland International, Inc 1993
- 2.3. Bile - Nabaiyoti - TURBO C++
The Waite Group's UNIX 1991

Bài tập Ngôn ngữ lập trình C

Phần 1 : Nhóm các bài tập về tính toán, hàm và chu trình .

Bài tập 1 :

Viết chương trình hiển thị tháp PASCAL :

```
1
121
12321
1234321
123454321
12345654321
1234567654321
123456787654321
12345678987654321
```

Viết chương trình hiển thị tháp đảo ngược.

Bài tập 2 :

Viết chương trình nhập ba số thực. Kiểm tra xem ba số đó có thể là chiều dài của ba cạnh của một tam giác được không? Nếu được thì tính chu vi và diện tích tam giác đó.

Bài tập 3 :

Viết chương trình tính hàm số :

$$f(x) = \frac{K_0}{K_1 + \frac{x}{K_2 + \frac{x}{K_3 + \frac{x}{K_4 + \dots + \frac{x}{K_{n-1} + \frac{x}{K_n}}}}}}$$

Bài tập 4 :

Viết chương trình tính tích hai ma trận $C_{m \times n} = A_{m \times n} * B_{n \times k}$.

Bài tập 5 :

Viết chương trình nhập vào một dãy số sau đó tách dãy này thành hai dãy chỉ chứa các số dương và chỉ chứa các số âm. Tính tổng số phần tử của mỗi dãy sau đó sắp xếp để hai dãy có giá trị giảm dần.

Bài tập 6 :

Viết chương trình nhập vào một ma trận $A_{n \times m}$. Tìm giá trị cực đại và cực tiểu của các phần tử của mảng .

Bài tập 7 :

Trăm trâu, trăm cò
 Trâu đứng ăn năm
 Trâu nằm ăn ba
 Lụ khụ trâu già
 Ba con một bó.

Tính số trâu mỗi loại .

Bài tập 8 :

Vừa gà vừa chó

Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn .

Tính số gà, số chó .

Bài tập 9 :