Minh: Bài 5,6
Sơn: Bài 7,8
Hùng: Bài 9, 10
Hiền: Bài 11,12

Bài 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.

Chú ý: Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên. Ví dụ khi lập chương trình giải phương trình bậc hai $ax^2+bx+c=0$, ta cần tính biểu 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 C:

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

volatile while

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

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

- Từ khoá phải được viết bằng chữ thường, ví dụ: 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, chữ số và gạch nối. Ký tự đầu tiên của tên phải là chữ cái hoặc gạch nối. Tên không được trùng với từ khoá. Độ dài cực đại của tên theo mặc định là 32 và ta 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 cách
Y-3	Sử dụng dấu -

Chú ý: Trong 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:

1.4.1. Kiểu ký tự - char:

Một giá trị kiểu char chiếm 1 byte (8 bit) trong bộ nhớ 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 char và unsigned char.

Kiểu	Phạm vi biểu diễn	Số ký tự	Kích thước
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:

```
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 nhóm 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.

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	Kích thước
float	3.4E-38 đến 3.4E+38	4 byte
double	1.7E-308 đến 1.7E+308	8 byte
long double	3.4E-4932 đến 1.1E4932	10 byte

1.5. Định nghĩa kiểu dữ liệu mới bằng typedef:

1.5.1. Ý nghĩa:

Sử dụng từ khoá typedef để khai báo một tên kiểu dữ liệu mới, sau đó có thể dùng tên này để khai báo kiểu dữ liệu cho các biến, mảng, cấu trúc, vv...

1.5.2. Cú pháp:

Viết từ khoá typedef, sau đó là kiểu dữ liệu, rồi đến tên của kiểu dữ liệu mới.

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

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

Tương tự ta có:

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.

Sử dụng các kiểu dữ liệu ở trên như sau:

```
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 hoạt động của chương trình.

1.6.1. Tên hằng:

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

Để khai báo một hằng, ta sử dụng cú pháp như sau:

#define tên hằng giá trị

Ví dụ 1: #define MAX 1000

Tất cả các tên MAX xuất hiện trong chương trình sau này đều được thay bằng 1000.

Ví dụ 2: #define pi 3.141593

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

1.6.2. Các loại hằng:

1.6.2.1. Hằng kiểu int:

Hằng kiểu 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 kiểu long:

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

Hằng kiểu 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 s2 8865056	Định nghiã hằng long s2 có giá trị là 8865056

1.6.2.3. Hằng kiểu int trong hệ cơ số 8:

Hằng kiểu int trong hệ cơ số 8 được viết theo cách 0c1c2c3... Ở đây ci là một số nguyên dương nhận giá trị từ 1 đến 7. Hằng kiểu 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 kiểu int trong hệ cơ số 16:

Hệ 16 sử dụng 16 ký tự: 0,1..,9,A,B,C,D,E,F để biểu diễn các giá trị

Kí hiệu	**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 các kí hiệu trong hệ 16.

Ví dụ:

#define H16A 0xa5

#define H16B 0xA5

#define H16C 0Xa5

#define H16D 0XA5

Cho ta các hằng số 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

#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 trong hệ cơ số 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 (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 mảng 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 cú pháp 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 có thể nhận 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 biến ngoài : Là các biến được khai báo bên ngoài các hàm. Phạm vi sử dụng của các biến ngoài được xác định từ ví trí khai báo đến cuối chương trình.

Các biến được khai báo bên trong các hàm, bên trong các khối lệnh được gọi là các biến cục bộ hay các biến trong. Các biến cục bộ cần phải được đặt ngay sau dấu { của mỗi khối lệnh và cần đứng trước mọi câu lệnh khác. Biến cục bộ chỉ có phạm vi sử dụng bên trong hàm, bên trong khối lệnh mà nó được khai báo.

Sau đây là một ví dụ về khai báo biến sai:

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

Khởi tạo giá trị ban 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 tạo giá trị ban 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 gán giá trị khởi đầu cho biến 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 tiên 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:

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

Kiểu dữ liệu của mảng: int, float, double...

Tên mảng.

Số chiều và kích thước mỗi chiều của mảng.

Khái niệm về kiểu của mảng và tên mảng cũng giống như khái niệm về kiểu của biến và tên biến. Số chiều và kích thước mỗi chiều của mảng thể hiện 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.

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

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

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

Khi biểu thức dùng làm chỉ số của mảng là số thực thì chỉ lấy phần nguyên của kết quả làm chỉ số.

Ví dụ: a[2.5] là a[2]
b[1.9] là a[1]

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

Khởi tạo giá trị ban đầu cho biến mảng: Để khởi tạo giá trị ban đầu cho biến mảng ta có thể sử dụng biểu thức hằng hoặc sử dụng các câu lệnh gán. Các ví dụ sau thể hiện việc khởi tạo giá trị ban đầu cho mảng bằng biểu thức hằng.

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 tạo giá trị ban đầu cho mảng ta có thể không 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 các 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 phải không nhỏ hơn kích thước của danh sách các giá trị khởi tạo.

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, số giá trị khởi tạo ban 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 tạo giá trị ban đầu của một mảng char có thể thực hiện theo hai cách sau:

Khởi tạo bằng một danh sách các hằng ký tự.

Khởi tạo bằng một hằng xâu ký tự.

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

Bài 2: CẤU TRÚC CƠ BẢN CỦA CHƯƠNG TRÌNH C

2.1. 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 luôn cần viết thêm những lời giải thích để chương trình rõ ràng và dễ hiểu hơn.

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 */
  }
  strcpy(str, "Hello");     /* copy "Hello" vào xâu */
  printf("String is %s\n", str);     /* Hiển thị xâu */
  free(str);     /* Giải phóng bộ nhớ */
  return 0;
}
```

2.2. Lệnh và khối lệnh:

2.2.1. Lệnh:

Một biểu thức kiểu như x=0 hoặc ++i hoặc scanf(...),… trở thành câu lệnh của C khi có đi kèm theo dấu ; ở cuối cùng.

Ví dụ:

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

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

2.2.2. 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);
}
```

C xem một khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác, chỗ nào có thể viết được một câu lệnh thì ở đó cũng có thể 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. Nói một cách khác là 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 lệnh ta khai báo và sử dụng một biến hay một mảng có tên là a thì điều này cũng không ảnh hưởng đến một biến khác cũng có tên là a (nếu có) được khai báo và 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.

2.3. Cấu trúc cơ bản của chương trình C:

Cấu trúc chương trình và hàm là một trong các vấn đề quan trọng của C. 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 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 hàm phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện từ câu lệnh đầu tiên của hàm main() và kết thúc khi gặp dấu } cuối cùng của hàm này hoặc gặp lệnh return. 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 dòng <...> là vị trí có thể đặt: các dòng lệnh #include ... (dùng để khai báo sử dụng các hàm chuẩn), các dòng lệnh #define ... (dùng để định nghĩa các hằng), định nghĩa kiểu dữ liệu bằng typedef, khai báo các biến ngoài, mảng ngoài, ... (các hàm viết sau khai báo các biến ngoài, mảng ngoài,… này có thể sử dụng chúng)

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

Sử dụng tham số của hàm.

Sử dụng biến ngoài, mảng ngoài, ...

Tóm lại cấu trúc cơ bản của chương trình C 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).

Các tệp chương trình nguồn của ngôn ngữ C có phần mở rộng là **.C**

Ví dụ: Chương trình tính x lũy thừa y và in ra màn hình:

```
#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 */
        printf("\n x= %8.2lf \n y=%8.2lf \n z=%8.2lf",x,y,z);
    }
```

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

Qui tắc 1: Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu **;**

Qui tắc 2: Các lời giải thích cần được đặt giữa các dấu /* và */ và có thể được viết trên một dòng, trên nhiều dòng hoặc trên phần còn lại của dòng.

Qui tắc 3: Trong chương trình, khi cần sử dụng các hàm chuẩn chúng ta phải gọi các files chứa các hàm chuẩn đó vào chương trình bằng lệnh #include, ví dụ cần sử dụng các hàm printf(), getch(),... mà các hàm này lại được chứa ở trong file stdio.h trong thư mục INCLUDE của C thì ở đầu chương trình ta phải khai báo sử dụng như sau: #include "stdio.h "

Qui tắc 4: Một chương trình có thể chỉ có một hàm chính (hàm main()) hoặc có thể có thêm vài hàm khác.

Bài 3: CÁC LỆNH VÀO RA

Thư viện vào/ra chuẩn là một tập các hàm được thiết kế sẵn để cung cấp một hệ thống vào/ra chuẩn cho các chương trình C.

3.1. Thư viện các hàm vào/ra chuẩn:

Mỗi tệp chương trình muốn sử dụng các hàm thư viện vào/ra chuẩn đều phải có các dòng lệnh:

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

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

ở đầu tệp chương trình.

Dùng dấu ngoặc < và > thay cho dấu nháy kép để chỉ thị cho trình biên dịch tìm kiếm tệp tương ứng trong thư mục INCLUDE của C.

3.2. Các hàm vào/ra chuẩn - getchar() và putchar(); getch() và putch():

3.2.1. Hàm getchar():

Cơ chế vào đơn giản nhất là đọc từng ký tự từ thiết bị vào chuẩn (nói chung là bàn phím) bằng hàm getchar().

Cú pháp: biến = getchar();

Nhận một ký tự vào từ bàn phím và ấn Enter để xách nhận. Hàm sẽ trả về ký tự nhận được và lưu vào biến. Ký tự nhập vào được hiển thị lên màn hình.

Ví dụ: int c;

```
c = getchar()
```

3.2.2. Hàm putchar():

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

Cú pháp: putchar(ch);

Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự in lên màn hình luôn có màu trắng.

Ví dụ: int c;

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

3.2.3. Hàm getch():

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

Cú pháp: getch();

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 cũng sẽ chứa ký tự đọc vào.

Ví dụ: c = getch();

3.2.4. Hàm putch():

Cú pháp: putch(ch);

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.

3.3. Đưa kết quả lên màn hình bằng printf:

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

Chức năng: Hàm printf thực hiện các công việc sau: chuyển đổi kiểu dữ liệu, tạo khuôn dạng và in các đối số của nó ra thiết bị ra chuẩn dưới sự điều khiển của xâu *điều khiển.*

Xâu *điều khiển* chứa hai kiểu dữ liệu:

Các ký tự thông thường, chúng sẽ được đưa ra trực tiếp.

Các đặc tả định dạng dữ liệu, mỗi đặc tả sẽ thực hiện việc định dạng và in giá trị của đối số tương ứng của lệnh 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 định dạng dữ liệu như sau: %[-][fw][.pp]<kí tự định dạng>

Mỗi đặc tả định dạng dữ liệu đều được đưa vào bằng ký tự % và kết thúc bởi một <ký tự định dạng>. Giữa ký tự % và <ký tự định dạng> có thể có:

Dấu trừ:

Khi không có dấu trừ thì kết quả in ra sẽ được căn theo bên phải nếu độ dài thực tế của kết quả in ra nhỏ hơn giá trị của tham số fw. 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 căn theo bên trái và các vị trí dư thừa ở bên phải (nếu có) luôn được lấp đầy bằng các khoảng trống.

fw: Khi fw lớn hơn độ dài thực tế của kết quả in 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 in ra 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 số tương ứng.

Ví dụ:

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

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

Khi đối số tương ứng có giá trị kiểu float hay double thì pp là độ chính xác của giá trị in ra. Khi vắng mặt pp thì độ chính xác sẽ được xem là bằng 6.

Khi đối số tương ứng 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 Độ dài

				ra	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 dữ liệu 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 số tương ứng. Như vậy sẽ có tình trạng cùng giá trị 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 định 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.
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
\'	'
\"	"
\\	\

Ví dụ:

```
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,y);
Vì n=8 tương ứng với vị trí *
```

3.4. Vào số liệu từ bàn phím bằng hàm scanf:

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

Cú pháp: scanf(điều khiển, biến 1, biến 2, ...);

Xâu *điều khiển* chứa các đặc tả định dạng dữ liệu, mỗi đặc tả sẽ thực hiện việc định dạng biến tương ứng của lệnh scanf. Đặc tả định dạng có thể viết một cách tổng quát như sau: %[*][d...d]<ký tự định dạng>

Dấu * nói lên rằng dữ liệu đầu 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 biến 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 dữ liệu đầu vào vào tương ứng thì toàn bộ dữ liệu vào sẽ được đọc và giá trị của nó được gán cho biến có địa chỉ tương ứng (nếu không có dấu *).

Nếu giá trị của d...d nhỏ hơn độ dài của dữ liệu đầu vào thì chỉ phần đầu của dữ liệu đầu vào có kích cỡ bằng d...d được đọc và gán cho biến có địa chỉ tương ứng. Phần còn lại của dữ liệu đầu vào sẽ được xem xét bởi các đặc tả và biến 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&ct);

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 định dạng: Ký tự định dạng xác định cách thức dò đọc các ký tự trên dòng dữ liệu đầu vào cũng như cách chuyển đổi kiểu dữ liệu trước khi gán nó cho các biến có địa chỉ tương ứng.

Các ký tự chuyển dạng và ý nghĩa của nó:

c	Vào một ký tự, biến tương ứng là con trỏ ký tự. Có xét ký tự khoảng trắng
d	Vào một giá trị kiểu int, biến tương ứng là con trỏ kiểu int. Dữ liệu đầu vào phải vào là số nguyên.
ld	Vào một giá trị kiểu long, biến tương ứng là con trỏ kiểu long. Dữ liệu đầu vào phải vào là số nguyên
o	Vào một giá trị kiểu int hệ 8, biến tương ứng là con trỏ kiểu int. Dữ liệu đầu vào phải vào là số nguyên hệ 8
lo	Vào một giá trị kiểu long hệ 8, biến tương ứng là con trỏ kiểu long. Dữ liệu đầu vào phải vào là số nguyên hệ 8
x	Vào một giá trị kiểu int hệ 16, biến tương ứng là con trỏ kiểu int. Dữ liệu đầu vào phải vào là số nguyên hệ 16
lx	Vào một giá trị kiểu long hệ 16, biến tương ứng là con trỏ kiểu long. Dữ liệu đầu vào phải vào là số nguyên hệ 16
f hay e	Vào một giá trị kiểu float, biến tương ứng là con trỏ float, dữ liệu đầu vào

	phải là số dấu phảy động
lf hay le	Vào một giá trị kiểu double, biến tương ứng là con trỏ double, dữ liệu đầu vào phải là số dấu phảy động
s	Vào một giá trị kiểu double, biến tương ứng là con trỏ kiểu char, dữ liệu đầu vào phải là dãy ký tự bất kỳ không chứa các dấu cách và các dấu xuống dòng

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 nha 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).

…

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, đối số 1, đối số 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.

Bài 4: TÓAN TỬ VÀ BIỂU THỨC

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

Biểu thức được lập 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 trả về một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức. Biểu thức thường được dùng trong:

Vế phải của câu lệnh gán.

Làm tham số thực sự của hàm (trong trường hợp truyền tham số theo giá trị).

Làm chỉ số cho các phần tử của một mảng.

Trong các biểu thức điều kiện của các cấu trúc điều khiển.

3.2. Lệnh gán:

Lệnh gán có dạng: v=e;

Trong đó v là một biến (hay phần tử của mảng), e là một biểu thức.

Lệnh gán có thể sử dụng trong các câu lệnh và các biểu thức khác.

Ví dụ 1: khi ta viết a=b=5; thì điều đó có nghĩa là gán giá trị của biểu thức gán b=5 cho biến a. Kết qủa là b=5 và a=5.

Ví dụ 2: 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 số học hai ngôi gồm:

Phép toán	**Ý nghĩa**	**Ví dụ**
+	Phép cộng	a+b
-	Phép trừ	a-b
*	Phép nhân	a*b

/	Phép chia	a/b
%	Phép lấy phần dư	a%b

Phép toán trừ 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 +, - có cùng thứ tự ưu tiên và 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.

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:

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

Chú ý: 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 thứ tự ưu tiên nhỏ hơn so với phép toán phủ định một ngôi !, nhưng lớn hơn so với && và ||, vì vậy biểu thức như:

(a<b)&&(c>d) là tương đương với biểu thức a<b&&c>d

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. Các phép toán trên bít :

Các phép toán trên bít xem xét các toán hạng dưới dạng một chuỗi bit chứ không phải là giá trị số thông thường. Ví dụ xét toán hạng có giá trị là 12, các phép toán trên bít sẽ coi số 12 này như 1100.

Các phép toán trên bít gồm : &, | , ^ , ~ , vv … được tổng kết qua bảng sau:.

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

Các phép toán trên bít xem kiểu dữ liệu số như là số nhị phân 32-bit, giá trị số được đổi thành giá trị bit để tính toán trước rồi sau đó sẽ trả về kết quả ở dạng số ban đầu. Ví dụ:

Biểu thức 10 & 15 có nghĩa là (1010 & 1111) trả về giá trị 1010 có nghĩa là 10.

Biểu thức 10 | 15 có nghĩa là (1010 | 1111) trả về giá trị 1111 có nghĩa là 15.

Biểu thức 10 ^ 15 có nghĩa là (1010 ^ 1111) trả về giá trị 0101 có nghĩa là 5.

Biểu thức ~10 có nghĩa là (~1010) trả về giá trị 1111.1111.1111.1111.1111.1111.1111.0101 có nghĩa là -11.

3.7. 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	! ~ & - (trừ một ngôi) ++ -- (type) sizeof	Phải qua trái
3	* / %	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

Chú thích:

Các phép toán trên một dòng có cùng thứ tự ưu tiên, các phép toán ở hàng trên có số thứ tự ưu tiên cao hơn các phép toán 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: Biểu thức *--px tương đương với biểu thức *(--px) vì các phép toán *, -- cùng mức độ ưu tiên nhưng được thực hiện từ phải qua trái.

Biểu thức 8/4*6 tương đương với (8/4)*6 (thực hiện từ trái qua phải)

Nên dùng các dấu ngoặc tròn để viết biểu thức một cách rõ ràng nhất.

Các phép toán khác:

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

& Phép toán lấy địa chỉ, ví dụ: &x

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

3.8. 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 dữ liệu.

Khi một giá trị kiểu dữ liệu này được gán cho một biến (hoặc phần tử mảng) kiểu dữ liệu khác. Điều này xảy ra trong lệnh gán, trong việc truyền giá trị các tham số.

Ngoài ra, ta có thể chuyển giá trị từ một kiểu dữ liệu này sang một kiểu dữ liệu bất kỳ mà ta muốn bằng phép chuyển kiểu như sau: (type) <biểu thức>

Ví dụ: (float) (a+b);

/* chú ý thứ tự ưu tiên các phép toán*/

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

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

Chuyển đổi kiểu dữ liệu tự động 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. Điều này được gọi là tăng cấp kiểu. Sự phát triển về kiểu dữ liệu theo thứ tự sau: char < int <long <float <double. Kết quả thu được là một giá trị kiểu cao hơn. Chẳng hạn:

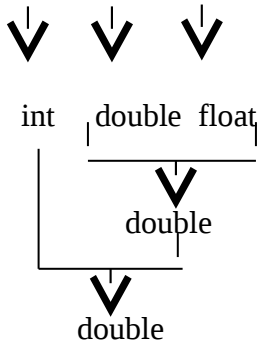
Giữa int và long thì int chuyển thành long.

Giữa int và float thì int chuyển thành float.

Giữa float và double thì float chuyển thành double.

Ví dụ:

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



Chuyển đổi kiểu thông qua phép gán:

Giá trị của vế phải được chuyển sang kiểu của 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 sau dấu phảy. Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int.

Ví dụ: int n;

n=15.6 /*giá trị của n là 15*/

Bài 5: CÁC CẤU TRÚC LỰA CHỌN

5.1. Cấu trúc lựa chọn - if-else:

Lệnh if cho phép chương trình lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào giá trị đúng hoặc sai của biểu thức điều kiện. Nó có hai cách viết như sau:

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

Dạng 1: Máy xác định 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.

Dạng 2: Máy xác định 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:Viết chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình.

Cách 1:

```
#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);
```

```
}
```

Cách 2:

```
#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 lệnh if:

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

Chú ý là máy sẽ ghép lệnh else với lệnh 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 thì ta viết như sau:

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

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 lần lượt từ biểu thức 1 trở đi. 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 sau khối lệnh n trong chương trình.

Nếu tất cả n-1 biểu thức đều có giá trị sai (bằng 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 – 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 phải nằm trong một hàm, có nghĩa là lệnh 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 lệnh 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: ");
    n1: printf("\n Gia tri cua N la: ");        .....
    .....                                     }
}                                             goto n1;
```

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 lựa chọn - switch:

Là cấu trúc tạo nhiều nhánh lựa chọn. Nó căn cứ vào giá trị của một biểu thức nguyên để chọn một trong nhiều nhánh.

Cú pháp:

```
switch (biểu thức nguyên )
  {
        case n1:
                khối lệnh 1
        case n2:
                khối lệnh 2
        .......
        case nk:
                khối lệnh k
        [ default:
                khối lệnh k+1 ]
}
```

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

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

Sự hoạt động của lệnh 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 của 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 lệnh 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 lệnh switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi lệnh 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 lệnh switch sẽ thực hiện nhiều nhất 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;
}
```

Bài 6: CÁC CẤU TRÚC LẶP

6.1. Cấu trúc lặp với lệnh while và for:

6.1.1. Cấu trúc lặp với lệnh while:

Cú pháp:

```
while (biểu thức điều kiện)
        Lệnh đơn hoặc khối lệnh;
```

Như vậy lệnh while gồm một biểu thức điều kiện và thân của vòng lặp. Thân của vòng lặp có thể là một lệnh đơn hoặc một khối lệnh.

Hoạt động của vòng lặp:

Bước 1: Máy xác định giá trị của biểu thức.

Bước 2: Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh đơn hoặc khối lệnh trong thân của while sau đó quay lại bước 1.

Bước 3: Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi vòng lặp while và chuyển tới thực hiện câu lệnh tiếp sau vòng lặp while trong chương trình.

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 vòng while lại có thể sử dụng các vòng while khác. Bằng cách đó ta đi xây dựng được các vòng lặp lồng nhau.

Khi gặp câu lệnh break trong thân vòng lặp while, máy sẽ thoát khỏi vòng lặp while chứa câu lệnh này.

Trong thân vòng lặp while có thể sử dụng lệnh goto để nhảy ra khỏi vòng lặp đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân vòng lặp 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);
    }
```

6.1.2. Cấu trúc lặp với lệnh for:

Cú pháp:

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

Lệnh đơn hoặc khối lệnh ;

Lệnh for gồm ba biểu thức và thân vòng for. Thân vòng for là một câu lệnh đơn 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 có 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 biểu thức quan hệ logic biểu thị điều kiện để tiếp tục vòng lặp, biểu thức ba là một toán tử gán dùng để thay đổi giá trị của biến điều khiển.

Hoạt động của lệnh for:

Bước 1: Tính biểu thức 1

Bước 2: Tính giá trị của biểu thức 2. Tuỳ thuộc vào giá trị đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh:

Nếu biểu thức 2 có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau for.

Nếu biểu thức 2 có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân vòng for một lần nữa sau đó thực hiện biểu thức 3 và quay lại bước 2 để bắt đầu một lượt mới của vòng lặp.

Chú ý:

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

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 vòng for khác, vì thế ta có thể xây dựng các vòng for lồng nhau.

Khi gặp câu lệnh break trong thân vòng for, máy ra sẽ ra khỏi vòng for sâu nhất chứa câu lệnh này. Trong thân vòng 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);/* số phần tử của mảng x*/
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;
        }
        printf(“\n Day so dao la \n\n”);
        for (i=0;i<n;++i)
        printf(“%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)
        printf(“\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;
        }
        printf(“\n Day so dao la \n\n”);
        for (i=-1;i++<n-1; printf(“%8.2f”,x[i]));
}
```

6.2. Vòng lặp do-while

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

Cú pháp:

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

Hoạt động của vòng lặp như sau:

Bước 1: Máy thực hiện các lệnh trong thân của vòng lặp.

Bước 2: 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 bước 1.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc vòng lặp và chuyển tới thực hiện lệnh đứng sau vòng lặp while.

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 ");
	}
```

6.3. Câu lệnh break:

Câu lệnh break cho phép ra khỏi các vòng lặp: for, while, do-while và lệnh switch. Khi có nhiều vòng lặp lồng nhau, câu lệnh break sẽ kết thúc vòng lặp bên trong nhất. 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);
        }
```

6.4. Câu lệnh continue:

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một lượt mới của vòng lặp chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện kiểm tra biểu thức điều kiện, còn trong lệnh for lệnh continue chuyển điều khiển về lệnh tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của vòng lặp.

Chú ý: Lệnh continue chỉ áp dụng cho vòng lặp chứ không áp dụng cho switch.

Ví dụ: Viết chương trình 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];
                        ++soptd;
                        if (cucdai<a[i][j]) cucdai=a[i][j];
                }
        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);
}
```

Bài 7: CON TRỎ VÀ MẢNG

Con trỏ là một biến chứa địa chỉ vùng nhớ của biến, chứ không lưu trữ giá trị của biến đó. Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là con trỏ trỏ đến biến thứ hai kia.

7.1. Con trỏ và địa chỉ:

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

Phép toán một ngôi &: Cho địa chỉ của biến tương ứng, nên câu lệnh sau: px=&x;

sẽ gán địa chỉ của biến x cho con trỏ px, và px bây giờ được gọi là con trỏ trỏ tới biến x. Phép toán & chỉ áp dụng được cho các biến và phần tử của mảng, không cho phép lấy địa chỉ của một biểu thức chung chung, ví dụ: &(x+1) và &3 là không hợp lệ.

Phép toán một ngôi *: Giả sử px là một con trỏ, khi đó *px sẽ cho kết quả là nội dung của ô nhớ do px trỏ đến.

Cú pháp khai báo biến con trỏ có dạng:

<tên kiểu> *<tên con trỏ>

Ví dụ: Khai báo con trỏ px kiểu int:

int *px;

Khai báo trên đã ngụ ý nói rằng 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.

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:

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 chúng ta khai báo: float a[10]; máy sẽ bố trí bố trí cho mảng a mười khoảng nhớ liên tiếp, mỗi khoảng nhớ là 4 byte. Như vậy, nếu biết địa chỉ của một

phần tử nào đó của mảng a, thì ta có thể dễ dàng suy ra địa chỉ của các phần tử khác của mảng.

Trong 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] của mảng a 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],*pa;
pa=a;
```

thì bốn cách viết sau có tác dụng như nhau và cùng truy cập đến phần tử thứ i của mảng a:

a[i] *(a+i) pa[i] *(pa+i)

Ví dụ: Vào số liệu cho các phần tử của một mảng và tính tổng các phần tử 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 dữ liệu.

7.2.4. Mảng, con trỏ và xâu ký tự:

Như ta đã biết trước đây, xâu ký tự là một dãy ký tự đặt trong hai dấu nháy kép, ví dụ như xâu ký tự: "Viet nam".

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

Cũng giống như tên mảng, xâu ký tự là một hằng địa chỉ biểu thị địa chỉ phần tử đầu tiên 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. 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 chúng ta sẽ xem xét 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" (1)
gets(xau);      (2)
```

Các câu lệnh (1) và (2) là không hợp lệ. Câu lệnh (1) sai vì ten là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác. Câu lệnh (2) không thực hiện được bởi 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 vì địa chỉ vùng nhớ của con trỏ **xau** còn chưa xác định. Nếu trỏ **xau** đã trỏ tới một vùng nhớ nào đó thì câu lệnh (2) hoàn toàn thực hiện được. Chẳng hạn như sau khi thực hiện câu lệnh:

xau=ten; thì cách viết: gets(ten) ; và gets(xau); đều cho phép nhập xâu kí tự vào con trỏ **xau**.

7.3. Con trỏ và mảng nhiều chiều:

7.3.1. 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	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
Địa chỉ	1	2	3	4	5	6

Tên mảng a biểu thị địa chỉ đầu tiên của mảng. Phép cộng địa chỉ ở đây được thực hiện như sau: C coi mảng hai chiều là mảng (một chiều) của mảng, như vậy khai báo

float a[2][3]; thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của mảng).

Vì vậy:

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

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

...

a+i trỏ phần tử đầu hàng thứ i của mảng: phần tử a[i][0].

7.3.2. 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;
```

khi đó:

pa trỏ tới a[0][0]

pa+1 trỏ tới a[0][1]

pa+2 trỏ tới a[0][2]

pa+3 trỏ tới a[1][0]

pa+4 trỏ tới a[1][1]

pa+5 trỏ tới a[1][2]

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

Cách 1:

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

Cách 2:

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

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

7.4.1. Kiểu con trỏ và kiểu địa chỉ:

Con trỏ dùng để lưu địa chỉ của biến. Mỗi kiểu địa chỉ của biến 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. Theo khai báo:

```
        float a[20][30],*pa,(*pm)[30];
/* Nếu khai báo float *p3[30]; thì p3 là mảng 30 con trỏ kiểu float;
p3[0]=a; */
```

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 (tuy nhiên sẽ có quá trình chuyển kiểu tự động).

Nhưng phép gán:

pm=a; là hợp lệ.

Ví dụ :float a[2][3]={{1.0,2.0,3.0}{4.0,5.0,6.0}}, *p, *mp3[3];

```
float b[3]; int i;
p=(float *)a;
pm3=a;
for (i=0;i<6;i++)
        printf("%f ", *(p+i));
b=(float*)pm3;
for (i=0;i<3;i++)
        printf("%f ", b[i]);
b=(float*)(pm3+1);
for (i=0;i<3;i++)
        printf("%f ", b[i]);
```

7.4.2. Các phép toán trên con trỏ:

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

Phép gán.

Phép tăng giảm địa chỉ.

Phép truy cập bộ nhớ.

Phép so sánh.

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

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

Phép tăng giảm địa chỉ: Để minh hoạ chi tiết cho phép toán này, ta xét ví dụ sau:

Các câu lệnh:

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

cho con trỏ px là con trỏ float trỏ tới phần tử x[10]. Ta có:

px+i trỏ tới phần tử x[10+i]

px-i trỏ tới phần tử x[10-i]

Phép truy cập bộ nhớ: Con trỏ float trỏ tới địa chỉ dài 4 byte, con trỏ int trỏ tới địa chỉ dài 2 byte, con trỏ char trỏ tới địa chỉ dài 1 byte. Giả sử ta có cá khai báo:

```
float *pf;
int *pi;
char *pc;
```

Khi đó ta có các nhận xét sau:

Nếu trỏ pf 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 1:

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

```
float a[100],*p,*pcuoi,tong=0.0;
pcuoi=(float*)(a+99); /* Địa chỉ cuối dãy*/
for (p=(float*)a;p<=pcuoi;++p)
    tong+=*p;
```

Ví dụ 2:

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

Giả sử ta có biến nguyên n được khai báo như sau:

```
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 địa chỉ kiểu bất kỳ. Chẳng hạn câu lệnh sau là hợp lệ:

```
void *pa;
float a[20][30];
pa=a;
```

Con trỏ void thường dùng làm tham số hình thức 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 xử 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 M);
    {       float *pa,*pb,*pc;
            int i,j;
            pa=(float*)a;   pb=(float*)b;   pc=(float*)c;
            for (i=1;i<N;++i)
                    for (j=1;j<M;++j)
                            *(pc+i*N+j)=*(pa+i*N+j)+*(pb+i*N+j);
    }
```

Vì tham số hình thức 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 con trỏ void trong thân hàm mà phải chuyển kiểu của chúng, trong trường hợp này là chuyển sang thành float.

7.5. Mảng con trỏ:

Mảng con trỏ là một mảng mà mỗi phần tử của nó là một con trỏ, có thể chứa được một địa chỉ của một biến nào đó. Mỗi phần tử của một mảng con trỏ kiểu int sẽ chứa được một địa chỉ kiểu int. Tương tự cho các mảng con trỏ của các kiểu khác.

Mảng con trỏ được khai báo theo mẫu:

<Kiểu dữ liệu> *<Tên_mảng_con_trỏ>[N];

Trong đó <Kiểu dữ liệu> 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. Ví dụ: double *pa[100]; Khai báo một mảng con trỏ kiểu double gồm 100 phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double.

Chú ý :

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

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

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

```
#include "stdio.h"
#include "ctype.h"
void tim(int code);
main()
        {
                int i;
                tt:printf("\n Tim nguoi co so TT la:");
                scanf("%d",&i);
                tim(i);
                printf("Co tiep tuc nua khong C/K: ');
                if (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. Cấp phát bộ nhớ cho biến con trỏ :

Trước khi sử dụng biến con trỏ, ta phải cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp các hàm:

void *malloc(size_t size): Cấp phát vùng nhớ có kích thước là size byte.

void *calloc(size_t nitems, size_t size): Cấp phát vùng nhớ có kích thước là nitems*size byte.

Ví dụ: Giả sử ta có khai báo:

```
int a, *pa, *pb;
pa = (int*)malloc(sizeof(int)); /* Cấp phát vùng nhớ có kích thước bằng với kích thước của một số nguyên */
pb= (int*)calloc(10, sizeof(int)); /* Cấp phát vùng nhớ có thể chứa được 10 số nguyên*/
```

7.7. Giải phóng vùng nhớ do biến con trỏ quản lý

Một vùng nhớ đã cấp phát cho biến con trỏ, khi không cần sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp: void free(void *block)

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Ví dụ: Ở ví dụ trên, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

free(pa);

free(pb);

7.8. Con trỏ tới hàm:

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

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

Ví dụ 1: Sử dụng câu lệnh: float (*f)(float),(*mf[50])(int);

Để khai báo:

f là con trỏ hàm kiểu float có một tham số cũng có kiểu float

mf là mảng 50 con trỏ hàm kiểu float có một tham số kiểu int

Ví dụ 2: Sử dụng câu lệnh: double (*g)(int, double),(*mg[30])(double, float);

Để khai báo:

g là con trỏ hàm kiểu double có các tham số kiểu int và double

mg là mảng con trỏ hàm kiểu double có các tham số kiểu double và float

7.8.2. 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));

	}
```
Ví dụ 3: Dùng mảng con trỏ để lập bảng giá trị cho các hàm: x*x, sin(x), cos(x), exp(x) và sqrt(x). Biến x chay từ 1.0 đến 10.0 theo bước 0.5
```
#include "stdio.h"
#include "math.h"
double bp(double x) /* Hàm tính x*x */
	{
		return x*x;
	}

main()
	{	int i,j;
		double x=1.0;
		typedef double (*ham)(double);
		ham f[6]; /* Khai bao mảng con trỏ hàm*/
		/* Có thể khai báo như sau double (*f[6](double)*/
		f[1]=bp; f[2]=sin; f[3]=cos; f[4]=exp; f[5]=sqrt;
		/* Gán tên hàm cho các phần tử mẩng 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;
                }
        }
```

7.8.3. Tham số của con trỏ hàm:

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

Nếu tham số của hàm được khai báo là con trỏ hàm, ví dụ: 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:

```
        #include <stdio.h>
        #include <conio.h>
        #include <math.h>
/* hàm gọi một hàm khác thông tham số con trỏ hàm */
double hamgoi(double x, double y, double (*hamthamso)(double, double))
        {
            return hamthamso(x,y);
        }

double sum(double x, double y)
        {       return x+y;
        }
main()
{
  printf("\nCon tro ham %f\n",hamgoi(4,5,sum) );
}
```

Bài 8: XÂU KÝ TỰ

Xâu kí tự trong C được cài đặt như là mảng của các ký tự, được kết thúc bởi ký tự NULL (‘\0’).

8.1. Các biến và hằng kiểu xâu kí tự

Các biến xâu kí tự được sử dụng để lưu trữ một dãy các ký tự. Như các biến khác, các biến này phải được khai báo trước khi sử dụng. Ví dụ khai báo một biến xâu kí tự:

char str[10]; str là một mảng các ký tự, nó có thể lưu tối đa 10 ký tự. Giả sử str được gán một hằng xâu kí tự: “WELL DONE”

Một hằng xâu kí tự là một dãy các ký tự nằm trong dấu nháy kép. Mỗi ký tự trong một xâu kí tự được lưu trữ như là một phần tử của mảng. Trong bộ nhớ, xâu kí tự được lưu trữ như sau:

‘W’	‘E’	‘L’	‘L’	‘ ’	‘D’	‘O’	‘N’	‘E’	‘\0’

Ký tự ‘\0’ (null) được tự động thêm vào trong cách biểu diễn bên trong của xâu kí tự để đánh dấu điểm kết thúc xâu kí tự. Vì vậy, khi khai báo một xâu kí tự, phải tăng kích thước của nó thêm một phần tử để chứa kí hiệu kết thúc null.

8.1.1. Con trỏ trỏ đến xâu kí tự

Xâu kí tự có thể được lưu và truy cập bằng cách sử dụng con trỏ kiểu ký tự. Một con trỏ kiểu ký tự trỏ đến một xâu kí tự được khai báo như sau: char *pstr = “WELCOME”;

pstr là một con trỏ được khởi tạo để trỏ đến một hằng xâu kí tự. Con trỏ pstr có thể thay đổi để trỏ đến bất kì một xâu kí tự nào khác.

8.1.2 Các thao tác nhập xuất xâu kí tự

Các thao tác nhập/xuất xâu kí tự trong C được thực hiện bằng cách gọi các hàm chuẩn. Một chương trình muốn sử dụng các hàm nhập/xuất xâu kí tự phải có câu lệnh khai báo sau ở đầu chương trình: #include <stdio.h>;

Khi chương trình có chứa câu lệnh này được biên dịch, thì nội dung của tập tin stdio.h sẽ trở thành một phần của chương trình.

8.1.2 .1. Các thao tác nhập/xuất xâu kí tự đơn giản:

Sử dụng hàm gets() là cách đơn giản nhất để nhập một xâu kí tự thông qua thiết bị nhập chuẩn. Các ký tự sẽ được nhập vào cho đến khi nhấn phím Enter. Hàm gets() thay thế ký tự kết thúc trở về đầu dòng ‘\n’ bằng ký tự ‘\0’. Cú pháp hàm này như sau: gets(str);

Trong đó str là một mảng ký tự đã được khai báo.

Tương tự, hàm puts() được sử dụng để hiển thị một xâu kí tự ra thiết bị xuất chuẩn. Cú pháp hàm như sau: puts(str);

Trong đó str là một mảng ký tự đã được khai báo và khởi tạo. Chương trình sau đây nhận vào xâu kí tự và hiển thị lên màn hình.

Ví dụ 1:

```
#include <stdio.h>
void main()
{       char name[20];
        clrscr();                               /* Xóa màn hình */
        puts("Enter your name:");       gets(name);
        puts("Hi there: ");             puts(name);
        getch();
}
```

Nếu tên Lisa được nhập vào, chương trình trên cho ra kết quả:

```
Enter your name:
Lisa
Hi there:
Lisa
```

8.1.2 .2. Các thao tác Nhập/Xuất xâu kí tự có định dạng:

Có thể sử dụng các hàm scanf() và printf() để nhập và hiển thị các giá trị xâu kí tự. Các hàm này được dùng để nhập và hiển thị các kiểu dữ liệu hỗn hợp trong một câu lệnh duy nhất. Cú pháp để nhập một xâu kí tự như sau: scanf(“%s”, str);

Trong đó ký hiệu định dạng %s cho biết rằng một giá trị xâu kí tự sẽ được nhập vào, str là một mảng ký tự đã được khai báo.

Tương tự, để hiển thị xâu kí tự, cú pháp sẽ là: printf(“%s”, str);

Trong đó ký hiệu định dạng %s cho biết rằng một giá trị xâu kí tự sẽ được hiển thị và str là một mảng ký tự đã được khai báo và khởi tạo. Hàm printf() có thể dùng để hiển thị ra các thông báo mà không cần kí tự định dạng.

Ví dụ 2: Dùng các hàm scanf và printf để nhập vào xâu kí tự, sau đó hiển thị lên màn hình.

```
#include <stdio.h>
void main()
{       char name[20];          clrscr();
        printf("Enter your name: ");    scanf(“%s”, name);
        printf("Hi there: %s", name); getch();
}
```

Nếu nhập vào tên Brendan , chương trình trên cho ra kết quả:

Enter your name: Brendan

Hi there: Brendan

8.2. Các hàm xử lý xâu kí tự

C hỗ trợ rất nhiều hàm về xâu kí tự. Các hàm này có thể tìm thấy trong tập tin string.h. Một số thao tác mà các hàm này thực hiện là:

Nối xâu kí tự

So sánh xâu kí tự

Định vị một ký tự trong xâu kí tự

Sao chép một xâu kí tự sang xâu kí tự khác

Xác định chiều dài của xâu kí tự.

8.2.1. Hàm strcat()

Hàm strcat() được sử dụng để ghép xâu kí tự, theo cú sau:

```
char * strcat(char *dest, const char *src);
```

trong đó str1 và str2 là hai xâu kí tự đã được khai báo và khởi tạo. Hàm này sẽ thực hiện nối xâu kí tự scr vào sau xâu kí tự dest.

Hàm này cũng trả về xâu kí tự dest.

Chương trình sau đây nhận vào họ và tên của một người, nối chúng với nhau và hiển thị ra họ tên đầy đủ.

Ví dụ 1:

```
#include<stdio.h>
#include<string.h>
void  main()
{       char firstname[15];
        char lastname[15];
        char *fullname;
        clrscr();
        printf("Enter your first name: "); scanf("%s", firstname);
        printf("Enter your last name:"); scanf("%s", lastname);
        fullname=strcat(firstname, lastname);
        printf("%s\n", firstname);
        printf("%s\n", fullname);
        getch();
}
```

Kết quả của chương trình trên được minh họa như sau:

Enter your first name: Carla

Enter your last name: Johnson

CarlaJohnson

8.2.2. Hàm strcmp()

Việc so sánh hai số có thể thực hiện bằng cách sử dụng các toán tử quan hệ. Tuy nhiên, để so sánh hai xâu kí tự kí tự, phải dùng một hàm. Hàm strcmp() so sánh hai xâu kí tự với nhau và trả về một số nguyên phụ thuộc vào kết quả so sánh. Cú pháp của hàm strcmp() như sau:

```
int   strcmp(const char *s1, const char *s2);
```

trong đó str1 và str2 là hai xâu kí tự đã được khai báo và khởi tạo. Hàm có thể trả về các giá trị sau:

nhỏ hơn 0 nếu str1<str2

0 nếu str1 = str2

lớn hơn 0 nếu str1>str2

Ví dụ 2: Chương trình so sánh biến name1 với các biến name2, name3, name4 và hiển thị kết quả của phép so sánh:

```
#include <stdio.h>
#include<string.h>
void  main()
{       char name1[15] = "Geena";
        char name2[15] = "Dorothy";
        char name3[15] = "Shania";
        char name4[15] = "Geena";
        int i;
        clrscr();
        i = strcmp(name1,name2);
        printf("%s compared with %s returned %d\n", name1, name2, i);
        i=strcmp(name1, name3);
        printf("%s compared with %s returned %d\n", name1, name3, i);
        i=strcmp(name1,name4);
        printf("%s compared with %s returned %d\n", name1, name4, i);
        getch();
}
```

Kết quả của chương trình trên được minh họa như sau:

Geena compared with Dorothy returned 3

Geena compared with Shania returned -12

Geena compared with Geena returned 0

Chú ý: Giá trị trả về trong mỗi phép so sánh ở ví dụ trên chính là sự khác nhau về mã ASCII của hai kí tự khác nhau đầu tiên tìm thấy trong hai xâu kí tự.

8.2.3. Hàm strchr()

Hàm strchr() xác định vị trí xuất hiện của một ký tự trong một xâu kí tự. Cú pháp hàm là: char * strchr(const char *str, int chr);

trong đó str là một mảng ký tự hay con trỏ kí tự. chr là một biến int chứa mã ASCII của ký tự cần tìm. Hàm trả về con trỏ trỏ đến giá trị tìm được đầu tiên trong xâu kí tự, hoặc NULL nếu không tìm thấy.

Chương trình sau đây xác định liệu ký tự 'a' có xuất hiện trong tên hai thành phố hay không.

Ví dụ 3:

```
#include <stdio.h>
#include<string.h>
void  main()
{
        char str1[15] = "New York";
        char str2[15] = "Washington";
        char chr = 'a', *loc;
        clrscr();
        loc = strchr(str1, chr);
if(loc != NULL)
        printf("%c occurs in %s\n", chr, str1);
else
        printf("%c does not occur in %s\n", chr, str1);
loc = strchr(str2, chr);
if(loc != NULL)
        printf("%c occurs in %s\n", chr, str2);
else
        printf("%c does not occur in %s\n", chr, str2);
getch();
}
```

Kết quả của chương trình trên được minh họa như sau:

a does not occur in New York

a occurs in Washington

8.2.4. Hàm strcpy()

Cú pháp hàm là: char * strcpy(char *dest, const char *src);

trong đó dest và scr là hai mảng ký tự đã được khai báo và khởi tạo. Hàm sao chép giá trị scr vào dest và trả về xâu kí tự dest.

Ví dụ 4: Chương trình sau đây minh họa việc sử dụng hàm strcpy(). Nó thay đổi tên của một khách sạn và hiển thị tên mới.

```
#include <stdio.h>
#include<string.h>
void  main()
{
        char hotelname1[15] = "Sea View";
        char hotelname2[15] = "Sea Breeze";
        clrscr();
        printf("The old name is %s\n", hotelname1);
        strcpy(hotelname1, hotelname2);
        printf("The new name is %s\n", hotelname1);
        getch();
}
```

Kết quả của chương trình trên được minh họa như sau:

The old name is Sea View

The new name is Sea Breeze

8.2.5 Hàm strlen()

Hàm strlen() trả về chiều dài của xâu kí tự. Chiều dài của xâu kí tự rất hay được sử dụng trong các vòng lặp truy cập từng kí tự của xâu kí tự.

Cú pháp của hàm là: int strlen(char* str);

trong đó str là mảng ký tự đã được khai báo và khởi tạo. Hàm trả về chiều dài của xâu kí tự str.

Ví dụ 5: Chương trình sau đây đưa ra ví dụ đơn giản sử dụng hàm strlen(). Nó tìm chiều dài của tên một công ty và hiển thị tên công ty đó với các ký tự được phân cách nhau bởi ký tự '*'.

```
#include<stdio.h>
#include<string.h>
```

```
void  main()
{       char compname[20] = "Microsoft";
        int len, ctr;
        clrscr();
        len = strlen(compname);
        for(ctr = 0; ctr < len; ctr++)
                printf("%c * ", compname[ctr]);
        getch();
}
```

Kết quả của chương trình trên được minh họa như sau:

M * i * c * r * o * s * o * f * t *

8.2.6. Ý nghĩa của một số hàm thường dùng khác trong thư viện string.h

char* strcat(char* s1, char* s2): ghép chuỗi s2 vào chuỗi s1.

char* strchr(char* s, int ch): trả về địa chỉ của kí tự ch trong s (việc tìm được thực hiện từ trái sang phải), nếu không trả về NULL.

char* strrchr(char* s, int ch): tương tự hàm trên, trả về địa chỉ của kí tự ch trong s (việc tìm được thực hiện từ phải sang trái), nếu không trả về NULL.

int strcmp(char* s1, char* s2): trả về âm nếu s1 < s2, 0 nếu s1 == s2, dương nếu s1 > s2

int strcmpi(char* s1, char* s2): tương tự như trên, không phân biệt hoa, thường.

char strcpy(char* s1, char* s2): sao nội dung chuỗi s2 vào chuỗi s1.

int strcspn(char* s1, char* s2): trả về độ dài đoạn đầu tiên lớn nhất cúa s1 mà mọi kí tự của đoạn không có mặt trong chuỗi s2.

int strspn(char* s1, char* s2): trả về độ dài đoạn đầu tiên dài nhất mà hai xâu s1 và s2 giống nhau.

char* strdup(char* s): cấp phát vùng nhớ gấp đôi chuỗi s, nếu không được trả về NULL.

int stricmp(char* s1, char* s2): giống strcmpi.

int strlen(char* s): trả về độ dài xâu.

char* strlwr(char* s): chuyển chữ hoa thành chữ thường.

char* strncat(char* s1, char* s, int n): ghép n kí tự đầu của chuỗi s2 vào chuỗi s1.

int strncmp(char* s1, char* s2, int n): so sánh n kí tự đầu tiên của 2 xâu.

int strnicmp(char* s1, char* s2, int n): tương tự trên, không phân biệt hoa, thường.

char* strncpy(char* s1, char* s2, int n): sao n kí tự đầu tiên của s2 sang s1.

char* strnset(char* s, int c, int n): gán n lần kí tự c cho xâu s.

char* strpbrk(char* s1, char* s2): trả về kí tự đầu tiên trong xâu s2 có xuất hiện trong xâu s1,

nếu không trả về NULL.

char* strrev(char* s): đảo ngược chuỗi kí tự.

char* strset(char* s, int kt): tạo xâu s toàn kí tự kt.

char* strstr(char* s1, char* s2): trả về địa chỉ của xâu con trong s2 trùng với xâu s1, nếu không trả về NULL.

char* strupr(char* s): chuyển chữ thường thành chữ hoa.

char * strtok(char *s1, const char *s2); chia cắt xâu s1 theo dấu hiệu s2.

8.2.7. Ví dụ

Viết chương trình nhập vào một xâu kí tự gồm các từ cách nhau bởi một dấu cách và in ra các từ này, mỗi từ trên một dòng.

Cách 1: Sử dụng hàm strtok()

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
 main(void)
 {
   int i;
   char *p,input[32] = "Tran Van Anh va Trieu The Hung";

        p = strtok(input, " ");
        printf("%s\n", p);

        for(i=1;i++;i<strlen(input))
    {
              p = strtok(NULL, " ");
              if (p)   printf("%s\n", p);
    }
   return 0;
 }
```

Cách 2:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
        char ch[20];
        int i,len;
        printf("\nNhap vao mot chuoi : ");
        gets(ch);
        len = strlen(ch);
        for(i=0;i<len;i++)
        {
         if(ch[i]!=' ')
                printf("%c",ch[i]);
         else
                printf("\n");
```

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

Bài 9: CẤU TRÚC

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

Một ví dụ thường sử dụng cấu trúc là 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, … một số trong các thuộc tính này lại có thể là một cấu trúc bởi trong nó có thể chứa nhiều thành phần: Tên (Họ, đệm, tên), Địa chỉ (Phố, số nhà), ...

9.1. Kiểu cấu trúc:

Trước khi sử dụng cấu trúc, ta cần định nghĩa kiểu của cấu trúc đó. Điều này cũng tương tự như việc ta 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.
- thành phần của cấu trúc có thể là: biến, mảng, cấu trúc khác đã được định nghĩa trước đó.

Ví dụ 1:

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

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

Ví dụ 2:

Tạo ra kiểu cấu trúc có tên là **nhancong** gồm có năm thành phần. Ba thành phần đầu là ten, diachi, bacluong. 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.

```
struct nhancong
        {

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

Định nghĩa cấu trúc bằng typedef:

Cú pháp: typedef <type difinition> <identifier> ;

Có thể dùng toán tử typedef để định nghĩa các kiểu dữ liệu mới có cấu trúc là: **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;
```

Đặc tính typedef đặc biệt tiện lợi khi định nghĩa các cấu trúc, vì ta không cần nhắc lại từ khóa struct mỗi khi cần khai báo một biến theo cấu trúc đó.

Ví dụ:

```
typedef struct
    {
        int day;
        int month;
        int year;
    } date;
```

date due_date; /* không cần nhắc lại từ khóa struct*/

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

Khai báo biến kiểu cấu trúc hoàn toàn giống như việc khai báo các biến và các mảng kiểu thông thường khác. Giả sử ta đã khai báo các kiểu cấu trúc **ngay** và **nhancong** như trong mục trên bằng từ khóa struct. Khi đó ta có thể khai báo các biến như sau:

Ví dụ 1: struct ngay ngaydi, ngayden;/* cần nhắc lại từ khóa struct */

sẽ cho ta hai biến với tên là **ngaydi** và **ngayden** kiểu cấu trúc **ngay**.

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

Cách 1: struct <tên_kiểu_cấu_trúc_đã_khai_báo> <danh_sách_tên_biến>; (2)

Chú ý:

Các biến kiểu 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 kiểu cấu trúc cũng có thể thực hiện đồng thời với việc khai báo biến kiểu cấu trúc. Muốn vậy, chỉ cần đặt danh sách tên các biến cần khai báo sau dấu } .

Nói cách khác, để vừa khai báo kiểu cấu trúc 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 1:
```
struct ngay
        {
                int ngaythu;
                char thang[12];
                int nam;
        } ngaydi, ngayden;
```

Ví dụ 2:
```
struct nhancong
        {

                char ten[15];
                char diachi[20];
                double bacluong;
```

```
        struc ngay ngaysinh;
        struc ngay ngaybatdaucongtac;

    } nhom1, nhom2;
```

Khi vừa khai báo kiểu cấu trúc, vừa khai báo biến kiểu cấu trúc như trong ví dụ trên, ta có thể không cần chỉ định tên kiểu cấu trúc theo cú pháp 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 biến kiểu cấu trúc, và có thể dùng kiểu cấu trúc này để khai báo cho các biến kiểu cấu trúc khác như trong (2), còn (4) chỉ khai báo được các biến kiểu cấu trúc.

Chú ý: Nếu dùng từ khoá typedef để định nghĩa kiểu cấu trúc thì khi khai báo các biến cấu trúc ta không cần dùng từ khoá struct, chỉ cần dùng tên kiểu.

9.3. Truy cập đến các thành phần của 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 biến kiểu cấu trúc **ngaydi, ngayden**

Các cách viết còn lại 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 các biến kiểu 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 của cấu trúc để nhập số liệu trực tiếp vào các thành phần của cấu trúc. Ví dụ như ta viết:

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

Tuy nhiên ta nên nhập số liệu vào một biến trung gian sau đó mới gán cho thành phần của cấu trúc như sau:

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

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

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

Ví dụ: Giả sử mỗi dữ liệu về một cán bộ gồm:

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

Yêu cầu:

- Xây dựng cấu trúc dữ liệu lưu trữ thông tin về cán bộ.
- Viết chương trình vào số liệu của một cán bộ.

- Viết chương trình 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);
        }
```

9.4. 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 biến kiểu cấu trúc và mảng kiểu cấu trúc.

Cách khai báo mảng cấu trúc:

struct <tên_kiểu_cấu_trúc_đã_định_nghĩa> <tên_mảng[số phần tử của mảng]>;

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 và 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;
```

9.5. Phép gán cấu trúc:

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

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

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

Ví dụ: Đoạn chương trình sau minh hoạ phép gán cấu trúc để sắp xếp n thí sinh theo thứ tự tăng dần của tổng điểm:

```
#define N 100
struct thisinh
        {       char ht[25];
                float td;
        } tg,ts[N];
main()
{int i,j,min;
    for (i=0;i<N;++i)
```

```
    {       min =i;
            for (j=i+1;j<N;++j)
                    if (ts[min].td>ts[j].td) min=j;
            if (min!=i)
                    {       tg=ts[i];
                            ts[i]=ts[j];
                            ts[j]=tg;
                    }
    }
}
```

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

9.6.1. 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 ta khai báo:

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

thì ta sẽ có:

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

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

Ví dụ:

```
struct nhancong *p,*p1,*p2,nc1,nc2,ds[100];
        p1=&nc1;        /* Gán địa chỉ nc1 cho p1 */
        p2=&ds[4];      /* Gán địa chỉ ds[4] cho p2 */
```

```
p=ds;            /* Gán địa chỉ ds[0] cho p */
```

9.6.2. Truy nhập qua con trỏ:

Có thể truy nhập đến các thành phần của cấu trúc 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: struct nhancong *p,*p1,*p2,nc1,nc2,ds[100];

nc1.ngaysinh.nam

p1-> ngaysinh.nam

ds[4].ngaysinh.thang

(*p2). ngaysinh.thang

9.6.3. Phép gán qua con trỏ:

Giả sử ta gán:

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

khi đó ta có thể dùng:

*p1 thay cho nc1

*p2 thay cho ds[4]

tức là việc chúng ta viết:

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

tương đương với:

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

9.6.4. Phép cộng địa chỉ:

Sau các phép gán:

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

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

Ví dụ: Sau các lệnh:

```
p=p+10;
```

p2=p2-4;

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

9.6.5. 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 của 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)

Bài 10: HÀM

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó phải 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 viết trong chương trình là bất kỳ, song chương trình bao giờ cũng bắt đầu thực hiện từ hàm main().

10.1. Giới thiệu về hàm:

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

10.1.1. Cấu trúc của hàm

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các tham số hình thức 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 tham số hình thức của hàm )
{
        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 tham số hình thức của hàm. Khai báo các tham số hình thức có theo dạng: <kiểu tham số 1> <tên tham số 1>, <kiểu tham số 2> <tên tham số 2>,..., <kiểu tham số n> <tên tham số n>. Ví dụ: float max3s(float a, float b, float c)

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: Viết chương trình tìm giá trị lớn nhất của ba số mà giá trị mà giá trị của chúng được đưa vào từ bàn phím.

Chúng ta tổ chức thành hai hàm: Hàm main() và hàm max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số đã được 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 như trên để tìm giá trị lớn nhất của ba số và 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*/
```

10.1.2. Quy tắc hoạt động của hàm:

Một cách tổng quát lời gọi hàm có dạng sau:

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

Số các tham số thực sự trong danh sách các tham số thực sự khi gọi hàm phải bằng số tham số hình thức khi định nghĩa hàm và lần lượt tương ứng chúng phải có kiểu giống 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 vùng nhớ cho các biến cục bộ của .

Nhận giá trị từ các tham số thực.

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 biến cục bộ, các tham số hình thức trong trường tham số được truyền theo giá trị 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.

10.1.3. Tham số hình thức và biến cục bộ:

Tham số hình thức và biến cục bộ đều chỉ có phạm vi hoạt động trong hàm mà chúng được khai báo, do đó tham số thực sự và biến cục bộ phải có tên khác nhau.

Tham số hình thức và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được gọi đến và bị xoá khi ra khỏi hàm.

Tham số hình thức và biến cục bộ chỉ có phạm vi hoạt động trong hàm hiện thời, do đó có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

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.

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

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

Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị của các tham số thực sự.

10.1.4. Truyền tham số thực sự cho hàm:

Các tham số thực sự được truyền cho hàm theo một trong hai cách sau:

Truyền tham số thực sự theo giá trị

Truyền tham số thực sự theo tham chiếu

10.1.4.1. Truyền tham số theo giá trị

Mặc nhiên trong C, tất cả các tham số thực sự được truyền cho hàm theo giá trị (truyền giá trị của tham số thực sự cho hàm). Khi đó các tham số hình thức sử dụng các vùng nhớ riêng và là bản sao của các tham số thực sự, do đó hàm được gọi không thể thay đổi giá trị của các tham số thực sự. Xem ví dụ sau:

```
#include <stdio.h>
main()
{       int a, b, c;
        a = b = c = 0;
        printf("\nEnter 1st integer: ");
```

```
        scanf("%d", &a);
        printf("\nEnter 2nd integer: ");
        scanf("%d", &b);
        c = adder(a, b);
        printf("\n\na & b in main() are: %d, % d", a, b);
        printf("\n\nc in main() is: %d", c);
        /* c gives the addition of a and b */
}

adder(int a, int b)
{       int c;
        c = a + b;
        a *= a;
        b += 5;
        printf("\n\na & b within adder function are: %d, %d ", a, b);
        printf("\nc within adder function is : %d",c);
        return(c);
}
```

Kết quả thực thi khi nhập vào 2 và 4:

a & b in main() are: 2, 4

c in main() is: 6

a & b within adder function are: 4, 9

c within adder function is : 6

Các biến được sử dụng trong hàm **main**() và **adder**() có cùng tên. Tuy nhiên, chúng được lưu trữ trong các vị trí bộ nhớ khác nhau. Điều này được thấy rõ từ kết quả của chương trình trên. Các biến a và b trong hàm **adder**() được thay đổi từ 2 và 4 thành 4 và 9. Tuy nhiên, sự thay đổi này không ảnh hưởng đến các giá trị của a và b trong hàm **main**(). Biến **c** trong **main**() thì khác với biến **c** trong **adder**().

Tóm lại, khi các tham số thực sự được **truyền theo giá trị** thì những thay đổi của các tham số hình thức không hưởng đến giá trị của các tham số thực sự tương ứng.

10.1.4.2. Truyền bằng tham chiếu

Khi các tham số thực sự được truyền cho hàm theo giá trị thì hàm không thể thay đổi giá trị của chúng. Nếu chúng ta muốn thay đổi giá trị của các tham số thực sự thì chúng ta cần **truyền chúng theo tham chiếu**. Khi **truyền theo tham chiếu (**truyền địa chỉ của tham

số thực sự cho hàm**)**, hàm được phép truy xuất đến vùng bộ nhớ thực của các tham số thực sự và vì vậy có thể thay đổi giá trị của chúng.

Ví dụ: Xét một hàm nhận hai đối số và hoán vị giá trị của chúng. Chương trình giống như chương trình dưới đây sẽ không bao giờ thực hiện được mục đích này.

```
#include <stdio.h>
main()
{
    int x, y;
    x = 15; y = 20;
    printf("x = %d, y = %d\n", x, y);
    swap(x, y);
    printf("\nAfter interchanging x = %d, y = %d\n", x, y);
}
swap(int u, int v)
{
    int temp;
    temp = u;
    u = v;
    v = temp;
    return;
}
```

Kết quả của chương trình trên như sau:

```
x = 15, y = 20
After interchanging x = 15, y = 20
```

Hàm **swap**() hoán vị các giá trị của u và v, nhưng không làm thay đổi giá trị của hai tham số thực sự của hàm là **x** và **y** trong **main**(). Điều này là bởi vì các tham số hình thức **u**, **v** trong **swap()** và các tham số thực sự **x, y** của hàm - các biến **x**, **y** trong **main**() được lưu trữ ở các vùng nhớ khác nhau.

Để thực hiện được mục đích trên, các tham số hình thức của hàm **swap()** phải là các con trỏ - tức là phải có một dấu * ở phía trước tên của tham số. Các tham số thực sự tương ứng với các tham số hình thức kiểu con trỏ có thể là một biến con trỏ hoặc một biến được tham chiếu theo dạng &tênbiến. Khi đó để thực hiện việc hoán vị hai số, ta viết lại hàm **main()** và hàm **swap**() như sau :

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

```
{
        int x, y, *px, *py;
        /* Storing address of x in px */
        px = &x;
        /* Storing address of y in py */
        py = &y;
        x = 15; y = 20;
        printf("x = %d, y = %d \n", x, y);
        swap (px, py);
        /* Passing addresses of x and y */
        printf("\n After interchanging x = %d, y = %d\n", x, y);
}
swap(int *u, int *v)
/*      Accept the values of px and py into u and v */
{
        int temp;
        temp = *u;
        *u = *v;
        *v = temp;
        return;
}
```

Kết quả của chương trình trên như sau:

```
x = 15, y = 20
After interchanging x = 20, y = 15
```

Hai biến kiểu con trỏ **px** và **py** được khai báo, và địa chỉ của biến **x** và **y** được gán cho chúng. Sau đó các biến con trỏ được truyền cho hàm **swap()**, hàm này hoán vị các giá trị lưu trong x và y thông qua các con trỏ.

Chú ý : Truyền theo tham chiếu chỉ áp dụng đối với biến, không áp dụng với biểu thức chung chung.

10.2. Hàm không trả về giá trị:

Các hàm không cho giá trị trả về giống như thủ tục (procedure) trong các ngôn ngữ lập trình khác như: PASCAL, VB, ... 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 hienthimax3s(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);

10.3. Truyền mảng vào hàm

Trong C, khi một mảng được truyền vào hàm như một tham số, thì chỉ có địa chỉ của mảng được truyền vào. Tên mảng không kèm theo chỉ số chính là địa chỉ của mảng. Đoạn mã dưới đây mô tả cách truyền địa chỉ của mảng ary cho hàm fn_ary():

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

Nếu tham số của hàm là mảng một chiều thì tham số đó có thể được khai báo theo một trong các cách sau:

```
Cách 1: fn_ary (int ary [10])    /* sized array */
        {
        }
Cách 2:fn_arry (int ary [])     /*unsized array */
        {
        }
```

Cả hai khai báo ở trên đều cho cùng kết quả. Kiểu thứ nhất sử dụng cách khai báo mảng chuẩn, chỉ rõ ra kích thước của mảng. Kiểu thứ hai, chỉ ra rằng tham số là một mảng kiểu int có kích thước bất kì.

Ví dụ 1: Chương trình sau đây nhập các số vào một mảng số nguyên. Sau đó mảng này sẽ được truyền vào hàm sum_arr(). Hàm sẽ tính toán và trả về tổng của các số nguyên trong mảng.

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

```
{
        int num[5], ctr, sum = 0;
        int sum_arr(int num_arr[]); /* Function declaration */
        clrscr();
        for(ctr = 0; ctr < 5; ctr++) /*Accepts numbers into the array*/
        {       printf("\nEnter number %d: ", ctr+1);          scanf("%d", &num[ctr]);
        }
        sum = sum_arr(num);
        printf("\nThe sum of the array is %d", sum);
        getch();
}
int sum_arr(int num_arr[]) /* Function definition */
{       int i, total;
        for(i = 0, total = 0; i < 5; i++)
                total += num_arr[i];
        return total;
}
```

Kết quả của chương trình trên được minh họa như sau:

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

10.4. Truyền xâu kí tự vào hàm

Xâu kí tự, hay mảng ký tự, có thể được truyền vào hàm. Ví dụ, chương trình sau đây sẽ nhận vào các xâu kí tự và lưu trong một mảng ký tự hai chiều. Sau đó, mảng này sẽ được truyền vào trong một hàm dùng để tìm xâu kí tự dài nhất trong mảng đó.

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

```
        }
        longctr = longest(lines);/* Passes the array to the function */
        printf("\nThe longest string is %s", lines[longctr]);
        getch();
}
int longest(char lines_arr[][20]) /* Function definition */
{       int i = 0, l_ctr = 0, prev_len, new_len;
        prev_len = strlen(lines_arr[i]); /* Determines the length of the first element */
        for(i++; i < 5; i++)
        {       new_len = strlen(lines_arr[i]); /* Determines the length of the next element */
                if(new_len > prev_len)
                        l_ctr = i; /* Stores the subscript of the longer string */
                prev_len = new_len;
        }
        return l_ctr;
}
```

Kết quả của chương trình trên được minh họa như sau:

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

10.5. Hàm đệ qui:

10.5.1. Mở đầu:

C không những cho phép từ một 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> và <các tham số nhận theo giá trị> mới hoàn toàn độc lập với tập <các biến cục bộ> và <các tham số nhận theo giá trị> 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;
}
```

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

Giải thích hoạt động của hàm đệ qui:

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 có tham số hình thức của hàm - tham số n. Ta gọi tham số n được tạo ra lần thứ nhất là n thứ nhất. Giá trị của tham số thực sự (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 sau 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 được thực hiện. Máy sẽ tạo ra tham số n mới, ta gọi đó là n thứ hai. Giá trị của n-1 ở đây lại là tham số thực sự 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 n thứ ba và gán cho giá trị bằng 1 cho nó. Lúc này điều kiện trong lệnh if được thoả mãn, máy đi thực hiện câu lệnh:

return 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 lần ba. Kết quả là 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, 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 ở đây 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:

Khi dùng hàm đệ qui, máy tính 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 bằng vòng lặp (không dùng đệ qui) thì ta nên dùng cách này.

10.5.2. 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.

10.5.3. Cách xây dựng hàm đệ qui:

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

```
if (trường hợp suy biến)
    {
        Thực hiện 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

}

10.5.4. Các ví dụ về dùng hàm đệ qui:

Ví dụ 1: Bài toán dùng hàm 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');
 }
```

10.6. Bao hàm file:

Để dễ dàng sử dụng lại một tập các định nghĩa (#define) và khai báo (trong các file khác), C đưa ra cách bao hàm các file khác vào file đang dịch theo cú pháp: #include "**tên file**".

Dòng khai báo này sẽ được thay thế bởi nội dung của file có tên là **tên file**. Các #include được phép lồng nhau. Tất nhiên khi thay đổi file được bao hàm vào thì mọi file phụ thuộc vào nó đều phải dịch lại.

10.7. Phép thế Macro:

Cú pháp định nghĩa Macro có dạng: #define <biểu thức 1> [biểu thức 2]

Ví dụ 1: #define YES 1

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

Ví dụ 2: Một người lập trình ưa thích PASCAL có thể định nghĩa các macro sau:

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

sau đó ta có thể viết đoạn chương trình như sau trong C:

```
if (i>0) then
```

```
begin
        a=i;
        .....
end;
```

Ta cũng có thể định nghĩa các macro có tham số, do vậy giá trị thay thế sẽ phụ thuộc vào giá trị của các tham số được truyền cho macro.

Ví dụ 3: Định nghĩa macro tính max của hai số 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));

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

Ví dụ 4: 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
```

Bài 11: TẬP TIN - FILE

11.1. Khái niệm về tệp tin:

Tệp tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau, được chứa trong một thiết bị nhớ ngoài của máy tính (đĩa cứng, CD, ...) dưới một cái tên nào đó.

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. Mỗi hộp tương đương với một tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng, CD, … 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 chúng ta đã tắt máy tính. Chính vì lý do trên, những dữ liệu nào cần lưu trữ lâu dài (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 dữ liệu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong khi định nghĩa.

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

Nhập/xuất nhị phân:

Dữ liệu được ghi lên tệp theo các byte nhị phân, 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 bằng hằng số EOF (được định nghĩa trong stdio.h bằng -1) và khi đó 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).

Mã chuyển dòng được xử lý như sau:

Khi ghi, nếu gặp một ký tự LF (mã 10) thì nó sẽ được chuyển thành hai ký tự CR (mã 13) và LF (mã 10).

Khi đọc, hai ký tự liên tiếp là CR và LF trên tệp chỉ cho ta một ký tự LF

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

11.2.1. Khai báo sử dụng tệp:

Cú pháp: FILE <biến_con_trỏ_tệp>;

Trong đó <biến_con_trỏ_tệp> có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phảy.

Ví dụ: FILE *vb, *np; /* Khai báo hai biến con trỏ tệp */

11.2.2. Mở tệp - hàm fopen :

Cú pháp: FILE *fopen(const char *tên_tệp, const char *kiểu);

Nguyên hàm được định nghĩa trong trong: stdio.h .

Trong đó: đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Ý nghĩa:

Hàm dùng để mở tệp. Nếu thành công hàm trả về con trỏ kiểu FILE trỏ đến tệp vừa mở. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng sau chỉ ra các giá trị của kiểu truy nhập:

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

Chú ý: Trong các kiểu đọc/ghi, ta nên lầm sạch vùng đệm trước khi chuyển từ chế độ đọc sang chế độ ghi hoặc ngược lại.

Ví dụ: f=fopen("TEPNP","wb"); Mở một tệp mới có tên là TENNP để ghi theo kiểu nhị phân.

11.2.3. Đóng tệp - hàm fclose:

Cú pháp: int fclose(FILE *fp);

Nguyên được định nghĩa hàm trong: stdio.h .

Trong đó: fp là con trỏ ứng với tệp cần đóng.

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

- Nếu đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
- Nếu 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.

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

Cú pháp: int fcloseall(void);

Nguyên hàm được định nghĩa trong: stdio.h .

Ý nghĩa: 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, ngược lại nó trả về giá trị EOF.

11.2.5. Làm sạch vùng đệm - hàm fflush:

Cú pháp: int fflush(FILE *fp);

Nguyên hàm được định nghĩa trong: stdio.h.

Ý nghĩa: Dùng làm sạch vùng đệm của tệp fp. Nếu lệnh thành công, hàm sẽ cho giá trị 0, ngược lại nó sẽ trả về giá trị EOF.

11.2.6. Làm sạch vùng đệm của tất cả các tệp đang mở - hàm fflushall:

Cú pháp: int fflushall(void);

Nguyên hàm được định nghĩa trong: stdio.h.

Ý nghĩa: 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, ngược lại nó sẽ trả về giá trị EOF.

11.2.7. Kiểm tra lỗi file - hàm ferror:

Cú pháp: int ferror(FILE *fp);
Nguyên hàm được định nghĩa trong: stdio.h.
Trong đó fp là con trỏ tệp.

Ý nghĩa: Hàm dùng để kiểm tra lỗi khi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không có lỗi, ngược lại hàm cho giá trị khác 0.

11.2.8. Kiểmtra cuối tệp - hàm feof:

Cú pháp : int feof(FILE *fp);
Nguyên hàm được định nghĩa trong: stdio.h.
Trong đó fp là con trỏ tệp.

Ý nghĩa: 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.

11.2.9. Truy nhập ngẫu nhiên - các hàm di chuyển con trỏ định vị:

11.2.9.1. Chuyển con trỏ định vị về đầu tệp - Hàm rewind:

Cú pháp: void rewind(FILE *fp);
Nguyên hàm được định nghĩa trong: stdio.h.
Trong đó fp là con trỏ tệp.

Ý nghĩa: Chuyển con trỏ định vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu tệp.

11.2.9.2. Chuyển con trỏ định vị trí về vị trí cần thiết - Hàm fseek:

Cú pháp: int fseek(FILE *fp, long sb, int xp);
Nguyên hàm được định nghĩa 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 định vị.
  - xp=SEEK_END hay 2: Xuất phát từ cuối tệp.

Ý nghĩa:

Chuyển con trỏ định vị của tệp fp về vị trí xác định thông qua các tham số xp và 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.

11.2.9.3. Vị trí hiện tại của con trỏ định vị - Hàm ftell:

Cú pháp: int ftell(FILE *fp);
Nguyên hàm được định nghĩa trong: stdio.h.
Trong đó: fp là con trỏ tệp.

Ý nghĩa: 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.

11.2.10. Ghi dữ liệu lên tệp - hàm fwrite:

Cú pháp: int fwrite(void *ptr, int size, int n, FILE *fp);

Nguyên hàm được định nghĩa 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

Ý nghĩa: Hàm ghi n mẫu tin, mỗi mẫu tin có 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;
}
```

11.2.11. Đọc dữ liệu từ tệp - hàm fread:

Cú pháp: int fread(void *ptr, int size, int n, FILE *fp);

Nguyên hàm được định nghĩa 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

Ý nghĩa: 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;
}
```

11.2.12. Nhập xuất ký tự:

11.2.12.1. Các hàm putc và fputc:

Cú pháp:

```
int putc(int ch, FILE *fp);
int fputc(int ch, FILE *fp);
```

Nguyên hàm được định nghĩa trong: stdio.h.

Trong đó:

ch là một giá trị nguyên

fp là một con trỏ tệp.

Ý nghĩa:

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, ngược lại nó sẽ trả về giá trị EOF.

Ví dụ:

```
#include "stdio.h"
main()
{
  char msg[] = "Hello world\n";
  int i = 0;
  while (msg[i])
    putc(msg[i++], stdout);  /* stdout thiết bị ra chuẩn - Màn hình*/
  return 0;
}
```

11.2.12.2. Các hàm getc và fgettc:

Cú pháp:

int gretc(FILE *fp);

int fputc(FILE *fp);

Nguyên hàm được định nghĩa trong: stdio.h.

Trong đó:

fp là một con trỏ tệp.

Ý nghĩa:

Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ:

```
#include "string.h"
#include "stdio.h"
#include "conio.h"
main()
{
  FILE *stream;
  char string[] = "Kiem tra";
```

```
  char ch;
  /* Mở tệp để cập nhật*/
  stream = fopen("DUMMY.FIL", "w+");
  /*Viết một xâu ký tự vào tệp */
  fwrite(string, strlen(string), 1, stream);
  /* Tìm vị trí đầu của tệp */
  fseek(stream, 0, SEEK_SET);
  do
  {
    /* Đọc một ký tự từ tệp */
    ch = fgetc(stream);
    /* Hiển thị ký tự */
    putch(ch);
  } while (ch != EOF);
  fclose(stream);
  return 0;
}
```

11.2.13. Xoá tệp - hàm unlink:

Cú pháp: int unlink(const char *tên_tệp)

Nguyên hàm được định nghĩa trong: **dos.h, io.h, stdio.h** .

Trong đó: **tên_tệp** là tên của tệp cần xoá.

Ý nghĩa:

Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ:

```
#include <stdio.h>
#include <io.h>
int main(void)
{
  FILE *fp = fopen("junk.jnk","w");
  int status;
  fprintf(fp,"junk");
  status = access("junk.jnk",0);
  if (status == 0)
```

```
    printf("Tệp tồn tại\n");
  else
    printf("Tệp không tồn tại\n");
  fclose(fp);
  unlink("junk.jnk");
  status = access("junk.jnk",0);
  if (status == 0)
    printf("Tệp tồn tại\n");
  else
    printf("Tệp không tồn tại\n");
  return 0;
}
```

Bài 12: ĐỒ HOẠ

Việc hiển thị thông tin trên màn hình máy tính được thực hiện thông qua một vỉ mạch điều khiển màn hình. Khi màn hình ở chế độ văn bản (text mode) chúng ta có thể hiển thị thông tin lên màn hình bằng các lệnh: printf(), putch(), putchar(), … Thông tin mà chúng ta cần đưa ra màn hình được chuyển tới vỉ mạch điều khiển màn hình dưới dạng mã kí tự ASCII. Vỉ mạch nói trên có nhiệm vụ đưa kí tự đó theo mẫu định sẵn ra màn hình ở vị trí được xác định bởi chương trình của chúng ta.

Ngoài chế độ văn bản, màn hình còn có thể làm việc trong chế độ đồ hoạ. Khi màn hình ở chế độ đồ họa chúng ta có thể vẽ đồ thị, viết chữ to hoặc thể hiện các hình ảnh khác - những việc mà chúng ta không thể thực hiện được trong chế độ văn bản.

Các hàm và thủ tục đồ hoạ được khai báo trong file graphics.h.

12.1. Khởi động đồ hoạ:

Cú pháp: 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 có thể của graphdriver và graphmode:

graphdriver	**graphmode**	**Độ phân giải**
CGA (1)	CGAC0 (0)	320x200
	CGAC1 (1)	320x200
	CGAC2 (2)	320x200
	CGAC3 (3)	320x200
	CGAHI (4)	640x200
MCGA (2)	MCGA0 (0)	320x200
	MCGA1 (1)	320x200
	MCGA2 (2)	320x200
	MCGA3 (3)	320x200
	MCGAMed (4)	640x200
	MCGAHI (5)	640x480
EGA (3)	EGAL0 (0)	640x200
	EGAHI (1)	640x350

EGA64 (4)	EGA64LO (0)	640x200
	EGA64HI (1)	640x350
EGAMONO (5)	EGAMONOHi (0)	640x350
VGA (9)	VGALO (0)	640x200
	VGAMED (1)	640x350
	VGAHI (2)	640x480
HERCMONO (7)	HERCMONOHI	720x348
ATT400 (8)	ATT400C0 (0)	320x200
	ATT400C1 (1)	320x200
	ATT400C2 (2)	320x200
	ATT400C3 (3)	320x200
	ATT400MED (4)	640x400
	ATT400HI (5)	640x400
PC3270 (10)	PC3270HI (0)	720x350
IBM8514 (6)	PC3270LO (0)	640x480 256 mầu
	PC3270HI (1)	1024x768 256 mầu

Bảng trên cho thấy độ phân giải của màn hình phụ thuộc cả vào kiểu 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. Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình.).

Ví dụ: Giả sử máy tính có màn hình VGA, các tập tin đồ hoạ chứa trong thư mục C:\TC \BGI, khi đó ta khởi động chế độ đồ hoạ cho màn hình 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 */
        }
```

Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver bằng DETECT hay giá trị 0. Khi đó, kết quả của initgraph sẽ là:

Kiểu màn hình đang sử dụng được phát hiện, giá trị của nó được gán cho biến graphdriver.

Mode đồ hoạ ở độ phân giải cao nhất ứng với màn hành đang sử dụng cũng được phát hiện và giá trị của nó được gán cho biến graphmode.

Như vậy dùng hằng số DETECT chẳng những có thể khởi tạo được chế độ đồ hoạ cho 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 hiện thời và thư của của trình biên dịch.

12.2. Các hàm đồ hoạ:

12.2.1. 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);

Các giá trị có thể của **màu** cho bởi bảng dưới đây:

Bảng các giá trị có thể 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 có thể của **mẫu** cho bởi bảng dưới đây:

Bảng các giá trị có thể của mẫu

Tên hằng	**Giá trị số**	**Kiểu mẫu tô**
EMPTY_FILL	0	Tô bằng mầu nền
SOLID_FILL	1	Tô bằng đường liền nét
LINE_FILL	2	Tô bằng đường --------
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

Chọn bảng màu:

Để thay đổi bảng màu đã được định nghĩa trong bảng trên, ta sử dụng hàm:

void setpalete(int số_thứ_tự_màu, int màu);

Ví dụ: Câu lệnh: setpalete(0,lightcyan); đổi màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng.

- **Lấy bảng màu hiện thời:**

+ Hàm getcolor() trả về mầu đã xác định bằng thủ tục setcolor ngay trước nó.

+ Hàm getbkcolor() trả về mầu đã xác định bằng hàm setbkcolor ngay trước nó.

12.2.2. 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.

12.2.2.1. 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();
    }
```

12.2.3. Vẽ đường gấp khúc và đa giác:

- **Vẽ đường gấp khúc:** Muốn vẽ đường gấp khúc đi qua n điểm: (x1,y1), (x2,y2), ..., (xn,yn) thì trước hết ta phải gán các toạ độ (xi,yi) cho một mảng a kiểu int nào đó theo nguyên tắc sau:

Toạ độ x1 gán cho a[0]

Toạ độ y1 gán cho a[1]

Toạ độ x2 gán cho a[2]

Toạ độ y2 gán cho a[3]

....

Toạ độ xn gán cho a[2n-2]

Toạ độ yn gán cho a[2n-1]

Sau đó gọi hàm:

drawpoly(n,a);

Nếu điểm cuối cùng (xn,yn) trùng với điểm đầu (x1,y1) thì ta nhận được một đường gấp khúc khép kín.

- **Tô màu đa giác:** Giả sử ta có a là mảng đã đề cập đến trong mục trên, khi đó ta gọi hàm:

fillpoly(n,a);

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm (x1,y1), (x2,y2), ..., (xn,yn)

Ví dụ: Vẽ một đường gấp khúc và hai đường tam giác.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
    {
        int md=0,mode;
        initgraph(&md,&mode,"C:\\TC\\BGI");
        setbkcolor(CYAN);
        setcolor(YELLOW);
        setfillstyle(SOLID_FILL,MAGENTA);
        drawpoly(3,poly1);
        fillpoly(3,poly2);
        fillpoly(4,poly3);
        getch();
        closegraph();
    }
```

- **Vẽ đường thẳng:**

Để vẽ đường thẳng nối hai điểm bất kỳ có toạ độ (x1,y1) và (x2,y2) ta sử dụng hàm sau: void line(int x1, int y1, int x2, int y2);

Con chạy đồ hoạ giữ nguyên vị trí.

Để vẽ đường thẳng nối từ điểm hiện thời của con chạy đồ hoạ đến một điểm bất có toạ độ (x,y) ta sử dụng hàm sau: void lineto(int x, int y);

Con chạy sẽ chuyển đến vị trí (x,y).

Để vẽ một đường thẳng từ ví trí hiện thời của con chạy(giả sử là điểm x,y) đến điểm có toạ độ (x+dx,y+dy) ta sử dụng hàm sau: void linerel(int dx, int dy);

Con chạy sẽ chuyển đến vị trí (x+dx,y+dy).

- **Di chuyển con chạy đồ hoạ:** Để di chuyển con chạy đến vị trí (x,y), ta sử dụng hàm sau:

  void moveto(int x, int y);

- **Chọn kiểu đường:**

Hàm void setlinestyle(int kiểu_đường, int mẫu, int độ_dày);

tác động đến nét vẽ của các thủ tục vẽ đường line, lineto, linerel , circle, rectangle (hàm vẽ hình chữ nhật).

Hàm này 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 có thể 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 có thể của **độ_dày:**

Tên hằng	**Giá trị số**	**Bề dày**
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Mẫu tự tạo: Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số **mẫu.** Ví dụ ta xét đoạn chương trình:

```
int pattern = 0x1010;
setlinestile(USERBIT_LINE,pattern,NORM_WIDTH);
```

```
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010, trong hệ 2 là:

```
0001 0000 0001 0000
```

Bit 1 sẽ cho điểm sáng, bit 0 sẽ làm tắt điểm ảnh.

Ví dụ: Chương trình vẽ một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau:

(20,20),(620,20),(620,180),(20,180) và (320,100)

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
    {
        int mh=0, mode;
        initgraph(&mh,&mode,"C:\\TC\\BGI");
        setbkcolor(BLUE);
        setcolor(YELLOW);
        setlinestyle(SOLID-LINE,0,THICK_WIDTH);
        moveto(320,100);  /* con chạy ở vị trí ( 320,100 ) */
        line(20,20,620,20); /* con chạy vẫn ở vị trí ( 320,100 ) */
        linerel(-300,80);
        lineto(620,180);
        lineto(620,20);
        getch();
        closegraph();
    }
```

12.2.4. 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).

- **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);
```

Trong đó:

(x,y) là toạ độ của một điểm nào đó gọi là điểm gieo.

Tham số border chứa mã của màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x ,y, border và trạng thái màn hình.

- Khi trên màn hình có một đường cong khép kín hoặc đường gấp khúc khép kín mà mã màu của nó bằng giá trị của border thì:

+ Nếu điểm gieo (x,y) nằm trong miền này thì miền giới hạn phía trong đường sẽ được tô màu.

+ Nếu điểm gieo (x,y) nằm ngoài miền này thì miền phía ngoài đường sẽ được tô màu.

- Trong trường hợp khi trên màn hình không có đường cong nào như trên thì cả màn hình sẽ được tô màu.

Ví dụ: Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Toạ độ (x,y) của điểm gieo được nạp từ bàn phím. Tuỳ thuộc giá trị cụ thể của x,y chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include "graphics.h"
#include "stdio.h"
main()
        {
                int mh=mode=0, x, y;
                printf("\nVao toa do x,y:");
                scanf("%d%d",&x,&y);
                initgraph(&mh,&mode,"");
                if (graphresult != grOk) exit(1);
                setbkcolor(BLUE);
                setcolor(RED);
                setfillstyle(11,YELLOW);
                circle(320,100,50);
                moveto(1,150);
                floodfill(x,y,RED);
                closegraph();
        }
```

12.2.5. Hình chữ nhật:

- Hàm:

void rectangle(int x1, int y1, int x2, int y2);

sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm:

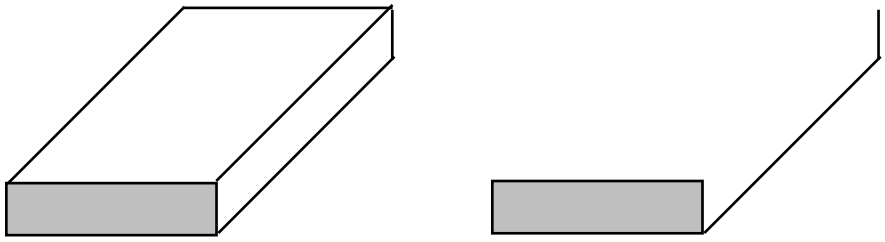
void bar(int x1, int y1, int x2, int y2);

sẽ vẽ và tô màu một hình chữ nhật. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm:

void bar3d(int x1, int y1, int x2, int y2, int depth, int top);

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các toạ độ (x1,y1), (x2,y2). Hình chữ nhật này được tô màu thông qua hàm setfillstyle . Tham số **depth** xác định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số **top** có thể nhận các giá trị 1 hay 0 và khối 3 chiều tương ứng sẽ có nắp hoặc không.



top=1 top=0

Ví dụ: Chương trình dưới đây tạo nên một hình chữ nhật, một khối hình chữ nhật và một hình hộp có nắp:

```
#include "graphics.h"
main()
        {
                int mh=mode=0;
                initgraph(&mh,&mode,"");
                if (graphresult != grOk) exit(1);
                setbkcolor(GREEN);
                setcolor(RED);
                setfillstyle(CLOSE_DOT_FILL,YELLOW);
                rectangle(5,5,300,160);
                bar(3,175,300,340);
                bar3d(320,100,500,340,100,1);
                closegraph();
```

```
}
```

12.2.6. Cửa sổ (Viewport):

- **Thiết lập viewport:**

Viewport là một vùng chữ nhật trên màn hình đồ hoạ. Để thiết lập viewport ta dùng hàm:

```
void setviewport(int x1, int y1, int x2, int y2, int clip);
```

trong đó (x1,y1) là toạ độ góc trên bên trái, (x2,y2) là toạ độ góc dưới bên phải. Bốn giá trị này vì thế phải thoả mãn:

$$0 \le x1 \le x2$$

$$0 \le y1 \le y2$$

Tham số clip có thể nhận một trong hai giá trị:

clip=1 không cho phép vẽ ra ngoài viewport.

clip=0 cho phép vẽ ra ngoài viewport.

Ví dụ:

```
setviewport(100,50,200,150,1);
```

Lập nên một vùng viewport hình chữ nhật có toạ độ góc trái cao là (100,50) và toạ độ góc phải thấp là (200,150) (là toạ độ trước khi đặt viewport).

Chú ý: Sau khi lập viewport, ta có hệ toạ độ mới mà góc trên bên trái sẽ có toạ độ (0,0).

- **Nhận diện viewport hiện hành:**

Để nhận viewport hiện thời ta dùng hàm:

```
void getviewsetting(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau:

```
struct viewporttype
    {
        int left,top,right,bottom;
        int clip;
    };
```

- **Xóa viewport:**

Sử dụng hàm:

```
void clearviewport(void);
```

- **Xoá màn hình, đưa con chạy về tạo độ (0,0) của màn hình:**

Sử dụng hàm:

```
void cleardevice(void);
```

- **Toạ độ âm dương:**

Nhờ sử dụng viewport có thể viết các chương trình đồ hoạ theo toạ độ âm dương. Muốn vậy ta thiết lập viewport và cho clip bằng 0 để có thể vẽ ra ngoài giới hạn của viewport.

Sau đây là đoạn chương trình thực hiện công việc trên:

```
int xc,yc;
xc=getmaxx()/2;
yc=getmaxy()/2;
setviewport(xc,yc,getmaxx(),getmaxy(),0);
```

Như thế, màn hình sẽ được chia làm bốn phần với toạ độ âm dương như sau:

Phần tư trái trên: x âm, y âm.

x: từ -getmaxx()/2 đến 0.

y: từ -getmaxy()/2 đến 0.

Phần tư trái dưới: x âm, y dương.

x: từ -getmaxx()/2 đến 0.

y: từ 0 đến getmaxy()/2.

Phần tư phải trên: x dương, y âm.

x: từ 0 đến getmaxx()/2.

y: từ -getmaxy()/2 đến 0.

Phần tư phải dưới: x dương, y dương.

x: từ 0 đến getmaxx()/2.

y: từ 0 đến getmaxy()/2.

Ví dụ: Chương trình vẽ đồ thị hàm sin x trong hệ trục toạ độ âm dương. Hoành độ x lấy các giá trị từ -4π đến 4π. Trong chương trình có sử dụng hai hàm mới là settextjustify và outtextxy ta sẽ đề cập ngay trong phần sau.

```
#include "graphics.h"
#include "conio.h"
#include "math.h"
#define TYLEX 20
#define TYLEY 60
main()
    {
        int mh=mode=DETECT;
        int x,y,i;
        initgraph(mh,mode,"");
        if (graphresult!=grOK ) exit(1);
```

```
        setviewport(getmaxx()/2,getmaxy()/2,getmaxx(),getmaxy(),0);
        setbkcolor(BLUE);
        setcolor(YELLOW);
        line(-getmaxx()/2,0,getmaxx()/2,0);
        line(0,-getmaxy()/2,0,getmaxy()/2,0);
        settextjustify(1,1);
        setcolor(WHITE);
        outtextxy(0,0,"(0,0)");
        for (i=-400;i<=400;++i)
                {
                        x=floor(2*M_PI*i*TYLEX/200);
                        y=floor(sin(2*M_PI*i/200)*TYLEY);
                        putpixel(x,y,WHITE);
                }
        getch();
        closegraph();
}
```

12.3. Xử lý văn bản trên màn hình đồ hoạ:

- **Hiển thị văn bản trên màn hình đồ hoạ:**

Hàm:

```
void outtext(char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí con trỏ đồ hoạ hiện thời.

Hàm:

```
void outtextxy(int x, int y,char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí (x,y).

Ví dụ:

Hai cách viết dưới đây:

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
outtext(" Say HELLO");
```

cho cùng kết quả.

- **Sử dụng các Fonts chữ:**

Các Fonts chữ nằm trong các tập tin *.CHR trên đĩa. Các Fonts này cho các kích thước và kiểu chữ khác nhau, chúng sẽ được hiển thị lên màn hình bằng các hàm outtext và outtextxy. Để chọn và nạp Fonts ta dùng hàm:

void settextstyle(int font, int direction, int charsize);

Tham số font để chọn kiểu chữ và nhận một trong các hằng sau:

DEFAULT_FONT=0

TRIPLEX_FONT=1

SMALL_FONT=2

SANS_SERIF_FONT=3

GOTHIC_FONT=4

Tham số derection để chọn hướng chữ và nhận một trong các hằng sau:

HORIZ_DIR=0 văn bản hiển thị theo hướng nằm ngang từ trái qua phải.

VERT_DIR=1 văn bản hiển thị theo hướng thẳng đứng từ dưới lên trên.

Tham số charsize là hệ số phóng to của ký tự và có giá trị trong khoảng từ 1 đến 10.

Khi charsize=1, font hiển thị trong hình chữ nhật 8*8 pixel.

Khi charsize=2 font hiển thị trong hình chữ nhật 16*16 pixel.

.....

Khi charsize=10, font hiển thị trong hình chữ nhật 80*80 pixel.

Các giá trị do settextstyle lập ra sẽ giữ nguyên tới khi gọi một settextstyle mới.

Ví dụ:

Các dòng lệnh:

```
settextstyle(3,VERT_DIR,2);
outtextxy(30,30,"GODS TRUST YOU");
```

sẽ hiển thị tại vị trí (30,30) dòng chữ GODS TRUST YOU theo chiều từ dưới lên trên, font chữ chọn là SANS_SERIF_FONT và cỡ chữ là 2.

- **Đạt vị trí hiển thị của các xâu ký tự cho bởi outtext và outtextxy:**

Hàm settextjustify cho phép chỉ định ra nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy và của outtextxy theo quan hệ với toạ độ (x,y);

Hàm này có dạng sau:

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau:

```
LEFT_TEXT=0      ( Văn bản xuất hiện bên phải con chạy).
CENTER_TEXT      ( Chỉnh tâm văn bản theo vị trí con chạy).
```

RIGHT_TEXT (Văn bản xuất hiện bên trái con chạy).

Tham số vert có thể là một trong các hằng số sau:

BOTTOM_TEXT=0 (Văn bản xuất hiện phía trên con chạy).

CENTER_TEXT=1 (Chỉnh tâm văn bản theo vị trí con chạy).

TOP_TEXT=2 (Văn bản xuất hiện phía dưới con chạy).

Ví dụ:

```
settextjustify(1,1);
outtextxy(100,100,"ABC");
```

sẽ cho dòng chữ ABC trong đó điểm (100,100) sẽ nằm dưới chữ B.

- **Bề rộng và chiều cao của kí tự:**

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 của kí tự:

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.