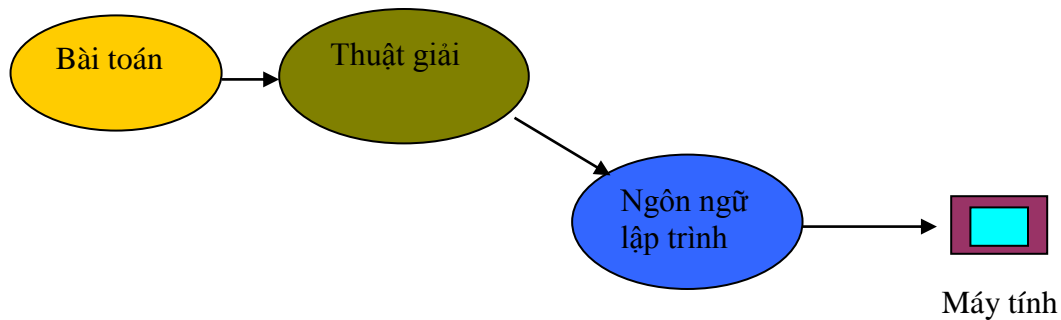


BÀI 1

KHÁI NIỆM CƠ BẢN

1. Dẫn nhập



Ngôn ngữ lập trình C là phương tiện để mô tả thuật giải. Nó có một số đặc điểm sau:

- * Có bộ lệnh phù hợp với phương pháp lập trình cấu trúc.
- * Kiểu dữ liệu phong phú.
- * Một chương trình C bao giờ cũng gồm một hoặc nhiều hàm. Các hàm rời nhau (không lồng vào nhau).
- * Là một ngôn ngữ rất linh động về cú pháp, có thể chấp nhận nhiều cách thể hiện chương trình, do đó đòi hỏi người lập trình phải nắm vững ngôn ngữ để có thể dự đoán được kết quả và hiệu quả của các lệnh hoặc cách thể hiện mà mình sử dụng.

2. CHƯƠNG TRÌNH C

□ Ví dụ về chương trình C

```
/* Chương trình chào */      Lời giải thích  
#include <stdio.h>          Thư viện  
void main()                  Kiểu hàm  
{  
  
    printf("\nTôi là Turbo C ");  
    printf("\nChào các bạn ");  
}
```

Hàm chính Chỉ thị Dấu ; là dấu ngăn cách của C

3. CÁC KIỂU DỮ LIỆU CƠ SỞ

Các kiểu dữ liệu cơ sở trong C gồm:

- ☐ [Kiểu ký tự \(char\)](#)
- ☐ [Kiểu số nguyên \(int\)](#)
- ☐ [Kiểu dấu phẩy động \(chính xác đơn \(float\), chính xác kép \(double\)\)](#)
- ☐ [Kiểu void](#)

3.1. Kiểu ký tự (char)

Một giá trị kiểu ký tự (char) chiếm 1 byte trong bộ nhớ và biểu diễn một ký tự thông qua bảng mã ASCII.

Ví dụ

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
a	097

Có hai kiểu ký tự (char) sau:

	Phạm vi biểu diễn	Số ký tự	Kích thước
Thứ nhất là: char (signed char)	-128 -> 127	256	1 byte
Thứ hai là: unsigned char	0 -> 255	256	1 byte

Ví dụ

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

Khi đó thực chất:

Ch1=-56

Ch1 + 56=0

Ch2 + 56=256

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

3.2. Kiểu số nguyên (int)

Trong C cho phép sử dụng các kiểu số nguyên sau:

Kiểu	Phạm vi biểu diễn	Kích thước
Int	-32768 → 32767	2 byte
Unsigned int	0 → 65535	2 byte
Long (int)	-2147483648 → -2147483647	4 byte
Unsigned long (int)	0 → 4294967295	4 byte

Nhận xét: Các kiểu ký tự cũng có thể xem là một dạng của kiểu số nguyên

3.3. Kiểu dấu phẩy động

Trong C sử dụng ba loại giá trị dấu phẩy động: float (độ chính xác đơn), double và long double (độ chính xác kép).

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
Float	3.4E-38 → 3.4E+38	7-8	4 byte
Double	1.7E-308 → 1.7E+308	15-16	8 byte
Long double	3.4E-4932 → 1.1E+4932	17-184	10 byte

3.4. Kiểu void

Kiểu không giá trị, dùng để biểu diễn kết quả của hàm hay của con trỏ.

Ví dụ

Xét hai hàm sau:

<pre> ... int gtri1() { ... return 1; } </pre>	<pre> ... void gtri2() { ... return; } </pre>
Hàm <i>gtri1()</i> có kiểu int nên trả về giá trị 1 kiểu int.	Hàm <i>gtri2()</i> có kiểu void nên không trả về giá trị.

4. HẰNG, BIẾN

4.1. HẰNG

□ Khái niệm

Là một giá trị bất biến trong chương trình không thay đổi, không biến đổi. Các loại hằng được sử dụng trong C tương ứng với các kiểu dữ liệu nhất định.

□ Trong C, thường có các loại hằng sau:

Hằng số

Hằng ký tự

Hằng chuỗi

□ **Hằng số**

Đó là các giá trị số đã xác định, một hằng số có thể là nguyên hay thực và được viết trong chương trình một cách bình thường.

Hằng nguyên: Giá trị chỉ bao gồm các chữ số, dấu +, - được lưu trữ theo kiểu int. Ví dụ: 12, -12

Nếu giá trị vượt quá miền giá trị của int hoặc có ký tự l (hay L) theo sau giá trị thì lưu theo kiểu long int. Ví dụ: 43L hoặc 43l là hằng nguyên lưu theo kiểu long int.

Hằng thực: Trong giá trị có dấu chấm thập phân, hoặc ghi dưới dạng số có mũ, và được lưu theo kiểu float, double, long double. Ví dụ: 1.2, 2.1E-3 (2.1E-3=0.0021) hoặc 3.1e-2 (3.1e-2=0.031).

□ **Hằng ký tự**

Một hằng kiểu ký tự được viết trong dấu ngoặc đơn (') như 'A' hoặc 'z'. Hằng ký tự 'A' thực sự đồng nghĩa với giá trị nguyên 65, là giá trị trong bảng mã ASCII của chữ hoa 'A' (Như vậy giá trị của hằng chính là mã ASCII của nó). Đối với một vài hằng ký tự đặc biệt, ta cần sử dụng cách viết thêm dấu \, như '\t' tương ứng với phím tab:

Cách viết	Ký tự
<code>"\n"</code>	Xuống hàng;
<code>"\t"</code>	Tab
<code>"\0"</code>	"null"—ứng với giá trị nguyên 0 trong bảng mã ASCII (khác với số '0')
<code>"\b"</code>	Backspace

"\r"	CR(về đầu dòng)
"\f"	LF(sang trang)
"\\"	\
"\""	\"
"\""	'

Hằng ký tự có thể tham gia vào phép toán như mọi số nguyên khác:
'8' - '1' = 56-49=7.

□ **Hằng chuỗi**

Là chuỗi ký tự nằm trong cặp dấu nháy kép "". Các ký tự này cũng có thể là các ký tự được biểu diễn bằng chuỗi thoát.

Ví dụ: "Turbo C", "Ngôn ngữ C++ \n\r"

Một hằng chuỗi được lưu trữ tận cùng bằng một ký tự Nul (\0), ví dụ chuỗi "Turbo C" được lưu trữ trong bộ nhớ như sau:

T	U	R	B	O		C	\0
---	---	---	---	---	--	---	----

-

□ **Cách định nghĩa hằng sử dụng trong chương trình**

Với các giá trị hằng thường được dùng trong một chương trình ta nên định nghĩa ở đầu chương trình (sau các dòng khai báo những thư viện chuẩn) theo cú pháp:

#define <tên hằng> <giá trị>

Ví dụ: #define PI 3.1415

4.2. BIẾN

□ **Cách khai báo**

Mỗi biến trong chương trình đều phải được khai báo trước khi sử dụng. Cú pháp khai báo biến như sau:

<Kiểu dữ liệu> <danh_sách_tên_biến>;

Lưu ý: nếu có nhiều tên biến thì giữa các tên biến phải có dấu ,

Ví dụ: int a,b;
float x;

□ **Khởi đầu cho các biến**

Ngay trên dòng khai báo ta có thể gán cho biến một giá trị. Việc làm này gọi là khởi đầu cho biến.

Ví dụ: int a,b=6,d=1;

□ **Truy xuất đến địa chỉ của biến**

Một số hàm của C dùng đến địa chỉ của biến ví dụ như hàm *scanf*. Để nhận địa chỉ của biến dùng toán tử: &

Ví dụ: &tên_biến.

5. CÁC BÀI TẬP MINH

HỌA http://www.uit.edu.vn/data/gtrinh/TH012/Htm/th012_u015.doc

5.1. Các chương trình minh họa việc sử dụng ký tự điều khiển (\n)

Chương trình 1

```
/* Chương trình in ra dòng chữ Hello, word trên màn hình */
#include <stdio.h>
void main () /* Hàm chính */
{
    printf(" Hello, world \n ");
    /* in chu Hello, world roi xuong dong (\n) */
}
```

Chương trình 2

```
/* Chương trình in ra hai dòng:
    Hello,
    Word */
#include <stdio.h>
void main ()
{
    printf(" \n Hello, \n world \n ");
    /* Xuong dong */
    /* In chu "Hello ," roi xuong dong */
    /* In tiep chu "world" roi xuong dong */
}
```

5.2. Các chương trình minh họa cách khai báo, khởi đầu biến

Chương trình 3

```
/*Chương trình này minh họa cách khai báo biến trong C*/
#include <stdio.h>
void main()
{
    char ki_tu;      /* Khai báo một kí tự*/
    int so_nguyen;   /* Khai báo một số nguyên*/
    float so_thuc;   /* Khai báo một số thực*/
    ki_tu = 'a';
    so_nguyen = 15;
    so_thuc = 27.62;

    printf("%c la mot ki tu.\n",ki_tu);
    printf("%d la mot so nguyen.\n",so_nguyen);
    printf("%f la mot so thuc.\n",so_thuc);
}
```

Kết quả

a là một kí tự.
15 là một số nguyên
27.620001 là một số thực

Chương trình 4

```
/*Chương trình này minh họa cách vừa khai báo, vừa khởi đầu một biến trong C */
#include <stdio.h>
void main()
{
    char ki_tu = 'a';      /* Khai báo/khởi đầu một kí tự. */
    int so_nguyen = 15;    /* Khai báo/khởi đầu một số nguyên. */
    float so_thuc = 27.62; /* Khai báo/khởi đầu một số thực. */

    printf("%c là một kí tự.\n", ki_tu);
    printf("%d là một số nguyên.\n", so_nguyen);
    printf("%f là một số thực.\n", so_thuc);
}
```

Kết quả
Giống bài trên

5.3. Giải các bài toán đơn giản

Chương trình 5

```
/* Chương trình tính chu vi và diện tích hình tròn, biết bán kính r là một hằng số có
giá trị = 3.1 */
# include <stdio.h> /* sử dụng thư viện chứa các hàm nhập xuất chuẩn */
# include <math.h> /* sử dụng thư viện chứa các hàm toán học */
#define r 3.1
void main ()
{
    float cv, dt;      /* khai báo 2 biến chu vi và diện tích kiểu số thực */
    cv = 2 * r * M_PI; /* tính chu vi, trong đó sử dụng hằng M_PI có trong math.h */
    dt = M_PI * r * r; /* Tính diện tích */
    printf("\nChu vi    = %10.2f\nDiện tích = %10.2f", cv, dt);
    /* In kết quả lên màn hình */
    getch(); /* Tam dụng chương trình */
}
```

Kết quả
Chu vi = 19.47
Diện tích= 30.18

Chú ý:

%d : là mã đặc tả biểu diễn các biến có kiểu số nguyên (int)
%f : là mã đặc tả biểu diễn các biến có kiểu số thực (float, double...)
%m.nf : in ra một biến kiểu số thực có chiều dài bằng m và có n chữ số sau dấu
chấm thập phân.

Ví dụ: %6.2f biểu diễn được số:

$$\frac{m=6}{132.22} \quad \frac{n=2}{n=2}$$

{Phần này sẽ được chi tiết trong bài 2}

Chương trình 6

```
/*Chương trình biết chu vi là hằng cv có giá trị =9.1, tính diện tích hình tròn */
#include <stdio.h> /* su dung thu vien chua cac ham nhap xuat chuan */
#include <math.h> /* su dung thu vien chua cac ham toan hoc */
#define cv 9.1
void main ()
{
    float r,dt; /* khai bao 2 bien ban kinh va dien tich kieu so thuc */
    r=cv/(2*M_PI); /* tinh ban kinh, trong do su dung hang M_PI co trong math.h */
    dt=M_PI*r*r; /* Tinh dien tich */
    printf("\nDien tich = %10.2f",dt);
    /* In ket qua len man hinh */
    getch(); /* Tam dung chuong trinh */
}
```

Kết quả
Dien tich= 6.59

Chương trình 7

```
/*Chương trình biết diện tích là biến dt có giá trị khởi đầu =6.1, tính chu vi hình tròn */
#include <stdio.h> /* su dung thu vien chua cac ham nhap xuat chuan */
#include <math.h> /* su dung thu vien chua cac ham toan hoc */
void main ()
{
    float r,dt=6.1,cv; /* khai bao bien ban kinh kieu so thuc va khoi dau bien dien tich dt */
    r= sqrt(dt/M_PI); /* tinh ban kinh, trong do su dung ham ca bac hai sqrt() co trong math.h */
    cv=2*M_PI*r; /* Tinh chu vi */
    printf("\nChu vi = %10.2f",cv);
    /* In ket qua len man hinh */
    getch(); /* Tam dung chuong trinh */
}
```

Kết quả
Chu vi= 8.75

5.4. Một số qui tắc cần nhớ khi viết chương trình C

- Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu chấm phẩy (;) .
- Muốn biểu diễn một dãy hằng ký tự (dãy ký tự đặt trong cặp dấu nháy kép) trên nhiều dòng, thì ta phải đặt thêm dấu \ trước khi xuống dòng.

Ví dụ: `printf("\nChu vi = %10.2f \nDien tich = %10.2f",cv,dt);`

- Các lời giải thích phải đặt giữa dấu `/*` và dấu `*/`
- Muốn sử dụng các hàm chuẩn của C thì ở đầu chương trình ta phải khai báo tập tin có chứa hàm muốn sử dụng (ở cuối dòng khai báo này không có dấu `;`).

Ví dụ: `# include <stdio.h>
include <math.h>`

BÀI 2

BIỂU THỨC VÀ CÁC PHÉP TOÁN TRONG C

1. BIỂU THỨC TRONG C

1.1. Khái niệm

Là sự kết hợp hợp lệ của những phép toán thực hiện trên các biến, hằng hoặc các giá trị của hàm. Một biểu thức bao gồm các *toán tử* (phép toán) và các *toán hạng* (biến, hằng...).

1.2. Các ví dụ

Ví dụ 1:

`int x=2,y=7;`

`x=(x+2*y);` /* *x,y* là các biến tương ứng với các toán hạng ; phép cộng (+) và (*) và

dấu (=) là các toán tử của biểu thức*/

Ví dụ 2:

`int i, a=3;`

`a=(i=a*11);` /*là một biểu thức hợp lệ. (với *a* là một biến đã có giá trị trước đó).*/

-

2. CÁC PHÉP TOÁN

2.1. Các phép toán số học

Bao gồm: **Cộng** (+), **trừ** (-), **nhân** (*), **chia** (/) thực hiện trên các kiểu dữ liệu `int`, `char`, `float`, `double`.

Ví dụ:

```

float so_1 = 15.0;          /* Toán hạng thứ nhất.
*/
float so_2 = 3.0;          /* Toán hạng thứ hai.
*/
float ketqua_phepcong = so_1 + so_2; /* Phép cộng :
15.0+3.0=18.000000*/
float ketqua_pheptru = so_1 - so_2;   /* Phép trừ : 15.0-3.0
=12.000000*/
float ketqua_phepnhan = so_1 * so_2; /* Phép nhân :
15.0*3.0=45.000000*/
float ketqua_phepchia = so_1/so_2;    /* Phép chia : 15.0/3.0= 5.000000
*/

```

Chú ý: Phép chia của hai số nguyên cho ra kết quả là số nguyên. Như vậy, để lấy phần dư của phép chia hai số nguyên phải sử dụng phép **modulo (%)** (phép **modulo(%)** chỉ thực hiện trên các toán hạng có kiểu dữ liệu nguyên (int)).

Ví dụ:

```

int so_1=10;
int so_2=3;
int so_3=so_1/so_2 /* phép chia 2 số nguyên cho kết quả là số nguyên:10/3=3
*/
int so_4=so_1%so_2 /* phép modulo(%) cho phần dư của phép
chia:10%3=1(10/3=3 dư 1)*/

```

2.2. Các phép thao tác bit

Cho phép xử lý từng bit của một số nguyên (không dùng cho kiểu float và double). Bao gồm:

- ☐ [Các toán tử AND\(&\), OR\(|\), XOR\(^\)](#)
- ☐ [Các toán tử dịch trái \(<<\), dịch phải \(>>\)](#)
- ☐ [Toán tử bù bit \(~\)](#)
- ☐ § **Các toán tử AND(&), OR(|), XOR(^)**

Toán hạng 1	Toán hạng 2	Kết quả		
		&		^
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1

0	0	0	0	0
---	---	---	---	---

Ví dụ

Giả thiết một số nguyên (int) được biểu diễn bằng 16 bit

```
unsigned int a=3737; /*0000111010011001*/
unsigned int b=7474; /*0001110100110010*/
unsigned int c= a/ b; /*0001111110111011*/
unsigned int d= b&c; /*0001110100110010*/
unsigned int e= c^d; /*0000001010001001*/
```

□ § Các toán tử dịch trái(<<), dịch phải (>>)

Biểu thức $a \ll n$ sẽ dịch chuyển các bit trong a sang trái n vị trí.

Biểu thức $a \gg n$ sẽ dịch chuyển các bit trong a sang phải n vị trí.

Đối với kiểu không dấu thì:

$$a \ll n = a * (2^n)$$

$$a \gg n = a / (2^n).$$

Ví dụ

Giả thiết một số nguyên (int) được biểu diễn bằng 16 bit

```
unsigned int a=3737; /*0000111010011001*/
unsigned int b=a<<1; /*0001110100110010*/
unsigned int c= a>>2; /*0000001110100110*/
unsigned int d= c<<2<<3; /*0111010011000000*/
unsigned int e= 1<<15; /*1000000000000000*/
int f=3737 /*0000111010011001*/
int g=f<<4 /*1110100110010000*/
int h=g>>4 /*????000011101001*/
```

□ § Bù bit ~

$$\sim 1 = 0$$

$$\sim 0 = 1.$$

Ví dụ

Giả thiết một số nguyên (int) được biểu diễn bằng 16 bit

```
unsigned int a=3737; /*0000111010011001*/
unsigned int b=~a; /*1111000101100110*/
```

2.3. Các phép toán quan hệ và logic

Gồm :

[§ Các phép toán quan hệ](#)

[§ Các phép toán lô gíc](#)

□ § **Các phép toán quan hệ**

So sánh giá trị của các toán hạng rồi cho ra kết quả 0(sai) hoặc 1(đúng).

Gồm:

$==, !=, >, >=, <, <=$

Ví dụ:

```
float ket_qua1 = (3 > 5);          /* ket_qua1 = (3 > 5) = 0.000000
*/
float ket_qua2 = (5 > 3);          /* ket_qua2 = (5 > 3) = 1.000000
*/
float ket_qua3 = (3 >= 5); /* ket_qua3 = (3 >= 5) = 0.000000 */
float ket_qua4 = (15 >= 3*5); /* ket_qua4 = (15 >= 3*5) = 1.000000
*/
float ket_qua5 = (8 < (10-2)); /* ket_qua5 = (8 < (10-2)) = 0.000000
*/
float ket_qua6 = (2*3 < 24/3); /* ket_qua6 = (2*3 < 24/3) = 1.000000
*/
float ket_qua7 = (10 < 5);          /* ket_qua7 = (10 < 5) = 0.000000
*/
float ket_qua8 = (24 <= 15); /* ket_qua8 = (10 < 5) = 0.000000
*/
float ket_qua9 = (36/6 <= 2*3); /* ket_qua9 = (36/6 <= 2*3) = 1.000000
*/
float ket_qua10 = (8 == 8);          /* ket_qua10 = (8==8) = 1.000000
*/
float ket_qua11 = (12+5 == 15); /* ket_qua11 = (12+ 5==15) = 0.000000
*/
float ket_qua12 = (8 != 5);          /* ket_qua12 = (8!=5) = 1.000000
*/
float ket_qua13 = (15 != 3*5); /* ket_qua13 = (15!=3*5) = 0.000000
*/
```

□ § **Các phép toán logic**

Dùng kết hợp các biểu thức khác nhau thành một biểu thức logic.

Gồm:

Toán tử NOT !

Toán tử AND &&

Toán tử OR ||

Kết quả của các phép toán này cũng có giá trị 0 (sai) hoặc 1(đúng)

Bảng giải thích

Toán hạng 1	Toán hạng 2	Kết quả
-------------	-------------	---------

		! A	A&&B	A B
Bằng 0	Bằng 0	1	0	0
Bằng 0	Khác 0	1	0	1
Khác 0	Bằng 0	0	0	1
Khác 0	Khác 0	0	1	1

Ví dụ

x	y	$x\&\&y$	$x//y$	$!x$	$!!x$
0	0	0	0	1	0
0	7	0	1	1	0
5	0	0	1	0	1
5	7	1	1	0	1
8	7	1	1	0	1

chú ý: $!!x$ có thể không bằng x

2.4. Chuyển đổi kiểu dữ liệu

- § Khi hai toán hạng trong một phép toán khác kiểu dữ liệu thì kiểu dữ liệu thấp được nâng thành kiểu dữ liệu cao trước khi tính toán. *Ví dụ:* nếu f có kiểu *float*, i có kiểu *int*, thì trong biểu thức $f+i$, i sẽ tạm thời được chuyển sang kiểu *float* để thực hiện phép cộng.
- § Nếu f có kiểu *float*, $i1$ và $i2$ có kiểu *int* và có giá trị 10 và 3 thì biểu thức $f=i1/i2$ sẽ gán vào f giá trị 3.0. Trong trường hợp này, để thu được kết quả chính xác, cần sử dụng **phép ép kiểu**: $f=(float) i1/i2$

Ví dụ:

int a,b=4;c=5;

float x,y=6.8,z=3.8;

*a=y; /*a =6*/*

*a=-y; /*a =-6*/*

*x=a/b+c; /*x=4.0*/*

*x=a/b+(float) c; /*x=4.0*/*

*x=(float) a/b+ c; /*x=3.5*/*

*a=y-z /*a=2 (nếu kết quả là 2.999...) hoặc 3*/*

2.5. Các phép toán tăng giảm

Đó là các phép toán ++ và -- dùng để tăng hoặc giảm một giá trị đối với các biến nguyên hoặc thực.

Các phép toán này có 2 dạng:

++biến hay biến++

--biến hay biến--

Các dạng này có cùng mục đích nhưng khác nhau ở chỗ khi sử dụng các toán tử này chung với các phép toán khác thì ++biến, --biến sẽ thực hiện phép toán tăng hoặc giảm giá trị của biến trước rồi mới thực hiện những phép toán khác, còn biến++, biến-- thì ngược lại.

Phân tích Nếu $i=3$ thì
 câu lệnh $tong=++i$;
 sẽ gán 4 cho $tong$, còn câu lệnh
 $tong=i++$;
 sẽ gán 3 cho $tong$. Trong cả hai trường hợp i đều trở thành 4.

Ví dụ

```
int a=11,j=5;
double g,f,h;
g=(double)++a/j;          /* g=2.4 */
f=g*a/j--;                /* f=5.76 */
h=f+(double) (a=j) /++j;  /* (double) (a=j) /++j bằng 1.0 và
h=6.76 */
```

2.6. Câu lệnh gán

Trong C lệnh gán có hai dạng:

Dạng 1:

Biến = biểu thức

Ví dụ:

```
int c,a=3,b=4;
c=a+b;
```

Dạng 2:

Biến <toán tử>=biểu thức; dạng này được gọi là lệnh gán phức hợp.

Trong đó toán tử là một trong các phép toán số học +, -, *, /, % hoặc các phép thao tác bit đối với các toán hạng nguyên & | ^ << >>

Ví dụ:

```
int so = 10;          /* Giá trị ban đầu */
so += 5;              /* tương đương so=so+5 cho kết quả so=15 */
so -= 3;              /* tương đương so=so-3 cho kết quả so=12 */
so *= 3;              /* tương đương so=so*3 cho kết quả so=36 */
so /= 5;              /* tương đương so=so/5 cho kết quả so=7 */
so %= 3;              /* tương đương so=so%3 cho kết quả so=1 */
```

Chú ý:

```
§ x+=(y=5);          /* nghĩa là gán giá trị 5 vào biến y rồi cộng 5 vào x */
§ x+=(y+=2); /* nếu x=5 và y=6 thì sau khi thi hành câu lệnh trên ta được y=8 và
x=13 */
§ a=b=c=0; /*biểu thức này hoàn toàn đúng */
§ Toán tử gán được thực hiện từ phải sang trái vì vậy ở hai ví dụ trên có thể bỏ dấu
ngoặc
```

2.7. Toán tử phủ

Đây là một toán tử đặc biệt có ký hiệu là “,”. Một biểu thức phẩy bao gồm một cặp biểu thức cách nhau bằng dấu phẩy và được tính toán từ biểu thức bên trái rồi sau đó đến biểu thức bên phải. Kết quả của toàn bộ biểu thức phẩy sẽ là kết quả của biểu thức bên phải.

Ví dụ:

int m,t,h;

m=(t=2,h=t*t+3); /*sẽ cho t=2, h=7 và m=7.*/

-

2.8. Toán tử điều kiện - biểu thức điều kiện

Có ký hiệu là ?. Toán tử này đòi hỏi phải có 3 thành phần như sau:

<điều kiện> ? <biểu thức 1>:<biểu thức 2>

một biểu thức như trên gọi là biểu thức điều kiện. Kết quả của biểu thức này bằng biểu thức 1 nếu điều kiện có kết quả khác 0, ngược lại kết quả của biểu thức này bằng biểu thức 2.

Ví dụ

§ a=1>=5 ? 1:-1 /* biểu thức a cho kết quả bằng -1*/

§ char c,t;

t='D';

c=(t>='A') &&(t<='Z')? t:'A'; /* biểu thức c cho kết quả bằng ký tự

D*/

2.1. 2.9. Thứ tự ưu tiên của các phép toán

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

3. CÁC BÀI TẬP MINH HỌA

3.1. Bài tập mẫu về phép toán số học: phép chia nguyên (/) và phép MODULO(%)

Bài 1: Cho biết giá trị của $8/-5$ và $8\%-5$

Kết quả:

$/*\ 8/-5=-1;\quad 8\%-5=3\ */$

-

3.2. Bài tập mẫu về phép toán thao tác trên bit

Bài 2: Giả sử ta đang xét các số nguyên 16 bit, $a=0xc0b3$, $b=0x2435$, a và b đều là kiểu *unsigned*. Cho biết kết quả từ các biểu thức sau:

- (a) (a) $\sim a$
- (b) (b) a/b
- (c) (c) a^b
- (d) (d) $a>>2$
- (e) (e) $a<<2$
- (f) (f) $a<<-2$

Kết quả:

$/*\quad a=0xc0b3=1100000010110011$
 $\quad b=0x2435=0010010000110101$
(a) (a) $\sim a=0011111101001100=0x3f4c$
(b) (b) $a/b=1110010010110111=0xe4b7$
(c) (c) $a^b=1110010010000110=0xe486$
(d) (d) $a>>2=0011000000101100=0x302c$
(e) (e) $a<<2=0000001011001100=0x02cc$
(f) $a<<-2=0000000000000000=0x0$

$*/$

3.3. Bài tập mẫu về biểu thức quan hệ và lô gíc

Bài 3: Hãy cho biết giá trị của j sau đoạn chương trình:

$int\ j;$

$char\ c='l';$

$j=(c<='9')\ \&\&(c>='0');$

Kết quả:

$/*j=1*/$

3.4. Bài tập mẫu về phép toán ép kiểu dữ liệu trong C

Bài 4: Hãy cho biết giá trị của $(int) 3.5$, $(int) 3.1$, $(int) 3.9$, $(int) -3.1$, $(int) -3.5$, $(int) -3.9$
Kết quả:
 $/(int) 3.5=3.0; (int) 3.1=3.0; (int) 3.9=3.0; (int) -3.1= -3.0; (int) -3.5= -3.0;$
 $(int) -3.9= -3.0;*/$

3.5. Bài tập mẫu về phép toán tăng (++), giảm (--)

Bài 5: Cho b bằng 5 và c bằng 8. Hãy cho biết giá trị của a, b, c sau khi thi hành riêng biệt từng dòng lệnh sau:

1. $a=b++ + c++;$
2. $a=b++ + ++c;$
3. $a=++b + c++;$
4. $a=++b + ++c;$

Kết quả:
 $/*Dòng lệnh 1 cho kết quả: a=13*/$
 $/*Dòng lệnh 2 cho kết quả: a=14*/$
 $/*Dòng lệnh 3 cho kết quả: a=14*/$
 $/*Dòng lệnh 4 cho kết quả: a=15*/$

Bài 6: Giả sử a bằng 1. Hãy cho biết giá trị của a, b sau dòng lệnh:

$b=a++ + ++a;$
 Rồi kiểm tra tiếp xem $a+=a+=a=?$

Kết quả:
 $/* Sau dòng lệnh b=a++ + ++a; b sẽ bằng 4 và a sẽ bằng 3*/$
 $/* Sau dòng lệnh a+=a+=a; a sẽ bằng 12 */$

3.6. Bài tập mẫu về câu lệnh gán

Bài 7:

(a) (a) Đoạn mã sau sẽ làm gì?

$a^-=b; /*Giải thích: a=a^b*/$
 $b^-=a; /* b=b^a*/$
 $a^-=b; /* a=a^b*/$

(b) (b) Xét câu lệnh :

$a^-=b^-=a^-=b; /*Giải thích: a=a^b ; b=b^a; a=a^b$
Như vậy (b) tương đương với (a)/*

3.7. Bài tập mẫu về toán tử phủ

Bài 8: Hãy cho biết giá trị của b và a sau đoạn chương trình:

$int a, b=2;$
 $b=(a=3, (5*b)+(a*=b));$

Kết quả

```
/*b=16 a=6*/
```

Bài 9: Hãy cho biết giá trị của n và x sau đoạn chương trình:

```
int n,x=2;
```

```
x=x-1;
```

```
n=(n=5,n*=10+x++);
```

Kết quả

```
/*n=55 x=2*/
```

3.8. Bài tập mẫu về biểu thức điều kiện

Bài 10: Hãy cho biết giá trị của b sau đoạn chương trình:

```
int a=1,b=(a)?1:2;
```

```
b+=1;
```

Kết quả

```
/*b=2*/
```

Bài 11: Cho khai báo biến sau

```
int a,b;
```

cho biết kết quả từ các biểu thức sau:

(a) $a=(b==2)?1:2;$

(b) $a=(b=2)?1:3;$

(c) $a=(b=2)?1:2;$

Kết quả

```
/*
```

(a) (a) $a=2;$

(b) (b) $a=1;$

(c) (c) $a=1; */$

4. CÁC BÀI TẬP TỰ LÀM

Bài 1

Cho khai báo biến sau:

```
int pint;
```

```
float a;
```

```
char c;
```

```
double pd;
```

Hãy chọn phát biểu đúng

a. $a. (double) pd=a;$

- b. *b. c=+pint+;*
- c. *c. print=(int) pd;*
- d. *d. a=&pint;*

Bài 2

Cho khai báo biến sau:

int a,p;

double b,c;

Hãy chọn phát biểu đúng

- a. *a. p=(int) b+(c*=2);*
- b. *b. p=a+(1,b-=1);*
- c. *c. p=c;*
- d. *d. a="abc";*

Bài 3

Cho khai báo biến sau:

char a,p;

int b,pint;

Hãy chọn phát biểu sai

- a. *a. pint<=<a;*
- b. *b. p->>b;*
- c. *c. a+=1+b- (double) 1;*
- d. *d. b=(char) a;*

Bài 4

Cho chương trình sau:

#include<stdio.h>

unsigned t=1266;

int x,y;

char c1,c2;

long l;

main()

{

*x=t%10*y;*

c1=t%100-x;

c2=c1+2;

*l=c1-c2*y;*

printf("%c%d",c1,c2) ;

}

Kết quả in ra là:

- a. *a. Chương trình sai cú pháp*
- b. *b. B 68*
- c. *c. Chương trình in ra trị không xác định*
- d. *d. Cả 3 câu đều sai*

Bài 5

Cho chương trình sau:

```
#include<stdio.h>
main()
{
    int a=11,i=5;
    double f;
    f=(double) ++a/i;
    f*=a/i--;
    f+=(double) (a=1) /++i;
    printf("a=%d,i= %d,f=%f",a,i,f);
}
```

Kết quả in ra là:

- a. a. a=12,i=5,f=6.72
- b. b. a=5,i=6,f=6.533333
- c. c. a=4,i=5,f=5.6
- d. d. a=1,i=5,f=5.000000

Bài 6

Cho biết kết quả của các chương trình sau

Chương trình 1:

```
#include <stdio.h>
main()
{
    char a='2';
    unsigned char b='7';
    int c=-23;
    unsigned d=124;
    float re=675.89;
    float rm=0.000887;
    float rt=0.000887;
    printf("\n%c\t%c ",a,b);
    printf("\n%4d\t%3d\t%4d\t%3d ",c,d,c,d);
    printf("\n%6.3f\t%6.3g\t%6.3g\t%6.3G ",re,rm,rt,rm,rt);
}
```

Chương trình 2:

```
#include <stdio.h>
void main()
{
    int n=5,p=9;
    int q1,q2,q3,q4,q5;
    float x1,x2,x3,x4;
    q1=(n<p);
    q2=(n==p);
    q3=p%n+p>n;
    q4= n%(p>n? n:p);
    q5= n%(p<n? p:n);
    x1=p/n;
```

```

x2=(float) p/n;
x3=(p+0.5)/n;
x4=(int) (p+0.5)/n;
printf("\n q1=%d",q1);
printf("\n q2=%d",q2);
printf("\n q3=%d",q3);
printf("\n q4=%d",q4);
printf("\n q5=%d",q5);
printf("\n x1=%10.3f",x1);
printf("\n x2=%10.3f",x2);
printf("\n x3=%10.3f",x3);
printf("\n x4=%10.3f",x4);
}

```

Chương trình 3:

```

#include <stdio.h>
void main()
{
    int n=10,p=5,q=10,r;
    r=(n==(p=q));
    printf("I: n=%d p=%d q=%d r=%d\n",n,p,q,r);
    n=p=q=5;
    n+=p+=q;
    printf("II: n=%d p=%d q=%d r=%d\n",n,p,q,r);
    q=(n<p) ? n++:p++;
    printf("III: n=%d p=%d q=%d r=%d\n",n,p,q,r);
    q=(n>p) ? n++:p++;
    printf("IV: n=%d p=%d q=%d r=%d\n",n,p,q,r);
}

```

BÀI 3

NHẬP XUẤT DỮ LIỆU TRONG C

1. CÁC HÀM NHẬP XUẤT TRONG STDIO.H

Bao gồm:

1.1. Hàm printf

printf("Dòng điều khiển",[các biểu thức]);

Dòng điều khiển gồm 3 loại:

- § Chuỗi ký tự mang tính chất thông báo (hằng chuỗi)
- § Các ký tự điều khiển(\n, \r, \t..)

- § Các mã đặc tả để in các biểu thức tương ứng (mỗi biểu thức khi in phải có một đặc tả). Các đặc tả được dùng trong hàm printf như sau:

Mã đặc tả	Kiểu dữ liệu	Tác dụng
%c	char	in một ký tự có mã ASCII tương ứng
%[n]d	int	in một số nguyên với chiều dài tối thiểu là n
%[n]ld	long int	in một số nguyên (long int)
%[n]u	int	in một số nguyên ở hệ 10 không dấu
%[n]o	int	in một số nguyên ở hệ bát phân tương ứng.
%[n]x	int	in một số nguyên ở hệ 16 tương ứng
%[n.m]f	float	in một số thực với chiều dài n và lấy m số thập phân
%s	mảng ký tự	in ra một chuỗi ký tự.

Ví dụ

```

/* Chương trình này minh họa các kiểu dữ liệu trong C */
#include <stdio.h>
void main()
{
    printf("gia tri 92 dung truong kieu d = %d. \n", 92); /* gia tri 92 dung truong kieu d = 92. */
    printf("gia tri 92 dung truong kieu i = %i. \n", 92); /* gia tri 92 dung truong kieu i = 92. */
    printf("gia tri 92 dung truong kieu u = %u. \n", 92); /* gia tri 92 dung truong kieu u = 92. */
    printf("gia tri 92 dung truong kieu o = %o. \n", 92); /* gia tri 92 dung truong kieu o = 134. */
    printf("gia tri 92 dung truong kieu x = %x. \n", 92); /* gia tri 92 dung truong kieu x = 5c. */
    printf("gia tri 92 dung truong kieu X = %X. \n", 92); /* gia tri 92 dung truong kieu X = 5C. */
    printf("gia tri 92.0 dung truong kieu f = %f. \n", 92.0); /* gia tri 92.0 dung truong kieu f = 92.000000. */
    printf("gia tri 92.0 dung truong kieu e = %e. \n", 92.0); /* gia tri 92.0 dung truong kieu e = 9.200000e+01. */
    printf("gia tri 92.0 dung truong kieu E = %E. \n", 92.0); /* gia tri 92.0 dung truong kieu E = 9.200000E+01. */
    printf("gia tri 92.0 dung truong kieu g = %g. \n", 92.0); /* gia tri 92.0 dung truong kieu g = 92. */
    printf("gia tri 92.0 dung truong kieu G = %G. \n", 92.0); /* gia tri 92.0 dung truong kieu G = 92. */
    printf("gia tri 92 dung truong kieu c = %c. \n", 92); /* gia tri 92 dung truong kieu c = \. */
    printf("ky tu '9' dung truong kieu c = %c. \n", '9'); /* ky tu '9' dung truong kieu c = 9. */
    printf("chuoi 92 dung truong kieu s = %s. \n", "92"); /* chuoi 92 dung truong kieu s = 92. */
}

```

1.1. 1.2. Hàm scanf

scanf("các đặc tả", <danh sách địa chỉ các biến tương ứng với các đặc tả>);

- § Các đặc tả có dạng %<ký tự đặc tả> (Mỗi biến muốn nhập giá trị phải có một đặc tả tương ứng).

Ta có các ký tự đặc tả sau:

Mã đặc tả	Kiểu dữ liệu của biến	Tác dụng
%c	Char	nhập một ký tự
%d	Int	nhập một số nguyên
%ld	long int	nhập một số nguyên (long int)
%o	int	nhập một số nguyên ở hệ bát phân.
%x	int	nhập một số nguyên ở hệ 16 tương ứng

%f %lf %s	float double mảng ký tự	nhập một số thực nhập một số thực nhập ra một chuỗi ký tự. Mảng tương ứng phải đủ lớn để chứa chuỗi nhập vào và ký tự kết thúc chuỗi (\0), lệnh <i>scanf</i> tự động chèn ký hiệu kết thúc chuỗi vào.
-----------------	-------------------------------	---

-

Ví dụ

-

ví dụ 1:

```
/* Chương trình minh họa nhập dữ liệu (nhập một số thực) */
#include <stdio.h>
void main()
{
    float value;                /* Một số được nhập bởi người sử dụng chương
trình. */
    printf("Nhập một số => ");
    scanf("%f", &value);        /* Nhập một số => 23 */
    printf("Kết quả là => %f", value); /* Kết quả là => 23.000000 */
}
```

-

ví dụ 2:

```
/* Chương trình minh họa nhập dữ liệu (nhập một chuỗi (xâu) ký tự) bằng hàm scanf() */
#include <stdio.h>
char a[5]; /* Khai báo một chuỗi gồm 5 ký tự */
void main () /* Hàm chính */
{
    int i;
    printf("Nhập chuỗi 5 ký tự: ");
    scanf("%s", a);
```

/* giả sử nhập vào là ABCDEFGH (chú ý không nhập được khoảng trắng bằng lệnh này)*/

/* Khi nhập chuỗi ta không cần phải lấy địa chỉ */

```
printf("%s", a); /* In ra ABCDEFGH */
}
```

Ghi chú: Để nhập được dòng ký tự có khoảng trắng bằng hàm *scanf* phải viết *scanf("%[^\n]", a)* /* a là chuỗi (xâu) ký tự được nhập */

1.3. Hàm gets

gets(Tên của mảng ký tự);

Hàm này cho phép nhận một chuỗi từ bàn phím cho đến khi gặp ký tự \n (cho phép nhập khoảng trắng giữa các từ)

Ví dụ

```
/* Chương trình minh họa nhập dữ liệu (nhập một chuỗi (xâu) ký tự) bằng hàm gets() */
#include <stdio.h>
char a[5]; /* Khai báo một chuỗi (xâu) gồm 5 ký tự */
void main () /* Hàm chính */
{
    int i;
    printf("Nhap chuoi 5 ky tu: ");
    gets(a);

    /* giả sử nhập vào là ABCDEFGH , có thể nhập được khoảng trắng */

    /* Khi nhập chuỗi ta không cần phải lấy địa chỉ */

    printf("%s",a); /* In ra ABCDEFGH */
}
```

1.4. Hàm getchar

getchar(void)

Nhận một ký tự từ bàn phím và trả về ký tự nhận được

Ví dụ

```
/* Chương trình minh họa hàm getchar() */
#include <stdio.h>
void main () /* Hàm chính */
{
    int j;
    printf("Nhap 1 ky tu: ");
    j=getchar(); /* Nhận một ký tự từ bàn phím, rồi gán cho j. Giả sử j='A' */
    printf("%c\n",j); /* in ra màn hình ký tự A */
    printf("%d",j); /* in ra màn hình mã ASCII của A là 65 */
}
```


1.5. Lưu ý về các hàm scanf, gets, getchar

Các hàm trên là những hàm nhận dữ liệu từ stdin. (dòng nhập chuẩn có thể hiểu như một vùng nhớ đặc biệt). Chúng nhận dữ liệu theo nguyên tắc sau:

- Nếu trên stdin có đủ dữ liệu thì chúng sẽ nhận một phần dữ liệu mà nó yêu cầu. Phần dữ liệu còn lại vẫn ở trên stdin
- Nếu trên stdin không đủ dữ liệu theo yêu cầu của hàm, thì máy tạm dừng để chờ người sử dụng đưa dữ liệu từ bàn phím lên stdin(cho đến khi gặp phím).
- Xóa khỏi stdin phần dữ liệu đã lấy.
- Hàm gets sẽ xóa ký tự \n trong stdin.
- Hàm scanf và getchar không xóa \n trong stdin.

Như vậy, ta thấy nếu trong chương trình có sử dụng các lệnh scanf, getchar thì các lệnh sử dụng sau sẽ bị trôi (không có tác dụng) do mã phím \n còn lại trong stdin của lệnh scanf hoặc getchar trước đó.

Ví dụ:

```
#include <stdio.h>
void main () /* Ham chinh */
{
    char a,b;
    printf("Nhap mot so: ");
    scanf("%d",&a);
    printf("Nhap mot ky tu : ");
    scanf("%c",&b);
    printf("\n%d %c",a,b);
}
```

Giả sử ta nhập 65 <enter>

Khi đó lệnh scanf thứ 2 bị trôi và kết quả là a=65 và b=<enter>

Giả sử ta nhập 65ABC <enter>

Khi đó lệnh scanf thứ 2 bị trôi và kết quả là a=65 và b=A

+ Để các hàm sau hoạt động đúng ta phải khử ký tự \n còn trong stdin bằng lệnh fflush(stdin) sau lệnh scanf hoặc dùng ký tự đặc tả %*c trong lệnh scanf

Đoạn chương trình trên có thể sửa lại:

```
•••
printf("Nhap mot so: ");
scanf("%d",&a);
fflush(stdin);          /*Hoặc scanf("%d%c",&a)*/
printf("Nhap mot ky tu : ");
scanf("%c",&b);
•••
```

1.6. Hàm putchar

int **putchar**(int ch)

Xuất một ký tự lên màn hình. Kết quả của hàm trả về ký tự xuất nếu thành công, ngược lại cho kết quả EOF (-1)

Ví dụ:

```
char c='A';  
putchar(c);/*in ra màn hình ký tự A*/
```

1.7. Hàm puts

*int puts(char *s)*

Xuất một chuỗi lên màn hình với *s là con trỏ kiểu char trỏ tới ô nhớ đầu của vùng nhớ chứa chuỗi ký tự muốn xuất. Hàm này khi xuất sẽ đưa thêm ký tự \n vào cuối, kết quả của hàm =\n nếu thành công ngược lại = EOF.

Ví dụ

```
/*Chương trình minh họa hàm puts*/  
#include <stdio.h>  
void main()  
{  
    /* Chức năng, công dụng của hàm puts */  
    puts("");  
    puts("Hàm puts có chức năng: Đưa một chuỗi ký tự ra màn hình ");  
    puts("Khi thành công, hàm trả về kí tự cuối cùng được xuất. ");  
    puts("Khi có lỗi hàm trả về EOF.");  
}
```

Kết quả:

Hàm puts có chức năng: Đưa một chuỗi ký tự ra màn hình
Khi thành công, hàm trả về kí tự cuối cùng được xuất.
Khi có lỗi hàm trả về EOF.

2. CÁC HÀM NHẬP XUẤT TRONG CONIO.H

Gồm:

· [Hàm getch\(\) và getche\(\)](#)

· [Hàm cprintf\(\)](#)

· [Hàm cscanf\(\)](#)

• Một số hàm thao tác màn hình thuộc conio.h

2.1. Hàm getch() và getche()

Cú pháp : `int getch(void)`

`int getche(void)`

- Hai hàm này chờ nhận một ký tự trực tiếp từ bộ đệm bàn phím. Nếu bộ đệm rỗng thì chờ. Khi một phím được ấn thì nhận ngay ký tự đó mà không cần phải enter như các hàm nhập từ stdin.

- Hàm getche() cho hiện ký tự lên màn hình còn getch() thì không

- Kết quả trả về của hàm là ký tự được ấn.

Lưu ý: Cách nhận các phím đặc biệt:

Với các phím đặc biệt khi được ấn thì có hai giá trị được gửi lên bộ đệm bàn phím:

Ví dụ khi ta ấn phím F1 thì giá trị đầu là 0, và giá trị kế là 59 được gửi lên bàn phím. Vì vậy để nhận được mã của các phím này ta phải đọc bộ đệm bàn phím thêm một lần nữa..

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
void main() /* Hàm chính */
{
    int a,b;
    char ch;
    do
    {
        clrscr();
        printf("b= ");
        scanf("%c%c", &b); /*nhập số b*/
        printf("a= ");
        scanf("%c", &a); /*nhập số a*/
        printf("a + b=%d ", a+b); /* Tính tổng a+b*/
        ch=getch(); /*Nhận mã lần thứ nhất*/
        if(ch==0) ch=getch(); /*Nhận mã lần thứ hai*/
    }
    while (ch!=59); /*Nếu ch == F1 sẽ thoát khỏi vòng do.. while*/
}
```

2.2. Hàm cprintf()

Cú pháp : hàm **cstdio** có cú pháp giống như hàm printf nhưng màu của nội dung được in bởi hàm cprintf được ấn định bởi hàm textcolor.

2.3. Hàm cscanf()

Hàm **csprintf** có cú pháp và công dụng như hàm scanf, nhưng khác nhau ở hai điểm:

- + Nội dung nhập có màu được ấn định bởi hàm `textcolor`
- + Nhận nội dung trực tiếp từ bộ đệm bàn phím. Vì vậy với hàm `cscanf` ta cũng phải khử ký tự `\n` trong bộ đệm bằng `%"c` hoặc bằng hàm `getch()`

Ví dụ

```
/*Chương trình minh họa hàm Hàm cprintf() và Hàm cscanf() */
#include <stdio.h>
#include <conio.h>
void main()                /* Ham chinh */
{
    char a,b;               /*Khai báo hai biến a,b */
    clrscr();               /*Xóa màn hình */
    textcolor(YELLOW);      /*Đặt màu chữ*/
    textbackground(BLUE);   /*Đặt màu nền*/
    cprintf("\nNhap mot so: ");
    cscanf("%d%"c",&a);     /*Nhap số a, nếu không có %"c sẽ trôi lệnh cscanf dưới*/
    cprintf("\nNhap mot ky tu : ");
    cscanf("%c",&b);        /*Nhap ký tự b*/
    cprintf("\n%d %c",a,b); /*in số a và ký tự b*/
    getch();               /*Dừng màn hình để xem kết quả*/
}
```

2.4. Một số hàm thao tác màn hình thuộc `conio.h`

- § [Hàm xóa màn hình `clrscr`](#)
- § [Hàm đặt tọa độ `gotoxy`](#)
- § [Hàm đặt màu nền `textbackground`](#)
- § [Hàm đặt màu chữ `textcolor`](#)

- § Hàm xóa màn hình `clrscr`

`clrscr();`

(Clear Screen) là hàm xóa toàn bộ màn hình và sau khi xóa con trỏ sẽ ở vị trí góc phía bên trái.

- § Hàm đặt tọa độ `gotoxy`

`gotoxy(int x, int y);`

là hàm đặt con trỏ màn hình vào tọa độ X, Y của màn hình. Tọa độ X là tọa độ cột, tính từ 1 đến 80, tọa độ Y là tọa độ dòng, tính từ 1 đến 25. Màn hình gồm 25 dòng và 80 cột.

Ví dụ

`gotoxy (30,10);`

□ § Hàm đặt màu nền *textbackground*

Cú pháp : **void textbackground(int color);**

Chức năng: Chọn màu nền

Color là một biểu thức nguyên có giá trị từ 0 đến 7 tương ứng với một trong 8 hằng số màu đầu tiên của bảng màu văn bản.

Ví dụ

*textbackground(3); tương đương textbackground(CYAN); /*Màu xanh cẩm thạch*/*

□ § Hàm đặt màu chữ *textcolor*

Cú pháp : **void textcolor(int newColor);**

Chức năng: Lựa chọn màu ký tự mới newColor.

newColor là một biểu thức nguyên có giá trị từ 0 đến 15 tương ứng với một trong các hằng số màu của bảng màu văn bản.

Ví dụ

*textcolor(4); tương đương textcolor(RED); /*Màu đỏ*/*

3. CÁC BÀI TẬP MINH HỌA

Bài 1: Viết chương trình nhập vào số dặm đổi ra số km và ngược lại (biết 10000 km=5400 dặm).

```
/* Chương trình nhập vào số dặm, tính số km */
#include <stdio.h>
main()
{
    float sdam,skm; /*Khai báo biến */
    printf(" Nhập số dặm => ");
    scanf("%f", &sdam);
    skm=sdam* (float) 10000/5400;
    printf("\nKết quả là: %.2f dặm => %.2f km", sdam,skm);
}
```

Kết quả:
Nhập số dặm =>23
Kết quả là: 23.00 dặm =>42.59 km

```
/* Chương trình nhập vào số km, tính số dặm */
#include <stdio.h>
main()
{
    float sdam,skm; /* Khai báo biến. */
    printf(" Nhập số km => ");
```

```
scanf("%f", &skm);
sdam=skm* (float) 5400/10000;
printf("\nKết quả là: %.2f km => %.2f dam", skm, sdam);
}
```

Kết quả:

Nhập số km =>23

Kết quả là: 23.00 km =>12.42 dam

Bài 2: Viết chương trình nhập vào a,b,c (giả sử a,b,c thỏa điều kiện là 3 cạnh của tam giác: $a < b + c$; $c < a + b$; $b < a + c$). Tính diện tích của tam giác. Biết:

$$s = \sqrt{p(p-a)(p-b)(p-c)},$$

$$\text{với } p = (a+b+c)/2;$$

```
/*      Chương trình tính diện tích của tam giác biết ba cạnh a,b,c      */
#include <stdio.h>
#include <math.h>
main()
{
    int a,b,c;                                /* Khai báo biến. */
    float s,p;
    printf(" Nhập cạnh a => ");
    scanf("%d", &a);
    printf(" Nhập cạnh b => ");
    scanf("%d", &b);
    printf(" Nhập cạnh c => ");
    scanf("%d", &c);
    p=(float) (a+b+c)/2;                      /*Xác định p*/
    s=sqrt(p*(p-a)*(p-b)*(p-c));              /*Xác định diện tích*/
    printf("\nDiện tích tam giác=%.2f",s);    /*In kết quả ra màn hình*/
}
```

Kết quả:

Nhập cạnh a =>3

Nhập cạnh b =>4

Nhập cạnh c =>5

Diện tích tam giác=6.00

Bài 3: Viết chương trình nhập từ bàn phím và sau đó xuất lên màn hình các thông tin của một mặt hàng bao gồm: Tên mặt hàng, trọng lượng, đơn giá, mã chất lượng, số lượng.

```
/*      Chương trình nhập, xuất thông tin của một mặt hàng      */
#include <stdio.h>
main()
{
    char ten_mat_hang[20];                    /* Khai báo một chuỗi tối đa 20 ký tự. */
    float trong_luong;
    long don_gia;
    char ma_chat_luong;
    unsigned so_luong;
    printf(" \nNhập dữ liệu từ bàn phím:");
    printf(" \nTên mặt hàng=> "); gets(ten_mat_hang);
    printf(" \nTrọng lượng=> "); scanf("%f", & trong_luong);
    printf(" \nĐơn giá=> "); scanf("%ld%*c", & don_gia);
}
```

```

printf("\nMã chất lượng=> "); scanf("%c", &ma_chat_luong);
printf("\n số lượng => "); scanf("%u", &so_luong);
printf("\nTên mặt hàng: %s Trọng lượng :%.2f Đơn giá:%ld Mã chất lượng: %c Số lượng:%u ",
      ten_mat_hang, trong_luong, don_gia, ma_chat_luong, so_luong);
/*In kết quả ra màn hình*/
}

```

4. 4. CÁC BÀI TẬP TỰ LÀM

Bài 1

Viết chương trình nhập thông tin tiêu thụ điện của khách hàng gồm: Tên khách hàng (kiểu chuỗi), chỉ số cũ (số nguyên), chỉ số mới (số nguyên), đơn giá (số nguyên), và xuất thông tin lên màn hình gồm tên khách hàng, tháng, số kwh tiêu thụ và số tiền phải trả.

Bài 2

Làm lại các bài tập trong *Bài một*, với dữ liệu nhập từ bàn phím.

BÀI 4

CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG C

1. CÂU LỆNH, KHỐI LỆNH

1.1. Khái niệm về câu lệnh

Mỗi câu lệnh thực hiện một chức năng nào đó (như lệnh gán, lệnh xuất dữ liệu ra màn hình), câu lệnh có thể được viết trên một hoặc nhiều dòng và được kết thúc bằng dấu chấm phẩy (;).

1.2. Khái niệm về khối lệnh trong chương trình C

Một dãy các câu lệnh được đặt trong cặp dấu { và } được gọi một là một khối lệnh,

khi đó khối lệnh này được xem như một câu lệnh riêng lẻ. Những câu lệnh của một

hàm, những câu lệnh của một cấu trúc phải được đặt vào dấu {}

Các toán tử điều khiển cho phép thay đổi trật tự thực hiện các câu lệnh (khối lệnh) do đó máy có thể đang từ một câu lệnh này nhảy tới thực hiện một câu lệnh ở trước, hoặc sau nó.

2. TOÁN TỬ IF

· Cấu trúc if (biểu thức ĐK) <khối lệnh >;

· Cấu trúc if (biểu thức ĐK) <khối lệnh 1>; else <khối lệnh 2>

· Ví dụ minh họa

2.1. if (biểu thức ĐK)

<khối lệnh >;

Nếu biểu thức cho kết quả khác không thì thực hiện khối lệnh. Nếu khối lệnh có từ hai lệnh trở lên thì phải đặt vào dấu { }

Ví dụ

```
/*Chương trình minh họa cấu trúc if*/
#include <stdio.h>
void main()
{
    float number; /* Giá trị được nhập bởi người sử dụng. */
    printf("Nhập một số trong khoảng từ 1 đến 10 => ");
    scanf("%f",&number);
    if (number > 5)
        printf("Số bạn nhập lớn hơn 5.\n");

    printf("%f là số bạn nhập.",number);
}
```

Kết quả:

Nhập một số trong khoảng từ 1 đến 10 =>7

Số bạn nhập lớn hơn 5.

7.000000 là số bạn nhập.

2.2. if (biểu thức ĐK) <khối lệnh 1>;

else <khối lệnh 2>;

Nếu biểu thức cho kết quả khác không thì thực hiện khối lệnh 1, ngược lại thì cho thực hiện khối lệnh thứ hai.

Ví dụ

```
/*Chương trình minh họa cấu trúc if ... else*/
#include <stdio.h>
void main()
{
    float number; /* Giá trị số được nhập bởi người sử dụng. */
    printf("\n\n Nhập một số trong khoảng từ 1 đến 10 =>");
    scanf("%f",&number);
    if (number > 5.0)
        printf("Số bạn nhập lớn hơn 5.\n");
    else
        printf("Số bạn nhập nhỏ hơn hoặc bằng 5.\n");
    printf("Giá trị số bạn nhập là %f ",number);
}
```

Kết quả:

Nhập một số trong khoảng từ 1 đến 10 =>3
Số bạn nhập nhỏ hơn hoặc bằng 5.
Giá trị số bạn nhập là 7.000000

Lưu ý: Biểu thức ĐK phải đặt vào dấu ngoặc tròn. C cho phép ta viết các cấu trúc if lồng vào nhau.

2.3. Bài toán minh họa

*/*Cấu trúc rẽ nhánh (if, if.. else) để giải quyết bài toán :*

*Nhập vào 3 số và xuất ra theo thứ tự tăng dần */*

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a,b,c;
    clrscr();
    printf("Nhap vao 3 so a,b,c:");
    scanf("%d%d%d",&a,&b,&c);
    if (a>b)
    {
        a=a+b; b=a-b; a=a-b;
    }
    if (c<a)
        printf("%d %d %d",c,a,b);
    else
    if (c<b)
        printf("%d %d %d",a,c,b);
    else
        printf("%d %d %d",a,b,c);
    getch();
}
```

Kết quả:

Nhap vao 3 so a,b,c: 5 3 7
3 5 7

3. TOÁN TỬ SWITCH

3.1. Cú pháp

switch (Biểu thức)

```
{    case n1:                                các câu lệnh
    case n2:                                các câu lệnh
    ....
    case nk:                                các câu lệnh
```

[*default*:

các câu lệnh]

}

- ni là các hằng số nguyên, ký tự.
- Phụ thuộc vào giá trị của biểu thức viết sau **switch**, nếu:
 - Giá trị này = ni thì thực hiện câu lệnh sau case ni;
 - Khi giá trị biểu thức khác tất cả các ni thì thực hiện câu lệnh sau **default** nếu có, hoặc thoát khỏi câu lệnh **switch**.
- Khi chương trình đã thực hiện xong câu lệnh của **case** ni nào đó thì nó sẽ thực hiện luôn các lệnh thuộc **case** bên dưới nó mà không xét lại điều kiện (do các ni được xem như các nhãn). Vì vậy, để chương trình thoát khỏi lệnh switch sau khi thực hiện xong một trường hợp, ta dùng lệnh **break**.

Ví dụ

```
/*Chương trình minh họa cấu trúc switch*/
#include <stdio.h>
void main()
{
    char selection; /*Dữ liệu được chọn từ người sử dụng. */
    printf("\n\n Chọn từ (A,B hoặc C) để xác định:\n");
    printf("A] Điện thế B] Dòng điện C] Điện trở\n");
    printf("Lựa chọn của bạn (A, B, or C) => ");
    scanf("%c",&selection);
    switch(selection)
    {
        case 'A' : printf("V = I * R"); break;
        case 'B' : printf("I = V / R"); break;
        case 'C' : printf("R = V / I"); break;
        default : printf("Không có một chọn lựa nào.");
    } /* Kết thúc switch. */
}
```

Kết quả:

Chọn từ (A,B hoặc C) để xác định:
A] Điện thế B] Dòng điện C] Điện trở
Lựa chọn của bạn (A, B, or C) => A
V=I*R

3.2. Bài toán minh họa

/ Sử dụng cấu trúc switch để xếp loại học sinh theo điểm như sau:
Từ 0 đến 3: Kém. 4: Yếu. Từ 5 đến 6: Trung bình. Từ 7 đến 8: Khá. Từ 9 đến 10:
Giỏi.*

```
*/
#include <stdio.h>
void main()
{
    char diem; /* Điểm nhập bởi người sử dụng. */
    printf("\n\nNhập diem\n");
    scanf("%d",&diem);
    switch(diem)
```

<pre> { 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"); } } </pre>
<p>Kết quả: Nhập điểm 9 Gioi</p>

4. TOÁN TỬ FOR

4.1. Cú pháp

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

<Khối lệnh> ;

Toán tử **for** gồm 3 biểu thức và phần lệnh của toán tử này. Bất kỳ biểu thức nào trong 3 biểu nói trên đều có thể vắng mặt nhưng phải giữ dấu chấm phẩy. *Ví dụ: for(i=-1; ++i<n;)*

Hoạt động của toán tử **for**:

1. Tính giá trị của biểu thức 1
2. Tính giá trị của biểu thức 2
3. Nếu biểu thức 2 $\neq 0$ thì cho thực hiện các lệnh của vòng lặp, ngược lại cho thoát khỏi lệnh for.
4. Tính giá trị biểu thức 3 rồi quay lại bước 2

Ví dụ

<pre> /*Chương trình minh họa vòng for */ #include <stdio.h> main() { int time; /* Biến đếm. */ /* Vòng lặp bắt đầu ở đây. */ for(time = 1; time <= 5; time = time + 1) printf("Giá trị của biến đếm time = %d \n",time); /* Vòng lặp dừng ở đây. */ printf("Kết thúc vòng lặp."); } </pre>
<p>Kết quả: Giá trị của biến đếm time =1 Giá trị của biến đếm time =2 Giá trị của biến đếm time =3</p>

Giá trị của biến đếm time =4
Giá trị của biến đếm time =5
Kết thúc vòng lặp.

4.2. Bài tập minh họa

/ Sử dụng cấu trúc for để giải quyết bài toán đổi tiền:
Tìm tất cả các cách đổi 1000 đồng bằng các tờ giấy bạc 500 đ, 200đ, 100 đ.*

```
*/  
#include <stdio.h>  
main()  
{  
    int so_cach,to_500,to_200,to_100;  
    so_cach=0;  
    for (to_500=0;to_500<=2; to_500++)  
        for (to_200=0;to_200<=5; to_200++)  
            for (to_100=0;to_100<=10; to_100++)  
                if ( 100*to_100+200*to_200+500*to_500==1000)  
                    {so_cach++;  
                     printf("\n Tien 1000 d=");  
                     if (to_100) printf(" %2dX100d",to_100);  
                     if (to_200) printf(" %2dX200d",to_200);  
                     if (to_500) printf(" %2dX500d",to_500); }  
                     printf("\n Co tat ca %d cach", so_cach);  
}
```

Kết quả

```
Tien 1000 d= 10X100 d  
Tien 1000 d= 8X100 d  1X200 d  
Tien 1000 d= 6X100 d  2X200 d  
Tien 1000 d= 4X100 d  3X200 d  
Tien 1000 d= 2X100 d  4X200 d  
Tien 1000 d= 5X200 d  
Tien 1000 d= 5X100 d  1X500 d  
Tien 1000 d= 3X100 d  1X200 d  1X500 d  
Tien 1000 d= 1X100 d  2X200 d  1 X500 d  
Tien 1000 d= 2X500 d  
Co tat ca 10 cach
```

4.3. Trong lệnh for còn có thể sử dụng lệnh break và continue:

- ☐ Khi gặp câu lệnh **break** bên trong thân của toán tử **for**, máy sẽ thoát khỏi vòng **for** ngay lập tức.

Ví dụ

```
/* Chương trình minh họa vòng for với câu lệnh break */  
#include <stdio.h>  
void main()
```

```

{
    int dem;    /* Biến đếm. */
    /* Vòng lặp bắt đầu ở đây. */
    for(dem = 1; dem <= 5; dem = dem + 1)
    {
        printf("\ndem=%d",dem);
        break; /*Gặp break máy sẽ thoát khỏi vòng for ngay lập tức */
        printf("\nThu nhien lenh break");
    }
}

```

Kết quả in ra là:

Dem=1;

□ Trái với lệnh **break** là lệnh **continue**. Câu lệnh **continue** dùng để bắt đầu một vòng mới của chu trình bên trong nhất chứa nó.

□ Nói một cách chính xác hơn:

Khi gặp câu lệnh **continue** bên trong thân của toán tử **for**, máy sẽ chuyển đến bước khởi đầu lại (bước 4 trong điểm “Sự hoạt động của **for**”). Một điểm cần lưu ý: Lệnh **continue** chỉ áp dụng cho các chu trình chứ không áp dụng cho **switch**.

Ví dụ

```

/* Chương trình minh họa vòng for với câu lệnh continue */
#include <stdio.h>
main()
{
    int dem;    /* Biến đếm. */
    /* Vòng lặp bắt đầu ở đây. */
    for (dem = 1; dem <= 5; dem = dem + 1)
    {
        printf("\ndem=%d",dem);
        continue; /*Gặp continue máy sẽ quay lại bước 4 trong hoạt động của toán tử for */
        printf("\nThu nhien lenh continue"); /*lệnh này không được thực hiện*/
    }
}

```

Kết quả in ra là:

Dem=1;
Dem=2;
Dem=3;
Dem=4;
Dem=5;

5. TOÁN TỬ WHILE

while (biểu thức)

<khối lệnh>;

Nếu biểu thức còn khác không thì còn thực hiện các lệnh của vòng lặp. Biểu thức

trong ngoặc của **while** có thể chứa nhiều biểu thức phân cách nhau bởi dấu phẩy, khi đó giá trị của biểu thức là giá trị của biểu thức cuối cùng.

Ví dụ:

```

/*Chương trình minh họa cấu trúc while*/
#include <stdio.h>
main()
{
    int time = 1;    /* Biến đếm. */
    while(time <= 5)
    {
        printf("Gia tri cua bien dem = %d\n", time);
        time++;
    } /* Kết thúc vòng while. */
    printf("Ket thuc vong lap.");
}

```

Kết quả
 Gia tri cua bien dem =1
 Gia tri cua bien dem =2
 Gia tri cua bien dem =3
 Gia tri cua bien dem =4
 Gia tri cua bien dem =5
 Ket thuc vong lap.

6. TOÁN TỬ DO .. WHILE

do

<khối lệnh>;

while (biểu thức);

Thực hiện khối lệnh cho đến khi biểu thức có giá trị bằng 0.

Ví dụ:

```

/*Chương trình minh họa cấu trúc do .. while*/
#include <stdio.h>
main()
{
    int time;    /* Biến đếm. */
    time = 1;
    do {
        printf("Gia tri cua bien dem = %d\n", time);
        time++;
    } while(time <= 5);
    printf("Ket thuc vong lap. ");
}

```

Kết quả
 Gia tri cua bien dem =1
 Gia tri cua bien dem =2
 Gia tri cua bien dem =3
 Gia tri cua bien dem =4
 Gia tri cua bien dem =5
 Ket thuc vong lap.

7. CHÚ Ý KHI GẶP LỆNH *break* VÀ *continue* TRONG *while* VÀ *do... while*

*Ghi chú: Khi gặp câu lệnh **continue** bên trong thân của **while** hoặc **do while**, máy sẽ chuyển đến xác định giá trị biểu thức viết sau từ khóa **while**, và sau đó tiến hành kiểm tra điều kiện kết thúc chu trình. Trường hợp gặp câu lệnh **break**, máy sẽ thoát khỏi vòng **while** hay **do..while** ngay lập tức.*

Ví dụ:

ví dụ 1:

```
/* Chương trình minh họa vòng while với câu lệnh break */
#include <stdio.h>
void main()
{
    int dem=1;    /* Biến đếm. */
    /* Vòng lặp bắt đầu ở đây. */
    while(dem <= 5)
    {
        printf("\ndem=%d",dem);
        break; /*Gặp break máy sẽ thoát khỏi vòng while ngay lập tức */
        dem++;
    }
}
```

Kết quả in ra là:

Dem=1;

ví dụ 2:

```
/*Chương trình minh họa cấu trúc do .. while với câu lệnh continue*/
#include <stdio.h>
main()
{
    int time;    /* Biến đếm. */
    time = 1;
    do {
        printf("Gia tri cua bien dem = %d\n", time);
        time++;
        continue;
        printf("Thu nghiem cau lenh continue \n", time); /*Lệnh này không được thực
hiện*/
    } while(time <= 5);
    printf("Ket thuc vong lap. ");
}
```

```

Kết quả
Gia trị của biến dem =1
Gia trị của biến dem =2
Gia trị của biến dem =3
Gia trị của biến dem =4
Gia trị của biến dem =5
Kết thúc vòng lặp.

```

8. TOÁN TỬ GOTO

Nhãn có cùng dạng như tên biến và có dấu hai chấm (:) đứng sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình. Ví dụ: *t:s+=a; /* t là nhãn của câu lệnh gán */*

Cú pháp của toán tử **goto**

goto nhãn;

Khi gặp toán tử này máy sẽ nhảy tới thực hiện câu lệnh viết sau từ khóa **goto**.

Ví dụ:

• • •

int s=1,a=3;

*goto t; /*Nhảy tới thực hiện lệnh sau t*/*

++a;

*t: s+=a; /*Kết quả s=4, bỏ qua lệnh ++a*/*

• • •

Chú ý:

-Câu lệnh **goto** và nhãn cần nằm trong một hàm.

-Không cho phép dùng toán tử **goto** nhảy từ ngoài vào trong một khối lệnh, tuy nhiên nhảy từ trong ra ngoài khối lệnh là hoàn toàn hợp lệ.

9. BÀI TẬP MINH HỌA

Bài 1: Cho đoạn chương trình dưới đây, xác định kết quả in ra:

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

```
#include<conio.h>
```

```
main()
```

```
{
```

```
int a,b=4;
```

```
clrscr();
```

```
switch((a=2)?5:2)
```

```
{
```

```
case 5:b+=2;
```

```
default:a=b--;
```

```
case 2:a--;
```

```
}
```

```
printf("%d %d",a,--b);
```



```

    getch();
}

```

Kết quả in ra là:

1 4 Bài 2: Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
int a=-1,b;
main()
{
    clrscr();
    while(a>=b&&--b)
    if(a==b)
    break;
    printf("%d %d",a,b);
    getch();
}

```

Kết quả in ra là:-1 0 **Bài 3:** Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int a,b=0;
    clrscr();
    while(a=b++==1)
    if(a==b)
    break;
    else
    a++;
    printf("%d %d",b,a);
    getch();
}

```

Kết quả in ra là:1 0 **Bài 4:** Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int i,a=0;
    clrscr();
    for(i=0;i<3;i++)
    {
        if(i==2)
        continue;
        a+=i;
        if(i>1)
        break;
        printf("%d ",a);
    }
}

```

```

}
getch();
}

```

Kết quả in ra: 0 1 Bài 5: Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
int i=0 ;
main()
{
int a=2;
clrscr();
for(;i<a;i++);
printf("%d ",i*a);
getch();
}

```

Kết quả in ra là: 4 Bài 6: Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
main()
{
int i,j;
clrscr();
for(i=0,j=1;i<5;i++,j+=i++)
printf("i=%d \t j=%d \t i+j=%d\n ",i,j,i+j);
}

```

Kết quả in ra là: i=0 j=1 i+j=1 i=2 j=2 i+j=4 i=4

j=5 i+j=9 Bài 7: Viết chương trình tìm tất cả các số nguyên có ba chữ số sao cho tổng tam thừa của ba chữ số hàng trăm, hàng chục, hàng đơn vị sẽ bằng số nguyên đó. Ví dụ: $1^3+5^3+3^3=153$

/ Tìm tất cả các số nguyên abc sao cho $abc=a^3+b^3+c^3$. */*

```

#include <stdio.h>
main()
{
int a,b,c;
for (a=1;a<=9; a++)
    for (b=0;b<=9; b++)
        for (c=0;c<=9; c++)
            if (a*a*a+b*b*b+c*c*c==a*100+b*10+c)
                printf("\n%d%d%d\n",a,b,c);
}

```

Kết quả
153
370
371
407

Bài 8: Cho đoạn chương trình dưới đây, xác định kết quả in ra:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int i=1,n=0;
    clrscr();
    while(i>n)
        {while (i<4)
            i+=2; n++;
            i-=3;
        }
    printf("%d %d",i,n);
    getch();
}

```

Kết quả in ra là: 1 2

Bài 9: Viết chương trình nhập vào ngày tháng kiểm tra xem đó là ngày thứ mấy trong năm.

```

/* Sử dụng cấu trúc for kết hợp với switch để giải bài toán:
Nhập vào ngày, tháng: xác định đó là ngày thứ mấy trong năm */
#include <stdio.h>
main()
{
    unsigned ngay,thang,ngay_thu,i;
    printf("Nhap vao ngay, thang:");
    scanf("%u%u",&ngay,&thang);
    ngay_thu=ngay;
    for(i=1;i<thang;i++)
    {
        switch(i)
        {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: ngay_thu+=31; break;
            case 4:
            case 6:
            case 9:
            case 11: ngay_thu+=30; break;
            default: ngay_thu+=28;
        }
    }
    printf("Ngày %u thang %u la ngay thu %u trong nam",ngay,thang,ngay_thu);
}

```

```
}
```

Kết quả

Nhap vào ngay, thang: 15 2

Ngày 15 tháng 2 là ngày thu 46 trong năm

10. BÀI TẬP TỰ LÀM

Bài 1: Cho biết kết quả chương trình sau

```
#include <stdio.h>
void main()
{
    int n,p;
    n=0;
    while (n<=5) n++;
    printf("I: n=%d\n",n);
    n=p=0;
    while (n<=8) n+=p++;
    printf("II: n=%d\n",n);
    n=p=0;
    while (n<=8) n+=++p;
    printf("III: n=%d\n",n);
    n=p=0;
    while (p<=5) n+=p++;
    printf("IV: n=%d\n",n);
    n=p=0;
    while (p<=5) n+=++p;
    printf("IV: n=%d\n",n);
}
```

Bài 2: Hãy viết lại đoạn lệnh sau bằng cách dùng toán tử điều kiện ? :

```
if (x>=0)
if (x<=5)
    printf("0<=x<=5\n");
else
    printf("x>5\n");
else
    printf("x<0\n");
```

Bài 3: Viết chương trình tính tổng của n số đầu tiên của dãy số sau:
 $1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$.

Bài 4: Hãy viết chương trình in ra tất cả các cặp số dương (a,b) sao cho $a < b < 1000$ và $(a^2 + b^2 + 1)/a * b$ là một số nguyên. Có phương pháp nào biểu diễn mọi lời giải có thể không.

Bài 5: Phương trình $a^5+b^5+c^5+d^5+e^5=f^5$ có nghiệm nguyên duy nhất thỏa $0 < a \leq b \leq c \leq d \leq e \leq f \leq 75$. Hãy viết chương trình tìm ra lời giải đó. Chương trình có thể chạy nhanh hơn không nếu biết rằng b,c,d lớn hơn hay bằng 40 và nhỏ hơn 50.

Gợi ý: Sử dụng các vòng for lồng nhau

Bài 6: Nhập vào ngày tháng kiểm tra xem ngày tháng đó có hợp lệ không.

Gợi ý: Sử dụng lệnh switch. Lưu ý các tháng 1,3,5,7,8,10,12 có 31 ngày, các tháng 4,6,9,11 có 30 ngày và tháng 2 có 28 ngày.

Bài 7: Viết chương trình in ra lịch 12 tháng của năm 1998 cùng lúc trên màn hình.

Gợi ý: Đầu tiên phải xác định được ngày 1 của mỗi tháng là ngày thứ mấy, từ đó xác định tọa độ cột bắt đầu cho tháng đó. Chú ý: Sử dụng lệnh gotoxy(cot,hang); đặt con trỏ tại vị trí cot, hang.

cot : cột đặt con trỏ
hang : hàng đặt con trỏ.

Bài 8: Viết chương trình in ra hình sau

Gợi ý: xuất lần lượt 4 tam giác vuông tạo thành hình

```
*      *****
vẽ đồ
**     *****
***    *****
*****
*****
      ** *****
      ***  *****
      **** *****
      *****
```

Bài 9: Viết chương trình xuất ra cây thông.

Gợi ý: Xuất lần lượt 3 tam giác cân có chiều dài khác nhau, chồng lên nhau.

Bài 10: Viết chương trình in ra bản cửu chương trên màn hình.

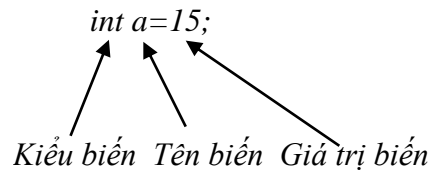
BÀI 5

CON TRỎ VÀ ĐỊA CHỈ

1. TOÁN TỬ ĐỊA CHỈ

1.1. Các khái niệm liên quan đến biến

- § **Tên biến**
- § **Kiểu biến**
- § **Giá trị của biến**



Ví dụ:

1.2. Địa chỉ của biến

§ **Khái niệm**

Nhận xét từ ví dụ ở mục 1.1 :

```
int a=15;
```

Theo khai báo này, máy sẽ cấp phát cho biến a một khoảng nhớ gồm 2 byte liên tiếp.

Địa chỉ của biến là số thứ tự của byte đầu tiên trong một dãy các byte liên tiếp mà máy dành cho biến.

§ **Phân loại địa chỉ biến**

Từ khái niệm về địa chỉ, ta nhận xét thấy : Địa chỉ của hai biến kiểu int liên tiếp cách nhau 2 byte, địa chỉ của hai biến kiểu float liên tiếp cách nhau 4 byte. Nên có thể phân biệt được các kiểu địa chỉ: *Địa chỉ kiểu int, địa chỉ kiểu float, địa chỉ kiểu double.....*

§ **Phép lấy địa chỉ của một biến**

Phép toán

&x
cho ta địa chỉ của biến x.

2. CON TRỎ

2.1. Khái niệm biến con trỏ

Là một loại biến dùng để lưu địa chỉ, mỗi loại địa chỉ sẽ có một kiểu con trỏ tương ứng (phụ thuộc vào loại dữ liệu lưu trữ trong địa chỉ đó).

2.2. Phân loại con trỏ

Con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu int. Tương tự ta có con trỏ kiểu float, kiểu double vv...

2.3. Khai báo biến con trỏ

§ Con trỏ không kiểu

Có thể chứa bất kỳ một địa chỉ nào
void *tên biến con trỏ ;

*ví dụ: void *p, *q;*

§ Con trỏ có kiểu

Chỉ chứa được những địa chỉ của loại dữ liệu phù hợp với kiểu dữ liệu mà ta đã khai báo cho con trỏ:

<kiểu dữ liệu> *tên biến con trỏ;

*Ví dụ: int *p, *q;
float *x;*

3. QUI TẮC SỬ DỤNG CON TRỎ TRONG CÁC BIỂU THỨC

Có hai dạng sử dụng:

1.1. Tên con trỏ

sử dụng địa chỉ chứa trong con trỏ:

Ví dụ:

```
float a, *p, *q;  
p=&a; /* lưu địa chỉ của biến a vào con trỏ Program Manager */  
q=p; /* lưu địa chỉ trong p vào con trỏ q*/
```

1.2. Dạng khai báo của con trỏ

sử dụng giá trị lưu tại vùng nhớ mà con trỏ trỏ tới.

Ví dụ:

```
float x=5,y,z=20, *px, *pz;  
px=&x; /* khi đó *px = x = 5 */  
pz=&z; /* *pz = z = 20 */  
khi đó ba biểu thức sau là tương đương:  
y=3*x+z;  
*py=3*x+z;  
*py=3*( *px )+ *pz;
```

Tóm lại: px, pz có kiểu là con trỏ float thì *px, *pz thuộc kiểu số thực.

4. QUI TẮC VỀ KIỂU GIÁ TRỊ TRONG KHAI BÁO

Mọi thành phần của cùng một khai báo (biến, phần tử mảng, hàm, con trỏ) khi xuất hiện trong biểu thức đều cho cùng một kiểu giá trị.

5. BỘ ĐỊNH TÍNH CONST VỚI CON TRỎ

Từ khóa const có hai tác dụng:

- ☐ Dùng để khai báo và khởi đầu giá trị trong các biến trong mà sau này giá trị của nó không cho phép thay đổi bởi các lệnh trong chương trình. Chúng được gọi là các đối tượng hằng.
- ☐ Dùng để khai báo các đối con trỏ mà trong thân hàm ta không được phép làm thay đổi giá trị các đối tượng được trỏ bởi các đối này.

Ví dụ 1

```
#include <stdio.h>  
void main()  
{  
    const int a=10;  
    a++; /* Sai, không được thay đổi biến const a */  
    printf("\n a=%d",a);  
}
```


Ví dụ 2

Biên dịch hàm:

```
void thu_nghiem(const int *stt)
{
    *stt = *stt+1; /* Sai, do gia tri duoc tro boi con tro stt khong duoc thay doi */
}
```

6. BÀI TẬP MINH HỌA

Bài 1: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include <stdio.h>
#include <conio.h>
void main () /* Ham chinh */
{
    int temp,a=7,b=3;
    int *pa,*pb;
    clrscr();
    *pa=a;
    *pb=b;
    printf("Truoc: A = %d B= %d\n",*pa,*pb);
    temp=*pa;
    *pa=*pb;
    *pb=temp;
    printf("Sau: A = %d B= %d\n",*pa,*pb);
    getch();
}
```

Kết quả in ra:

Truoc: A=7 B=3

Sau : A =3 B= 7

Bài 2: Cho đoạn chương trình sau. Xác định kết quả in ra

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

```
#include<conio.h>
```

```
void main()
```

```
{
```

```
    int *x,y=2;
```

```
    clrscr();
```

```
    *x=y;
```

```
    *x+=y++;
```

```
printf("%d %d",*x,y);
getch();
}
```

Kết quả in ra:

4 3

-

Bài 3: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
#include<conio.h>
void main()
{
int tam=1;
int *x,y=1;
*x=0;
clrscr();
while(*x<=y)
{
*x+=tam;
tam++;
}
printf("%d %d",y,*x);
getch();
}
```

Kết quả in ra

1 3

7. BÀI TẬP TỰ LÀM

Bài 1: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x=1,y=2;
int *a;
clrscr();
*a=x;
*a-=1;
*a=++y;
y-=2;
printf("%d %d",*a,y);
getch();
}
```

}

Bài 2: Cho khai báo biến sau

`int *pint;`

`float a;`

`char c;`

`double *pd;`

Hãy chọn phát biểu sai cú pháp:

a. `a = *pint;`

b. `c = *pd;`

c. `*pint = *pd;`

d. `pd = a`

BÀI 6

HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

1. 1. VÀI NÉT VỀ HÀM VÀ CHƯƠNG TRÌNH

1.1. Khái niệm về chương trình

Một chương trình bao gồm một hoặc nhiều hàm. Hàm `main()` là thành phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện từ câu lệnh đầu tiên của hàm `main()` cho đến khi gặp dấu `}` cuối cùng của hàm này.

1.2. Khái niệm về Hàm

☐ Khái niệm

Là một đoạn chương trình độc lập thực hiện trọn vẹn một công việc nhất định, rồi trả về một giá trị cho chương trình gọi nó.

☐ Đặc điểm của hàm

- ☐ Là một đơn vị độc lập của chương trình.
- ☐ Không cho phép xây dựng một hàm bên trong một hàm khác.

1.3. Tổ chức một chương trình trong C

☐ Bố cục một chương trình trong C

Một chương trình C được tổ chức theo mẫu:

hàm 1

.....

hàm 2

.....

.....

hàm n

Ở các vị trí bên ngoài hàm (những vị trí ...) ta có thể đặt các toán tử `#include`, `#define`, định nghĩa kiểu dữ liệu, khai báo biến ngoài, biến tĩnh ngoài.

□ Cách truyền dữ liệu giữa các hàm

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

- Sử dụng đối của hàm.
- Sử dụng biến ngoài, biến tĩnh ngoài.

2. 2. VÍ DỤ VỀ CHƯƠNG TRÌNH CÓ HÀM

Chương trình tính lũy thừa 3 của x

```
/*Chương trình tính lũy thừa 3 của x*/
#include <stdio.h>
long luy_thua(int x);
main () /* Ham chinh */
{
    int x;
    printf(" \nNhap so muon tinh luy thua: ");
    scanf("%d",&x);
    printf("\nKet qua = %ld",luy_thua(x));
    getch();
}
long luy_thua(int x)
{
    long ketqua;
    ketqua=x*x*x;
    return ketqua;
}
```

Nguyên mẫu của hàm

Các đối số của hàm

khai báo hàm (nội dung của hàm)

Gán tên hàm cho kết quả trả về

Kết quả in ra:

Nhap so muon tinh luy thua: 3

Ket qua =9

3. 3. CÁCH VIẾT MỘT HÀM

3.1. Xác định mục đích hàm

Để viết một hàm trước hết ta phải xác định mục đích của hàm là dùng để làm gì, trên cơ sở đó, ta mới xác định được các thành phần của hàm.

3.2. Xác định các thành phần của hàm

Các thành phần của hàm bao gồm:

- ☐ Nguyên mẫu của hàm
- ☐ Kiểu giá trị của hàm
- ☐ Tên hàm
- ☐ Các tham số của hàm
- ☐ Nội dung của hàm

☐ Nguyên mẫu của hàm

Bao gồm:

<kiểu dữ liệu của hàm> <tên hàm(danh sách các tham số)>;

Ta có thể không ghi nguyên mẫu của hàm, tuy nhiên không nên làm như vậy, vì đối với một hàm có nguyên mẫu thì khi biên dịch C sẽ kiểm tra việc truyền tham số, giá trị trả về có phù hợp hay không rồi mới cho thực hiện hàm.

Nguyên mẫu của các hàm có trong chương trình ta nên đặt trước hàm main() và nhóm vào một chỗ để dễ kiểm soát.

☐ Kiểu giá trị của hàm

Giá trị trả về của hàm phải được xác định dựa vào mục đích của hàm và trong thân hàm ta phải trả về đúng kiểu giá trị đã định ban đầu. Nếu các hàm không trả về giá trị ta phải khai báo kiểu void.

☐ Tên hàm

Đặt theo qui định đối với danh định, nên đặt ngắn gọn và phản ánh phần nào mục đích của hàm. Tên hàm trong nguyên mẫu và khi khai báo phải giống nhau.

□ **Tham số của hàm**

□ Phân loại theo cách sử dụng

Gồm:

□ Tham số hình thức

Các tham số mà ta ghi trong nguyên mẫu hay ghi lúc khai báo hàm gọi là **tham số hình thức**.

□ Tham số thực

Các giá trị, biến mà ta ghi sau tên hàm khi gọi hàm đó để thực hiện gọi là **tham số thực**. Trong C, các tham số thực lại chia ra làm hai loại:

□ Tham chiếu: Là các tham số thực mà ta truyền cho Hàm dưới dạng con trỏ (dạng địa chỉ). Tham chiếu mới ghi nhận lại được những kết quả vừa tính toán trong Hàm khi Hàm kết thúc.

□ Tham trị: Là các tham số thực mà ta truyền cho Hàm dưới dạng biến. Tham trị không bảo lưu lại những kết quả thay đổi của nó được tính toán trong Hàm khi Hàm kết thúc.

□ Phân loại theo công dụng

Tham số của một hàm có hai công dụng:

- □ Cung cấp các giá trị cho hàm khi ta gọi nó thực hiện .
- □ Lưu các kết quả tính toán được trong quá trình hàm hoạt động .

Như vậy ta có các loại tham số:

□ Tham số vào: Cung cấp giá trị cho hàm.

□ Tham số ra: Lưu kết quả tính toán được trong hàm.

□ Tham số vừa vào, vừa ra: vừa cung cấp giá trị cho hàm, vừa lưu kết quả tính toán được trong hàm.

Khi viết một hàm ta phải xác định xem hàm có bao nhiêu tham số ? (bao nhiêu *tham số vào* và bao nhiêu *tham số ra*).

Đồng thời qua hai kiểu phân loại, chúng ta có nhận xét sau:

- □ Các *tham số ra* phải là *tham chiếu (dạng con trỏ)*.
- □ Các *tham số vào* mà không muốn giá trị của nó bị thay đổi khi hàm kết thúc thì *không được là con trỏ (phải là tham trị)*. (xem ví dụ ở (mục 4)).

□ **Nội dung của hàm**

□ Cách phân chia nhiều hàm dựa theo chức năng

Với thân hàm ta không nên viết một hàm có nội dung quá lớn mà nên chia ra làm nhiều hàm để nội dung được rõ ràng và giảm độ phức tạp của hàm.

□ Vị trí khai báo nội dung hàm

Ta có thể khai báo nội dung của một hàm ở bất cứ vị trí nào và các hàm phải độc lập, nhưng để dễ kiểm tra ta nên khai báo nội dung của các hàm sau nội dung của hàm main(). Trường hợp các hàm không có nguyên mẫu thì phải được khai báo trước hàm main().

4. HÀM VÀ CON TRỎ

4.1. Hàm có đối con trỏ (tham chiếu)

Nếu đối của hàm là con trỏ kiểu *int* (*float*, *double*, ...) thì tham số thực tương ứng phải là địa chỉ của biến kiểu *int* (*float*, *double*, ...). Khi đó địa chỉ của biến được truyền cho đối con trỏ tương ứng. Do đã biết địa chỉ của biến, nên ta có thể gán cho nó một giá trị mới bằng cách sử dụng các câu lệnh thích hợp trong thân hàm.

Ví dụ minh họa

Chương trình hoán vị giá trị của hai biến

```
#include <stdio.h>
#include <conio.h>
int a,b;
void swap(int a, int b);
main () /* Hàm chính */
{
    clrscr();
```

a,b là hai tham số vừa vào vừa ra, nhưng khi khai báo không sử dụng kiểu con trỏ nên khi gọi thực hiện hàm này các biến mà ta truyền cho nó sẽ không thay đổi khi hàm kết thúc.

```
    a=3; b=7;
    printf("\n Truoc khi gọi ham: ")
    printf("A= %d ",a);
    printf("B= %d ",b);
    swap(a,b);
    printf("\nSau khi gọi ham: A = %d B= %d \n",a,b);
    getch();
}
void swap(int a, int b)
{
    int temp ;
    temp=a;
    a=b;
```

```

    b=temp;
    printf("\nTrong ham swap: A= %d B= %d ",a,b);
}

```

Kết quả thu được:

Truoc khi gọi ham: A=3 B=7

Trong ham swap: A= 7 B= 3

Sau khi gọi ham: A = 3 B= 7

Như vậy hàm trên không đạt được yêu cầu đặt ra. Để giải quyết bài toán trên, ta viết lại chương trình như sau:

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

```
#include <conio.h>
```

```
int a,b;
```

```
void swap(int *a, int *b);
```

```
main () /* Ham chinh */
```

```
{
```

```
clrscr();
```

```
a=3; b=7;
```

```
printf("\n Truoc khi gọi ham: ")
```

```
printf("A= %d ",a);
```

```
printf("B= %d ",b);
```

```
swap(&a,&b);
```

```
printf("Sau khi gọi ham: A = %d B= %d \n",a,b);
```

```
getch();
```

```
}
```

```
void swap(int *a, int *b)
```

```
{
```

```
int temp ;
```

```
temp=*a;
```

```
*a=*b;
```

```
*b=temp;
```

```
printf("\nTrong ham swap A= %d B= %d \n",*a,*b);
```

```
}
```

a,b là hai tham số vừa vào vừa ra, và sử dụng kiểu con trỏ

Khi tham số hình thức là con trỏ thì tham số thực phải là con trỏ (dạng địa chỉ)

Kết quả thu được:

Truoc khi gọi ham: A=3 B=7

Trong ham swap: A= 7 B= 3

Sau khi gọi ham: A = 7 B= 3

Như vậy, kết quả hoàn toàn có thể chấp nhận được.

4.1. 4.2. Khi nào sử dụng đối con trỏ (tham chiếu)

Như đã nói trong mục 3: Khi muốn bảo lưu lại kết quả tính toán được của các đối số trong hàm để sử dụng cho chương trình gọi hàm có đối số thì chúng ta phải khai báo đối số của hàm là *tham chiếu* (con trỏ hay dạng địa chỉ).

5. 5. BÀI TẬP MINH HỌA

Bài 1: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
#include<conio.h>
void doi(int *a,int b);
main()
{
    int x,y;
    clrscr();
    doi(&x,y=2);
    printf("%d %d",x,y);
    getch();
}
void doi(int *a,int b)
{
    *a=b;
    *a+=b++;
}
```

Kết quả in ra:

4 2

Bài 2: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
int x=2;
int swap(int x,int y);
main()
{
    int y=x=1;
    swap(x,y);
    printf("%d %d",x,y);
}
int swap(int x,int y)
{
    int tam;
    tam=x;
    x=y;
    y=tam;
}
```

Kết quả in ra:

1 1

Bài 3: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
#include<conio.h>
void doi(int *a);
main()
{
    int x=0,y=1;
    clrscr();
    while(x<=y)
        doi(&x);
    printf("%d %d",y,x);
    getch();
}
```

Kết quả in ra

1 2

Bài 4: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
int doitri(int *a);
main()
{
    int x=1,y=2;
    x=doitri(&x);
    printf("%d %d",x,y);
}
int doitri(int *a)
{
    *a-=1;
    *a=++y;
    y-=2;
    return y;
}
```

Kết quả :

Chương trình báo lỗi syntax

Bài 5: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
int y=2;
```

```

int doitri(int *a);
main()
{
    int x=1;
    x=doitri(&x);
    printf("%d %d",x,y);
}
int doitri(int *a)
{
    *a-=1;
    *a=++y;
    y-=2;
    return y;
}

```

Kết quả in ra:
1 1

Bài 6: Cho đoạn chương trình sau. Xác định kết quả in ra

```

#include<stdio.h>
#include<conio.h>
int doi(int *a,int *b);
main()
{
    int x=1,y=2;
    clrscr();
    doi(&y,&y);
    printf("%d %d",x,y);
    getch();
}
int doi(int *a,int *b)
{
    *a-=1;
    *a=++(*b);
    *b-=2;
}

```

Kết quả in ra:
1 0

Bài 7: Cho đoạn chương trình sau. Xác định kết quả in ra

```

#include<stdio.h>
#include<conio.h>
void ham1(void);
void ham2(int);
void main()
{
    clrscr();

```

```

ham1();
ham2(3);
return;
}
void ham1(void)
{
printf("Bai tap lap trinh C\n");
}
void ham2(int n)
{
int I;
for(I=0;I<n;I++)
printf(" kinh chao cac ban\n");
}

```

Kết quả in ra:

```

Bai tap lap trinh C
  kinh chao cac ban
  kinh chao cac ban
  kinh chao cac ban

```

Bài 8: Cho đoạn chương trình sau. Xác định kết quả in ra

```

#include<stdio.h>
int a,b,c,d;
void confuse(int *,int);
void main()
{
a=1;b=2;c=3;d=4;
confuse(&b,c);
printf("\na=%d b=%d c=%d d=%d",a,b,c,d);
confuse(&b,a);
printf("\na=%d b=%d c=%d d=%d",a,b,c,d);
}
void confuse(int *a,int b)
{
int c;
*a=5;b=10;c=4;
printf("\na=%d b=%d c=%d d=%d",*a,b,c,d);
}

```

Kết quả in ra:

```

a=5 b=10 c=4 d=4
a=1 b=5 c=3 d=4
a=5 b=10 c=4 d=4
a=1 b=5 c=3 d=4

```

-
-

Bài 9: Viết chương trình dùng một hàm nhận hai đối số thực và một ký hiệu ứng với một trong bốn phép tính +, -, *, /, hàm trả về kết quả tính toán.

```
/*Chương trình sử dụng hàm minh họa bốn phép toán +, -, *, / */
#include<stdio.h>
float sohoc(float,float,char);
void main()
{
    float x,y;
    printf("\n x="); scanf("%f",&x);
    printf("\n y="); scanf("%f",&y);
    printf("\nTong la:%f",sohoc(x,y,'+'));
    printf("\nHieu la:%f",sohoc(x,y,'-'));
    printf("\nTich la:%f",sohoc(x,y,'*'));
    printf("\nThuong la:%f",sohoc(x,y,'/'));
}
float sohoc(float v1,float v2,char tinh)
{
    float ketqua;
    switch(tinh)
    {
        case '+':ketqua=v1+v2; break;
        case '-':ketqua=v1-v2; break;
        case '*':ketqua=v1*v2; break;
        case '/':ketqua=v1/v2; break;
    }
    return ketqua;
}
```

Kết quả :

```
x=3
y=4
Tong la:7.000000
Hieu la:-1.000000
Tich la:12.000000
Thuong la:0.750000
```

6. 6. CÁC BÀI TẬP TỰ LÀM

Bài 1: Cho đoạn chương trình sau. Chọn kết quả đúng

```
#include<stdio.h>
#include<conio.h>
int doi(int *a);
main()
{
    int x=0;
    clrscr();
    while(x< doi(&x))
    printf("%d ", x);
    getch();
}
```

```
int doi(int *a)
{
    static int tam=-1;
    *a+=tam;
    return tam++;
}
```

Kết quả in ra là:

- a. a. -1
- b. b. Chương trình sai cú pháp
- c. c. 0
- d. d. Chương trình lặp vô tận

Bài 2: Cho khai báo biến sau

```
int *pint;
float a;
char c;
double *pd;
Hãy chọn phát biểu sai cú pháp:
```

- e. a. a=*pint;
- f. b. c=*pd;
- g. c. *pint= *pd;
- h. d. pd=a;

Bài 3: Cho đoạn chương trình sau. Xác định kết quả in ra

```
#include<stdio.h>
int swap(char a,char b);
main()
{
    int x='a',y=256;
    swap(x,y);
}
int swap(char a,char b)
{
    int tam;
    tam=a;
    a=b;
    b=tam;
    printf("%d %d",a,b);
}
```

Kết quả in ra:

- a. a. Cả 3 câu đều sai
- b. b. 256 97
- c. c. Chương trình sai cú pháp
- d. d. 97 256

Bài 4: Giải phương trình bậc hai:

- - Xây dựng hàm nhập các hệ số a,b,c.
- - Xây dựng hàm để tính nghiệm số.
- - Xây dựng hàm để in kết quả.

Bài 5: Viết chương trình xuất ra các hình chữ nhật lớn dần trên màn hình tới tối đa, rồi sau đó giảm dần.

Gợi ý: Xây dựng một hàm xuất một hình chữ nhật với các tham số là tọa độ, chiều dài và chiều rộng.

Bài 6: Viết chương trình in ra bàn cờ ca rô trên màn hình.

Gợi ý: Sử dụng hàm sau:

```
void in_l_hang(int x, int y, int rong, int so_o, char c1, char c2, char c3, char c4)
{
    int i,j;
    printf("%c",c1);
    for (i=0;i<so_o;i++)
    {
        for (j=0;j<rong-1;j++)
        { printf("%c",c2); }
        printf("%c",c3);
    }
    printf("%c",c4);}
```

Bài 7: Chuyển các chương trình trong bài 7, bài 8, bài 9 thuộc phần *các bài tập tự làm trong Bài 4 Mục 10* sang dạng có sử dụng hàm.

BÀI 7

MẢNG

2. 1. KHÁI NIỆM

Là một tập hợp nhiều biến có cùng kiểu dữ liệu và cùng tên, khi đó mỗi phần tử của mảng được truy xuất thông qua chỉ số.

2. 2. CÁCH KHAI BÁO

2.1. Để khai báo mảng ta dùng cú pháp sau

<kiểu dữ liệu> <tên mảng> <Danh sách các chiều của mảng>;

2.2. Các ví dụ

Ví dụ 1:

`int a[10], b[3][2];`

Dòng lệnh trên khai báo hai mảng, mảng a là mảng một chiều có 10 phần tử thuộc kiểu int, mảng b là mảng hai chiều (3 dòng, 2 cột) có 6 phần tử thuộc kiểu int.

Ví dụ 2:

`float c[100], d[5][7];`

Dòng lệnh trên khai báo hai mảng, mảng c là mảng một chiều có 100 phần tử thuộc kiểu số thực float, mảng d là mảng hai chiều (5 dòng, 7 cột) có 35 phần tử thuộc kiểu float.

3. 3. CHỈ SỐ CỦA MẢNG

3.1. Kiểu dữ liệu của chỉ số

Chỉ số của mảng phải là một giá trị kiểu int không vượt qua kích thước của mảng, chỉ số của mảng bắt đầu từ 0.

3.2. Ví dụ

Ví dụ: `int a[5];`

*/*Gồm 5 phần tử tương ứng các chỉ số từ 0...4 (`a[0], a[1], a[2], a[3], a[4]`) */*

4. 4. LẤY ĐỊA CHỈ CỦA PHẦN TỬ MẢNG

4.1. Cú pháp

Chỉ lấy được địa chỉ của các phần tử thuộc mảng một chiều thông qua toán tử & theo cú pháp: `&tên_biến[i]` (i là chỉ số của mảng).

4.2. Các điểm cần lưu ý

*Chú ý: Tên của mảng sẽ chứa địa chỉ đầu của mảng,
ví dụ: có `int a[10]` thì `a=&a[0]`*

5. NHẬP XUẤT DỮ LIỆU CHO CÁC PHẦN TỬ MẢNG

5.1. Nhập xuất trực tiếp

□ Phạm vi ứng dụng

chỉ sử dụng được với mảng một chiều và mảng hai chiều có các phần tử kiểu int.

□ Ví dụ

Ví dụ: nhập xuất dữ liệu cho một mảng một chiều (kiểu int).

```
# include <stdio.h>
# include <conio.h>
void main () /* Ham chinh */
{
    int a[5];
    int i;
    clrscr();
    /*Nhập dữ liệu*/
    for (i=0;i<5;i++)
    {printf("\na[%d]",i);
      scanf("%d",&a[i]); /*nhập trực tiếp bằng phép lấy địa chỉ*/
    }
    /*Đưa các kết quả ra màn hình*/
    for (i=0;i<5;i++)
        printf("%d ",a[i]);
    getch();
}
```

5.2. Nhập xuất gián tiếp

□ Phạm vi ứng dụng

Nhập gián tiếp, sử dụng tốt với cả mảng một chiều, cũng như đa chiều.

□ Ví dụ

Ví dụ: Nhập, xuất dữ liệu cho mảng hai chiều (kiểu số thực(float)).

```
# include <stdio.h>
# include <conio.h>
void main () /* Ham chinh */
{
    float temp,a[3][3];
    int i,j;
    clrscr();
    /*Nhập dữ liệu */
    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            {printf("\na[%d][%d]",i,j);
              scanf("%f",&temp); /*Nhập gián tiếp thông qua biến temp*/
              a[i][j]=temp;      /*Gán giá trị của temp cho phần tử mảng*/
            }
}
```

```

    }
    /*Đưa giá trị các phần tử ra màn hình*/
    for (i=0;i<3;i++)
    {
        printf("\n");
        for (j=0;j<3;j++)
            printf("%.2f ",a[i][j]);
    }
    getch();
}

```

6. MỘT SỐ VẤN ĐỀ LIÊN QUAN

6.1. Biến, mảng nội

☐ Khái niệm

Là các biến (mảng) khai báo bên trong thân của một hàm.

☐ Thời gian tồn tại

Từ lúc máy bắt đầu làm việc với hàm cho đến khi hàm đó kết thúc. Vậy các biến, mảng nội khai báo trong hàm main tồn tại trong suốt thời gian làm việc của chương trình.

☐ Phạm vi sử dụng

Các biến (mảng) nội của một hàm chỉ có tác dụng bên trong hàm mà nó được khai báo.

☐ Quy tắc khởi đầu khi khai báo mảng nội

Muốn khởi đầu cho một mảng nội, *phải sử dụng toán tử gán*

☐ Ví dụ minh họa

Ví dụ 1

```

/*Minh họa mảng số một chiều, khởi đầu cho một mảng nội */
#include <stdio.h>
void main()
{
    int a[3]={ 10,20,30}; /*Khởi đầu mảng một chiều với 3 phần tử, nằm trong thân hàm main()*/
    /*Đưa các kết quả ra màn hình*/
    printf("Nội dung của a[0] => %d\n",a[0]);
    printf("Nội dung của a[1] => %d\n",a[1]);
    printf("Nội dung của a[2] => %d\n",a[2]);
}

```

Kết quả:

```

Nội dung của a[0] =>10
Nội dung của a[1] =>20
Nội dung của a[2] =>30

```

Ví dụ 2

```
/* Chương trình minh họa mảng hai chiều, khởi đầu cho một mảng nội */
#include <stdio.h>
void main()
{
    static int a[2][3] = {
                                { 10, 20, 30},
                                { 11, 21, 31}
    }; /*Khởi đầu mảng hai chiều với 2 hàng, 3 cột trong thân hàm
main()*/
    int hang;
    int cot;
    /*Đưa các kết quả ra màn hình*/
    for(hang = 0; hang < 2; hang++)
    {
        for(cot = 0; cot < 3; cot++)
            printf("%5d",a[hang][cot]);
        printf("\n\n");
    }
}
```

Kết quả:

10	20	30
11	21	31

Biến, mảng nội chưa khởi đầu thì giá trị của nó hoàn toàn không xác định.

6.2. Biến, mảng ngoại

☐ Khái niệm

Là các biến (mảng) khai báo bên ngoài các hàm.

☐ Thời gian tồn tại

Tồn tại trong suốt thời gian làm việc của chương trình.

☐ Phạm vi sử dụng

Phạm vi hoạt động của các biến (mảng) ngoại là từ vị trí nó được khai báo cho đến cuối chương trình.

☐ Các qui tắc khởi đầu khi khai báo mảng ngoại

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

Ví dụ

```
char sao='*';  
int a=6*45;  
long b 24*3*2467;  
float x=32.5;  
float y[6]={3.2,0,5.1,23,0,41};  
int z[3][2]={  
                {25,31},  
                {46,54},  
                {93,81}  
            };  
  
void main()  
{  
    ...  
}
```

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

Ví dụ

```
float a[]={2,5.1,15};  
int t[][4]={  
            {5,6,7,8},  
            {1,12,15,6}  
        };  
};
```

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

Ví dụ

```
float bt[8]={6.2,5.1,15};  
int h[6][4]={  
            {5,6,7,8},  
            {1,12,15,6},  
            {0,2,5,9}  
        };  
};
```

4. Đối với mảng hai chiều có thể khởi đầu theo hai cách sau (số giá trị khởi đầu trên mỗi mảng có thể khác nhau):

```
float a[][3]={  
            {5},  
            {1.2,2,3.6},  
            {-6.9}  
        };  
hoặc :  
int h[10][4]={  
            {5},
```

```

        {1,12,15,6},
        {9,10}
        {1,2,3,4}
    };

```

5. Bộ khởi đầu của một mảng char có thể:
- ☐ ☐ Hoặc là danh sách các hằng kí tự;
 - ☐ ☐ Hoặc là một hằng xâu kí tự.

Ví dụ

```

char name[]={'h','a','n','g','\0'};
char name[]="hang";
char name[10]={'h','a','n','g','\0'};
char name[10]="hang";

```

☐ **Chương trình minh họa**

```

/*Chương trình minh họa qui tắc khởi đầu biến mảng ngoại*/
#include <stdio.h>
int a=41, t[][3]={
    {25,30,40},
    {145,83,10}
}; /*Khởi đầu biến a và mảng hai chiều t kiểu int*/
float y[8]={-45.8,32.5}; /*Khởi đầu mảng một chiều y kiểu float*/
float x[10][2]={
    {-125.3,48.9},
    {145.6,83.5}
}; /*Khởi đầu mảng hai chiều x kiểu float*/
char n1[]={'T','h','u','\0'};
char n2[]="Thu";
char n3[10]={'T','h','u','\0'};
char n4[10]="Thu"; /*Khởi đầu một mảng char theo bốn cách*/
void main()
{
    /*In các kết quả ra màn hình*/
    printf("\na=%6d,t(1,2)=%6d,t(1,1)=%6d",a,t[1][2],t[1][1]);
    printf("\ny(0)=%8.2f, y(1)=%8.2f",y[0],y[1]);
    printf("\nx(1,1)=%8.2f, x(2,0)=%8.2f",x[1][1],x[2][0] );
    printf("\n\n%8s%8s%8s%8s",n1,n2,n3,n4 );
}

```

Kết quả thực hiện chương trình

```

a=41, t(1,2)=10,t(1,1)=83
y(0)= -45.8, y(1)= 32.5
x(1,1)=83.50, x(0,2)=0.00

```

```

Thu  Thu  Thu  Thu

```

7. CON TRỎ VÀ MẢNG

7.1. Nhắc lại một số khái niệm liên quan đến con trỏ

Bài trước, chúng ta đã đề cập đến khái niệm con trỏ, phần này sẽ tìm hiểu chi tiết hơn một số vấn đề liên quan:

□ **Kiểu con trỏ và kiểu địa chỉ**

Con trỏ dùng để lưu trữ địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng.

Phép gán địa chỉ cho con trỏ chỉ diễn ra suôn sẻ khi kiểu địa chỉ phù hợp với kiểu con trỏ.

Ví dụ:

theo khai báo:

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

thì

- □ *pa là con trỏ kiểu float,*
- □ *pm là con trỏ kiểu float[30],*
- □ *a là địa chỉ kiểu float[30].*

Vì vậy phép gán

pa=a;

là sai. Nhưng phép gán

pm=a;

thì hoàn toàn hợp lệ.

□ **Các toán tử một ngôi (Toán tử & và toán tử *)**

□ **Toán tử một ngôi &**

Toán tử một ngôi & cho ta địa chỉ của một đối tượng; như vậy câu lệnh

```
p=&c; /* c là biến kí tự */
```

gán địa chỉ của biến ký tự c cho con trỏ p, và chúng ta nói rằng p "chỉ đến" c. Toán tử & chỉ áp dụng được cho các đối tượng trong bộ nhớ: đó là các biến và các phần tử của mảng. Toán tử này không thể áp dụng cho các biểu thức, các hằng và các biến có kiểu register (các biến có kiểu register chứa trong các thanh ghi trong CPU để tăng tốc độ truy nhập).

□ □ **Toán tử một ngôi ***

Toán tử một ngôi * cho ta nội dung của một đối tượng con trỏ. Toán tử này cho phép truy nhập đến các đối tượng được trỏ bởi các con trỏ. Giả sử rằng x và y là hai biến số nguyên và pi là biến con trỏ int. Dãy các câu lệnh sau minh họa cách khai báo con trỏ và sử dụng các toán tử & và *:

```
int x=1,y=2,z[10];
```

```
int *pi; /*pi là một biến con trỏ có kiểu nguyên*/
```

```
pi = &x; /*Địa chỉ của x được gán cho pi, và pi trỏ tới biến x*/
```

```
y=*pi; /*y có giá trị bằng 1*/
```

```
*pi=0; /*Từ bây giờ x có giá trị bằng 0*/
```

*pi=&z[0]; /*Từ đây pi chứa địa chỉ của z[0], tức là địa chỉ của mảng z*/*

Chúng ta đã nói về cách khai báo các biến x,y và z. Khai báo con trỏ pi,

*int *pi;*

chỉ ra rằng biểu thức *pi là một số nguyên.

Nếu pi trỏ tới biến số nguyên x, chúng ta có thể viết *pi thay cho x ở tất cả mọi nơi có x xuất hiện; vì vậy

**pi = *pi+10;*

thêm 10 vào *pi (nghĩa là x).

Các toán tử một ngôi * và & có độ ưu tiên cao hơn các toán tử số học; vì vậy, phép gán

*y = *pi + 10;*

lấy giá trị của đối tượng được trỏ bởi pi, cộng thêm 10 và ghi kết quả trong y, còn

**pi += 1;*

tăng giá trị của đối tượng được trỏ bởi pi thêm 1. Phép toán này hoàn toàn tương đương với

*++*pi; (*pi)++;*

các dấu ngoặc cần phải có trong ví dụ cuối cùng; nếu không có, chúng ta tăng giá trị của pi chứ không phải giá trị của đối tượng mà pi trỏ tới. Điều này là do các toán tử * và ++ có cùng mức ưu tiên và được thực hiện từ phải qua trái.

Cuối cùng, vì con trỏ cũng là một biến, do đó chúng ta có thể sử dụng bản thân tên các biến con trỏ trong các câu lệnh.

□ Các phép toán trên con trỏ

Có 4 nhóm phép toán liên quan đến con trỏ và địa chỉ:

□ **Phép gán**

Chỉ nên thực hiện phép gán cho các con trỏ cùng kiểu

Xét ví dụ sau:

*int x=1, *pi, *qi;*

pi = &x;

qi=pi;

câu lệnh thứ ba sao chép nội dung của pi vào trong qi, nhờ vậy mà pi và qi trỏ đến cùng một đối tượng (ở đây là biến x).

Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu:

Ví dụ

int x;

*char *pc;*

pc=(char) (&x); /*ép kiểu*/*

□ **Phép tăng giảm địa chỉ**

Ví dụ

*float x[30], *px;*

$px = \&x[10];$
 cho biết px là con trỏ float trỏ đến phần tử $x[10]$.
 $px+i$ trỏ đến phần tử $x[10+i]$
 $px-i$ trỏ đến phần tử $x[10-i]$.

□ Nguyên tắc truy nhập bộ nhớ

Con trỏ float truy nhập tới 4 byte, con trỏ int truy nhập 2 byte, con trỏ char truy nhập một byte.

Ví dụ
 $float *pf;$
 $int *pi;$
 $char *pc;$

Khi đó:

- Nếu pf trỏ đến byte thứ 10001, thì $*pf$ biểu thị vùng nhớ 4 byte liên tiếp từ byte 10001 đến byte 10004.
- Nếu pi trỏ đến byte thứ 10001, thì $*pi$ biểu thị vùng nhớ 2 byte liên tiếp từ byte 10001 đến byte 10002.
- Nếu pc trỏ đến byte thứ 10001, thì $*pc$ biểu thị vùng nhớ 1 byte là byte 10001.

Chú ý: Hai phép toán trên không dùng được cho con trỏ kiểu void.

□ Phép so sánh

Cho phép so sánh các con trỏ cùng kiểu, ví dụ nếu $p1$ và $p2$ là 2 con trỏ float thì:

- $p1 < p2$ nếu địa chỉ $p1$ trỏ tới thấp hơn địa chỉ $p2$ trỏ tới,
- $p1 = p2$ nếu địa chỉ $p1$ trỏ tới bằng địa chỉ $p2$ trỏ tới,
- $p1 > p2$ nếu địa chỉ $p1$ trỏ tới cao hơn địa chỉ $p2$ trỏ tới.

7.2. Con trỏ và mảng một chiều

Khi ta khai báo một mảng thì tên của mảng là một hằng địa chỉ, chứa địa chỉ của phần tử đầu tiên.

Phân tích ví dụ

$float a[10]$ thì $a = \&a[0]$ và $a+i = \&(a[i])$ với i là một số nguyên. Vậy để truy xuất đến các phần tử của mảng ta có thể dùng chỉ số hoặc dùng con trỏ.

Nếu ta có con trỏ p và p trỏ vào phần tử thứ k của mảng a thì $p+i$ sẽ trỏ đến phần tử thứ $k+i$ của mảng a

*Ví dụ: $float a[10], *p, *q;$*

...
 $p = a; /* p trỏ vào phần tử 0 */$
 $q = p + 5; /* q trỏ vào phần tử thứ 5 */$

Khi đó các cách viết sau là tương đương :

$p+i = \&a[i], q = \&a[5]$
 $a[i] = *(a+i) = *(p+i) = p[i]$

Lưu ý : Ta có thể gán giá trị cho p,q nhưng không thể gán giá trị mới cho a vì nó là một hằng địa chỉ.

Các ví dụ minh họa con trỏ trong mảng số một chiều:

Ví dụ 1:

```
/*Con trỏ và mảng một chiều*/
```

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

```
main()
```

```
{
```

```
    int a[3]={ 10,20,30};
```

```
    int *ptr;
```

```
    ptr = a;
```

```
    printf("Noi dung cua a[0] => %d\n",*ptr);
```

```
    printf("Noi dung cua a[1] => %d\n",*(ptr + 1));
```

```
    printf("Noi dung cua a[2] => %d\n",*(ptr + 2));
```

```
}
```

Kết quả:

Noi dung cua a[0] =>10

Noi dung cua a[1] =>20

Noi dung cua a[2] =>30

Ví dụ 2

```
/*Minh họa việc nhập dữ liệu vào mảng bằng con trỏ*/
```

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

```
# include <conio.h>
```

```
main () /* Ham chinh */
```

```
{
```

```
    int a[5],*p;
```

```
    int i;
```

```
    clrscr();
```

```
    p=a;
```

```
    for (i=0;i<5;i++)
```

```
    {printf("\na[%d]",i);
```

```
    scanf("%d",p+i); /* nhập vào địa chỉ &a[i] */
```

```
}
```

```
    for (i=0;i<5;i++)
```

```
    printf("%d ",*(a+i)); /* *(a+i) tương đương *(p+i) tương đương a[i]*/
```

```
    getch();
```

```
}
```

Kết quả

a[0] = 10;

a[1] = 20;

a[2] = 30;

a[3] = 40;

a[4] = 50;

10 20 30 40 50

7.3. Con trỏ và mảng nhiều chiều

Việc xử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều. Không phải mọi qui tắc đúng đối với mảng một chiều đều có thể đem ra áp dụng đối với mảng nhiều chiều.

- Phép toán lấy địa chỉ nói chung không dùng được đối với các thành phần của mảng nhiều chiều (trừ trường hợp mảng hai chiều các số nguyên). Xét chương trình sau với ý định nhập số liệu cho ma trận thực.

```
#include <stdio.h>
void main()
{
    float a[10][20];
    int i,j,n;
    printf("Nhập vào kích thước ma trận n=");
    scanf("%n",&n); /*Chương trình chạy đúng cho đến đây*/
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            printf("a[%d][%d] = ",i,j);
            scanf("%f",&a[i][j]);
        }
}
```

chương trình chạy sai vì phép toán $\&a[i][j]$ với a là mảng hai chiều các số thực là không hợp lệ. Để tính toán địa chỉ của thành phần $a[i][j]$ chúng ta sử dụng công thức sau $(\text{float} *)a+i*n+j$. Chương trình đúng được viết lại như sau:

```
#include <stdio.h>
void main()
{
    float a[10][20];
    int i,j,n;
    printf("Nhập vào kích thước ma trận n=");
    scanf("%n",&n);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            printf("a[%d][%d] = ",i,j);
            scanf("%f",(\text{float} *)a+i*20+j);
        }
}
```

Các bạn để ý rằng, a là một hằng con trỏ trỏ đến các dòng của một ma trận hai chiều, vì vậy

- a trỏ đến dòng thứ nhất
- $a+1$ trỏ đến dòng thứ hai
- $a+2$ trỏ đến dòng thứ ba

.v.v.

Để tính toán được địa chỉ của phần tử ở dòng i cột j chúng ta phải dùng phép chuyển đổi kiểu bắt buộc đối với a : $(\text{float } *)a$, đây là con trỏ trỏ đến thành phần $a[0][0]$ của ma trận. Và vì vậy thành phần $a[i][j]$ sẽ có địa chỉ là $(\text{float } *a) + i*n + j$. Một cách tổng quát, nếu mảng có kiểu type và có kích thước các chiều tương ứng là n_1, n_2, \dots, n_k (Giả sử mảng a có k chiều). Địa chỉ của thành phần $a[0]..[0]$ (k chỉ số 0) là $(\text{type } *)a$, và địa chỉ của $a[i_1][i_2]..[i_k]$ được tính như sau

8. HÀM, CON TRỎ VÀ MẢNG

8.1. Hàm và mảng một chiều

Nếu tham số thực là tên mảng a (một chiều) kiểu int (float , double ,...) thì đối pa tương ứng cần phải là một con trỏ kiểu int (float , double ,...).

Đối pa có thể khai báo theo hai cách:

Cách 1: Kiểu con trỏ

```
int *pa;  
float *pa;  
double *pa;
```

...

Cách 2: có thể khai báo như một mảng hình thức:

```
int pa[];  
float pa[];  
double pa[];
```

...

Hai cách khai báo trên là tương đương.

Khi hàm bắt đầu làm việc thì giá trị của a được truyền cho pa . Vì a là hằng địa chỉ biểu diễn địa chỉ đầu của mảng, nên con trỏ pa chứa địa chỉ phần tử đầu tiên của mảng. Như vậy khi muốn truy nhập đến phần tử $a[i]$ trong thân hàm có thể dùng một trong hai cách viết sau:

$*(pa+i)$ và $pa[i]$

Ví dụ:

```
/*Minh họa hàm và mảng số một chiều*/  
#include <stdio.h>  
/*Dưới đây sẽ khai báo hàm và đối của hàm */  
*/  
void ham1(int pa[]);  
void ham2(int *pa);  
void ham3(int pa[]);  
void ham4(int *pa);  
void main()  
{  
    int a [3]={10,20,30};  
    printf ("\\n Ket qua thuc hien ham 1:\\n");  
    ham1(a);/*Gọi ham1*/  
    printf ("\\n Ket qua thuc hien ham 2:\\n");  
    ham2(a);/*Gọi ham2*/  
}
```

```

        printf ("\n Ket qua thuc hien ham 3:\n");
        ham3(a); /*Goi ham3*/
        printf ("\n Ket qua thuc hien ham 4:\n");
        ham4(a); /*Goi ham4*/
    }

void ham1(int pa[])
{
    printf("Noi dung cua a[0] => %d\n",pa[0]);
    printf("Noi dung cua a[1] => %d\n",pa[1]);
    printf("Noi dung cua a[2] => %d\n",pa[2]);
}
void ham2(int *pa)
{
    printf("Noi dung cua a[0] => %d\n",pa[0]);
    printf("Noi dung cua a[1] => %d\n",pa[1]);
    printf("Noi dung cua a[2] => %d\n",pa[2]);
}
void ham3(int pa[])
{
    printf("Noi dung cua a[0] => %d\n",*(pa));
    printf("Noi dung cua a[1] => %d\n",*(pa+1));
    printf("Noi dung cua a[2] => %d\n",*(pa+2));
}
void ham4(int *pa)
{
    printf("Noi dung cua a[0] => %d\n",* (pa));
    printf("Noi dung cua a[1] => %d\n", *(pa+1));
    printf("Noi dung cua a[2] => %d\n", *(pa+2));
}

```

Kết quả:

Ket qua thuc hien ham 1:

Noi dung cua a[0] =>10

Noi dung cua a[1] =>20

Noi dung cua a[2] =>30

Ket qua thuc hien ham 2:

Noi dung cua a[0] =>10

Noi dung cua a[1] =>20

Noi dung cua a[2] =>30

Ket qua thuc hien ham 3:

Noi dung cua a[0] =>10

Noi dung cua a[1] =>20

Noi dung cua a[2] =>30

Ket qua thuc hien ham 4:

Noi dung cua a[0] =>10

Noi dung cua a[1] =>20

Noi dung cua a[2] =>30

Như vậy kết quả thực hiện của bốn hàm là giống nhau.

8.2. Hàm và mảng đa chiều

Nếu tham số thực là tên mảng hai chiều a.

Giả sử a là mảng hai chiều:

```
float a[50][30];
```

Làm thế nào để có thể dùng tên mảng hai chiều a trong lời gọi hàm.

Có hai cách:

Cách 1:

Dùng đối con trỏ kiểu float, khai báo theo một trong hai mẫu sau:

```
float (*pa)[30];
```

```
float pa[][30];
```

Để truy nhập đến phần tử a[i][j] trong thân hàm, dùng:

```
pa[i][j] /*Với cách này hạn chế số cột của mảng hai chiều*/
```

Cách 2:

Dùng hai đối:

```
float *pa; /* biểu thị địa chỉ đầu của mảng a */
```

```
int N; /* biểu thị số cột của mảng a */
```

Để truy nhập đến phần tử a[i][j] trong thân hàm, dùng công thức:

```
*(pa + i*N + j)
```

Theo cách này mảng hai chiều được qui về như mảng một chiều. Việc xử lý phức tạp hơn so với cách nhưng không hạn chế số cột nên có thể dùng cho bất kỳ mảng hai chiều nào.

Ví dụ minh họa

```
/*Hàm và mảng hai chiều giải bài toán tính tổng cột đầu tiên của ma trận*/
```

```
#include <stdio.h>
int cong_cot(int pa[][3]); /*Khai báo hàm và đối của hàm */
main()
{
    static int a[2][3] = { { 10,20,30}, { 11,21,31} };

    int hang;
    int cot;
    int tong_cot_dau;

    for(hang = 0; hang < 2; hang++)
    {
        for(cot = 0; cot < 3; cot++)
            printf("%5d", a[hang][cot]);

        printf("\n\n");
    }

    tong_cot_dau = cong_cot(a); /*Gọi hàm*/
    printf("Tổng của cot đầu tiên là %d", tong_cot_dau);
}
int cong_cot(int pa[][3])
{
    int hang;
```

```

    int tong_cot;

    tong_cot = 0;

    for(hang = 0; hang < 2; hang++)
        tong_cot += pa[hang][0];

    return(tong_cot);
}

```

Kết quả:

10 20 30

11 21 31

Tong cua cot dau tien la 21

9. CÁC BÀI TẬP MINH HỌA

9.1. Các bài tập về mảng một chiều

Bài 1

*/*Nhập mảng một chiều, tính tổng các phần tử mà không dùng con trỏ */*

#include<stdio.h>

#include<conio.h>

void main()

{

int i;

float m[5],s;

clrscr();

for(i=0;i<5;i++)

{

printf("\n m[%d]=",i); scanf("%f",&m[i]);

}

for(s=0,i=0;i<5;i++)

s+=m[i];

printf("\n Tong=%8.2f",s);

getch();

return;

}

Bài 2

*/*Nhập mảng một chiều, tính tổng các phần tử dùng con trỏ*/*

#include<stdio.h>

#include<conio.h>

void main()

{

int i;

*float m[5],s,*pm;*

clrscr();

pm=m;

for(i=0;i<5;i++)

```

{
    printf("\n m[%d]=",i); scanf("%f",pm+i);
}
for(s=0,i=0;i<5;i++)
    s+=*(pm+i);
printf("\n Tong=%8.2f",s);
getch();
}

```

Bài 3

*/*Nhập và mảng một chiều n phần tử. Kiểm tra mảng có đối xứng không*/*

```

#include<stdio.h>
#include<conio.h>
#define n 10
void main()
{
    int a[n],i;
    clrscr();
    for(i=0;i<n;i++)
    {
        printf("\n a[%d]=",i); scanf("%d",&a[i]);
    }

    for(i=0;(i<n/2)&&(a[i]==a[n-i-1]);i++);
    if (i==n/2) printf("\n Co doi xung");
    else printf("\n Khong doi xung");
    getch();
    return;
}

```

Bài 4

*/*Viết chương trình nhập vào tọa độ hai vectơ 5 chiều. Tính module và tổng hai vectơ đó*/*

Tính module và tổng hai vectơ đó/*

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void nhap(int x[],char chu);
void hienthi(int x[],char chu);
double module(int x[]);
void tong(int x[],int y[]);
void main()
{
    int a[5],b[5];
    clrscr();
    nhap(a,'a');

```

```

    nhap(b,'b');
    hienthi(a,'a');
    hienthi(b,'b');
    tong(a,b);
    printf("\n Module cua vecto a la %.2f",module(a));
    printf("\n Module cua vecto b la %.2f",module(b));
    getch();
    return;
}
/*Nhap du lieu*/
void nhap(int x[],char chu)
{
    int k;
    for(k=0;k<5;k++)
    {
        printf("%c[%d]=",chu,k+1);
        scanf("%d",&x[k]);
    }
    printf("\n");
}
/*Hien thi len man hinh */
void hienthi(int x[],char chu)
{
    printf("\n%c=(%d,%d,%d,%d,%d)",chu,x[0],x[1],x[2],x[3],x[4]);
}
/*Tinh module */
double module(int x[])
{
    int k;
    float s=0;
    for(k=0;k<5;k++)
        s+=x[k]*x[k];
    s=sqrt((double) s);
    return s;
}
/*Tinh tong*/
void tong(int x[],int y[])
{
    int z[5],k;
    for(k=0;k<5;k++)
        z[k]=x[k]+y[k];
    hienthi(z,'c');
}

```

Bài 5

*/*Sắp xếp mảng một chiều n phần tử tăng dần theo phương pháp bubble sort*/*


```

#include<stdio.h>
#include<conio.h>
#define n 10
#define TRUE 1
#define FALSE 0
void bubble(int a[],int m);
void main()
{
    int a[n],i;
    clrscr();
    for(i=0;i<n;i++)
    {
        printf("\na[%d]=",i); scanf("%d",&a[i]);
    }
    bubble(a,n);
    /*Dua ket qua ra man hinh*/
    printf("\n Mang sau khi sap xep:");
    for(i=0;i<n;i++)
    {
        printf("\n%d",a[i]);
    }
}
/*Giai thuat bubble*/
void bubble(int a[],int m)
{
    int trung_gian,j,chay;
    int kiemtra=TRUE;
    for(chay=0;chay<n-1 &&kiemtra==TRUE;chay++)
    {
        kiemtra=FALSE;
        for(j=0;j<n-chay-1;j++)
            if(a[j]>a[j+1])
            {
                kiemtra=TRUE;
                trung_gian=a[j];
                a[j]=a[j+1];
                a[j+1]=trung_gian;
            }
    }
}

```

Bài 6

```

/*Sắp xếp mảng một chiều n phần tử tăng dần theo phương pháp chọn trực tiếp
select sort.*/
#include<stdio.h>
#include<conio.h>
#define n 10

```

```

void selectsort(int a[],int m);
void main()
{
    int a[n],i;
    clrscr();
    for(i=0;i<n;i++)
    {
        printf("\na[%d]=",i); scanf("%d",&a[i]);
    }
    selectsort(a,n);
    /*Dua ket qua ra man hinh*/
    printf("\n Mang sau khi sap xep:");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
/*Giai thuat select sort*/
void selectsort(int a[],int m)
{
    int i,vt,j,phan_tu;
    for(i=n-1;i>0;i--)
    {
        phan_tu=a[0];
        vt=0;
        for(j=1;j<=i;j++)
            if(a[j]>phan_tu)
            {
                phan_tu=a[j];
                vt=j;
            }
        a[vt]=a[i];
        a[i]=phan_tu;
    }
}

```

Bài 7

/*Sắp xếp mảng một chiều n phần tử tăng dần theo phương pháp chèn trực tiếp

```

insert sort.*/
#include<stdio.h>
#include<conio.h>
#define n 10
void insertsort(int a[],int m);
void main()
{
    int a[n],i;

```

```

clrscr();
for(i=0;i<n;i++)
{
printf("\na[%d]=",i); scanf("%d",&a[i]);
}
insertsort(a,n);
/*Dua ket qua ra man hinh*/
printf("\n Mang sau khi sap xep:");
for(i=0;i<n;i++)
{
printf("%d ",a[i]);
}
}
/*Giai thuat insert sort*/
void insertsort(int a[],int m)
{
int i,k,y;
for(k=1;k<n;k++)
{
y=a[k];
for(i=k-1;i>=0&&y<a[i];i--)
a[i+1]=a[i];
a[i+1]=y;
}
}

```

Bài 8

*/*Sắp xếp mảng một chiều n phần tử tăng dần theo phương pháp nhị phân.*/*

```

#include<stdio.h>
#include<conio.h>
#define n 10
void sort(int a[],int m);
void main()
{
int a[n],i;
clrscr();
for(i=0;i<n;i++)
{
printf("\na[%d]=",i); scanf("%d",&a[i]);
}
sort(a,n);
/*Dua ket qua ra man hinh*/
printf("\n Mang sau khi sap xep:");
for(i=0;i<n;i++)
{
printf("%d ",a[i]);
}
}

```

```

}
}
/*Giai thuat sap xep nhi phan*/
void sort(int a[],int m)
{
    int i,j,top,bottom,middle,t;
    for(i=1;i<n;i++)
    {
        t=a[i];
        bottom=0;
        top=i-1;
        while(bottom<=top)
        {
            middle=(bottom+top)/2;
            if(t<a[middle])
                top=middle-1;
            else
                bottom=middle+1;
        }
        for(j=i-1;j>=bottom;j--)
            a[j+1]=a[j];
        a[bottom]=t;
    }
}

```

9.2. Các bài tập về mảng hai chiều

Bài 9

/*Viết chương trình nhập vào mảng hai chiều và tìm phần tử âm đầu tiên trong mảng*/

```

#include<stdio.h>
#include<conio.h>
void main()
{
    float a[3][4]={4,2.5,6.2,-8},{3.2,25,7,0.5},{1.5,-3,0,5}};
    int i,j;
    clrscr();
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if (a[i][j]<0) goto ketqua;
    printf("Mang khong co phan tu am");
    goto ketthuc;
ketqua:printf("\n Phan tu am dau tien la a[%d][%d]=%.2f",i+1,j+1,a[i][j]);
ketthuc:
    getch();
    return;
}

```

Bài 10

*/*Viết chương trình xoay một ma trận 5x5 một góc 90 độ theo chiều kim đồng hồ*/*

```
#include<stdio.h>
#include<conio.h>
#define N 5
void main()
{
float a[N][N],tam;
float b[N][N];
int i,j;
clrscr();
printf("\n Nhập các phần tử của ma trận\n");
for(i=0;i<N;i++)
for(j=0;j<N;j++)
{
printf("\n Phần tử a[%d][%d]=",i,j);
scanf("%f",&tam);
a[i][j]=tam;
}
printf("\n Hien thi ma tran vua nhap vao:\n");
for(i=0;i<N;i++)
{for(j=0;j<N;j++)
printf("%.2f ",a[i][j]);
printf("\n");
}
printf("\n Hien thi ma tran xoay 90 do theo chieu thuan:\n");
for(i=0;i<N;i++)
for(j=0;j<N;j++)
{
b[j][N-i-1]=a[i][j];/*dùng ma trận phụ b */
}
for(i=0;i<N;i++)
{for(j=0;j<N;j++)
printf("%.2f ",b[i][j]);
printf("\n");
}
getch();
return;
}
```

Chú ý: khi nhập mảng hai chiều các số nguyên ≥ 0 với kích thước lớn, ta có thể dùng hàm **random()** thuộc thư viện `#include<stdlib.h>`.

Ví dụ:

```
#include<stdlib.h>
```

```
•••
```

```
main()
```

```
{
```

```

    ...
    randomize(); /* Khởi tạo dãy random */
    for(i=0; i<7; i++)
        for(j=0; j<7; j++)
        {
            printf("\n Phan tu a[%d][%d]=", i, j);
            a[i][j]=random(50); /* nhập các số nguyên dương trong khoảng
[0..50] */
        }
    ...
}

```

10. CÁC BÀI TẬP TỰ LÀM

Bài 1: Viết chương trình nhập vào mảng, hãy xuất ra màn hình:

- Dòng 1: Phần tử âm lớn nhất của mảng.
- Dòng 2: Phần tử dương nhỏ nhất của mảng.
- Dòng 3: Tổng các phần tử có căn bậc hai nguyên.
- Dòng 4: Gồm các số lẻ, tổng cộng có bao nhiêu số lẻ.
- Dòng 5: Gồm các số chẵn, tổng cộng có bao nhiêu số chẵn.
- Dòng 6: Gồm các số nguyên tố.
- Dòng 7: Gồm các số không phải nguyên tố.

Bài 2: Viết chương trình nhập vào một số nhỏ hơn 1000. Trình bày dòng chữ cho biết giá trị của số đó.

Bài 3: Viết chương trình cộng, trừ hai số nguyên có nhiều chữ số (dùng chuỗi).

Bài 4: Viết chương trình nhập vào một mảng hai chiều $m \times n$. Hãy biến đổi dấu của tất cả các số của 1 hàng hoặc một cột sao cho số lần biến đổi là ít nhất để được một ma trận có tổng của mọi con số ở mọi hàng và mọi cột đều lớn hơn hoặc bằng 0.

Bài 5: Viết chương trình sắp xếp một mảng hai chiều $n \times n$ tăng dần theo cột và theo hàng. ($a[0,0] < a[0,1] < a[0,2] \dots < a[0,n] < a[1,0] < \dots < a[1,n] < a[2,0] < \dots < a[n,n]$).

Gợi ý:

Cách 1: Chuyển mảng hai chiều thành mảng một chiều, sau đó sắp xếp mảng một chiều và cuối cùng chuyển mảng một chiều thành hai chiều.

```

    ...
    void chuyen_2_1(int a[][n], int b[], int n)
    {
        int i, j;
        for(i=0; i<n; i++)
            for(j=0; j<n; j++)
                b[i*n+j]=a[i][j];
        return;
    }

```

• • •

Cách 2: Không dùng mảng 1 chiều (sử dụng phép / và %).

Bài 6

Viết chương trình xoay một ma trận 5x5 một góc 90 độ theo chiều kim đồng hồ, không dùng ma trận phụ b.

-

Bài 7

Viết chương trình xoay một ma trận 5x5 một góc 180 độ theo chiều kim đồng hồ, theo hai cách :

Dùng ma trận phụ.

Không dùng ma trận phụ.

Bài 8

Nhập vào một mảng hai chiều nxn:

- Kiểm tra xem mảng này có đối xứng qua đường chéo chính không
- Tính tổng bình phương các phần tử mà giá trị của nó là số nguyên tố
- Tính tổng các phần tử trên dòng và trên cột.
- Hiển thị các phần tử vừa lớn nhất trên dòng vừa nhỏ nhất trên cột.
- Chuyển các phần tử âm xuống dưới đường chéo chính, và các phần tử dương lên trên đường chéo chính (giả sử các phần tử âm bằng các phần tử dương) .

Bài 9

Sắp xếp một mảng hai chiều 5x5 tăng dần theo hình zic zắc ngang

$(a[0,0] < a[0,1] < \dots < a[0,n] < a[1,n] < a[1,n-1] < a[1,n-2] < \dots < a[1,0] < a[2,0] < a[2,1] < a[2,2] < a[2,3] < \dots < a[2,n] < a[3,n] < a[3,n-1] < a[3,n-2] < \dots < a[3,0] < a[4,0] < \dots < a[4,4])$.

Gợi ý: Xem cách 1 bài 5

Bài 10

Tương tự như bài trên, nhưng là zic zắc đứng.

Bài 11

Sắp xếp một mảng hai chiều 5x5 tăng dần theo hình xoắn ốc.

Bài 12

Viết chương trình thực hiện phép cộng và nhân hai ma trận kích thước 5x5.

BÀI 8

CHUỖI KÝ TỰ

1. CHUỖI KÝ TỰ

1.1. Khái niệm

Trong C chuỗi ký tự là một dãy các ký tự đặt trong hai dấu nháy kép. Chuỗi rỗng được ký hiệu như sau "" bao gồm hai dấu nháy kép đi liền nhau. Khi gặp một chuỗi ký tự, máy sẽ cấp phát một khoảng nhớ cho một mảng kiểu char đủ lớn để chứa các ký tự của chuỗi và chứa thêm ký tự '\0' là ký tự kết thúc chuỗi.

1.2. Một số điểm cần lưu ý

- Mỗi ký tự của chuỗi được chứa trong một phần tử của mảng. Như vậy chuỗi ký tự là một trường hợp riêng của mảng một chiều khi mỗi thành phần của mảng là ký tự. Tuy nhiên khác với mảng các ký tự, chuỗi ký tự được kết thúc bằng một ký hiệu đặc biệt NUL có mã ASCII bằng 0 (ký hiệu '\0'). Kích thước của chuỗi ký tự phải đủ để chứa được ký tự này. Một chuỗi ký tự được khai báo **char ch[20]** chỉ có thể chứa được nhiều nhất 19 ký tự, vì ký tự còn lại phải là '\0'.
- Cần phân biệt giữa ký tự và chuỗi bao gồm một ký tự. Ví dụ 'A' là ký tự A được mã hóa bằng 1 byte, trong khi "A" là một chuỗi ký tự chứa ký tự A chuỗi này được mã hóa bằng 2 bytes cho ký tự A và ký tự '\0'.

1.3. Các ví dụ minh họa

```
/*Minh họa về chuỗi kí tự*/
#include <stdio.h>
char chuoi[] = "Hello"; /*Khai báo chuỗi*/
void main()
{
    printf("Xau ki tu la %s. \n",chuoi); /*In ra chuỗi ký tự*/
    printf("Cac ki tu se la:\n");      /*In từng phần tử trong chuỗi */
    printf("%c\n",chuoi[0]);
    printf("%c\n",chuoi[1]);
    printf("%c\n",chuoi[2]);
    printf("%c\n",chuoi[3]);
    printf("%c\n",chuoi[4]);
    printf("%c\n",chuoi[5]);
}
```

Kết quả:
Xau ki tu la Hello.
Cac ki tu se la:
H
E
l
l
o

2. CÁC THAO TÁC TRÊN CHUỖI KÝ TỰ

2.1. Một số hàm thông dụng thuộc string.h

Trong C không tồn tại các phép toán so sánh, gán nội dung của chuỗi này cho chuỗi khác. Để thực hiện các công việc này C cung cấp cho người lập trình một thư viện các hàm chuẩn, được khai báo trong tệp header có tên là **string.h**. Để sử dụng các hàm thao tác chuỗi, trên đầu chương trình cần phải có dòng khai báo `#include <string.h>`.

Sau đây là một số hàm thông dụng thuộc `#include <string.h>`:

Mô tả tóm tắt chức năng của các hàm:

□ **Hàm strlen**

`int strlen(char s[])`

Trả về độ dài của chuỗi `s`, chính là chỉ số của ký tự NUL trong chuỗi.

□ **Hàm strcpy**

`strcpy(char dest[], char source[])`

Sao chép nội dung chuỗi `source` vào chuỗi `dest`.

□ **Hàm strncpy**

`strncpy(char dest[], char source[], int n)`

Tương tự như `strcpy()`, nhưng ngừng sao chép sau `n` ký tự. Trong trường hợp không có đủ số ký tự trong `source` thì hàm sẽ điền thêm các ký tự trắng vào chuỗi `dest`.

□ **Hàm strcat**

`strcat(char ch1[], char ch2[])`

Nối chuỗi `ch2` vào cuối chuỗi `ch1`. Sau lời gọi hàm này độ dài chuỗi `ch1` bằng tổng độ dài của cả hai chuỗi `ch1` và `ch2` trước lời gọi hàm.

□ **Hàm strncat**

`strncat(char ch1[], char ch2[], int n)`

Tương tự như `strcat` nhưng chỉ giới hạn với `n` ký tự đầu tiên của `ch2`

□ **Hàm strcmp**

`int strcmp(char ch1[], char ch2[])`

So sánh hai chuỗi `ch1` và `ch2`. Nguyên tắc so sánh theo kiểu từ điển. Giá trị trả về:

- 0 nếu chuỗi `ch1` bằng chuỗi `ch2`
- >0 nếu chuỗi `ch1` lớn hơn chuỗi `ch2`
- <0 nếu chuỗi `ch1` nhỏ hơn chuỗi `ch2`

□ **Hàm strncmp**

`int strncmp(char ch1[], char ch2[], int n)`

Tương tự như hàm `strcmp()`, nhưng chỉ giới hạn việc so sánh với `n` ký tự đầu tiên của hai chuỗi.

□ **Hàm strcmp**

int strcmp(char ch1[], char ch2[])

Tương tự như strcmp(), nhưng không phân biệt chữ in và chữ thường

□ **Hàm strncmp**

int strncmp(char ch1[], char ch2[], int n)

Tương tự như strcmp(), nhưng việc so sánh chỉ giới hạn ở n ký tự đầu tiên của mỗi chuỗi.

□ **Hàm strchr**

char *strchr(char s[], char c)

Tìm lần xuất hiện đầu tiên của ký tự c trong chuỗi s, trả về địa chỉ của ký tự này.

□ **Hàm strrchr**

char *strrchr(char s[], char c)

Tương tự như hàm strchr(), nhưng việc tìm kiếm bắt đầu từ cuối chuỗi .

□ **Hàm strlwr**

strlwr(char s[])

Chuyển đổi các chữ in trong chuỗi s sang chữ thường.

□ **Hàm struppr**

struppr(char s[])

Ngược lại với hàm strlwr()

□ **Hàm strset**

strset(char s[], char c)

Khởi đầu tất cả các ký tự của s bằng ký tự c

□ **Hàm strnset**

strnset(char s[], char c, int n)

Khởi đầu giá trị cho n ký tự đầu tiên của s bằng ký tự c

□ **Hàm strstr**

char *strstr(char s1[], char s2[])

Tìm kiếm chuỗi s2 trong chuỗi s1, Trả về địa chỉ của lần xuất hiện đầu tiên của s2 trong s1 hoặc NULL khi không tìm thấy.

2.2. Các hàm nhập xuất chuỗi thuộc stdio.h

Ngoài ra còn có hai hàm vào ra dùng riêng cho các chuỗi ký tự thuộc stdio.h. Khai báo #include <stdio.h>.

□ **Hàm gets**

Hàm gets() đọc từ bàn phím tất cả các ký tự và điền vào chuỗi s. Việc đọc kết thúc khi số ký tự đọc vào bằng chiều dài cực đại của chuỗi hoặc gặp ký tự xuống dòng '\n' sinh ra khi ta ấn ENTER. Kết thúc việc đọc ký tự, một ký tự '\0' (NUL) được gắn thêm vào cuối chuỗi.

□ **Hàm puts**

Hàm puts() in nội dung chuỗi ký tự s ra màn hình và thay thế ký hiệu '\0' (NUL) kết thúc chuỗi bằng ký hiệu xuống dòng '\n'.

□ **Các ví dụ minh họa**

Lập chương trình:

- Dùng hàm gets() để nhập vào một chuỗi ký tự.
- In chuỗi đó ra màn hình nhờ hàm puts().

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char chuoi[81];
    clrscr();
    printf("\n Nhập vào dòng moi");
    gets(chuoi);
    printf("\n Ban da nhap vào xau");
    puts(chuoi);
    getch();
}
```

kết quả

Nhập vào dòng moi: Hôm nay trời đẹp quá, tôi ngại nhó den em

Ban da nhap vào xau: Hôm nay trời đẹp quá, tôi ngại nhó den em

3. CON TRỎ VÀ CHUỖI KÝ TỰ

3.1. Cách khai báo

Trong một chương trình nếu chúng ta có sử dụng chuỗi ký tự, thì máy sẽ cung cấp một vùng nhớ cho một mảng kiểu char đủ lớn để lưu các ký tự của chuỗi ký tự này và ký tự '\0' ở cuối. Khi đó bản thân chuỗi ký tự này là một hằng địa chỉ. Nó chứa địa chỉ đầu của mảng lưu trữ nó.

3.2. Phân tích các ví dụ

Phân tích trường hợp:

*char *p;*

thì lệnh sau hoàn toàn có nghĩa:

p="Turbo C";

khi đó $*p = 'T'$ $*(p+1) = 'u'$

Và nếu ta dùng lệnh: `printf("%s",p)` thì chuỗi Turbo được in lên màn hình.

Các chương trình minh họa

Ví dụ 1:

/*Minh họa về con trỏ chuỗi*/

```
#include <stdio.h>
char string[] = "Hello";
void main()
{
    char *ptr;
    ptr = string;
    printf("Xau ki tu la %s.\n",string);
    printf("Cac ki tu se la:\n");
    printf("%c\n",*ptr);
    printf("%c\n",*(ptr + 1));
    printf("%c\n",*(ptr + 2));
    printf("%c\n",*(ptr + 3));
    printf("%c\n",*(ptr + 4));
    printf("%c\n",*(ptr + 5));
}
```

Kết quả:

Xau ki tu la Hello.

Cac ki tu se la:

H

e

l

l

o

Ví dụ 2

/*Minh họa về địa chỉ con trỏ trong chuỗi*/

```
#include <stdio.h>
char string[] = "Hello";
void main()
{
    char *ptr;
    ptr = string;
    printf("Xau ki tu la %s. \n",string);
    printf("Cac ki tu se la:\n");
    printf("Address => %d | %c |\n",ptr, *ptr);
    printf("Address => %d | %c |\n",ptr + 1, *(ptr + 1));
    printf("Address => %d | %c |\n",ptr + 2, *(ptr + 2));
    printf("Address => %d | %c |\n",ptr + 3, *(ptr + 3));
    printf("Address => %d | %c |\n",ptr + 4, *(ptr + 4));
    printf("Address => %d | %c |\n",ptr + 5, *(ptr + 5));
}
```

Kết quả:

Xau ki tu la Hello.

Cac ki tu se la:

Address => 5321 | H |

Address => 5322 | e |

Address => 5323		1	
Address => 5324		1	
Address => 5325		o	
Address => 5326			

Ví dụ 3: đoạn chương trình sau có chỗ nào không hợp lệ:

```
char *p,t[25];
t="Turbo c";
scanf("%s",t);
scanf("%s",p);
```

Trả lời: /*đòng lệnh t="Turbo c"; là không hợp lệ vì t là một hằng địa chỉ chứa địa chỉ của mảng gồm 25 ký tự , chứ không phải là một biến con trỏ*/

3.3. Hàm và chuỗi ký tự

Nếu tham số thực là tên chuỗi ký tự chuoi[20], đối ten_chuoi của hàm được khai báo theo hai mẫu:

*char ten_chuoi[] hoặc char *ten_chuoi*

Ví dụ

```
/*Hàm và xâu kí tự: minh họa bài toán in ra một chuỗi*/
#include <stdio.h>
void ham(char ten_chuoi[]);/*Khai báo hàm và đối của hàm */
void main()
{
    char chuoi[20];
    printf("Ten em la gi? => ");
    gets(chuoi);
    ham(chuoi); /*Gọi hàm*/
}
void ham(char ten_chuoi[])
{
    printf("%s yeu dau cua long toi!",ten_chuoi);
}
```

Kết quả:

Ten em la gi? =>Bich Thao
Bich Thao yeu dau cua long toi!

4. MẢNG VÀ CHUỖI KÝ TỰ

Mảng và chuỗi ký tự có thể phối hợp với nhau. Để ứng dụng mảng cho chuỗi ký tự, chúng ta phải làm quen với hai khái niệm mới đó là: *mảng các con trỏ* và *con trỏ chỉ đến con trỏ*. Như vậy, nội dung mục này sẽ gồm:

4.1. Mảng các con trỏ

Một dạng sử dụng con trỏ đặc biệt hơn mà chúng ta sẽ xét ở đây là việc sử dụng một mảng các biến con trỏ. Các mảng này có đặc điểm gì?

Trước tiên chúng ta có thể khai báo một mảng các con trỏ bằng câu lệnh sau:

*type *pointer_array[size];*

ví dụ câu lệnh,

```
char *ma[10];
```

sẽ khai báo một mảng 10 con trỏ *char* có thể được dùng để khai báo một mảng để lưu trữ địa chỉ của mười chuỗi ký tự nào đó.

Việc khai báo như vậy sẽ chỉ cho chúng ta thực sự là một mảng có tên là *pointer_array* gồm size biến con trỏ kiểu *type*, mỗi con trỏ được hiểu là chỉ đến một biến có kiểu dữ liệu *type* (Thực tế tại thời điểm khai báo, các con trỏ thành phần này vẫn chưa được chuẩn bị để chỉ đến đâu cả).

Nếu các con trỏ được chuẩn bị để chỉ đến một biến nào đó đã có, thì như vậy, chúng ta có thể truy xuất được các biến này thông qua một mảng mà không cần đến vị trí thực sự của các biến đó có liên tiếp hay không. Điều đó khiến cho chúng ta dường như đã có được một mảng đặc biệt gồm các biến mà vị trí thực sự của chúng trong bộ nhớ là bất kỳ. Và bằng việc đổi chỗ các con trỏ thành phần này chúng ta có thể đổi thứ tự sắp xếp của các biến trong mảng này mà không cần thay đổi thực sự vị trí của chúng. Xét ví dụ sau:

Ví dụ

Xem xét một mảng các con trỏ *ptr_array* được gán các địa chỉ của các biến *int* có giá trị và vị trí bất kỳ. Chúng ta sẽ dùng một hàm để sắp xếp lại các địa chỉ này trong mảng để sao cho các địa chỉ của các số bé được xếp trước địa chỉ của các số lớn hơn. Lúc đó dù chúng ta không làm thay đổi vị trí hoặc thay đổi các giá trị của các biến nhưng mảng vẫn có vẻ như là một mảng chỉ đến các giá trị đã sắp xếp có thứ tự.

```
#include <stdio.h>
void main()
{
    int i,j,*x;
    int d=10,e=3,f=7;
    int a=12,b=2,c=6;
    int *ptr_array[6];
    /*Gán các thành phần của mảng*/
    ptr_array[0]=&a;
    ptr_array[1]=&b;
    ptr_array[2]=&c;
    ptr_array[3]=&d;
    ptr_array[4]=&e;
    ptr_array[5]=&f;
    /*Sắp xếp lại dãy số*/
    for(i=0;i<5;i++)
        for(j=i+1;j<6;j++)
            if(*ptr_array[i]>*ptr_array[j])
            {
                x=ptr_array[i];
                ptr_array[i]=ptr_array[j];
                ptr_array[j]=x;
            }
    /*In kết quả sau khi sắp xếp*/
    for(i=0;i<6;i++)
        printf(" %d \n",*ptr_array[i]);
    getch();
}
```

Kết quả chạy chương trình

2
3
6
7
10
12

Nếu các phần tử của một các con trở thành phần lại được gán địa chỉ của các mảng khác thì ta sẽ được một mảng của các mảng. Không giống như các mảng hai chiều, các mảng con của chúng ta có thể nằm ở vị trí bất kỳ, và cũng vì ta chỉ lưu trữ địa chỉ của chúng nên việc sắp xếp lại thứ tự các của các mảng này so với nhau thực chất chỉ là việc sắp xếp lại các địa chỉ của chúng trong mảng các con trở của chúng ta mà thôi. Xét ví dụ sau

Ví dụ

Chúng ta cần viết một chương trình nhận nhiều tên người vào từ bàn phím, sắp xếp lại theo thứ tự và in kết quả đã sắp xếp ra. Công việc bao gồm:

-Đọc tất cả các tên người đã được nhập vào cho đến khi hết.

-Nếu có tên được nhập vào, thì:

+Sắp xếp lại các tên này theo thứ tự alphabet

+In các tên ra theo thứ tự đó.

*/*Đoạn chương trình minh họa như sau:*/*

#include <stdio.h>

*#include <string.h> /*Khai bao cac ham thao tac tren xau*/*

*#include <alloc.h> /*Khai bao cac ham cap phat bo nho dong*/*

*#define MAXLINES 100 /*Toi da co 100 ten*/*

*#define MAXLEN 20 /*Ten dai toi da 20 ky tu*/*

void main()

{

*char *strlist[MAXLINES];*

*char name[MAXLEN], *p;*

*int nlines=0; /*Ban dau chua co ten nhap vao*/*

int i,j,len;

printf("CHUONG TRINH SAP XEP DANH SACH TEN\n");

*/*Doc vao cac ten*/*

while(nlines <MAXLINES)

{

printf("hay nhap ten thu %d:",nlines+1);

gets(name);

if ((len=strlen(name))==0)

break;

*if ((p=(char *)malloc(len+1))==NULL) /*Ham*

alloc cap cho p vung nho len + 1 byte/*

*break; /*Khong du bo nho de cap phat*/*

strcpy(p,name);

strlist[nlines++]=p;

}

if (nlines ==0)

```

    {
        printf("Khong doc duoc ten nhap vao");
    }
else
    {
        /*Sap xep cac ten nhap vao*/
        for(i=0;i<nlines-1;i++)
            for(j=i+1;j<nlines;j++)
                if(strcmp(strlist[i],strlist[j])>0)
                {
                    p=strlist[i];
                    strlist[i]=strlist[j];
                    strlist[j]=p;
                }
        /*In danh sach len man hinh*/
        for(i=0;i<nlines;i++)
            printf("%d - %s\n",i+1,strlines[i]);
        /*Giai phong vung nho da cap phat*/
        for(i=0;i<nlines;i++)
            free(strlist[i]); /*Ham nay giai phong vung bo nho da cap phat cho
                                strlist[i]*/
    }
}

```

Như vậy, qua ví dụ trên, ta thấy rằng việc sử dụng một mảng các con trỏ có nhiều ý nghĩa gần giống như sử dụng một mảng hai chiều. Ví dụ, nếu các biến n và m được khai báo là:

```

int m[10][9];
int *n[10];

```

thì cách viết để truy xuất được các phần tử của các mảng này có thể tương tự nhau, chẳng hạn:

$m[6][5]$ và $n[6][5]$ đều cho ta một kết quả là một số *int*.

Khai báo:

```

char slist[10][100];
char *plist[10];

```

đều khiến cho C hiểu rằng $slist[i]$ và $plist[i]$ là các con trỏ chỉ đến một đối tượng là *char*, hoặc là mảng *char*.

Tuy vậy, giữa mảng nhiều chiều và mảng các con trỏ cũng tồn tại nhiều điểm khác nhau:

- Mảng nhiều chiều thực sự là mảng có khai báo, do đó có chỗ đầy đủ cho tất cả các phần tử của nó. Còn mảng các con trỏ chỉ mới có chỗ cho các biến con trỏ mà thôi, giả sử các con trỏ này lại cần chỉ đến một mảng n phần tử, thì như vậy việc xin chỗ cho các mảng và gán địa chỉ của chúng cho các con trỏ là công việc của chúng ta. Như vậy xem ra mảng các con trỏ tốn chỗ hơn mảng nhiều chiều, vì vừa phải lưu trữ các con trỏ, vừa phải có chỗ cho các phần tử sử dụng, và còn mất công để chuẩn bị chỗ và gán cho các con trỏ. Bên cạnh đó, việc sử dụng mảng các con trỏ có hai ưu điểm, đó là

- Việc truy xuất đến các phần tử là truy xuất gián tiếp thông qua các con trỏ và như vậy, vị trí của các mảng con này có thể là bất kỳ, và chúng có thể là những mảng đã có bằng cách xin cấp phát chỗ động hay bằng khai báo biến mảng bình thường, tùy ý [Xem (bài 9)].
- Các mảng con của nó được chỉ đến bởi các con trỏ, có thể có độ dài tùy ý, hoặc có thể không có (nếu con trỏ đó không được chuẩn bị, hoặc được gán bằng NULL).

Đối với mảng các con trỏ, ta có thể hoán chuyển thứ tự của các mảng con được chỉ đến bởi các con trỏ này, bằng cách chỉ hoán chuyển bản thân các con trỏ trong mảng là đủ. Trong khi đối với mảng nhiều chiều, việc hoán đổi thứ tự này phải thực sự là hoán chuyển vị trí của toàn bộ phần tử trong mảng con. Điều này sẽ làm cho chúng ta rất mất thời gian khi kích thước của các mảng lớn.

4.2. Con trỏ chỉ đến con trỏ

Con trỏ cũng là một kiểu dữ liệu, một biến con trỏ cũng có địa chỉ và chúng ta vẫn có thể lấy địa chỉ của nó được, lúc đó ta có một con trỏ chỉ đến một con trỏ. Chúng ta có thể khai báo một con trỏ chỉ đến một con trỏ chỉ đến một biến kiểu type như sau:

```
type **ptr_ptr;
```

ví dụ, chúng ta có khai báo sau,

```
int **int_ptr_ptr;
```

với khai báo này, chúng ta có chỗ cho một biến con trỏ *int_ptr_ptr*, và được ghi nhận rằng nếu lấy đối tượng của biến *int_ptr_ptr* này ta sẽ được kết quả là một con trỏ chỉ đến một biến số nguyên.

Một lần nữa cần phải nhắc lại rằng thực sự biến *int_ptr_ptr* cho đến lúc này vẫn chưa có một đối tượng gì có ý nghĩa, để có thể lấy đối tượng được, và chúng ta phải tự mình gán địa chỉ của một con trỏ nguyên nào đó cho biến *int_ptr_ptr*.

Thật ra khi sử dụng mảng các con trỏ, thì cũng như đã nói ở phần mảng, bản thân tên mảng được hiểu là địa chỉ của con trỏ đầu tiên, và địa chỉ đó có thể được gán cho một con trỏ chỉ đến con trỏ. Xét ví dụ sau:

Ví dụ

```
char *monthname[20] =
{
    "January", "February", "March", "April",
    "May", "June", "July", "August", "September",
    "October", "November", "December"
};
char **pp;
pp=monthname; /*Tên mảng là địa chỉ của phần tử đầu tiên*/
```

khi đó **pp* sẽ chỉ đến con trỏ đầu tiên của mảng, con trỏ này lại chỉ đến chuỗi *"January"* như đã khởi đầu cho mảng. Nếu tăng *pp* lên 1 thì khi đó *pp* sẽ chỉ đến phần tử kế tiếp của mảng có giá trị là con trỏ đến xâu ký tự *"February".v.v.*

4.3. Bài tập ví dụ

*/*Viết chương trình yêu cầu người sử dụng nhập vào một số nguyên giữa 1 và 7. Hiện lên màn hình tên ngày trong tuần theo các số đưa vào (1:Thu hai,2:Thu ba)*/*

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i;
    char *ngay[7]={"Thu hai","Thu ba","Thu tu",
                  "Thu Nam","Thu sau","Thu bay","Chu nhat"};

    clrscr();
    do
    {
        printf("\nNhap vao mot so nguyen giua 1 va 7:");
        scanf("%d",&i);
    } while(i<=0||i>7);
    printf("\n Ngày số %d của tuần là %s ",i,ngay[i-1]);
    getch();
    return;
}
```

5. CÁC BÀI TẬP MINH HỌA

Bài 1

*/*Viết chương trình đếm số lần xuất hiện của một ký tự trong một xâu ký tự*/*

```
#include<stdio.h>
#include<conio.h>
#define HANG 128
void main()
{
    char dong[HANG];
    int i,na;
    clrscr();
    printf("\nNhap mot dong chu:");
    gets(dong);
    na=i=0;
    while(dong[i])
        if (dong[i++]=='a') na++;
    printf("\nDong co %d chu a",na);
    getch();
    return;}

```

Bài 2

*/*Viết chương trình nhập một chữ, xuất ra chữ đó nhiều lần dùng con trỏ*/*

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

```

char *lap(char chu,int lan);
void main()
{
char c;
clrscr();
printf("\n%s",lap('s',3));
c=getchar();
printf("\n%s",lap(c,3));
getch();
}

```

```

char *lap(char chu,int lan)
{
char *supp;
int i;
supp=calloc(lan,sizeof(char));
for(i=0;i<=lan-1;i++)
supp[i]=chu;
return (supp);
}

```

Bài 3
*/*Viết chương trình cho một dòng quảng cáo chạy từ phải sang trái của màn hình*/*

```

#include<stdio.h>
#include<conio.h>
void main()
{
char st[123]="Turbo C kinh chao cac ban ";
char *st_show;
int i;
clrscr();
st_show=(char *) calloc(80,sizeof(char));
for(i=0;i<80;i++)
strcat(st_show," ");
strrev(st);
strcat(st,st_show);
strrev(st);
for(i=0;i<80;i++)
{
strncpy(st_show,st+i,79);
gotoxy(1,1);
cprintf("%s",st_show);
delay(100);
}
free(st_show);
}

```

6. CÁC BÀI TẬP TỰ LÀM

Bài 1: Viết chương trình nhập vào một số nhỏ hơn 1000. Trình bày dòng chữ cho biết giá trị của số đó.

Bài 2: Viết chương trình cộng, trừ hai số nguyên có nhiều chữ số (dùng chuỗi).

BÀI 9

CẤP PHÁT VÀ GIẢI PHÓNG BỘ NHỚ ĐỘNG

1. NHẮC LẠI VỀ CON TRỎ

Chúng ta đã làm quen với kiểu con trỏ trong “Bài 6”. Con trỏ là biến chứa địa chỉ chứ không phải chứa giá trị số liệu.

Nhận xét:

*p có nghĩa: p là con trỏ trỏ tới biến chứa giá trị *p.

&x là địa chỉ của biến x.

2. BIẾN ĐỘNG

2.1. Khái niệm

□ Thế nào là một biến động

Là các biến được tạo ra lúc chạy chương trình, tùy theo nhu cầu. Do vậy, mà số biến này hoàn toàn không được xác định từ trước. Các biến được tạo ra như vậy được gọi là các biến động.

Các biến động không có tên (vì việc đặt tên thực chất là gán cho nó một địa chỉ xác định).

□ Cách tạo ra biến động và truy nhập đến biến động được tiến hành như sau

Việc tạo ra biến động và xóa nó đi (để thu hồi lại bộ nhớ) được thực hiện nhờ các hàm như malloc() và free() đã có sẵn trong stdlib.h

Việc truy nhập đến biến động được tiến hành nhờ các biến con trỏ. Các biến con trỏ được định nghĩa như các biến tĩnh (được khai báo ngay từ đầu trong phần khai báo biến) và được dùng để chứa địa chỉ các biến động.

□ Ví dụ minh họa

```
int *p;                               /* Khai báo biến con trỏ p*/
p = (int *) malloc(100)               /* Tạo biến động*/
```

Đoạn chương trình trên sẽ cấp phát cho chúng ta 100 bytes trong bộ nhớ và gán địa chỉ khối bộ nhớ này cho p. 100 bytes này được dùng để lưu trữ 50 số nguyên.

Giả sử, muốn cấp phát bộ nhớ chính xác cho 80 số nguyên, chúng ta có thể viết:

```
p=(int *) malloc(80*sizeof(int));
```

Hoặc muốn cấp phát bộ nhớ chính xác cho 75 ký tự, có thể viết:

```
char *cp; /* Khai báo biến con trỏ kiểu char */  
cp=(char *) malloc(75* sizeof(char)); /* Tạo biến động */
```

Để chi tiết hơn về vấn đề này, chúng ta xem mục 2.2

2.2. Cấp phát và giải phóng bộ nhớ động (các hàm thuộc stdlib.h và alloc.h)

Để cấp phát bộ nhớ động ta sử dụng các hàm trong thư viện stdlib.h hoặc alloc.h.

Gồm:

□ Cấp phát bộ nhớ động bằng hàm malloc()

Cú pháp

```
void *malloc(size_t size)
```

Chức năng: Hàm malloc cấp phát một vùng nhớ có kích thước là size.

Trong đó:

- size là một giá trị kiểu size_t (là một kiểu dữ liệu định sẵn trong thư viện stdlib.h, ta có thể khai báo kiểu unsigned cũng được).
- Hàm này trả về con trỏ kiểu void chứa địa chỉ ô nhớ đầu của vùng nhớ được cấp phát. Nếu không đủ vùng nhớ để cấp phát nó sẽ trả về giá trị **NULL**, vì vậy ta phải kiểm tra giá trị trả về khi sử dụng hàm malloc.

Phân tích một số trường hợp minh họa

Ví dụ 1:

```
# include <stdlib.h>  
# include <conio.h>  
void main () /* Hàm chính */  
{  
    void *v;  
    clrscr();  
    if ((v=malloc(100))!=NULL)  
    {  
        printf("Khong du bo nho");  
        exit(1);  
    }  
}
```

```
printf("Bo nho da duoc cap phat");
getch();
}
```

- Nếu muốn cấp phát vùng nhớ để lưu một loại dữ liệu xác định nào đó thì ta có thể ép kiểu đối với con trỏ trả về của hàm malloc:

Ví dụ 2

```
int *num;
```

```
num=(int*)malloc(50*sizeof(int));
```

Đòng lệnh trên yêu cầu cấp phát vùng nhớ để lưu giữ 50 giá trị kiểu nguyên. Giá trị trả về là con trỏ kiểu nguyên lưu địa chỉ đầu của vùng nhớ được cấp phát.

Chú ý: Toán tử *sizeof* cho ta kích cỡ tính theo byte của một kiểu dữ liệu (*int, float, hay các kiểu được định nghĩa trong chương trình bằng typedef, enum...*) cũng như một đối tượng dữ liệu (biến, mảng, cấu trúc). Nó được viết như sau:

sizeof(kiểu dữ liệu)

sizeof(đối tượng dữ liệu)

Ví dụ: Như trên *sizeof(int)* cho ta kích cỡ tính theo byte của kiểu *int* (bằng 2 byte).

- **Cấp phát bộ nhớ động bằng hàm calloc()**

Cú pháp

```
(datatype *) calloc(n, sizeof(object));
```

Chức năng của hàm là: cấp phát bộ nhớ động cho các kiểu dữ liệu (có thể là những kiểu dữ liệu không phải kiểu cơ sở)

Trong đó:

- (*datatype **) là kiểu con trỏ trỏ tới kiểu dữ liệu *datatype*.
- *n* là số lượng object thuộc kiểu *datatype* mà ta cần cấp phát bộ nhớ.
- Kiểu *datatype* có thể là những kiểu dữ liệu mới do người lập trình sáng tạo ra.

Phân tích trường hợp minh họa

```
struct sv {
    char ht[30];
    unsigned int tuoi;
    unsigned int diem;
} /* Khai báo cấu trúc */
```

```
struct sv *p_sv; /* Khai báo con trỏ sv*/
```

Cấp phát bộ nhớ cho 10 người:

```
calloc(10, sizeof(struct sv));
```

- **Cấp phát bộ nhớ động bằng hàm realloc()**

Cú pháp

*(datatype *) realloc(buf_p, newsize);*

Phân tích

Chức năng: Hàm cấp phát lại bộ nhớ

Trong đó:

- ☐ buf_p là con trỏ đang trỏ đến vùng ô nhớ đã được cấp phát từ trước.
- ☐ newsize là kích thước mới cần cấp phát, có thể to hoặc nhỏ hơn.

- ☐ **Giải phóng bộ nhớ động bằng hàm free()**

Cú pháp

*void free(void *ptr)*

Chức năng:

Hàm free giải phóng vùng nhớ được trỏ đến bởi con trỏ ptr.

Nếu ptr = NULL thì free không làm gì cả.

Chương trình ví dụ

Viết chương trình nhập các số nguyên từ bàn phím. Xuất các số ra màn hình (không sử dụng mảng):

```
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
void main () /* Ham chinh */
{
    int *num,i,j;
    clrscr();
    printf("Nhap so phan tu");
    scanf("%d",&i);

    /* Cấp phát bộ nhớ bằng hàm malloc */
    num=(int*)malloc(i*sizeof(int));
    for (j=0;j<i;j++)
    {
        printf("\n Ký tự thứ %d",j);
        scanf("%d",(num+j));
    }
    for (j=0;j<i;j++)
        printf("%c \n",*(num+j));
    getch();

    /*Giải phóng bộ nhớ*/
    free(num);
}
```

□ Ví dụ sử dụng hàm malloc

Ví dụ: Dùng hàm malloc cấp phát bộ nhớ cho một chuỗi ký tự

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>
int main(void)
{
    char *str;

    /*Cấp phát bộ nhớ cho chuỗi */
    if ((str = (char *) malloc(10)) == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát \n");
        exit(1); /* Kết thúc chương trình, nếu tràn bộ nhớ*/
    }

    /* Sao chép "Hello" vào chuỗi */
    strcpy(str, "Hello");

    /* Hiển thị chuỗi */
    printf("Chuỗi ký tự là: %s\n", str);

    /* Giải phóng bộ nhớ */
    free(str);

    return 0;
}
```

□ Ví dụ sử dụng hàm calloc

Ví dụ: Dùng hàm calloc cấp phát bộ nhớ cho một chuỗi ký tự

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

int main(void)
{
    char *str = NULL;

    /* Cấp phát bộ nhớ cho chuỗi */
    str = (char *) calloc(10, sizeof(char));

    /* Sao chép "Hello" vào chuỗi */
    strcpy(str, "Hello");
}
```



```

    /* Hiển thị chuỗi */
    printf("Chuỗi ký tự là %s\n", str);
    /* Giải phóng bộ nhớ */
    free(str);

    return 0;
}

```

□ Ví dụ sử dụng hàm realloc

Ví dụ: Dùng hàm realloc cấp phát lại bộ nhớ cho một chuỗi ký tự

```

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

int main(void)
{
    char *str;

    /* Cấp phát bộ nhớ cho chuỗi */
    str = (char *) malloc(10);

    /*Sao chép "Hello" vào chuỗi */
    strcpy(str, "Hello");

    printf("Chuỗi ký tự là %s\n Địa chỉ là %p\n", str, str);
    str = (char *) realloc(str, 20);
    printf("Chuỗi ký tự là %s\n Địa chỉ mới là %p\n", str, str);

    /*Giải phóng bộ nhớ */
    free(str);

    return 0;
}

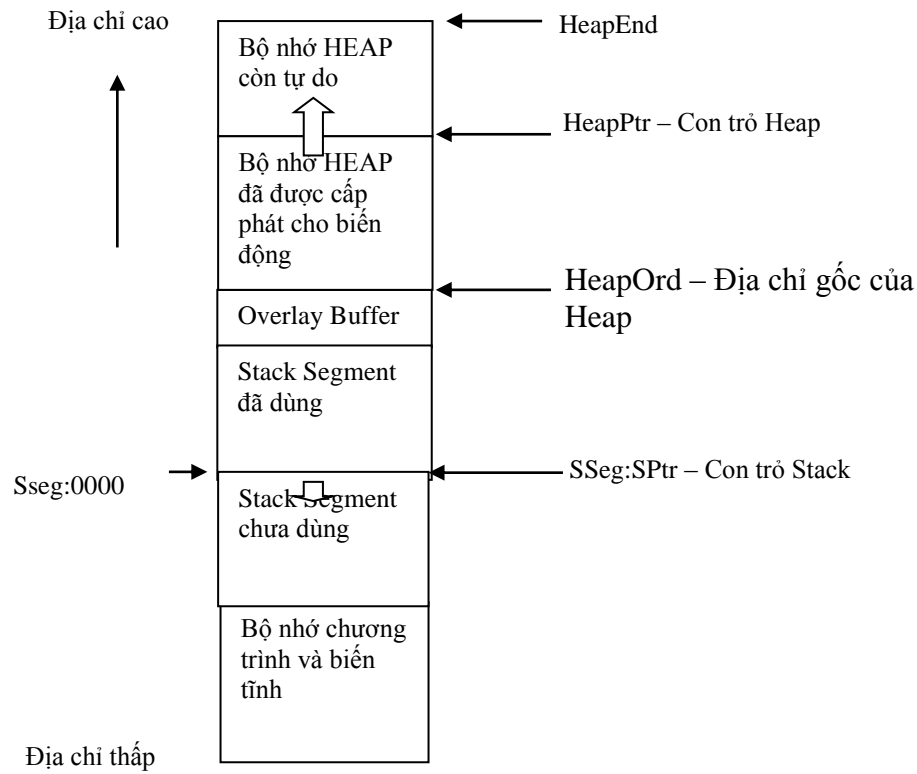
```

3. BỘ NHỚ HEAP VÀ CƠ CHẾ TẠO BIẾN ĐỘNG

3.1. Khái niệm

Các biến động do malloc tạo ra được C xếp vào một vùng ô nhớ tự do theo kiểu xếp chồng và được gọi là HEAP (bộ nhớ cấp phát động). Ngôn ngữ C quản lý HEAP thông qua một con trỏ của HEAP là HEAPPTR. Nó luôn trỏ vào byte tự do đầu tiên của vùng ô nhớ còn tự do của HEAP. Mỗi

lần gọi malloc(), con trỏ của HEAP được dịch chuyển về phía đỉnh của vùng ô nhớ tự do một số byte tương ứng với kích thước của biến động mới tạo ra.



Hình ảnh khái quát việc sử dụng bộ nhớ và Heap

Ngược lại, mỗi khi giải phóng bộ nhớ biến động, bộ nhớ biến động được thu hồi. Tuy nhiên, nếu việc tạo và thu hồi không phải là quá trình liên tục và kế cận thì có thể xảy ra trường hợp: Có những vùng thu hồi nằm lọt trong vùng các biến động khác vẫn còn đang hoạt động.

3.2.Ví dụ áp dụng

Tìm các số nguyên tố trong dãy số tự nhiên.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h>
void main()
{
long *primes=NULL, *start=NULL, *open=NULL, trial=0;

int i=0, found=0, total=0;
printf("Bao nhieu so nguyen to ban can tim ?");
scanf("%d",&total);
primes=(long *) malloc(total*sizeof(long));
if (primes==NULL)
{
printf("\n Khong du bo nho");
return;
}
/*3 so nguyen to dau tien da biet*/

*primes=2;
*(primes+1)=3;
*(primes+2)=5;
open=primes+3; /*Lay dia chi phan o nho tu do tiep theo*/
trial=5;
do
{
trial+=2; /*Gia tri tiep theo de kiem tra*/
start=primes; /*Start tro vao phan dau cua prime*/
found=0;
for(i=0;i<open-primes;i++)
if (found=(trial%*start++)==0) break;
if (found==0) /*Tim thay mot so moi*/
*open++=trial;
} while (open-primes<=total);

for(i=0;i< 5*(total/5);i+=5) /*Hien thi 5 so 1 lan*/
printf("\n %12ld %12ld %12ld %12ld %12ld",
*(primes+i),*(primes+i+1),*(primes+i+2),*(primes+i+3),*(primes+i+4));
}
```

Kết quả chạy chương trình:

Bao nhieu so nguyen to ban can tim ?10

2	3	5	7	11
13	17	19	23	29

4. BÀI TẬP MINH HỌA

Bài 1

*/*Viết chương trình nhập một chữ, xuất ra chữ đó nhiều lần dùng con trỏ*/*

```
#include<stdio.h>
#include<conio.h>
char *lap(char chu,int lan);
void main()
{
    char c;
    clrscr();
    printf("\n%s",lap('s',3));
    c=getchar();
    printf("\n%s",lap(c,3));
    getch();
}

char *lap(char chu,int lan)
{
    char *supp;
    int i;
    supp=calloc(lan,sizeof(char));
    for(i=0;i<=lan-1;i++)
        supp[i]=chu;
    return (supp);
}
```

Bài 2

*/*Viết chương trình cho một dòng quảng cáo chạy từ phải sang trái của màn hình*/*

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char st[123]="Turbo C kinh chao cac ban ";
    char *st_show;
    int i;
    clrscr();
    st_show=(char *) calloc(80,sizeof(char));
    for(i=0;i<80;i++)
        strcat(st_show," ");
    strrev(st);
    strcat(st,st_show);
```

```

strrev(st);
for(i=0;i<80;i++)
{
strncpy(st_show,st+i,79);
gotoxy(1,1);
cprintf("%s",st_show);
delay(100);
}
free(st_show);
}

```

5. CÁC BÀI TẬP TỰ LÀM

Bài tập 1

Sử dụng việc cấp phát bộ nhớ động (không sử dụng mảng) để nhập hai dãy số số phần tử của mỗi dãy được nhập từ bàn phím. Sau đó in ra tổng của mỗi dãy, và tổng của hai dãy này (dựa vào ví dụ phân cấp phát động).

Bài tập 2

Nhập vào mảng a và b theo kiểu cấp phát động (không dùng mảng). Với:

1. 1. Các phần tử của a và b không trùng nhau.
2. 2. Xếp theo thứ tự tăng dần hai mảng a, b.
3. 3. Nối hai mảng này thành một mảng duy nhất sao cho mảng vẫn tăng.

Bài tập 3

Viết chương trình thực hiện công việc sau:

1. 1. Nhập vào số nguyên dương N. Cấp phát động một mảng A có N phần tử. Thực hiện việc nhập giá trị cho mảng này.
2. 2. Kiểm tra mảng A có phải là mảng đan dấu hay không
3. 3. Tìm số nguyên tố lớn nhất trong mảng nếu không có phải thông báo

Bài 4

Viết chương trình tạo ngẫu nhiên hai ma trận vuông a,b (nxn) theo kiểu cấp phát động.

1. 1. In hai ma trận a,b đã được tạo.
2. 2. In ra ma trận tổng
3. 3. In ra ma trận tích.

BÀI 10

HÀM MAIN CÓ THAM SỐ – CON TRỎ HÀM

1. HÀM MAIN CÓ THAM SỐ

1.1. 1.1. Qui định về giá trị truyền cho tham số của hàm main

Với hàm main ta có thể dùng tham số. Tuy nhiên phải tuân theo quy định:
Các giá trị truyền cho tham số của hàm main sẽ được ghi sau tên chương trình khi gọi nó thực hiện tại dấu nhắc của dos. Các tham số này được xem như các chuỗi ký tự.

1.2. Các dạng tham số

- **Có hai tham số :**
- Tham số thứ nhất có kiểu int: Tham số này để ghi nhận số giá trị mà ta đã gõ khi gọi thực hiện chương trình (tính luôn cả tên chương trình, các tham số được phân biệt qua khoảng trắng).
- Tham số thứ hai là một mảng con trỏ ký tự khai báo dạng char *arr[]. Mảng này sẽ ghi nhận con trỏ vào vùng nhớ của các chuỗi giá trị mà ta truyền cho chương trình. Phần tử thứ 0 sẽ chứa con trỏ chỉ đến vùng nhớ chứa tên chương trình.

1.3. Ví dụ minh họa

Xét đoạn chương trình tính tổng mảng các số thực:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main(int n,char *par[])
{ float s=0;
  int i;
  for (i=1;i<n;i++)
    s+=atof(par[i]);
  printf("Ket qua %.2f ",s);
  getch();
}
```

Sau khi nhập nội dung lưu lên đĩa với tên thidu biên dịch ra file thidu.exe, rồi thoát ra dos.

Tại dấu nhắc ta gõ thidu 1 2 3 enter

thì trên màn hình hiện ra “Kết quả 6”

Nếu gõ thidu 1 2 3 4 5 enter ==> “Kết quả 15”

2. CON TRỎ HÀM

2.1. Khái niệm

Mặc dù một hàm không phải là một biến, nhưng nó vẫn chiếm vị trí trong bộ nhớ và ta có thể gán vị trí của nó cho một con trỏ. Con trỏ này trỏ đến điểm xâm nhập vào hàm. Con trỏ hàm có thể sử dụng thay cho tên hàm, và việc sử dụng con trỏ cho phép các hàm cũng được truyền như là tham số cho các hàm khác.

2.2. Cách biên dịch và gọi một hàm trong C

Để hiểu được các con trỏ hàm làm việc như thế nào, ta cần hiểu một chút về cách biên dịch và gọi một hàm. Khi biên dịch hàm, trình biên dịch chuyển chương trình nguồn sang dạng mã máy và thiết lập một điểm xâm nhập vào hàm (chính là vị trí của chỉ thị mã máy đầu tiên của hàm). Khi có lời gọi thực hiện hàm, máy tính sẽ thực hiện một chỉ thị call chuyển điều khiển đến điểm xâm nhập này. Trong trường hợp gọi hàm bằng tên hàm thì điểm xâm nhập này là trị tức thời (gần như là một hằng và không chứa trong biến nào cả), cách gọi hàm này gọi là cách gọi hàm trực tiếp. Trái lại, khi gọi hàm gián tiếp thông qua một biến trỏ thì biến trỏ đó phải trỏ tới chỉ thị mã máy đầu tiên của hàm đó. Cách gọi hàm thông qua biến trỏ là cách gọi hàm gián tiếp.

Ví dụ như chương trình dưới đây :

Có hai lý do khai báo strcmp() trong main(). Trước hết, chương trình phải biết kiểu của giá trị mà hàm strcmp trả về và thứ hai, trình biên dịch phải nhận biết được tên của nó là tên của một hàm. Trong C, không giống như những biến khác, một hàm không có cách nào để trực tiếp khai báo một biến trỏ chỉ đến hàm đó, nghĩa là không có cách khai báo con trỏ kiểu hàm mà phải khai báo con trỏ kiểu void. Khi gọi hàm Check, các tham số truyền vào cho hàm này là hai con trỏ trỏ tới hai chuỗi ký tự và một con trỏ trỏ tới hàm strcmp. Bên trong hàm Check, hàm strcmp được gọi bằng phát biểu:

*(*cmp)(a,b)*

Các tham số a và b được truyền bình thường cho con trỏ hàm như trường hợp gọi theo tên.

*/*Đoạn chương trình so sánh hai chuỗi*/*

```
main()
{
    int strcmp();           /* Khai báo một hàm */
    char s1[80], s2[80];
    void *p;
    p = strcmp;
    gets(s1);
    gets(s2);

    check( s1, s2, p);
}
check( char *a, char *b, int (*cmp)())
{
    printf("Kiểm tra hai chuỗi giống nhau không ?\n");
    if(!(*cmp)(a,b)) printf("Giống nhau");
    else printf("Khác nhau");
}
```

Chú ý rằng, cũng có thể gọi check theo cách trực tiếp sử dụng strcmp như sau:

check(s1, s2, strcmp);

Phát biểu này hạn chế bớt việc phải dùng thêm một biến trả. Tuy nhiên hầu hết các chương trình con đều sử dụng con trả chỉ đến hàm để chương trình linh động hơn. Chẳng hạn xem xét ví dụ dưới đây :

```
/*Đoạn chương trình so sánh hai chuỗi*/
#include <ctype.h>

void main()
{
    int strcmp();           /* Khai báo hai hàm */
    int numcmp();
    char s1[80], s2[80];
    void *p;
    gets(s1);
    gets(s2);

    if (tolower(*s1) <= 'z' && tolower(*s1) >= 'a')
        p = strcmp;
    else
        p = numcmp;

    check( s1, s2, p);
}

check( char *a, char *b, int (*cmp)())
{
    printf("Kiểm tra hai chuỗi giống nhau không ?\n");
    if(!(*cmp)(a,b)) printf("Giống nhau");
    else printf("Khác nhau");
}

int numcmp( char *a, char *b)
{
    if (atoi(a)==atoi(b)) return 0;
    else return 1;
}
```

Như vậy trong chương trình này, bằng một lần gọi check duy nhất, ta có thể kiểm tra được hai chuỗi chữ cái, hoặc hai chuỗi chữ số giống nhau hay không tùy theo kí tự đầu tiên của chuỗi nhập vào là chữ cái hay chữ số bằng cách gọi hàm với biến trả p mà biến trả này tùy tình huống có thể trả đến hàm strcmp (khi kí tự đầu tiên là chữ cái) hay numcmp (khi kí tự đầu tiên là chữ số).

2.3. Cách khai báo con trả hàm và mảng con trả hàm

- **Ý nghĩa câu lệnh float (*f) (float), (*mf[50]) (int);**

Tác dụng câu lệnh:

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

là khai báo:

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

- **Ý nghĩa câu lệnh double (*g) (int, double), (*mg[30]) (double, float);**

Tác dụng câu lệnh:

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

là khai báo:

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

2.4. 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 minh họa

ứng dụng cho hàm tính max

Ví dụ 1: Vừa gán vừa khai báo

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

```
double fmax(double x, double y)
{
    return (x>y?x:y);
}
```

*/*Khai báo và gán tên hàm cho con trỏ hàm */*

```
double (*pf) (double,double)=fmax;
```

/ sử dụng con trỏ hàm */*

```
void main()
```

```
{
    printf("\n max=%f",pf(1.3, 6.7));
}
```

Ví dụ 2: Gán sau khi khai báo

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

```
double fmax(double x, double y)
{
```

```

return (x>y?x:y);
}
/*Khai báo con trỏ hàm*/
double (*pf) (double,double);
void main()
{
/* Gán tên hàm cho con trỏ hàm*/
pf=fmax;
printf("\n max=%f",pf(1.3, 6.7));
}

```

2.5. Đối con trỏ hàm

□ Định nghĩa

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

□ Cách dùng con trỏ trong thân hàm

Nếu có đối được khai báo:

*double (*f) (double, int)*

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

*f(x,m) (f) (x,m) (*f) (x,m)*

ở đây x là biến double, m là biến int.

□ Ví dụ

Lập hàm tính tích phân của f(x) trên đoạn [a,b] theo phương pháp hình thang bằng cách chia [a,b] thành 1000 khoảng có độ dài như nhau.

Dùng hàm trên để tính:

S1= Tích phân trên [0,PI/2] của sin(x)

S2= Tích phân trên [0,PI/2] của cos(x)

S3= Tích phân trên [0,1.0] của exp(x)

S4= Tích phân trên [-1.2,3.5] của $g(x) = (\exp(x) - 2 \cdot \sin(x \cdot x)) / (1 + \text{pow}(x, 4))$;

*/*Chương trình giải bài toán*/*

```

#include<stdio.h>
#include<math.h>
double tp(double (*f) (double), double a, double b);
double g(double);
double tp(double (*f) (double), double a, double b)

```

```

{
    int i,n=1000;
    double s,h=(b-a)/n;
    s=(f(a)+f(b))/2;
    for(i=1; i<n;++i)
        s+=f(a+i*h);
    return h*s;
}

double g(double)
{
    double s;
    s=(exp(x)-2*sin(x*x))/(1+pow(x,4));
    return s;
}

void main()
{
    printf("\n S1=%f", tp(sin,0,M_PI/2));
    printf("\n S2=%f", tp(cos,0,M_PI/2));
    printf("\n S3=%f", tp(exp,0,1.0));
    printf("\n S4=%f", tp(g,-1.2,3.5));
}

```

2.6. Một số điểm cần lưu ý

Cần lưu ý rằng:

- Các hàm mà nó có thể trở thành đối tượng được gọi trong hàm khác thì nên khai báo trước.
- Một con trỏ hàm được khai báo tổng quát như sau:

$$\text{kiểu } (f^*)()$$
 trong đó f là tên con trỏ hàm và kiểu là kiểu trả về
- Khi gọi một hàm bằng con trỏ hàm f cần phải có dấu ngoặc, hàm được gọi theo kiểu

$$(*f)(\text{danh sách các tham số nếu có})$$
- Không có sự khác biệt nào giữa việc gọi bằng tên của hàm và gọi bằng con trỏ hàm.

3. CÁC BÀI TẬP MINH HỌA

Bài 1

```

/*Chương trình in ra tham số dòng lệnh
   Tên tập tin thực hiện chương trình là IN_TSDL
*/
#include<stdio.h>
void main(int n, char **a)

```

```

{
    int i;
    printf("\n Ten chương trình:%s",a[0]);
    printf("\n Cac tham so nhan duoc:");
    for(i=1;i<n;i++)
        printf("\n %s",a[i]);
}

Thực hiện các thao tác:
C:\TC\BIN> IN_TSDL Hom nay ngay 10 thang 10

Các kết quả in ra của chương trình:
Ten chương trình: C:\TC\BIN> IN_TSDL.EXE
Cac tham so nhan duoc:
Hom
nay
ngay
10
thang
10

```

Bài 2

Chương trình dưới đây sẽ:

- ☐ **Xây dựng hàm sắp xếp dãy n đối tượng đặt trong vùng nhớ do con trỏ buf (kiểu void) trỏ tới. Độ dài của đối tượng là size byte. Tiêu chuẩn sắp xếp cho theo hàm ss.**
- ☐ **Dùng hàm trên để sắp xếp dãy số thực theo thứ tự tăng dần.**

```

/* Sắp xếp tổng quát:
   n đối tượng
   chiều dài đối tượng là size
   Địa chỉ đầu buf
   Theo tiêu chuẩn ss là một hàm
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<mem.h>
#include<alloc.h>
void sort(void *buf, int size, int n, int (*ss) (void *, void *));
int tang(void *u, void *v);
int tang(void *u, void *v)
{
    return (*(float *) u) <= (*(float *) v);
}
void sort(void *buf, int size, int n, int (*ss) (void *, void *))

```

```

{
    void *tg; char *p; int i, j;
    p=(char *) buf;
    tg=(char *) malloc(size);
    for(i=0;i<n-1; i++)
        for(j=i+1; j<n; j++)
            if (!ss(p+i*size,p+j*size))
            {
                memcpy(tg,p+i*size,size);
                memcpy(p+i*size,p+j*size,size);
                memcpy(p+j*size,tg,size);
            }
}

float x[]={20,25,10,5,15};
void main()
{
    int j;
    sort(x,4,5,tang);
    clrscr();
    for(j=0;j<5;j++)
        printf("%10.2f",x[j]);
}

```

Bài 3

Chương trình dưới đây minh họa cách dùng mảng con trỏ hàm để lập bảng giá trị của các hàm: $x*x$, $\sin(x)$, $\cos(x)$, $\exp(x)$ và \sqrt{x} . Biến x chạy từ 1.0 đến 10.0 theo bước 0.5.

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
double bp(double x) /*tính x*x*/
{
    return x*x;
}

void main()
{
    int j; double x=1.0;
    typedef double (*ham)(double); /*Khai bao mang con tro ham*/
    ham f[6];
    /*Gan ten ham cho cac phan tu mang con tro ham*/
    f[1]=bp; f[2]= sin; f[3]=cos; f[4]=exp; f[5]=sqrt;
}

```

```

/*Lap bang gia tri*/
while (x<=10.0)
{
    printf("\n");
    for(j=1; j<=5; j++) printf("%10.2f",f[j](x) );
    x+=5;
}
}

```

4. CÁC BÀI TẬP TỰ LÀM

Viết hàm thực hiện các bài toán dưới đây:

Bài 1: Giải hệ phương trình bậc 1

$$ax + by = c$$

$$dx + ey = c$$

Hàm có 6 đối vào a,b,c,e,f và hai đối ra x,y.

Bài 2: Tính đa thức cấp n

Hàm có hai đối là biến nguyên n và mảng thực a.

Bài 3: Tìm cực đại và cực tiểu của một dãy số.

Bài 4: Giải phương trình

$$AX = B$$

Bằng phương pháp khử Gauss (trong đó A là ma trận vuông cấp n, và b là véc tơ cấp n).

Bài 5: Tìm nghịch đảo của ma trận vuông bằng phương pháp jordan.

Bài 6: Tính tích phân của hàm f(x) trên đoạn [a,b].

Bài 7: Xây dựng hàm h(x) là max của hàm f(x) và g(x) trên đoạn [a,b].

Bài 8: Tính các giá trị nhỏ nhất và lớn nhất của hàm f(x) trên đoạn [a,b].

Bài 9: Tìm một nghiệm của phương trình f(x)=0 trên đoạn [a,b]. Giả thiết hàm f(x) liên tục trên [a,b] và f(a)f(b)<0.

Bài 10

Viết chương trình giải phương trình bậc 2 theo dạng truyền tham số cho hàm main.

BÀI 11

KIỂU CẤU TRÚC

1. KIỂU ENUM

□ □ Cú pháp

□ □ Ví dụ

1.1. 1.1. Cú pháp

Câu lệnh kiểu **enum** có thể viết theo bốn dạng sau:

```
enum tk {pt1,pt2,...} tb1,tb2,...;  
enum tk {pt1,pt2,...};  
enum {pt1,pt2,...} tb1,tb2,...;  
enum {pt1,pt2,...};
```

Trong đó :

Tk là tên kiểu **enum** (một kiểu dữ liệu mới),
pt1,pt2,... là tên các phần tử,
tb1,tb2,... là tên biến kiểu enum.

1.2. Ví dụ

Ví dụ để làm việc với các ngày trong tuần ta có thể dùng kiểu weekday và biến day như sau:

```
enum weekday{SUNDAY,MONDAY,TUESDAY,WEDSDAY,THURSDAY,  
FRIDAY, SATURDAY} day;
```

Chú ý biến kiểu **enum** thực chất là biến nguyên, nó được cấp phát 2 byte bộ nhớ và nó có thể nhận một giá trị nguyên bất kỳ.

2. KIỂU CẤU TRÚC

2.1. Khái niệm và định nghĩa cấu trúc

Cấu trúc là một kiểu dữ liệu bao gồm nhiều thành phần có thể thuộc nhiều kiểu dữ liệu khác nhau. Các thành phần được truy nhập thông qua tên. Khái niệm cấu trúc trong C có nhiều nét tương tự như khái niệm về bản ghi (record) trong PASCAL hay trong FOXPRO.

2.2. Khai báo cấu trúc

- **Cú pháp tổng quát để định nghĩa một cấu trúc như sau:**

```
struct [tên_cấu_trúc]  
{  
    khai báo các thành phần  
} [danh sách các biến cấu trúc];
```

trong đó: *struct* là từ khóa đứng trước một khai báo cấu trúc,
tên_cấu_trúc là một tên hợp lệ được dùng làm tên cấu trúc;
danh sách các biến cấu trúc liệt kê các biến có kiểu cấu trúc vừa khai báo,
khai báo các thành phần là một danh sách các khai báo tên và kiểu dữ liệu của các thành phần tạo nên cấu trúc này.

- **Ví dụ**

```
câu lệnh  
struct hoc_sinh {  
    char ho_ten[20];  
    float diem;  
} hs, dshs[100];
```

khai báo một kiểu cấu trúc có tên là *hoc_sinh* gồm hai thành phần:

- Họ tên học sinh (*ho_ten*) và
- Điểm thi (*diem*) của học sinh.

Kèm theo khai báo kiểu cấu trúc, chúng ta định nghĩa hai biến: *hs* là một biến đơn, còn *dshs* là một mảng có các thành phần là cấu trúc *hoc_sinh*.

2.3. Định nghĩa kiểu bằng typedef

- **Cú pháp**

Ngôn ngữ C cho phép ta đặt lại tên kiểu cho riêng mình bằng câu lệnh như sau:

```
typedef kiểu_đã_có tên_kiểu_mới;
```

trong đó :

- *kiểu_đã_có* là kiểu dữ liệu mà ta muốn đổi tên.
- *tên_kiểu_mới* là tên mới mà ta muốn đặt.

- **Ví dụ**

Xét câu lệnh sau

```
typedef unsigned char byte;
```

sau câu lệnh này, *byte* được xem như là kiểu dữ liệu tương đương với *unsigned char* và có thể được sử dụng trong các khai báo biến như các kiểu dữ liệu khác.

Chương trình ví dụ

```
#include <stdio.h>
typedef unsigned char byte;
void main()
{
    byte ch=12, ch1;
    int i;
    ch1=ch;
    for (i=0;i<5;i++)
        ch>>=1;
    printf("byte %X sau khi dịch phải nam lan là %X",ch1,ch);
    getch();
}
```

Kết quả chạy chương trình

Byte C sau khi dịch phải nam lan là 0

typedef thường được sử dụng để định nghĩa các kiểu dữ liệu phức hợp thành một tên duy nhất để dễ dàng viết hơn trong khi viết.

Ví dụ

```
typedef int * PTR_INT;
```

định nghĩa một kiểu dữ liệu con trỏ nguyên. Sau câu lệnh này để khai báo một biến con trỏ nguyên chúng ta chỉ cần viết:

```
PTR_INT ptr_int;
```

Đặc biệt đối với các kiểu dữ liệu cấu trúc, typedef cho phép đơn giản hóa cách viết khai báo. Xét ví dụ sau:

```
typedef struct hoc_sinh{
    char ho_ten[20];
    float diem;
}t_hoc_sinh,*ptr_hoc_sinh;
```

với câu lệnh này tên mới của cấu trúc **struct hoc_sinh** sẽ là **t_hoc_sinh**. Đồng thời câu lệnh này còn định nghĩa một kiểu con trỏ cấu trúc có tên là **ptr_hoc_sinh**. Khi đó câu lệnh khai báo biến cấu trúc viết gọn lại như sau:

```
t_hoc_sinh hs,dshs[100];
```

Để khai báo một biến con trỏ cấu trúc ta có thể sử dụng câu lệnh sau:

```
ptr_hoc_sinh ptrhs;
```

câu lệnh này rõ ràng ngắn gọn hơn so với câu lệnh

*struct hoc_sinh *ptrhs.*

2.4. Truy nhập đến các thành phần của cấu trúc

□ Nguyên tắc chung

Các thành phần của cấu trúc được truy nhập thông qua tên biến cấu trúc và tên thành phần. Nguyên tắc chung như sau:

tên_biến_cấu_trúc.tên_thành_phần

chẳng hạn để truy nhập đến các thành phần của biến *hs* chúng ta viết như sau:

hs.ho_ten
hs.diem

□ Ví dụ mẫu

Chương trình sau đây in nhập vào họ tên và điểm thi của các học sinh của một lớp học và in ra danh sách các học sinh phải thi lại.

```
#include <stdio.h>
#include <string.h>
void main()
{
    struct hoc_sinh
    {
        char ho_ten[20];
        float diem;
    }dshs[100];
    int n; /*Số lượng học sinh của lớp được nhập*/
    int k; /*Số lượng học sinh thi lại*/
    int i;
    char name[20];
    float d;
    /*Nhập thông tin các học sinh*/
    n=0; /*Ban đầu chưa có học sinh nào*/
    do {
        printf("Nhap vao hoc sinh thu %d\n",n+1);
        printf(" Ho ten : ");gets(name);
        if(strcmp(name,"")!=0)
        {
            strcpy(dshs[n].ho_ten,name);
            printf(" Diem : ");
            scanf("%f",d);
            dshs[n++].diem=d;
        }
    } while (strcmp(name,"")!=0);
```

```

if (!n) /*Không nhập được học sinh nao cả!*/
{
    printf("Chua nhap hoc sinh nao\n");
    return 0;
}
else
{
    k=0;
    printf(" Danh sach sinh vien phai thi lai\n");
    for (i=0;i<n;i++)
        if(dshs[i].diem <5)
            printf("%d.%s %6.3f ",++k,dshs[i].ho_ten,dshs[i].diem);
    if (!k)
        printf("Khong co sinh vien thi lai\n");
    }
    getch();
}

```

Kết quả:

Nhap vao thong tin cua hoc sinh 1

Ho ten : Nguyen Bich Thao

Diem : 9

Nhap vao thong tin cua hoc sinh 2

Ho ten : Pham Hai Yen

Diem : 4

Nhap vao thong tin cua hoc sinh 3

Ho ten : Nguyen Van Anh

Diem : 9

Nhap vao thong tin cua hoc sinh 4

Ho ten : Do thanh Xuan

Diem : 3

Nhap vao thong tin cua hoc sinh 5

Ho ten : Do bich Ha

Diem : 7

Nhap vao thong tin cua hoc sinh 6

Ho ten :

Danh sach sinh vien phai thi lai

1. Do thanh Xuan 3.0

2. Pham Hai Yen 4.0

Co tong so 2 hoc sinh phai thi lai

□ **Lưu ý:**

- Không nên sử dụng toán tử & đối với các thành phần cấu trúc (đặc biệt đối với các thành phần không nguyên) trong khi nhập dữ liệu vì điều đó hay dẫn đến việc treo máy.
- Đối với các biến cấu trúc đơn ví dụ hs có thể sử dụng #define để rút gọn "đường truy nhập" đến các thành phần cấu trúc,

ví dụ

```
#define ht hs.ho_ten
#define ds hs.diem
```

Có thể áp dụng phép gán cho các biến cấu trúc có cùng kiểu.

Chương trình minh họa:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main()
{
    struct {
        char ht[20];
        int x,y;
    }a,b;

    clrscr();
    a.x=10;a.y=10;
    strcpy(a.ht,"hhsdhs");
    b=a;
    printf("a: {%s %d %d}\n",a.ht,a.x,a.y);
    printf("b: {%s %d %d}\n",b.ht,b.x,b.y);
    getch();
}
```

Kết quả

A: {hhsdhs 10 10}

B: {hhsdhs 10 10}

2.5. Con trỏ cấu trúc

□ Cách khai báo

Một biến cấu trúc cũng là một biến trong bộ nhớ, do đó, ta cũng có thể lấy địa chỉ của một biến cấu trúc bằng toán tử lấy địa chỉ &. Giá trị trả lại là địa chỉ đến trường đầu của cấu trúc.

Ta có thể khai báo một biến con trỏ chỉ đến một cấu trúc để có thể "cất" địa chỉ của một biến cấu trúc nào đó cần thiết. Cú pháp khai báo một biến con trỏ cấu trúc như sau:

```
struct tên_cấu_trúc *tên_con_trỏ;
```

Ví dụ

```
struct hoc_sinh *ptrhs;
```

Tất nhiên khi đó biến con trỏ cấu trúc ptrhs này chỉ mới được cấp chỗ, và được ghi nhận là con trỏ chỉ đến cấu trúc hoc_sinh, còn đối tượng mà con trỏ này trỏ tới vẫn chưa được chuẩn bị gì cả, và do đó chưa thể dùng được đối tượng này. Chúng ta phải gán cho nó một địa chỉ của một biến cấu trúc cùng kiểu nào đó đã được khai báo hoặc cũng có thể xin cấp phát một vùng bộ nhớ chỉ bởi con trỏ thông qua các hàm cấp phát động.

Ví dụ

```
struct hoc_sinh hs={"Nguyen van A",6.5};/*khai báo và khởi đầu biến cấu trúc*/  
struct hoc_sinh *ptrhs;  
ptrhs= &hs
```

kể từ thời điểm này biến ptrhs đã chỉ đến biến cấu trúc hs và chúng ta có thể truy xuất đến các thành phần của biến hs một cách gián tiếp thông qua con trỏ ptrhs.

□ Truy xuất đến các thành phần cấu trúc

Việc truy xuất đến một thành phần của cấu trúc thông qua một con trỏ được thực hiện bằng phép toán kép -> (bao gồm - và > viết liền nhau). Chẳng hạn có thể in ra các thành phần của hs bằng hai câu lệnh sau:

```
printf("\nHo va ten hoc sinh %s",ptrhs->ho_ten);  
printf("\nDiem %6.3f",ptrhs->diem);
```

kết quả thực hiện hai câu lệnh này tương đương với hai câu lệnh sau:

```
printf("\nHo va ten hoc sinh %s",hs.ho_ten);  
printf("\nDiem %6.3f",hs.diem);
```

□ Ứng dụng

Việc sử dụng con trỏ chỉ đến cấu trúc thường được sử dụng để truyền cấu trúc đến cho một hàm. Tất nhiên chúng ta có thể gửi trực tiếp một cấu trúc làm đối số cho một hàm, nhưng nếu cấu trúc đó lớn, việc chép lại toàn bộ cấu trúc đó để gửi đi sẽ làm mất nhiều thời gian. Hơn nữa, có thể chúng ta lại muốn thay đổi nội dung của cấu trúc đó từ một hàm khác. Trong những trường hợp đó, nên gửi địa chỉ của cấu trúc đó cho hàm.

Một ứng dụng khác của con trỏ cấu trúc mà chúng ta sẽ đề cập đến trong các phần sau là việc xây dựng các cấu trúc tự trỏ như: danh sách liên kết (còn gọi là danh sách móc nối).

2.6. Thành phần kiểu FIELD (nhóm bit)

□ Cách khai báo

Ngôn ngữ C cho phép chúng ta khai thác đến từng bit như là một thành phần riêng biệt của một cấu trúc. Một thành phần như vậy được gọi là một field (tạm dịch là một vùng). Khai báo cho một cấu trúc có các thành phần là các field được thực hiện tương tự như khai báo một cấu trúc bình thường:

```
struct struct_field {
    unsigned field1 :số_bit1;
    int      field2 :số_bit2;
    ...
}var_field;
```

trong khai báo cấu trúc field trên, tên các thành phần là field1, field2 v.v. Kiểu của các thành phần phải là int hay unsigned int. Độ dài tính theo bit của các field được ghi ngay sau tên trường và được phân tách với tên trường bằng dấu hai chấm. Độ dài này không được vượt quá 16 bits. Xét câu lệnh khai báo sau:

Ví dụ

```
struct date
{
    unsined int day: 5; /*giá trị ngày từ 1 đến 31*/
    unsined month : 4; /*tháng từ 1 đến 12*/
    unsined year   :5; /*chỉ xét 32 năm:1980-2011*/
    int t: 2;      /*0:mm-dd*/
}x;
```

câu lệnh khai báo trên định nghĩa một kiểu cấu trúc field với tên là date kèm theo một biến là x. Theo khai báo trên, kích thước của cấu trúc là 24 bit hay 3 byte.

□ Lưu ý

Liên quan đến các cấu trúc kiểu field chúng ta có một số điểm cần lưu ý sau:

- Không cho phép lấy địa chỉ của các thành phần kiểu field. Nghĩa là phép toán dạng &x.a là không hợp lệ.
- Không thể xây dựng các mảng có kiểu cấu trúc field.
- Một hàm không thể trả về một giá trị có kiểu cấu trúc field.
- Khi muốn bỏ qua một số bit nào đó chúng ta bỏ trống tên trường. *Ví dụ*

```
struct {
    int :8; /*Bỏ qua 8 bit*/
    int :x; /*x là thành phần chứa 8 bit cao của một word*/
}y;
```

□ Ứng dụng

Các cấu trúc có thành phần kiểu field được sử dụng nhằm:

- Tiết kiệm bộ nhớ.
- Dùng trong các khai báo kiểu hợp (union) để lấy ra các bit của một từ nào đó (xem thêm phần UNION để hiểu rõ hơn nhận xét này).

2.1. 2.7. Mảng các cấu trúc

□ Khai báo

Mảng mà các thành phần có kiểu cấu trúc được gọi là mảng cấu trúc. Khai báo một mảng các cấu trúc hoàn toàn tương tự như đối với khai báo một mảng bình thường, chỉ có một điểm khác là thay cho tên các kiểu dữ liệu bình thường là một tên kiểu dữ liệu cấu trúc.

□ Ví dụ về khai báo một mảng có cấu trúc:

```
struct hoc_sinh dshs[100]; /*hoc_sinh là kiểu cấu trúc trong 1.2*/
```

Việc sử dụng các mảng cấu trúc sẽ làm cho việc xử lý một tập hợp các biến cấu trúc trở nên dễ nhìn hơn. Các quy định về mảng cũng được áp dụng đối với mảng các cấu trúc.

3. BÀI TẬP MINH HỌA

Bài 1

*/*Chương trình minh họa kiểu cấu trúc enum (dùng khai báo typedef)*/*

```
#include <stdio.h>
typedef enum light_status {Red, Green, Off};
void test_it(void);
int check_lights(enum light_status condition);
main()
{
    test_it();
}
void test_it()
{
    char input;
    enum light_status reading;
    printf("\n\n1] Red 2] Green 3] Off \n\n");
    printf("Chon kha nang bang so => ");
    input = getchar();
    switch (input)
    {
        case '1': reading = Red;
                break;
        case '2': reading = Green;
                break;
        case '3': reading = Off;
                break;
    } /* End of switch */

    check_lights(reading);
}
int check_lights(enum light_status condition)
{
```

```

switch(condition)
{
    case Red : printf("Kiem tra ap suat nguon.");
                break;
    case Green: printf("He thong OK.");
                break;
    case Off : printf("Kiem tra cau tri he thong.");
                break;
} /* Kết thúc switch */
}

```

Kết quả:
 1] Red 2] Green 3] Off
 Chon kha nang bang so =>2
 He thong OK.

Bài 2

*/*Chương trình minh họa cách nhập dữ liệu vào một cấu trúc */*

```

#include <stdio.h>
main()
{
    struct
    {
        char manufacturer[20]; /* sản phẩm điện trở. */
        int quantity; /* Số lượng chuyển giao. */
        float price_each; /* Giá của mỗi điện trở. */
    } resistors; /* Biến cấu trúc. */

    float total_value; /* Tổng giá trị. */

    /* Hiện thị các biến: */

    /* Nhập tên của sản phẩm: */
    printf("Tên của sản phẩm => ");
    gets(resistors.manufacturer);

    /*Nhập số lượng xuất: */

    printf("Số lượng xuất => ");
    scanf("%d",&resistors.quantity);

    /* Nhập giá của mỗi sản phẩm: */

    printf("Giá của mỗi sản phẩm => ");
    scanf("%f",&resistors.price_each);

    /*Tính tổng giá trị: */

    total_value = resistors.quantity * resistors.price_each;

    /* Xuất các giá trị: */
    printf("\n\n");
    printf("Mục: Điện trở\n\n");
    printf("Sản phẩm: %s\n",resistors.manufacturer);
    printf("Đơn giá: $%f\n",resistors.price_each);
}

```


<pre>printf("Số lượng: %d\n",resistors.quantity); printf("Tổng giá trị: \$%f\n", total_value); }</pre>
<p>Kết quả</p> <p>Tên của sản phẩm => Ohmite Số lượng xuất => 10 Giá của mỗi sản phẩm => 0.05</p> <p>Mục: Điện trở Sản phẩm: Ohmite Đơn giá: \$0.050000 Số lượng: 10 Tổng giá trị: \$0.500000</p>

Bài 3

<pre>/*Chương trình minh họa con trỏ cấu trúc*/ #include <stdio.h> typedef struct { char manufacturer[20]; /*Tên điện trở. */ int quantity; /* Số lượng chuyển giao. */ float price_each; /* Đơn giá. */ } parts_record; main() { parts_record *rcd_ptr; /* Con trỏ cấu trúc. */ float total_value; /* Tổng giá trị. */ /* Nhập tên điện trở: */ printf("Tên của sản phẩm => "); gets(rcd_ptr -> manufacturer); /* Số lượng chuyển giao: */ printf("Số lượng xuất => "); scanf("%d",rcd_ptr -> quantity); /* Nhập đơn giá: */ printf("Giá của mỗi sản phẩm => "); scanf("%f",rcd_ptr -> price_each); /* Tính tổng giá trị: */ total_value = rcd_ptr -> quantity * rcd_ptr -> price_each; /* Xuất: */ printf("\n\n");</pre>

```

printf("Mục:      Điện trở\n\n");
printf("Sản phẩm:  %s\n",rcd_ptr -> manufacturer);
printf("Đơn giá:    $%f\n",rcd_ptr -> price_each);
printf("Số lượng:   %d\n",rcd_ptr -> quantity);
printf("Tổng giá trị: $%f\n", total_value);
}

```

Kết quả

Tên của sản phẩm => Ohmite
Số lượng xuất => 10
Giá của mỗi sản phẩm => 0.05

Mục: Điện trở
Sản phẩm: Ohmite
Đơn giá: \$0.050000
Số lượng: 10
Tổng giá trị: \$0.500000

Bài 4

*/*Chương trình minh họa hàm với kiểu cấu trúc*/*

```

#include <stdio.h>
typedef struct
{
    char manufacturer[20];    /* Tên điện trở. */
    int quantity;             /* Số lượng xuất. */
    float price_each;         /* Đơn giá. */
} parts_record;

parts_record get_input(void); /* Phần nhập dữ liệu. */
void display_output(parts_record resistor_record); /* Phần xuất dữ liệu. */

main()
{
    parts_record resistor_box; /* Khai báo một biến cấu trúc. */
    resistor_box = get_input(); /* Nhập dữ liệu. */
    display_output(resistor_box); /* Xuất dữ liệu. */
}

parts_record get_input(void) /* Nhập dữ liệu. */
{
    parts_record resistor_information;

    /* Nhập tên sản phẩm: */

    printf("Tên của sản phẩm => ");
    gets(resistor_information.manufacturer);

    /* Số lượng xuất: */

    printf("Số lượng xuất => ");
    scanf("%d",&resistor_information.quantity);

    /* Đơn giá: */

    printf("Giá của mỗi sản phẩm => ");

```

```

scanf("%f",&resistor_information.price_each);

return(resistor_information);
}

void display_output(parts_record resistor_information)
{
    /* Xuất:      */

    printf("\n\n");
    printf("Mục:      Điện trở\n\n");
    printf("Sản phẩm:  %s\n",resistor_information.manufacturer);
    printf("Đơn giá:   $%f\n",resistor_information.price_each);
    printf("Số lượng:   %d\n",resistor_information.quantity);
}

```

Kết quả

Tên của sản phẩm => Ohmite
Số lượng xuất => 10
Giá của mỗi sản phẩm => 0.05

Mục: Điện trở
Sản phẩm: Ohmite
Đơn giá: \$0.050000
Số lượng: 10

4. CÁC BÀI TẬP TỰ LÀM

Bài 1

Viết chương trình dùng cấu trúc để ghi ngày sinh, ngày tuyển dụng và bậc lương của nhân viên.

Bài 2

Viết chương trình sắp xếp một danh sách các cấu trúc
Gợi ý:

```

...
typedef struct {
    char Ho[18];
    char Ten[8];
    int  Bacluong;
} Person;
...

void Compare(Person *p, Person *q)
{
    int k;
    k=strcmp(p->Ten,q->Ten);
    if(k==0)
        k=strcmp(p->Ho,q->Ho);
    return k;
}

```

}

...

Bài 3

Viết chương trình nhập một danh sách học sinh bao gồm Họ, Tên, Tuổi, Điểm Toán, Điểm Lý, Điểm Hóa và Điểm Trung Bình. Sắp xếp Tên, Họ theo thứ tự tăng dần. Sau đó xuất ra danh sách này gồm Họ, Tên và Điểm Trung Bình.

BÀI 12

CÁC CẤU TRÚC TỰ TRỞ - KIỂU HỢP

1. CÁC CẤU TRÚC TỰ TRỞ

1.1. Khái niệm

Cấu trúc có ít nhất một thành phần là con trỏ chỉ đến bản thân cấu trúc được gọi là cấu trúc tự trở.

1.2. Các ví dụ về khai báo dữ liệu

Dưới đây đưa ra một số ví dụ về cấu trúc tự trở:

Ví dụ

```
struct h_sinh{  
    char ho_ten[20];  
    float diem;  
    struct h_sinh *sau; /*con trỏ chỉ đến học sinh tiếp theo trong danh sách*/  
};
```

khai báo này định nghĩa một cấu trúc tự trở có thể dùng để quản lý danh sách họ tên học sinh cùng điểm số của họ (trước đây, ở ví dụ BÀI 11 chúng ta đã dùng một mảng số để quản lý danh sách này). Như sau này các bạn thấy, cách quản lý mới bằng cấu trúc tự trở linh hoạt hơn so với cách quản lý bằng mảng. Hạn chế của cách quản lý trong trường hợp này là chỉ duyệt được danh sách theo một chiều nhất định. Để khắc phục nhược điểm này, chúng ta sử dụng khai báo sau:

Ví dụ

```
struct h_sinh{  
    char ho_ten[20];  
    float diem;  
    struct h_sinh *truoc, *sau;  
};
```

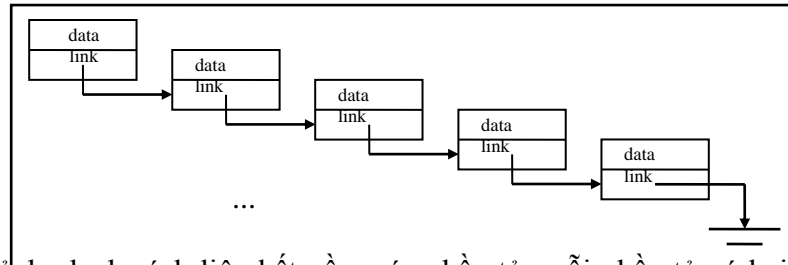
trong câu lệnh khai báo này, nhờ hai thành phần con trỏ trước và sau tương ứng chỉ địa chỉ cấu trúc chứa thông tin về các học sinh đứng liền trước và liền sau trong danh sách, chúng ta có thể dễ dàng duyệt danh sách theo cả hai chiều "tiền" hoặc "lui".

Cấu trúc tự trở là một cơ chế linh hoạt để xây dựng danh sách liên kết (móc nối).

1.3. Danh sách liên kết

□ Định nghĩa cấu trúc

Khi cần quản lý một danh sách nào đó, nói chung thường xuyên phải cập nhật, thêm, bớt các thành viên của danh sách. Nếu sử dụng mảng, sẽ rất tốn kém (khi số thành phần mảng vượt quá số thành viên của danh sách) hoặc là không đủ chỗ để chứa hết các thành phần mới thêm vào. Trái lại bằng danh sách liên kết, mặc dù phải thêm các trường con trỏ trong thành phần cấu trúc, nhưng bù lại danh sách của chúng ta có thể dài tùy ý (tùy theo khả năng của bộ nhớ) và hơn thế nữa do sử dụng cấp phát động chúng ta sẽ không bị lãng phí bộ nhớ như trong trường hợp sử dụng mảng.



Một cách hình ảnh, danh sách liên kết gồm các phần tử, mỗi phần tử có hai vùng chính: vùng dữ liệu danh sách và vùng liên kết. Vùng liên kết là một hoặc nhiều con trỏ chỉ đến các phần tử trước hoặc sau phần tử đang được xem xét tùy thuộc vào yêu cầu của công việc cụ thể. Hình vẽ trên minh họa một kiểu danh sách liên kết.

Cú pháp chung cho khai báo danh sách liên kết sử dụng kiểu dữ liệu con trỏ như sau:

```
typedef struct kiểu_dữ_liệu{  
    <khai báo phần dữ liệu>  
    <khai báo các con trỏ liên kết>  
}t_kiểu_dữ_liệu;
```

Dòng **typedef struct** kiểu_dữ_liệu t_kiểu_dữ_liệu định nghĩa một kiểu dữ liệu mới. Trong kiểu dữ liệu này có hai phần, phần đầu tiên khai báo các trường thông thường, phần thứ hai là các con trỏ chỉ đến chính các kiểu dữ liệu đó. Sau đây chúng ta phân tích một chương trình ví dụ minh họa cách dùng danh sách móc nối quản lý một lớp học gồm tên học sinh và điểm thi của từng người.

□ Bài tập mẫu

Xây dựng chương trình quản lý họ tên học sinh và điểm thi học kỳ của từng người. Danh sách học sinh được sắp theo thứ tự alphabet.

□ Phân tích đưa ra phương pháp Giải bài tập

Chúng ta định nghĩa cấu trúc như sau:

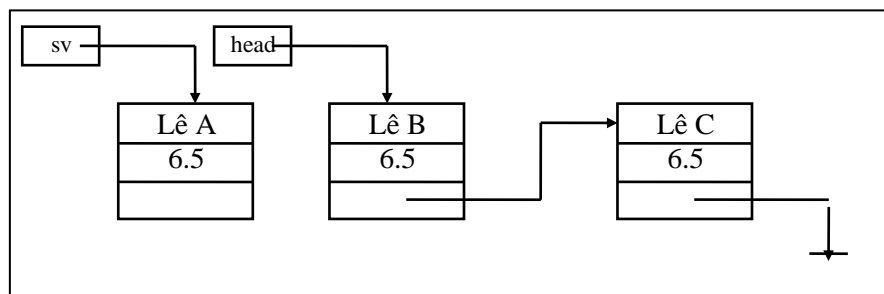
```
typedef struct sinh_vien{  
    char ho_ten[20];  
    float diem;  
    struct sinh_vien *tiep;  
}t_sinh_vien;  
t_sinh_vien *head; /*con trỏ đến đầu danh sách*/
```

Mỗi phần tử chứa thông tin của một học sinh và một con trỏ chỉ tới địa chỉ của cấu trúc chứa các thông tin của học sinh tiếp theo trong danh sách. Danh sách kết thúc bằng một thành phần NULL. Như vậy ban đầu danh sách (đầu) được gán bằng NULL chỉ ra rằng chưa có học sinh nào được nhập.

Mỗi khi có một học sinh được nhập vào, nghĩa là có một thành phần mới được tạo ra, chúng ta phải xin cấp phát một vùng bộ nhớ có kích thước đủ để chứa phần tử đó. Toán tử `sizeof(tên_kiểu)` cho ta biết kích thước cần thiết để cấp phát cho một biến có kiểu là `tên_kiểu`. Tất nhiên khi cấp phát không quên kiểm tra xem liệu có còn đủ bộ nhớ hay không, điều này là cần thiết để chương trình không bị treo.

```
#define SIZESV sizeof(t_sinh_vien)
t_sinh_vien *sv;
sv=NULL;
if((sv=(t_sinh_vien *)malloc(SIZESV))==NULL){
    printf("\nkhong con du bo nho de cap phat");
    getch();
}
```

Chú ý, cần gán `sv=NULL` để phòng trường hợp `sv` đang trỏ tới một thành phần

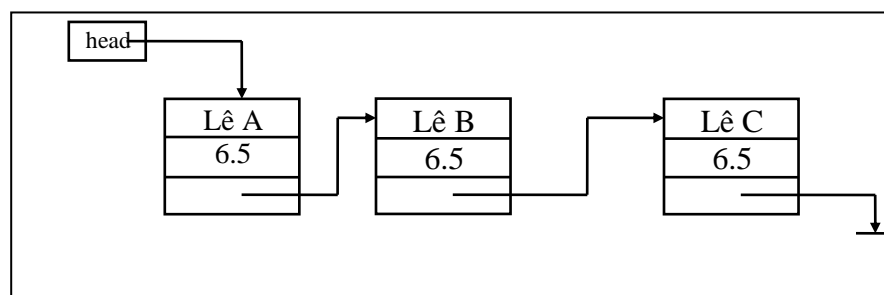


cấu trúc nào đó trong danh sách.

Danh sách trước khi thêm phần tử mới

Khi thêm phần tử mới vào danh sách, chương trình sẽ tự động tìm vị trí thích hợp cho phần tử. Chúng ta sử dụng hàm `strcmp()` để thực hiện các phép so sánh họ tên của họ tên học sinh mới nhập vào với họ tên các học sinh trong danh sách (chú ý rằng danh sách của chúng ta luôn luôn được sắp xếp theo vần alphabet của họ tên các học sinh). Các trường hợp có thể xảy ra khi thêm một phần tử mới vào danh sách:

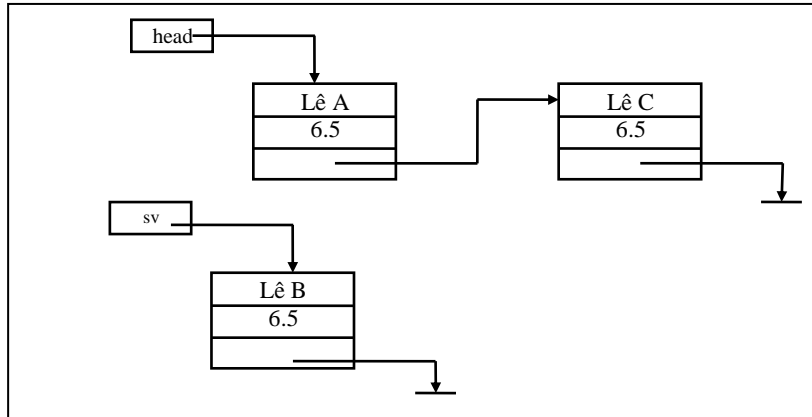
Nếu phần tử mới ở đầu danh sách, cần phải sửa lại con trỏ `head`.



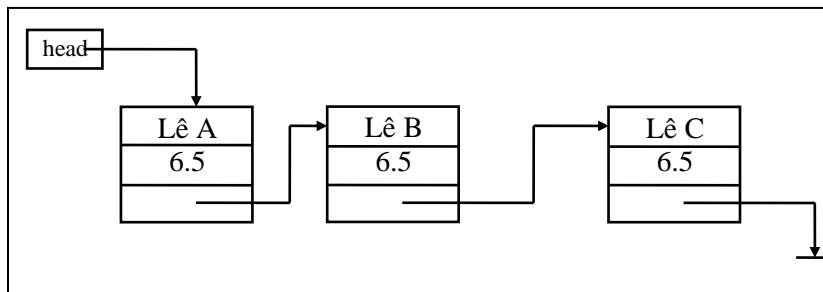
Danh sách sau khi chèn phần tử mới vào đầu.

Nếu phần tử đó đã có (hai tên trùng nhau) phải có sự lựa chọn: liệu có phải hai học sinh khác nhau hay chỉ là một.

Trong các trường hợp khác cần phải sửa lại con trỏ các thành phần một cách trực quan như sau:



Danh sách trước khi chèn



Danh sách sau khi thêm sv

Khi loại bỏ, công việc được thực hiện theo chiều ngược lại. Chúng ta cũng cần phải phân biệt các trường hợp khi danh sách rỗng hoặc phần tử cần loại bỏ nằm ở đầu danh sách.

□ **Chương trình sau minh họa những vấn đề được phân tích ở trên.**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
typedef struct hoc_sinh {
    char ho_ten[20];
    float diem;
    struct hoc_sinh *tiep;
} t_hoc_sinh;
t_hoc_sinh *head=NULL;
```

```

#define SIZESV sizeof(t_hoc_sinh)

main()
{
    t_hoc_sinh *sv,*p,*q;
    char name[20];
    float d;
    char ch;
    int n=0; /*Ban dau chua co hoc sinh nao*/
    clrscr();
    /*Nhập vào thông tin các học sinh cho đến khi nhập vào một tên rỗng */
    do {
        printf("Nhap vao thong tin cua hoc sinh %d\n",n+1);
        fflush(stdin); /*Xóa đệm bàn phím, cần thiết khi nhập x(c)u*/
        printf(" Ho ten : ");
        gets(name);
        if(strcmp(name,"")!=0)
        {
            if ((sv=(t_hoc_sinh *) (malloc(SIZESV)))==NULL)
            {
                printf("Khong du bo nho\n");
                break;
            }
            else
            {
                n++;
                strcpy(sv->ho_ten,name);
                printf(" Diem : ");
                scanf("%f",&d);
                sv->diem = d;
                sv->tiep =NULL;
                /*chèn phần tử mới vào danh sách*/
                if (head == NULL)
                    head=sv;
                else {
                    p=head;
                    while(p!=NULL&&strcmp(p->ho_ten,sv->ho_ten) <0)
                    {
                        q=p;
                        p=p->tiep;
                    }
                    if (p!=NULL && strcmp(p->ho_ten,sv->ho_ten)==0)
                    {
                        do {
                            fflush(stdin);
                            printf("Co hai hoc sinh trung ten(D/S)?");

```



```

        } while (toupper(ch=getchar())!='D' && toupper(ch)!='S');
    if (toupper(ch) == 'S') {
        free(sv);
        n--;
        continue;
    }
    else while(p!=NULL&&strcmp(p->ho_ten,sv->ho_ten)==0)
    {
        q=p;
        p=p->tiep;
    }
    if (p==head)
    {
        sv->tiep = head;
        head=sv;
    }
    else {
        q->tiep = sv;
        sv->tiep=p;
    }
}
}
}while (strcmp(name , "")!=0);

/*Duyệt lại danh sách và in ra danh sách sinh viên thi lại*/
p=head;
while(p!=NULL && p->diem >=5.0)
{
    if (head==p)
        head=q=p->tiep;
    else {
        q->tiep=p->tiep;
        q=p->tiep;
    }
    free(p);
    p=q;
    while(p!=NULL && p->diem <5.0)
    {
        q=p;
        p=p->tiep;
    }
}
if (head==NULL)

```

```

        printf("Khong co hoc sinh thi lai!\n");
else{
    printf("Danh sach sinh vien phai thi lai\n\n");
    n=0;
    p=head;
    while(p!=NULL)
    {
        printf("%3d. %20s %6.1f\n", ++n, p->ho_ten, p->diem);
        p=p->tiep;
    }
    printf("\nCo tong so %d hoc sinh phai thi lai", n);
}
/*Giải phóng bộ nhớ trước khi kết thúc chương trình*/
p=head;
while (p!=NULL)
{
    q=p->tiep;
    free(p);
    p=q;
}
getch();
}

```

Kết quả

Nhap vao thong tin cua hoc sinh 1

Ho ten : Nguyen Bich Thao

Diem : 9

Nhap vao thong tin cua hoc sinh 2

Ho ten : Pham Hai Yen

Diem : 4

Nhap vao thong tin cua hoc sinh 3

Ho ten : Nguyen Van Anh

Diem : 9

Nhap vao thong tin cua hoc sinh 4

Ho ten : Do thanh Xuan

Diem : 3

Nhap vao thong tin cua hoc sinh 5

Ho ten : Do bich Ha

Diem : 7

Nhap vao thong tin cua hoc sinh 6

Ho ten :

Danh sach sinh vien phai thi lai

1. Do thanh Xuan 3.0

2. Pham Hai Yen 4.0

2. 2. KIỂU HỢP (UNION)

2.1. Khái niệm

Một biến kiểu **union** cũng bao gồm nhiều thành phần giống như một biến cấu trúc, nhưng khác biến cấu trúc ở chỗ: các thành phần của biến cấu trúc được cấp phát các vùng nhớ khác nhau, còn các thành phần của biến **union** được cấp phát chung một vùng nhớ. Độ dài của vùng nhớ đủ để chứa được thành phần dài nhất trong **union**. Khai báo một **union** được thực hiện nhờ từ khóa **union**.

2.2. Định nghĩa một cấu trúc union - Truy nhập đến thành phần của union

Việc định nghĩa một biến kiểu **union**, mảng các **union**, con trỏ **union**, cũng như cách thức truy nhập đến các thành phần của biến **union** được thực hiện hoàn toàn tương tự như đối với các cấu trúc. Một cấu trúc có thể có thành phần union và ngược lại các thành phần của union lại có thể là cấu trúc. Dưới đây là hai thí dụ về khai báo **union**.

Ví dụ

```
typedef union{
    unsigned int ival;
    float fval;
    unsigned char ch[2];
}val;
val a,b,x[10];
```

Câu lệnh trong ví dụ trên định nghĩa kiểu **union** **val** gồm ba thành phần là **ival**, **fval** và mảng ký tự **ch**. Độ dài của **val** bằng độ dài của trường **fval** và bằng 4. Tiếp đó khai báo các biến **union** **a**, **b** và mảng các **union** **x**.

Phép gán

```
a.ival=0xa1b2;
```

một số hệ 16 là 0xa1b2 cho thành phần **ival** của **a**. Do **ival** chỉ chiếm hai byte đầu của **union**, nên sau câu lệnh trên ta có:

```
a.ch[0]=0xa1 và a.ch[1]=0xb2
```

như vậy ta đã dùng khai báo **union** để tác ra byte cao và thấp của một số nguyên.

2.3. Ví dụ khai báo một union có thành phần là cấu trúc

Ví dụ

```
struct ng{
    int ngay;
    int thang;
    int nam;
```

```

};
struct diachi{
    int sonha;
    char *tenpho;
};
union u{
    struct ng date;
    struct diachi address;
}diachi_ngaysinh;

```

Ví dụ này khai báo một **union** có thành phần là cấu trúc.

3. CÁC BÀI TẬP MINH HỌA

Bài 1

/ Minh họa cách khởi tạo một danh sách liên kết */*

```

#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    char data;
    struct node *link;
} LNODE;

void show_list(LNODE *ptr);

main()
{
    LNODE *n1, *n2, *n3;

    n1 = malloc(sizeof(LNODE));
    n2 = malloc(sizeof(LNODE));
    n3 = malloc(sizeof(LNODE));

    n1->data = 'c';
    n1->link = n2;
    n2->data = 'a';
    n2->link = n3;
    n3->data = 't';
    n3->link = NULL;

    printf("Danh sách liên kết sẽ là: ");
    show_list(n1);
}

void show_list(LNODE *ptr)
{
    while(ptr != NULL)
    {
        printf("%c", ptr->data);
    }
}

```

```

        ptr = ptr->link;
    }
    printf("\n");
}

```

Kết quả
 Danh sách liên kết sẽ là:cat

Bài 2

/ Minh họa cách nhập dữ liệu vào danh sách liên kết từ người sử dụng */*

```

#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    char data;
    struct node *link;
} LNODE;

void show_list(LNODE *ptr);
void add_node(LNODE **ptr, char item);

main()
{
    LNODE *n1 = NULL;
    char item;

    do
    {
        printf("\nNhập dữ liệu: ");
        item = getche();
        if(item != '\r')
            add_node(&n1,item);
    } while(item != '\r');
    printf("\nDanh sách liên kết mới sẽ là : ");
    show_list(n1);
}

void show_list(LNODE *ptr)
{
    while(ptr != NULL)
    {
        printf("%c",ptr->data);
        ptr = ptr->link;
    }
    printf("\n");
}

void add_node(LNODE **ptr, char item)
{
    LNODE *p1, *p2;

    p1 = *ptr;

```

```

if(p1 == NULL)
{
    p1 = malloc(sizeof(LNODE));
    if(p1 != NULL)
    {
        p1->data = item;
        p1->link = NULL;
        *ptr = p1;
    }
}
else
{
    while(p1->link != NULL)
        p1 = p1->link;
    p2 = malloc(sizeof(LNODE));
    if(p2 != NULL)
    {
        p2->data = item;
        p2->link = NULL;
        p1->link = p2;
    }
}
}

```

Kết quả

Nhập dữ liệu:c

Nhập dữ liệu:p

Nhập dữ liệu:u

Nhập dữ liệu:

Danh sách liên kết mới sẽ là :cpu

Bài 3

*/*Chương trình minh họa kiểu cấu trúc Union*/*

```

#include <stdio.h>
main()
{
    union /* Định nghĩa union. */
    {
        int integer_value;
        float value;
    } integer_or_float;
    printf("Kích cỡ kiểu union=> %d bytes.\n", sizeof(integer_or_float));

    /* Nhập một số và hiện thị nó: */

    integer_or_float.integer_value = 123;
    printf("Giá trị của số nguyên= %d\n", integer_or_float.integer_value);
    printf("Địa chỉ bắt đầu là => %d\n", &integer_or_float.integer_value);

    /* Nhập một số thực và hiện thị nó: */

    integer_or_float.value = 123.45;
    printf("Giá trị của số thực là %f\n", integer_or_float.value);
}

```

<pre>printf("Địa chỉ bắt đầu là => %d\n", &integer_or_float.value); }</pre>
<p>Kết quả:</p> <p>Kích cỡ kiểu union=> 4</p> <p>Giá trị của số nguyên= 123</p> <p>Địa chỉ bắt đầu là => 7042</p> <p>Giá trị của số thực là 123.45</p> <p>Địa chỉ bắt đầu là => 7042</p>

4. CÁC BÀI TẬP TỰ LÀM

Bài 1

Viết chương trình nhập thêm một nút vào đầu danh sách.

Bài 2

Viết chương trình nhập thêm một nút vào cuối danh sách.

Bài 3

Viết chương trình loại bỏ một nút ở đầu danh sách

Bài 4

Viết chương trình loại bỏ một nút ở cuối danh sách

BÀI 13

ĐỆ QUI

1. 1. KHÁI NIỆM CHUNG VỀ ĐỆ QUI

1.1. Khái niệm

Các chương trình mà chúng ta đã xem xét đều có chung cấu trúc dạng hàm gọi các hàm khác dưới dạng mô hình phân cấp. Tuy nhiên đối với một số bài toán, việc dùng hàm gọi ngay chính nó rất hữu dụng. Có thể định nghĩa hàm đệ qui là hàm sẽ gọi đến chính nó trực tiếp hay gián tiếp thông qua các hàm khác. Vấn đề đệ qui là một vấn đề rất phức tạp, là chủ đề của nhiều khoá học nâng cao trong tin học, vì vậy trong phần này chỉ giới thiệu những khía cạnh cùng với những ví dụ đơn giản nhất của vấn đề đệ qui.

Trước tiên ta xem xét khái niệm đệ qui, sau đó kiểm tra trên một vài chương trình có chứa các hàm đệ qui. Cách tiến hành giải một bài toán đệ qui nhìn chung có những điểm chung sau.

Trước tiên gọi hàm đệ qui để giải bài toán, hàm đệ qui thực ra chỉ biết cách giải bài toán trong trường hợp đơn giản nhất (hay còn gọi là trường hợp cơ sở). Nếu hàm đệ qui được gọi trong trường hợp cơ sở, hàm chỉ cần đơn giản trả lại kết quả. Nếu hàm được gọi trong các trường hợp phức tạp hơn, hàm đệ qui sẽ chia công việc cần giải quyết thành hai phần. Một phần hàm biết cách giải quyết như thế nào, còn phần kia vẫn không biết cách giải quyết như thế nào tuy nhiên để được gọi là có khả năng đệ qui, phần sau phải giống với bài toán ban đầu nhưng đơn giản hơn hay nhỏ hơn bài toán ban đầu. Bởi vì bài toán mới giống với bài toán ban đầu nên hàm sẽ thực hiện gọi chính nó để giải quyết công việc đơn giản hơn này - đây chính là lời gọi đệ qui hay còn gọi là một bước đệ qui. Để đảm bảo việc đệ qui có kết thúc, mỗi một lần gọi đệ qui thì bài toán phải đảm bảo đơn giản hơn và các bước đệ qui này còn thực hiện tiếp cho đến khi nào bài toán đơn giản dần, đơn giản tới mức trở thành trường hợp cơ sở. Có thể nhận thấy hàm đệ qui xử lý trường hợp cơ sở để trả lại kết quả tính được cho các hàm mức phức tạp hơn, rồi đến lượt các hàm này lại tính trả lại kết quả cho các hàm phức tạp hơn nữa ... cứ như vậy cho đến lời gọi hàm ban đầu.

1.2. Ví dụ minh họa

Mới nghe có vẻ lạ lùng khi so sánh với cách giải quyết các bài toán thông thường. Nhưng trước khi nhận xét tiếp ta xem xét ví dụ tính n giai thừa ($n!$).

Theo định nghĩa $n!$ bằng tích của tất cả các số tự nhiên từ 1 đến n : $n! = n * (n-1) * \dots * 2 * 1$

ví dụ $5! = 5 * 4 * 3 * 2 * 1 = 120$, $0!$ được định nghĩa bằng 1.

Để tính $n!$ có thể dùng vòng lặp như sau :

```
factorial = 1;
for (i = n; i >= 1; i--)
    factorial *= i;
```

Tuy nhiên cũng có thể định nghĩa đệ qui hàm giai thừa như sau :

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

Khi đó để tính $3!$ quá trình tính phải lần ngược trở về tính $0!$, sau đó lấy kết quả để tính $1!$, lại lấy kết quả để tính $2!$ và cuối cùng mới tính được $3!$:

```
3! = 3*(2!) = 3*(2*(1!)) = 3*(2*(1*(0!)))
3! = 3*(2*(1*1))) = 3*(2*1) = 3*2 = 6
```

```
/* Tính giai thừa theo phương pháp đệ qui */
#include <stdio.h>
```

```
long factorial( long ); /* Hàm nguyên mẫu */
```

```
main()
{
    int i;
```

```
    for ( i=0; i<=10; i=i+2)
        printf("%2d! = %ld\n", i, factorial(i));
```



```

}

/* Định nghĩa đệ qui hàm factorial */
long factorial( long number)
{
    if ( number = 0 )
        return 1;
    else
        return (number*factorial(number-1));
}

```

Kết quả

$0! = 1$
 $2! = 2$
 $4! = 24$
 $6! = 720$
 $8! = 40320$
 $10! = 3628800$

Chương trình trên : Tính giai thừa theo phương pháp đệ qui

Một ví dụ thứ hai dùng đệ qui là tính dãy số Fibonacci

Dãy số Fibonacci gồm những số

0, 1, 1, 2, 3, 5, 8, 13, 21 ...

bắt đầu từ hai số 0 và 1, tiếp sau đó các số Fibonacci sau bằng tổng của 2 số Fibonacci trước nó.

Dãy Fibonacci gặp rất nhiều trong thực tế, đặc biệt khi mô tả các mô hình xoắn ốc. Tỉ số giữa 2 số Fibonacci liên tiếp khoảng 1,618... Số này gặp rất nhiều trong thực tế và được gọi là tỉ số vàng. Các kiến trúc sư thường thiết kế kích thước cửa sổ, phòng, nhà ở có tỉ lệ giữa chiều dài và chiều rộng bằng tỉ số vàng. Các bư thiếp cũng thường có tỉ lệ giữa chiều dài và chiều rộng bằng tỉ số vàng.

Dãy Fibonacci có thể định nghĩa đệ qui như sau:

fibonacci(0) = 0; fibonacci(1) = 1;

fibonacci(n) = fibonacci(n-1) + fibonacci(n-2) (n>1) Chương trình trong

hình dưới đây tính đệ qui số Fibonacci thứ i bằng cách dùng hàm **fibonacci**. Chú ý rằng dãy số Fibonacci tăng rất nhanh, vì vậy chúng ta chọn kiểu long cho tham số và giá trị trả về của hàm fibonacci.

```

/* Tính dãy số Fibonacci phương pháp đệ qui */
#include <stdio.h>
long fibonacci( long ); /* Hàm nguyên mẫu */
main()
{
    long result, number;
    printf("Hãy nhập vào một số nguyên : ");
}

```

```
scanf("%ld", &number);
result = fibonacci(number);
printf("Fibonacci thứ %ld là : %ld\n", number, result);
return 0;
}
```

```
/* Định nghĩa đệ qui hàm fibonacci */
long fibonacci( long n)
{
    if ( n = 0 || n = 1 )
        return n;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Kết quả

Hãy nhập vào một số nguyên : 0
Fibonacci thứ 0 là : 0

Hãy nhập vào một số nguyên : 1
Fibonacci thứ 1 là : 1

Hãy nhập vào một số nguyên : 2
Fibonacci thứ 2 là : 1

Hãy nhập vào một số nguyên : 3
Fibonacci thứ 3 là : 2

Hãy nhập vào một số nguyên : 4
Fibonacci thứ 4 là : 3
Hãy nhập vào một số nguyên : 5
Fibonacci thứ 5 là : 5

Hãy nhập vào một số nguyên : 6
Fibonacci thứ 8 là : 8

Hãy nhập vào một số nguyên : 10
Fibonacci thứ 10 là : 55

Hãy nhập vào một số nguyên : 20
Fibonacci thứ 20 là : 6765

Hãy nhập vào một số nguyên : 35
Fibonacci thứ 35 là : 9227465

Chương trình trên: Tạo ra các số Fibonacci bằng đệ qui.

Mỗi một lần hàm fibonacci được gọi, nó kiểm tra ngay lập tức xem liệu đây có phải là trường hợp cơ sở, n bằng 0 hay 1 hay không. Nếu đúng n được trả lại, bằng không, tức là khi $n > 1$, nó sẽ tạo ra hai lời gọi đệ qui đơn giản hơn lời gọi đến hàm fibonacci ban đầu.

1.3. So sánh đệ qui và lặp

Ta đã xem xét cách tính giai thừa của một số, có thể tính theo phương pháp lặp (phi đệ qui) hoặc phương pháp đệ qui. Trong phần này ta sẽ so sánh hai phương pháp này và tìm hiểu tại sao trong thực tế thường chọn phương pháp lặp hơn.

Lập trình đệ qui sử dụng cấu trúc lựa chọn, còn lặp sử dụng cấu trúc lặp. Cả hai phương pháp đều liên quan đến quá trình lặp, tuy nhiên phương pháp lặp sử dụng vòng lặp tường minh, còn phương pháp đệ qui có được quá trình lặp bằng các sử dụng liên tục các lời gọi hàm. Cả hai phương pháp đều phải kiểm tra khi nào thì kết thúc. Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai, còn phương pháp đệ qui kết thúc khi đến trường hợp cơ sở, lặp thay đổi biến đếm trong vòng lặp cho đến khi nó làm cho điều kiện lặp sai còn đệ qui làm cho các lời gọi hàm đơn giản dần cho đến khi đơn giản tới trường hợp cơ sở. Cả hai phương pháp đều có thể dẫn đến trường hợp chạy vô hạn mãi, lặp sẽ không thoát ra được khi điều kiện lặp không bao giờ sai còn đệ qui không thoát ra được khi các bước đệ qui không làm cho bài toán đơn giản hơn và cuối cùng hội tụ về trường hợp cơ sở.

Tuy nhiên đệ qui tồi hơn, nó liên tục đưa ra các lời gọi hàm làm tốn thời gian xử lý của bộ vi xử lý và không gian nhớ. Mỗi lần gọi hàm, lại cần thêm một bản sao của hàm, tốn thêm bộ nhớ (lưu các biến của hàm, địa chỉ trở về của hàm ...). Vì vậy phương pháp lặp được chuộng hơn. Tuy nhiên tồn tại nhiều bài toán chỉ có thể giải bằng đệ qui.

2. CÁCH DÙNG ĐỆ QUI

2.1. Các bài toán có thể dùng 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. Ta gọi đây 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. Và sau một số hữu hạn bước biến đổi Đệ Qui sẽ dẫn đến trường hợp suy biến.

Nhận xét: Bài toán tính $n!$ ở trên thể hiện rất rõ các đặc điểm này.

2.2. Cách xây dựng hàm Đệ Qui

Hàm Đệ Qui thường được viết theo thuật toán sau:

```
if ( trường hợp suy biến)
{
```

```

trình bày cách giải bài toán
(giả định đã có cách giải)
}
else /* trường hợp tổng quát*/
{
gọi đệ qui tới hàm (đang lập) với
giá trị khác của tham số
}

```

3. MỘT VÀI ỨNG DỤNG CỦA ĐỆ QUI

3.1. Bài toán ước số chung lớn nhất của hai số nguyên dương x và y

- ☐ Trường hợp suy biến là trường hợp $x=y$. Khi đó $\text{usc}(x,y)=x$.
- ☐ Trường hợp chung x khác y có thể đệ qui như sau:
 $\text{usc}(x,y)=\text{usc}(x-y,y)$ nếu $x>y$
 $\text{usc}(x,y)=\text{usc}(x,y-x)$ nếu $x<y$

Hàm tìm ước số chung lớn nhất có thể viết như sau:

```

int usc(int x, int y)
{
if (x==y) return x;
else
    if (x>y) return usc(x-y,y);
    else return usc(x,y-x);
}

```

3.2. Bài toán tháp Hà nội

Bài toán tháp Hà nội được đặt ra như sau: Có một tháp m tầng đang đặt tại vị trí (x_1, y_1) , cần chuyển đến vị trí mới (x_2, y_2) . Khi chuyển cho phép dùng vị trí trung gian (x_3, y_3) .

Qui ước như sau:

- ☐ Số tầng m lớn hơn hoặc bằng 1.
- ☐ Số thứ tự tầng đánh từ 1 đến m từ đỉnh xuống đáy.
- ☐ Mỗi tầng được biểu diễn bằng một hình chữ nhật có chiều cao là một đơn vị ký tự màn hình, và chiều rộng là một số lẻ đơn vị ký tự. Tầng dưới rộng hơn tầng trên.
- ☐ Tọa độ hiểu là tọa độ màn hình hình ở chế độ văn bản.
- ☐ Tâm tháp là tâm của tầng cuối. Như vậy (x_1, y_1) là tâm tháp ban đầu.

Yêu cầu xây dựng qui trình chuyển tháp theo nhiều bước. Mỗi bước chuyển một tầng từ vị trí này đến vị trí khác. Ngoài ra không được đặt tầng lớn đè lên tầng nhỏ.

Thuật toán có thể diễn đạt như sau:

- Trường hợp suy biến: $m=1$, khi đó chỉ cần chuyển tầng 1 từ $(x1,y1)$ đến $(x2,y2)$.
- Trường hợp tổng quát $m>1$ có thể giải quyết đệ qui như sau:
 - Chuyển tháp $m-1$ tầng từ $(x1,y1-1)$ đến $(x3,y3)$, dùng $(x2,y2)$ làm vị trí trung gian.
 - Chuyển tầng m từ $(x1,y1)$ đến $(x2,y2)$.
 - Chuyển tháp $m-1$ tầng từ $(x3,y3)$ đến $(x2,y2-1)$, dùng $(x1,y1)$ làm vị trí trung gian.

Theo thuật toán này có thể lập hàm chuyển tháp và chương trình như sau. Chương trình sẽ in ra qui trình vận chuyển.

```

/* Tháp Hà nội*/
#include<stdio.h>
#include<conio.h>
void cthap(int m, int x1, int y1, int x2, int y2, int x3, int y3)
{
    if (m<1)
        return;
    else
        if (m==1)
            printf("\n chuyen tang1 tu (%d,%d) den (%d,%d)",x1,y1,x2,y2);
        else
        {
            cthap(m-1,x1,y1-1,x3,y3,x2,y2);
            printf("\n chuyen tang %d tu (%d, %d) den (%d,%d)",m,x1,y1,x2,y2);
            cthap(m-1,x3,y3,x2,y2-1,x1,y1);
        }
}
void main()
{
    int m;
    printf("\n so tang: "); scanf("%d",&m);
    cthap(m,10,24,30,24,50,24);
}

```

3.3. Các vòng for lồng nhau và bài toán tổ hợp

Bài toán tổ chức m chu trình lồng nhau có đặc trưng đệ qui. Tại chu trình $k < m$ sẽ gọi tới chu trình $k+1$. Tại chu trình m (trường hợp suy biến) sẽ thực hiện một công việc tùy theo yêu cầu của mỗi bài toán. Xét hàm tạo một vòng lặp.

Hàm này có 3 đối là:

- ☐ Số hiệu vòng lặp k;
- ☐ Cận dưới của biến điều khiển d;
- ☐ Cận trên của biến điều khiển t.

Hàm có thể xây dựng theo lược đồ:

```
void chu_trinh(int k, int d, int t)
{
    int i; /* Biến điều khiển */
    for(i=d; i<=t; i++)
        if (k==m)
        {
            /* là công việc tùy thuộc vào yêu cầu bài toán */
        }
    else
    {
        /* Gọi đệ qui, chú ý d(i) và t(i) phụ thuộc vào i */

        chu_trinh(k+1, d(i), t(i));
    }
}
```

Xét bài toán liệt kê tổ hợp chập m của n số : 1,2,...,n. Một tổ hợp i_1, i_2, \dots, i_m cần thỏa mãn điều kiện:

$$1 \leq i_1 < i_2 < i_3 \dots < i_m \leq n$$

Ta dùng mảng ngoài cs[20] để chứa tổ hợp theo cách:

cs[1] chứa phần tử thứ nhất i_1

cs[2] chứa phần tử thứ hai i_2

...

Khi $m=3$ thì bài toán được giải quyết bằng 3 vòng for lồng nhau như sau:

```
k=1;
for( i1=1; i1<=n-2; i1++)
{
    ck[k]=i1; ++k;
    for(i2=i1+1; i2<=n-1; i2++)
    {
        cs[k]=i2; ++k;
        for(i3=i2+1; i3<=n; i3++)
        {
            cs[k]=i3;
            /* Do k=3; in cs[1], ...,cs[k] */
        }
    }
}
```

Trong trường hợp tổ hợp chập m, ta phải tổ chức m chu trình lồng nhau. Muốn vậy phải dùng hàm:

```
void chu_trinh(int k, int d, int t)
```

Với bài toán tổ hợp thì:

- ☐ Khi $k=1$, cận dưới $d=1$;
- ☐ Tại vòng lặp k cận trên bằng $n-m+k$, vì vậy có thể bỏ đối t.
- ☐ $d(i) = i+1$.

Từ trên, có thể viết hàm chu_trinh cho bài toán tổ hợp:

```
#include<stdio.h>
int m,n,cs[20];
void chu_trinh(int k, int d)
{
    int i; /*Bien dieu khien*/
    for (i=d; i<=n-m+k;++i)
    {
        cs[k]=i;
        if (k==m)
        {
            /*In mot to hop*/
            int j;
            printf("\n");
            for(j=1;j<=k;++j)
                printf("%d",cs[j]);
        }
        else chu_trinh(k+1,i+1);
    }
}
```

hàm chu_trinh sẽ được áp dụng trong main như sau:

```
/*Liet ke to hop*/
#include<conio.h>
#include<stdio.h>
void main()
{
    clrscr();
    printf("\n Vao m va n (m<=n)");
    scanf("%d%d",&m,&n);
    chu_trinh(1,1);
    getch();
}
```

3.4. Một bài toán ứng dụng tổ hợp

Cho dãy n số thực hãy tìm m số có tích lớn nhất. Một bộ m số chính là tổ hợp chập m của n số, vì vậy đây chính là bài toán duyệt các tổ hợp. Khi nhận được một tổ hợp ta cần tính tích của chúng và so sánh với các giá trị trước đó để chọn tích lớn nhất. Ta đưa các biến, mảng ngoài như sau:

- ☐ Mảng thực $x[20]$ chứa dãy số thực
- ☐ Mảng nguyên $cs[20]$ chứa các tổ hợp được duyệt,
- ☐ Mảng nguyên $csmx[20]$ chứa tổ hợp cần tìm (có tích lớn nhất),
- ☐ Biến thực max chứa tích lớn nhất.

Hàm `chu_trinh` được sửa đổi chút: khi $k=m$ (tức là nhận được một tổ hợp) thì tính tích của chúng và so sánh với phương án cũ để tìm tích lớn hơn.

Phương án ban đầu chính là tổ hợp gồm các chỉ số từ 1 đến m được xây dựng trong hàm `main`.

Dưới đây là chương trình tìm bộ m phần tử (trong dãy n số) có tích lớn nhất.

```
/* tìm bộ m phần tử (trong n số) có tích lớn nhất*/
```

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

```
#include<conio.h>
```

```
int m,n,cs[20], csmx[20];
```

```
float max, x[20];
```

```
void chu_trinh(int k, int d)
```

```
{
```

```
    int i;
```

```
    for(i=d;i<=n-m+k;++i)
```

```
        {
```

```
            cs[k]=i;
```

```
            if(k==m)
```

```
            {
```

```
                int j; float s=1.0;
```

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

```
                    s*=x[cs[j]];
```

```
                if (s>max)
```

```
                {
```

```
                    max=s;
```

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

```
                    csmx[j]=cs[j];
```

```
                }
```

```
            }
```

```
            else
```

```
                chu_trinh(k+1,i+1);
```

```
        }
```

```
}
```

```
void main()
```

```
{
```



```

    int j;
    clrscr();
    printf("\n vào m và n(m<=n)");
    scanf("%d%d",&m,&n);
    for(j=1;j<=n;++j)
    {
        printf("\n x[%d]=",j); scanf("%f",&x[j]);
    }
    max=1;

    for(j=1; j<=m;++j)
    {
        csmax[j]=j;
        max*=x[j];
    }
    chu_trinh(1,1);
    for(j=1;j<=m;++j)
    {
        printf("\n x[%d]=%f", csmax[j],x[csmax[j]]);
    }
    printf("\n Tich=%f",max);
}

```

3.5. Duyệt trên cây và bài toán biểu diễn tổng

Bài toán duyệt trên cây có thể giải theo đệ qui như sau:

Xuất phát từ một đỉnh nào đó ta thực hiện hai động tác:

- ☐ Xét đỉnh này
- ☐ Đi đến các đỉnh kề nó bằng phép đệ qui

Bằng thủ tục nói trên có thể đi qua tất cả các đỉnh của một cây. Ta minh họa điều này bằng cách xét bài toán phân tích số n nguyên dương thành tổng các số nguyên dương nhỏ hơn n . Đỉnh xuất phát là n . Từ đỉnh này có thể đi đến các đỉnh gồm một cặp hai số có tổng bằng n , các đỉnh đó là:

$(i, n-i)$, với $i=1, \dots, n/2$.

Tại mỗi đỉnh hai số lại có thể đi đến các đỉnh 3 số. Một cách tổng quát từ đỉnh k số:

$(a_1, \dots, a_{k-1}, a_k)$

có thể đi đến đỉnh $k+1$ số sau:

$(a_1, \dots, a_{k-1}, i, a_{k-i})$, với $i=a_{k-1}, \dots, a_k/2$

Nhận xét: Mỗi dãy số tại đỉnh là đơn điệu tăng.

Có thể chứng minh mọi tổng đều ứng với một đỉnh nào đó thực vậy xét tổng $(k+1)$ số:

$n=t_1+t_2+\dots+t_k+t_{k+1}$

(Dãy t_i không giảm). Rõ ràng tổng này có thể suy ra từ đỉnh gồm k số:

$(t_1, \dots, t_{k-1}, t_k+t_{k+1})$

Như vậy mỗi tổng $(k+1)$ số đều có thể suy ra từ đỉnh k số. Do tập các tổng một số là đầy đủ chỉ gồm n , nên các tập k số cũng đầy đủ với mọi $k > 1$. Đó là điều cần chứng minh.

Ta dùng mảng ngoài $a[100]$ để chứa bộ số của đỉnh. Với đỉnh xuất phát thì:

$k=1$ và $a[1]=n$

Vì cần dùng đến $a[k-1]$, nên đưa thêm $a[0]=1$. Ta xây dựng hàm duyệt đỉnh k với nội dung sau:

1. Xét đỉnh $k > 1$ in các số $a[1], \dots, a[k]$ dưới dạng tổng.

$a[1] + a[2] + \dots + a[k]$

(không xét đỉnh với $k=1$)

2. Đi đến các đỉnh $(k+1)$ số :

$(a_1, \dots, a_{k-1}, i, a_{k-i})$, với $i = a_{k-1}, \dots, a_{k/2}$

Hàm

void duyet_dinh(int k);

chương trình dưới đây sẽ lập theo thuật toán này:

/ Phân tích n và tổng các số nhỏ hơn n
duyet các đỉnh của cây bằng thủ tục đệ qui*/*

#include <stdio.h>

#include <conio.h>

int a[100];

*/*Đã biết a[0],..., a[k]*/*

void duyet_dinh(int k)

{

int x,i;

*/*xet dinh k>1*/*

if(k>1)

for(i=1;i<=k;++i)

printf("%c%d",i>1?'+':'\n',a[i]);

*/*đi đến các đỉnh (k+1) số*/*

x=a[k];

for (i=a[k-1];i<=x/2;++i)

{

a[k]=i; a[k+1]=x-i;

duyet_dinh(k+1);

}

}

void main()

{

int n;

clrscr();

```
printf("\nn="); scanf("%d",&n);
a[0]=1; a[1]=n;
duyet_dinh(1);
getch();
}
```

4. BÀI TẬP MINH HOẠ

Bài 1

Viết hàm đệ qui để tính tổng sau:

$$S=1+2+3+\dots+n$$

```
/*Chương trình tính tổng S bằng hàm đệ qui*/
#include<stdio.h>
int tongs(int n);
main()
{
    int n;
    printf("Nhap vao so n de tinh tong: ");
    scanf("%d",&n);
    /*In ra ket qua */
    printf("\ntong S:=%d",tongs(n));

}
int tongs(int n)
{
    if (n==1)
        return 1;
    else
        return n+tong(n-1);
}
```

Bài 2

/*Chương trình minh họa tìm ước số chung lớn nhất bằng phương pháp đệ qui*/

```
#inclde<stdio.h>
int uscln(int n,int m)
void main()
{
    int so1,so2;
    printf("Nhap so thu nhat:\n"); scanf("%d",&so1);
    printf("Nhap so thu hai :\n"); scanf("%d",&so2);
    printf("USCLN la %d",uscln(so1,so2));
    return;
}
int uscln(int n,int m)
{
    int r;
```


$$F(n) = \sqrt[3]{3 \cdot n + \dots \sqrt[3]{9 + \sqrt[2]{6 + \sqrt[1]{3}}}}$$

Viết chương trình áp dụng

Bài 3

Cho trước hai số nguyên dương x, n . Viết các hàm đệ qui để tính giá trị biểu thức sau:

$$F(x, n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$$

Viết chương trình áp dụng (yêu cầu nhập vào x, n và hiển thị giá trị biểu thức)

Bài 4

Tính các phần tử của tam giác Pascal.

Gợi ý:

```

...
int nhithuc(int n, int p)
{
    if (p == 1 || p == n) return 1;
    else return (nhithuc(n-1, p-1) + nhithuc(n-1, p));
}
...

```

Bài 5

Viết chương trình tính tháp Hà Nội

Sử dụng hàm sau:

```

...
void chuyen(int n, int s, int e, int m)
{
    if (n > 0)
    {
        chuyen(1, s, m, e);
        printf("Chuyen dich tu coc %d toi coc %d\n", s, e);
        chuyen(n-1, m, e, s);
    }
    return;
}
...

```

BÀI 14

KIỀU TẬP TIN (FILE)

1. 1. KHÁI NIỆM VỀ TẬP TIN TRONG C

Là một loại dữ liệu có thể ghi lên đĩa để dùng nhiều lần. Trong C chỉ có một loại File, nhưng cấu trúc của mỗi File có thể khác nhau. Cấu trúc này được hình thành khi ta ghi dữ liệu lên File, nó phụ thuộc vào hàm mà ta dùng để ghi dữ liệu lên đĩa.

2. KHAI BÁO DỮ LIỆU

2.1. Phân loại hàm thao tác trên tập tin

Dữ liệu có tập tin có tên là FILE.

Trong C có hai loại hàm thao tác trên tập tin:

□ Hàm cấp 1

Là những hàm cấp thấp làm việc với tập tin thông qua một số hiệu tập tin (file handle).

□ Hàm cấp 2

Là những hàm được xây dựng từ những hàm cấp 1, dễ sử dụng hơn. Có các hàm phục vụ cho việc đọc ghi trên từng loại dữ liệu (số, chuỗi, ký tự, cấu trúc...)

Các **hàm cấp 2** làm việc với tập tin thông qua một con trỏ tập tin. Con trỏ này được xác định khi ta mở tập tin. Do đó để dùng các hàm cấp hai ta phải khai báo biến con trỏ tập tin.

2.2. Cách khai báo tập tin

Ví dụ

```
FILE *f;  
.....  
f=fopen("data.dat", "wt");
```

3. CÁC KIỂU NHẬP XUẤT

3.1. Nhập xuất kiểu nhị phân:

Dữ liệu ghi lên tập tin không bị thay đổi và khi ta đóng tập tin thì mã kết thúc tập tin sẽ được ghi lên đĩa là -1.

3.2. Nhập xuất kiểu văn bản:

- □ Chỉ khác kiểu nhập xuất nhị phân khi xử lý ký tự xuống dòng và khi ta đóng tập tin thì mã kết thúc tập tin sẽ được ghi lên đĩa là 26.
- □ Khi ghi một ký tự chuyển dòng lên đĩa (mã 10) sẽ ghi thành 2 ký tự mã 13 và mã 10.

- □ Khi đọc nếu gặp hai ký tự liên tiếp là mã 10 và mã 13 sẽ gom lại thành một ký tự là mã 10.

3.3. Các điểm lưu ý

Từ những khác biệt trên nên ta lưu ý:

- Tập tin khi ghi lên đĩa dưới dạng nào thì phải đọc dưới dạng đó. Nếu không việc xử lý sẽ không chính xác.
- Trong C có hàm dùng để nhập xuất cho cả hai kiểu, có hàm chỉ dùng để nhập xuất cho một kiểu nào đó.

3.4. Ví dụ so sánh hai kiểu nhập xuất

Ví dụ

Ta có chương trình sau sử dụng phép đọc ghi trên một tập tin khác nhau

/ Chương trình ví dụ về hai hình thức đọc/ghi tập tin theo kiểu văn bản và nhị phân */*

```
#include <stdio.h>
#include <conio.h>
main()
{
    FILE *f;
    char c,c1;
    clrscr();
    f=fopen("sl","wt");
    fprintf(f,"%c%c%c%c%c%c%c",65,66,67,68,10,26);
    /* ghi lên tập tin 6 byte theo kiểu text
       do vậy khi gặp byte 10 nó sẽ ghi lên 13,10*/
    fclose(f);
    getch();
    f=fopen("sl","rt");
    while (!feof(f))
    { c=getc(f);
      printf("%d ",c);
    }
    /* đọc kết quả của tập tin ghi trên theo kiểu văn bản và in lên màn hình
       ta thấy trên màn hình sẽ có kết quả là 65,66,67,68,10,-1. Trong đó -1 là mã kết thúc tập
       tin.==> thiếu mã 26 do 26 được hiểu là mã kết thúc tập tin trong kiểu nhập xuất văn bản
       nếu thay dòng f=fopen("sl","rt"); bằng f=fopen("sl","rb") thì kết quả in
       lên màn hình là 65 66 67 68 13 10 26 -1*/
    fclose(f);
    getch();
}
```

Một ví dụ khác là: tập tin `command.com` được ghi dưới dạng nhị phân, nhưng lệnh `type` của `dos` lại mở file dưới dạng văn bản do vậy ta thấy nó chỉ đọc được một phần của tập tin (do gặp mã 26).

4. CÁC THAO TÁC TRÊN TẬP TIN

Các hàm sau đây dùng chung cho cả hai kiểu nhị phân và văn bản.

1. *FILE *fopen(const char *tên_tập_tin, const char *kiểu);*

Mở một tập tin. Nếu thành công hàm trả về kết quả là con trỏ `FILE` tương ứng với file vừa mở, ngược lại trả về giá trị `NULL`. Vì vậy sau khi mở file ta phải kiểm tra xem thao tác mở tập tin có thành công hay không.

**tên_tập_tin:* Là một hằng chuỗi, hoặc một con trỏ chỉ đến vùng nhớ chứa tên tập tin.

**kiểu:* là hằng chuỗi cho biết kiểu truy nhập:

Kiểu	Ý nghĩa
"r" "rt"	Mở tập tin để đọc theo kiểu văn bản. Tập tin phải có trên đĩa nếu không sẽ có lỗi.
"w" "wt"	Mở tập tin để ghi theo kiểu văn bản. Nếu tập tin đã có trên đĩa sẽ bị xóa.
"a" "at"	Mở tập tin để ghi bổ sung theo kiểu văn bản. Nếu tập tin chưa có thì tạo mới.
"r+" "rt+"	Mở tập tin để đọc/ghi theo kiểu văn bản. Tập tin phải có trên đĩa nếu không sẽ có lỗi.
"w+" "wt+"	Mở tập tin để đọc/ghi theo kiểu văn bản. Nếu tập tin đã có trên đĩa sẽ bị xóa.
"a+" "at+"	Mở tập tin để đọc/ghi bổ sung theo kiểu văn bản. Nếu tập tin chưa có thì tạo mới.
"rb"	Mở tập tin để đọc theo kiểu nhị phân. Tập tin phải có trên đĩa nếu không sẽ có lỗi.
"wb"	Mở tập tin để ghi theo kiểu nhị phân. Nếu tập tin đã có trên đĩa sẽ bị xóa.
"ab"	Mở tập tin để ghi bổ sung theo kiểu nhị phân. Nếu tập tin chưa có thì tạo mới.
"r+b"	Mở tập tin để đọc/ghi theo kiểu nhị phân. Tập tin phải có trên đĩa nếu không sẽ có lỗi.
"w+b"	Mở tập tin để đọc/ghi theo kiểu nhị phân. Nếu tập tin đã có trên đĩa sẽ bị xóa.
"a+b"	Mở tập tin để đọc/ghi bổ sung theo kiểu nhị phân. Nếu tập tin chưa có thì tạo mới.

2. *int fclose(FILE *f)*

Đóng tập tin được chỉ đến bởi con trỏ f. Nếu thành công thì giá trị của hàm = 0 ngược lại có giá trị EOF. Sau khi đóng con trỏ f sẽ không còn trỏ đến file trước đó nữa (nó có thể dùng vào việc khác).

3. *int fcloseall(void)*

Đóng tất cả các tập tin đang mở. Nếu thành công giá trị của hàm bằng số tập tin đã đóng, ngược lại cho giá trị EOF

4. *int fflush(FILE *f)*

Làm sạch vùng đệm của tập tin được chỉ đến bởi con trỏ f. Nếu thành công cho giá trị 0, ngược lại cho giá trị EOF.

5. *int flushall(void)*

Làm sạch vùng đệm của tất cả các tập tin đang mở. Nếu thành công giá trị của hàm bằng số tập tin đang mở, ngược lại cho giá trị EOF

6. *int unlink(const char *tên_tập_tin)*

Xóa một tập tin trên đĩa. Nếu thành công giá trị của hàm bằng 0 , ngược lại cho giá trị EOF

7. *int rename(const char *tên_cũ, const char *tên_mới)*

Đổi một tập tin trên đĩa. Nếu thành công giá trị của hàm bằng 0 , ngược lại cho giá trị EOF

8. *int feof(FILE *f)*

Cho giá trị khác không nếu ở cuối tập tin, ngược lại =0

5. CÁC HÀM NHẬP XUẤT

5.1. Nhập xuất ký tự : (Dùng cho kiểu nhị phân và văn bản):

- Ghi ký tự lên tập tin:
*int putc(int ch, FILE *f)*
*int fputc(int ch, FILE *f)*
Ghi lên file f ký tự có mã = ch % 256 Nếu thành công kết quả = mã của ký tự đã ghi, ngược lại =EOF (-1)
Trong trường hợp ghi theo văn bản thì khi gặp mã 10 sẽ ghi thành 13 và 10
- Đọc ký tự từ tập tin:
*int getc(FILE *f)*
*int fgetc(FILE *f)*
Đọc một ký tự từ file f . Nếu thành công kết quả = mã của ký tự đọc được, ngược lại = -1

5.2. Nhập xuất chuỗi: (Dùng cho kiểu văn bản)

- Ghi một chuỗi:
`int fputs(const char *s, FILE *f)`

Ghi một chuỗi được chỉ tới bởi con trỏ s vào file f.
Kết quả = ký tự cuối được ghi nếu thành công, ngược lại =EOF

- Đọc một chuỗi:
`char *fgets(const char *s, int n, FILE *f)`

Đọc một chuỗi từ File f và đưa vào vùng nhớ do s trỏ đến.
Việc đọc kết thúc khi đã đọc được n-1 ký tự , hoặc gặp ký tự xuống dòng , hoặc gặp ký tự kết thúc File.
Nếu việc đọc có lỗi kết quả của hàm =NULL.

5.3. Đọc ghi dữ liệu theo khuôn dạng: (Dùng cho kiểu văn bản)

- Ghi dữ liệu theo khuôn dạng:
`int fprintf(FILE *f , const char *đặc tả,...)`

... là danh sách các đối số tương ứng với các đặc tả.
Sử dụng giống như hàm printf, dữ liệu sẽ được ghi lên file.

- Đọc dữ liệu theo khuôn dạng:
`fscanf(FILE *f , const char *đặc tả,...)`

... là danh sách các đối số tương ứng với các đặc tả.
Sử dụng giống như hàm scanf, dữ liệu sẽ được đọc từ File f rồi đưa vào các đối số tương ứng.

Ví dụ: Đọc nội dung của một tập tin theo một cấu trúc như sau:

Tập tin ghi thông tin của n đỉnh của một đa giác. Mỗi đỉnh có 2 tọa độ là x,y kiểu số nguyên.

Tập tin có n+1 dòng.

Dòng đầu chứa số đỉnh (n)

n dòng còn lại , mỗi dòng chứa tọa độ của một đỉnh.

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

```
#define MAX 50
```

```
main()
```

```
{FILE *f;
```

```
int i,n,x[MAX],y[MAX];
```

```
f=fopen("dagiac.sl", "rt");
```

```
fscanf(f, "%d", &n);
```

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

```
fscanf(f, "%d%d", &x[i], &y[i]);
```

```
fclose(f);
```

```
}
```

5.4. Đọc ghi số nguyên : (Dùng cho kiểu nhị phân)

- Ghi một số nguyên:
`int putw(int n, FILE *f)`

- Ghi số nguyên n (2 byte) lên File f. Nếu không thành công hàm có kết quả EOF
Đọc một số nguyên:
`int getw(FILE *f)`

Đọc một số nguyên n (2 byte) từ file f. Nếu thành công kết quả của hàm bằng giá trị đọc được, ngược lại = EOF.

5.5. Đọc ghi cấu trúc : (Dùng cho kiểu nhị phân)

- Ghi mẫu tin lên tập tin:
`int fwrite(void *ptr, int size, int n, FILE *f)`

ptr: con trỏ chỉ tới vùng nhớ chứa dữ liệu cần ghi;

size: Kích thước của một mẫu tin (tính theo byte);

n: Số mẫu tin cần ghi.

Ghi n mẫu tin có kích thước của mỗi mẫu tin là size lên File f. Kết quả của hàm bằng số mẫu tin được ghi thật sự.

- Đọc mẫu tin lên tập tin:
`int fread(void *ptr, int size, int n, FILE *f)`

ptr: con trỏ chỉ tới vùng nhớ chứa dữ liệu đọc được;

size: Kích thước của một mẫu tin (tính theo byte);

n: Số mẫu tin cần đọc.

Đọc n mẫu tin có kích thước của mỗi mẫu tin là size từ File f rồi lưu vào vùng nhớ do ptr chỉ đến. Kết quả của hàm bằng số mẫu tin được ghi thật sự.

6. CÁC HÀM DI CHUYỂN CON TRỎ TẬP TIN

6.1. Đưa con trỏ về đầu tập tin

Cú pháp: `void rewind(FILE *f)`

Chức năng: Đưa con trỏ về đầu tập tin.

6.2. Dời con trỏ tập tin

Cú pháp: `int fseek(FILE *f, long số_byte, int vt_bắt_đầu)`

Chức năng:

dời con trỏ tập tin đi số_byte tính từ vt_bắt_đầu.

Nếu :

số_byte >0 chuyển xuống dưới,

số_byte <0 chuyển ngược lên

số_byte=0 đứng tại vt_bắt_đầu,

vt_bắt_đầu có thể mang một trong các giá trị sau:

SEEK_SET (0) : Vị trí đầu tập tin

SEEK_CUR (1): Vị trí hiện hành

SEEK_END (2): Vị trí cuối tập tin.

6.3. Cho biết vị trí hiện tại của con trỏ

Cú pháp: **long ftell(FILE *f)**

Chức năng:

cho biết vị trí hiện tại của con trỏ, tính từ đầu tập tin.

7. CÁC HÀM QUẢN LÝ THƯ MỤC

Các hàm này định nghĩa trong dir.h

1. **int chdir(char *s)**

Đổi thư mục hiện hành, s là con trỏ chỉ tới vùng nhớ chứa tên thư mục cần đổi (có thể chứa cả ổ đĩa và đường dẫn).

Kết quả =0 nếu thành công, ngược lại =-1

2. **char *getcwd(char *s, int n)**

Hàm này lấy tên thư mục hiện hành và gán vào vùng nhớ do s trỏ tới. n là độ dài tối đa của đường dẫn và thư mục.

3. **int mkdir(char *s)**

Tạo thư mục có tên được lưu tại vùng nhớ do s chỉ tới.

Kết quả =0 nếu thành công, ngược lại =-1

4. **int rmdir(char *s)**

Xóa thư mục có tên được lưu tại vùng nhớ do s chỉ tới, thư mục cần xóa phải rỗng.

Kết quả =0 nếu thành công , ngược lại =-1

5. **int findfirst(char *path, struct ffblk *fd, int attrib)**

Tìm tập tin đầu tiên thỏa điều kiện được chỉ ra trong đó:

path: là đường dẫn chứa tên thư mục hoặc tên tập tin cần tìm.

fd : là con trỏ chỉ đến vùng nhớ để lưu các thông tin của tập tin trong trường hợp tìm được. Các thông tin này lưu dưới dạng cấu trúc ffblk được định nghĩa như sau:

struct ffblk

```
{    char ff_attrib;  
    unsigned ff_fsize;  
    unsigned ff_fdate;  
    long ff_fsize;  
    char ff_fname[13];
```

```

        char ff_freserved[21];
    }
    attrib : là thuộc tính của tập tin cần tìm, có thể là một hoặc là hợp của các giá trị sau:
    FA_RDONLY -
    FA_HIDDEN
    FA_SYSTEM
    FA_LABEL
    FA_DIREC
    FA_ARCH

```

Công dụng của hàm:

Tìm trong phạm vi được chỉ ra bởi path tập tin đầu tiên có thuộc tính attrib. Ta nên hiểu ở đây thư mục cũng là một tập tin nhưng có thuộc tính FA_DIREC

Nếu tìm thấy: thì thông tin của File này được đặt vào vùng nhớ do fd trỏ đến. Thông tin này sẽ dùng cho hàm findnext để tiếp tục tìm các tập tin có thuộc tính như vậy. Kết quả =0 nếu thành công, ngược lại =-1.

6. *int findnext(struct fblk *fd)*

Tiếp tục tìm tập tin có yêu cầu như trong hàm findfirst. Nếu tìm thấy thông tin này lại được lưu vào vùng nhớ do fd chỉ đến.

8. CÁC BÀI TẬP MINH HỌA

Bài 1

*/*Chương trình minh họa cách tạo, mở một file và nhập một kí tự vào file */*

```

#include <stdio.h>
main()
{
    FILE *file_pointer;    /* Đây là con trỏ file. */

    /* Khởi tạo tập tin myfile.dta và gán địa chỉ của nó
       đến con trỏ File file_pointer: */
    file_pointer = fopen("MYFILE.DTA", "w");

    /* Nhập một kí tự vào file đã mở: */
    putc('C', file_pointer);

    /* Đóng File vừa tạo. */
    fclose(file_pointer);
}

```

-

Bài 2

*/*Chương trình minh họa cách mở một File đã tồn tại trên đĩa để đọc*/*

```

#include <stdio.h>
main()
{

```

```

FILE *file_pointer; /* Đây là con trỏ File. */
Char file_character; /* Kí tự được đọc từ file. */

/* Mở file đang tồn tại (myfile.dta) và gán địa chỉ của nó đến con trỏ file
file_pointer: */
file_pointer = fopen("MYFILE.DTA", "r");

/* Nhập kí tự đầu tiên từ file đã mở: */
file_character = getc(file_pointer);

/* Xuất kí tự lên màn hình. */
printf("Kí tự là %c\n", file_character);

/* Đóng file đã tạo. */
fclose(file_pointer);
}
-

```

Bài 3

/*Chương trình minh họa cách nhập một chuỗi kí tự vào file*/

```

#include <stdio.h>
main()
{
    FILE *file_pointer; /*Đây là con trỏ File. */
    char file_character; /* Kí tự được đọc từ file. */

    /* Tạo file myfile.dta và gán địa chỉ của nó đến con trỏ file_pointer: */
    file_pointer = fopen("MYFILE.DTA", "w");
    /* Nhập một chuỗi dữ liệu vào file đã mở : */
    while((file_character = getche()) != '\r')
        file_character = putc(file_character, file_pointer);

    /* Đóng file vừa tạo. */
    fclose(file_pointer);
}

```

Bài 4

/* Chương trình minh họa cách đọc một chuỗi kí tự từ file đã tồn tại trên đĩa*/

```

#include <stdio.h>
main()
{
    FILE *file_pointer; /* Đây là con trỏ File. */

```

```

char file_character; /* kí tự được đọc từ file. */

/* Mở file đang tồn tại (myfile.dta) và gán địa chỉ của nó đến con trỏ file
file_pointer: */
file_pointer = fopen("MYFILE.DTA","r");

/* Đọc chuỗi kí tự từ file đã mở và xuất nó ra màn hình. */
while((file_character = getc(file_pointer)) != EOF)
    printf("%c", file_character);

/* Đóng file đã tạo. */
fclose(file_pointer);
}

```

Bài 5

/*Chương trình Sao chép hai tập tin:*/

```

#include <stdio.h>
#include <conio.h>
main(int n,char *parm[])
{
    FILE *f1,*f2;
    char c;
    clrscr();
    if (n!=3) exit();
    f1=fopen(parm[1],"rt");
    if (f1==NULL) { printf("Khong co file %s tren dia ",parm[1]); exit();}
    f2=fopen(parm[2],"wt");
    if (f2==NULL)
    { printf("Duong dan khong hop le %s tren dia ",parm[2]); exit();}
    while (!feof(f1))
    { c=fgetc(f1);
      fputc(c,f2);
    }
    fclose(f1);
    fclose(f2);
    getch();
}

```

Bài 6

/*Chương trình tổng quát minh họa cách khởi tạo, nhập, xuất dữ liệu trên file */

```

#include <stdio.h>
main()
{
    char selection[2]; /* Chọn lựa. */
    char file_name[13]; /* Tên file. */

```

```

char user_choice[2]; /* Chọn chế độ kích hoạt. */
int selection_value; /* Chọn giá trị. */
int file_character; /* Kí tự File được lưu. */
FILE *file_pointer; /* Con trỏ File. */

/* Hiện thị các mục chọn. */

printf("Chọn một trong các mục sau:\n");
printf("1] Tạo một File mới. 2] Viết đè lên một file đã tồn tại.\n");
printf("3] Cộng thêm dữ liệu mới đến file đã tồn tại.\n");
printf("4] Nhập dữ liệu từ file đang tồn tại.\n");

/* Nhập các giá trị. */

do
{
    printf("Lựa chọn của bạn => ");
    gets(user_choice);
    selection_value = atoi(user_choice);

    switch(selection_value)
    {
        case 1 : /* Tạo một file mới. */
        case 2 : strcpy(selection, "w"); /* Viết chồng lên dữ liệu đã có. */
                break;
        case 3 : strcpy(selection, "a"); /* Cập nhập dữ liệu đến file đang
                                           tồn tại. */
                break;
        case 4 : strcpy(selection, "r"); /* Nhập dữ liệu từ file đang tồn tại.*/
                break;
        default : {
                    printf("Điều này không được chọn.\n");
                    selection_value = 0;
                }
    }
} while(selection_value == 0);

/* Nhập file từ người sử dụng. */
printf("Nhập tên của file => ");
gets(file_name);

/* Mở file để vận hành. */
if((file_pointer = fopen(file_name, selection)) == NULL)
{
    printf("Không thể mở được file %s!", file_name);
}

```



```

        exit(-1);
    }

    /* Viết hoặc đọc từ file. */
    switch(selection_value)
    {
        case 1 : break;
        case 2 :
        case 3 : {
            printf("Vào chuỗi để lưu: \n");
            while((file_character = getche()) != '\r')
                file_character = putc(file_character, file_pointer);
            }
            break;
        case 4 : {
            while((file_character = getc(file_pointer)) != EOF)
                printf("%c", file_character);
            }
            break;
    }
    /* Đóng file. */
    fclose(file_pointer);
}

```

Bài 7

/*Chương trình minh họa con trỏ file và kiểu cấu trúc*/

```

#include <stdio.h>
typedef struct
{
    char part_name[15];    /* Tên. */
    int quantity;          /* Số lượng. */
    float cost_each;       /* Đơn giá. */
} parts_structure;

main()
{
    parts_structure parts_data;    /* biến cấu trúc. */
    FILE *file_pointer;            /* Con trỏ File. */

    /* Mở một file để viết. */

    file_pointer = fopen("B:PARTS.DTA","wb");

    /*Nhập dữ liệu bởi người sử dụng chương trình. */

```

```

do
{
    printf("\nName of part => ");
    gets(parts_data.part_name);
    printf("Number of parts => ");
    scanf("%d", &parts_data.quantity);
    printf("Cost per part => ");
    scanf("%f", &parts_data.cost_each);

    /* Viết cấu trúc đến file đã mở. */

    fwrite(&parts_data, sizeof(parts_data), 1, file_pointer);

    /* Thực hiện lặp cho nhiều lần nhập. */

    printf("Add more parts (y/n)? => ");
} while (getche() == 'y');

/* Đóng file. */
fclose(file_pointer);
}

```

Bài 8

/*Chương trình sau đây liệt kê tên các tập tin có trong một thư mục và trong tất cả các thư mục con của nó.*/

```

#include <stdio.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>
char bs[] = "\\*.\"", name[50], ext[3];
# define attr\
(FA_RDONLY|FA_HIDDEN|FA_SYSTEM|FA_LABEL|FA_DIREC|FA_ARCH)
int sf, nn;
void scandir(char *dir);
main()
{
    struct ffblk *ff;
    char dirname[50];
    clrscr();
    printf("Ten thu muc muon xem : ");
    gets(dirname);
    scandir(dirname);
    getch();
    clrscr();
}

```

```

void scandir(char *dir)
{
    char d1[80],d2[80];
    int first=1,s;
    struct ffblk f;
    sprintf(d1,"%s%s",dir,bs);
    printf("\n\n%s",d1);
    getch();
    while (1)
    {
        if (first)
        { s=findfirst(d1,&f,attr);
          first=0;
        }
        else s=findnext(&f);
        if (s!=0) return;
        if(f.ff_name[0]=='.') continue;
        if (f.ff_attrib==FA_DIREC)
        {
            sprintf(d2,"%s\\%s",dir,f.ff_name);
            scandir(d2);
        }
        else
        {
            ++sf;
            printf("\n%15s %15d %10d ",f.ff_name,f.ff_fdate,f.ffftime);
            if (sf%20==0)
            { printf("\nBam enter xem tiep ");
              getch();
            }
        }
    }
}

```

9. CÁC BÀI TẬP TỰ LÀM

Bài 1

Viết chương trình đọc vào một File, biến các chữ hoa thành chữ thường và lưu trên đĩa.

Bài 2

Viết đoạn chương trình để nhập dữ liệu cho file dagiac.sl trên.

Bài 3

Viết chương trình đếm các ký tự chữ cái có trong một tập tin.

-

Bài 4

Đếm số từ của một tập tin.

Bài 5

Đếm số lần lặp lại của một từ trong một tập tin.

Bài 6

Đổi toàn bộ các ký tự trong một tập tin sang chữ hoa.

Bài 7

Đổi ngược nội dung các dòng trong một tập tin

Bài 8

Viết chương trình gồm 2 chức năng:

Nhập và lưu các hệ số a,b,c của các phương trình bậc hai vào một tập tin .
Tìm nghiệm của các pth 2 có hệ số a,b,c được lưu trong tập tin trên và lưu kết quả vào một tập tin khác.

Bài 9

Viết chương trình ghi một danh sách cấu trúc xuống tập tin sau đó đọc lên kiểm tra lại.

Bài 10

Viết chương trình ghi các dòng văn bản đánh từ bàn phím lên tập tin.

Bài 11

Viết chương trình nhập dữ liệu và ghi vào đĩa một dãy số nguyên bất kỳ. Sắp xếp dãy theo thứ tự tăng dần, rồi lại ghi vào đĩa.

Bài 12

Viết chương trình nhập dữ liệu ghi vào đĩa các thành phần HO,TEN, TUOI, CHUCVU, BACLUONG thành tập tin LUONG.DTA. Khi nào không muốn nhập nữa thì nhấn phím ESC.

Bài 13

Chương trình trên liệt kê tất cả các tập tin có trong một thư mục. Hãy viết lại để chương trình chỉ liệt kê các tập tin theo ý muốn.

Ví dụ như *.txt, *.bak

Bài 14

Viết chương trình có công dụng như lệnh CD của dos.

Bài 15

Viết chương trình có công dụng như lệnh RD của dos.

-

Bài 16

Viết chương trình xóa các tập tin có trong một thư mục và trong các thư mục con của nó. Với đường dẫn và loại tập tin được nhập từ bàn phím.

BÀI 15

DỰ ÁN (PROJECT)

1. PHÂN CHIA CHƯƠNG TRÌNH THÀNH NHIỀU TẬP TIN

1.1. Phân tích nguyên nhân

Với các bài toán lớn, nếu viết chương trình giải quyết bài toán trên một tập tin thì việc xử lý các dòng lệnh sẽ gặp nhiều khó khăn: Do có quá nhiều dòng lệnh nên dễ dẫn tới sự nhầm lẫn giữa các dòng lệnh với nhau. Hơn nữa, Turbo C không cho phép dịch tệp chương trình có kích thước lớn hơn 64 KB.

1.2. Đánh giá, kết luận việc phân chia chương trình thành nhiều tập tin

Như vậy, việc phân chia một chương trình lớn thành các tập tin nhỏ hơn, trong đó mỗi tập tin chứa các hàm và dữ liệu thực hiện một số chức năng nhất định là cần thiết.

Ví dụ: Một tập tin chứa toàn các hàm tính toán và một tập tin chứa toàn các hàm xử lý ký tự vv....

2. DỊCH MỘT CHƯƠNG TRÌNH CHỨA NHIỀU TẬP TIN

2.1. Cách dịch

Muốn dịch một chương trình chứa nhiều tập tin, đầu tiên phải dịch từng tập tin cá thể, sau đó thực hiện liên kết lại thành một tập tin có đuôi exe để chạy.

Quá trình dịch được thực hiện trong C, nhờ cấu trúc tập tin project (dự án).

Việc dịch được thực hiện như sau:

Nhìn bảng chọn lệnh trên màn hình C, sẽ thấy mục project. Chọn mục này và chọn tiếp mục nhỏ Open Project. Nếu đã có tập tin Project sẵn, chỉ cần gõ Ctrl+F9 để dịch và chạy.

Trường hợp chưa có, phải tạo mới Project. Việc tạo mới này, được thực hiện bằng cách tạo tên Project. Sau đó, chọn mục Add Item để đưa các tập tin trong cùng một dự án vào tệp Project chung.

2.2. Ví dụ minh họa

Nhập vào ma trận các số nguyên dương. Sắp xếp và in ra ma trận tăng dần theo hình xoắn ốc.

Dùng ba tập tin riêng rẽ:

Tập tin 1: *TVXOANOC.CPP*

```
#include<c:\borlandc\bin\cxoan.h>
void SortHCN(int D[][N])
{
    int L=0, R=N-1, T=0, B=M-1, H=M*N;
    int k=0, i, j, C[M*N];
    for(i=0, k=0 ; i<M ; i++)
        for(j=0 ; j<N ; j++)
            C[k++]=D[i][j];

    k=0;
    while(k<H)
    {
        for(j=L ; j<R ; j++) D[T][j]=C[k++];
        if(k<H)
            for(i=T ; i<B ; i++) D[i][R]=C[k++];
        if(k<H)
            for(j=R ; j>L ; j--) D[B][j]=C[k++];
        if(k<H)
            for(i=B ; i>T ; i--) D[i][L]=C[k++];
        L++; T++; R--; B--;
    }
}
```

Tập tin 2: *XOANOC.CPP*

//Chương trình sắp xếp ma trận theo hình xoắn ốc

```
#include<c:\borlandc\bin\cxoan.h>
void main()
{
    int i, j, *p, tmp, H, A[M][N];
    clrscr(); randomize();
    printf("Khoi tao ma tran : \n");
    for(i=0 ; i<M ; i++)
    {
        for(j=0 ; j<N ; j++)
        {
            A[i][j]=random(50);
            printf("%4d", A[i][j]);
        }
        printf("\n");
    }

    p=&A[0][0];
    H=M*N;
```

```

    for(i=0 ; i<H-1 ; i++)
        for(j=i+1 ; j<H ; j++)
            if(*(p+i)>*(p+j))
            {
                tmp=*(p+i);
                *(p+i)=*(p+j);
                *(p+j)=tmp;
            }
    SortHCN(A);
    printf("\nSau khi sap theo hình tron oc :\n\n");
    for(i=0 ; i<M ; i++)
    {
        for(j=0 ; j<N ; j++)
            printf("%4d",A[i][j]);
        printf("\n");
    }
    getch();
}

```

Tập tin 3: *CXOAN.H* là tập tin lưu các định nghĩa hằng và hàm được đặt trong thư mục:
C:\BORLANDC\BIN

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define M 7
#define N 6
void SortHCN(int C[][N]);

```

Và cuối cùng thì tạo ra tập tin dự án có tên *VDU.PRJ* để dịch ra tập tin chương trình có tên là *VDU.EXE*, tập tin dự án có hai dòng:

```

TVXOANOC.CPP
XOANOC.CPP

```

Từ đây, mỗi khi gọi chương trình dịch của *C* bằng cách ấn *Ctrl+F9*, máy sẽ dịch các tập tin đang soạn thảo và các tập tin khác, nếu các tập tin này có sự thay đổi.

3. BIẾN TOÀN CỤC

3.1. Cách khai báo- phân tích ảnh hưởng

Để có thể truy nhập vào các biến toàn cục ở mọi nơi, mọi lúc khi viết chương trình gồm các hàm chứa trong nhiều tập tin riêng lẻ. *C* cho phép khai báo lại các biến toàn cục một lần nữa trong tất cả các tập tin. Tuy nhiên chỉ một tập tin trong số các tập tin này có các khai báo bình thường. Trong các tập tin khác chúng ta sử dụng biến này thì khai báo có thêm từ khóa *extern*.

```
extern int so_nguyen;
```

Từ khóa *extern* thông báo biến *so_nguyen* không phải là biến mới, nó đã được định nghĩa và khai báo ở những chỗ khác và đừng có cấp phát bộ nhớ cho biến số nguyên trong tập tin này. Chương trình kết nối sẽ tìm ra khai báo biến này với điều kiện nó được khai báo trong một, và chỉ một mà thôi trong các tập tin được kết nối.

3.2. Ví dụ minh họa

Tập tin 1:TVZZD.CPP

```
#include<c:\borlandc\bin\czc.h>
```

```
extern int k_h,k_c;
```

```
void sort(int B[],int r, int c)
```

```
{
    int spt,i,j,min,tmp;
    printf("\n Ma tran cap %d*%d sau khi sap xep:\n",k_h,k_c);
    spt=r*c;
    for(i=0 ; i<spt-1 ; i++)
    {
        min=i;
        for(j=i+1 ; j<spt ; j++)
            if(B[j]<B[min]) min=j;
        tmp=B[i];
        B[i]=B[min];
        B[min]=tmp;
    }
}
```

```
void ziczac_v(int B[],int A[][C],int r, int c)
```

```
{
    int spt,i,j,k,t;
    for(i=0,k=0 ; i<c ; i++)
    {
        if(i%2==0) t=0;
        else t=r-1;
        for(j=0 ; j<r ; j++)
            A[abs(j-t)][i]=B[k++];
    }
}
```

Tập tin 2:ZZD.CPP

```
// ZICZAC_V
```

```
#include<c:\borlandc\bin\czc.h>
```

```
int k_h=H,k_c=C;
```

```
void main(void)
```

```
{
    int A[H][C],i,j,B[H*C];
    clrscr();
}
```



```

    randomize();
    printf("\n Ma tran cap %d*%d truoc khi sap xep:\n",k_h,k_c);
    for(i=0 ; i<H ; i++)
    {
        for(j=0 ; j<C ; j++)
        {
            A[i][j]=random(100);
            printf("%4d",A[i][j]);
        }
        printf("\n");
    }
    memcpy(B,A,sizeof(int)*H*C);
    sort(B,H,C);
    printf("\n");
    ziczac_v(B,A,H,C);
    for(i=0 ; i<H ; i++)
    {
        for(j=0 ; j<C ; j++)
        printf("%4d",A[i][j]);
        printf("\n");
    }
    getch();
}

```

Tập tin 3:CZZ.H được đặt trong thư mục:C:\BORLANDC\BIN

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <mem.h>
#include <math.h>
#define H 5
#define C 6
void sort(int B[],int r,int c);
void ziczac_v(int B[],int A[][C],int r,int c);

```

Nếu không có từ khóa **extern** trong tập tin 1: TVZZD.CPP thì máy sẽ báo lỗi.

4. CÁC BÀI TẬP MINH HỌA

Chương trình tìm phần tử yên ngựa trong ma trận (Maxdòng- Mincột hoặc Maxcột- Mindòng)

Tập tin 1: TVYEN.CPP

```

#include<c:\borlandc\bin\cyen.h>
extern int A[DONG][COT];
int CheckYenNgua(int x, int n, int m)

```

```

{
    int i, Min_d, Max_d, Min_c, Max_c;
    Min_d=Max_d=Min_c=Max_c=x;
    for(i=0 ; i<COT ; i++)
    {
        Min_d=(Min_d>A[n][i] ? A[n][i]:Min_d);
        Max_d=(Max_d<A[n][i] ? A[n][i]:Max_d);
    }
    for(i=0 ; i<DONG ; i++)
    {
        Min_c=(Min_c>A[i][m] ? A[i][m]:Min_c);
        Max_c=(Max_c<A[i][m] ? A[i][m]:Max_c);
    }
    if((Min_d==x && Max_c==x) || (Min_c==x && Max_d==x)) return 1;
    return 0;
}

```

Tập tin 2: *YENNG.CPP*

//Tim cac phan tu Yen Ngua trong mang 2 chieu

#include<c:\borlandc\bin\cyen.h>

int A[DONG][COT];

int CheckYenNgua(int, int, int);

void main()

```

{
    int i, j, k;
    do
    {
        clrscr();
        randomize();
        for(i=0 ; i<DONG ; i++)
            for(j=0 ; j<COT ; j++)
                A[i][j] = random(10);

//In ma tran
        textcolor(BLUE);
        for(i=0 ; i<DONG ; i++)
        {
            for(j=0 ; j<COT ; j++) cprintf("%4d",A[i][j]);
            printf("\n");
        }
        printf("\nKet qua sau khi tim phan tu yen ngua\n\n");
        textcolor(WHITE);
//Hien thi ma tran ket qua
        for(i=0 ; i<DONG ; i++)
        {

```

```

        for(j=0 ; j<COT ; j++)
        {
            if(CheckYenNguoa(A[i][j],i,j)==1)
            {
                textcolor(RED);
                cprintf("%4d",A[i][j]);
                textcolor(WHITE);
            }
            else cprintf("%4d",A[i][j]);
        }
        printf("\n");
    }
}while(getch()!=27);
}

```

Tập tin 3: *CYEN.H* được đặt trong thư mục: *C:\BORLANDC\BIN*

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define DONG 7
#define COT 4

```

Liên kết ba tập tin tạo thành tập tin *BT_VD.PRJ* và dịch thành tập tin *BT_VD.EXE*.
 Các bạn có thể thử nghiệm và kiểm tra kết quả chạy chương trình.

5. BÀI TẬP TỰ LÀM

Các bạn hãy tự chọn một số bài tập trong các bài tập từ trước đến nay, và xây dựng tệp dự án để giải quyết những bài toán đó.

PHỤ LỤC

Tài liệu tham khảo



- JAMES L.ANTONAKOS & KENNETH C.MANSFIELD JR.: "*Application Programming in Structured C*". Prentice hall international editions-1996.
- MARK ALLEN WEISS: "*Efficient c programming a practical approach*". Prentice hall international editions-1997.
- BRIAN W.KERNIGHAN & DENNIS M.RITCHI: "The C programming language". Prentice hall international editions-1997.
- PHẠM VĂN ÁT: "*Kỹ thuật lập trình C: cơ sở và nâng cao*". Nhà xuất bản khoa học kỹ thuật-1996 .
- NGUYỄN THANH THỦY, LÊ ĐĂNG HÙNG, TRẦN VIỆT LINH, LÊ ĐỨC TRUNG: "*Ngôn ngữ lập trình C thật là đơn giản*". Trường đại học bách khoa Hà nội-1996
- QUÁCH TUẦN NGỌC: "Ngôn ngữ lập trình C". Nhà xuất bản giáo dục – 1998.



	<u>Bài 3 Nhập xuất dữ liệu trong C</u>
	<u>Giới thiệu tổng quan</u>
	<u>Cc hàm nhập xuất thuộc stdio.h</u>
	<u>Cc hàm nhập xuất trong conio</u>
	<u>Cc bài tập minh họa</u>
	<u>Cc bài tập tự luận</u>
	<u>Bài 4 Các cấu trúc điều khiển trong C</u>
	<u>Giới thiệu tổng quan</u>
	<u>Cu lệnh, khối lệnh</u>
	<u>Tổn tử if</u>
	<u>Tổn tử switch</u>
	<u>Tổn tử for</u>
	<u>Tổn tử while</u>
	<u>Tổn tử do... while</u>
	<u>Ch ý khi gặp break v continue trong while hay do...while</u>
	<u>Tổn tử goto</u>
	<u>Cc bài tập minh họa</u>
	<u>Cc bài tập tự luận</u>
	<u>Bài 5 Con trỏ và địa chỉ</u>
	<u>Giới thiệu tổng quan</u>
	<u>Toán tử địa chỉ</u>
	<u>Con trỏ</u>
	<u>Qui tắc sử dụng con trỏ trong biểu thức</u>
	<u>Qui tắc về kiểu gí trị trong khai báo</u>
	<u>Bổ định tính const với con trỏ</u>
	<u>Cc bài tập minh họa</u>
	<u>Cc bài tập tự luận</u>
	<u>Bài 6 Hàm và cấu trúc chương trình</u>
	<u>Giới thiệu tổng quan</u>
	<u>Vài nét về hàm và chương trình</u>
	<u>Ví dụ về chương trình cĩ hàm</u>
	<u>Cách viết một hàm</u>
	<u>Hàm v con trỏ</u>
	<u>Cc bài tập minh họa</u>
	<u>Cc bài tập tự luận</u>
	<u>Bài 7 Mảng (Array)</u>
	<u>Giới thiệu tổng quan</u>
	<u>Khi niệm về mảng</u>

	Cch khai bo
	Chỉ số của mảng
	Lấy địa chỉ của phần tử mảng
	Nhập, xuất dữ liệu cho cc phần tử mảng
	Một số vấn đề liên quan
	Con trỏ v mảng
	Hm, con trỏ v mảng
	Cc bi tập minh hoa
	Cc bi tập tư lm
	Bài 8 Chuỗi ký tự
	Giới thiệu tổng quan
	Khi niệm
	Cch thao tc trn chuỗi ký tự
	Con trỏ v chuỗi k tự
	Mảng chuỗi ký tự
	Cc bi tập minh hoa
	Cc bi tập tư lm
	Bài 9 Cấp phát và giải phóng bộ nhớ động
	Giới thiệu tổng quan
	Nhắc lại về con trỏ
	Biến động
	Bộ nhớ heap và cơ chế tạo biến động
	Cc bi tập minh hoa
	Cc bi tập tư lm
	Bài 10 Hm main cũ tham số - Con trỏ hm
	Giới thiệu tổng quan
	Hm main cũ tham số
	Con trỏ hm
	Cc bi tập minh hoa
	Cc bi tập tư lm
	Bài 11 Kiểu cấu trúc
	Giới thiệu tổng quan
	Kiểu enum
	Kiểu cấu trúc
	Cc bi tập minh hoa
	Cc bi tập tư lm
	Bài 12 Cc cấu trúc tự trỏ - Kiểu hợp
	Giới thiệu tổng quan

	Cc cấu trúc tự trở
	Kiểu hợp
	Cc bài tập minh họa
	Cc bài tập tự luận
	Bài 13 Đề qui
	Giới thiệu tổng quan
	Khái niệm chung về đề qui
	Cách dùng đề qui
	Một vài ứng dụng của đề qui
	Cc bài tập minh họa
	Cc bài tập tự luận
	Bài 14 Kiểu tập tin (File)
	Giới thiệu tổng quan
	Khái niệm về tập tin
	Khai báo dữ liệu
	Cc kiểu nhập xuất
	Cc thao tác trên tập tin
	Cc hàm nhập xuất
	Cc hàm di chuyển con trỏ
	Cc hàm quản lý thư mục
	Cc bài tập minh họa
	Cc bài tập tự luận
	Bài 15 Dự án (Project)
	Giới thiệu tổng quan
	Phân chia chương trình thành nhiều tập tin
	Dịch một chương trình chứa nhiều tập tin
	Biến toàn cục
	Cc bài tập minh họa
	Cc bài tập tự luận
	Phụ lục
	Phụ lục A: Cc bài tập tự luận
	Phụ lục B: Các thư viện trong C
	Phụ lục C: Bảng mã ASCII
	Phụ lục D: Sử dụng cc công cụ Debug
	Tài liệu tham khảo
	Tài liệu tham khảo
	C Programming, v2.7
	A Guide to C++

-  [C++ Technical Information Docs](#)
-  [C++ Frequently Asked Questions](#)

