NGUYỄN LINH GIANG (Chủ biên)
LÊ VĂN THÁI - KIỀU XUÂN THỰC

GIÁO TRÌNH

KỸ THUẬT LẬP TRÌNH C

NHÀ XUẤT BẢN GIÁO DỤC

NGUYỄN LINH GIANG (Chủ biên)
LÊ VĂN THÁI – KIỀU XUÂN THỰC

Giáo trình

KỸ THUẬT LẬP TRÌNH C

(DÙNG CHO SINH VIÊN HỆ CAO ĐẲNG)

NHÀ XUẤT BẢN GIÁO DỤC

192–2007/CXB/6–411/GD

Mã số: 7B681M7–DAI

MỤC LỤC

Trang

Chương 1

CÁC KHÁI NIỆM CƠ BẢN

I – GIỚI THIỆU

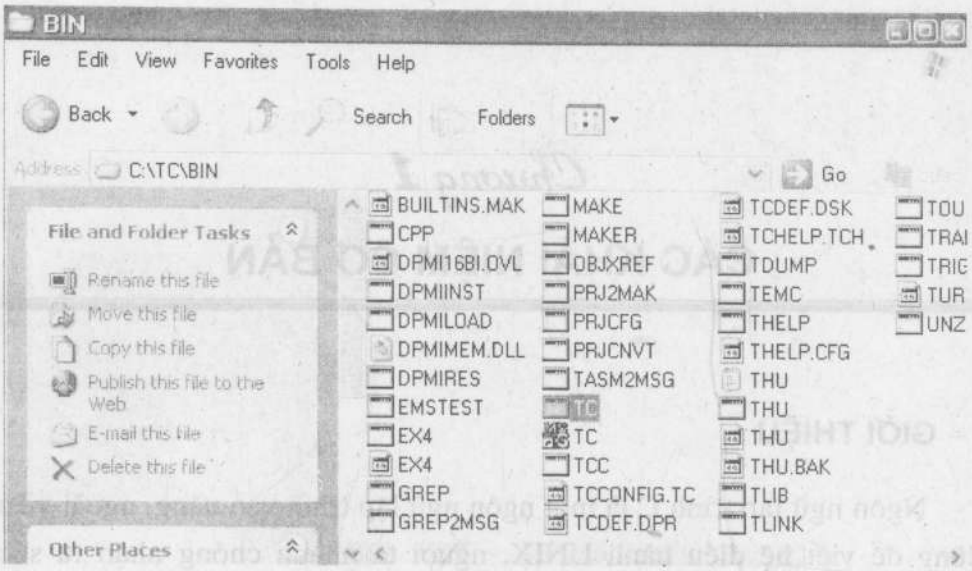
Ngôn ngữ lập trình C là một ngôn ngữ lập trình vạn năng, ngoài việc dùng để viết hệ điều hành UNIX, người ta nhanh chóng nhận ra sức mạnh của nó trong việc xử lý các vấn đề của Tin học. Ngôn ngữ lập trình C được gọi là "Ngôn ngữ lập trình hệ thống" vì nó được dùng cho việc viết hệ điều hành; nó cũng tiện lợi cho việc viết các chương trình xử lý số, xử lý văn bản, cơ sở dữ liệu, các chương trình ứng dụng trong công nghiệp và dân dụng. Trong thực tế, người ta thường dùng trình dịch Turbo C hoặc Borland C của hãng Borland. Ngày nay, do xu hướng chuyển sang lập trình hướng đối tượng nên ngôn ngữ lập trình C còn được phát triển thành ngôn ngữ lập trình hướng đối tượng có tên là C++. Ngoài Borland, hãng Microsoft cũng cung cấp bộ phát triển tích hợp Visual C++ trong bộ Visual Studio. Giáo trình này tập trung đề cập vào trình dịch Turbo C (TC) của Borland.

1. Khởi động TC, giao diện soạn thảo chương trình C

1.1. Khởi động TC

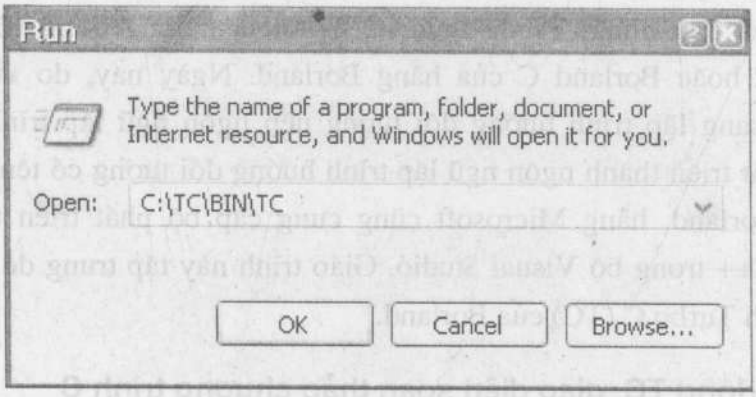
Phần mềm TC thường được cài đặt trong ổ đĩa C và trong thư mục TC. Để khởi động TC ta có thể sử dụng nhiều cách khác nhau như khởi động từ DOS, NC, Explore, hoặc RUN trên menu START.

– Muốn khởi động từ DOS, NC, Explore, người sử dụng vào thư mục TC\BIN và chạy file TC.EXE (hình 1.1).



Hình 1.1

– Khởi động từ menu START ta thực hiện như sau : Kích chuột vào menu START, chọn RUN, gõ vào C:\TC\BIN\TC, sau đó chọn OK hoặc ấn phím ENTER (hình 1.2).



Hình 1.2

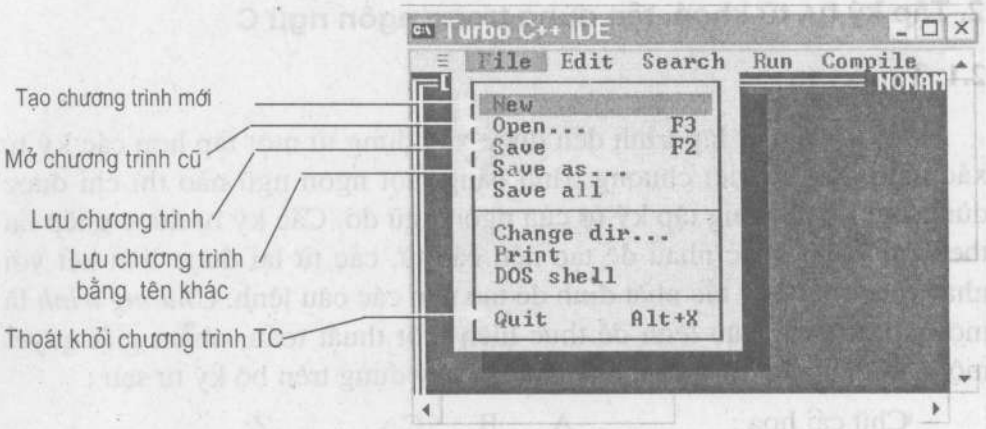
1.2. Giao diện của TC

Sau khi khởi động TC, ta có giao diện của TC, trong đó :

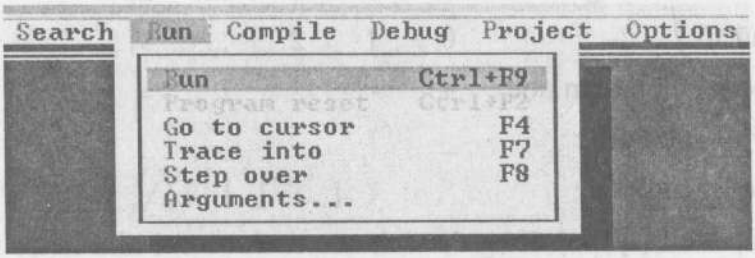
– Menu *FILE* : Gồm tạo chương trình mới, mở chương trình cũ,... (hình 1.3).

– Menu *RUN* : Chọn Run để chạy chương trình đang mở (hình 1.4).

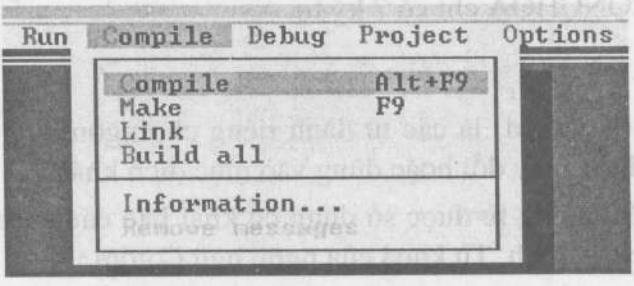
– Menu *COMPILE* : Chọn Compile để dịch chương trình và kiểm tra lỗi (hình 1.5).



Hình 1.3



Hình 1.4



Hình 1.5

Lưu ý : Ta có thể sử dụng một số phím tắt khi thực hiện chương trình TC :

F2 : Lưu chương trình đang soạn;

F3 : Mở chương trình cũ;

F7 : Chạy từng dòng lệnh trong chương trình;

F9 : Dịch chương trình và kiểm tra lỗi;

Ctrl + F9 : Chạy chương trình đang mở.

2. Tập ký tự, từ khoá, tên dùng trong ngôn ngữ C

2.1. Tập ký tự

Mỗi ngôn ngữ lập trình đều được xây dựng từ một tập hợp các ký tự xác định. Khi ta viết chương trình bằng một ngôn ngữ nào thì chỉ được dùng các ký tự trong tập ký tự của ngôn ngữ đó. Các ký tự được ghép lại theo các cách khác nhau để tạo nên các từ, các từ lại được liên kết với nhau theo một quy tắc nhất định để tạo nên các câu lệnh. *Chương trình* là một tập hợp các câu lệnh để thực hiện một thuật toán, nhằm giải quyết một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau :

- Chữ cái hoa : A B C ... Z;
- Chữ cái thường : a b c ... z;
- Chữ số : 0 1 2 ... 9;
- Các ký hiệu toán học : + - * / = ( );
- Ký tự gạch nối : _
- Các ký tự khác : . , : ; [ ] { } ! \ & % # $...

– Dấu cách (space) dùng để tách các từ được coi như là ký tự trắng. Ví dụ, chữ CONG HOA có 8 ký tự (tính cả dấu cách giữa CONG và HOA), còn CONGHOA chỉ có 7 ký tự.

2.2. Từ khóa

Từ khoá (keyword) là các từ dành riêng cho ngôn ngữ mà người sử dụng không được thay đổi hoặc dùng vào mục đích khác.

Từ khoá là những từ được sử dụng để khai báo các kiểu dữ liệu, các toán tử và các câu lệnh. Từ khoá của ngôn ngữ C gồm :

`break`	`char`	`continue`	`case`	`do`
`double`	`default`	`else`	`float`	`for`
`goto`	`int`	`if`	`long`	`return`
`struct`	`switch`	`unsigned`	`while`	`typedef`
`union`	`void`	`volatile`	`...`	

Ý nghĩa và cách sử dụng của mỗi từ khoá sẽ được đề cập ở những phần sau.

Khi lập trình cần chú ý :

– Không được dùng các từ khoá để đặt tên cho các đối tượng như hằng, biến, mảng,...

– Từ khoá sử dụng trong chương trình phải viết bằng chữ thường.

Ví dụ : Viết từ khoá khai báo biến kiểu thực là float chứ không phải là Float, FLOAT, fLOAT,...

2.3. Tên

Tên là một dãy các ký tự liền nhau bao gồm : chữ cái, số và gạch nối để định danh các biến, hằng, hàm, cấu trúc, con trỏ, tệp, nhãn,... trong chương trình. Trong C tên được đặt phải thoả mãn các yêu cầu sau :

– Ký tự đầu tiên của tên phải là chữ hoặc gạch nối;

– Không được trùng với từ khoá;

– Không được chứa các ký tự đặc biệt như dấu cách, dấu chấm câu,...

– Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 (trong cửa sổ Turbo C vào Option → Compiler → Source → Identifier length).

Ví dụ :

– Các tên đúng :

a_1 delta x1 _Line GAMA

– Các tên sai :

3PP	(ký tự đầu tiên là số)	printf	(trùng với từ khoá)
X#2	(sử dụng ký tự #)	Den ta	(sử dụng dấu cách)
F(x)	(sử dụng các dấu (,))	X - 3	(sử dụng dấu -)

Trong ngôn ngữ C, tên bằng chữ thường và chữ hoa là khác nhau. Ví dụ, tên AB khác với ab. Trong ngôn ngữ C người ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết cho các đại lượng khác như biến, biến mảng, hàm và cấu trúc.

II – CÁC KIỂU DỮ LIỆU CƠ SỞ

Ngôn ngữ C có các kiểu dữ liệu cơ sở là : kiểu ký tự (char), kiểu số nguyên (int), kiểu dấu phẩy động (chính xác đơn (float), chính xác kép (double)), kiểu void.

1. Kiểu ký tự

Một giá trị kiểu ký tự (char) chiếm một byte trong bộ nhớ và biểu diễn một ký tự. Giá trị lưu trong bộ nhớ chính là mã ASCII của ký tự cần biểu diễn.

Ví dụ :

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

Có hai kiểu ký tự sau :

Kiểu ký tự	Phạm vi biểu diễn	Số ký tự	Kích thước
char (signed char)	−128 → 127	256	1 byte
unsigned char	0 → 255	256	1 byte

Phân loại ký tự :

Có thể chia 256 ký tự thành ba nhóm :

– Nhóm 1 : Nhóm các ký tự điều khiển có mã từ 0 đến 31;

– Nhóm 2 : Nhóm các ký tự văn bản có mã từ 32 đến 126;

– Nhóm 3 : Nhóm các ký tự đồ hoạ có mã từ 127 đến 255.

2. Kiểu số nguyên

Trong C có bốn kiểu số nguyên :

– Số nguyên (int);

– Số nguyên không dấu (unsigned int);

– Số nguyên dài (long hay long int);

– Số nguyên dài không dấu (unsigned long hay unsigned long int).

Phạm vi biểu diễn và kích thước của chúng như sau :

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

3. Kiểu số thực dấu phẩy động

Trong ngôn ngữ C sử dụng ba loại giá trị dấu phẩy động để biểu diễn số thực là float, double và long double. Phạm vi biểu diễn và kích thước của chúng được cho trong bảng sau :

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

Theo bảng này thấy rằng, máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ 3.4E-38 ($3,4 \times 10^{-38}$) đến 3.4E+38 ($3,4 \times 10^{38}$). Các số có giá trị tuyệt đối nhỏ hơn 3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double, long double được hiểu theo nghĩa tương tự.

4. Kiểu không xác định (void)

Kiểu không xác định có thể gán cho bất kỳ một biến có kiểu nào và thường dùng để biểu diễn kết quả trả về của hàm hay của con trỏ.

5. Định nghĩa kiểu dữ liệu mới

Để định nghĩa một kiểu dữ liệu mới, ta sử dụng từ khoá **typedef**. Tên kiểu dữ liệu này sẽ được dùng để khai báo dữ liệu sau này. Nên chọn tên kiểu ngắn gọn để dễ nhớ.

Cú pháp :

```
typedef kiểu_dữ_liệu tên_kiểu_dữ_liệu_mới
```

Ví dụ : Đặt tên một kiểu int là **nguyen** ta khai báo như sau :

```
typedef int nguyen;
```

Sau đó có thể dùng kiểu **nguyen** để khai báo các biến, các mảng int :

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

III – HẰNG, BIẾN, MẢNG

1. Hằng

Hằng là một giá trị bất biến trong chương trình. Tương ứng với các kiểu dữ liệu ta có các loại hằng sau : hằng nguyên, hằng thực, hằng ký tự, hằng xâu ký tự (chuỗi),...

Hằng trong C được định nghĩa như sau :

```
#define  tên_hằng  giá_trị
```

Ví dụ :

```
#define  MAX 100
#define  pi 3.1415
```

1.1. Hằng int

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

Ví dụ :

```
#define dem 22
```

(Định nghĩa hằng int dem có giá trị là 22)

Chú ý : Cần phân biệt hai hằng 22 và 22.0, ở đây 22 là hằng nguyên, còn 22.0 là hằng thực.

1.2. Hằng long

Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647. Hằng long được viết theo cách thêm L hoặc *l* vào cuối.

Ví dụ : 124L hoặc 124*l*.

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

Ví dụ :

```
#define sl 9865056L
```

(Định nghĩa hằng long sl có giá trị là 9865056)

```
#define sl 9865056
```

(Định nghĩa hằng long sl có giá trị là 9865056)

1.3. Hằng int hệ 8

Hằng int hệ 8 được viết theo cách $0c_1c_2c_3$..., ở đây c_i là một số nguyên dương trong khoảng từ 0 đến 7. Hằng int hệ 8 luôn luôn nhận giá trị dương.

Ví dụ :

```
#define h8 0345
```

(Định nghĩa hằng int hệ 8 có giá trị là 3*8*8 + 4*8 + 5 = 229)

1.4. Hằng int hệ 16

Hằng in hệ 16 sử dụng 16 ký tự : 0, 1..., 9, A, B, C, D, E, F. Hằng số hệ 16 có dạng $0xc_1c_2c_3$... hoặc $0Xc_1c_2c_3$..., ở đây c_i là một số trong hệ 16, c_i có giá trị trong khoảng từ 0 đến F, trong đó :

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

Ví dụ : `#define h16 0xa5`

1.5. Hằng ký tự

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

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

Ví dụ : '9' - '0' = 57 - 48 = 9.

Ví dụ : `#define kt 'a'`

(Định nghĩa hằng ký tự kt có giá trị là 97)

Hằng ký tự còn có thể được viết theo cách sau : '$\c_1c_2c_3$'

Trong đó $c_1c_2c_3$ là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

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

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

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

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

1.6. Hằng xâu ký tự

Hằng xâu ký tự là một dãy ký tự bất kỳ, được đặt trong cặp dấu nháy kép (" "). Hằng được lưu trữ trong một mảng ký tự mà ký tự cuối cùng là rỗng (\0).

Ví dụ : `#define xau1 "Ha noi"`

Cần phân biệt hai hằng 'a' và "a" : 'a' là hằng ký tự được lưu trữ trong 1 byte; còn "a" là hằng xâu ký tự được lưu trữ trong 2 byte, vì nó là một mảng hai phần tử (phần tử thứ nhất chứa chữ 'a', còn phần tử thứ hai chứa giá trị null (\0)).

2. Biến (variable)

Biến là nơi để chứa các đối tượng của chương trình. Ví dụ, để chứa và thao tác một số nguyên ta cần khai báo một biến kiểu nguyên, để chứa và thao tác một số thực ta cần khai báo một biến kiểu thực,...

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

```
Kiểu_dữ_liệu_của_biến  tên_biến
```

Nếu có nhiều biến cùng kiểu dữ liệu, thì có thể khai báo trên cùng một dòng và ngăn cách giữa các tên biến là dấu phẩy ','. Tên biến là tên tự đặt, nên nó phải tuân theo các yêu cầu của tên tự đặt.

Ví dụ :

```
int a, b, c;        (khai báo ba biến nguyên là a, b, c)
char kt1, kt2;      (khai báo hai biến ký tự là kt1 và kt2)
float x, y;         (khai báo hai biến thực là x và y)
```

Các điểm cần lưu ý :

– Biến được khai báo theo kiểu nào thì nhận được các giá trị kiểu đó. Các biến kiểu char chỉ chứa được một ký tự. Để lưu trữ được một xâu ký tự cần sử dụng một mảng kiểu char.

– Có thể khởi tạo ngay giá trị ban đầu cho các biến bằng cách khai báo :

```
int a, b = 20, c, d = 40;
```

– Khi khai báo một biến thì máy tính sẽ dành ra số byte liên tiếp trong bộ nhớ trong đúng bằng số byte nhớ cần thiết để biểu diễn cho kiểu dữ liệu của biến dùng để lưu trữ dữ liệu cho biến. Số hiệu của byte đầu chính là địa chỉ của biến.

– Để lấy địa chỉ của một biến, sử dụng phép toán : &tên_biến, ví dụ &a trả về cho ta địa chỉ của biến a.

– Các biến khai báo bên trong thân một hàm, kể cả hàm main được gọi là *biến cục bộ* hay *biến tự động*. Các biến này chỉ có tác dụng ở bên trong hàm có khai báo chúng, chúng được cấp phát bộ nhớ khi hàm được thực hiện và bị thu hồi bộ nhớ khi kết thúc hàm. Ngược lại, các biến khai báo bên ngoài hàm được gọi là *biến ngoài* hay *biến toàn cục*. Các biến ngoài có thể được sử dụng sau vị trí khai báo đến hết chương trình, và chúng được cấp phát bộ nhớ trong suốt thời gian làm việc của chương trình.

3. Mảng (array)

Mảng là một tập hợp nhiều phần tử, có cùng một kiểu giá trị và chung một tên. Kiểu của mảng chính là kiểu của các phần tử mảng, vì vậy có bao nhiêu kiểu biến thì có bấy nhiêu kiểu mảng.

Khai báo mảng :

```
kieu_mang  ten_mang[chỉ số 1][chỉ số 2][...];
```

Ví dụ : Các khai báo :

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

xác định bốn mảng là :

– Mảng a là mảng kiểu nguyên, một chiều, gồm 10 phần tử nguyên từ a[0] đến a[9].

– Mảng b là mảng kiểu nguyên, hai chiều, gồm 8 phần tử nguyên là b[0][0], b[0][1], b[1][0], b[1][1], b[2][0], b[2][1], b[3][0] và b[3][1].

– Mảng x là mảng kiểu thực, một chiều, gồm 5 phần tử kiểu thực từ x[0] đến x[4].

– Mảng y là mảng kiểu thực, hai chiều, gồm 9 phần tử thực là y[0][0], y[0][1], y[0][2], y[1][0], y[1][1],... đến y[2][2].

Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác, các phần tử của mảng có địa chỉ liên tiếp nhau. Phần tử cụ thể của mảng được xác định theo tên và chỉ số của nó. Ví dụ, phần tử ở hàng 2 cột 1 sẽ là a[2][1].

Chú ý : Biểu thức dùng làm chỉ số có thể thực, khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng.

Ví dụ : a[2.5] là a[2];

b[1.9] là a[1].

– Khi chỉ số vượt ra ngoài kích thước của mảng, máy không báo lỗi khi dịch và chạy chương trình, nhưng sẽ truy cập đến vùng nhớ bên ngoài mảng và có thể gây lỗi cho chương trình.

– Trong mảng thì tên mảng biểu thị địa chỉ đầu tiên của mảng. Như vậy, có thể dùng a thay cho &a[0].

– Có thể khởi tạo giá trị ban đầu cho biến mảng bằng cách thực hiện như sau :

```
float x[6] = {3.2, 0, 5.1, 23, 0, 42};
int a[3][2] =   {
                    {25,31},
                    {12,13},
                    {45,15}
                }
```

– Khi khởi tạo 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[] = {0, 5.1, 23, 0, 42};`

– Khởi tạo của một mảng char có thể là một danh sách các hằng ký tự, hoặc một hằng xâu ký tự. Ví dụ, có thể khai báo :

```
char ten[] = {'h','a','g'};
char ho[] = 'tran';
```

IV – CẤU TRÚC CỦA CHƯƠNG TRÌNH C

1. Chú thích trong chương trình

Các lời chú thích cho chương trình, đoạn chương trình, hàm, câu lệnh có thể đưa vào ở bất kỳ chỗ nào của chương trình để cho chương trình dễ hiểu, dễ đọc hơn mà không làm ảnh hưởng đến các phần khác. Lời chú thích được đặt giữa cặp dấu /* và */, hoặc nếu có một dòng thì chỉ cần đặt dấu (//) ở đầu dòng.

Ví dụ :

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

2. Lệnh và khối lệnh

2.1. Lệnh

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

Ví dụ :
```
x = 0;
++ i;
```

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

2.2. Khối lệnh

Khối lệnh là một dãy các câu lệnh được bao bởi cặp dấu { }.

Ví dụ :

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

Khối lệnh thường được sử dụng trong trường hợp có nhiều lệnh được thực hiện khi có một điều kiện nào đó thoả mãn.

2.3. Sự lồng nhau của các khối lệnh và phạm vi hoạt động của các biến, mảng

Bên trong một khối lệnh lại có thể viết khối lệnh khác được gọi là sự lồng nhau của các khối lệnh. Ngôn ngữ C không hạn chế số lượng khối lệnh lồng nhau.

Trong khối lệnh có thể sử dụng các biến, mảng đã được khai báo ở ngoài khối lệnh và các biến, mảng được khai báo ở đầu khối lệnh.

Ví dụ :

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

Tuy nhiên cần lưu ý, khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại trong thời gian máy thực hiện các lệnh ở bên trong khối lệnh và chúng lập tức bị giải phóng ngay sau khi máy ra khỏi khối lệnh.

Cần lưu ý :

– Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bên ngoài khối lệnh đó và ở bên ngoài

một khối lệnh không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh.

– Nếu bên trong một khối lệnh dùng một biến, mảng trùng tên với một biến, mảng ở ngoài khối lệnh thì các lệnh trong khối lệnh chỉ tác động trên biến, mảng trong khối lệnh mà không làm thay đổi giá trị của biến, mảng đó ở bên ngoài khối lệnh.

Ví dụ : Xét đoạn chương trình sau :

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

Khi thực hiện đoạn chương trình sẽ cho kết quả :

a trong = 4 b = 6

a ngoài = 5 b = 6

3. Cấu trúc cơ bản của chương trình C

Cấu trúc cơ bản của chương trình C như sau :

- Các #include (khai báo các tệp tin header sử dụng trong chương trình)
- Các #define (các định nghĩa hằng)
- typedef (các định nghĩa kiểu dữ liệu)
- Khai báo các đối tượng dữ liệu ngoài (biến, mảng, cấu trúc,...).
- Khai báo nguyên mẫu các hàm.
- Hàm main().
- Định nghĩa các hàm (hàm main có thể đặt sau hoặc xen vào giữa các hàm khác).

Trong đó cần lưu ý :

– Một chương trình bao gồm một hoặc nhiều hàm, trong đó hàm main() là thành phần bắt buộc của chương trình.

– Hàm là một đơn vị độc lập của chương trình, do đó không được phép xây dựng một hàm bên trong hàm khác, mỗi hàm có các biến, mảng... riêng của nó và chúng chỉ được sử dụng nội bộ bên trong hàm.

– Chương trình bắt đầu thực hiện các câu lệnh đầu tiên của hàm main() và kết thúc khi gặp dấu '}' cuối cùng của hàm này. Khi chương trình làm việc gặp lời gọi hàm, máy sẽ chuyển từ hàm này sang hàm khác.

– Việc truyền dữ liệu và kết quả từ hàm này sang hàm khác được thực hiện bằng cách sử dụng đối của hàml hoặc sử dụng biến ngoài, mảng ngoài, biến tĩnh ngoài, mảng tĩnh ngoài.

– Các hàm cũng có thể được định nghĩa trước hàm main() và ngay sau khai báo nguyên mẫu hàm.

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

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

4. Một số điểm cần lưu ý khi viết chương trình

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

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

– Trong chương trình, khi sử dụng các hàm chuẩn, ví dụ như printf(), getch(),... mà các hàm này lại chứa trong file stdio.h ở thư mục của C, vì vậy ở đầu chương trình phải khai báo sử dụng stdio.h bằng toán tử # : #include "stdio.h ".

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

V – BÀI TẬP MINH HOẠ

✦ Chương trình minh hoạ việc sử dụng ký tự điều khiển (\n)

Bài 1. Viết chương trình in ra màn hình dòng chữ "Hello, world!".

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

Bài 2. Viết chương trình in ra màn hình dòng chữ :

Hello,

world.

```
/* Chương trình in ra hai dòng : Hello, World */
# include<stdio.h>
void main()
  {
       printf(" \n Hello, \n world. \n");
  }
```

✦ **Chương trình minh hoạ cách khai báo, khởi đầu biến**

Bài 3. Viết chương trình minh hoạ các cách khai báo biến trong C.

```
/* Chương trình này minh hoạ cách khai báo biến
                                        trong C */
#include<stdio.h>
void main()
{
   char ki_tu;       /* Khai báo một ký 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 sau khi chạy chương trình trên :

a la mot ki tu.

15 la mot so nguyen!

27.620 la mot so thuc.

Bài 4. Chương trình minh hoạ cách vừa khai báo, vừa khởi đầu một biến trong C.

```
/* Chương trình này minh hoạ 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 ký 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 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);
}
```

✦ **Giải các bài toán đơn giản**

Bài 5. Chương trình tính chu vi và diện tích hình tròn khi biết bán kính r là một hằng số có giá trị là 3.1.

```
/* Chương trình tính chu vi và diện tích hình tròn,
biết bán kính r là một hằng số có giá trị là 3.1*/
# 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 r 3.1
void main()
{
   float cv,dt;     /* khai bao 2 bien chu vi va
                           dien tich kieu so thuc */
   cv = 2*r*M_PI;   /* tinh chu vi, su dung hang
                            M_PI co trong math.h */
   dt = M_PI*r*r;       /* Tinh dien tich */
   printf("\n Chu vi = %10.2f \n Dien tich =
                  %10.2f",cv,dt); /* In ket qua */
   getch();             /* Tam dung chuong trinh */
}
```

Kết quả sau khi thực hiện chương trình :

Chu vi = 19.47

Dien tich = 30.18

Chú ý :

%d là mã đặc tả biểu diễn các biến có kiểu số nguyên.

%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.

Bài 6. Chương trình tính diện tích hình tròn khi biết chu vi là hằng cv có giá trị là 9.1.

```
# 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 thuc */
   r = cv/(2*M_PI);     /* tinh ban kinh, su dung
                           hang M_PI co trong math.h */
   dt = M_PI*r*r;       /* Tinh dien tich */
   printf("\n Dien tich = %10.2f",dt);
                       /* In ket qua len man hinh*/
   getch();             /* Tam dung chuong trinh */
}
```

Kết quả sau khi thực hiện chương trình :

Dien tich = 6.59

Bài 7. Chương trình tính chu vi hình tròn khi biết diện tích là biến dt có giá trị khởi đầu là 6.1.

```
# 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 thuc va khoi tao bien dt */
   r = sqrt(dt/M_PI);
       /* tinh ban kinh, trong do su dung ham can
                  bac hai sqrt() co  trong math.h */
   cv = 2*M_PI*r;     /* Tinh chu vi */
```

```
    printf("\n Chu vi = %10.2f",cv);
                    /* In ket qua len man hinh */
    getch();        /* Tam dung chuong trinh */
}
```

Kết quả sau khi thực hiện chương trình :

Chu vi = 8.75

✦ Một số quy 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("\n Chu vi = %10.2f \
                \n Dien 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 tệp tin có chứa hàm muốn sử dụng (ở cuối dòng khai báo này không có dấu ;).

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

VI – BÀI TẬP TỰ GIẢI

Bài 1. Viết chương trình tính diện tích của tam giác khi biết ba cạnh là ba biến a, b, c được khởi đầu : a = 3.1, b = 2.2, c = 4.3 và công thức tính diện tích là :

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \text{ với } p = \frac{a+b+c}{2}.$$

Bài 2. Viết chương trình tính chu vi hình chữ nhật khi biết diện tích là 9.1 và tỷ lệ giữa chiều dài và chiều rộng là 2.

Chương 2

BIỂU THỨC VÀ CÁC PHÉP TOÁN TRONG C

I – BIỂU THỨC

Biểu thức là một sự kết hợp các giá trị (hằng, biến, hàm) bằng các phép tính để sinh ra một giá trị mới. Kiểu của biểu thức là kiểu của giá trị mà nó sinh ra.

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 phép nhân (*)
   và dấu bằng (=) 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). */
```

II – CÁC PHÉP TOÁN

1. Các phép toán số học

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  x = 15.0;
float  y = 3.0;
float  phepcong = x + y;
            /* 15.0 + 3.0 = 18.000000 */
float  phepchia = x/y;
            /* 15.0/3.0 = 5.000000 */
```

Chú ý rằng, 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 x = 10; int y = 3;
int KQ1 = x/y;
        /* cho kết quả là số nguyên : 10/3 = 3 */
int KQ2 = x%y;
        /* 10%3 = 1    (10/3 = 3 dư 1) */
```

Ngoài các phép toán trên, ngôn ngữ C còn trang bị cho chúng ta một số hàm toán học chuẩn được khai báo trong thư viện math.h (tham khảo trong các tài liệu liên quan).

2. Các phép toán thao tác bit

Các phép toán 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 (^)**

Bảng giá trị chân lý

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 (>>), bù bit (~)**

– Biểu thức a << n sẽ dịch chuyển các bit trong a sang trái n vị trí.

– Biểu thức a >> n sẽ dịch chuyển các bit trong a sang phải n vị trí.

Đối với kiểu không dấu ta có :

$a << n = a*(2^n)$;

$a >> n = a/(2^n)$.

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 ~ : ~1 = 0; ~0 = 1.

Ví dụ : Giả thiết một số nguyên (int) được biểu diễn bằng 16 bit.

```
unsigned int a = 3737;        /* 0000111010011001 */
```

thì sau khai báo

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

b có giá trị là : 1111000101100110.

3. Các phép toán quan hệ và lôgic

3.1. Các phép toán quan hệ

Các phép toán quan hệ so sánh giá trị của các toán hạng, rồi cho kết quả 0 (sai), hoặc 1 (đúng). Các phép so sánh bao gồm : bằng (==), khác (!=), lớn hơn (>), lớn hơn hoặc bằng (>=), nhỏ hơn (<), nhỏ hơn hoặc bằng (<=).

Ví dụ :

```
float kq1 = (3 > 5); /* kq1 = (3 > 5) = 0.000000*/
float kq2 = (5 >= 3); /*kq2 = (5 > 3) = 1.000000*/
float kq3 = (12 + 5 > != 15);
                    /* kq3 = (8 != 5) = 1.000000 */
float kq4 = (15 != 3*5);
                  /* kq4 = (15 != 3*5) = 0.000000 */
```

3.2. Các phép toán lôgic

Các phép toán lôgic dùng để kết hợp các biểu thức khác nhau thành một biểu thức lôgic. Kết quả của các phép toán này cho giá trị 0 (sai), hoặc 1 (đúng). Bao gồm các phép toán : NOT (!); AND (&&); OR (||).

Bảng giá trị chân lý của các phép toán này như sau :

Toán hạng 1	Toán hạng 2	Kết quả		
A	B	!A	A&&B	A||B
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

4. Chuyển đổi kiểu dữ liệu (ép kiểu)

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 lần lượt là 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 */
```

5. Các phép toán tăng giảm

Các phép toán tăng giảm 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 hai 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 với biến ++, biến -- thì ngược lại.

Ví dụ :

Nếu i = 3 thì câu lệnh `tong = ++ i;` sẽ gán 4 cho tong, còn câu lệnh `tong = i ++;` sẽ gán 3 cho tong mặc dù trong cả hai trường hợp i đều trở thành 4.

6. Câu lệnh gán

Trong C, lệnh gán được thực hiện bằng toán tử =. Cấu trúc câu lệnh gán như sau :

```
Biến = biểu_thức
```

Lệnh gán có thể sử dụng để khởi tạo giá trị ngay trong khi khai báo biến.

Ví dụ : `int c, a = 3, b = 4; c = a + b;`

Một số lưu ý đối với lệnh gán :

– Có thể rút gọn các phép toán hai ngôi dạng :

i = i + 3 thành i+ = 3;

i = i*x thành i* = x;

x = x*(y + z) thành x* = y + z;

– Có thể sử dụng biểu thức gán để gán giá trị cho một biến.

Ví dụ : `a = b = 5;`

có nghĩa là gán giá trị biểu thức b = 5 cho biến a, kết quả là b = 5 và a = 5.

```
z = (x = 2)*(y = 6);
```

cho kết quả là x = 2, y = 6 và z = 12.

III – BIỂU THỨC ĐIỀU KIỆN

Biểu thức điều kiện là biểu thức có dạng :

```
bt1 ? bt2 : bt3;
```

trong đó : bt1, bt2, bt3 là các biểu thức.

Giá trị trả về là bt2 nếu bt1 khác 0 (bt1 đúng) và là bt3 nếu bt1 bằng 0 (bt1 sai).

Ví dụ :

```
z = x < y ? x : y;
```

có nghĩa là nếu x nhỏ hơn y thì z = x, nếu z không nhỏ hơn y thì z = y.

IV – BÀI TẬP MINH HOẠ

✦ **Bài tập mẫu về phép toán số học :** phép chia nguyên (/) và phép modulo(%)

Bài 1. Cho biết giá trị trả về của hai phép toán 8 / -5 và 8% -5.

Kết quả sau khi thực hiện chương trình :

8/ -5 = -1; 8% - 5 = 3

✦ **Bài tập mẫu về phép toán thao tác trên bit**

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

a) ~ a;

b) a | b;

c) a ^ b;

d) a >> 2;

e) a << 2;

f) a << -2.

Kết quả :

a = 0xc0b3 = 1100000010110011

b = 0x2435 = 0010010000110101

a) ~a = 0011111101001100 = 0x3f4c

b) a | b = 1110010010110111 = 0xe4b7

c) a ^ b = 1110010010000110 = 0xe486

d) a >> 2 = 0011000000101100 = 0x302c

e) a << 2 = 0000001011001100 = 0x02cc

f) a << -2 = 0000000000000000 = 0x0

✦ Bài tập mẫu về biểu thức quan hệ và lôgic

Bài 3. Hãy cho biết giá trị của j khi thực hiện đoạn chương trình :

```
int j;
char c = '1';
j = (c <= '9') &&(c >= '0');
```

Kết quả :

j = 1

✦ 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;

✦ 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.

✦ Bài tập mẫu về câu lệnh gán

Bài 7.

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) 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).

✦ Bài tập mẫu về biểu thức điều kiện

Bài 8. Hãy cho biết giá trị của b khi thực hiện đoạn chương trình sau :

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

Kết quả :

b = 2

Bài 9. 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 = 2;

b) a = 1;

c) a = 1.

V – BÀI TẬP TỰ GIẢI

Bài 1. Cho khai báo biến sau :

```
int pint;
float a;
char c;
double pd;
```

Hãy chọn phát biểu đúng :

a) `(double) pd = a;`

b) `c = + pint +;`

c) `print = (int) pd;`

d) `a = &pint;`

Bài 2. Cho khai báo biến sau :

```
int a,p;
double b,c;
```

Hãy chọn phát biểu đúng :

a) `p = (int) b + (c* = 2);`

b) `p = a + (1, b - = 1);`

c) `p = c;`

d) `a = "abc";`

Bài 3. Cho khai báo biến sau :

```
char a, p;
int b, pint;
```

Hãy chọn phát biểu sai :

a) `pint <<= a;`

b) `p ->> b;`

c) `a + = 1 + b - (double) 1;`

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) ;
}
```

Hãy chọn kết quả in ra đúng :

a) Chương trình sai cú pháp

b) B 68

c) Chương trình in ra trị không xác định

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);
}
```

Hãy chọn kết quả in ra đúng :

a) a = 12, i = 5, f = 6.72

b) a = 5, i = 6, f = 6.533333

c) a = 4, i = 5, f = 5.6

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\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);
}
```

Chương 3

NHẬP, XUẤT DỮ LIỆU TRONG C

I – CÁC HÀM NHẬP, XUẤT THUỘC STDIO.H

1. Hàm đưa kết quả ra màn hình (printf)

Cú pháp :

```
printf ("chuỗi_điều_khiển",[các_biểu_thức])
```

Chuỗi điều khiển gồm ba loại :

– Chuỗi ký tự mang tính chất thông báo (hằng chuỗi).

– Các ký tự điều khiển :

\n : sang dòng mới;

\t : dấu tab;

\b : lùi lại một bước;

\f : sang trang mới.

– Các mã đặc tả để in các biểu thức tương ứng (mỗi biểu thức khi in phải có một đặc tả).

Dạng tổng quát của đặc tả :

```
%[-][f][.p] ký_tự_chuyển_dạng
```

Trong đó :

+ Dấu % là ký tự để đưa ký tự chuyển dạng vào, bắt buộc phải có.

+ Dấu trừ [-] :

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

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

+ [f] :

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

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

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

+ [.p] :

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

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

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

• Khi đối là xâu ký tự : Nếu p nhỏ hơn độ dài của xâu thì chỉ p ký tự đầu tiên của xâu được in ra. Nếu không có p hoặc nếu p lớn hơn hay bằng độ dài của xâu thì cả xâu ký tự sẽ được in ra.

Các ký tự chuyển dạng và ý nghĩa của nó :

Ký tự chuyển dạng	Kiểu dữ liệu	Ý nghĩa
c	char	Đối là ký tự
d/di	int	Đối là số nguyên
ld /li	long	Đối là số nguyên dài
f	float hoặc double	Đối là số thực, dạng thập phân [-]m...m.n...n, trong đó độ dài của n...n là p.
e	float hoặc double	Đối là số thực, dạng thập phân [-]m.n...nE[+hoặc -]xx, trong đó độ dài của n...n là p - 1.
s	xâu ký tự (chuỗi)	Đối là chuỗi
u	int	Số nguyên hệ 10 không dấu

Ký tự chuyển dạng	Kiểu dữ liệu	Ý nghĩa
o	int	Số nguyên hệ 8 không dấu
lo	long	Số nguyên hệ 8 không dấu
x	int	Số nguyên hệ 16 không dấu
lx	long	Số nguyên hệ 16 không dấu
g	float hay double	Không in ra các số 0 vô nghĩa
c	float hoặc double	Đối trong dạng thập phân

Ví dụ :

```
/* Chương trình này minh hoạ các kiểu dữ liệu
                                        trong C */
printf("gia tri 92 dung kieu d = %d. \n", 92);
   /* gia tri 92 dung kieu d = 92. */
printf("gia tri 92 dung kieu i = %i. \n", 92);
   /* gia tri 92 dung kieu i = 92. */
printf("gia tri 92 dung kieu u = %u. \n", 92);
   /* gia tri 92 dung kieu u = 92. */
printf("gia tri 92 dung kieu o = %o. \n", 92);
   /* gia tri 92 dung kieu o = 134. */
printf("gia tri 92 dung kieu x = %x. \n", 92);
   /* gia tri 92 dung kieu x = 5c. */
printf("gia tri 92 dung kieu c = %c. \n", 92);
   /* gia tri 92 dung kieu c = \. */
printf("ky tu '9' dung kieu c = %c. \n", '9');
   /* ky tu '9' dung kieu c = 9. */
printf("chuoi 92 dung kieu s = %s. \n"," 92");
   /* chuoi 92 dung kieu s = 92. */
```

2. Hàm scanf

Cú pháp :

```
scanf (các_đặc_tả, < danh_sách_địa_chỉ_các_biến
                        tương_ứng_với_các_đặc_tả >);
```

Trong đó các đặc tả có dạng % < ký tự đặc tả >, mỗi biến muốn nhập giá trị phải có một đặc tả tương ứng.

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

```
%[*][d...d] ký tự chuyển dạng.
```

Ở đây :

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

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

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

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

Ví dụ :

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

Với dòng vào : 54.32e-1 25 12452348a. Kết quả là lệnh scanf sẽ gán :

5.432 cho x

25.0 cho y

124 cho a

xâu "523" và dấu kết thúc \0 cho ch

xâu "48a" và dấu kết thúc \0 cho ct.

– Ký tự chuyển dạng :

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

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

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

Các ký tự chuyển dạng và ý nghĩa của nó :

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

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

Ví dụ :

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

Với dòng vào : 35 13145 xyz 584235.

Sẽ gán :

35 cho a

xâu "13145" cho ch

xâu "xyz' cho ck

584 cho b.

Ví dụ 1 : Chương trình minh hoạ 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 hoạ 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()      /* Ham chinh */
{
   int i;
   printf("Nhap chuoi 5 ky tu : ");
   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), với a là chuỗi (xâu) ký tự được nhập.

3. Hàm gets

Cú pháp :

```
gets(tên_của_mảng_ký_tự)
```

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

Ví dụ : Chương trình minh hoạ 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()     /* Ham chinh */
{
   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 */
}
```

4. Hàm getchar

Cú pháp :

```
getchar(void)
```

Hàm getchar dùng để nhận một ký tự từ bàn phím và trả về ký tự nhận được.

Ví dụ : Chương trình minh hoạ hàm getchar.

```
# include<stdio.h>
void main() /* Ham chinh */
{
   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 */
}
```

5. Lưu ý về các hàm scanf, gets và getchar

Các hàm trên là những hàm nhận dữ liệu từ stdin (dòng nhập chuẩn từ bàn phím, 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 Enter.

– Phần dữ liệu đã lấy bởi các hàm này được tự động xóa khỏi stdin, trong đó hàm gets sẽ xóa ký tự \n trong stdin; hàm scanf và getchar không xóa \n trong stdin. Vì vậy, 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 hai bị trôi và kết quả là a = 65 và b = < Enter >.

Giả sử ta nhập 65ABC < Enter >, khi đó lệnh scanf thứ hai bị trôi và kết quả là a = 65 và b = A.

Để các hàm sau hoạt động đúng, 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);
...
```

6. Hàm putchar

Cú pháp :

```
int putchar(int ch)
```

Hàm này xuất ký tự ch 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 */
```

7. Hàm puts

Cú pháp :

```
int puts(const char *s)
```

Hàm này 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 hiển thị. Hàm này khi xuất sẽ đưa thêm ký tự \n vào cuối. Kết quả trả về của hàm là '\n ' nếu thành công, là EOF khi có lỗi.

Ví dụ :

```
/* Chương trình minh hoạ 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 ký 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 ký tự cuối cùng được xuất. Khi có lỗi hàm trả về EOF.

II – CÁC HÀM NHẬP, XUẤT THUỘC CONIO.H

Các hàm nhập, xuất thuộc conio.h gồm : getch, getche, cprintf, cscanf và một số hàm thao tác màn hình, kiểm tra bàn phím thuộc conio.h.

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 ấn phím 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 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 cần phải đọc bộ đệm bàn phím thêm một lần nữa.

Ví dụ :

```
# include<stdio.h>
# include<conio.h>
void main() /* Ham chinh */
{
   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. Hàm cprintf

Hàm cprintf giống như hàm printf, nhưng màu của nội dung được in bởi hàm cprintf được ấn định bởi hàm textcolor.

3. Hàm cscanf

Hàm cscanf 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 cần phải khử ký tự \n trong bộ đệm bằng %*c hoặc bằng hàm getch.

Ví dụ :

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

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

4.1. Hàm xóa màn hình (clrscr)

Cú pháp :

```
clrscr()
```

Chức năng : clrscr (clear screen) là hàm xóa toàn bộ màn hình và sau khi xóa con trỏ sẽ ở vị trí góc phía trên bên trái.

4.2. Hàm đặt toạ độ (gotoxy)

Cú pháp :

```
gotoxy(int x, int y)
```

Chức năng : Hàm đặt con trỏ màn hình vào toạ độ (x, y) của màn hình, màn hình gồm 25 dòng và 80 cột. x là toạ độ cột, tính từ 1 đến 80; y là toạ độ dòng, tính từ 1 đến 25.

Ví dụ : `gotoxy(30,10);`

4.3. Hàm đặt màu nền (textbackground)

Cú pháp :

```
void textbackground(int color)
```

Chức năng : Chọn màu nền, trong đó color là một biểu thức nguyên có giá trị từ 0 đến 15 tương ứng với một trong các hằng số màu đầu tiên của bảng màu văn bản.

Ví dụ : `textbackground(3);`

tương đương với :

```
textbackground(CYAN);  /* Màu xanh cẩm thạch */
```

4.4. Hàm đặt màu chữ (textcolor)

Cú pháp :

```
void textcolor(int newColor)
```

Chức năng : Lựa chọn màu ký tự mới newColor, trong đó 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 với :

```
textcolor(RED);  /* Màu đỏ */
```

5. Hàm kiểm tra bộ đệm bàn phím (kbhit)

Cú pháp :

```
int kbhit(void)
```

Hàm này trả về giá trị 0 nếu bộ đệm bàn phím rỗng, khác 0 nếu bộ đệm bàn phím không rỗng.

III – BÀI TẬP MINH HOẠ

Bài 1. Viết chương trình nhập vào số dặm, sau đó đổi ra số kilômét 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("\n Kết quả là : %.2f dam => %.2f km",
                                          sdam,skm);
}
```

Kết quả :

Nhập số dặm => 23

Kết quả là : 23.00 dam => 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("\n Kết 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 thoả mãn điều kiện là ba cạnh của tam giác : a < b + c; c < a + b; b < a + c). Tính diện tích của tam giác. Biết :

$$s = \sqrt{p(p-a)(p-b)(p-c)}, \text{ với } p = \frac{a+b+c}{2}.$$

```
/* 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("\n Diện 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, khối 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 xâu tối đa 20 ký tự. */
```

```
    float khoi_luong;
    long don_gia;
    char  ma_chat_luong;
    unsigned so_luong;
    printf(" \n Nhập dữ liệu từ bàn phím : ");
    printf(" \n Tên mặt hàng => ");
    gets(ten_mat_hang);
    printf(" \n Khối lượng => ");
    scanf("%f", & khoi_luong);
    printf(" \n Đơn giá => ");
    scanf("%ld%*c", & don_gia);
    printf(" \n Mã chất lượng => ");
    scanf("%c ", &ma_chat_luong);
    printf(" \n Số lượng => ");
    scanf("%u ", &so_luong);
    printf("\n Tên mặt hàng : %s Khối lượng : %.2f
     Đơn gia : %ld Mã chất lượng : %c Số lượng : %u
                ",ten_mat_hang, khoi_luong, don_gia,
                              ma_chat_luong, so_luong);
    /* In kết quả ra màn hình */
}
```

IV – BÀI TẬP TỰ GIẢI

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. Biết tần số dao động của một mạch dao động LC được tính bằng công thức :

$$f = -\frac{2\pi}{\sqrt{LC}}.$$

Hãy viết chương trình tính tần số của một mạch dao động khi biết L và C.

Bài 3. Hệ số khuếch đại điện áp tính theo dB được chuyển bằng công thức :

$$K(dB) = 20\lg K$$

Hãy viết chương trình tính hệ số khuếch đại điện áp theo dB khi biết K (số lần).

Bài 4. Hãy lập chương trình chuyển đổi đơn vị từ ^0C sang ^0K.

Bài 5. Nhập vào từ bàn phím một số có ba chữ số. Hãy tính và in ra màn hình tổng các chữ số của số vừa nhập.

Bài 6. Hãy viết chương trình nhập độ dài ba cạnh của một tam giác, tính và in ra màn hình diện tích của tam giác đó.

Chương 4

CẤU TRÚC ĐIỀU KHIỂN TRONG C

Chương trình bao gồm nhiều lệnh được viết theo một trật tự xác định. Bình thường các lệnh được thực thi lần lượt theo thứ tự mà chúng được viết ra, nhưng trong thực tế để giải các bài toán thường phát sinh nhu cầu thay đổi trình tự thực hiện các lệnh hoặc thực hiện một lệnh (khối lệnh) nhiều lần. 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.

I – TOÁN TỬ IF

1. Câu lệnh if

Cú pháp :

```
if (biểu thức lôgic) <khối lệnh>
```

Nếu biểu thức lôgic cho kết quả khác 0 thì thực hiện khối lệnh. Nếu khối lệnh có từ hai lệnh trở lên thì phải đặt trong cặp dấu { }.

Ví dụ :

```
/* Chương trình minh hoạ 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. Câu lệnh if... else

Cú pháp:

```
if (biểu_thức_lôgic)      < khối lệnh 1 >;
                  else    < khối lệnh 2 >
```

Nếu biểu thức cho kết quả khác 0 thì thực hiện khối lệnh 1, ngược lại thì thực hiện khối lệnh 2.

Ví dụ 1 :

```
#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 3.000000

Lưu ý :

– Biểu thức lôgic phải đặt trong cặp dấu ngoặc tròn.

– Có thể sử dụng các cấu trúc if lồng nhau, khi đó nên sử dụng khối lệnh để tránh nhầm lẫn, nhất là trong trường hợp số từ khoá else ít hơn số từ khóa if.

Ví dụ 2 :

```
/* 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

II – TOÁN TỬ SWITCH

Cú pháp :

```
switch (Biểu thức)
    {   case n1 :   các câu lệnh
        case n2 :   các câu lệnh
        ...
        case nk :   các câu lệnh
        [default :  các câu lệnh]
    }
```

Trong đó : n_i là các hằng số nguyên hoặc ký tự, nó tuỳ thuộc vào giá trị của biểu thức sau switch, nếu :

– Giá trị này = n_i thì thực hiện câu lệnh sau case n_i;

– Khi giá trị biểu thức khác tất cả các n_i 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 n_i nào đó thì nó sẽ thực hiện luôn các lệnh thuộc case bên dưới nó mà không xét lại điều kiện (do các n_i 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 1 :

```
/* Chương trình minh hoạ 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
```

Ví dụ 2 :

```
/* 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 \n Nhap diem\n");
      scanf("%d",&diem);
      switch(diem)
      {
          case 0 :
          case 1 :
          case 2 :
          case 3 : printf("Kem\n"); break;
          case 4 : printf("Yeu\n"); break;
          case 5 :
          case 6 : printf("Trung binh\n"); break;
          case 7 :
          case 8 : printf("Kha\n"); break;
          case 9 :
          case 10 : printf("Gioi\n"); break;
          default : printf("Vao sai\n");
       }
   }
```

Kết quả :

Nhap diem : 9

Gioi

III – TOÁN TỬ FOR

Cú pháp :

```
for (biểu_thức_1; biểu_thức_2; biểu_thức_3) <khối lệnh>
```

Toán tử for gồm ba biểu thức và phần lệnh của toán tử này. Bất kỳ biểu thức nào trong ba 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 khác 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 1 :

```
/* Chương trình minh hoạ 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.");
}
```

Kết quả :

Giá trị của biến đếm time = 1
Giá trị của biến đếm time = 2
Giá trị của biến đếm time = 3
Giá trị của biến đếm time = 4
Giá trị của biến đếm time = 5
Kết thúc vòng lặp.

Ví dụ 2 :

```
/* 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 500đ, 200đ, 100đ. */
```

```
#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(" %2d x 100d",to_100);
      if (to_200) printf(" %2d x 200d",to_200);
      if (to_500) printf(" %2d x 500d",to_500);
   }
   printf("\n Co tat ca %d cach", so_cach);
}
```

Kết quả :

Tien 1000 d = 10 x 100 d

Tien 1000 d = 8 x 100 d 1 x 200 d

Tien 1000 d = 6 x 100 d 2 x 200 d

Tien 1000 d = 4 x 100 d 3 x 200 d

Tien 1000 d = 2 x 100 d 4 x 200 d

Tien 1000 d = 5 x 200 d

Tien 1000 d = 5 x 100 d 1 x 500 d

Tien 1000 d = 3 x 100 d 1 x 200 d 1 x 500 d

Tien 1000 d = 1 x 100 d 2 x 200 d 1 x 500 d

Tien 1000 d = 2 x 500 d

Co tat ca 10 cach

Sử dụng lệnh break và continue trong vòng lặp for:

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 hoạ 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("\n dem = %d",dem);
          break;
        /* Gặp break máy sẽ thoát khỏi vòng lặp for */
          printf("\n Thu nghiem 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 (bước 4 trong điểm "Sự hoạt động của for"). Cần lưu ý, lệnh continue chỉ áp dụng cho các chu trình chứ không áp dụng cho switch.

Ví dụ :

```
/* Chương trình minh hoạ 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("\n Dem = %d",dem);
            continue;
            /* Gặp continue máy sẽ quay lại bước 4 */
            printf("\n Thu nghiem lenh continue");
 /* lệnh này không được thực hiện vì lệnh continue */
        }
}
```

Kết quả in ra là :

Dem = 1

Dem = 2

Dem = 3

Dem = 4

Dem = 5

IV – TOÁN TỬ WHILE

Cú pháp :

```
while  (biểu_thức) < khối_lệnh >
```

Hoạt động :

– Bước 1 : Xác định giá trị của biểu thức.

– Bước 2 :

+ Nếu biểu thức sai (có giá trị 0) máy sẽ thoát khỏi chu trình và thực hiện câu lệnh sau khối lệnh của while.

+ Nếu biểu thức đúng (có giá trị khác 0), máy sẽ thực hiện các lệnh bên trong <khối lệnh> đến khi gặp }, cuối cùng máy sẽ trở lại từ bước 1.

Biểu thức 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 hoạ 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.
```

Một số lưu ý :

– Thân của while có thể được thực hiện nhiều lần, một lần hoặc thậm chí không được thực hiện lần nào nếu ngay từ đầu biểu thức điều kiện sai.

– Bên trong thân của while ta có thể dùng các toán tử lặp for hoặc while khác.

V – TOÁN TỬ DO... WHILE

Cú pháp :

```
do
    <khối lệnh>;
while (biểu thức điều kiện)
```

Hoạt động :

– Bước 1 : Thực hiện các lệnh trong <khối lệnh>.

– Bước 2 : Tính giá trị của biểu thức điều kiện. Nếu biểu thức điều kiện sai (giá trị 0) thì máy sẽ thoát khỏi vòng lặp do... while. Ngược lại,

nếu biểu thức điều kiện đúng (giá trị khác 0) máy sẽ quay trở lại bước 1 để thực hiện một chu trình mới.

Ví dụ :

```
/* Chương trình minh hoạ 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.
```

Chú ý khi gặp break và continue trong while hay do... while:

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 khoá while, 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 1 : Chương trình minh hoạ vòng lặp 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("\n Dem = %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 hoạ 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 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.

VI – TOÁN TỬ GOTO

Cú pháp :

```
goto nhãn
```

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 */
```

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.

VII – BÀI TẬP MINH HOẠ

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 a, b, c 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("Ngay %u thang %u la ngay thu %u trong
                              nam", ngay, thang, ngay_thu);
}
```

Kết quả :

Nhap vao ngay, thang : 15 2

Ngay 15 thang 2 la ngay thu 46 trong nam

VIII – BÀI TẬP TỰ GIẢI

Bài 1. Cho biết kết quả của 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 :

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n} \quad (n \in \mathbb{N}^*)$$

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

Bài 5. Giả sử phương trình $a^5 + b^5 + c^5 + d^5 + e^5 = f^5$ có nghiệm nguyên duy nhất thoả mãn $0 < a \le b \le c \le d \le e \le f \le 75$. 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 hoặc 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 toạ 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 xuất ra hình cây thông trên màn hình.

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

Bài 9. Viết chương trình in ra bảng cửu chương trên màn hình.

Chương 5

HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

I – HÀM VÀ CHƯƠNG TRÌNH

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.

2. Khái niệm về hàm

Hàm 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.

Hàm là một đơn vị độc lập của chương trình, không cho phép xây dựng một hàm bên trong một hàm khác.

3. Tổ chức một chương trình trong C

Một chương trình trong C được tổ chức theo mẫu :

```
...
hàm 1
...
hàm 2
...
hàm n
```

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

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

– Sử dụng đối của hàm.

– Sử dụng biến ngoài, biến tĩnh ngoài.

II – VÍ DỤ VỀ CHƯƠNG TRÌNH CÓ HÀM

Chương trình tính lũy thừa 3 của x :

```
#include<stdio.h>
long luy_thua(int a);  /* Nguyên mẫu của hàm */
main()                 /* Hàm main */
{
   clrscr();
   int x;
   printf("\n Nhap so muon tinh luy thua : ");
   scanf("%d",&x);
   printf("\n Ket qua : %ld",luy_thua(x));
   getch();
}
long luy_thua(int a)
{
   /* Thân hàm (nội dung hàm) */
   long ketqua;
   ketqua = a*a*a;
   return ketqua;  /* Giá trị trả về của hàm */
}
```

Kết quả khi chạy chương trình :

```
Nhap so muon tinh luy thua : 3
Ket qua : 9
```

III – CÁCH VIẾT MỘT HÀM

1. Xác định mục đích hàm

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

2. Xác định các thành phần của hàm

Các thành phần của hàm bao gồm :

• **Nguyên mẫu của hàm**

```
<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 và giá trị trả về có phù hợp hay không rồi mới cho thực hiện hàm.

Nguyên mẫu của các hàm có trong chương trình nên đặt trước hàm main() và nhóm vào một vùng 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, trong thân hàm phải trả về đúng kiểu giá trị đã định ban đầu. Nếu các hàm không trả về giá trị phải khai báo kiểu void.

• **Tên hàm**

Tên hàm đặt theo quy định đối với tên, 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 có :

+ Tham số hình thức là các tham số mà ta ghi trong nguyên mẫu hay ghi lúc khai báo hàm. Trong ví dụ ở mục II, thì a là tham số hình thức.

+ Tham số thực là các giá trị, biến mà ta ghi sau tên hàm khi gọi hàm. Trong C các tham số thực sự lại chia ra làm hai loại : tham chiếu và tham trị.

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 thay đổi giá trị 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, có các loại tham số : tham số vào; tham số ra; tham số vừa vào, vừa ra. Trong đó tham số vào cung cấp giá trị cho hàm; tham số ra lưu kết quả tính toán được trong hàm; còn tham số vừa vào, vừa ra vừa cung cấp giá trị cho hàm, vừa lưu kết quả tính toán được trong hàm.

Lưu ý :

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

– 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 phải là tham trị (không được là con trỏ).

• Nội dung của hàm

Phần nội dung hàm còn được gọi là *thân hàm*. Với thân hàm 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 : 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 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().

IV – BÀI TẬP MINH HOẠ

Bài 1. 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 = 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 : 2 1

Bài 2. 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

V – BÀI TẬP TỰ GIẢI

Bài 1. Cho đoạn chương trình sau. Hãy 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);
}
```

Bài 2. Giải phương trình bậc hai, bằng cách :

– 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 3. 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 toạ độ, chiều dài và chiều rộng.)

Bài 4. Viết chương trình in ra bàn cờ ca rô trên màn hình.

(*Gợi ý :* Sử dụng hàm sau :

```
void in_1_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);
}).
```

Chương 6

DỮ LIỆU KIỂU MẢNG

I – KHÁI NIỆM

Mảng 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.

II – CÁCH KHAI BÁO

1. Khai báo mảng

Cú pháp :

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

2. 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.

III – CHỈ SỐ CỦA MẢNG

1. Kiểu dữ liệu của chỉ số

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

2. Ví dụ

Ví dụ :

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

IV – LẤY ĐỊA CHỈ CỦA PHẦN TỬ MẢNG

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)
```

Cần lưu ý : Tên của mảng chứa địa chỉ đầu của mảng. Ví dụ, có int a[10] thì a = &a[0].

V – NHẬP, XUẤT DỮ LIỆU CHO CÁC PHẦN TỬ MẢNG

1. Nhập, xuất trực tiếp

Nhập, xuất trực tiếp chỉ sử dụng được với mảng một chiều và mảng hai chiều có các phần tử kiểu int.

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("\n a[%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();
    }
```

2. Nhập, xuất gián tiếp

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

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("\n a[%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();
}
```

VI – MỘT SỐ VẤN ĐỀ LIÊN QUAN

1. Mảng nội

– Khái niệm : *Mảng nội* là các 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 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 1 :

```
/* Minh hoạ 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("Noi dung cua a[0] => %d\n",a[0]);
   printf("Noi dung cua a[1] => %d\n",a[1]);
   printf("Noi dung cua a[2] => %d\n",a[2]);
}
```

Kết quả khi thực hiện :

```
Noi dung cua a[0] => 10
Noi dung cua a[1] => 20
Noi dung cua a[2] => 30
```

Ví dụ 2 :

```
/* Chương trình minh hoạ 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.

2. Mảng ngoại

– Khái niệm : *Mảng ngoại* là các 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 mảng ngoại là từ vị trí nó được khai báo cho đến cuối chương trình.

Các quy tắc khởi đầu khi khai báo mảng ngoại:

a) Các 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 0.

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()
{
    ...
}
```

b) 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}
               };
```

c) 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}
               };
```

d) 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}
                  };
```

e) Bộ khởi đầu của một mảng char có thể là danh sách các hằng ký tự hoặc là một hằng xâu ký 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 hoạ :

```
/*Chương trình minh hoạ quy tắc khởi đầu 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("\n a = %6d,t(1,2) = %6d,t(1,1) = %6d",
                                    a,t[1][2],t[1][1]);
   printf("\n y(0) = %8.2f, y(1) = %8.2f",y[0],y[1]);
   printf("\n x(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 khi 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

VII – BÀI TẬP MINH HOẠ

1. Bài tập về mảng một chiều

Bài 1.

```
/* Nhập mảng một chiều, tính tổng các phần tử */
#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 vào 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;
}1
```

Bài 3.

```
/* Viết chương trình nhập vào toạ độ hai vectơ 5
chiều, tính môđun 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 4.

```
/* 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("\n a[%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 5.

```
  /* 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("\n a[%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 6.

```
/* 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("\n a[%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 7.

```
/* 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("\n a[%d] = ",i); scanf("%d",&a[i]);
   }
   ort(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;
    }
}
```

2. Bài tập về mảng hai chiều

Bài 8.

```
/* 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 9.

```
/* 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 Nhap cac phan tu cua ma tran\n");
   for(i = 0;i < N;i ++)
       for(j = 0;j < N;j ++)
       {
           printf("\n Phan tu 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 dotheo
                                  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 không âm 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);
    }
     ...
}
```

VIII – BÀI TẬP TỰ GIẢI

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 × n. Hãy biến đổi dấu của tất cả các số của một 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 × n tăng dần theo cột và theo hàng (a[0, 0] < a[0, 1] < a[0, 2]... < a[0, n] < a[1, 0] <... < a[1, n] < a[2, 0] <... < 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 một chiều (sử dụng phép / và %).

Bài 6. Viết chương trình xoay một ma trận 5 × 5 một góc 90 độ theo chiều kim đồng hồ (không dùng ma trận phụ b).

Bài 7. Viết chương trình xoay một ma trận 5 × 5 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 n × n :

– 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 5×5 tăng dần theo hình ziczắc ngang (a[0, 0] < a[0, 1] <... < a[0, n] < a[1, n] < a[1, n – 1] < a[1, n – 2] <... < a[1, 0] < a[2, 0] < a[2, 1] < a[2, 2] < a[2, 3] <... < a[2, n] < a[3, n] < a[3, n – 1] < a[3, n – 2] <... < a[3, 0] < a[4, 0] <... < a[4, 4].

Gợi ý : Xem cách 1 bài 5.

Bài 10. Tương tự như bài trên, nhưng là ziczắc đứng.

Bài 11. Sắp xếp một mảng hai chiều 5 × 5 tăng dần theo hình xoắn ốc.

Bài 12. Viết chương trình thực hiện phép cộng và nhân hai ma trận kích thước 5 × 5.

Chương 7

CHUỖI KÝ TỰ

I – KHÁI NIỆM

1. Một số khái niệm cơ bản

Trong C chuỗi ký tự là một dãy các ký tự đặt trong cặp dấu nháy kép " ". Chuỗi rỗng được ký hiệu là "" (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 null ('\0') là ký tự kết thúc chuỗi.

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 null có mã ASCII là 0 (ký hiệu '\0'). Kích thước của chuỗi ký tự phải đủ để chứa được cả 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 byte cho ký tự 'A' và ký tự '\0'.

3. Các ví dụ minh hoạ

```
/* Minh hoạ về chuỗi ký tự */
#include<stdio.h>
char chuoi[] = "HELLO"; /* Khai báo chuỗi */
```

```
void main()
{  printf("Xau ky tu la %s. \n",chuoi);
   /* In ra chuỗi ký tự */
   printf("Cac ky 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 ky tu la HELLO

Cac ky tu se la :

H

E

L

L

O

II – CÁCH THAO TÁC TRÊN CHUỖI KÝ TỰ

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> :

• **Hàm strlen**

Cú pháp :

```
int strlen(char s[])
```

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

• **Hàm strcpy**

Cú pháp :

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

Công dụng : Sao chép nội dung chuỗi source vào chuỗi dest.

• **Hàm strncpy**

Cú pháp :

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

Công dụng : Tương tự như hàm 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**

Cú pháp :

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

Công dụng : 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**

Cú pháp :

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

Công dụng : Tương tự như hàm strcat, nhưng chỉ giới hạn với n ký tự đầu tiên của ch2.

• **Hàm strcmp**

Cú pháp :

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

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

– là 0 nếu chuỗi ch1 bằng chuỗi ch2;

– lớn hơn 0 nếu chuỗi ch1 lớn hơn chuỗi ch2;

– nhỏ hơn 0 nếu chuỗi ch1 nhỏ hơn chuỗi ch2.

• **Hàm strncmp**

Cú pháp :

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

Công dụng : Tương tự như hàm strcmp, nhưng chỉ giới hạn việc so sánh với n ký tự đầu tiên của hai chuỗi.

• **Hàm stricmp**

Cú pháp :

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

Công dụng : Tương tự như hàm strcmp, nhưng không phân biệt chữ in và chữ thường.

• **Hàm strincmp**

Cú pháp :

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

Công dụng : Tương tự như hàm stricmp, nhưng việc so sánh chỉ giới hạn ở n ký tự đầu tiên của mỗi chuỗi.

• **Hàm strchr**

Cú pháp :

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

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

• **Hàm strrchar**

Cú pháp :

```
char *strrchar(char s[],char c)
```

Công dụng : Tương tự như hàm strchr, nhưng việc tìm kiếm bắt đầu từ cuối chuỗi.

• **Hàm strlwr**

Cú pháp :

```
strlwr(char s[])
```

Công dụng : Chuyển đổi các chữ in trong chuỗi s sang chữ thường.

• **Hàm struppr**

Cú pháp :

```
struppr(char s[])
```

Công dụng : Ngược lại với hàm strlwr.

• **Hàm strset**

Cú pháp :

```
strset(char s[], char c)
```

Công dụng : Khởi đầu tất cả các ký tự của s bằng ký tự c.

• **Hàm strnset**

Cú pháp :

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

Công dụng : Khởi đầu giá trị cho n ký tự đầu tiên của s bằng ký tự c.

• **Hàm strstr**

Cú pháp :

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

Công dụng : 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. Các hàm nhập, xuất chuỗi thuộc stdio.h

Ngoài các hàm nhập, xuất đã giới thiệu ở chương 3, trong C còn có hai hàm vào, ra dùng riêng cho các chuỗi ký tự thuộc stdio.h.

• **Hàm gets**

Hàm gets(s) đọc từ bàm 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(s) 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 '.

Ví dụ minh hoạ : 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 Nhap vao dong moi :");
    gets(chuoi);
    printf ("\n Ban da nhap vao xau :");
    puts(chuoi);
     getch();
  }
```

Kết quả :

Nhap vao dong moi : Hom nay troi dep qua, toi ngoi nho den em

Ban da nhap vao xau : Hom nay troi dep qua, toi ngoi nho den em

III – BÀI TẬP MINH HOẠ

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("\n Nhap mot dong chu : ");
   gets(dong);
   na = i = 0;
   while(dong[i])
       if (dong[i ++ ] == 'a') na ++;
```

```
            printf("\n Dong co %d chu a",na);
        getch();
        return;
    }
```

IV – BÀI TẬP TỰ GIẢI

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 (sử dụng chuỗi).

Chương 8

CON TRỎ VÀ ĐỊA CHỈ

I – TOÁN TỬ ĐỊA CHỈ

1. Các khái niệm liên quan đến biến

Liên quan đến biến có ba khái niệm :

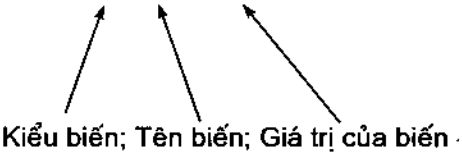
– Tên biến;

– Kiểu biến;

– Giá trị của biến.

Ví dụ :

```
int a = 15;
```



2. Địa chỉ của biến

2.1. Khái niệm

Trong khai báo ở ví dụ trên :

```
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.

2.2. Phân loại địa chỉ biến

Từ khái niệm về địa chỉ, ta nhận 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,...

2.3. Phép lấy địa chỉ của một biến

Toán tử một ngôi & cho ta địa chỉ của một đối tượng. Như vậy, câu lệnh

```
p = &c; /* c là biến ký tự */
```

gán địa chỉ của biến ký tự c cho con trỏ p và chúng ta nói rằng p "trỏ 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).

II – CON TRỎ

1. Khái niệm biến con trỏ

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. Phân loại con trỏ

Tuỳ thuộc vào kiểu biến mà con trỏ trỏ đến ta có con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu int, con trỏ kiểu float chứa địa chỉ của biến kiểu float,...

3. Khai báo biến con trỏ

3.1. Con trỏ không kiểu

Con trỏ không kiểu có thể chứa bất kỳ một địa chỉ nào. Cách khai báo như sau :

```
void *tên biến con trỏ
```

Ví dụ :

```
void *p,*q;
```

3.2. Con trỏ có kiểu

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. Cách khai báo như sau :

```
< kiểu dữ liệu > *tên biến con trỏ
```

Ví dụ :
```
int     *p,*q;
float   *x;
```

4. Hằng con trỏ

Đối với các biến, từ khóa const 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. Đối với con trỏ, const được 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, khong duoc thay doi bien 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 tro boi
                con tro stt khong duoc thay doi */
}
```

III – QUY TẮC SỬ DỤNG CON TRỎ TRONG BIỂU THỨC

1. Tên con trỏ

Tên con trỏ được đặt tuân theo quy tắc đặt tên đã trình bày ở chương 1. Trong biểu thức 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 */
```

2. Ví dụ về dạng khai báo của con trỏ

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

```
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.

3. Các phép toán trên con trỏ

Có bốn nhóm phép toán liên quan đến con trỏ và địa chỉ là phép gán; phép tăng, giảm địa chỉ; phép truy nhập bộ nhớ và phép so sánh.

3.1. 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 */
```

3.2. 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].

3.3. 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 1 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.

3.4. Phép so sánh

C cho phép so sánh các con trỏ cùng kiểu. Ví dụ, nếu p1 và p2 là hai 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.

4. Toán tử *

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 hoạ 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 */
```

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;` sẽ lấy giá trị của đối tượng được trỏ bởi pi, cộng thêm 10 và ghi kết quả trong y, còn lệnh `*pi + = 1;` sẽ tăng giá trị của đối tượng được trỏ bởi pi thêm 1. Phép toán này hoàn toàn tương đương với `++ *pi; (*pi) ++;`

Lưu ý : Các dấu ngoặc `(*pi) ++;` 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.

IV – QUY TẮC VỀ KIỂU GIÁ TRỊ TRONG KHAI BÁO

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 [20] [30].

Vì vậy phép gán `pa = a;` là sai, phép gán `pm = a;` hoàn toàn hợp lệ.

V – CON TRỎ VỚI MẢNG

1. Con trỏ với mảng một chiều

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

Ví dụ : Với khai báo float a[10] thì a = &a[0] và a + i = &(a[i]), 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 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.

Sau đây là một số ví dụ minh hoạ 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 hoạ 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("\n a[%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

2. Con trỏ với mảng nhiều chiều

Việc xử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều, không phải mọi quy tắc đối với mảng một chiều đều có thể áp dụng được 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("Nhap vao kich thuoc ma tran 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 (float *)a + i*n + j.

Chương trình được viết lại cho đúng như sau :

```
#include<stdio.h>
void main()
{
   float a[10][20];
   int i,j,n;
   printf("Nhap vao kich thuoc ma tran n = ");
   scanf("%n",&n);
   for(i = 0; i < n; i ++)
```

```
        for(j = 0;j < n;j ++)
            {
                printf("a[%d][%d] = ",i,j);
                scanf("%f",(float *)a + i*20 + j);
            }
    }
```

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;

...

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 : (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 (float *a) + i*n + j.

3. Mảng 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.

Một mảng các con trỏ được khai báo bằng câu lệnh sau :

```
type *pointer_array[size];
```

Đây là khai báo một mảng có tên là pointer_array gồm size biến con trỏ kiểu type, mỗi con trỏ được hiểu là trỏ 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 trỏ đến đâu cả).

Ví dụ câu lệnh `char *ma[10];` được dùng để 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.

Nếu các con trỏ được chuẩn bị để trỏ đến một biến nào đó đã có, 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 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 : 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. Khi đó dù 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 trỏ 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];
       /* Khai báo một mảng 6 con trỏ nguyên */
       /* 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 sếp */
    for(i = 0;i < 6;i ++)
        printf(" %d \n",*ptr_array[i]);
    getch();
}
```

Kết quả khi chạy chương trình:

2
3
6
7
10
12

Nếu các phần tử của các con trỏ 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 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 của các mảng này so với nhau thực chất chỉ là việc sắp xếp lại các địa chỉ của chúng trong mảng các con trỏ mà thôi. Xét ví dụ sau:

Ví dụ : Viết một chương trình nhập tên người vào từ bàn phím, sau đó sắp xếp lại theo thứ tự và in ra kết quả đã sắp xếp. 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 hoạ 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]);
        /* Giai phong vung bo nho da cap phat cho
                                        strlist[i] */
    }
}
```

Như vậy, qua ví dụ trên ta thấy việc sử dụng một mảng các con trỏ 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];
```

điều khiển cho C hiểu slist[i] và plist[i] là các con trỏ trỏ 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 trỏ đến một mảng n phần tử, thì việc xin cấp chỗ cho các mảng và gán địa chỉ của chúng cho các con trỏ là công việc của chúng ta. Như vậy, 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.

– Các mảng con của nó được trỏ đến bởi các con trỏ, có thể có độ dài tuỳ 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 trỏ 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 mất thời gian khi kích thước của các mảng lớn.

Thật ra khi sử dụng mảng các con trỏ, như đã trình bày 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 trỏ 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 trỏ đến con trỏ đầu tiên của mảng, con trỏ này lại trỏ đến chuỗi "January" như đã khởi đầu cho mảng. Nếu tăng pp lên 1 thì pp sẽ trỏ đến phần tử kế tiếp của mảng có giá trị là con trỏ đến xâu ký tự "February",...

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("\n Nhap vao mot so nguyen giua 1 va 7 : ");
   scanf("%d",&i);
   }
   while(i <= 0||i > 7);
  printf("\n Ngay so %d cua tuan la %s", i,ngay[i - 1]);
   getch();
   return;
}
```

VI – HÀM VỚI CON TRỎ VÀ MẢNG

1. Hàm với con trỏ

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 có thể gán cho nó một giá trị mới bằng cách sử dụng các câu lệnh thích hợp trong thân hàm.

Ví dụ minh hoạ : 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() /* Ham chinh */
```

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.

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

Kết quả thu được :

Truoc khi goi ham : A = 3 B = 7

Trong ham swap : A = 7 B = 3

Sau khi goi 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 goi ham : ")
   printf("A = %d ",a);
   printf("B = %d ",b);
   swap(&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)

```
    printf("Sau khi goi ham : A = %d B = %d \n",a,b);
    getch();
}
void swap(int *a, int *b)
{
  int temp;
  temp = *a;  *a = *b;  *b = temp;
  printf("\n Trong ham swap A = %d B = %d \n",*a,*b);
}
```

Kết quả thu được :

Truoc khi goi ham : A = 3 B = 7

Trong ham swap : A = 7 B = 3

Sau khi goi ham : A = 7 B = 3

Qua hai ví dụ trên, ta thấy 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 phải khai báo đối số của hàm là tham chiếu (con trỏ hay dạng địa chỉ).

2. Hàm với 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) hoặc pa[i].

Ví dụ :

```
/* Minh hoạ 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); /* Gọi ham3 */
        printf ("\n Ket qua thuc hien ham 4 : \n");
   ham4(a); /* Gọi 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.

3. Hàm với mảng hai chiều

Giả sử tham số thực là mảng hai chiều a đã được khai báo

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

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 số cột của mảng hai chiều bị hạn chế.

** 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 quy về như mảng một chiều. Việc xử lý phức tạp hơn so với cách 1, nhưng không hạn chế số cột nên có thể dùng cho bất kỳ mảng hai chiều nào.

Ví dụ :

```
/* 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);/* Goi ham */
    printf("Tong cua cot dau tien la %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

VII – CON TRỎ VỚI CHUỖI KÝ TỰ

1. Cách khai báo

Trong một chương trình nếu 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.

2. Phân tích các ví dụ

Phân tích trường hợp : `char *p;` thì lệnh `p = "Turbo C";` hoàn toàn có nghĩa, khi đó `*p = 'T' * (p + 1) = 'u'`. Nếu ta dùng lệnh `printf("%s", p);` thì chuỗi Turbo được in lên màn hình.

Các ví dụ minh họa

Ví dụ 1 :

```
/* Minh hoạ 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 hoạ 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 | l |
Address => 5324 | l |
Address => 5325 | o |
Address => 5326 | |

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

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

Dòng 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. Hàm với chuỗi ký tự

Nếu tham số thực là tên một chuỗi ký tự, đối ten_chuoi của hàm được phép khai báo theo một trong hai mẫu : char ten_chuoi[] hoặc char *ten_chuoi.

Ví dụ :

```
/*Hàm và xâu ký tự: Minh hoạ 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!

VIII – BÀI TẬP MINH HOẠ

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>
#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);
}
```

Kết quả in ra :

Truoc : A = 7 B = 3

Sau : A = 3 B = 7

Bài 3. 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);
}
```

Kết quả in ra : 4 3

Bài 4. 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);
}
```

Kết quả in ra : 1 3

Bài 5. 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 6. 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);
}
```

Bài 7. 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();
}
void doi(int *a)
{
    int tam = 1;
    *a + = tam;
    tam ++;
}
```

Kết quả in ra : 1 2

Bài 8. 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 9. 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 10. 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 11. 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 hoạ 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("\n Tong la : %f",sohoc(x,y,' + '));
      printf("\n Hieu la : %f",sohoc(x,y,' - '));
      printf("\n Tich la : %f",sohoc(x,y,'*'));
      printf("\n Thuong 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

Bài 12.

```
/* 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();
}
```

IX – BÀI TẬP TỰ GIẢI

Bài 1. Cho đoạn chương trình sau, xác định kết quả in ra.

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

Bài 2. Cho các 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 3. Viết chương trình cho dòng chữ "Trường đại học Công nghiệp Hà Nội" chạy từ trái sang phải màn hình.

Bài 4. Cho đoạn chương trình sau:

```
#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 ++;
}
```

Chọn kết quả đúng khi thực hiện đoạn chương trình trên:

a) –1;

b) Chương trình sai cú pháp;

c) 0;

d) Chương trình lặp vô tận.

Chương 9

CẤP PHÁT VÀ GIẢI PHÓNG BỘ NHỚ ĐỘNG

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.

I – BIẾN ĐỘNG

1. Khái niệm

Thế nào là một biến động ?

Biến động là các biến được tạo ra lúc chạy chương trình, tuỳ 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 hoạ

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

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

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

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

Hoặc muốn cấp phát bộ nhớ chính xác cho 75 ký tự, ta viết

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

Trong đó sizeof(type) là hàm để lấy kích thước của kiểu dữ liệu type.

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 sau.

2.1. 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.

Ví dụ 1 :

```
# include <stdlib.h>
# include<conio.h>
void main() /* Ham chinh */
```

```
{
   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));
```

Dòng 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 : Trong khai báo như trên, sizeof(int) cho ta kích cỡ tính theo byte của kiểu int (2 byte).

2.2. Cấp phát bộ nhớ động bằng hàm calloc

Cú pháp :

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

Chức năng : 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 tạo ra.

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

Cấp phát bộ nhớ cho 10 người :

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

2.3. Cấp phát bộ nhớ động bằng hàm relloc

Cú pháp :

```
(datatype *) realloc(buf _p, newsize);
```

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.

2.4. Giải phóng bộ nhớ động bằng hàm free

Cú pháp :

```
void free(void *prt)
```

Chức năng : Hàm free giải phóng vùng nhớ được trỏ đến bởi con trỏ ptr. Nếu ptr = NULL thì free không làm gì cả.

Ví dụ : Viết chương trình nhập các số nguyên từ bàn phím. Xuất các số đó 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 Ky tư thu %d",j);
        scanf(" %d",(num + j));
    }
    for (j = 0;j < i;j ++)
        printf("%c \n",*(num + j)); getch();
        /* Giải phóng bộ nhớ */
    free(num);
}
```

2.5. Các ví dụ

Ví dụ 1 : 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("Khong du bo nho de cap phat \n");
       exit(1);
      /* Kết thúc chương trình, nếu tràn bộ nhớ */
   }
       /* Sao chép "Hello" vào chuỗi */
   strcpy(str, "Hello");
       /* Hiển thị chuỗi */
   printf("Chuoi ky tu la : %s\n", str);
       /* Giải phóng bộ nhớ */
```

```
    free(str);
    return 0;
}
```

Ví dụ 2 : 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("Chuoi ky tu la %s\n", str);
  /* Giải phóng bộ nhớ */
  free(str);
  return 0;
}
```

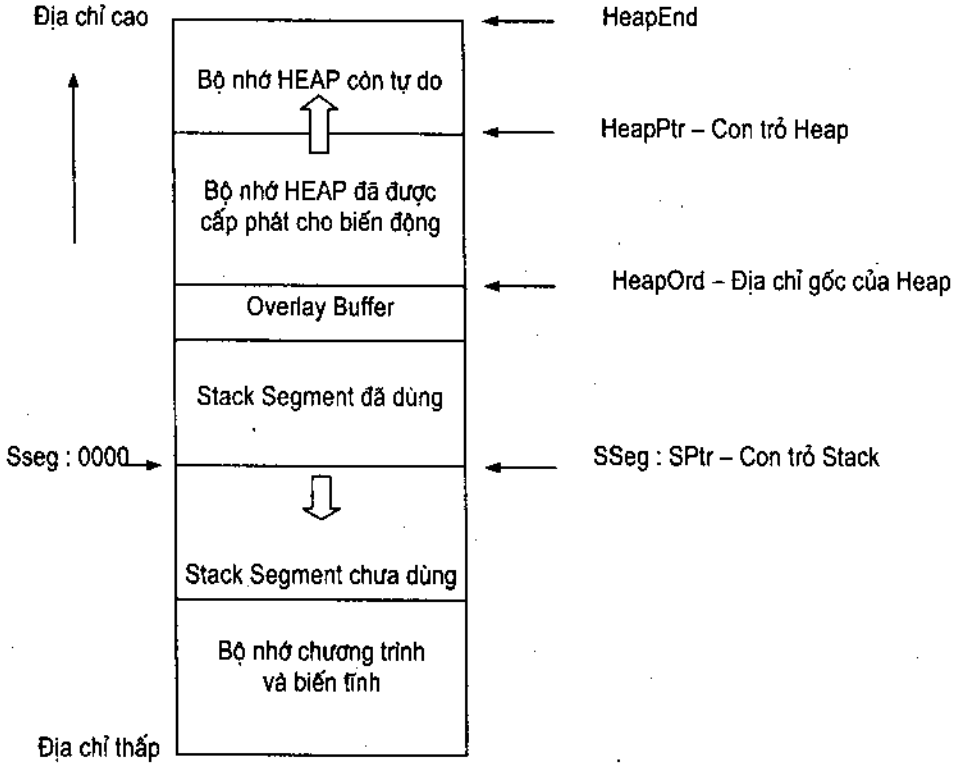
Ví dụ 3 : Dùng hàm realloc cấp phát lại bộ nhớ cho một chuỗi ký tự.

```
#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("Chuoi ky tu la %s\n Dia chi la %p\n",
                                        str, str);
```

```
    str = (char *) realloc(str, 20);
    printf("Chuoi ky tu la %s\n Dia chi moi la
                                        %p\n", str, str);
    /* Giải phóng bộ nhớ */
    free(str);
    return 0;
}
```

II – BỘ NHỚ HEAP VÀ CƠ CHẾ TẠO BIẾN ĐỘNG

1. Khái niệm



Khái quát việc sử dụng bộ nhớ và Heap

Các biến động do malloc tạo ra được 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.

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.

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 %121d %121d %121d %121d %121d",
          *(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
```

IV – BÀI TẬP MINH HOẠ

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);
}
```

V – BÀI TẬP TỰ GIẢI

Bài 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 phần cấp phát động).

Bài 2. Nhập vào mảng a và b theo kiểu cấp phát động (không dùng mảng). Với :

1. Các phần tử của a và b không trùng nhau;

2. Xếp theo thứ tự tăng dần hai mảng a, b;

3. Nối hai mảng này thành một mảng duy nhất sao cho mảng vẫn tăng.

Bài 3. Viết chương trình thực hiện công việc sau :

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. Kiểm tra mảng a có phải là mảng đan dấu hay không.

3. Tìm số nguyên tố lớn nhất trong mảng nếu không có thì thông báo.

Bài 4. Viết chương trình tạo ngẫu nhiên hai ma trận vuông a, b (n × n) theo kiểu cấp phát động và thực hiện các công việc sau :

1. In hai ma trận a, b đã được tạo.

2. In ra ma trận tổng.

3. In ra ma trận tích.

Chương 10

HÀM MAIN CÓ THAM SỐ – CON TRỎ HÀM

I – HÀM MAIN CÓ THAM SỐ

1. Quy 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.

2. Các dạng tham số

Có hai dạng tham số :

– *Tham số 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 là một mảng con trỏ ký tự khai báo dạng char *arr[]* : Mảng này sẽ ghi nhận con trỏ trỏ vào vùng nhớ của các chuỗi giá trị mà ta truyền cho chương trình. Phần tử thứ 0 sẽ chứa con trỏ trỏ đến vùng nhớ chứa tên chương trình.

3. Ví dụ minh hoạ

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 %0.2f ",s);
   getch();
}
```

Sau khi nhập nội dung lưu trên đĩa với tên vidu, C sẽ biên dịch ra file vidu.exe rồi thoát ra DOS. Tại dấu nhắc lệnh của DOS ta gõ vidu 1 2 3 Enter thì trên màn hình hiện ra "Kết quả 6". Nếu gõ vidu 1 2 3 4 5 Enter thì trên màn hình xuất hiện "Kết quả 15".

II – CON TRỎ HÀM

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. 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 xét chương trình dưới đây :

```
/* Đoạn chương trình so sánh hai xâu */
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 xâu giống nhau không ?\n");
   if(!(*cmp)(a,b)) printf("Giống nhau");
   else printf("Khác nhau");
}
```

Có hai lý do khai báo strcmp() trong main(). Trước hết, chương trình phải biết kiểu của giá trị mà hàm strcmp trả về; 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 trỏ 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 xâu 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.

Chú ý rằng, cũng có thể gọi hàm 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 trỏ. Tuy nhiên, hầu hết các chương trình con đều sử dụng con trỏ trỏ đến hàm để chương trình linh động hơn. Chẳng hạn, xét chương trình dưới đây :

```
/* Đoạn chương trình so sánh hai xâu */
#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 xâu 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 tuỳ theo ký tự đầu tiên của chuỗi nhập vào là chữ cái hay chữ số, bằng cách gọi hàm với biến trỏ p mà biến trỏ này tuỳ tình huống có thể trỏ đến hàm strcmp (khi ký tự đầu tiên là chữ cái) hay numcmp (khi ký tự đầu tiên là chữ số).

3. Cách khai báo con trỏ hàm và mảng con trỏ hàm

Câu lệnh

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

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).

Câu lệnh

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

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).

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.

Các ví dụ minh hoạ 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));
}
```

5. Đối con trỏ hàm

C cho phép thiết kế các hàm mà tham số thực trong lời gọi tới nó lại là tên của các hàm 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*sin(x*x))/(1 + 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));
}
```

6. Một số điểm cần lưu ý

– 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 :

```
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) (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.

III – BÀI TẬP MINH HOẠ

Bài 1.

```
/* Chương trình in ra tham số dòng lệnh Tên tệp tin
thực hiện chương trình là IN_TSDL */
#include<stdio.h>
void main(int n, char **a)
{
   int i;
   printf("\n Ten chuong trinh : %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 chuong trinh : 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 hoạ cách dùng mảng con trỏ hàm để lập bảng giá trị của các hàm : x*x, sin(x), cox(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) /* tinh 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;
   }
}
```

IV – BÀI TẬP TỰ GIẢI

Viết hàm thực hiện các bài toán dưới đây :

Bài 1. Giải hệ phương trình bậc nhất.

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

Hàm có 6 đối vào là a, b, c, d, e, f và hai đối ra là x, y.

Bài 2. Tính đa thức bậc n.

Hàm có hai đối là biến nguyên n và mảng thực a.

Bài 3. Tìm giá trị lớn nhất và nhỏ nhất của một dãy số cho trước.

Bài 4. Giải phương trình

$$AX = B$$

bằng phương pháp khử Gauss (trong đó A là ma trận vuông cấp n và B là vectơ 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.

Chương 11

DỮ LIỆU KIỂU CẤU TRÚC

I – KIỂU ENUM

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.

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.

II – KIỂU CẤU TRÚC

1. Khái niệm và định nghĩa kiểu cấu trúc

Kiểu 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 kiểu cấu trúc trong C có nhiều nét tương tự như khái niệm về bản ghi (record) trong PASCAL hay trong FOXPRO.

2. Khai báo kiểu cấu trúc

Cú pháp :

```
struct [tên_cấu_trúc]
    {
        khai báo các thành phần
    } [danh sách các biến kiểu cấu trúc];
```

Trong đó :

– struct là từ khoá đứng trước một khai báo kiểu cấu trúc;

– tên_cấu_trúc là một tên hợp lệ được dùng làm tên kiểu cấu trúc;

– khai báo các thành phần là một danh sách các khai báo tên và kiểu dữ liệu của các thành phần tạo nên kiểu cấu trúc này;

– danh sách các biến kiểu cấu trúc liệt kê các biến có kiểu cấu trúc vừa khai báo.

Ví dụ :

```
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 kiểu cấu trúc hoc_sinh.

3. Định nghĩa kiểu bằng typedef

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:

```
typedef unisgned 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 dich phai nam lan
                                        là %X",ch1,ch);
   getch();
}
```

Kết quả chạy chương trình :

Byte C sau khi dich phai nam lan la 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 hoá 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 kiểu 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 kiểu cấu trúc viết gọn lại như sau :

```
t_hoc_sinh hs,dshs[100];
```

Để khai báo một biến con trỏ kiểu cấu trúc ta có thể sử dụng câu lệnh

```
ptr_hoc_sinh ptrhs;
```

Câu lệnh này rõ ràng ngắn ngọn hơn so với câu lệnh

```
struct hoc_sinh *ptrhs;
```

4. Truy nhập đến các thành phần của kiểu cấu trúc

4.1. Nguyên tắc chung

Các thành phần của kiểu cấu trúc được truy nhập thông qua tên biến kiểu 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 ta viết như sau :

```
hs.ho_ten
hs.diem
```

4.2. Ví dụ minh hoạ

Chương trình sau đây nhập vào Họ tên và Điểm thi của các học sinh, 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 lương 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 học sinh 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 hoc sinh 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 học sinh phai thi lai

1. Do thanh Xuan 3.0
2. Pham Hai Yen 4.0

Lưu ý :

– Không nên sử dụng toán tử & đối với các thành phần của kiểu 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 kiểu cấu trúc đơn, ví dụ như hs có thể sử dụng #define để rút gọn "đường truy nhập" đến các thành phần kiểu cấu trúc, ví dụ :

```
#define ht hs.ho_ten
#define ds hs.diem
```

Có thể áp dụng phép gán cho các biến kiểu cấu trúc có cùng kiểu.

Chương trình minh hoạ :

```
#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}

5. Con trỏ kiểu cấu trúc

5.1. Cách khai báo

Một biến kiểu cấu trúc cũng là một biến trong bộ nhớ, do đó ta cũng có thể lấy địa chỉ của một biến kiểu cấu trúc bằng toán tử lấy địa chỉ &. Giá trị trả lại là địa chỉ đến trường đầu của kiểu cấu trúc.

Ta có thể khai báo một biến con trỏ chỉ đến một kiểu cấu trúc để có thể "cất" địa chỉ của một biến kiểu cấu trúc nào đó cần thiết. Cú pháp khai báo một biến con trỏ kiểu cấu trúc như sau :

```
struct tên_cấu_trúc *tên_con_trỏ;
```

Ví dụ :

```
struct hoc_sinh *ptrhs;
```

Khi đó biến con trỏ kiểu cấu trúc ptrhs chỉ mới được cấp chỗ và được ghi nhận là con trỏ trỏ đến kiểu cấu trúc hoc_sinh; còn đối tượng mà con trỏ này trỏ tới vẫn chưa được chuẩn bị gì cả, do đó chưa thể dùng được đối tượng này. Chúng ta phải gán cho nó một địa chỉ của một biến kiểu cấu trúc cùng kiểu nào đó đã được khai báo hoặc cũng có thể xin cấp phát một vùng bộ nhớ thông qua các hàm cấp phát động.

Ví dụ :

```
struct hoc_sinh hs = {"Nguyen Van A",6.5};
  /* khai báo và khởi đầu biến kiểu cấu trúc */
struct hoc_sinh *ptrhs;
ptrhs = &hs
```

Kể từ thời điểm này, biến ptrhs đã trỏ đến biến kiểu cấu trúc hs và ta có thể truy xuất đến các thành phần của biến hs một cách gián tiếp thông qua con trỏ ptrhs.

5.2. Truy xuất đến các thành phần kiểu cấu trúc

Việc truy xuất đến một thành phần của kiểu 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("\n Ho va ten hoc sinh %s",ptrhs -> ho_ten);
printf("\n Diem %6.3f",ptrhs -> diem);
```

Kết quả thực hiện hai câu lệnh trên tương đương với hai câu lệnh sau :

```
printf("\n Ho va ten hoc sinh %s",hs.ho_ten);
printf("\n Diem %6.3f",hs.diem);
```

5.3. Ứng dụng

Việc sử dụng con trỏ trỏ đến kiểu cấu trúc thường được sử dụng để truyền kiểu cấu trúc đến cho một hàm. Chúng ta có thể gửi trực tiếp một

kiểu cấu trúc làm đối số cho một hàm, nhưng nếu kiểu cấu trúc đó lớn thì việc chép lại toàn bộ kiểu cấu trúc đó để gửi đi sẽ làm mất nhiều thời gian. Có trường hợp ta còn muốn thay đổi nội dung của kiểu cấu trúc đó từ một hàm khác. Trong những trường hợp đó ta nên gửi địa chỉ của kiểu cấu trúc đó cho hàm.

Một ứng dụng khác của con trỏ kiểu cấu trúc mà chúng ta sẽ đề cập đến trong các phần sau là việc xây dựng các kiểu cấu trúc tự trỏ như danh sách liên kết (còn gọi là danh sách móc nối).

6. Thành phần kiểu FIELD (nhóm bit)

6.1. Cách khai báo

Ngôn ngữ C cho phép chúng ta khai thác đến từng bit như là một thành phần riêng biệt của một kiểu cấu trúc. Một thành phần như vậy được gọi là *một field* (tạm dịch là một vùng). Khai báo cho một kiểu cấu trúc có các thành phần là các field được thực hiện tương tự như khai báo một kiểu cấu trúc bình thường :

```
struct  struct_field
        {
            unsigned field1 : số_bit1;
            int field2 : số_bit2;
            ...
        } var_field;
```

Trong khai báo kiểu cấu trúc field trên, tên các thành phần là field1, field2,... 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 cách với tên trường bằng dấu hai chấm. Độ dài này không được vượt quá 16 bit.

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 kiểu 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;
```

6.2. Ứng dụng

Các kiểu 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).

7. Mảng cấu trúc

7.1. Khai báo

Mảng mà các thành phần có kiểu cấu trúc được gọi là mảng cấu trúc. Khai báo một mảng 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 kiểu cấu trúc.

7.2. Ví dụ về khai báo một mảng cấu trúc

```
struct hoc_sinh dshs[100];
/* hoc_sinh là kiểu cấu trúc */
```

Việc sử dụng các mảng cấu trúc sẽ làm cho việc xử lý một tập hợp các biến kiểu cấu trúc trở nên dễ dàng hơn. Các quy định về mảng cũng được áp dụng đối với mảng cấu trúc.

III – CÁC CẤU TRÚC TỰ TRỎ

1. Khái niệm

Khi lập một chương trình quản lý mà bản thân số biến (cấu trúc) chưa được biết trước bằng cách sử dụng mảng (cấp phát bộ nhớ tĩnh) thì ta phải khai báo một mảng có kích thước bằng số tối đa các phần tử. Như vậy sẽ có rất nhiều vùng nhớ được cấp phát mà không bao giờ dùng đến. Để giải quyết vấn đề này ta dùng cách cấp phát bộ nhớ động, khi đó số vùng nhớ cấp ra là vừa đủ.

Kiểu cấu trúc có ít nhất một thành phần là con trỏ trỏ đến bản thân cấu trúc được gọi là *cấu trúc tự trỏ*.

2. Các ví dụ về khai báo dữ liệu

Dưới đây đưa ra một số ví dụ về cấu trúc tự trỏ.

Ví dụ 1 :

```
struct h_sinh{
                char ho_ten[20];
                float diem;
                struct h_sinh *sau;
              /* con trỏ trỏ đến học sinh tiếp theo
                                    trong danh sách */
            };
```

Khai báo này định nghĩa một kiểu cấu trúc tự trỏ có thể dùng để quản lý danh sách họ tên học sinh cùng điểm số của họ (ở trên chúng ta đã dùng một mảng để quản lý danh sách này). Như sau này chúng ta thấy, cách quản lý mới bằng cấu trúc tự trỏ 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, ta sử dụng khai báo sau :

Ví dụ 2 :

```
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 truoc 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, ta có thể dễ dàng duyệt danh sách theo cả hai chiều "tiến" hoặc "lùi".

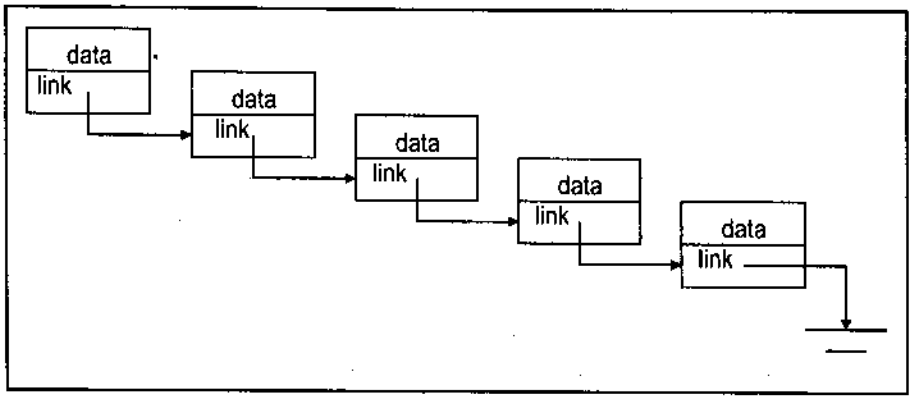
Kiểu cấu trúc tự trỏ là một cơ chế linh hoạt để xây dựng danh sách liên kết (móc nối).

3. Danh sách liên kết

3.1. Định nghĩa

Khi quản lý một danh sách nào đó thì thường xuyên phải cập nhật, thêm, bớt các phần tử của danh sách. Nếu sử dụng mảng sẽ rất tốn kém (khi số thành phần mảng vượt quá số phần tử 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 tuỳ ý (tuỳ theo khả năng của bộ nhớ) và hơn thế nữa do sử dụng cấp phát động 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 trỏ đến các phần tử trước hoặc sau phần tử đang được xét tuỳ thuộc vào yêu cầu của công việc cụ thể. Hình sau minh hoạ 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 khai báo các trường thông thường, phần thứ hai là các con trỏ trỏ đến chính các kiểu dữ liệu đó. Sau đây chúng ta phân tích một chương trình để minh hoạ cách dùng danh sách móc nối quản lý một lớp học gồm Họ tên học sinh và Điểm thi.

3.2. Bài tập mẫu

Xây dựng chương trình quản lý Họ tên học sinh và Điểm thi học kỳ của từng học sinh. Danh sách học sinh được sắp theo thứ tự alphabet.

Kiểu cấu trúc được định nghĩa như sau :

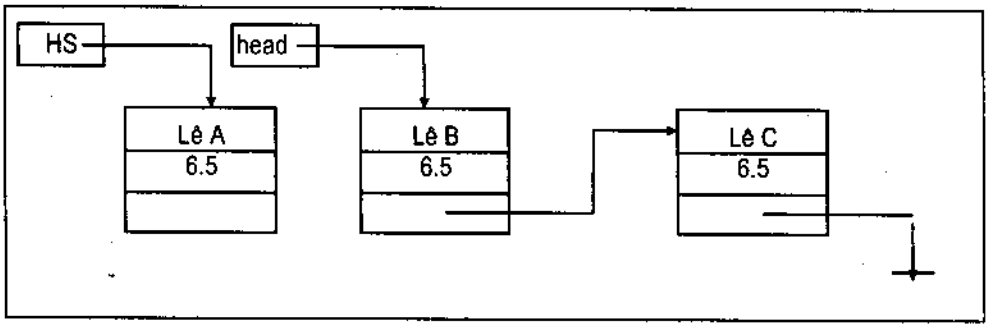
```
typedef struct hoc_sinh{
            char ho_ten[20];
            float diem;
            struct hoc_sinh *tiep;
        }t_hoc_sinh;
t_hoc_sinh *head; /* con trỏ đến đầu danh sách */
```

Mỗi phần tử chứa thông tin của một học sinh và một con trỏ trỏ 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 được gán bằng NULL chỉ ra 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, ta phải xin cấp phát một vùng bộ nhớ có kích thước đủ để chứa phần tử đó. Toán tử sizeof(tên_kiểu) cho biết kích thước cần thiết để cấp phát cho một biến có kiểu là tên_kiểu. Tất nhiên khi cấp phát nên kiểm tra xem liệu có còn đủ bộ nhớ hay không, điều này là cần thiết để chương trình không bị treo.

```
#define SIZEHS sizeof(t_hoc_sinh)
    t_hoc_sinh *hs;
    hs = NULL;
if((hs = (t_hoc_sinh *)malloc(SIZEHS)) == NULL)
{
    printf("\n Khong con du bo nho de cap phat");
    getch();
}
```

Chú ý, cần gán hs = NULL để đề phòng trường hợp hs đang trỏ tới một thành phần cấu trúc nào đó trong danh sách.

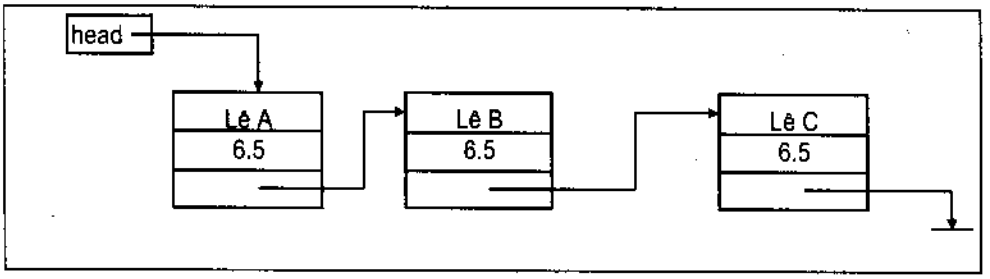


Danh sách trước khi thêm phần tử mới

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 :

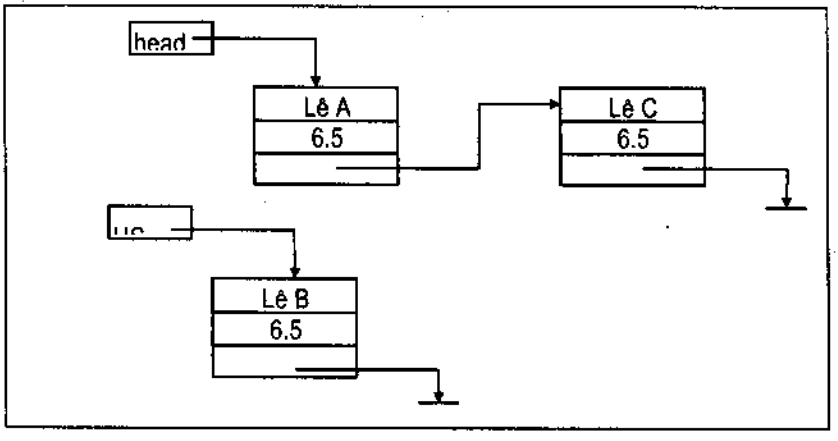
– Nếu phần tử mới ở đầu danh sách, cần phải sửa lại con trỏ head.



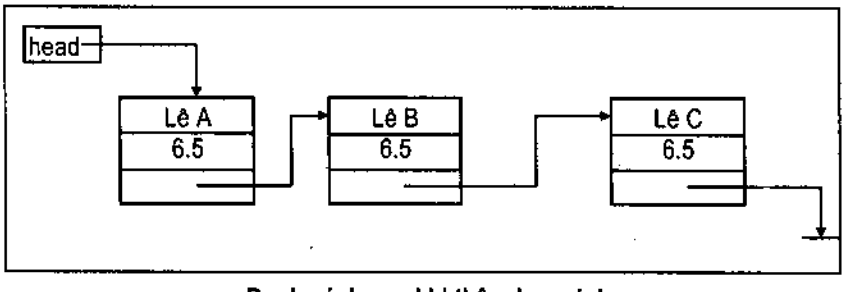
Danh sách sau khi chèn phần tử mới vào đầu

– 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 thêm học sinh



Danh sách sau khi thêm học sinh

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 hoạ 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 SIZEHS sizeof(t_hoc_sinh)
main()
{
   t_hoc_sinh *hs,*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 ((hs = (t_hoc_sinh *)(malloc(SIZEHS))) == NULL)
{
   printf("Khong du bo nho\n");
   break;
}
else
{
   n ++;
   strcpy(hs -> ho_ten,name);
   printf(" Diem : ");
   scanf("%f",&d);
   hs -> diem = d;
   hs -> tiep = NULL;
   /* chèn phần tử mới vào danh sách */
   if (head == NULL)
       head = hs;
   else
{
   p = head;
   while(p != NULL&&strcmp(p -> ho_ten, hs ->
                                        ho_ten) < 0)
  {
       q = p;
       p = p -> tiep;
  }
   if (p != NULL && strcmp(p -> ho_ten, hs ->
                                        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(hs);
          n --;
          continue;
      }
     else while(p != NULL&&strcmp(p -> ho_ten, hs ->
                                          ho_ten) == 0)
      {
          q = p;
          p = p -> tiep;
      }
   }
      if (p == head)
      {
          hs -> tiep = head;
          head = hs;
      }
          else {
                  q -> tiep = hs;
                  hs -> tiep = p;
               }
          }
      }
   }
}
while (strcmp(name,"") != 0);
/* Duyệt lại danh sách và in ra danh sách học sinh
                                                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 hoc sinh 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("\n Co 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 hoc sinh phai thi lai
1. Do Thanh Xuan 3.0
2. Pham Hai Yen 4.0
Co tong so 2 hoc sinh phai thi lai

IV – KIỂU HỢP (UNION)

1. Khái niệm

Một biến kiểu union cũng bao gồm nhiều thành phần giống như một biến kiểu cấu trúc, nhưng khác biến kiểu cấu trúc ở chỗ : các thành phần

của biến kiểu cấu trúc được cấp phát các vùng nhớ khác nhau, còn các thành phần của biến kiểu 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. Định nghĩa 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.

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 2 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 tách ra byte cao và thấp của một số nguyên.

Ví dụ : Khai báo một union có thành phần là kiểu cấu trúc.

```
struct ng{
            int ngay;
            int thang;
            int nam;
        };
```

```
struct diachi{
                int sonha;
                char *tenpho;
            };
union u {
            struct ng date;
            struct diachi address;
        } diachi_ngaysinh;
```

V – BÀI TẬP MINH HOẠ

Bài 1.

```
/* Chương trình minh hoạ 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 \n 1] 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 chi 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 hoạ 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. */
                      /* Hiển 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);
  printf("Số lượng : %d\n",resistors.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 3.

```
/* Chương trình minh hoạ 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");
   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 hoạ 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

Bài 5.

```
/* Minh hoạ 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 6.

```
/* Minh hoạ 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("\n Nhập dữ liệu : ");
      item = getche();
      if(item != '\r')
      add_node(&n1,item);
   } while(item != '\r');
   printf("\n Danh 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 7.

```
/* Chương trình minh hoạ 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 hiển 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 hiển thị nó : */
   integer_or_float.value = 123.45;
   printf("Giá trị của số thực là %f\n",
                              integer_or_float.value);
   printf("Địa chỉ bắt đầu là => %d\n",
                             &integer_or_float.value);
}
```

Kết quả :

Kích cỡ kiểu union => 4

Giá trị của số nguyên = 123

Địa chỉ bắt đầu là => 7042

Giá trị của số thực là 123.45

Địa chỉ bắt đầu là => 7042

VI – BÀI TẬP TỰ GIẢI

Bài 1. Viết chương trình dùng kiểu 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 có kiểu 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 Hoá 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 4. Viết chương trình nhập thêm một nút vào đầu danh sách.

Bài 5. Viết chương trình nhập thêm một nút vào cuối danh sách.

Bài 6. Viết chương trình loại bỏ một nút ở đầu danh sách.

Bài 7. Viết chương trình loại bỏ một nút ở cuối danh sách.

Chương 12

THAO TÁC VỚI TỆP TIN (FILE)

I – KHÁI NIỆM VỀ TỆP TIN

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

Tệp tin được chứa trong bộ nhớ ngoài, được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện.

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

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

II – KHAI BÁO DỮ LIỆU

Trong C có hai loại hàm thao tác trên tệp tin :

– *Hàm cấp 1 :* Là những hàm cấp thấp làm việc với tệp tin thông qua một số hiệu tệp 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 tệp tin thông qua một con trỏ tệp tin. Con trỏ này được xác định khi ta mở tệp tin. Do đó để dùng các hàm cấp 2 ta phải khai báo biến con trỏ tệp tin.

Ví dụ :

```
FILE *f;
     .....
f = fopen("data.dat","wt");
```

III – CÁC KIỂU NHẬP, XUẤT TRONG TỆP TIN

1. Nhập, xuất kiểu nhị phân

Dữ liệu ghi lên tệp tin không bị thay đổi và khi ta đóng tệp tin thì mã kết thúc tệp tin EOF sẽ được ghi lên đĩa là -1.

2. Nhập, xuất kiểu văn bản

Kiểu nhập, xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc - ghi như nhau.

Mã chuyển dòng :

– Khi ghi một ký tự LF (mã 10) được chuyển thành hai ký tự : CR (mã 13) và LF.

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

3. Các điểm lưu ý

– Tệp 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.

4. Ví dụ so sánh hai kiểu nhập, xuất

Ví dụ : Ta có chương trình sau sử dụng phép đọc/ghi trên một tệp tin khác nhau.

```
/* Chuong trinh vi du ve hai hinh thuc doc/ghi tep
tin theo kieu van ban va nhi phan */
#include<stdio.h>
#include<conio.h>
main()
{
   FILE *f;
   char c,c1;
   clrscr();
   f = fopen("s1","wt");
   fprintf(f,"%c%c%c%c%c%c",65,66,67,68,10,26);
   /* ghi len tep tin 6 byte theo kieu text
        do vay khi gap byte 10 no se ghi len 13,10 */
   fclose(f);
   getch();
   f = fopen("s1","rt");
   while (!feof(f))
   {  c = getc(f);
      printf("%d ",c);
   }
/* doc ket qua cua tep tin ghi tren theo kieu van ban
va in len man hinh ta thay tren man hinh se co ket
qua la 65,66,67,68,10, - 1. Trong do - 1 la ma ket thuc
tep tin. Suy ra thieu ma 26 do 26 duoc hieu la ma ket
thuc tep tin trong kieu nhap, xuat van ban. Neu thay
dong f = fopen("s1","rt") bằng f = fopen("s1","rb")
thi ket qua in len man hinh là 65 66 67 68 13 10 26
-1 */
   fclose(f);
   getch();
}
```

Một ví dụ khác là tệp 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 tệp tin (do gặp mã 26).

IV – CÁC HÀM THAO TÁC TRÊN TỆP 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)

Hàm này dùng để mở một tệp 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 tệp tin có thành công hay không.

– *Tên tệp tin* là một hằng chuỗi, hoặc một con trỏ trỏ đến vùng nhớ chứa tên tệp tin.

– *Kiểu* là hằng chuỗi cho biết kiểu truy nhập :

Kiểu	Ý nghĩa
"r" "rt"	Mở tệp tin để đọc theo kiểu văn bản. Tệp tin phải có trên đĩa nếu không sẽ có lỗi.
"w" "wt"	Mở tệp tin để ghi theo kiểu văn bản. Nếu tệp tin đã có trên đĩa nó sẽ bị xoá.
"a" "at"	Mở tệp tin để ghi bổ sung theo kiểu văn bản. Nếu tệp tin chưa có thì tạo mới.
"r + " "r + t"	Mở tệp tin để đọc/ghi theo kiểu văn bản. Tệp tin phải có trên đĩa nếu không sẽ có lỗi.
"w + "."w + t"	Mở tệp tin để đọc/ghi theo kiểu văn bản. Nếu tệp tin đã có trên đĩa nó sẽ bị xoá.
"a + " "a + t"	Mở tệp tin để đọc/ghi bổ sung theo kiểu văn bản. Nếu tệp tin chưa có thì tạo mới.
"rb"	Mở tệp tin để đọc theo kiểu nhị phân, tệp tin phải có trên đĩa, nếu không sẽ có lỗi.
"wb"	Mở tệp tin để ghi theo kiểu nhị phân. Nếu tệp tin đã có trên đĩa nó sẽ bị xoá.
"ab"	Mở tệp tin để ghi bổ sung theo kiểu nhị phân. Nếu tệp tin chưa có thì tạo mới.
"r + b"	Mở tệp tin để đọc/ghi theo kiểu nhị phân, tệp tin phải có trên đĩa nếu không sẽ có lỗi.
"w + b"	Mở tệp tin để đọc/ghi theo kiểu nhị phân. Nếu tệp tin đã có trên đĩa nó sẽ bị xoá.
"a + b"	Mở tệp tin để đọc/ghi bổ sung theo kiểu nhị phân. Nếu tệp tin chưa có thì tạo mới.

2. int fclose(FILE *f)

Đóng tệp tin được trỏ đến bởi con trỏ f. Nếu thành công thì giá trị của hàm bằng 0, ngược lại có giá trị là 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 tệp tin đang mở. Nếu thành công giá trị của hàm bằng số tệp tin đã đóng, ngược lại cho giá trị EOF.

4. int fflush(FILE *f)

Làm sạch vùng đệm của tệp tin được trỏ 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 tệp tin đang mở. Nếu thành công giá trị của hàm bằng số tệp tin đang mở, ngược lại cho giá trị EOF.

6. int unlink(const char *tên_tập_tin)

Xóa một tệp 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 tệp 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 0 nếu ở cuối tệp tin, ngược lại bằng 0.

V – CÁC HÀM NHẬP, XUẤT

1. Nhập, xuất ký tự (Dùng cho kiểu nhị phân và văn bản)

- **Ghi ký tự lên tệp tin**

```
int putc(int ch, FILE *f)
int fputc(int ch, FILE *f)
```

Ghi lên file f ký tự có mã là ch %256. Nếu thành công kết quả bằng mã của ký tự đã ghi, ngược lại là EOF.

Trong trường hợp ghi theo văn bản thì khi gặp mã 10 sẽ ghi thành 13 và 10.

• **Đọc ký tự từ tệp tin**

```
int getc(FILE *f)
int fgetc(FILE *f)
```

Đọc một ký tự từ file f. Nếu thành công kết quả bằng mã của ký tự đọc được, ngược lại bằng -1.

2. Nhập, xuất chuỗi (Dùng cho kiểu văn bản)

• **Ghi một chuỗi**

```
int fputs(const char *s, FILE *f)
```

Ghi một chuỗi được trỏ tới bởi con trỏ s vào file f.

Kết quả là ký tự cuối được ghi nếu thành công, ngược lại là 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 là NULL.

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,...)
```

Ở đây "..." là danh sách các đối 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,...)
```

Ở đây "..." là danh sách các đối 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 tương ứng.

Ví dụ : Đọc nội dung của một tệp tin theo một cấu trúc như sau :

– Tệp tin ghi thông tin của n đỉnh của một đa giác. Mỗi đỉnh có toạ độ là (x, y) kiểu số nguyên;

– Tệp tin có n + 1 dòng;

– Dòng đầu chứa số đỉnh (n);

– n dòng còn lại, mỗi dòng chứa toạ 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);
}
```

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 số nguyên n (2 byte) từ file f. Nếu thành công kết quả của hàm bằng giá trị đọc được, ngược lại là EOF.

5. Đọc/ghi kiểu cấu trúc

• **Ghi mẫu tin lên tệp tin**

```
int fwrite(void *ptr, int size, int n, FILE *f)
```

Ở đây : ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi; size là kích thước của một mẫu tin (tính theo byte); n là 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 tệp tin**

```
int fread(void *ptr, int size, int n, FILE *f)
```

Ở đây : ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu đọc được; size là kích thước của một mẫu tin (tính theo byte); n là 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 trỏ đến. Kết quả của hàm bằng số mẫu tin được đọc thật sự.

VI – CÁC HÀM DI CHUYỂN CON TRỎ

1. Đưa con trỏ về đầu tệp tin

Cú pháp :

```
void rewind(FILE *f)
```

Chức năng : Đưa con trỏ về đầu tệp tin.

2. Dời con trỏ tệp tin

Cú pháp :

```
int fseek(FILE *f, long số_byte, int vt_bắt_đầu)
```

Chức năng : Dời con trỏ tệp 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 tệp tin;

SEEK_CUR (1) : vị trí hiện hành;

SEEK_END (2) : vị trí cuối tệp tin.

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 tệp tin.

VII – CÁC HÀM QUẢN LÝ THƯ MỤC

Các hàm này được định nghĩa trong dir.h.

1. int chdir(char *s)

Đổi tên thư mục hiện hành, s là con trỏ trỏ tới vùng nhớ chứa tên mới của thư mục (có thể chứa cả ổ đĩa và đường dẫn).

Kết quả bằng 0 nếu thành công, ngược lại bằng -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 trỏ tới.

Kết quả bằng 0 nếu thành công, ngược lại bằng -1.

4. int rmdir(char *s)

Xóa thư mục có tên được lưu tại vùng nhớ do s trỏ tới, thư mục cần xóa phải rỗng.

Kết quả bằng 0 nếu thành công, ngược lại bằng -1.

5. int findfirst(char *path, struct ffblk *fd, int attrib)

Tìm tệp tin đầu tiên thoả mãn điều kiện được chỉ ra.

Trong đó : path là đường dẫn chứa tên thư mục hoặc tên tệp tin cần tìm; fd là con trỏ trỏ đến vùng nhớ để lưu các thông tin của tệp 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_ftime;
     unsigned ff_fdate;
     long ff_fsize;
     char ff_fname[13];
     char ff_freserved[21];
}
```

attrib là thuộc tính của tệp tin cần tìm, có thể là một hoặc là hợp của các giá trị : 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 tệp tin đầu tiên có thuộc tính attrib. Ta nên hiểu ở đây thư mục cũng là một tệp 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 tệp tin có thuộc tính như vậy. Kết quả bằng 0 nếu thành công, ngược lại bằng -1.

6. int findnext(struct ffblk *fd)

Tiếp tục tìm tệp tin có yêu cầu như trong hàm findfirst. Nếu tìm thấy thông tin này lại được lưu vào vùng nhớ do fd trỏ đến.

VIII – BÀI TẬP MINH HOẠ

Bài 1.

```
/* Chương trình minh hoạ cách tạo, mở một file và
nhập một ký tự vào file */
#include<stdio.h>
main()
{
   FILE *file_pointer;   /* Đây là con trỏ file. */
      /* Khởi tạo tệp tin  myfile.dta và gán địa chỉ
            của nó đến con trỏ file file_pointer : */
   file_pointer = fopen("MYFILE.DTA","w");
```

```
      /* Nhập một ký tự vào file đã mở : */
   putc('C', file_pointer);
      /* Đóng file vừa tạo. */
   fclose(file_pointer);
}
```

Bài 2.

```
/* Chương trình minh hoạ 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; /* Ký 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 ký tự đầu tiên từ file đã mở. */
   file_character = getc(file_pointer);
      /* Xuất ký tự lên màn hình. */
   printf("Ký tự là %c\n", file_character);
      /* Đóng file đã tạo. */
   fclose(file_pointer);
}
```

Bài 3.

```
/* Chương trình minh hoạ cách nhập một chuỗi ký tự
vào file */
#include<stdio.h>
main()
{
   FILE *file_pointer;   /* Đây là con trỏ file. */
   char file_character; /* Ký 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 hoạ cách đọc một chuỗi ký tự từ
file đã tồn tại trên đĩa */
#include<stdio.h>
main()
{
   FILE *file_pointer;   /* Đây là con trỏ file. /*
   char file_character; /* Ký 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 ký 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 tệp 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 hoạ 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;    /* Ký tự file được lưu. */
    FILE *file_pointer;    /* Con trỏ file. */
           /* Hiển 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("Khong the mo duoc 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 hoạ 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("\n Name 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 tệp 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.ff_ftime);
     if (sf%20 == 0)
    { printf("\n Bam enter xem tiep ");
     getch();
    }
    }
   }
  }
```

IX – BÀI TẬP TỰ GIẢI

Bài 1. Viết chương trình đọc vào một file, sau đó chuyển các chữ hoa thành chữ thường và lưu trên đĩa.

Bài 2. Viết chương trình đếm các ký tự chữ cái có trong một tệp tin.

Bài 3. Viết chương trình đếm số từ của một tệp tin.

Bài 4. Viết chương trình đếm số lần lặp lại của một từ trong một tệp tin.

Bài 5. Viết chương trình đổi toàn bộ các ký tự trong một tệp tin sang chữ hoa.

Bài 6. Viết chương trình đổi ngược trật tự các ký tự của các dòng trong một tệp tin.

Bài 7. Viết chương trình gồm hai chức năng :

– Nhập và lưu các hệ số a, b, c của các phương trình bậc hai vào một tệp tin.

– Tìm nghiệm của các phương trình bậc hai có hệ số a, b, c được lưu trong tệp tin trên và lưu kết quả vào một tệp tin khác.

Bài 8. Viết chương trình ghi một danh sách cấu trúc vào tệp tin, sau đó đọc để kiểm tra lại.

Bài 9. Viết chương trình ghi các dòng văn bản nhập từ bàn phím vào tệp tin.

Bài 10. 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 11. Viết chương trình nhập dữ liệu ghi vào đĩa các thành phần HO, TEN, TUOI, CHUCVU, BACLUONG thành tệp tin LUONG.DTA. Khi nào không muốn nhập nữa thì nhấn phím ESC.

Bài 12. Viết chương trình liệt kê tất cả các tệp tin có trong một thư mục. Hãy viết lại để chương trình chỉ liệt kê các tệp tin theo ý muốn.

Ví dụ như : *.txt, *.bak,...

Bài 13. Viết chương trình có công dụng như lệnh CD của DOS.

Bài 14. Viết chương trình có công dụng như lệnh RD của DOS.

Bài 15. Viết chương trình xoá các tệp tin có trong một thư mục và trong các thư mục con của nó (với đường dẫn được nhập từ bàn phím).

Chịu trách nhiệm xuất bản :
Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Tổ chức bản thảo và chịu trách nhiệm nội dung :
Chủ tịch HĐQT kiêm Giám đốc Công ty CP Sách ĐH – DN
TRẦN NHẬT TÂN

Biên tập và sửa bản in :
ĐỖ HỮU PHÚ

Trình bày bìa :
LƯU CHÍ ĐỒNG

Chế bản :
QUANG CHÍNH

GIÁO TRÌNH KĨ THUẬT LẬP TRÌNH C

Mã số : 7B681M7 – DAI

In 1.500 cuốn (QĐ 76), khổ 16 x 24. In tại Công ty In – Thương mại TTXVN.
Địa chỉ : 70/342 Khương Đình, Hạ Đình, Thanh Xuân, Hà Nội.
Số ĐKKH xuất bản : 192 – 2007/CXB/6 – 411/GD.
In xong và nộp lưu chiểu tháng 10 năm 2007.

CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYÊN – HÀ NỘI
Website : www.hevobco.com.vn

VƯƠNG MIỆN KIM CƯƠNG
CHẤT LƯỢNG QUỐC TẾ

TÌM ĐỌC SÁCH THAM KHẢO KỸ THUẬT
CỦA NHÀ XUẤT BẢN GIÁO DỤC

1.	Giáo trình Vật lý đại cương tập một	Lương Duyên Bình
2.	Giáo trình Vật lý đại cương tập hai	Lương Duyên Bình
3.	Bài tập Vật lý đại cương tập một	Lương Duyên Bình
4.	Bài tập Vật lý đại cương tập hai	Lương Duyên Bình
5.	Giáo trình Kỹ thuật xung số	Đặng Văn Chuyết (CB)
6.	Giáo trình Kỹ thuật lập trình C	Nguyễn Linh Giang (CB)
7.	Giáo trình Kỹ thuật mạch điện tử	Đặng Văn Chuyết (CB)
8.	Giáo trình Linh kiện điện tử	Nguyễn Viết Nguyên (CB)
9.	Giáo trình Lý thuyết điều khiển tự động	Phan Xuân Minh (CB)
10.	Giáo trình Xử lý số tín hiệu	Nguyễn Quốc Trung (CB)
11.	Giáo trình Vi xử lý và cấu trúc máy tính	Ngô Diên Tập (CB)
12.	Giáo trình Khí cụ điện	Phạm Văn Chới
13.	Lập trình C# từ cơ bản đến nâng cao	Phạm Công Ngô
14.	Cơ sở tự động điều khiển quá trình	Nguyễn Văn Hoà

Bạn đọc có thể mua tại các Công ti Sách - Thiết bị trường học ở các địa phương hoặc các Cửa hàng của Nhà xuất bản Giáo dục :

Tại Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;

Tại Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh ;

Tại Thành phố Hồ Chí Minh : 104 Mai Thị Lựu, Quận 1 ; Cửa hàng 451B - 453, Hai Bà Trưng, Quận 3 ; 240 Trần Bình Trọng – Quận 5.

Tại Thành phố Cần Thơ : Số 5/5, đường 30/4 ;

Website : www.nxbgd.com.vn

8 934980 777671

Giá: **23.500 đ**