ỦY BAN NHÂN DÂN QUẬN 5

TRƯỜNG TRUNG CẤP NGHỀ KỸ THUẬT CÔNG NGHỆ HÙNG VƯƠNG

GIÁO TRÌNH

Lập trình căn bản

Nghề: Quản trị mạng máy tính

TRÌNH ĐỘ TRUNG CẤP

TPHCM - 2019

LỜI GIỚI THIỆU

Hoạt động biên soạn giáo trình là một hoạt động nghiên cứu khoa học . Mỗi giáo viên, giảng viên trên cơ sở các phương pháp và nguyên tắc chung sẽ có sự vận dụng sáng tạo vào điều kiện cụ thể của mình để sáng tạo ra các nội dung giảng dạy hấp dẫn, thu hút người học.

Nhằm giúp đội ngũ giáo viên có tài liệu cơ sở để tiến hành các buổi lên lớp được hiệu quả cũng như việc cung cấp tài liệu giúp cho sinh viên nói chung, đặc biệt là sinh viên chuyên ngành Quản trị mạng. Để đáp ứng nhu cầu thực tiễn này khoa Công nghệ thông tin đã tổ chức biên soạn cuốn giáo trình "Lập trình căn bản" do nhóm giáo viên thuộc chuyên ngành Quản trị mạng đang công tác tại trường TCN KTCN Hùng Vương biên soạn.

Đây là công trình được viết bởi đội ngũ giáo viên đã và đang công tác tại nhà trường cùng với sự góp ý và phản biện của các doanh nghiệp trong lĩnh vực liên quan, tuy vậy, cuốn sách chắc chắn vẫn không tránh khỏi những khiếm khuyết. Chúng tôi mong nhận được ý kiến đóng góp của bạn đọc để cuốn sách được hoàn thiện hơn trong lần tái bản.

Xin trân trọng giới thiệu cùng bạn đọc!

Quận 5, ngày tháng năm 2014

Tham gia biên soạn

1. Chủ biên Võ Đức Thiện

2. Nguyễn Minh Vũ

3. Huỳnh Phan Diệu Hiền

MỤC LỤC

GIỚI THIỆU VỀ MÔN HỌC LẬP TRÌNH CĂN BẢN

Vị trí, ý nghĩa, vai trò môn học

- Vị trí: Môn học được bố trí sau khi sinh viên học xong các môn học chung, các môn học tin đại cương, tin văn phòng.
- Tính chất: Là môn học cơ sở nghề bắt buộc.

Mục tiêu của môn học

- Trình bày được khái niệm về lập máy tính;
- Mô tả được ngôn ngữ lập trình: cú pháp, công dụng của các câu lệnh;
- Phân tích được chương trình: xác định nhiệm vụ chương trình;
- Thực hiện được các thao tác trong môi trường phát triển phần mềm: biên tập chương trình, sử dụng các công cụ, điều khiển, thực đơn lệnh trợ giúp, gỡ rối, bẫy lỗi,v.v.;
- Viết chương trình và thực hiện chương trình trong máy tính.
- Bố trí làm việc khoa học đảm bảo an toàn cho người và phương tiện học tập.

Nội dung môn học:

- Các khái niệm cơ bản
- Các lệnh vào ra
- Biểu thức
- Cấu trúc cơ bản của chương trình
- Cấu trúc điều khiển
- Hàm

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

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

Khi đó thực chất:

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

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

Phân loại ký tự:

Có thể chia 256 ký tự làm ba nhóm:

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

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

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

1.4.2. Kiểu nguyên:

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

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

Chú ý:

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

1.4.3. Kiểu dấu phảy động:

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

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

Giải thích:

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

1.5. Định nghĩa kiểu bằng TYPEDEF:

1.5.1. Công dụng:

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

1.5.2. Cách viết:

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

Ví dụ câu lệnh:

```
typedef int nguyen;
```

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

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

Tương tự cho các câu lệnh:

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

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

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

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

Sau này ta sẽ dùng các kiểu trên khai báo:

```
mt50 a,b;
m_20_30 x,y;
```

1.6. Hằng:

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

1.6.1. Tên hằng:

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

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

#define tên hằng giá trị

Ví dụ:

```
#define MAX 1000
```

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

Một ví dụ khác:

#define pi 3.141593

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

1.6.2. Các loại hằng:

1.6.2.1. Hằng int:

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

Ví dụ:

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

Chú ý:

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

1.6.2.2. Hằng long:

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

Hằng long được viết theo cách:

1234L hoặc 1234l

(thêm L hoặc l vào đuôi)

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

Ví dụ:

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

1.6.2.3. Hằng int hệ 8:

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

Ví dụ:

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

1.6.2.4. Hằng int hệ 16:

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

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

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

Ví dụ:

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

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

10*16+5=165.

1.6.2.5. Hằng ký tự:

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

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

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

Ví dụ:

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

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

' \c1c2c3'

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

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

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

Chú ý:

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

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

1.6.2.5. Hằng xâu ký tự:

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

Ví dụ:

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

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

Chú ý:

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

1.7. Biến:

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

Kiểu dữ liệu của biến tên biến;

Ví dụ:

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

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

Vị trí của khai báo biến:

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

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

```
main()
    {
        int a,b,c;
        a=2;
        int d; /* Vị trí của khai báo sai */
        .....
    }
```

Khởi đầu cho biến:

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

Ví dụ:

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

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

Lấy địa chỉ của biến:

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

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

& tên biến

1.8 Mảng:

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

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

Loại mảng: int, float, double...

Tên mảng.

Số chiều và kích thước mỗi chiều.

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

Các khai báo:

```
int a[10],b[4][2];
float x[5],y[3][3];
```

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

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

Chú ý:

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

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

Chỉ số mảng:

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

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

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

Chú ý:

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

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

Ví dụ:

a[2.5]	là	a[2]
b[1.9]	là	a[1]

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

Lấy địa chỉ một phần tử của mảng:

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

Địa chỉ đầu của một mảng:

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

Khởi đầu cho biến mảng:

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

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

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

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

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

Ví dụ:

```
....
float y[6]={3.2,0,5.1,23,0,42};
int z[3][2]={
                {25,31},
                {12,13},
                {45,15}
            {
....
main()
{
        ....
}
```

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

Ví dụ:

```
....
float a[]={0,5.1,23,0,42};
int m[][3]={
                {25,31,4},
                {12,13,89},
                {45,15,22}
            };
```

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

Ví dụ:

```
....
float m[6]={0,5.1,23,0};
int z[6][3]={
                {25,31,3},
                {12,13,22},
                {45,15,11}
            };
....
```

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

Ví dụ:

```
....
float z[][3]={
                {31.5},
```

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

Khởi đầu của một mảng char có thể là

Một danh sách các hằng ký tự.

Một hằng xâu ký tự.

Ví dụ:

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

Chương 2: CÁC LỆNH VÀO RA

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

2.1. Thâm nhập vào thư viện chuẩn:

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

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

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

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

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

2.2.1. Hàm getchar ():

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

Cách dùng:

Dùng câu lệnh sau:

biến = getchar();

Công dụng:

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

Ví dụ:

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

2.2.2. Hàm putchar ():

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

Cách dùng:

Dùng câu lệnh sau:

```
putchar(ch);
```

Công dụng:

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

Ví dụ:

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

2.2.3. Hàm getch():

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

Cách dùng:

Dùng câu lệnh sau:

```
getch();
```

Công dụng:

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

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

Nếu dùng:

```
biến=getch();
```

Thì biến sẽ chứa ký tự đọc vào.

Ví dụ:

c = getch();

2..2.4. Hàm putch():

Cách dùng:

Dùng câu lệnh sau:

putch(ch);

Công dụng:

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

Hàm cũng trả về ký tự được hiển thị.

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

Cách dùng:

prinf(điều khiển, đối số 1, đối số 2,...);

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

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

\n sang dòng mới

\f sang trang mới

\b lùi lại một bước

\t dấu tab

Dạng tổng quát của đặc tả:

%[-][fw][.pp]ký tự chuyển dạng

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

Dấu trừ:

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

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

fw:

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

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

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

Ví duï:

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

pp:

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

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

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

Khi đối là xâu ký tự:

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

Ví dụ:

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

Các ký tự chuyển dạng và ý nghĩa của nó:

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

Ký tự chuyển dạng	ý nghĩa
d	Đối được chuyển sang số nguyên hệ thập phân

o	Đối được chuyển sang hệ tám không dấu (không có số 0 đứng trước)
x	Đối được chuyển sang hệ mười sáu không dấu (không có 0x đứng trước)
u	Đối được chuyển sang hệ thập phân không dấu
c	Đối được coi là một ký tự riêng biệt
s	Đối là xâu ký tự, các ký tự trong xâu được in cho tới khi gặp ký tự không hoặc cho tới khi đủ số lượng ký tự được xác định bởi các đặc tả về độ chính xác pp.
e	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m.n..nE[+ hoặc -] với độ dài của xâu chứa n là pp.
f	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m..m.n..n với độ dài của xâu chứa n là pp. Độ chính xác mặc định là 6. Lưu ý rằng độ chính xác không xác định ra số các chữ số có nghĩa phải in theo khuôn dạng f.
g	Dùng %e hoặc %f, tuỳ theo loại nào ngắn hơn, không in các số 0 vô nghĩa.

Chú ý:

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

Để hiển thị các ký tự đặc biệt:

Cách viết	Hiển thị
\'	'
\"	"
\\	\

Các ví dụ:

1 printf("\" Nang suat tang: %d % \" "Nang suat tang; 30 %"

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

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

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

Cách dùng:

scanf(điều khiển,đối 1, đối 2,...);

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

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

%[*][d...d]ký tự chuyển dạng

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

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

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

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

Ví dụ:

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

Với dòng vào: 54.32e-1 25 12452348a

Kết quả là lệnh scanf sẽ gán

5.432 cho x

25.0 cho y

124 cho a

xâu "523" và dấu kết thúc \0 cho ch

xâu "48a" và dấu kết thúc \0 cho ct

Ký tự chuyển dạng:

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

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

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

Các ký tự chuyển dạng và ý nghĩa của nó:

c	Vào một ký tự, đối tương ứng là con trỏ ký tự. Có xét ký tự khoảng trắng
d	Vào một giá trị kiểu int, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên
ld	Vào một giá trị kiểu long, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên
o	Vào một giá trị kiểu int hệ 8, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 8

lo	Vào một giá trị kiểu long hệ 8, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 8
x	Vào một giá trị kiểu int hệ 16, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 16
lx	Vào một giá trị kiểu long hệ 16, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 16
f hay e	Vào một giá trị kiểu float, đối tương ứng là con trỏ float, trường vào phải là số dấu phảy động
lf hay le	Vào một giá trị kiểu double, đối tương ứng là con trỏ double, trường vào phải là số dấu phảy động
s	Vào một giá trị kiểu double, đối tương ứng là con trỏ kiểu char, trường vào phải là dãy ký tự bất kỳ không chứa các dấu cách và các dấu xuống dòng

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

Ví dụ:

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

Với dòng vào:

```
35 13145 xyz 584235
```

Sẽ gán:

35 cho a

xâu "13145" cho ch

xâu "xyz' cho ck

584 cho b

Chú ý:

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

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

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

Cách 1:

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

Cách 2:

Đưa ba số vào ba dòng khác nhau.

Cách 3:

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

Cách 4:

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

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

2.5. Đưa kết quả ra máy in:

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

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

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

Điều khiển có dạng đặc tả như lệnh printf.

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

Ví dụ:

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

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

Chương 3: BIỂU THỨC

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

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

3.1. Biểu thức:

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

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

Biểu thức gán.

Biểu thức điều kiện.

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

Biểu thức thường được dùng trong:

Vế phải của câu lệnh gán.

Làm tham số thực sự của hàm.

Làm chỉ số.

Trong các toán tử của các cấu trúc điều khiển.

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

3.2. Lệnh gán và biểu thức:

Biểu thức gán là biểu thức có dạng:

v=e

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

v=e;

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

a=b=5;

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

Hoàn toàn tương tự như:

a=b=c=d=6; gán 6 cho cả a, b, c và d

Ví dụ:

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

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

3.3. Các phép toán số học:

Các phép toán hai ngôi số học là

Phép toán	ý nghiã	Ví dụ
+	Phép cộng	a+b
-	Phép trừ	a-b
*	Phép nhân	a*b
/	Phép chia	a/b (Chia số nguyên sẽ chặt phần thập phân)
%	Phép lấy phần dư	a%b (Cho phần dư của phép chia a cho b)

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

Ví dụ:

11/3=3

11%3=2

-(2+6)=-8

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

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

3.4. Các phép toán quan hệ và logic:

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

Các phép toán quan hệ là:

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

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

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

i<n-1

được hiểu là i<(n-1).

Các phép toán logic:

Trong C sử dụng ba phép toán logic:

Phép phủ định một ngôi !

A	!a
khác 0	0
bằng 0	1

Phép và (AND) &&

Phép hoặc (OR) ||

a	B	a&&b	a||b
khác 0	khác 0	1	1
khác 0	bằng 0	0	1
bằng 0	khác 0	0	1
bằng 0	bằng 0	0	0
a	B	a&&b	a||b

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

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

có thể viết lại thành:

a<b&&c>d

Chú ý:

Cả a và b có thể là nguyên hoặc thực.

3.5. Phép toán tăng giảm:

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

Ví dụ:

n=5

++n Cho ta n=6

--n Cho ta n=4

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

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

Ví dụ:

n=5

x=++n Cho ta x=6 và n=6

x=n++ Cho ta x=5 và n=6

3.6. Thứ tự ưu tiên các phép toán:

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

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

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

Chú thích:

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

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

Ví dụ:

--px=(--px) (Phải qua trái)

8/4*6=(8/4)*6 (Trái qua phải)

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

Các phép toán lạ:

Dòng 1

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

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

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

Dòng 2

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

& Phép toán lấy địa chỉ, ví dụ: &x

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

Dòng 15

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

3.7. Chuyển đổi kiểu giá trị:

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

Khi gán biểu thức gồm các toán hạng khác kiểu.

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

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

(type) biểu thức

Ví dụ:

(float) (a+b)

Chuyển đổi kiểu trong biểu thức:

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

Giữa int và long thì int chuyển thành long.

Giữa int và float thì int chuyển thành float.

Giữa float và double thì float chuyển thành double.

Ví dụ:

1.5*(11/3)=4.5

1.5*11/3=5.5

(11/3)*1.5=4.5

Chuyển đổi kiểu thông qua phép gán:

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

Ví dụ:

int n;

n=15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức:

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

Ví dụ:

Phép toán: (int)a

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

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

sqrt((double)n)

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

Chú ý:

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

((float)a)/b

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

(int)(r+0.5)

Chú ý thứ tự ưu tiên:

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

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

Chương 4 CẤU TRÚC CƠ BẢN CỦA CHƯƠNG TRÌNH

4.1. Lời chú thích:

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

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

Ví dụ:

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

4.2. Lệnh và khối lệnh:

4.2.1. Lệnh:

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

Ví dụ:

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

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

4.2.2. Khối lệnh:

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

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

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

Khai báo ở đầu khối lệnh:

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

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

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

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

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

Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh

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

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

Ví dụ:

Xét đoạn chương trình sau:

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

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

a trong =4 b=6

a ngoài =5 b=6

Do tính chất biến a trong và ngoài khối lệnh.

4.3. Cấu trúc cơ bản của chương trình:

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

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

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

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

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

Các chương trình C được tổ chức theo mẫu:

.....

hàm 1

.....

hàm 2

.....

hàm n

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

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

Sử dụng đối của hàm.

Sử dụng biến ngoài, mảng ngoài...

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

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

Ví dụ:

Chương trình tính x lũy thừa y 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 luỹ thừa y luỹ thừa x */
        fprintf(stdprn,"\n x= %8.2lf \n y=%8.2lf \n z=%8.2lf",x,y,z);
    }
```

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

Qui tắc đầu tiên cần nhớ là:

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

Qui tắc thứ hai là:

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 ví trí trước hoặc sau câu lệnh hiện thời.

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

Nhảy không có điều kiện.

Rẽ nhánh.

Tổ chức chu trình.

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

5.1. Cấu trúc có điều kiện:

5.1.1. Lệnh if-else:

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

```
if ( biểu thức)
khối lệnh 1;
/* Dạng một */
```

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

Hoạt động của biểu thức dạng 1:

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

Hoạt động của biểu thức dạng 2:

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

Ví dụ:

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

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

```
        if (a>b) max=a;
        else max=b;
        printf(" \n Max cua hai so a=%8.2f va b=%8.2f la
Max=%8.2f",a,b,max);
    }
```

Sự lồng nhau của các toán tử if:

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

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

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

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với:

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

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

```
if ( n>0)      /* if thứ nhất*/
        {
```

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

5.1.2. Lệnh else-if:

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

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

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

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

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

Ví dụ:

Chương trình giải phương trình bậc hai.

```
#include "stdio.h"
main()
 {
    float a,b,c,d,x1,x2;
```

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

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

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

Ví dụ:

ts: s=s++;

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

Toán tử goto có dạng:

goto nhãn;

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

Khi dùng toán tử goto cần chú ý:

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

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

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

Ví dụ:

Tính tổng s=1+2+3+....+10

```
#include "stdio.h"
main()
    {
        int s,i;
        i=s=0;
        tong:
        ++i;
        s=s+i;
        if (i<10) goto tong;
        printf("\n tong s=%d",s);
    }
```

5.3. Cấu trúc rẽ nhánh - toán tử switch:

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

Cấu trúc tổng quát của nó là:

```
switch ( biểu thức nguyên)
  {
        case n1
                khối lệnh 1
        case n2
                khối lệnh 2
```

```
.....
case nk
khối lệnh k
[ default
        khối lệnh k+1        ]
}
```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch. default là một thành phần không bắt buộc phải có trong thân của switch.

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

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

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

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

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

Chú ý:

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

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

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

kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ:

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

```
#include "stdio.h"
main()
    {
        int diem;
        tt: printf("\nVao du lieu:");
        printf("\n Diem =");
        scanf("%d",&diem);
        switch (diem)
        {
            case 0:
            case 1:
            case 2:
            case 3:printf("Kem\n");break;
            case 4:printf("Yeu\n");break;
            case 5:
            case 6:printf("Trung binh\n");break;
            case 7:
            case 8:printf("Kha\n");break;
            case 9:
            case 10:printf("Gioi\n");break;
            default:printf(Vao sai\n);
        }
        printf("Tiep tuc 1, dung 0:")
        scanf("%d",&diem);
        if (diem==1) goto tt;
```

```
    getch();
    return;
}
```

5.4. Cấu trúc lặp:

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

5.4.1.1. Cấu trúc lặp với toán tử while:

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

```
while ( biểu thức)
        Lệnh hoặc khối lệnh;
```

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

Hoạt động của chu trình như sau:

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

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

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

Chú ý:

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

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

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

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

Ví dụ:

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

Cách 1:

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

Cách 2:

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

Cách 3:

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

5.4.1.2. Cấu trúc lặp với toán tử for:

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

for (biểu thức 1; biểu thức 2; biểu thức 3)

Lệnh hoặc khối lệnh;

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

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

Hoạt động của toán tử for:

Toán tử for hoạt động theo các bước sau:

Xác định biểu thức 1

Xác định biểu thức 2

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

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

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

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

Chú ý:

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

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

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

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

Ví dụ 1:

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

Cách 1:

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

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

Cách 2:

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

Cách 3:

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

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

Ví dụ 2:

Tính tích hai ma trận mxn và nxp.

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

```
        }
for (i=0;i<=3;++i)
for (j=0;j<=4;++j)
z[i][j]
}
```

5.4.2. Chu trình do-while

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

Chu trình do while có dạng sau:

```
do
    Lệnh hoặc khối lệnh;
while ( biểu thức);
```

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

Hoạt động của chu trình như sau:

Máy thực hiện các lệnh trong thân chu trình.

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

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

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

Chú ý:

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

Ví dụ:

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

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

```
float x[5],c;
main()
    {
        int i=0;
        printf("\n nhap gia tri cho ma tran x ");
        for (i=0;i<=4;++i)
        {
            printf("\n x[%d]=",i);
            scanf("%f",&c);
            y[i]=c;
        }
        do
            ++i;
        while (x[i]>=0 && i<=4);
        if (i<=4)
            printf("\n Phan tu am dau tien = x[%d]=%8.2f",i,x[i]);
        else
            printf("\n Mang khong có phan tu am ");
    }
```

5.5. Câu lệnh break:

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

Ví dụ:

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

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

```
# include "math.h"
unsigned int n;
main()
    {
        int i,nt=1;
        printf("\n cho n=");
        scanf("%d",&n);
        for (i=2;i<=sqrt(n);++i)
        if ((n % i)==0)
        {
            nt=0;
            break;
        }
        if (nt)
            printf("\n %d la so nguyen to",n);
        else
            printf("\n %d khong la so nguyen to",n);
    }
```

5.6. Câu lệnh continue:

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

Chú ý:

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

Ví dụ:

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

Tính tổng các phần tử dương của a.

Xác định số phần tử dương của a.

Tìm cực đại trong các phần tử dương của a.

```
#include "stdio.h"
float a[3[4];
main()
	{
		int i,j,soptd=0;
		float tongduong=0,cucdai=0,phu;
		for (i=0;i<3;++i)
		for (j=0;i<4;++j)
		{
			printf("\n a[%d][%d]=",i,j);
			scanf("%f",&phu);
			a[i][j]=phu;
			if (a[i][j]<=0) continue;
			tongduong+=a[i][j];
			if (cucdai<a[i][j]) cucdai=a[i][j];
			++soptd;
		}
		printf("\n So phan tu duong la: %d",soptd);
		printf("\n Tong cac phan tu duong la: %8.2f",tongduong);
		printf("\n Cuc dai phan tu duong la: %8.2f",cucdai);
	}
```

Chương 6: HÀM

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

6.1. Cơ sở:

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

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

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

Dòng tiêu đề:

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

Ví dụ:

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

khai báo các đối có dạng:

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

Thân hàm:

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

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

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

Dạng tổng quát của nó là:

return [biểu thức];

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

Ví dụ:

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

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

Chương trình được viết như sau:

```
#include "stdio.h"
float max3s(float a,float b,float c); /* Nguyên mẫu hàm*/
main()
    {
        float x,y,z;
        printf("\n Vao ba so x,y,z:");
        scanf("%f%f%f",&x&y&z);
        printf("\n Max cua ba so x=%8.2f y=%8.2f z=%8.2f la: %8.2f",
        x,y,z,max3s(x,y,z));
```

```
}       /* Kết thúc hàm main*/

float max3s(float a,float b,float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
                if (max<c) max=c;
                return(max);
        } /* Kết thúc hàm max3s*/
```

Quy tắc hoạt động của hàm:

Một cách tổng quát lời gọi hàm có dạng sau:

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

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

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

Cấp phát bộ nhớ cho các biến cục bộ.

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

Thực hiện các câu lệnh trong thân hàm.

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

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

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

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

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

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

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

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

Chú ý:

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

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

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

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

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

6.2. Hàm không cho các giá trị:

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

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

```
void htmax3s(float a, float b, float c)
```

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

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

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

6.3. Hàm đệ qui:

6.3.3. Mở đầu:

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

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

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

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

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

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

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

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

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

```
#include "stdio.h"
main()
    {
        printf("\n 3!=%d",gtdq(3));
    }
```

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

n*gtdq(n-1) (*)

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

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

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

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

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

n*gtdq(1) đây là kết quả của (**)

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

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

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

3!=6

Chú ý:

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

6.3.2. Các bài toán có thể dùng đệ qui:

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

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

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

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

6.3.3. Cách xây dựng hàm đệ qui:

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

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

6.3.4. Các ví dụ về dùng hàm đệ qui:

Ví dụ 1:

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

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

Trong trường hợp chung:

uscln(a,b)=uscln(a-b,b) nếu a>b

uscln(a,b)=uscln(a,b-a) nếu a<b

Ta có thể viết chương trình như sau:

```
#include "stdio.h"
int uscln(int a,int b); /* Nguyên mẫu hàm*/
```

```
main()
        {       int m,n;
                printf("\n Nhap cac gia tri cua a va b:");
                scanf("%d%d",&m,&n);
                printf("\n USCLN cua a=%d va b=%d la:%d",m,m,uscln(m,n))
        }
int uscln(int a,int b)
        {
                if (a==b)
                        return a;
                else
                 if (a>b)
                        return uscln(a-b,b);

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

Ví dụ 2:

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

```
# include "stdio.h"
# include "conio.h"
void prind(int n);
main()
{
 int a;
 clrscr();
 printf("n=");
 scanf("%d",&a);
 prind(a);
```

```
 getch();
}
void prind(int n)
 {
 int i;
 if (n<0)
 { putchar('-');
  n=-n;
 }
 if ((i=n/10)!=0)
 prind(i);
 putchar(n%10+'0');
 }
```

6.4. Bộ tiền sử lý C:

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

Bao hàm file:

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

#include "tên file"

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

Phép thế MACRO:

Định nghĩa có dạng:

#define biểu thức 1 [biểu thức 2]

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

Ví dụ:

#define YES 1

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

Phạm vi cho tên được định nghĩa bởi #define là từ điểm định nghĩa đến cuối file gốc. Có thể định nghĩa lại tên và một định nghĩa có thể sử dụng các định nghĩa khác trước đó. Phép thế không thực hiện cho các xâu dấu nháy, ví dụ như YES là tên được định nghĩa thì không có việc thay thế nào được thực hiện trong đoạn lệnh có "YES".

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

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

sau đó viết đoạn chương trình:

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

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

Ví dụ:

Định nghĩa macro gọi max như sau:

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

Việc sử dụng:

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

tương đương với:

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

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

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

Chú ý:

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

Ví dụ:

Xét chương trình sau:

```
main()
    {
        int x,y,z;
        x=5;
        y=10*5;
        z=x+y;
        z=x+y+6;
        z=5*x+y;
        z=5*(x+y);
```

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

Chương trình sử dụng MACRO sẽ như sau:

```
#define BEGIN {
#define END }
#define INTEGER int
#define NB 10
#define LIMIT NB*5
#define SUMXY x+y
#define SUM1 (x+y)
#define SUM2 ((x)+(y))
main()
    BEGIN
        INTEGER x,y,z;
        x=5;
        y=LIMIT;
        z=SUMXY;
        z=5*SUMXY;
        z=5*SUM1;
        z=5*SUM2;
        printf("\n Z=%d",z);
        getch();
        return;
    END
```

TÀI LIỆU THAM KHẢO

- Giáo trình môn lập trình C của Tiến Sĩ Lê Mạnh Thạnh, nhà xuất bản gio dục Năm 2000.
- Giáo trình kỹ thuật lập trình C - Nguyễn Linh Giang, Nguyễn Xuân Thực, Lê Văn Thái – Nhà xuất bản giáo dục – Năm 2005