BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

TH.S. TRẦN THỊ HOA

GIÁO TRÌNH
LẬP TRÌNH CĂN BẢN

Hà Nội, 2013

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

TH.S. TRẦN THỊ HOA

GIÁO TRÌNH
LẬP TRÌNH CĂN BẢN

Hà Nội, 2013

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH

LỜI NÓI ĐẦU

Theo khung chương trình của Bộ Giáo Dục và Đào Tạo, Lập trình căn bản là một phần quan trọng trong học phần Tin học Đại cương thuộc các khối ngành Khoa học Tự nhiên, đặc biệt là ngành Công nghệ thông tin.

Nhằm đáp ứng yêu cầu học tập của sinh viên bước đầu làm quen với công việc lập trình, tác giả đã biên soạn Giáo Trình Lập trình căn bản nhằm giúp cho sinh viên có một tài liệu học tập, rèn luyện tốt khả năng lập trình, tạo nền tảng vững chắc cho các môn học tiếp theo trong chương trình đào tạo kỹ sư Công nghệ thông tin.

Giáo trình bao gồm những kiến thức từ đơn giản đến phức tạp. Nội dung của giáo trình được chia thành bảy chương. Trong mỗi chương đều có phần giới thiệu lý thuyết, phần bài tập mẫu và cuối cùng là phần bài tập tự giải để sinh viên tự mình kiểm tra những kiến thức và kinh nghiệm đã được học. Sau đây là nội dung chính của các chương:

- Chương 1 ngoài việc giới thiệu các khái niệm cơ bản của ngôn ngữ lập trình C còn đưa ra một số chương trình C đơn giản và cách thực hiện chúng trên máy để giúp người đọc mau chóng tiếp cận với máy.

- Chương 2 trình bày về các kiểu dữ liệu, cách biểu diễn các giá trị dữ liệu và cách tổ chức (lưu trữ) dữ liệu trong biến và mảng. Ngoài ra chương này còn giới thiệu về biểu thức, về các cách xử lý dữ liệu đơn giản nhờ các phép toán, biểu thức và câu lệnh gán.

- Chương 3 trình bày về các hàm vào ra dữ liệu trên bàn phím và màn hình.

- Chương 4 trình bày về lớp toán tử rất quan trọng dùng để thể hiện các thuật toán, đó là toán tử rẽ nhánh if, toán tử lựa chọn switch, các toán tử tạo lập chu trình (vòng lặp) for, while, do – while và toán tử goto.

- Chương 5 trình bày cách tổ chức chương trình thành các hàm, các quy tắc xây dựng và sử dụng hàm. Các vấn đề hay và khó ở đây là con trỏ, con trỏ hàm và kỹ thuật đệ quy.

- Chương 6 trình bày về một kiểu dữ liệu quan trọng là cấu trúc, cấu trúc tự trỏ và danh sách liên kết.

- Chương 7 trình bày về các thao tác trên tệp như: tạo một tệp mới, ghi dữ liệu từ bộ nhớ lên tệp, đọc dữ liệu từ tệp vào bộ nhớ,…

Khi viết, tác giả đã cố gắng để giáo trình được hoàn chỉnh, song chắc chắn không tránh khỏi thiếu xót, vì vậy rất mong nhận được sự đóng góp ý kiến của độc giả để giáo trình ngày càng hoàn thiện hơn.

Tác giả cũng xin chân thành cảm ơn các đồng nghiệp ở Khoa Công nghệ Thông tin, Học viện Kỹ thuật Mật mã đã giúp đỡ, đóng góp ý kiến để hoàn chỉnh nội dung giáo trình này.

Hà Nội, tháng 11, 2013

TÁC GIẢ

CHƯƠNG 1: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C

Trong chương này, tác giả sẽ giới thiệu những thành phần cơ bản của ngôn ngữ lập trình C (cũng như của bất kỳ ngôn ngữ lập trình nào khác) đó là: tập ký tự, từ khóa, tên,…Ngoài ra, để giúp bạn đọc mau chóng tiếp cận với máy, tác giả cũng giới thiệu một vài chương trình đơn giản và cách vận hành chúng trên máy để có thể nhận được kết quả cuối cùng.

1.1. Giới thiệu ngôn ngữ C

Lập trình cấu trúc là phương pháp tổ chức, phân chia chương trình thành các hàm, thủ tục, chúng được dùng để xử lý dữ liệu nhưng lại tách rời các cấu trúc dữ liệu. Thông qua các ngôn ngữ Foxpro, Pascal, C đa số những người làm Tin học đã khá quen biết với phương pháp lập trình này.

Vào đầu những năm 70 tại phòng thí nghiệm Bell đã phát triển ngôn ngữ C. C được sử dụng lần đầu trên một hệ thống cài đặt hệ điều hành UNIX. C có nguồn gốc từ ngôn ngữ BCPL do Martin Richards phát triển. BCPL sau đó đã được Ken Thompson phát triển thành ngôn ngữ B, đây là người khởi thủy ra C.

Trong khi BCPL và B không hỗ trợ kiểu dữ liệu, thì C đã có nhiều kiểu dữ liệu khác nhau. Những kiểu dữ liệu chính gồm: kiểu ký tự (character), kiểu số nguyên (interger) và kiểu số thực (float).

C được dùng để lập trình hệ thống. Một chương trình hệ thống có ý nghĩa liên quan đến hệ điều hành của máy tính hay những tiện ích hỗ trợ nó. Hệ điều hành (OS), trình thông dịch (Interpreters), trình soạn thảo (Editors), chương trình Hợp Ngữ (Assembly) là các chương trình hệ thống. Hệ điều hành UNIX được phát triển dựa vào C. C đang được sử dụng rộng rãi bởi vì tính hiệu quả và linh hoạt. Trình biên dịch (compiler) C có sẵn cho hầu hết các máy tính. Mã lệnh viết bằng C trên máy này có thể được biên dịch và chạy trên máy khác chỉ cần thay đổi rất ít hoặc không thay đổi gì cả. Trình biên dịch C dịch nhanh và cho ra mã đối tượng không lỗi.

Ngoài việc C được dùng để viết hệ điều hành Unix , người ta đã nhanh chóng nhận ra sức mạnh của C trong việc xử lý các vấn đề khác nhau của tin học như xử lý số, văn bản, cơ sở dữ liệu và đặc biệt là trong lập trình hướng đối tượng khi C được phát triển thành C++.

1.2. Tập ký tự, từ khóa, tên gọi

1.2.1. Tập ký tự

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Tiếp theo, đến lượt mình, các từ lại được liên kết theo một nguyên tắc nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh để diễn đạt một thuật toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau:

- 26 chữ cái latinh thường: a, b, c,…, x, y, z
- 26 chữ cái latinh hoa: A, B, C,..., X, Y, Z
- 10 chữ số: 0, 1,…, 8, 9
- Các ký tự toán học: + - * / = ( )
- Ký tự gạch nối _
- Các ký tự đặc biệt khác như: . , : ; [ ] { } ? ! \ & | % # $...

Chú ý:

- Dấu cách thực sự là một khoảng cách dùng để tách các từ. Ví dụ Ha noi gồm 6 ký tự còn Hanoi gồm 5 ký tự.
- Khi viết chương trình ta không được sử dụng bất kỳ ký hiệu nào khác ngoài tập ký tự nói trên.

1.2.2. Từ khóa

Từ khóa là một số từ mà ngôn ngữ C đặt trước có ý nghĩa hoàn toàn xác định và thường được dùng để khai báo các kiểu dữ liệu, viết các toán tử và các câu lệnh. Các từ khóa phải được viết bằng chữ thường. Trong C có các từ khóa sau:

Bảng 0.1. Danh sách từ khóa của ngôn ngữ C

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

Chú ý: Các từ khóa không được viết hoa mà phải viết bằng chữ thường. Chẳng hạn không viết là INT mà viết là int.

1.2.3. Tên gọi

Khái niệm tên rất quan trọng trong quá trình lập trình, nó không những thể hiện rõ ý nghĩa trong chương trình mà còn dùng để xác định các đối tượng khác nhau khi thực hiện chương trình.

Tên thường được đặt cho hằng, biến, mảng, con trỏ, nhãn… Chiều dài tối đa của tên là 32 ký tự.

Tên hợp lệ là một chuỗi ký tự liên tục gồm: ký tự chữ, ký tự số và dấu gạch dưới. Ký tự đầu của tên phải là ký tự chữ hoặc dấu gạch dưới. Khi đặt tên không được đặt trùng với các từ khóa.

Ví dụ 1.1. Một số tên gọi

Các tên đúng: delta, a_1, Num_ODD, Case

Các tên sai:	3a_1	(ký tự đầu là ký tự số)
	num-odd	(sử dụng dấu gạch ngang)
	int	(đặt tên trùng với từ khóa)
	del ta	(có khoảng trắng)
	f(x)	(có dấu ngoặc tròn)

Chú ý: Trong C, tên phân biệt chữ hoa và chữ thường. Chữ hoa thường được dùng để đặt tên cho các hằng và chữ thường hay dùng để đặt tên cho hầu hết các đối tượng khác như: biến, mảng, hàm, cấu trúc.

Ví dụ 1.2 : number khác Number

case khác Case (case là từ khóa, do đó đặt tên là Case thì vẫn đúng)

1.3. Câu lệnh, khối lệnh, lời giải thích

1.3.1. Câu lệnh

Chương trình C được tổ chức thành nhiều câu lệnh, mỗi câu lệnh có thể thực hiện một công việc nào đó nhưng cuối mỗi câu lệnh phải được kết thúc bằng dấu chấm phẩy ‘;’. Trên một dòng có thể viết nhiều câu lệnh và một câu lệnh cũng có thể viết trên nhiều dòng.

Tại những vị trí của câu lệnh mà cho phép xuất hiện một hoặc nhiều dấu khoảng cách thì ta có thể ngắt phần còn lại của câu lệnh xuống dòng tiếp theo.

Ví dụ 1.3: Cho đoạn chương trình sau

```
void    main()
{
        int  x,y;
        x = 7; y = 8;
        if
        ( x>y)
        printf("x > y");
}
```

Trong ví dụ trên, ở dòng lệnh thứ hai, gồm có hai câu lệnh. Câu lệnh if được viết trên ba dòng thứ 3, 4 và 5.

1.3.2. Khối lệnh

Các câu lệnh được đặt trong cặp dấu { } được gọi là một khối lệnh. Khối lệnh được xem như một câu lệnh riêng biệt.

Trong cú pháp của các câu lệnh, tại những vị trí chỉ cho phép xuất hiện một câu lệnh mà ta muốn thực hiện nhiều câu lệnh thì ta phải đặt những câu lệnh đó trong một khối lệnh.

1.3.3. Lời giải thích

Trong khi lập trình cần phải ghi chú để giải thích các biến, hằng, các thao tác xử lý giúp cho chương trình rõ ràng dễ hiểu, dễ nhớ, dễ sửa chữa và để người khác đọc vào dễ hiểu. Trong C có các kiểu ghi chú sau:

- // nội dung ghi chú. Khi đó nội dung ghi chú được ghi trên một dòng hoặc phần còn lại của một dòng.

- /* nội dung ghi chú */. Khi đó nội dung ghi chú có thể ghi trên một dòng, trên nhiều dòng hoặc trên phần còn lại của một dòng.

Ví dụ 1.4: Chương trình tìm số lớn của hai số a và b

```
void main()
{
```

```
int a, b;          //khai bao biến a, b kiểu int
a = 1;             //gán 1 cho a
b =3;              //gán 3 cho b
/*  thuật toán tìm số lớn nhất là nếu a lớn hơn b thì a là số lớn nhất
 ngược lại b là số lớn nhất */
if (a > b)         printf("max = %d", a);
else     printf("max: %d", b);
}
```

Khi biên dịch chương trình, C gặp các cặp dấu ghi chú thì sẽ bỏ qua và không dịch ra ngôn ngữ máy.

1.4. Chỉ thị tiền biên dịch

Cú pháp: # include <[đường dẫn]\tên tệp> hoặc (1)

include "[đường dẫn]\tên tệp" (2)

Công dụng: Nếu trong một chương trình C có sử dụng các hàm, hằng, biến, các kiểu dữ liệu được khai báo trong một chương trình C khác (có thể của ngôn ngữ C hoặc do người dùng tự viết) thì ta phải sử dụng chỉ thị trên để chỉ thị cho bộ tiền xử lý chép nội dung của các tệp chứa các hàm, hằng, biến, kiểu dữ liệu trên vào vị trí đặt nó.

Sự khác nhau giữa hai dạng khai báo trên: Dạng (1) khác dạng (2) chỉ khi không có thông tin về đường dẫn (tức là người lập trình chỉ ghi tên tệp mà không ghi đầy đủ cả đường dẫn). Với dạng (1) thì chương trình sẽ tìm tệp có tên đưa ra trong thư viện INCLUDE của trình biên dịch. Còn với dạng (2) thì chương trình trước tiên sẽ tìm tệp trong thư mục chủ, nếu không tìm thấy thì sẽ tiếp tục tìm trong thư viện INCLUDE.

Chú ý:

- Các tệp thư viện có sẵn trong C được gọi là các tệp tiêu đề và có đuôi là .h
- Các hàm vào ra chuẩn nằm trên hai tệp stdio.h và conio.h
- Các hàm toán học nằm trong tệp math.h
- Các hàm xử lý chuỗi nằm trong tệp string.h.
- Các hàm chuyển đổi dữ liệu nằm trong tệp ctype.h

- Mỗi một chỉ thị #include chỉ khai báo được một tệp, nếu muốn khai báo nhiều tệp thì ta phải sử dụng nhiều dòng #include.

Ví dụ 1.5. Giả sử một chương trình C cần sử dụng đến các câu lệnh nhập/xuất dữ liệu, các hàm toán học và các hàm xử lý chuỗi thì ta phải khai báo các tệp stdio.h, math.h và string.h như sau:

#include”stdio.h”

#include”math.h”

#include”string.h”

1.5. Cấu trúc của một chương trình C

Ngôn ngữ C không có khái niệm thủ tục. Mọi chương trình con trong C đều được tổ chức thành các hàm. Hàm là một đơn vị độc lập trong chương trình. Tính độc lập của hàm thể hiện trên hai điểm sau:

- Không cho phép xây dựng một hàm bên trong hàm khác. Điều này khác biệt với ngôn ngữ Pascal

- Mỗi một hàm, có thể có các biến, mảng, các kiểu dữ liệu riêng của hàm đó và chúng chỉ được dùng nội bộ bên trong thân hàm chứa nó.

Một chương trình C bao gồm một hoặc nhiều hàm trong đó bắt buộc phải có một hàm main(). Chương trình được bắt đầu thực hiện từ câu lệnh đầu tiên của hàm main() và kết thúc khi thực hiện xong câu lệnh cuối cùng và gặp ký tự đóng ‘ } ‘ của hàm này.

Khi thực hiện chương trình, thông qua các lời gọi hàm mà máy có thể đi từ hàm này đến hàm khác.

Thông thường, cấu trúc chung của một chương trình C như sau:

……

Khai báo nguyên mẫu hàm 1

…..

Khai báo nguyên mẫu hàm n

Khai báo hàm n + 1

……

Khai báo hàm n + m

```
void  main()
{
        <thân hàm main>
}
Khai báo hàm 1
…….
Khai báo hàm n
```

Bên ngoài các hàm, tại các vị trí … ta có thể đặt các câu lệnh sau:

```
# include   //Khai báo các tệp
# define    // Khai báo các tên hằng
typedef     // Định nghĩa kiểu dữ liệu
khai báo biến ngoài, mảng ngoài,…
```

Một số lưu ý khi viết chương trình C: Để viết chương trình có cấu trúc được rõ ràng, dễ kiểm tra và tránh nhầm lẫn, ta nên viết chương trình theo các quy tắc sau:

- Các câu lệnh và khối lệnh nằm trong một toán tử điều khiển thì viết thụt vào bên phải.
- Các câu lệnh và khối lệnh cùng cấp thì viết trên cùng một cột (thẳng cột).
- Điểm đầu và điểm cuối của một khối lệnh cũng viết thẳng cột.

1.6. Một vài chương trình C đơn giản

Ví dụ 1.6: Chương trình hiện lên màn hình hai dòng chữ

```
TURBO C HÂN HẠNH
LÀM QUEN VỚI BẠN
```

```
# include “stdio.h”
# include “conio.h”
void  main()
{
        printf(“TURBO C HAN HANH\n”);
```

```
        /* In dòng chữ TURBO C HAN HANH rồi xuống dòng*/
        printf("LAM QUEN VOI BAN");
        /* In dòng chữ LAM QUEN VOI BAN ở đầu dòng tiếp theo */
        getch();      // tạm dừng máy để xem kết quả
}
```

Ví dụ 1.7: Chương trình tính chu vi và diện tích của hình chữ nhật với hai cạnh được nhập từ bàn phím

```
# include "stdio.h"
# include "conio.h"
void  main()
{
        int dai,rong;                   //khai báo các biến
        float cv, dt;
        printf("Nhap chieu dai, rong:"); //In ra màn hình chuỗi trong cặp dấu
                                         // nháy kép
        scanf("%d %d",&dai,&rong);   //Nhập số liệu cho các biến
        cv = (float)(dai+rong)*2;                 // Tính chu vi
        dt = (float)(dai*rong);                   // Tính diện tích
        printf("\nchu vi = %d",cv);         // In ra màn hình giá trị chu vi
        printf("\ndien tich = %d",dt);      // In ra màn hình diện tích
        getch(); //Dừng màn hình để xem kết quả
}
```

1.7. Vận hành chương trình trên máy

Phần việc còn lại sau khi đã có chương trình là thực hiện chương trình trên máy tính để nhận được kết quả cuối cùng. Đây là phần việc ít đòi hỏi suy nghĩ và sáng tạo hơn so với công việc lập trình. Quá trình vận hành trên máy tính bao gồm các bước cơ bản sau:

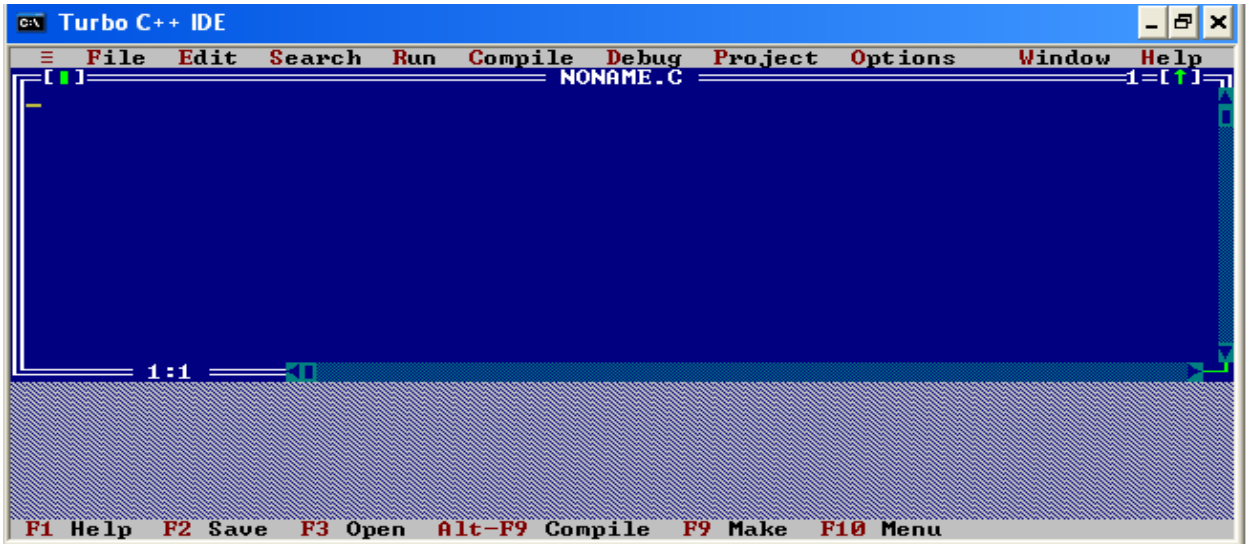
- Tạo tệp chương trình gốc đuôi C (soạn thảo chương trình)

- Dịch chương trình (tạo tệp chương trình thực hiện đuôi EXE)
- Chạy chương trình.

Giả sử TURBO C đã được cài đặt trong thư mục C:\TC.

1.7.1. Tạo tệp chương trình gốc

Chọn chương trình TC.EXE nằm trong đường dẫn C:\TC\BIN sẽ nhận được menu chính trên màn hình với nội dung sau:



Hình 1.1. Màn hình soạn thảo của Turbo C

Bây giờ ta cần dùng chức năng New nằm trong Menu File bằng cách đưa con trỏ đến hộp sáng New và bấm Enter để tạo một tệp chương trình mới. Khi đó chức năng New được thực hiện và máy sẵn sàng nhận các dòng chương trình đưa vào. Ta sẽ sử dụng bàn phím để nạp chương trình.

Chương trình mặc dù đã đưa lên màn hình nhưng nó vẫn chưa được ghi lên đĩa. Muốn vậy sau khi soạn thảo chương trình ta dùng chức năng Save trên Menu File sau đó gõ tên tệp gốc, giả sử gõ tên tệp là THU.C rồi ấn phím enter. Đến đây quá trình soạn thảo được hoàn thành và tệp có tên là THU.C đã được lưu trên đĩa.

1.7.2. Dịch chương trình

Để dịch chương trình ta dùng chức năng Compile trên menu Compile hoặc nhấn tổ hợp phím Alt + F9, chức năng này có tác dụng dịch chương trình gốc và liên kết với các hàm thư viện nhằm tạo ra tệp chương trình thực hiện có đuôi là EXE. Các sai sót (error) và cảnh báo (warning) nếu có sẽ được chỉ ra. Nếu có error hoặc warning thì ta cần trở lại bước 1 để sửa chữa chương trình. Trong trường hợp

trái lại, ta nhận được một chương trình đúng, nó được ghi lên đĩa dưới dạng một tệp có tên là: THU.EXE. Đến đây quá trình dịch xem như được hoàn thành.

Tóm lại sau khi thực hiện hai bước vừa nêu trên ta nhận được tệp chương trình gốc THU.C và tệp chương trình thực hiện THU.EXE, cả hai đều được đặt trong thư mục C:\TC và máy đang sẵn sàng làm việc với các tệp trong thư mục này.

1.7.3. Chạy chương trình

Để khởi động một tệp chương trình thực hiện ta sử dụng tên của nó. Chẳng hạn, để khởi động chương trình THU.EXE ta bấm THU và enter (trong môi trường DOS) hoặc ta có thể chạy trong môi trường Turbo C bằng cách bấm đồng thời hai phím Ctrl + F9. Khi đó chương trình bắt đầu làm việc. Nếu nhập đủ các số liệu cần thiết chương trình sẽ tính toán và đưa ra kết quả cuối cùng trên máy tính

Bài tập cuối chương

Câu 1. Viết chương trình tính x lũy thừa y với các giá trị x và y được nhập vào từ bàn phím.

Câu 2. Viết chương trình tính tích của bốn số thực được nhập vào từ bàn phím.

Câu 3. Viết chương trình nhập vào ba số a, b, c là ba cạnh của một tam giác. Sau đó in ra màn hình chu vi và diện tích của tam giác trên.

CHƯƠNG 2: KIỂU DỮ LIỆU, HẰNG, BIẾN, MẢNG VÀ BIỂU THỨC

Trong chương này, tác giả sẽ giới thiệu về các kiểu dữ liệu đơn giản thông dụng như nguyên, thực, ký tự và kiểu dữ liệu có cấu trúc như mảng của ngôn ngữ C. Đưa ra các khái niệm về hằng, biến và cách sử dụng chúng trong chương trình. Tiếp theo là các định nghĩa về biểu thức và các phép toán để phục vụ công việc tính toán trong khi lập trình.

2.1. Kiểu dữ liệu

2.1.1. Các kiểu dữ liệu đơn giản thông dụng

2.1.1.1. Kiểu ký tự (char)

Một giá trị kiểu char chiếm một byte bộ nhớ và biểu diễn được một ký tự thông qua bảng mã American Stardard Code Information Interchange (ASCII – phụ lục 1)

Trong ngôn ngữ C có hai kiểu char là:

Kiểu	Phạm vi biểu diễn	Kích thước	Số ký tự
[signed] char	-128 – 127	1byte	256
Unsigned char	0 – 255	1byte	256

Như vậy có hai kiểu char là signed char và unsigned char. Kiểu thứ nhất biểu diễn một số nguyên từ -128 đến 127 còn kiểu thứ hai có giá trị từ 0 đến 255.

Chú ý: Giá trị kiểu char là một số nguyên, do đó có thể thực hiện các phép tính số học trên đó.

Ví dụ 2.1: Đoạn chương trình sau minh họa sự khác nhau giữa hai kiểu char trên:

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

Khi đó nếu ta dùng câu lệnh in ra màn hình printf(“ch1 = %d, ch2 = %d”, ch1, ch2); thì sẽ cho kết quả như sau ch1 = -56, ch2 = 200 nhưng cả ch1 và ch2 đều biểu diễn chung một ký tự có mã trong bảng mã ASCII là 200, tức là nếu ta dùng

câu lệnh in printf("ch1 = %c, ch2 = %c", ch1, ch2) thì sẽ ra màn hình hai giá trị ký tự của ch1 và ch2 là giống nhau.

Giải thích: Số 200 được biểu diễn ở hệ nhị phân là 11001000. Vì ch2 là unsigned char (số không dấu) nên tất cả các bit trong dãy nhị phân 11001000 đều được dùng để tính giá trị vậy giá trị của ch2 là 200, còn ch1 là kiểu signed char (có dấu) nên bit đầu tiên trong dãy nhị phân là bit dấu, ở đây là bit 1 nên đó là số âm. Như vậy dãy nhị phân trên là biểu diễn số âm ở dạng mã bù. Để chuyển về biểu diễn ở dạng mã ngược ta lấy dãy nhị phân trên trừ cho 1 được kết quả là 11000111. Từ mã ngược trên ta đảo ngược các bit sẽ được biểu diễn ở dạng mã thuận như sau 10111000. Giá trị của dãy nhị phân 10111000 là -56.

2.1.1.2. Kiểu nguyên.

Trong C bao gồm các kiểu số nguyên sau:

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

Chú ý: Các kiểu ký tự cũng được xem là một dạng của số nguyên.

2.1.1.3. Kiểu thực

Trong C có các kiểu số thực sau:

Kiểu	Phạm vi biểu diễn	Kích thước
float	3.4E-38 – 3.4E+38	4 byte
double	1.7E-308 – 1.7E+308	8 byte
long double	3.4E-4932 – 3.4E+4932	10 byte

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

2.1.2. Kiểu dữ liệu liệt kê

Cú pháp:

enum [tên_kiểu] {tên 1,…, tên n} [danh sách biến];

- Các tên 1, …, tên n được coi như là các số nguyên liên tiếp được tính từ 0. Nói cụ thể hơn tên 1 = 0, tên 2 = 1,…

- 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.

Ví dụ 2.2: Xét chương trình sau

```
#include<stdio.h>
void  main()
{
        enum   ngay {cn,t2,t3,t4,t5,t6,t7}  ngay1, ngay2;
        int  j = 2000;
        ngay1 = t4;
        ngay2 = j;
        printf("ngay1=%d, ngay2 =%d",ngay1,ngay2);
        getch();
}
```

sẽ in ra màn hình kết quả sau: ngay1 = 3, ngay2 = 2000.

2.2. Hằng, biến

2.2.1. Hằng

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán. Tên hằng được khai báo theo cú pháp sau:

const kiểu dữ liệu tên_hằng = giá trị;

Ví dụ 2.3: Cho khai báo hằng sau: const int max = 100; khi đó tất cả các tên max xuất hiện trong chương trình đều có giá trị là 100.

Chú ý: Trong chương trình không được dùng bất kỳ câu lệnh nào để làm thay đổi giá trị của các hằng.

2.2.1.1. Hằng số thực

Hằng số thực được viết theo hai cách:

- Theo dạng thập phân: Số thực gồm phần nguyên, dấu chấm thập phân và phần thập phân. Ví dụ 2.4: các hằng số thực 214.55, 12.0

- Theo dạng mũ: Số thực được tách gồm hai phần là định trị và bậc. Phần định trị là một số nguyên hoặc số thực được viết dưới dạng thập phân, phần bậc là một số nguyên biểu diễn số mũ của cơ số 10, hai phần này cách nhau bởi ký tự e hoặc E. Ví dụ 2.5: 0.12E3 biểu diễn giá trị $0.12*10^3 = 120.0$, 1.23e-2 biểu diễn giá trị $1.23 * 10^{-2} = 0.0123$.

2.2.1.2. Hằng số nguyên

Là một số nguyên có giá trị trong khoảng từ -32768 đến 32767. Ví dụ 2.6:

-45, 543 là các hằng số nguyên. Chú ý phân biệt 12 và 12.0. Số 12 là hằng số nguyên còn 12.0 là hằng số thực.

Một hằng số nguyên có thể được biểu diễn như sau:

- Theo cách thông thường: 2006,12,…tức là số được biểu diễn ở hệ cơ số 10
- Theo cơ số 8: Thêm số 0 vào đằng trước số 017|8 = 15|10
- Theo cơ số 16: Thêm 0x ở đầu số: 0x 1A|16 = 26|10
- Kiểu char cũng được xem như một hằng nguyên:

Ví dụ 2.7:

printf(“so=%d”,’A’+5);sẽ cho ra kết quả là so = 70

printf(“chu=%c”,’A’+2);sẽ cho ra kết quả là chu =‘C’

2.2.1.3. Hằng ký tự

Là một ký tự được viết trong hai dấu nháy đơn.

Ví dụ 2.8: ‘a’ là một hằng ký tự. Giá trị của ‘a’ chính là mã của ký tự ‘a’ trong bảng mã ASCII. Như vậy giá trị của ‘a’ là 97.

Đối với một vài hằng ký tự đặc biệt, ta cần sử dụng cách viết sau (thêm dấu \ vào trước ký tự đó):

Cách viết	**Ký tự**
‘\’’	‘
‘\”’	“
‘\\’	\
‘\n’	chuyển dòng
‘\0’	\0 (null)

‘\t’	tab
‘\b’	backspace
‘\r’	CR (về đầu dòng)
‘\f’	sang trang

2.2.1.4. Hằng xâu ký tự

Là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép.

Ví dụ 2.9: hằng xâu ký tự “Ha noi”, “” (xâu không gồm ký tự nào hay còn gọi là xâu rỗng). Xâu ký tự được lưu trữ trong máy dưới dạng một mảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự ‘\0’ vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

Chú ý: Phân biệt giữa hằng ký tự và hằng xâu ký tự: ‘A’ là hằng ký tự khác với “A” là hằng xâu ký tự.

2.2.2. Biến

2.2.2.1. Khai báo biến

Biến là một đại lượng có giá trị thay đổi trong quá trình tính toán. Mọi biến trong chương trình cần phải được khai báo trước khi sử dụng. Khai báo biến theo cú pháp sau:

kiểu dữ liệu danh sách tên biến;

Chú ý: Các biến có cùng kiểu dữ liệu sẽ được ghi trên một dòng trong cùng một khai báo, khi đó các biến được viết cách nhau bởi dấu phẩy.

Ví dụ 2.10: Ví dụ về khai báo biến trong chương trình

```
void main()
{
        int a,b,c;              //khai báo 3 biến a, b, c kiểu int
        float x,y;              //khai báo 2 biến x, y kiểu float
        char  ch1, ch2;         //khai báo 2 biến ch1, ch2 kiểu char
      …….
}
```

Chú ý: Biến kiểu int chỉ nhận được các giá trị kiểu int. Các biến khác cũng có ý nghĩa tương tự.

2.2.2.2. Khởi đầu giá trị cho biến

Ta có thể vừa khai báo biến vừa gán cho biến đó một giá trị đầu tiên bằng cách sau:

kiểu dữ liệu tên biến = giá trị đầu tiên;

Vì biến là đại lượng có giá trị thay đổi nên trong chương trình ta có thể dùng các câu lệnh gán để thay đổi giá trị của biến.

2.2.2.3. Lấy địa chỉ của biến

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Để nhận địa chỉ biến ta dùng phép toán: &tên_biến

2.2.3. Kiểu mảng

Mỗi biến chỉ có thể chứa một giá trị. Để chứa một dãy số ta có thể dùng nhiều biến nhưng cách này không tiện lợi. Trong trường hợp này ta sẽ sử dụng kiểu dữ liệu là mảng. Mảng là một tập hợp nhiều phần tử có cùng một kiểu giá trị và chung một tên và được phân biệt với nhau bởi chỉ số mảng. Mỗi phần tử mảng có vai trò như một biến và chứa được một giá trị. Có bao nhiêu kiểu biến thì có bấy nhiêu kiểu mảng, như vậy sẽ có mảng số nguyên, mảng số thực, mảng ký tự. Mảng cần được khai báo để xác định rõ:

- Kiểu mảng (int, float,…)
- Tên mảng
- Số chiều và kích thước mỗi chiều.

2.2.3.1. Khai báo mảng

Mảng được khai báo theo cú pháp sau:

kiểu dữ liệu mảng_1, mảng_2,…, mảng_n;

trong đó mỗi mảng_i (với i=1..n) được khai báo theo cú pháp sau: tên_mảng[k1][k2]…[km] với k1, k2,…, km là kích thước của mỗi chiều.

Ví dụ 2.11: Các khai báo

int a[10];

float b[3][3];

Sẽ xác định hai mảng với ý nghĩa:

- Mảng thứ nhất tên là a có kiểu là int, số chiều là một, kích thước là 10. Mảng gồm 10 phần tử được đánh số như sau: a[0], a[1],.., a[9] và mỗi phần tử chứa được một giá trị kiểu int.

- Mảng thứ hai tên là b, kiểu là float, số chiều là 2, kích thước của các chiều đều là 3. Mảng có 9 phần tử, như sau:

b[0][0] b[0][1] b[0][2]

b[1][0] b[1][1] b[1][2]

b[2][0] b[2][1] b[2][2]

trong đó mỗi phần tử b[i][j] chứa được một giá trị kiểu float

Chú ý:

- Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ.

- Khi cần khai báo một biến chuỗi ký tự ta khai báo mảng một chiều mà mỗi phần tử của mảng có kiểu là kiểu ký tự (char) như ví dụ sau: char chuoi[15];

2.2.3.2. Khởi tạo giá trị mảng

Muốn khởi đầu giá trị của mảng nào đó, ta viết như sau:

- Với mảng một chiều:

tên_mảng[n] = {gt1, gt2,…, gtk}; với k<=n

tên_mảng[] = {gt1, gt2,…, gtk}; số n có thể vắng mặt

- Với mảng hai chiều

tên_mảng[n][m] = {{gt11,…, gt1m},

…… ,

{gtn1},…, gtnm}};

tên_mảng[][m] = {{gt11,…, gt1m},

…… ,

{gtn1},…, gtnm}}; số n có thể vắng mặt.

2.2.3.3. Truy nhập tới các phần tử mảng

Mỗi phần tử cụ thể của mảng được xác định nhờ các chỉ số của nó. Chỉ số của mảng phải có giá trị kiểu số nguyên và không vượt quá kích thước của chiều tương ứng, số chỉ số phải bằng số chiều của mảng. Trong C chỉ số mỗi chiều của mảng luôn được tính bắt đầu từ 0.

Để truy nhập đến các phần tử của mảng n chiều ta viết:

tên_mảng[chỉ số 1][chỉ số 2]…[chỉ số n]

Chú ý:

- Chỉ số có thể là số thực, khi đó sẽ lấy phần nguyên của số thực làm chỉ số mảng.

- Khi chỉ số vượt ra ngoài kích thước mảng, máy không báo lỗi, nhưng sẽ cho ra một giá trị ngẫu nhiên, khi đó máy sẽ truy nhập đến một vùng nhớ bên ngoài mảng và có thể làm rối loạn chương trình.

2.2.3.4. Lấy địa chỉ phần tử mảng

Mỗi một phần tử mảng được xem như là một biến do đó cách lấy địa chỉ mảng cũng giống như lấy địa chỉ của biến. Ví dụ &a[i], &b[i][j].

Tên mảng biểu thị địa chỉ đầu của mảng. Vậy với mảng một chiều tên là a thì a chính là địa chỉ của phần tử đầu tiên của mảng, tức là a = &a[0], với mảng hai chiều tên là b thì b chính là địa chỉ đầu tiên của mảng hay ta viết b = &b[0][0].

2.2.4. Toán tử sizeof

Toán tử sizeof được dùng để tính kích thước (theo byte) của một kiểu dữ liệu cũng như một đối tượng. Nó được viết như sau:

sizeof(kiểu dữ liệu) hoặc

sizeof(đối tượng dữ liệu)

Kiểu dữ liệu có thể là kiểu đơn giản thông dụng như int, float và kiểu được định nghĩa trong chương trình (enum, struct, union – chương 6)

Đối tượng dữ liệu bao gồm biến, mảng, cấu trúc,…(tên của vùng nhớ dữ liệu).

Ví dụ 2.12. Cho các khai báo sau

```
int   x, a[10];       float   y;
```

ta có:

sizeof(x) = sizeof(int) = 2 byte.

sizeof(a) = 10*sizeof(int) = 10 * 2 = 20 byte

sizeof(y) = sizeof(float) = 4 byte

2.3. Định nghĩa kiểu dữ liệu mới

Từ khóa typedef dùng để đặt tên cho một kiểu dữ liệu mới. Tên kiểu dữ liệu mới này sẽ được dùng để khai báo dữ liệu của biến sau này.

Cú pháp: typedef kiểu dữ liệu tên kiểu dữ liệu mới;

Như vậy ta thấy khai báo kiểu dữ liệu mới cũng giống như khai báo biến chỉ khác là có thêm từ khóa typedef ở đầu và không có chức năng khởi đầu giá trị.

Ví dụ 2.13: Khai báo typedef int nguyen; sẽ đặt tên kiểu int là nguyen. Sau đó có thể dùng nguyen (cũng như int) để khai báo các biến nguyên, các mảng nguyên như sau:

```
nguyen   x,y,a[10]; /*Khai báo hai biến x, y và mảng a có kiểu là
                          nguyen đã được định nghĩa*/.
```

Chú ý: Nếu ta muốn định nghĩa một kiểu dữ liệu mảng để khai báo các biến a, b có kiểu dữ liệu là kiểu mảng đã định nghĩa thì làm như sau:

```
typedef   int   mang[10];
mang    a, b;
```

Ở đây ta thực hiện như khai báo mảng thông thường nhưng có thêm từ khóa typedef vào phía trước, khi đó tên mang chính là tên kiểu dữ liệu mới và có thể lấy tên mang để khai báo cho các biến về sau.

2.4. Biểu thức

2.4.1. Khái niệm biểu thức

Biểu thức có thể là một toán hạng hoặc là sự kết hợp giữa các phép toán và các toán hạng, dùng để diễn đạt một công thức toán học nào đó. Toán hạng có thể là hằng, biến, phần tử mảng, các hàm,…Biểu thức được phân loại theo kiểu giá trị: nguyên, thực và biểu thức logic. Khi viết biểu thức nên dùng các dấu ngoặc tròn để thể hiện đúng trình tự tính toán trong biểu thức.

Biểu thức thường được dùng trong: vế phải của câu lệnh gán, làm tham số thực sự của hàm, làm chỉ số và trong các câu lệnh if, switch, for, while, do…while.

2.4.2. Các phép toán

2.4.2.1. Các phép toán số học:

Gồm các phép toán một ngôi và phép toán hai ngôi.

Phép toán hai ngôi được cho như trong bảng sau:

Bảng 0.1. Các phép toán hai ngôi

Phép toán	**Ý nghĩa**	**Ví dụ**
+	Cộng	a + b
-	Trừ	a – b
*	Nhân	a * b
/	Chia	a / b
%	Lấy phần dư	a % b

Chú ý:

1. Có phép toán một ngôi – , ví dụ 2.14: -(a+b) nhưng không có phép toán + .

2. Phép chia hai số nguyên sẽ cho kết quả bỏ đi phần thập phân, ví dụ 2.15: 11/3 = 3

3. Phép % cho phần dư của phép chia nguyên, không áp dụng cho các giá trị kiểu float và double.

2.4.2.2. Các phép toán với bit.

Các phép toán với bit được cho như trong bảng sau:

Bảng 0.2. Các phép toán với bit

Phép toán	**Ý nghĩa**	**Giải thích**
&	Phép và (and)	1&1=1 0&0=0 1&0=0 0&1=0
|	Phép hoặc (or)	1|1=1 0|0=0 1|0=1 0|1=1
^	Phép hoặc loại trừ (xor)	1^1=0 0^0=0 1^0=1 0^1=1
<<	Dịch trái	a << n = $a*2^n$
>>	Dịch phải	a >> 4 = $a/2^n$
~	Lấy phần bù	~0=1 ~1=0

2.4.2.3. Các phép toán so sánh và logic.

Các phép toán so sánh và logic cho ra kết quả hoặc giá trị đúng (1) hoặc giá trị sai (0). Các phép toán so sánh cho trong bảng sau:

Bảng 0.3. Các phép toán so sánh

Phép toán	**Ý nghĩa**	**Ví dụ**
>	So sánh lớn hơn	3>7 có giá trị 0
<	So sánh bé hơn	'a'<'b' có giá trị 1
>=	So sánh lớn hơn hoặc bằng	2>=2 có giá trị 1
<=	So sánh bé hơn hoặc bằng	12>=20 có giá trị 0
==	So sánh bằng nhau	3==0 có giá trị 0
!=	So sánh khác nhau	5!=9 có giá trị 1

Các phép toán logic được cho trong bảng sau:

Bảng 0.4. Các phép toán logic

Phép toán	**Ý nghĩa**	**Giải thích**		
!	Phép phủ định một ngôi	A	!a	
		Khác 0	0	
		Bằng 0	1	
&&	Phép và (and)	a	b	a&&b
		Khác 0	Khác 0	1
		Khác 0	Bằng 0	0
		Bằng 0	Khác 0	0
		Bằng 0	Bằng 0	0
||	Phép hoặc (or)	a	b	a||b
		Khác 0	Khác 0	1
		Khác 0	Bằng 0	1
		Bằng 0	Khác 0	1
		Bằng 0	Bằng 0	0

Chú ý: a và b có thể nguyên hay thực.

2.4.2.4. Phép tăng, giảm

Phép tăng: toán tử ++ sẽ cộng thêm một đơn vị vào toán hạng của nó.

Phép giảm: toán tử -- sẽ trừ đi một đơn vị vào toán hạng của nó.

Chú ý: Dấu ++, -- có thể đứng trước hoặc sau toán hạng, vậy có thể viết: ++n, n++, -- n, n--;

Sự khác nhau giữa hai cách viết:

- Trong phép n++ thì n được tăng giá trị lên một đơn vị sau khi giá trị của nó đã được sử dụng.

- Còn phép ++n thì n được tăng giá trị lên một đơn vị trước khi nó được sử dụng.

Ví dụ 2.16: Nếu n có giá trị bằng 5 thì khi thực hiện câu lệnh m = n++; thì máy sẽ thực hiện câu lệnh gán m = n trước tức là m = 5, sau đó mới thực hiện đến phép tăng n = n + 1, tức là giá trị n= 6.

Nếu có câu lệnh gán m = ++n; thì máy sẽ thực hiện phép tăng n = n + 1 trước, tức là giá trị của n = 6, sau đó mới thực hiện phép gán m = n, như vậy trong trường hợp này giá trị của m = 6.

2.4.3. Chuyển đổi kiểu giá trị

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

- Khi biểu thức gồm các toán hạng khác kiểu: khi hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán. Kết quả thu được là một giá trị có kiểu cao hơn. Chẳng hạn giữa hai toán hạng có kiểu int với long thì toán hạng kiểu int sẽ được chuyển thành kiểu long, giữa hai toán hạng có kiểu int và float thì toán hạng kiểu int sẽ được chuyển thành kiểu float,…Ví dụ 2.17: 1.5*(11/3) = 4.5

- Khi gán một giá trị kiểu này cho một biến/phần tử mảng kiểu kia. Khi đó giá trị của vế phải được chuyển sang kiểu của vế trái, đó là kiểu của kết quả. Kiểu int có thể chuyển thành kiểu float, kiểu float có thể chuyển thành kiểu int do loại bỏ phần thập phân,…Ví dụ 2.18: nếu n là biến nguyên thì sau khi thực hiện câu lệnh n = 15.6; thì n sẽ có giá trị là 15.

Ngoài ra ta có thể chuyển từ một kiểu giá trị sang một kiểu bất kỳ mà ta muốn bằng phép ép kiểu như sau: (type) (biểu thức). Khi đó kiểu của biểu thức sẽ được đổi thành kiểu type theo các trường hợp nêu ở trên.

Ví dụ 2.19: (int)a sẽ cho một giá trị kiểu int. Nếu a là biến kiểu float thì ở đây có sự chuyển đổi từ float sang int. Chú ý bản thân a vẫn có kiểu float chỉ (int)a là có kiểu int.

Chú ý:

- Muốn có giá trị chính xác trong phép chia hai biến nguyên cần dùng phép ép kiểu: (float)(a/b)

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

2.4.4. Câu lệnh gán và Biểu thức gán

2.4.4.1. Câu lệnh gán

Được dùng để gán giá trị cho một biến hay một phần tử mảng, câu lệnh gán có cú pháp như sau:

biến/phần tử mảng = biểu thức;

Ví dụ 2.20: Xét các câu lệnh

```
n = 2;        //gán giá trị 2 cho biến n
ch = 'a';     //gán giá trị cho biến ch là ký tự 'a'
```

Chú ý: các câu lệnh gán có dạng như x = x +1; trong đó vế trái được lặp lại ở vế phải thì ta có thể viết gọn hơn như sau: x +=1; Tương tự với câu lệnh x = x – 1; ta cũng có thể viết là x – = 1;

2.4.4.2. Biểu thức gán

Là biểu thức có dạng v = e trong đó v là một biến hay phần tử mảng, e là một biểu thức. Giá trị của biểu thức gán là giá trị của e, kiểu của biểu thức gán là kiểu của v. Nếu đặt dấu chấm phẩy vào biểu thức gán thì ta được câu lệnh gán.

Ví dụ 2.21:

- Khi viết a = b = 5; thì điều đó có nghĩa là gán giá trị của biểu thức b = 5 cho biến a. Kết quả là b = 5 và a = 5.
- Tương tự câu lệnh a = b = c = d = 8; sẽ gán giá trị 8 cho a, b, c, d.
- Câu lệnh x = (y = 2) * (z = 6); thì sẽ cho kết quả y = 2, z = 6 và z = 12.

2.4.5. Biểu thức điều kiện

Là biểu thức có dạng:

biểu thức 1 ? biểu thức 2: biểu thức 3

Giá trị của biểu thức điều kiện bằng giá trị của biểu thức 2 nếu biểu thức 1 khác 0 (đúng) hoặc bằng giá trị của biểu thức 3 nếu biểu thức 1 bằng 0 (sai). Kiểu của biểu thức điều kiện là kiểu cao nhất trong các kiểu của biểu thức 2 và biểu thức 3.

Ví dụ 2.22: Cho biểu thức điều kiện sau: (a>b) ? a : b Biểu thức cho kết quả là giá trị cực đại của hai số a và b. Tức là, nếu biểu thức a>b có giá trị là đúng thì số cực đại sẽ là a và ngược lại, nếu biểu thức a>b có giá trị là sai thì số cực đại sẽ là b.

2.4.6. Thứ tự ưu tiên trong phép toán

Các phép toán có độ ưu tiên khác nhau, thứ tự ưu tiên của các phép toán được cho trong bảng sau:

Bảng 0.5. Thứ tự ưu tiên của các phép toán

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

Chú ý:

- Đối với các phép toán có cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại tùy thuộc vào "Trình tự kết hợp".
- Khi ta viết một biểu thức, nếu không nhớ đến thứ tự ưu tiên của các phép toán thì ta nên sử dụng các cặp dấu ngoặc tròn ().
- Dưới đây là một vài chỉ dẫn về các phép toán lạ trong bảng trên

+ Trên dòng 1

[] dùng để biểu diễn phần tử mảng, ví dụ a[3]

. dùng để biểu diễn thành phần của cấu trúc, ví dụ sv.ten

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

+ Trên dòng 2

* dùng để truy nhập đến vùng nhớ thông qua con trỏ, ví dụ *a

& phép toán lấy địa chỉ, ví dụ &a

+ Trên dòng 15

, là toán tử dùng để ngăn cách các biểu thức trong một dãy biểu thức.

Bài tập cuối chương

Câu 1. Hãy tìm các biểu thức đúng trong các biểu thức sau đây

1. (i = j)++
2. i + j++
3. ++(i + j)++ i +++j

Câu 2. Xác định giá trị của các biểu thức sau:

a. 5.6 + 2.7 + 20/6 + 8.0

b. 5.6 + 2.7 + 20/6.0 + 8.0

Câu 3. Sử dụng các hàm toán học trong C để biểu diễn các công thức toán học sau:

a.

$$S = \sqrt{\frac{x^2 + y^2}{x^2 - y^2}} + \sqrt{\frac{x^2 - y^2}{x^2 + y}}$$

b.

$$P = 1 + e^{\frac{1}{x}} - \sqrt{y^3}$$

Câu 4. Biểu diễn các công thức toán học cho các biểu thức sau:

a. sqt(sqrt(a) + sqrt(b))

b. sqrt(sqr(x))/sqrt(sqr(y))*ln(exp(x)/exp(y))

CHƯƠNG 3: VÀO - RA DỮ LIỆU

Trong các ví dụ của các chương trước, chúng ta đã nhiều lần sử dụng các hàm printf, scanf để nhập và xuất dữ liệu. Chương này sẽ giới thiệu chi tiết hơn về các hàm này. Ngoài ra, còn bổ sung nhiều hàm mới khác như: nhập xuất chuỗi, nhập xuất ký tự, xóa màn hình,…

3.1. Hàm printf

Hàm printf nằm trong tệp stdio.h, có khả năng chuyển dạng, tạo khuôn và đưa giá trị các đối của hàm ra màn hình.

3.1.1. Khai báo hàm

Cú pháp:

```
int     printf(const  char *cdk [, danh sách các đối]);
```

Chú ý:

- int là kiểu kết quả trả về của hàm.

- Đối cdk là hằng con trỏ kiểu char chứa địa chỉ của chuỗi điều khiển. Chuỗi điều khiển sẽ được giải thích rõ hơn trong phần tiếp theo.

3.1.2. Chuỗi điều khiển

Chuỗi điều khiển là một dãy các ký tự, có thể chứa ba loại ký tự sau:

- Các loại ký tự điều khiển gồm:

Ký tự	**ý nghĩa**
\n	xuống dòng
\f	sang trang
\t	tab
\b	backspace

- Các đặc tả chuyển dạng và tạo khuôn (được gọi tắt là đặc tả) cho các đối tương ứng.

- Các ký tự không phải là đặc tả cũng không phải là các ký tự điều khiển gọi là ký tự hiển thị (được đưa ra màn hình). Ngoài các ký tự hiển thị thông thường, có các ký tự đặc biệt sau:

Ký tự	Ý nghĩa
\’	in ra dấu ‘
\”	in ra dấu ”
\\	in ra dấu \

3.1.3. Đặc tả

Đặc tả có thể được viết theo cú pháp sau:

%[-][n][.m]ký tự chuyển dạng

trong đó:

- % và ký tự chuyển dạng là những thành phần bắt buộc phải có, Các thành phần khác có thể vắng mặt.

- Số n dùng để quy định số vị trí để in ra giá trị của đối.

+ Nếu không có tham số n này hoặc có mà nhỏ hơn số vị trí cần thiết để in ra giá trị đối tương ứng thì khi đó n được hiểu là bằng đúng số vị trí cần thiết để in ra giá trị của đối..

+ Nếu có n và n lớn hơn số vị trí cần thiết để in ra giá trị đối tương ứng thì khi đó phụ thuộc vào có hay không có dấu trừ ‘ – ‘. Trường hợp không có ‘– ‘ thì giá trị của đối sẽ được in ra màn hình dồn về bên phải tạo ra các khoảng trống ở bên trái. Trường hợp có dấu ‘ – ‘ thì ngược lại, giá trị của đối sẽ được in ra màn hình dồn về bên trái và tạo ra các khoảng trống ở bên phải.

Ví dụ 3.1: Xét đoạn chương trình sau

```
int   n = 5, m = 6;
printf(“n = %-4d, m = %5d”);
```

sẽ cho kết quả: n = 5_ _ _ , m = _ _ _ _6: (Chú ý: ‘_’ được hiểu là khoảng trắng) Máy sẽ dành 4 vị trí để in giá trị của n. Số n được in trên một ví trí ở bên trái và tạo ra 3 khoảng trống ở bên phải. Tiếp đó máy dành 5 vị trí để in giá trị của m. Số m được in ra trên một vị trí ở bên phải và tạo ra 4 khoảng trống ở bên trái

- Số m: Chỉ có ý nghĩa khi đối tương ứng của nó thuộc kiểu số thực hoặc kiểu chuỗi.

+ Nếu đối là kiểu số thực thì m dùng để quy định số các chữ số thập phân. Trong trường hợp nếu không có m thì phần thập phân được mặc định là 6.

+ Nếu đối là biến kiểu chuỗi và m nhỏ hơn độ dài của chuỗi thì chỉ m ký tự đầu tiên của chuỗi được in ra màn hình. Nếu không có m hoặc m lớn hơn độ dài của chuỗi thì cả chuỗi sẽ được in ra màn hình.

Ví dụ 3.2: Xét đoạn chương trình sau

```
float   x = 7.5;
char  lop[ ] = “happy new year”;
printf(“x = %7.2f, lop = %10.4s”, x, lop);
```

sẽ cho kết quả như sau: x =_ _ _7.50, lop = _ _ _ _ _ _happ

- Với biến x: máy dành 7 vị trí để in giá trị của biến x, giá trị của x chiếm 4 vị trí ở bên phải trong đó có 2 vị trí dành cho phần thập phân, và 3 vị trí dư thừa được dồn về bên trái.

- Với biến lop: Máy dành 10 vị trí để in giá trị của biến lop, trong đó 4 ký tự đầu tiên của lop được in ra trên 4 vị trí ở bên phải, 6 vị trí còn lại bên trái là các khoảng trắng.

3.1.4. Ký tự chuyển dạng

Là một hoặc một dãy ký hiệu, nó xác định quy tắc chuyển dạng và dạng in ra của đối tương ứng. Như vậy sẽ có trường hợp cùng một giá trị sẽ được in ra theo các dạng khác nhau khi sử dụng các ký tự chuyển dạng khác nhau. Tuy nhiên ta không được sử dụng các ký tự chuyển dạng một cách tùy tiện mà phải tuân theo các quy tắc định sẵn: với mỗi kiểu giá trị tương ứng với một hoặc một vài ký tự chuyển dạng nhất định, như trong bảng sau:

Bảng 0.1. Danh sách các ký tự chuyển dạng của hàm printf

Ký tự chuyển dạng	**Kiểu của đối**	**Cách chuyển dạng**
c	char	Đối được coi là một ký tự
d/i	int	Đối được coi là số nguyên hệ 10 có dấu (nếu giá trị âm)
ld/li	long	Đối được coi là số nguyên hệ 10 có dấu (nếu giá trị âm)
u	int	Đối được coi là số nguyên hệ 10 không dấu
o	int	Đối được coi là số hệ 8 không dấu (không có số 0 đứng trước)
lo	long	Đối được coi là số hệ 8 không dấu (không có số 0 đứng trước)
x	int	Đối được coi là số hệ 16 không dấu (không có số 0x đứng trước)
lx	long	Đối được coi là số hệ 8 không dấu (không có số 0x đứng trước)
f	float hay double	Đối là số thực được in ra dưới dạng dấu phẩy tĩnh. In ra cả các số 0 vô nghĩa.
e	float hay double	Đối là số thực được in ra dưới dạng dấu phẩy động. In ra cả các số 0 vô nghĩa.
g	float hay double	Dùng %f hay %e tùy thuộc dạng nào ngắn hơn. Không in ra các số 0 vô nghĩa.
s	Xâu ký tự	Đối là một xâu ký tự

3.1.5. Danh sách đối

Các đối trong danh sách đối được ngăn cách nhau bởi dấy phẩy. Đối là một biểu thức như: một hằng, một biến, một phần tử mảng, một lời gọi hàm. Giá trị của đối sẽ được chuyển dạng và in ra theo cách của đặc tả tương ứng.

Chú ý:

- Mỗi một đặc tả phải có một đối tương ứng . Khi một đặc tả không tìm thấy một đối tương ứng, hoặc kiểu dữ liệu của đối tương ứng không phù hợp với đặc tả thì máy sẽ bị nhầm và đưa ra kết quả vô nghĩa.

- Số đối có thể nhiều hơn số đặc tả, nhưng các đối không có đặc tả ở phía sau sẽ không được in ra.

- Tại vị trí của n trong đặc tả ta có thể thay bằng dấu *, khi đó trước đối tương ứng ta phải đưa vào một số nguyên để thay cho giá trị của tham số n . Mục đích của tham số này là ta có thể căn chỉnh động đối với hàm printf.

Ví dụ 3.3: Cho đoạn chương trình:

```
int  n = 6;
float x = 7.5;
printf("x = %*.2f ",n,x);
```

sẽ cho kết quả như sau: x = _ _7.50

3.1.6. Giá trị của hàm printf

Khi hàm printf thực hiện thành công thì sẽ trả về một giá trị đúng bằng số ký tự được in ra kể cả ký tự điều khiển.

Ví dụ 3.4: Cho đoạn chương trình

```
int  i;
i = printf("Happy\nNew year");
printf("i = %d",i);
```

Hàm sẽ cho ra kết quả là: i = 14

3.2. Hàm scanf

Hàm scanf được khai báo trong tệp stdio.h. Có tác dụng đọc thông tin từ thiết bị vào chuẩn (bàn phím), chuyển dịch chúng (thành số nguyên, số thực,…) và lưu trữ vào bộ nhớ theo các địa chỉ xác định.

3.2.1. Khai báo hàm

Cú pháp:

int scanf(const char *cdk [,danh sách các đối]);

trong đó cdk là hằng con trỏ kiểu char chứa địa chỉ của chuỗi điều khiển. Danh sách đối sẽ được định nghĩa ở phần sau.

3.2.2. Danh sách đối

Các đối được ngăn cách nhau bởi dấu phẩy. Mỗi đối là một con trỏ chứa địa chỉ của một vùng nhớ (địa chỉ biến, mảng,…) dùng để lưu trữ một giá trị đọc vào từ bàn phím. Để lấy địa chỉ của biến hay của phần tử mảng ta viết: &biến/phần tử mảng.

3.2.3. Dòng vào và trường vào

- Dòng vào là một dãy ký tự liên tiếp nhau (bao gồm cả khoảng trắng) trên thiết bị vào (khoảng trắng được hiểu là dấu cách, dấu tab hoặc ký tự xuống dòng)

- Trường vào là một dãy ký tự khác khoảng trắng. Như vậy khoảng trắng dùng để ngăn cách các trường vào trong một dòng vào.

Ví dụ 3.5: Trên dòng vào: 123_abc\t25\n gồm ba trường là 123, abc và 25

3.2.4. Chuỗi điều khiển

Gồm các ký tự đặc tả chuyển dạng, mỗi đặc tả thường có một đối tương ứng, gần giống chuỗi điều khiển của hàm printf, dùng để định danh các giá trị nhập vào từ bàn phím rồi gán cho các biến. Mỗi đặc tả có dạng sau:

<%>[*][n]<ký tự chuyển dạng> trong đó:

- Nếu có ký tự * thì trường vào vẫn được dò đọc bình thường nhưng giá trị của nó sẽ bị bỏ qua (không được đưa vào bộ nhớ) và đặc tả này được xem là không có đối tương ứng.

- n là một dãy số xác định chiều dài cực đại của trường vào. Ý nghĩa của tham số n như sau:

+ Nếu n vắng mặt hoặc lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ nội dung của trường vào sẽ được đọc, sau đó định dạng và gán cho đối tương ứng (nhưng phải không có dấu *).

+ Nếu n nhỏ hơn độ dài của trường vào tương ứng thì chỉ n ký tự đầu của trường vào được dò đọc, sau đó định dạng và gán cho đối tương ứng, phần còn lại của trường vào sẽ được xem xét bởi các đặc tả và đối tương ứng tiếp theo.

Ví dụ 3.6: Xét đoạn chương trình

int m,n; scanf("%*d %2d %d", &n, &m);

Nếu từ bàn phím, nhập 125 _ 678910 enter thì số 125 vẫn được dò đọc nhưng giá trị của nó sẽ bị bỏ qua và không có đối tương ứng. Còn lại hai ký tự tiếp theo là 67 được dò đọc, định dạng và gán cho biến n. Cuối cùng các ký tự 8910 được dò đoc, định dạng và gán cho biến m.

3.2.5. Ký tự chuyển dạng

Xác định cách thức dò đọc các ký hiệu trên dòng vào trước khi gán nó cho các địa chỉ tương ứng. Cách dò đọc thứ nhất là dò đọc theo trường vào, khi đó các khoảng trắng bị bỏ qua. Cách dò đọc thứ hai là dò đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét bình đẳng như các ký tự khác. Cách thứ hai chỉ xảy ra khi ta sử dụng một trong ba ký tự chuyển dạng sau đây: c, […], [^…]

Dưới đây là danh sách các ký tự chuyển dạng:

Bảng 0.2. Danh sách ký tự chuyển dạng và kiểu đối của hàm scanf

Ký tự chuyển dạng	**Kiểu đối**
c	char
d	int
ld	long
o	int
lo	long
x	int
lx	long
f/e	float
lf/le	double
s	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 hai dấu [ ]. Đối tương ứng là con trỏ kiểu char
[^dãy ký tự]	Các ký tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một ký tự thuộc tập các ký tự đặt trong hai dấu [ ]. Đối tương ứng là con trỏ kiểu char

Ví dụ 3.7: Cho đoạn chương trình

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

Với dòng vào nhập như sau:

35_13243 _xyz 545

thì sẽ gán kết quả lần lượt cho các biến như sau: a= 35, xâu ch = "_13243_", xâu ck = "xyz", b = 545.

Chú ý:

- Khi ta dùng hàm scanf để nhập dữ liệu cho một biến nào đó thì máy sẽ dừng lại để đợi nhập dữ liệu. Nhưng trong chương trình giả sử có rất nhiều biến mà ta cần phải nhập dữ liệu cho tất cả các biến đó. Như vậy, khi máy dừng lại để đợi nhập dữ liệu nếu không có sự chỉ dẫn cụ thể thì người dùng sẽ khó phân biệt được thứ tự nhập các biến như thế nào. Do đó trước mỗi câu lệnh scanf ta nên sử dụng câu lệnh printf để đưa ra hướng dẫn cho người dùng công việc cần phải làm, như ví dụ sau:

```
printf("Nhap gia tri cua bien nguyen a = ");
scanf("%d", &a);
printf("\nNhap gia tri cua bien thuc b = ");
scanf("%f", &b);
printf("\nNhap ký tu ch = ");
scanf("%c", &ch);
```

- Giả sử có khai báo chuỗi s như sau: char string[10]; thì trong chương trình khi ta muốn nhập dữ liệu cho chuỗi s ta viết scanf("Nhap chuoi string = %s",

string); Ở đây ta không ghi là &string vì string là một mảng mà tên mảng là một con trỏ và nó chứa địa chỉ đầu tiên của mảng. Do đó trong trường hợp này ta chỉ ghi là string.

- Giả sử có khai báo mảng a như sau int a[10]; Trong chương trình khi ta muốn nhập dữ liệu cho một phần tử nào đó của mảng a thì ta viết như sau:

printf("Nhap gia tri cua phan tu a[%d] = ", i);

scanf("%d", &a[i]);

3.2.6. Giá trị của hàm scanf

Hàm cho ra kết quả là một số nguyên bằng các trường vào đã được dò đọc và được lưu trữ vào các đối tương ứng.

Chú ý: Số đối, số đặc tả, số trường vào nên bằng nhau và phù hợp về kiểu dữ liệu vì nếu không sẽ khó kiểm soát và các đối có thể nhận các giá trị vô nghĩa.

3.3. Dòng vào stdin và các hàm scanf, gets, getchar

stdin: là dòng vào chuẩn (bàn phím). Các hàm scanf, gets, getchar đều nhận dữ liệu từ stdin. Có các trường hợp xảy ra:

- Nếu trên stdin có đủ dữ liệu, thì các hàm sẽ nhận một phần dữ liệu mà chúng yêu cầu. Phần dữ liệu còn lại vẫn ở trên stdin.

- Khi trên stdin không đủ dữ liệu theo yêu cầu của các hàm, thì máy tạm dừng để người dùng đưa thêm dữ liệu từ bàn phím lên stdin (cho đến khi bấm phím enter).

- Nếu nhập thừa cho các đối thì phần thừa sẽ được đưa vào stdin để phục vụ cho các hàm gets, getchar, scanf tiếp sau.

- Phím enter sẽ sinh ra ký tự '\n' và cũng được đưa vào stdin, do đó nó sẽ làm trôi hàm gets, getchar và scanf ở phía sau.

3.3.1. Hàm gets

Hàm gets được khai báo trong tệp stdio.h.

Cú pháp: char *gets(char *s);

trong đó đối s là con trỏ kiểu char trỏ tới vùng nhớ chứa dãy ký tự nhận được.

Công dụng của hàm: Nhận dẫy ký tự từ stdin cho đến khi gặp ký tự '\n'. Ký tự '\n' bị loại khỏi stdin nhưng không được đặt vào chuỗi. Chuỗi được bổ sung

thêm ký tự '\0' và đặt vào vùng nhớ do s trỏ tới. Hàm trả về địa chỉ của chuỗi nhận được.

3.3.2. Hàm getchar

Hàm getchar được khai báo trong tệp stdio.h.

Cú pháp: int getchar(void)

Công dụng của hàm: nhận một ký tự từ stdin. Hàm trả về mã ký tự nhận được.

Chú ý:

- Xét câu lệnh ch = getchar(); Nếu bấm phím A enter thì ch = 'A', ký tự \n vẫn ở lại trên stdin và nó sẽ làm trôi các hàm getchar hoặc hàm gets sau đó. Nếu chỉ bấm enter thì ch = '\n' và '\n' bị loại khỏi stdin.

- Hàm scanf cũng để lại ký tự '\n' trên stdin. Ký tự này sẽ làm trôi hàm gets và getchar sau đó. Để các hàm này hoạt động đúng thì phải khử ký tự '\n' trong hàm scanf bằng cách thêm đặc tả %*c vào cuối chuỗi điều khiển như sau:

Ví dụ 3.8: Cho đoạn chương trình

```
char ht[25];   int  t;
printf("Nhap  t=");
scanf("%d %*c",&t);  //khử '\n' trong dòng vào
printf("Nhap ht = ");
gets(ht);
```

- Để nhận dữ liệu đúng đắn bằng cách nhập vào từ bàn phím thì trước khi thực hiện scanf, gets, getchar ta nên làm sạch stdin bằng hàm sau: fflush(stdin); Dùng hàm này sẽ tránh được mọi hậu quả của các thao tác nhập số liệu trước đó. Với ví dụ 3.8 trên ta có thể viết lại

Ví dụ 3.9: Viết lại ví dụ 3.8

```
char ht[25];   int  t;
printf("Nhap  t=");
scanf("%d ",&t);
printf("Nhap ht = ");
fflush(stdin);    //Xóa stdin
```

gets(ht);

3.4. Các hàm xuất ký tự puts, putchar

Các hàm này được khai báo trong tệp stdio.h. Các hàm printf, puts, putchar đều có tác dụng đưa dữ liệu lên dòng ra chuẩn stdout (màn hình).

3.4.1. Hàm puts

Hàm puts được khai báo trong tệp stdio.h

Cú pháp: int puts(const char *s); trong đó s là con trỏ kiểu char trỏ tới vùng nhớ chứa chuỗi ký tự cần xuất ra stdout.

Công dụng của hàm: đưa chuỗi s và đưa thêm ký tự '\n' lên stdout. Khi thành công, hàm trả về ký tự cuối cùng được đưa ra (chính là \n). Ngược lại khi có lỗi hàm trả về EOF.

Ví dụ 3.10: câu lệnh puts("\nHa noi"); sẽ đưa dòng chữ Ha noi lên một dòng mới của màn hình sau đó chuyển con trỏ xuống đầu dòng tiếp theo.

3.4.2. Hàm putchar

Hàm putchar được khai báo trong tệp stdio.h

Cú pháp: int putchar(int ch); trong đó ch là mã của một ký tự.

Công dụng của hàm: Đưa ký tự có mã là ch lên stdout. Khi thành công hàm trả về mã của ký tự được đưa ra. Có lỗi thì hàm cho EOF.

3.5. Các hàm vào ra trên màn hình, bàn phím

Các hàm trong mục này được khai báo trong tệp conio.h

3.5.1. Hàm getch

Cú pháp: int getch(void);

Công dụng của hàm: Hàm nhận một ký tự từ bộ đệm bàn phím, không cho hiện ký tự lên màn hình. Nếu có sẵn ký tự trong bộ đệm bàn phím, thì hàm nhận một ký tự trong đó, nếu bộ đệm rỗng thì máy tạm dừng. Khi gõ một ký tự thì hàm nhận ngay được ký tự đó (không cần bấm thêm phím enter như các hàm nhập từ stdin). Hàm trả về ký tự nhận được

3.5.2. Hàm getche

Cú pháp: int getche(void);

Công dụng của hàm: Hàm nhận một ký tự từ bộ đệm bàn phím và cho hiện ký tự vừa nhập lên màn hình. Điều này là khác với hàm getch() phía trên.

3.5.3. Hàm putch

Cú pháp: int putch(int ch); trong đó đối ch là mã ký tự cần hiển thị.

Công dụng: Đưa ký tự ch lên cửa sổ văn bản màn hình. Ký tự sẽ được hiển thị theo màu xác định bởi hàm textcolor. Trong khi đó hàm putchar luôn hiển thị ký tự theo màu mặc định là màu trắng. Hàm trả về ký tự được hiển thị

3.5.4. Hàm kbhit

Cú pháp: int kbhit(void);

Công dụng của hàm: Kiểm tra bộ đệm bàn phím. Hàm cho giá trị khác 0 nếu bộ đệm bàn phím khác rỗng, có giá trị 0 nếu ngược lại.

Chú ý:

1. Làm sao để phân biệt được khi bấm một phím thì ký tự tương ứng có thể bị gửi vào stdin hay vào bộ đệm bàn phím? Ta phân biệt như sau:

- Nếu gõ phím thì máy dừng chờ nhập dữ liệu trong các hàm scanf, gets và getchar thì ký tự được gửi vào stdin

- Gõ phím trong các trường hợp khác thì ký tự được gửi vào bộ đệm bàn phím.

2. Không nên dùng hàm scanf để nhập dữ liệu cho mảng ký tự mà nên dùng hàm gets vì nếu dùng hàm scanf có thể sẽ bị mất dữ liệu như trong ví dụ sau:

Ví dụ 3.11.

```
char  s[20];
void main()
{
   printf("Nhap chuoi s = ");
   scanf("%s", s); // Cách 2 là gets(s);
   printf("\nChuoi s = %s", s);
}
```

Nếu ta nhập từ bàn phím chuỗi ha noi thì kết quả in ra màn hình là Chuoi s = ha. Còn nếu dùng hàm gets trong trường hợp này thì sẽ cho kết quả là s = ha_noi (_

ở đây là dấu cách). Vì hàm scanf dò đọc dữ liệu theo từng trường vào sau đó định dạng và gán giá trị cho các đối tương ứng do đó chuỗi noi sẽ được để dành cho các hàm scanf, gets, getchar phía sau.

3. Không nên dùng hàm scanf để nhập dữ liệu cho ký tự mà dùng hàm getch, getchar.

3.5.5. Hàm xóa màn hình

Cú pháp: void clrscr();

Công dụng của hàm: Hàm dùng để xóa sạch màn hình và đưa con trỏ về vị trí đầu màn hình.

3.5.6. Hàm di chuyển con trỏ

Cú pháp: void gotoxy(x,y);

Công dụng của hàm: Di chuyển con trỏ màn hình đến vị trí có tọa độ (x,y) trong đó x là số hiệu cột nhận giá trị từ 1 đến 80, y là dòng có giá trị từ 1 đến 25.

Bài tập cuối chương

Câu 1. Viết chương trình

- Đầu tiên in ra các dòng chữ

  NGON NGU LAP TRINH

- Nếu không bấm phím hoặc bấm một phím khác C và P thì dòng chữ trên tiếp tục hiện ra.

  Nếu bấm C thì chương trình in ra dòng chữ TURBO C

  Nếu bấm P thì chương trình in ra dòng chữ TURBO PASCAL

- Bấm tiếp phím bất kỳ thì chương trình kết thúc.

Câu 2. Viết chương trình tạo hình sau:

```
* * * * * * * * * * *
* * * * *   * * * * *
* * * *       * * * *
* * *           * * *
* *               * *
* * * * * * * * * * *
```

Câu 3. Viết chương trình nhập phần thực x và phần ảo y của một số phức, sau đó in ra theo mẫu: (x, y) với các yêu cầu:

- Số in ra có hai chữ số sau dấu chấm thập phân
- Kết quả in ra tạo thành một dãy ký tự liên tiếp nhau (không có khoảng trống)

Câu 4. Viết chương trình nhập vào ba số thỏa mãn điều kiện là ba cạnh của một tam giác. Sau đó tính chu vi và diện tích của tam giác và in các giá trị vừa tính được ra màn hình.

Câu 5. Viết chương trình nhập tính giá trị chu vi và diện tích của một hình tròn sau đó in các giá trị vừa tính được ra màn hình trong đó dữ liệu bán kính của hình tròn được nhập vào từ bàn phím.

Câu 6. Viết chương trình nhập vào từ bàn phím giá trị GDP của năm 2011 và tốc độ tăng trưởng kinh tế bình quân (tính theo %). Sau đó tính giá trị GDP của các năm 2012, 2013 và in các giá trị vừa tính được ra màn hình.

CHƯƠNG 4: CÁC CÂU LỆNH ĐIỀU KHIỂN

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện theo thứ tự mà chúng được viết ra. Các toán tử điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể đang thực hiện câu lệnh này lại nhảy tới thực hiện một câu lệnh khác ở vị trí bất kỳ. Khi đó máy sẽ thực hiện công việc một cách linh hoạt hơn và hiệu quả hơn. Có thể chia các toán tử điều khiển thành ba nhóm chính:

+ Nhảy không điều kiện

+ rẽ nhánh

+ tổ chức chu trình

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

4.1. Các câu lệnh rẽ nhánh

4.1.1. Câu lệnh if

Câu lệnh if cho phép lựa chọn một trong hai nhánh tùy thuộc giá trị bằng hay khác 0 của một biểu thức, có hai cách viết câu lệnh if như sau:

- Dạng 1:
```
if (biểu thức)
    <Việc 1>
```
- Dạng 2:
```
if (biểu thức)
    <Việc 1>
else
    <Việc 2>
```

Sự hoạt động của câu lệnh if:

- Dạng 1: Trước tiên máy sẽ xác định giá trị của biểu thức. Nếu biểu thức đúng (có giá trị khác 0) máy sẽ thực hiện <Việc 1> và sau đó thực hiện các câu lệnh ở phía sau. Nếu biểu thức sai thì máy bỏ qua mà chuyển đến thực hiện các câu lệnh ở phía sau.

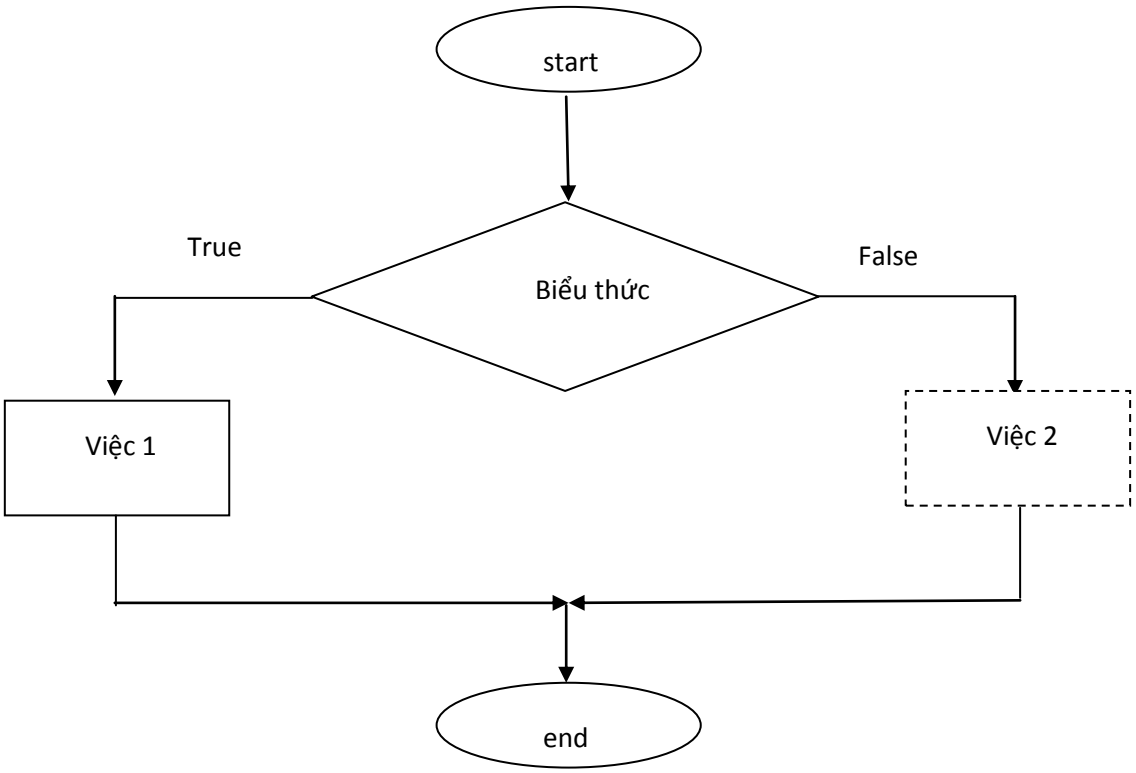
- Dạng 2: Trước tiên máy sẽ xác định giá trị của biểu thức. Nếu biểu thức đúng (có giá trị khác 0) máy sẽ thực hiện <Việc 1>và sau đó thực hiện các câu lệnh viết sau <Việc 2>. Nếu biểu thức sai (có giá trị bằng 0) thì máy thực hiện <Việc 2> và sau đó thực hiện các câu lệnh ở phía sau <Việc 2>.

Nhận xét:

- Trong cú pháp của cả hai dạng của câu lệnh if nếu <Việc 1>, <Việc 2> bao gồm nhiều câu lệnh thì các câu lệnh đó phải được viết trong một khối lệnh. Chú ý này cũng áp dụng với các câu lệnh điều khiển khác.

- Câu lệnh if dạng 1 thực ra là câu lệnh if dạng 2 thu gọn với <Việc 2> là rỗng.

Sự hoạt động của câu lệnh if có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.1. Sơ đồ khối của câu lệnh if dạng 1 (nhánh else nối tắt) và lệnh if dạng 2.

Ví dụ 4.1: Viết chương trình sử dụng câu lệnh if để tìm số lớn nhất của hai số a và b sau đó in giá trị lớn nhất đó ra màn hình

- Dạng 1:

```
#include “stdio.h”
void  main()
{
        float  a, b, max;
        printf(“Nhap hai so a va b =”);
```

```
        scanf("%f%f",&a,&b);
        max = a;
        if  (b > max)       max = b;
        printf("Gia tri lon nhat cua hai so la %8.2f",max);
        getch();
}
```

- Dạng 2

```
#include "stdio.h"
void  main()
{
        float  a, b, max;
        printf("Nhap hai so a va b =");
        scanf("%f%f",&a,&b);
        if  (a>b)    max = a;
        else         max = b;
        printf("Gia tri lon nhat cua hai so la %8.2f",max);
        getch();
}
```

Chú ý:

- Trong <Việc 1> và <Việc 2> có thể chứa các câu lệnh if khác, khi đó ta có trường hợp các câu lệnh if lồng nhau. Trong trường hợp này nên sử dụng các dấu đóng mở khối {} để không gây ra sự hiểu nhầm. Khi số từ khóa if bằng số từ khóa else thì ta dễ dàng xác định được từng cặp if – else tương ứng. Trong trường hợp số từ khóa else ít hơn số từ khóa if thì else sẽ được gắn với if không có else gần nhất trước đó.

Ví dụ 4.2: Xét đoạn chương trình sau:

```
if  (n>0)
        if  (a>b)    z = a;
        else         z = b;
```

Khi đó else sẽ đi với if bên dưới. Ở đây nếu sử dụng khối lệnh, ta sẽ có:

```
if  (n>0)
{
        if  (a>b)    z = a;
        else         z = b;
}
```

Câu lệnh else if: Khi muốn thực hiện một trong n quyết định ta có thể sử dụng câu lệnh if dưới dạng sau:

```
 if  (biểu thức 1)
        <Việc 1>
 else  if  (biểu thức 2)
        <Việc 2>
 …….
else  if (biểu thức n – 1)
        <Việc n – 1>
else    <Việc n>
```

Sự hoạt động của câu lệnh else if:

- Chỉ một trong n việc được thực hiện
- Nếu biểu thức thứ i là biểu thức đầu tiên khác 0 (i = 1, 2, …, n – 1) thì <Việc i> được thực hiện.
- Nếu cả n – 1 biểu thức đều có giá trị bằng 0 thì <Việc n> được thực hiện

Ví dụ 4.3. Giả sử để theo dõi trình độ của cán bộ ta dùng mã sau

Mã	Trình độ
1	Sơ cấp
2	Trung cấp
3	Đại học
4	Cao học
5	Tiến sĩ

Viết chương trình nhập vào mã sau đó in ra màn hình trình độ học vấn của một cán bộ bất kỳ.

Chương trình được viết như sau:

```
#include”stdio.h”
void  main()
{
        int   ma;
        printf(“Nhap ma:”);
        scanf(“%d”, &ma);
        if (ma == 1)
                printf(“Trinh do so cap”);
        else if ( ma == 2)
                printf(“Trinh do trung cap”);
        else if (ma == 3)
                printf(“Trinh do dai hoc”);
        else if (ma == 4)
                printf(“Trinh do cao hoc”):
        else if (ma == 5)
                printf(“Trinh do tien si”);
        else printf(“Nhap ma sai”);
        getch();
}
```

4.1.2. Câu lệnh switch

Nếu như câu lệnh if chỉ cho phép thực hiện một trong hai nhánh công việc (trừ trường hợp với câu lệnh else if có thể thực hiện một trong n nhánh công việc)thì câu lệnh switch lại có nhiều sự lựa chọn công việc hơn. Câu lệnh switch cho phép căn cứ vào giá trị của một biểu thức nguyên để chọn một trong nhiều cách nhảy. Cú pháp của câu lệnh switch như sau:

switch (biểu thức nguyên)

{

case n_1: <Việc 1>

case n_2: <Việc 2>

……

case n_k: <Việc k>

[default: <Việc k +1>]

}

trong đó n_i là các biểu thức hằng nguyên. Các n_i có giá trị khác nhau. Đoạn chương trình nằm giữa hai dấu {} được gọi là thân của câu lệnh switch. Do đó, dấu } ở đây được xem là dấu kết thúc thân của switch.

Sự hoạt động của câu lệnh switch: Phụ thuộc vào giá trị của biểu thức viết trong các dấu ngoặc tròn.

1. Khi giá trị này bằng n_i thì máy sẽ nhảy tới vị trí đặt nhãn n_i và thực hiện <Việc i>

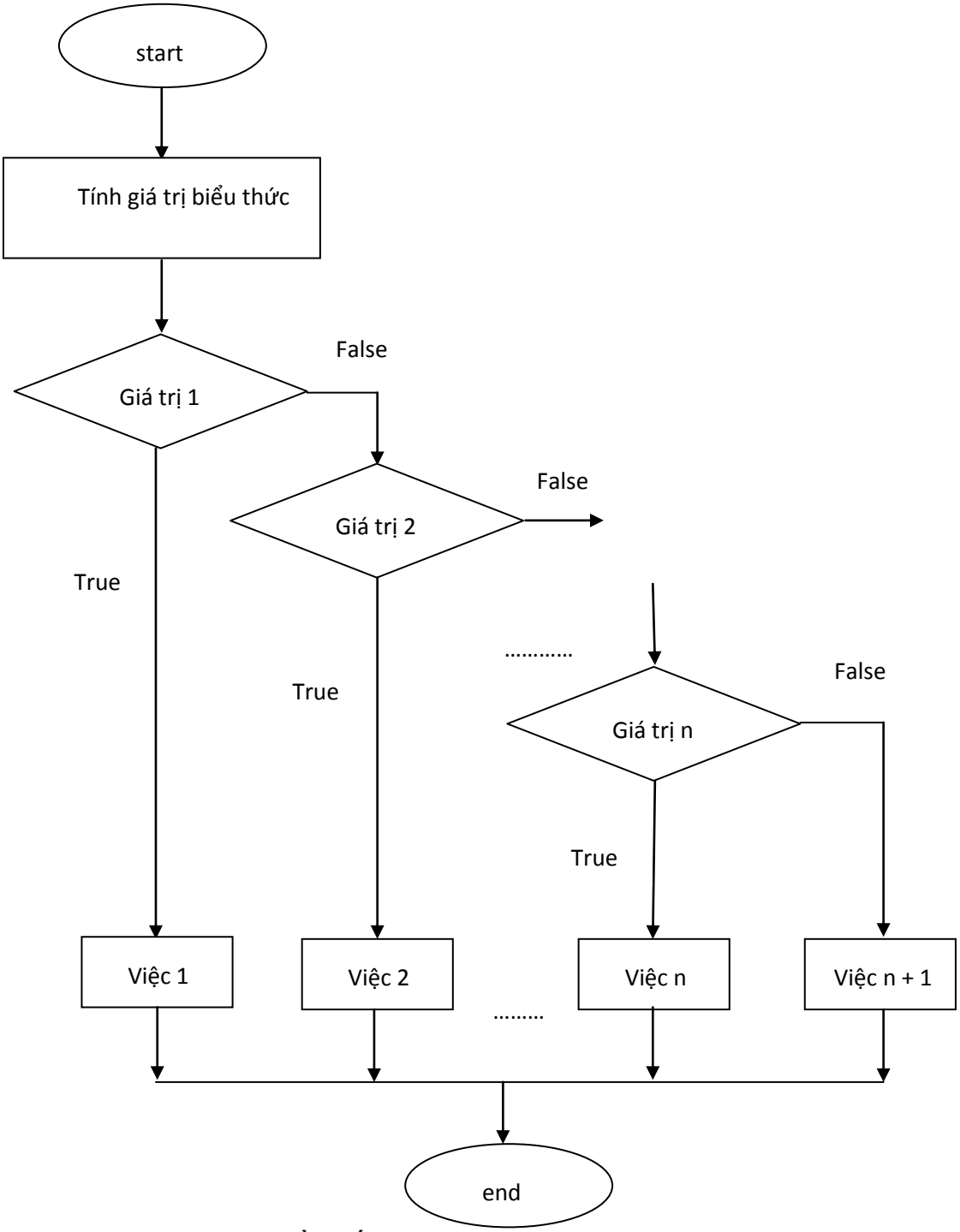
2. Khi giá trị của biểu thức khác tất cả các n_i thì

+ nếu có default thì máy sẽ nhảy tới vị trí đặt nhãn là default và thực hiện <Việc k+1>.

+ nếu không có default thì máy ra khỏi câu lệnh switch.

Ra khỏi câu lệnh switch: Máy sẽ ra khỏi câu lệnh switch khi nó gặp một câu lệnh break hoặc gặp dấu kết thúc thân của switch. Cũng có thể sử dụng câu lệnh goto trong thân của câu lệnh switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Sự hoạt động của câu lệnh switch có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.2. Sơ đồ khối của câu lệnh switch…case

Chú ý:

- Trong cú pháp của câu lệnh switch, nếu <Việc i> gồm nhiều câu lệnh thì các câu lệnh này có thể được đặt trong một khối lệnh hoặc cũng có thể không đặt trong

một khối lệnh. Đây là điểm khác biệt của câu lệnh switch với các câu lệnh điều khiển khác.

- Các <Việc i> có thể bỏ trống bằng cách không viết gì hoặc chỉ viết dấu chấm phẩy ";". Cách bỏ trống này được dùng khi có nhiều hằng cùng thực hiện một <Việc i>, khi đó các hằng ở phía trên ta bỏ trống hoặc viết dấu ;, còn hằng cuối cùng của nhóm hằng này ta viết các câu lệnh trong phần <Việc i> cần thực hiện.

- Nếu các câu lệnh trong <việc i> được kết thúc bằng câu lệnh break thì câu lệnh switch sẽ thực hiện chỉ một trong các <việc i> này (Với i = 1..k).

Ví dụ 4.4: Chương trình nhập vào điểm của một sinh viên (Giả sử điểm nhập vào là số nguyên thỏa mãn điều kiện 0 <= điểm <= 10) sau đó in ra màn hình xếp loại theo tiêu chuẩn sau:

9, 10 xếp loại giỏi

7, 8 xếp loại khá

5, 6 xếp loại trung bình

0..4 xếp loại yếu

Chương trình được viết như sau:

```
#include "stdio.h"
#include"conio.h"
void  main()
{
        int  diem;
        printf("Nhap diem:");
        scanf("%d",&diem);
        switch  (diem)
        {
                case 10:
                case 9:  printf("\nLoai gioi");    break;
                case 8:
```

```
                case 7:  printf("\nLoai kha");    break;
                case 6:
                case  5: printf("\nLoai trung binh");   break;
                case 4:
                case 3:
                case 2:
                case 1:
                case 0:  printf("\nLoai yeu");
        }
  getch();
}
```

Chú ý:

- Nếu kết thúc mỗi <Việc i> trong ví dụ trên ta không dùng câu lệnh break thì giả sử điểm nhập vào là 7 thì kết quả sẽ được in ra màn hình là

  Loại khá

  Loại trung bình

  Loại yếu

- Chương trình có thể được viết ngắn gọn hơn bằng cách sử dụng từ khóa default như sau:

```
#include "stdio.h"
#include"conio.h"
void  main()
{
        int  diem;
        printf("Nhap diem:");
        scanf("%d",&diem);
        switch  (điem)
        {
```

```
                case 10:
                case 9:  printf("Loai gioi");    break;
                case 8:
                case 7:  printf("Loai kha");    break;
                case 6:
                case  5: printf("loai trung binh");   break;
                default:  printf("loai yeu");
        }
  getch();
}
```

Nhận xét: Cả hai câu lệnh if và switch thuộc các câu lệnh rẽ nhánh, dùng để lựa chọn một trong các nhánh công việc. Với các bài toán thực hiện bằng câu lệnh switch thì có thể chuyển sang viết bằng câu lệnh if nhưng điều ngược lại chưa chắc đã đúng vì trong câu lệnh switch biểu thức yêu cầu phải có giá trị nguyên. Còn trong câu lệnh if giá trị của biểu thức có thể nguyên và cũng có thể là thực.

4.1.3. Nhãn và Câu lệnh goto.

Nhãn: Tên nhãn được đặt như quy tắc đặt tên biến, khi viết tên nhãn thường kèm theo dấu hai chấm ':' đứng sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình.

Ví dụ 4.5: trong câu lệnh sau:

```
tiếp_tục: n = 8;
```

thì tiếp tục là nhãn của câu lệnh gán n = 8;.

Cú pháp của câu lệnh goto như sau:

```
goto  tên_nhãn;
```

Sự hoạt động của câu lệnh goto: Khi gặp câu lệnh này thì máy sẽ nhảy tới câu lệnh có tên_nhãn viết sau từ khóa goto.

Chú ý:

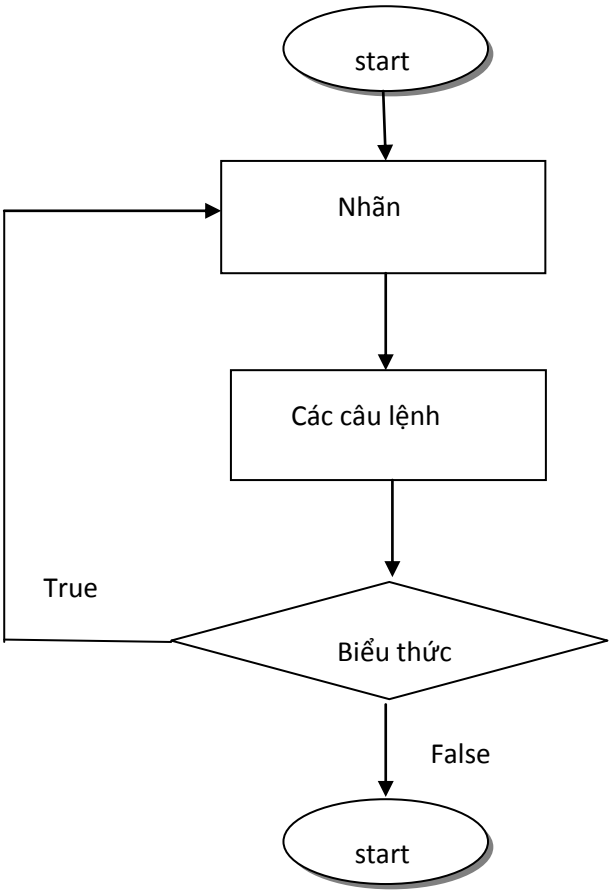
- Câu lệnh goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân của một hàm. Nó không thể dùng để nhảy từ thân hàm này sang thân hàm khác.

- Không cho phép dùng câu lệnh goto để nhảy từ ngoài vào trong một khối lệnh, nhưng lại cho phép việc nhảy từ trong ra ngoài khối lệnh.

- Sử dụng câu lệnh goto kết hợp với câu lệnh rẽ nhánh if ta sẽ tạo được một chu trình lặp, trong đó thân chu trình gồm n câu lệnh sẽ được lặp đi lặp lại chừng nào mà biểu thức trong if vẫn còn đúng như sau:

```
lặp_lại :
        câu lệnh 1
        câu lệnh 2
        ….
        câu lệnh n
if  (biểu thức)  goto   lăp_lại;
```

Sự hoạt động của câu lệnh goto kết hợp với câu lệnh if có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.3. Sơ đồ khối của câu lệnh goto kết hợp với câu lệnh if

Ví dụ 4.6: Trở lại ví dụ 4.3, trong đó ta sử dụng câu lệnh goto kết hợp với câu lệnh if để kiểm tra điều kiện của điểm nhập vào phải nằm trong khoảng từ 0 đến 10.

```
#include “stdio.h”
#include”conio.h”
void  main()
{
        int  diem;
        lap_lai:
                printf(“Nhap diem:”);
                scanf(“%d”,&diem);
        if  ((diem <0)||(diem >10))   goto  lap_lai;
        if (ma == 1)
            printf(“Trinh do so cap”);
        else if ( ma == 2)
            printf(“Trinh do trung cap”);
        else if (ma == 3)
            printf(“Trinh do dai hoc”);
        else if (ma == 4)
            printf(“Trinh do cao hoc”):
        else if (ma == 5)
            printf(“Trinh do tien si”);
        else printf(“Nhap ma sai”);
        getch();
}
```

4.2. Các câu lệnh có cấu trúc lặp

4.2.1. Câu lệnh for

Dùng để tạo một chu trình lặp (vòng lặp) có số bước lặp xác định. Cú pháp của câu lệnh như sau:

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

<Công việc>//thân vòng lặp for

Câu lệnh for gồm ba biểu thức và thân for. Thân vòng lặp for là một câu lệnh hoặc một khối lệnh. Bất kỳ biểu thức nào trong ba biểu thức nói trên đều có quyền vắng mặt nhưng không được thiếu dấu chấm phẩy.

Biểu thức 1 thường là một câu lệnh gán để tạo giá trị ban đầu cho biến điều khiển.

Biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục vòng lặp.

Biểu thức 3 là một câu lệnh gán dùng để thay đổi giá trị của biến điều khiển.

Sự hoạt động của câu lệnh for:

1. Xác định biểu thức 1

2. Xác định biểu thức 2

3. Tùy thuộc vào giá trị đúng sai của biểu thức 2 mà máy sẽ lựa chọn một trong hai nhánh:

- nếu biểu thức 2 có giá trị bằng 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân vòng lặp for

- nếu biểu thức 2 có giá trị khác 0 (đúng) máy sẽ thực hiện <Công việc> trong thân vòng lặp for. Khi gặp dấu ngoặc đóng } cuối cùng trong thân vòng lặp for hoặc gặp câu lệnh continue máy sẽ chuyển tới bước 4.

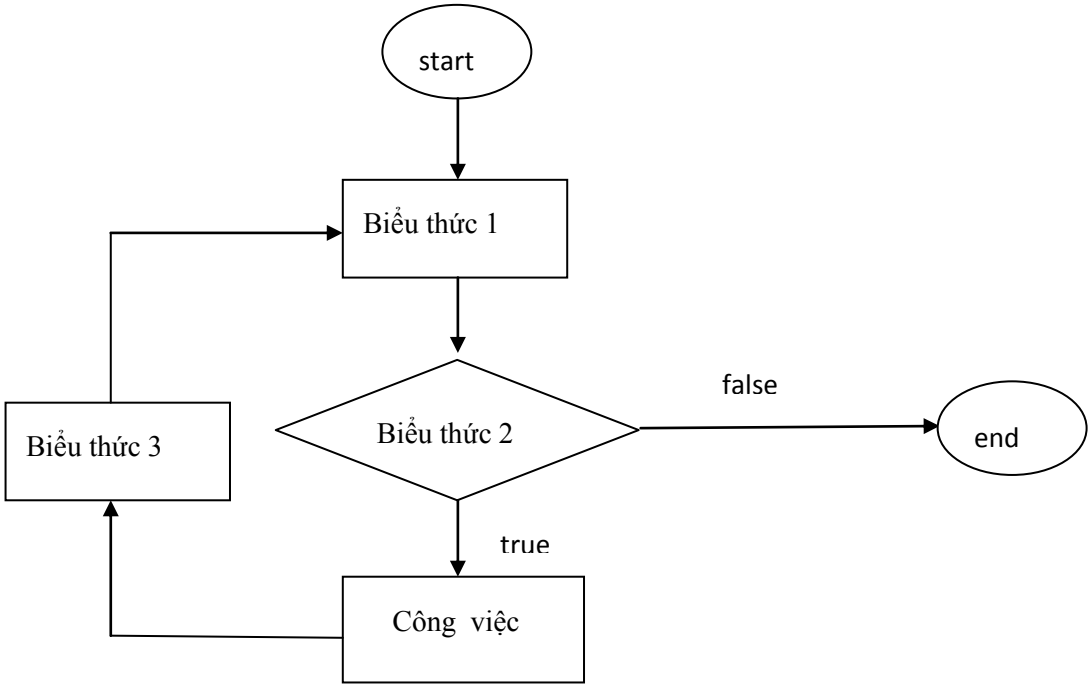
4. Tính biểu thức 3 sau đó quay trở lại bước 2 để bắt đầu một chu trình lặp mới.

Như vậy ta thấy:

- Biểu thức 1 bao giờ cũng chỉ được tính một lần

- Biểu thức 2, biểu thức 3 và thân vòng lặp for có thể thực hiện lặp đi lặp lại nhiều lần.

Sự hoạt động của câu lệnh for có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.4. Sơ đồ khối của câu lệnh for

Ví dụ 4.7: Viết chương trình in ra 3 dòng chữ TURBO C

Chương trình được viết như sau:

```
#include”stdio.h”
#include”conio.h”
void   main()
{
        int   i;
        for (i = 1; i<= 3; i++)
        printf(“TURBO  C\n”);
        getch();
}
```

Chú ý:

- Khi biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này để thoát ra khỏi câu lệnh for ta phải dùng câu lệnh break, goto hoặc return.

- Trong dấu ngoặc tròn sau từ khóa for gồm ba phần được phân cách nhau bởi dấu chấm phẩy. Trong mỗi phần có thể viết một hoặc một dãy biểu thức, khi đó mỗi biểu thức được phân cách nhau bởi dấu phẩy. Các biểu thức sẽ được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức trong phần thứ hai là tính đúng sai của biểu thức cuối cùng trong dãy biểu thức này.

- Bên trong thân câu lệnh for ta lại có thể viết một câu lệnh for khác, khi đó ta có các câu lệnh for lồng nhau

Ví dụ 4.8: Ví dụ về sử dụng câu lệnh for mà không có đầy đủ các thành phần trong dấu ngoặc tròn. Xét các đoạn chương trình sau:

```
1.      printf("Nhap so = ");
        scanf("%d", &so);
        for(; so < 100; )
          {
              printf("Nhap so = ");
              scanf("%d", &so);
          }
```

Câu lệnh trên không có phần khởi tạo giá trị (thiếu thành phần thứ nhất) và thay đổi giá trị của biến điều khiển (thiếu thành phần thứ ba). Vòng lặp dừng lại khi giá trị của số nhập vào lớn hơn hoặc bằng 100.

2. Câu lệnh for sau khi thiếu cả ba thành phần sẽ cho vòng lặp vô tận, khi đó muốn máy thoát ra khỏi vòng lặp ta có thể sử dụng câu lệnh break

```
for ( ; ; )
{
        printf(“vong lap vo tan”);
        i = getchar();
        if (i == ‘c’ || i == ‘C’)   break;
}
```

trong đoạn chương trình trên vòng lặp sẽ được thực hiện cho đến khi người dùng bấm phím ‘c’ hoặc ‘C’.

Ví dụ 4.9: Viết chương trình nhập vào một dãy số có n phần tử. In ra màn hình dãy số và phần tử lớn nhất của dãy đó.

Chương trình được viết như sau:

```
#include “stdio.h”
#include “conio.h”
void   main()
{
        float  day[20], max;
        int  n, i;
        printf(“Nhap so phan tu n = “);
        scanf(“%d”,&n);
        for  (i=1; i<=n; i++)
        {
                printf(“Nhap phan tu thu %d”,i);
                scanf(“%f”,&day[i]);
        }
        max = day[1];
        for (i = 2; i <=n; i++)
                if   (day[i] > max)    max = day[i];
        printf(“\n In day so ra man hinh:\n);
        for (i=1; i<=n; i++)
        printf(“%6.2f”,day[i]);
        printf(“Gia tri lon nhat cua day so la: %6.2f”, max);
        getch();
}
```

Ví dụ 4.10. Viết chương trình nhập vào một dãy số có n phần tử, sắp xếp dãy số theo chiều tăng dần và in ra màn hình dãy đã sắp xếp.

Chương trình được viết như sau:

```
#include “stdio.h”
#include “conio.h”
void   main()
{
        float  day[20];
        int  n, i, j;
        printf(“Nhap so phan tu n = “);
        scanf(“%d”,&n);
        for  (i=1; i<=n; i++)
        {
                printf(“Nhap phan tu thu %d”,i);
                scanf(“%f”,&day[i]);
        }
        printf(“\nDay so sau khi chua sap xep theo chieu tang dan”);
        for (i=1; i<=n; i++)
                printf(“%6.2f”,day[i]);
        //sắp xếp dãy số
        for  (i=1; i<=n - 1; i++)
                for (j =i+1; j<= n; j++)
                        if (a[i] > a[j])
                        {
                                tg = a[i]; a[i] = a[j]; a[j] = tg;
                        }
        printf(“\nDay so sau khi da sap xep theo chieu tang dan”);
        for (i=1; i<=n; i++)
                printf(“%6.2f”,day[i]);
        getch();
```

Ví dụ 4.11: Viết chương trình nhập vào một ma trận A_{n*n}. Sau đó in ra màn hình giá trị lớn nhất của nửa trên đường chéo chính của ma trận và đếm số giá trị max.

Chương trình được viết như sau:

```
#include “stdio.h”
#include “conio.h”
void   main()
{
        float  mt[20][20], max, x;
        int  n, m, i, đem;
        print(“Nhap co cua ma tran = “);
        scanf(“%d”,&n);
        for  (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {
                printf(“Nhap phan tu mt[%d][%d]”,i, j);
                scanf(“%f”,&mt[i][j]);
        }
        max = a[1][1];   đem  = 1;
        for  (i=1; i<=n; i++)
        for (j=i; j<=n; j++)
                if  (mt[i][j] >max)
                {
                        max = mt[i][j];
                        đem = 1;
                }
                else
                if    (mt[i][j] = max)   đem++;
```

```
        printf("\n In ma tran ra man hinh:\n);
        for (i=1; i<=n; i++)
        {
            for (j=1; j<=n; j++)
            printf("%6.2f",mt[i][j]);
            printf("\n");
        }
        printf("Gia tri lon nhat can tim la: %6.2f", max);
        printf("\n Co  %d gia tri %6.2f", đem, max);
        getch();
}
```

4.2.2. Câu lệnh while

Dùng để xây dựng một chu trình lặp mà số bước lặp chưa xác định. Cú pháp của câu lệnh như sau:

```
while   (biểu thức)
        <Công việc> /*thân vòng lặp*/
```

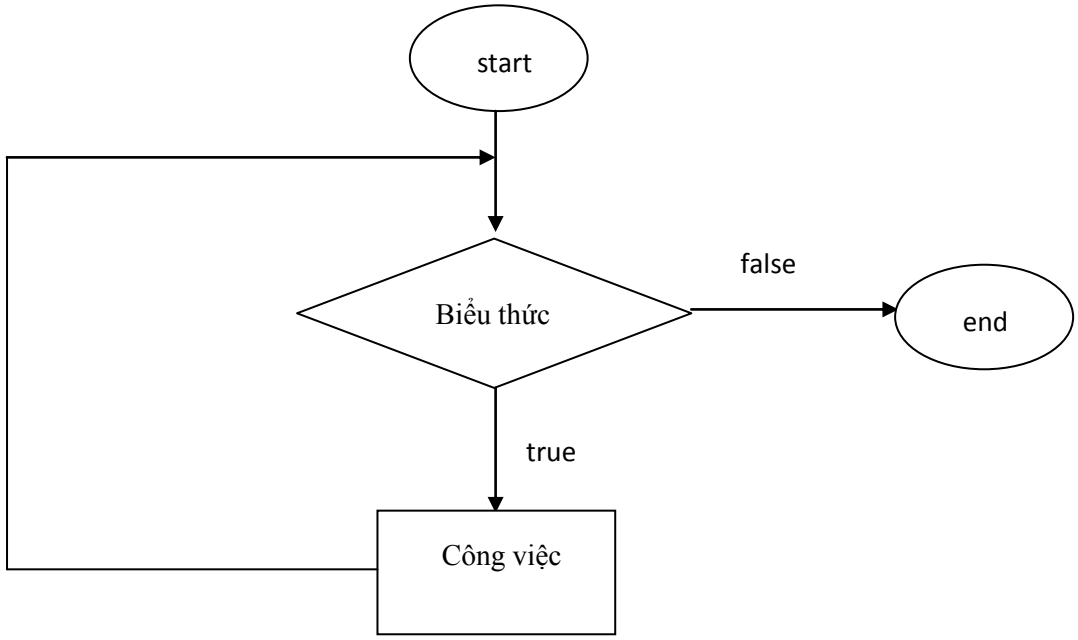
Sự hoạt động của câu lệnh:

1. Xác định giá trị của biểu thức.

2. Tùy thuộc vào giá trị đúng sai của biểu thức mà máy sẽ lựa chọn một trong hai nhánh: Nếu biểu thức có giá trị 0 (sai) thì máy sẽ thoát khỏi câu lệnh while và chuyển sang thực hiện câu lệnh phía sau thân while. Ngược lại nếu biểu thức có giá trị khác 0 (đúng) thì máy sẽ thực hiện <Công việc> . Khi gặp dấu ngoặc nhọn đóng cuối cùng của thân vòng lặp máy sẽ trở lại bước 1.

Nhận xét: Thân của câu lệnh while có thể được thực hiện một lần hoặc nhiều lần và cũng có thể không được thực hiện lần nào nếu ngay từ đầu biểu thức có giá trị bằng 0 (sai).

Sự hoạt động của câu lệnh while có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.5. Sơ đồ khối của câu lệnh while

Chú ý:

- Trong dấu ngoặc tròn sau từ khóa while có thể đặt một hay một dãy biểu thức phân cách nhau bởi dấu phẩy. Tính đúng sai của dãy biểu thức là tính đúng sai của biểu thức cuối cùng trong dãy.

- Bên trong thân của câu lệnh while cũng có thể viết các câu lệnh while hoặc for khác. Khi đó ta có các vòng lặp lồng nhau.

Ví dụ 4.12: Trở lại ví dụ 4.6 ta có thể sử dụng câu lệnh while thay câu lệnh goto và if để nhập giá trị điểm của một sinh viên theo yêu cầu điểm nằm trong khoảng từ 0 đến 10.

Chương trình được viết như sau:

```
#include “stdio.h”
#include”conio.h”
void  main()
{
        int  diem;
        printf(“Nhap diem:”);
```

```
        scanf("%d",&diem);
        while ((diem <0)||(diem >10))
        {
                printf("Nhap lai diem:");
                scanf("%d",&diem);
        }
  getch();
}
```

Ví dụ 4.13: Viết chương trình in ra 3 dòng chữ TURBO C trong ví dụ 4.7 bằng cách sử dụng câu lệnh while thay cho dùng câu lệnh for.

```
#include "stdio.h"
#include"conio.h"
void  main()
{
          int  i;
          while  (i <=3)  printf("TURBO C\n");
          getch();
}
```

Ví dụ 4.14: Viết chương trình tìm ước số chung lớn nhất của hai số a và b được nhập vào từ bàn phím.

```
#include "stdio.h"
#include"conio.h"
void  main()
{
        int  a, b, n, m, r;
        printf("Nhap a va b:");
        scanf("%d",&a, &b);
        n = a; m = b;
```

```
        r = n%m;
        while  (r)
        {
                n = m;
                m = r;
                r  = n % m;
        }
        printf("Uoc so chung lon nhat cua %d va %d la: %d",a,b,m);
        getch();
}
```

4.2.3. Câu lệnh do…while

Dùng để xây dựng một vòng lặp có số bước lặp chưa xác định. Nhưng câu lệnh do…while khác với câu lệnh while ở chỗ: trong câu lệnh while việc kiểm tra giá trị của biểu thức được thực hiện trước sau đó mới đến thực hiện <Công việc> trong thân vòng lặp. Còn đối với câu lệnh do…while thì ngược lại, việc kiểm tra giá trị của biểu thức đặt ở cuối vòng lặp, do đó thân của vòng lặp bao giờ cũng được thực hiện ít nhất một lần. Cú pháp của do…while như sau:

```
do
   < Công việc> /*thân vòng lặp*/
while  (biểu thức);
```

Sự hoạt động của do…while:

1. Thực hiện <Công việc> trong thân vòng lặp

2. Khi gặp dấu ngoặc nhọn } cuối cùng trong thân vòng lặp, máy sẽ xác định giá trị của biểu thức đứng sau từ khóa while.

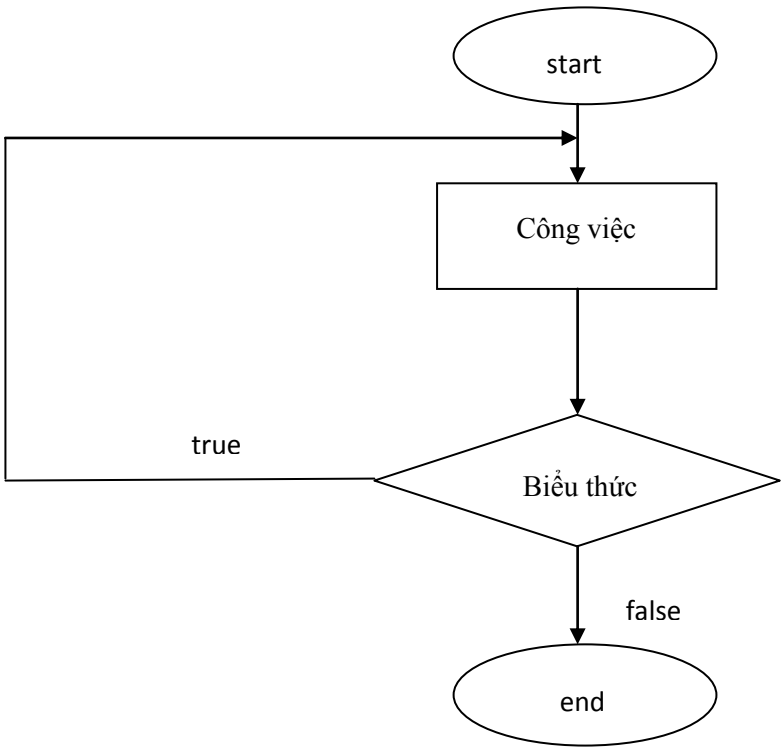
3. Tùy thuộc vào giá trị đúng sai của biểu thức mà máy sẽ lựa chọn một trong hai nhánh: Nếu biểu thức có giá trị khác 0 (đúng) thì máy sẽ trở lại bước 1 để tiếp tục thực hiện vòng lặp mới. Ngược lại nếu biểu thức có giá trị bằng 0 (sai) thì máy sẽ thoát khỏi câu lệnh do…while và chuyển tới thực hiện câu lệnh đứng sau dấu chấm phẩy của biểu thức.

Chú ý:

- Trong dấu ngoặc tròn sau từ khóa while có thể đặt một hay một dãy biểu thức phân cách nhau bởi dấu phẩy. Tính đúng sai của dãy biểu thức là tính đúng sai của biểu thức cuối cùng trong dãy.

- Bên trong thân của câu lệnh do…while cũng có thể viết các câu lệnh while, do…while hoặc for khác. Khi đó ta có các câu lệnh do…while lồng nhau.

Sự hoạt động của câu lệnh do…while có thể được biểu diễn bởi sơ đồ khối sau đây:



Hình 4.6. Sơ đồ khối của câu lệnh do… while

Ví dụ 4.15: Trở lại ví dụ 4.12 ta viết chương trình nhập điểm của sinh viên bằng câu lệnh do…while thay vì câu lệnh while.

```
#include “stdio.h”
#include”conio.h”
void  main()
{
        int  diem;
        do
        {
```

```
            printf("Nhap diem:");
            scanf("%d",&diem);
        }  while ((diem < 0) || (diem > 10));
          getch();
}
```

Ví dụ 4.16: Viết chương trình tìm ước số chung lớn nhất của hai số a và b bằng cách sử dụng câu lệnh do…while thay vì dùng câu lệnh while trong ví dụ 4.14

```
#include "stdio.h"
#include"conio.h"
void  main()
{
        int  a, b, n, m, r;
        printf("Nhap a va b:");
        scanf("%d",&a, &b);
        n = a; m = b;
        do
        {
          r  = n % m;
          n = m;
          m = r;
        }while (r);
        printf("Uoc so chung lon nhat cua %d va %d la: %d",a,b,n);
        getch();
}
```

Nhận xét:

- Những bài tập sử dụng câu lệnh lặp while đều sử dụng được câu lệnh do…while và ngược lại.

- Những bài tập sử dụng câu lệnh for đều sử dụng được câu lệnh while và câu lệnh do…while nhưng điều ngược lại không phải lúc nào cũng đúng.

4.3. Câu lệnh break và câu lệnh continue

4.3.1. Câu lệnh break

Cho phép thoát khỏi các câu lệnh lặp for, while, do…while và câu lệnh switch bên trong nhất chứa nó. Như vậy, break cho phép thoát ra khỏi một vòng lặp mà không cần dùng đến điều kiện kết thúc vòng lặp.

Ví dụ 4.17: Viết chương trình nhập vào số n sau đó in ra màn hình n là số nguyên tố hay không.

```
#include “stdio.h”
#include”conio.h”
void  main()
{
        int  n, i, kt;
        printf(“Nhap n:”);
        scanf(“%d”,&n);
        kt = 1;
        for (i=2; i < n; i++)
        {
                if  (n % i == 0)
                        {
                                kt = 0;
                                break;
                         }
        }
        if    (kt)   printf(“%d la so nguyen to”, n);
        else   printf(“%d khong la so nguyen to”, n);
        getch();
```

```
}
```

4.3.2. Câu lệnh continue

Khi gặp câu lệnh này bên trong thân của một câu lệnh for, while, do…while máy sẽ bỏ qua các câu lệnh còn lại trong thân vòng lặp và chuyển sang bắt đầu một vòng mới của chu trình.

Chú ý: Câu lệnh continue chỉ được sử dụng cho các chu trình chứ không sử dụng cho câu lệnh switch.

Ví dụ 4.18: Viết chương trình nhập vào một dãy số có n phần tử sau đó in ra tổng của các phần tử dương trong dãy

```
#include “stdio.h”
#include “conio.h”
void   main()
{
        float  day[20], tong;
        int  n, i;
        print(“Nhap so phan tu n = “);
        scanf(“%d”,&n);
        for  (i=1; i<=n; i++)
        {
                printf(“Nhap phan tu thu %d”,i);
                scanf(“%f”,&day[i]);
        }
        tong = 0;
        for  (i=1; i<=n; i++)
        {
                if (day[i] <=0)  continue;
                tong = tong + day[i];
        }
```

```
        printf("Tong cua cac phan tu duong trong day la %6.2f", tong);
        getch();
}
```

Bài tập cuối chương

Câu 1. Viết chương trình giải phương trình bậc hai

$ax^2 + bx + c = 0$

Câu 2. Viết chương trình giải hệ phương trình bậc nhất hai ẩn sau

$ax + by = c$

$dx + ey = f$

Câu 3. Viết chương trình nhập vào một số nguyên rồi in ra màn hình tất cả các ước số của số đó.

Câu 4. Viết chương trình nhập vào ba số a, b, c. Kiểm tra xem ba số đó có tạo thành ba cạnh của một tam giác hay không? Nếu có thì kiểm tra xem tam giác đó là tam giác gì?

Câu 5. Viết chương trình nhập vào một tháng và năm bất kỳ sau đó kiểm tra xem tháng đó có bao nhiêu ngày.

Câu 6. Viết chương trình để

- Nhập vào một dãy số từ bàn phím

- Tính trung bình cộng của các số dương và trung bình cộng của các số âm của dãy số trên.

Câu 7. Viết chương trình tính e^x theo công thức xấp xỉ

$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$ với độ chính xác 0.00001. Tức là n cần chọn sao cho $\frac{x^n}{n!} < 0.00001$

Câu 8. Viết chương trình tính tổng

$$S = 1 + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{n!}$$

Với giá trị của n được nhập vào từ bàn phím.

Câu 9. Viết chương trình tính tổng sau:

$$S = \frac{1}{1^2} + \frac{1}{2^2} + ... + \frac{1}{n^2}$$

Câu 10. Viết chương trình nhập vào một dãy số, sau đó in ra màn hình các số âm ở trên một dòng và các số dương ở trên một dòng.

Câu 11. Viết chương trình tìm phần tử âm cuối cùng của dãy số a_1, a_2, …, a_n.

Câu 12. Cho hai dãy số a_1, a_2, …, a_n. và b_1, b_2,…, b_n đều được sắp xếp theo thứ tự tăng dần. Viết chương trình để từ hai dãy số trên xây dựng một dãy mới cũng theo thứ tự tăng dần.

Câu 13. Cho dãy số gồm n phần tử: a_1, a_2,…, a_n. Đưa tất cả các số dương của dãy số cho vào một mảng b và các số âm của dãy vào một mảng c.

Câu 14. Viết chương trình nhập vào một số n sau đó in ra màn hình ma trận xoáy cấp n có dạng như sau:

Giả sử với n = 4

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Câu 15. Cho một dãy số gồm n phần tử, in ra màn hình giá trị tích của các phần tử âm trong dãy.

Câu 16. Viết chương trình nhập vào hai ma trận A_{n*m} và B_{m*p}. Tính ma trận tích $C_{n*p} = A_{n*m} * B_{m*p}$ sau đó in cả ba ma trận ra màn hình.

Câu 17. Viết chương trình nhập vào ma trận A_{n*m}. Tìm ma trận chuyển vị của ma trận A sau đó in ra màn hình ma trận chuyển vị vừa tìm được.

Câu 18. Viết chương trình nhập vào ma trận A_{n*n} sau đó thực hiện các công việc sau:

a. Kiểm tra xem ma trận vừa nhập có phải là ma trận đơn vị hay không.
b. Tính tổng của các phần tử nằm trên hàng thứ i, với i nhập từ bàn phím (1<= i <= n).
c. Tính tích của các phần tử nằm trên cột thứ j, với j nhập vào từ bàn phím (1<= j <= n).
d. Kiểm tra xem trên hàng thứ i có chứa số nguyên tố hay không? (i = 1..n) và hàng nào có chứa nhiều số nguyên tố nhất.

e. Ma trận trên có chứa bao nhiêu số dương và bao nhiêu số âm.

CHƯƠNG 5: CON TRỎ VÀ HÀM

5.1. Con trỏ

5.1.1. Địa chỉ

Khi làm việc với một biến ta quan tâm đến ba yếu tố: Tên biến, kiểu biến và giá trị của biến. Giả sử có một khai báo float x = 3.4; xác định một biến có tên là x, kiểu là float và giá trị bằng 3.4.

Ta cũng biết, với khai báo trên, máy sẽ cấp phát cho x một khoảng nhớ gồm 4 byte liên tiếp. Địa chỉ của biến là số thứ tự của byte đầu tiên trong dãy byte liên tiếp mà máy dành cho biến (các byte được đánh số từ 0).

Mặc dù địa chỉ của biến là một số nguyên nhưng không có nghĩa nó là các số nguyên thông thường dùng trong các phép tính. 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 máy sẽ phân biệt các kiểu địa chỉ: địa chỉ kiểu int, kiểu float, kiểu double,…Phép toán &x sẽ cho ta địa chỉ của biến x.

5.1.2. Con trỏ

Con trỏ là một biến dùng để chứa địa chỉ. Vì có nhiều kiểu địa chỉ nên cũng có nhiều kiểu con trỏ tương ứng. Con trỏ kiểu int dùng để chứa địa chỉ của biến kiểu int, con trỏ kiểu float dùng để chứa địa chỉ của biến kiểu float…

5.1.2.1. Khai báo con trỏ.

Cũng như các biến khác, con trỏ phải được khai báo trước khi sử dụng. Việc khai báo biến con trỏ như sau:

Kiểu dữ liệu *tên con trỏ;

Ví dụ 5.1: Cho các khai báo sau:

- int x, y, *p, *q; khai báo hai biến x, y kiểu int và hai con trỏ p, q kiểu int

- float *t; khai báo một con trỏ t kiểu float

Các câu lệnh: p = &x; và q = &y; sẽ lần lượt gán địa chỉ của biến x cho con trỏ p và địa chỉ của biến y cho con trỏ q.

Câu lệnh t = &x; sẽ bị báo lỗi vì t là con trỏ kiểu float, nó chỉ chứa được địa chỉ của các biến kiểu float. Câu lệnh này để gán địa chỉ của biến nguyên x cho con trỏ t là không được chấp nhận.

Chú ý:

1. Khi con trỏ p chứa địa chỉ của biến x thì ta còn có thể nói là con trỏ p trỏ tới x. Và hai cách viết sau là tương đương

- x tương đương với *p, khi đó *p được coi là bí danh của biến x

- &x tương đương với p

2. Mọi kiểu con trỏ đều có thể nhận hằng địa chỉ NULL (NULL là một giá trị hằng đặc biệt để chỉ ra rằng một con trỏ không trỏ vào đối tượng nào cả). Các biến con trỏ trong chương trình khi chưa được sử dụng thì ta nên gán giá trị cho nó là hằng địa chỉ NULL.

5.1.2.2. Các phép toán với con trỏ

- Phép gán: Có thể gán cho một con trỏ nhận giá trị hằng địa chỉ NULL bằng phép gán p = NULL;

Cũng có thể gán các con trỏ cùng kiểu, giả sử có con trỏ nguyên p trỏ vào biến nguyên x, q cũng là một con trỏ nguyên. Khi đó nếu có phép gán q = p; điều đó có nghĩa là hai con trỏ p và q cùng trỏ vào đối tượng x. Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu như sau:

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

- Phép tăng, giảm địa chỉ: Xét các câu lệnh

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

Cho biết px là con trỏ kiểu float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ bốn byte, nên phép tăng giảm địa chỉ được thực hiện trên bốn byte. Hay nói cách khác:

px++ trỏ tới phần tử x[11]

px - - trỏ tới phần tử x[9]

px + i trỏ tới phần tử x[10 + i]

px – i trỏ tới phần tử x[10 – i]

- Phép so sánh: Mọi con trỏ đều so sánh được với hằng NULL theo phép toán so sánh bằng (==) và so sánh khác nhau (!=).

Các con trỏ cùng kiểu cũng có thể so sánh với nhau bằng phép toán == và !=, ví dụ nếu p1 và p2 là hai con trỏ float thì:

p1 == p2 nếu hai con trỏ p1 và p2 cùng trỏ tới một đối tượng

p1 != p2 nếu hai con trỏ p1 và p2 trỏ tới các đối tượng khác nhau

5.1.2.3. Mảng con trỏ

Là mảng mà mỗi phần tử của mảng thuộc kiểu con trỏ dùng để chứa một địa chỉ nào đó. Cách khai báo mảng con trỏ cũng như khai báo mảng thông thường chỉ khác là thêm dấu * vào trước tên mảng như sau:

kiểu dữ liệu * mảng_1, *mảng_2,…, *mảng_n;

trong đó mỗi mảng được khai báo theo cú pháp:

tên_mảng[k1][k2]…[km] với ki là kích thước của mỗi chiều (i=1..m).

Ví dụ 5.2: Câu lệnh float *pa[100]; khai báo một mảng con trỏ kiểu float gồm 100 phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ địa chỉ kiểu float.

Chú ý: Ta cần lưu ý rằng trước khi sử dụng một mảng con trỏ ta cần gán giá trị cho mỗi phần tử của mảng này và giá trị này phải là địa chỉ của một biến hoặc của một phần tử mảng. Các phần tử của mảng con trỏ kiểu char có thể được khởi đầu bằng các xâu ký tự.

Ví dụ 5.3: Xét các khai báo sau:

1. int n, m ,*p[10]; khai báo hai biến nguyên n, m và mảng con trỏ nguyên p. Khi đó trong chương trình ta có thể sử dụng phép gán sau: p[1] = &n; tức là gán địa chỉ của biến n cho phần tử p[1]. Phép gán p[2] = &m; tức là gán địa chỉ của biến m cho phần tử p[2]

2. char *array[10] = {“Happy”,”New”,”Year”}; khai báo mảng con trỏ kiểu char tên là array và khởi đầu giá trị cho các phần tử của mảng.

5.2. Hàm

Hàm là một chương trình con thực hiện một khối công việc được lặp đi lặp lại nhiều lần trong khi chạy chương trình hoặc dùng để tách một khối công việc cụ thể để chương trình đỡ phức tạp.

Một chương trình viết theo ngôn ngữ C là một dãy các hàm trong đó có một hàm chính (hàm main). Thứ tự của các hàm trong chương trình là bất kỳ nhưng chương trình bao giờ cũng được thực hiện từ hàm main, mỗi hàm sẽ thực hiện một công việc nào đó giúp cho việc giải quyết bài toán trở nên hiệu quả hơn.

5.2.1. Cách tổ chức hàm

Trong C, hàm được xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không cho phép xây dựng một hàm bên trong các hàm khác. Hàm được viết theo thứ tự sau:

- Dòng tiêu đề: trong dòng đầu tiên của một hàm chứa các thông tin về: kiểu hàm, tên hàm, kiểu và tên mỗi đối của hàm.

Ví dụ 5.4: Xét khai báo

float max_3_số(float a, float b, float c)

là khai báo hàm tính số lớn nhất của ba số a, b và c có tên là max_3_số, trong đó a, b, c là các đối có kiểu là số thực. Hàm trả về số lớn nhất của ba số a, b, c là một số thực, tức là kiểu của hàm là kiểu số thực.

Chú ý: Đối với các hàm không có giá trị thì kiểu hàm là kiểu void

- Thân hàm: là nội dung chính của hàm bắt đầu bằng dấu { và kết thúc bởi dấu }. Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đề ra cho hàm. Trong thân hàm có thể sử dụng một câu lệnh return để trả về giá trị của hàm, có thể dùng nhiều câu lệnh return ở những chỗ khác nhau và cũng có thể không dùng câu lệnh này. Dạng tổng quát của nó là: return (biểu thức). Khi gặp câu lệnh này thì máy sẽ thoát khỏi hàm và giá trị của biểu thức sẽ được gán cho hàm.

Chú ý:

+ Hàm phải được khai báo ở vị trí trước nơi nó được gọi sử dụng.

+ Nếu có khai báo nguyên mẫu của hàm (nguyên mẫu của hàm chính là dòng tiêu đề của hàm kèm theo dấu chấm phẩy ở cuối cùng) thì nguyên mẫu phải đứng trước nơi hàm được gọi, còn hàm có thể khai báo ở vị trí bất kỳ nhưng phải sau nguyên mẫu. Không bắt buộc phải khai báo nguyên mẫu hàm nhưng nên có để chương trình phát hiện lỗi tốt hơn. Nếu có khai báo nguyên mẫu hàm thì trong danh sách tên đối chỉ cần viết tên kiểu dữ liệu của đối mà không cần viết cả tên đối.

Một chương trình C thường được tổ chức như sau:

- Các #include
- Các #define
- Các khai báo biến ngoài, mảng ngoài, cấu trúc ngoài
- Các khai báo hàm
- Các khai báo nguyên mẫu hàm.
- Hàm main
- Các khai báo hàm

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

- Sử dụng biến ngoài/mảng ngoài.
- Sử dụng tham số khi gọi hàm.

Ví dụ 5.5: Viết chương trình giải phương trình bậc hai $ax^2 + bx + c = 0$ bằng cách sử dụng hàm.

Chương trình được tổ chức thành các hàm gồm ba hàm delta_duong, delta_am và delta_bang_0, mỗi hàm thực hiện các công việc ứng với các trường hợp tương ứng delta > 0, delta < 0 và delta = 0. Trong hàm chính main(), chúng ta dùng phép gán để tính giá trị delta sau đó kiểm tra giá trị của delta lớn hơn 0, nhỏ hơn 0 hay bằng 0. Ứng với mỗi trường hợp của delta thì sẽ gọi hàm tương ứng. Trong đó việc truyền dữ liệu được thực hiện bằng cách sử dụng các biến ngoài là a, b, c và việc nhận kết quả được thực hiện bằng cách sử dụng các biến ngoài del_ta, x1, x2. Chương trình được viết như sau:

```
#include”stdio.h”
#include”conoi.h”
#include”math.h”
float  a, b, c, delta, x1, x2;
void   delta_duong(void);
void   delta_am(void);
void delta_bang_0(void);
void  main()
```

```
{
        printf("Nhap cac he so a, b, c:");
        scanf("%f %f %f", &a, &b, &c);
        delta = b*b – 4*a*c;
        if  (delta > 0)          delta_duong();
        else
        if  (delta == 0)        delta_bang_0();
        else        delta_am();
        getch();
}
void   delta_duong(void)
{
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b – sqrt(delta))/(2*a);
        printf("Phuong trinh co hai nghiem phan biet\n");
        printf("%6.2f \n",x1);
        printf("%6.2f \n", x2);
}
void   delta_bang_0(void)
{
        x1 = -b/(2*a);
        printf("Phuong trinh co nghiem kep x1 = x2 = %6.2f", x1);
}
void  delta_am()
{
        printf("Phuong trinh vo nghiem");
}
```

5.2.2. Biến/mảng tự động

Là biến/mảng được khai báo ở bên trong thân của một hàm bất kỳ, kể cả hàm main

Thân của một hàm được xem như là một khối lệnh và nó được coi như là một khối lệnh cực đại theo nghĩa nó có thể chứa nhiều khối lệnh khác bên trong và không thể có khối lệnh nào chứa nó. Các biến/mảng tự động được phép khai báo ở đầu bất kỳ khối lệnh nào nhưng phải đứng trước mọi câu lệnh không thuộc dạng khai báo.

Khi máy bắt đầu làm việc với một khối lệnh thì các biến/mảng tự động khai báo bên trong nó mới được cấp phát bộ nhớ và bắt đầu được sử dụng. Khi kết thúc khối lệnh này thì các biến/mảng tự động của khối lệnh sẽ được giải phóng.

Do chương trình bắt đầu thực hiện từ câu lệnh đầu tiên của hàm main và kết thúc khi gặp dấu } cuối cùng của hàm main do vậy các biến/mảng tự động được khai báo trong hàm main sẽ tồn tại trong suốt thời gian sử dụng chương trình

Các biến/mảng tự động chỉ có tác dụng bên trong thân của khối lệnh mà tại đó chúng được khai báo tức là không được đưa ra sử dụng ở bất kỳ vị trí nào bên ngoài khối lệnh, và ở bất kỳ vị trí nào bên ngoài khối lệnh cũng không thể can thiệp vào các biến/mảng bên trong khối lệnh.

Nếu bên trong một khối lệnh có một biến/mảng tự động có tên là A thì điều này cũng không làm thay đổi giá trị của một biến/mảng cũng có tên là A được khai báo bên ngoài khối lệnh đó vì khi đó máy coi đó là hai biến hoàn toàn khác nhau.

Nếu một biến/mảng được khai báo ở bên ngoài khối lệnh mà không trùng tên với biến/mảng nào ở bên trong khối lệnh thì biến/mảng này có thể sử dụng cả bên trong và bên ngoài khối lệnh.

Ta có thể áp dụng cơ chế khởi đầu cho biến/mảng tự động theo các cách đã giới thiệu ở các chương 2.

Ví dụ 5.6: Chương trinh mình họa những đặc điểm trên

```
#include”stdio.h”
#include”conio.h”
void   main()
{
```

```
        int   n =10; m = 20;
        textmode(C80);
        int p; //khai báo sai vị trí
        //Sửa lỗi: phải khai báo trước câu lệnh textmode(C80);
        printf("n tren = %d, m tren = %d ", n, m);
        {
            int   n = 30,  p = 80;
            m = m * 5;
            printf("\nn trong = %d, m trong = %d, p trong = %d",
            n, m, p);
        }
      printf("\n n duoi = %d, m duoi = %d", n, m);
      getch();
}
```

Sau khi sửa lỗi. Khi chạy chương trình trên máy cho kết quả như sau:

n tren = 10, m tren = 20

n trong = 30, m trong = 100, p trong = 80

n duoi = 10, m duoi = 100

5.2.3. Biến/mảng ngoài

Là các biến/mảng được khai báo bên ngoài các hàm. Biến/mảng ngoài được cấp phát bộ nhớ và tồn tại trong suốt thời gian thực hiện chương trình, phạm vi sử dụng của biến/mảng ngoài là từ vị trí khai báo nó cho tới cuối tệp gốc chương trình.

Quy tắc về khởi đầu giá trị:

- Các biến/mảng ngoài có thể vừa khai báo vừa được khởi đầu giá trị theo cú pháp đã biết. Nếu không được khởi đầu máy sẽ gán cho chúng giá trị không.

- Khi khởi đầu mảng ngoài có thể không cần chỉ ra kích thước của mảng. Khi đó máy sẽ dành cho mảng một khoảng nhớ đủ để thu nhận danh sách giá trị khởi

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

Ví dụ 5.7: float a[] = {1.2, 5, 9};

```
int b[][3] = {
                {1, 2, 3},
                {4, 5, 6}
            };
```

- Bộ khởi đầu cho một mảng char có thể là danh sách các hằng ký tự hoặc là một hằng xâu ký tự

Ví dụ 5.8: char name[] = {'H', 'a', 'n', 'o', 'i', '\0'};

char name[] = "Hanoi";

5.2.4. Cách truyền tham số khi gọi hàm

Sử dụng hàm: Hàm được sử dụng thông qua lời gọi tới nó. Cách viết một lời gọi hàm như sau:

- Với các hàm không kiểu (kiểu của hàm là void) thì khi gọi hàm ta viết: tên_hàm([danh sách các tham số thực sự]); (Có dấu chấm phẩy đằng sau).

- Với các hàm có kiểu (kiểu của hàm là int, float,…) thì khi gọi hàm ta phải viết tên hàm trong một biểu thức, và viết là tên_hàm([danh sách các tham số thực sự]) (Chú ý là không có dấu chấm phẩy đằng sau lời gọi).

Chú ý: Số tham số thực sự phải bằng với số đối, kiểu của tham số thực sự phải phù hợp với kiểu của đối tương ứng.

Nguyên tắc hoạt động của hàm:

Khi gặp một lời gọi hàm ở một vị trí nào đó thì hàm bắt đầu được thực hiện, tức là máy sẽ tạm rời vị trí đó và chuyển đến hàm tương ứng. Quá trình đó sẽ diễn ra theo trình tự bốn bước sau:

1. Cấp phát bộ nhớ cho các đối và các biến/mảng tự động

2. Gán giá trị của các tham số thực sự cho các đối tương ứng: Nếu đối không phải là con trỏ thì gán giá trị của tham số thực sự cho đối, nếu đối là con trỏ thì gán địa chỉ của tham số thực sự cho đối.

3. Thực hiện các câu lệnh trong thân hàm

4. Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ giải phóng bộ nhớ cho các đối, các biến/mảng tự động, thoát khỏi hàm và trở về nơi gọi hàm đó. Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm, tức là hàm mang theo một giá trị.

Chú ý:

- Do đối và biến/mảng tự động đều có phạm vi hoạt động trong cùng một hàm nên đối và biến/mảng tự động cần có tên khác nhau

- Đối và biến/mảng tự động được cấp phát bộ nhớ khi hàm được xét đến và sẽ bị giải phóng khỏi bộ nhớ trước khi ra khỏi hàm. Như vậy, không thể mang giá trị của đối ra khỏi hàm. Có nghĩa là không thể sử dụng đối để làm thay đổi giá trị của bất kỳ đại lượng nào bên ngoài hàm. Nhưng đối và biến/mảng tự động có thể trùng tên với bất kỳ đại lượng nào ở bên ngoài hàm mà không gây ra một nhầm lẫn nào.

- Khi một hàm được gọi, giá trị của các tham số thực sự được gán cho các đối không phải là con trỏ. Như vậy, các đối không phải là con trỏ chính là các bản sao của các tham số thực sự. Hàm chỉ làm việc trên các đối này, tức là chỉ làm việc trên các bản sao. Các đối này có thể bị biến đổi trong thân hàm, nhưng các tham số thực sự không bị thay đổi.

Ví dụ 5.9: Xét hàm hoán vị hai số a và b

```
void   hoan_vi(float  x, float  y)
{
        float  tg;
        tg = x;
        x = y;
        y = tg;
}
```

Hàm main sử dụng hàm hoan_vi

```
void   main()
{
        float  x = 5, y = 8;     //dòng lệnh 1
        hoán_vị(x,y);            // dòng lệnh 2
```

```
    printf("x = %6.2f, y = %6.2f", x, y);  //dòng lệnh 3
    getch();
}
```

Phân tích: khi gặp dòng lệnh 1 của hàm main, máy dành hai khoảng nhớ cho x và y và khởi đầu giá trị x = 5, y = 8. Câu lệnh thứ 2 là lời gọi hàm. Khi gặp lời gọi này máy thực hiện các công việc sau:

- Cấp phát bộ nhớ cho các đối x, y và biến tự động z. ở đây tuy có sự trùng tên giữa các biến x, y ở hàm main và các đối x, y ở hàm hoán_vị nhưng máy vẫn phân biệt được vì biến x, y ở hàm main và đối x, y ở hàm hoan_vi được bố trí ở các khoảng nhớ hoàn toàn khác nhau. Biến x, y tồn tại trong suốt quá trình thực hiện chương trình còn đối x, y chỉ tồn tại khi máy làm việc bên trong hàm hoan_vi.

- Vì trong hàm hoan_vi các đối x và y không phải là các con trỏ nên máy sẽ gán giá trị của các tham số thực sự x, y cho các đối x và y của hàm. Như vậy đối x có giá trị là 5 và đối y có giá trị là 8

- Máy thực hiện các câu lệnh trong thân hàm hoan_vi tức là làm nhiệm vụ hoán vị hai đối x và y. Kết quả đối x có giá trị 8 còn đối y là 5.

- Gặp dấu } cuối cùng trong thân hàm hoan_vi, máy sẽ giải phóng bộ nhớ đã cấp phát cho hai đối x và y và biến tự động z, thoát ra khỏi hàm hoan_vi và trở lại hàm main sau đó thực hiện câu lệnh printf ở phía sau.

Kết quả máy sẽ in ra màn hình giá trị của hai biến x, y, kết quả của câu lệnh này sẽ là x = 5, y = 8. Vậy hàm hoan_vi không hề làm thay đổi giá trị của các biến x, y. Do đó chương trình trên chưa đáp ứng được yêu cầu đề bài đưa ra.

5.2.5. Hàm có đối con trỏ

Ở phần trên, chúng ta đã thực hiện ví dụ với hàm mà trong đó đối của hàm có kiểu là int/float (không phải là kiểu con trỏ), thì khi đó tham số thực sự tương ứng phải là một biến kiểu int/float. Trong trường hợp, nếu đối của hàm là một con trỏ kiểu int, float,…thì tham số thực sự tương ứng phải là địa chỉ của biến/mảng kiểu int, float,… Khi đó địa chỉ của tham số thực sự được truyền cho đối con trỏ tương ứng. Do đã biết địa chỉ của tham số thực sự, nên ta có thể gán cho nó các giá trị mới bằng các câu lệnh bất kỳ trong thân hàm. Vậy bằng cách sử dụng đối con trỏ, ta có thể thay đổi giá trị của các tham số thực sự tương ứng. Ta xây dựng lại hàm hoan_vi trong ví dụ 5.9 trên sử dụng đối con trỏ:

Ví dụ 5.10: Viết lại hàm hoan_vi trong ví dụ 5.9 sử dụng đối con trỏ

```
void   hoan_vi(float  *px, float  *py)
{
        float  tg;
        tg =  *px;
        *px = *py;
        *py = tg;
}
```

Hàm main sử dụng hàm hoan_vi

```
void   main()
{
        float  x = 5, y = 8;   //dòng lệnh 1
        hoan_vi(&x, &y);   //dòng lệnh 2
        printf("x = %6.2f, y = %6.2f", x, y); //dòng lệnh 3
        getch();
}
```

Phân tích: Khi gặp lời gọi tới hàm hoan_vi máy sẽ thực hiện các công việc sau:

- Cấp phát bộ nhớ cho biến tự động tg

- Gán địa chỉ của các tham số thực sự cho các đối con trỏ tương ứng: tức là gán địa chỉ của x cho con trỏ px, gán địa chỉ của y cho con trỏ py

- Thực hiện các câu lệnh trong thân hàm. Câu lệnh tg = *px; tức là gán giá trị của địa chỉ mà con trỏ px trỏ vào cho biến tg, hay nói cách khác là gán giá trị của x cho tg, vậy câu lệnh trên cũng có thể viết là tg = x. Các câu lệnh sau cũng tương tự là x = y; y = tg; Như vậy x sẽ nhận giá trị của y và ngược lại.

- Thoát ra khỏi hàm, giải phóng biến tự động tg, trở về hàm main thực hiên câu lệnh printf ở phía sau. Vậy trên màn hình sẽ có kết quả x = 8, y = 5.

Chú ý: Trong lời gọi hàm ở dòng lệnh 2 ta viết hoan_vị(&x, &y); Bởi vì các đối px và py trong hàm hoan_vi là các con trỏ, khi gọi hàm ta phải gán địa chỉ của

các tham số thực sự x và y cho các đối px và py vì vậy phải dùng cách viết &x, &y, chứ không được viết là x, y.

Phân loại đối: Như vậy, đối của hàm được chia thành hai loại:

- Loại thứ nhất dùng để chứa các giá trị đã biết, gọi là đối vào

- Loại thứ hai dùng để chứa các kết quả nhận được, gọi là đối ra

Vậy từ các ví dụ trên ta đặt câu hỏi khi nào thì sử dụng đối con trỏ? câu trả lời là: Khi cần thay đổi giá trị của các tham số thực sự thì đối tương ứng phải là con trỏ hay nói cách khác các đối ra phải là con trỏ

5.2.6. Con trỏ và mảng một chiều

Phép toán lấy địa chỉ: Giả sử có khai báo float a[10]; khi đó phép toán &a[i] cho địa chỉ của phần tử a[i] trong mảng.

Tên mảng là một hằng địa chỉ: Với mảng a được khai báo như trên thì máy sẽ bố trí 10 khoảng nhớ liên tiếp (mỗi khoảng nhớ 4 byte). Như vậy nếu biết địa chỉ của một phần tử nào đó trong mảng a thì dễ dàng suy ra địa chỉ của các phần tử còn lại. Tên mảng là một hằng địa chỉ, nó chính là địa chỉ của phần tử đầu tiên của mảng. Như vậy, các cách viết sau là tương đương:

- a tương đương với &a[0]

- a + i tương đương với &a[i]

- *(a+i) tương đương với a[i]

Khi gán tên mảng a cho một con trỏ p cùng kiểu nào đó bằng phép gán p = a; thì:

- p trỏ tới phần tử đầu tiên của mảng, tức là a[0]

- p+i trỏ tới phần tử a[i]

- *(p+i) tương đương với p[i]

Chú ý:

- Nếu muốn truyền tên mảng một chiều (tên mảng là một địa chỉ) vào chương trình con trong lời gọi chương trình con thì đối phải là con trỏ cùng kiểu với kiểu phần tử mảng.

- Con trỏ có thể được khai báo thông qua mảng hình thức một chiều như sau:

int *p, q[]; //q[] tương đương với *q

Ví dụ 5.11: Viết chương trình minh họa cách dùng con trỏ để nhập vào hai dãy số theo chiều tăng dần, sau đó trộn hai dãy thành một dãy cũng theo chiều tăng dần. In ba dãy ra màn hình

```
#include “stdio.h”
#include”conio.h”
void  nhap_day(int  a[], int  *n, char  ten)
{
        int  i;
        print(“Nhap so phan tu cua day %c:”, ten);
        scanf(“%d”,n);
        for (i = 0; i<*n; i++)
        {
                printf(“Nhap %c[%d] = “, ten, i);
                scanf(“%d”, &a[i]); //có thể viết a + i
        }
}
void  in_day(int  a[], int  n, char  ten)
{
        int  i;
        printf(“In day %c \n”, ten);
        for (i = 0; i < n; i++)
        printf(“%6d”, *(a+i)); //có thể viết là a[i]
        printf(“\n”);
}
void  main()
{
        int  n, m, a[20], b[20], c[20], i, j, k;
        nhap_day(a, &n, ‘A’);
```

```
        nhap_day(b, &m, 'B');
        i = j = k = 0;
        while  (k<m+n)
        {
            if  (i = n)
                {
                    c[k++] = b[j++];
                    continue;
                }
            if  (j = m)
                {
                    c[k++] = a[i++];
                    continue;
                }
            if  (a[i] < b[j])  c[k++] = a[i++];
            else  c[k++] = b[j++];
        }
        in_day(a, n, 'A');
        in_day(b, m, 'B');
        in_day(c, n+m, 'C');
        getch();
}
```

5.2.7. Con trỏ và mảng nhiều chiều

Ta cũng có thể lấy địa chỉ của mảng nhiều chiều giống như lấy địa chỉ của mảng một chiều, giả sử với một mảng hai chiều ta sẽ viết như sau &a[i][j]

Trong C, mảng n chiều chính là mảng một chiều có các phần tử là mảng n – 1 chiều. Do vậy mảng hai chiều là mảng một chiều có các phần tử là mảng một chiều.

Như vậy với khai báo int a[2][3],*p; thì a là một mảng mà mỗi phần tử của nó là một dãy 3 số int (một hàng của bảng). Vì vậy:

- a trỏ đến phần tử a[0][0]

- a+1 trỏ tới đầu hàng thứ hai a[1][0]

…….

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta có thể dùng con trỏ. Nhưng nếu dùng phép gán p = a; là sai cú pháp vì kiểu địa chỉ của p và a là khác nhau. p là con trỏ int còn a là địa chỉ kiểu int[3]. Khi đó ta phải sử dụng phép gán p = (int *)a. Khi đó:

- p trỏ tới a[0][0]

- p + 1 trỏ tới a[0][1]……

- p + i*m + j trỏ tới a[i][j]

Chú ý: Khi muốn truyền tên mảng hai chiều vào hàm thì đối tương ứng phải được khai báo dạng p[][m] trong đó m là số cột của mảng truyền vào. Sau đó trong chương trình khi truy nhập đến các phần tử ta có thể viết p[i][j]

Ví dụ 5.12: Viết chương trình minh họa cách dùng con trỏ để nhập các phần tử của hai ma trận A, B sau đó tính ma trận tích C.

```
#include “stdio.h”
#include”conio.h”
void  nhap_mt(int  a[][20], int  n, int m, char  ten)
{
        int  i, j;
        printf(“Nhap ma tran %c”,ten);
        for( i=0; i<n; i++)
        for(j=0; j <m; j++)
        {
              printf(“\nNhap phan tu a[%d][%d]:”, i, j);
                scanf(“%f”,&a[i][j]);
        }
```

```
}
void  in_mt(int a[][20], int n, int m, char ten)
{
        int  i, j;
        printf("\n In ma tran %c", ten);
        for ( i = 0; i < n; i++);
        {
              for (j = 0; j < m; j++)
               printf("%6.2f",a[i][j]);
               printf("\n");
        }
}
void  main()
{
        float  a[20][20], b[20][20], c[20][20];
        int n, m, p, i, j, k;
        printf("\n Nhap cac so n, m, p =");
        scanf("%d %d %d", &n, &m, &p);
        nhap_mt(a, n, m, 'A');
        nhap_mt(b, m, p, 'B');
        for (i = 0; i < n; i++)
        for ( j =0; j < p; j++)
        {
              c[i][j] = 0;
              for ( k =0; k < m; k++)
              c[i][i] + =a[i][k]*b[k][j];
        }
```

```
        in_mt(a, n, m, 'A');
        in_mt(b, m, p, 'B');
        in_mt(c, n, p, 'C');
        getch();
}
```

5.2.8. Hàm kiểu con trỏ

Là một hàm được xây dựng giống như hàm thông thường nhưng khi khai báo dòng tiêu đề của hàm có thêm dấu sao "*" ở đầu và hàm trả về giá trị là một địa chỉ.

Ví dụ 5.13: Chương trình tìm số lớn nhất của hai số a và b sử dụng hàm kiểu con trỏ.

```
#include"stdio.h"
#include"conio.h"
int   * max(int  *a, int *b)
{
        if ( *a > *b) return a;
        else    return  b;
}
void  main()
{
        int  a = 10, b = 8, *c;
        c = max(&a, &b); //
        printf("Max cua %d va %d la %d", a, b, *c);
        getch();
}
```

Hàm max dùng để tính số lớn nhất của hai số a và b là hàm kiểu con trỏ nên được viết thêm dấu * trước tên hàm, Giá trị trả về của hàm là một địa chỉ kiểu int – đó chính là địa chi của số lớn hơn trong hai số a và b. Trong hàm main, khi gọi

hàm max ta dùng lệnh gán c = max(&a, &b); để gán địa chỉ của số lớn trong hai số a và b cho biến c, do đó biến c phải được khai báo là một con trỏ kiểu int.

5.2.9. Con trỏ tới hàm (Con trỏ hàm)

5.2.9.1. Tác dụng của con trỏ hàm

Trong C, mỗi một hàm được cấp phát một vùng bộ nhớ có địa chỉ chính là địa chỉ đầu của vùng nhớ đó và ta có thể lưu trữ địa chỉ này. Con trỏ hàm dùng để chứa địa chỉ của một hàm nào đó. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có nghĩa thì kiểu hàm và kiểu con trỏ hàm phải tương thích. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm ở bất kỳ vị trí nào.

5.2.9.2. Cách khai báo con trỏ hàm

- Với con trỏ hàm: Như khai báo hàm, nhưng tên hàm có dạng

(* tên hàm)(danh sách đối);

Chú ý: Danh sách đối chỉ có tên kiểu đối chứ không có tên đối.

Ví dụ 5.14: khai báo float (*f)(float) với f là con trỏ hàm kiểu float có một đối kiểu là float. Hàm f dùng để lưu địa chỉ của một hàm có khai báo như sau: float tên_hàm(float x)

- Với mảng con trỏ hàm: Là mảng mà mỗi phần tử là một con trỏ hàm. Mảng con trỏ hàm được khai báo như sau:

(* tên_mảng[n])(danh sách đối);

Ví dụ 5.15: int (*mảng_cth[100])(float, float) khai báo một mảng mà mỗi phần tử là một con trỏ hàm có thể lưu trữ địa chỉ của một hàm có khai báo như sau: int tên_hàm(float x, float y)

Ví dụ 5.16: Chương trình tìm số lớn nhất của hai số a và b sử dụng con trỏ hàm

```
#include”stdio.h”
int max(int a, int b)        //Hàm tính max
{
        return( x>y? x:y); //sử dụng biểu thức điều kiện
}
int (*pmax)(int, int);  //khai báo con trỏ hàm
```

```
void  main()
{
        int  a = 5, b = 9;
        pmax = max; //gán tên hàm max cho con trỏ hàm pmax
        printf("max cua %d va %d la %d", a, b, pmax(a, b));
        //thay vì dùng max(a,b) ta có thể dùng pmax(a, b)
        getch();
}
```

5.2.9.3. Con trỏ tới hàm có kiểu con trỏ.

Được dùng để chứa địa chỉ của một hàm và hàm đó có kiểu là con trỏ.

Ví dụ 5.17: Xét ví dụ minh họa con trỏ tới một hàm có kiểu con trỏ với bài toán tìm số lớn nhất của hai số a và b.

Giả sử hàm max để tìm số lớn của hai số a và b là hàm có kiểu con trỏ như sau:

```
int   *max(int * a, int  *b)
{
      if  (*a > *b) return  a; else  return b;
}
```

Thì khi khai báo con trỏ hàm trỏ tới hàm max phải có thêm dấu * ở đầu như sau:

```
int  *(*pmax)(int, int);
```

Sau đó dùng phép gán:

```
pmax = max;
```

Thì tại mọi nơi trong chương trình dùng pmax có ý nghĩa như dùng max. Khi đó hàm main được viết như sau:

```
void  main()
{
        int  a = 5, b = 9;
```

```
        pmax = max; //gán tên hàm max cho con trỏ hàm pmax
        printf("max cua %d va %d la %d", a, b, *pmax(a, b);
        getch();
}
```

5.2.10. Hàm có đối là con trỏ hàm

C cho phép thiết kế các hàm mà tham số thực sự trong lời gọi tới hàm đó lại là tên của một hàm khác. Khi đó đối tương ứng phải là một con trỏ hàm. Sau đó trong thân hàm, tại vị trí nào xuất hiện tên đối thì ta hiểu đó là tên hàm truyền vào.

Ví dụ 5.18: Chương trình sử dụng hàm có đối là con trỏ hàm để tìm số lớn nhất của ba số a, b, c.

Để tìm số lớn nhất của ba số ta đi tìm số lớn nhất của hai trong ba số đó, sau đó tìm số lớn nhất của kết quả tìm được với số còn lại. Chương trình gồm một hàm tìm số lớn nhất của ba số a, b và c có tên là max_ba_so. Các đối số của hàm là a, b, c và một con trỏ hàm trỏ tới hàm tính số lớn nhất của hai số. Chương trình được viết như sau:

```
#include"stdio.h"
int   max_hai_so(int  b, int  c)
{
        return (a>b? a:b);
}
int max_ba_so(int a, int b, int c, int (*pmax_hai_so)(int, int))
{
        return(a>pmax_hai_so(b, c) ? a : pmax_hai_so(b, c));
}
void  main()
{
        int   x, y, z;
        printf("Nhap gia tri x, y, z = ");
        scanf("%d%d%d", &x, &y, &z);
```

```
        printf("\Gia tri lon nhat cua ba so %d,%d,%d la ", x, y, z);
        printf("%d",max_ba_so(x,y,z, max_hai_so));
        getch();
}
```

5.2.11. Hàm đệ quy

5.2.11.1. Định nghĩa

Trong C cho phép từ một điểm trong thân của một hàm gọi tới chính hàm đó. Hàm như vậy gọi là hàm đệ quy.

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

Có bao nhiêu lần gọi tới hàm thì cũng có bấy nhiêu lần thoát ra khỏi hàm và cứ mỗi lần ra khỏi hàm thì một tập các biến cục bộ bị xóa. Sự tương ứng giữa các lần gọi tới hàm và các lần ra khỏi hàm được thực hiện theo thứ tự ngược lại.

5.2.11.2. Bài toán nào dùng đệ quy

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

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

- Bài toán có thể quy về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Và sau một số hữu hạn bước biến đổi đệ quy sẽ dẫn tới trường hợp suy biến. Ta gọi đây là trường hợp tổng quát.

5.2.11.3. Cách xây dựng hàm đệ quy

Hàm đệ quy thường được viết theo thuật toán sau:

```
if  (trường hợp suy biến)
{
      cách giải(thường đã có cách giải)
}
else
{
```

gọi đệ quy tới hàm với giá trị khác của tham số.

}

Ví dụ 5.19: Viết chương trình tính giai thừa của số n

Ta thấy n! được tính theo công thức truy hồi sau:

n! = 1 nếu n = 0 hoặc n = 1 (trường hợp suy biến)

n! = n*(n-1)! nếu n>0) (gọi đệ quy với giá trị tham số là n – 1)

Chương trình được viết như sau:

```
#include"stdio.n"
#include"conio.h"
long    giai_thua(int  n)
{
        if  (n=0 || n=1)   return  1;
        else  return (n*giai_thua(n – 1));
}
void  main()
{
        int  n;
        printf("Nhap so n = ");  scanf("%d",&n);
        printf("\n Giai thua cua %d la %ld", n, giai_thua(n));
}
```

Ví dụ 5.20: Bài toán tháp Hà nội được đặt như sau: Có một tháp m tầng đạt tại vị trí (x1, y1), cần chuyển đến vị trí mới (x2, y2). Khi chuyển cho phép dùng vị trí (x3, y3) làm vị trí trung gian. Khi chuyển có các quy ước sau:

- Số tầng m >= 1
- Số thứ tự tầng đánh từ 1 đến m từ đỉnh xuống đáy
- Mỗi bước chuyển chỉ được chuyển một tầng từ vị trí này đến vị trí khác
- Tại mọi thời điểm, tầng trên luôn nhỏ hơn tầng dưới nó.

Thuật toán chuyển tháp có thể diễn đạt như sau:

- Trường hợp suy biến: Nếu m = 1, khi đó chỉ cần chuyển tầng 1 từ (x1, y1)

- Trường hợp tổng quát (m>1):

+ chuyển m – 1 tầng tháp ở phía trên từ (x1, y1) đến (x3, y3), dùng (x2, y2) làm vị trí trung gian.

+ chuyển tầng m còn lại từ (x1, y1) đến (x2, y2)

+ chuyển m – 1 tầng tháp ở (x3, y3) đến (x2, y2) dùng (x1, y1) làm vị trí trung gian.

#include”stdio.n”

#include”conio.h”

Chương trình được viết như sau:

```
void  chuyen_thap(int m, int x1, int y1, int x2, int y2, int x3, int y3)
{
        if  ( m == 1)
                printf(“Chuyen tang %d tu (%d, %d) đen (%d, %d)”, m,
                x1, y1, x2, y2);
        else
        {
                chuyen_thap(m-1, x1, y1-1, x3, y3, x2, y2);
                printf(“chuyen tang %d tu (%d, %d) đen (%d, %d)”,
                m, x1, y1, x2, y2);
                chuyen_thap(m-1, x3, y3, x2, y2 -1, x1, y1);
        }
}
void  main()
{
        int st;
        printf(“Nhap so tang cua thap:”); scanf(“%d”, &st);
        chuyen_thap(st, 10, 25, 30, 25, 50, 25);
```

```
        getch();
}
```

Bài tập cuối chương

Câu 1. Viết chương trình sử dụng các hàm để giải hệ phương trình bậc nhất

ax + by = c

dx + ey = f

Câu 2. Viết chương trình sử dụng các hàm để nhập một dãy số, sau đó:

- In dãy số ra màn hình
- In ra màn hình giá trị lớn nhất và giá trị nhỏ nhất của dãy số
- Sắp xếp dãy số theo chiều tăng dần. Sau đó in ra màn hình dãy số sau khi đã sắp xếp.

Câu 3. Viết chương trình sử dụng để thực hiện các công việc sau:

- Nhập vào một dãy số có n phần tử
- Đếm xem trong dãy số có bao nhiêu số nguyên tố

Câu 4. Viết chương trình gồm có một hàm nhập vào các hệ số của một đa thức, một hàm tính giá trị của đa thức, một hàm in các hệ số của đa thức. Sau đó

- Nhập vào các hệ số của hai đa thức: $P=a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0$; $Q=b_mx^m + b_{m-1}x^{m-1} + \ldots + b_1x + b_0$.
- Nhập giá trị x sau đó in ra màn hình giá trị của hai đa thức P và Q.
- Tính hệ số của đa thức tổng T = P + Q;
- In ra màn hình hệ số của ba đa thức.

Câu 5. Viết chương trình sử dụng hàm để

- Nhập hai ma trận A_{m*n} và B_{n*p}
- Tính ma trận C_{m*p} là tích của hai ma trận A và B
- In ba ma trận ra màn hình.

Câu 6. Viết chương trình tính các hàm số sau đây bằng phương pháp đệ quy

1. $f(x, n) = x^n$
2. $s(n) = (2n)!!$

Câu 7. Có n người và n việc. Biết nếu người i làm việc j thì đạt hiệu quả a[i][j]. Hãy lập phương án phân công mỗi người một việc sao cho tổng hiệu quả là lớn nhất

Yêu cầu đối với bài toán trên:

- xây dựng các hàm ứng với bài toán đề ra
- viết thêm hàm main để sử dụng các hàm này
- chạy máy theo các dữ liệu tự chọn.

Câu 8: Làm lại các bài tập ở chương 4 trong đó có sử dụng các hàm.

CHƯƠNG 6: KIỂU CẤU TRÚC, KIỂU HỢP

Để lưu trữ và xử lý thông tin trong máy tính ta có các biến và các mảng. Mỗi biến chỉ lưu được một giá trị. Mảng là một tập hợp nhiều biến có cùng một kiểu giá trị và cùng tên mảng. Cấu trúc có thể xem như một sự mở rộng của các khái niệm biến và mảng, nó cho phép lưu trữ và xử lý các dạng thông tin phức tạp hơn. Khái niệm cấu trúc trong C có nhiều nét tương tự như khái niệm bản ghi trong pascal hay foxpro. Cấu trúc là một công cụ mạnh để mô tả và xử lý các cấu trúc dữ liệu phức tạp trong các bài toán quản lý, điển hình như bài toán quản lý sinh viên,khi đó mỗi sinh viên được xem như một cấu trúc gồm các thành phần như họ tên, quê quán, tuổi, địa chỉ,…

6.1. Kiểu cấu trúc

6.1.1. Định nghĩa kiểu cấu trúc

Khi định nghĩa một kiểu cấu trúc ta cần chỉ ra: tên của kiểu cấu trúc và các thành phần của nó. Việc này được thực hiện theo mẫu sau:

Mẫu 1:

```
struct    tên kiểu cấu trúc
{
        Khai báo các thành phần
};
```

trong đó thành phần của cấu trúc có thể là biến, mảng hoặc một cấu trúc khác mà kiểu của nó đã được định nghĩa từ trước.

Ví dụ 6.1:

```
struct   que_quan
{
        char   xa[20], huyen[20], tinh[20];
 } ;
```

Thiết kế một kiểu cấu trúc có tên là que_quan gồm ba thành phần: xa, huyen, tinh đều có cùng kiểu dữ liệu là kiểu mảng ký tự.

```
struct   sinh_vien
{
```

```
        char  ho_ten[30];
        int  tuoi;
        float diem;
        struct   que_quan     dia_chi;
} ;
```

Thiết kế một kiểu cấu trúc có tên là sinh_viên gồm bốn thành phần: thành phần thứ nhất có tên là ho_ten là một mảng char, thành phần thứ hai là tuoi có kiểu là int, thành phần thứ ba là diem có kiểu là float và thành phần cuối cùng là dia_chi, là một cấu trúc có kiểu là que_quan được định nghĩa ở trước đó.

Chú ý: Có thể dùng toán tử typedef để định nghĩa các kiểu cấu trúc như sau:

Mẫu 2:

```
typedef   struct
{
        Khai báo các thành phần
}  tên kiểu cấu trúc;
```

Ví dụ 6.2. Các kiểu cấu trúc que_quan và sinh_vien ở trên có thể định nghĩa như sau:

```
typedef   struct
{
        char   xa[20], huyen[20], tinh[20];
} que_quan;
typedef   struct
{
        char  ho_ten[30];
        int  tuoi;
        float diem;
        que_quan     dia_chi;
} sinh_vien;
```

6.1.2. Khai báo biến cấu trúc

Khai báo biến cấu trúc được thực hiện theo các mẫu sau:

Mẫu 1:

struct tên kiểu cấu trúc danh sách tên biến cấu trúc;

Ví dụ 6.3: struct sinh_vien sv1, sv2; sẽ cho ta hai biến cấu trúc sv1 và sv2. Cả hai đều được xây dựng theo kiểu sinh_viên đã được định nghĩa ở ví dụ 6.1 trên.

Mẫu 2: Cho phép vừa thiết kế kiểu cấu trúc vừa khai báo biến cấu trúc

```
struct    tên kiểu cấu trúc
{
        Khai báo các thành phần
} danh sách tên biến cấu trúc;
```

Ví dụ 6.4: Các biến cấu trúc sv1 và sv2 có thể được xây dựng theo cách sau:

```
struct   sinh_vien
{
        char  ho_ten[30];
        int  tuoi;
        float diem;
        struct    que_quan     dia_chi;
} sv1, sv2;
```

trong đó kiểu cấu trúc que_quan được định nghĩa như trong ví dụ 6.1.

Mẫu 3: Vừa định nghĩa kiểu cấu trúc vừa khai báo biến cấu trúc như trên, ta có thể không cần chỉ ra tên kiểu cấu trúc như sau

```
struct
{
        Khai báo các thành phần
} danh sách tên biến cấu trúc;
```

Ví dụ 6.5: Các cấu trúc sv1, sv2 có thể khai báo như sau:

```
struct
{
        char  ho_ten[30];
        int  tuoi;
        float diem;
        struct  que_quan     dia_chi;
} sv1, sv2;
```

với kiểu cấu trúc que_quan được định nghĩa như trong ví dụ 6.1.

Sự khác nhau giữa mẫu 2 và mẫu 3: ở mẫu 2 ngoài việc xây dựng được các biến cấu trúc ta còn tạo ra được kiểu cấu trúc. Kiểu này có thể được sử dụng để khai báo các biến cấu trúc khác. Còn mẫu 3 thì chỉ khai báo các biến cấu trúc, tức là chỉ thực hiện được một phần công việc của mẫu 2.

Chú ý: Nếu dùng typedef để định nghĩa kiểu cấu trúc, thì khi khai báo biến cấu trúc có kiểu đã định nghĩa đó thì ta chỉ cần dùng tên kiểu (bỏ từ khóa struct). Ví dụ nếu kiểu cấu trúc que_quan được định nghĩa như trong ví dụ 6.2 thì biến cấu trúc dịa_chi có thể khai báo trong các ví dụ 6.4 và 6.5 như sau:

```
struct   sinh_vien
{
        char  ho_ten[30];
        int  tuoi;
        float diem;
        que_quan     dia_chi;
} sv1, sv2;
```

6.1.3. Truy nhập tới các thành phần của cấu trúc

Từ các chương đầu, ta đã khá quen với việc sử dụng các biến, các phần tử mảng trong các câu lệnh. Các thành phần của một cấu trúc đóng vai trò như là biến, phần tử mảng. Do đó phép toán nào thực hiện được trên các biến, phần tử mảng thì cũng thực hiện được trên các thành phần đó. Câu lệnh nào dùng cho biến, phần tử mảng thì cũng có thể dùng được cho các thành phần của cấu trúc. Để truy nhập đến các thành phần của cấu trúc ta sử dụng một trong các cách sau:

tên cấu trúc1.tên cấu trúc2...tên cấu trúcn.tên thành phần

Ví dụ 6.6: Biến cấu trúc sv1 được khai báo ở ví dụ 6.4 có các thành phần sau:

sv1.ho_ten

sv1.tuoi

sv1.dia_chi.xa

sv1.dia_chi.huyen

sv1.dia_chi.tinh

Chú ý: Để truy nhập tới các thành phần của cấu trúc ta không những phải chỉ ra tên thành phần đó mà còn phải liệt kê tên các cấu trúc chứa thành phần này. Điều này gây ra sự mất công và tẻ nhạt khi viết chương trình. Ta có thể rút gọn việc truy nhập đến các thành phần cấu trúc bằng cách dùng chỉ thị #define như sau:

#define tên tên cấu trúc1.tên cấu trúc2...tên cấu trúcn

Khi đó, truy nhập đến các thành phần của một biến cấu trúc ta không cần phải chỉ ra tên cấu trúc1.tên cấu trúc2...tên cấu trúcn mà thay vào đó ta chỉ viết là tên.

Ví dụ 6.7: Sử dụng #define

#define ht sv1.ho_ten

#define t sv1.tuoi

#define đc sv1.dia_chi

Khi đó nếu viết ht tức là viết sv1.ho_ten, viết t tức là viết sv1.tuoi, tương tự đc.xã tương đương với sv1.dia_chi.xa

6.1.4. Sử dụng cấu trúc

Cách sử dụng kiểu cấu trúc cũng như các kiểu dữ liệu số (int, float…) chỉ khác ở những điểm sau:

- Để nhập dữ liệu cho các thành phần của cấu trúc ta thực hiện giống như nhập dữ liệu cho các biến/mảng .

- Khi có con trỏ cấu trúc trỏ tới một đối tượng cấu trúc thì ta có thể truy nhập đến các thành phần của cấu trúc thông qua con trỏ như sau:

Cách 1: tên con trỏ -> thành phần

Cách 2: (*tên con trỏ).thành phần

Ví dụ 6.8:

Giả sử ta có khai báo struct sinh_vien sv, *p;

sau đó dùng phép gán p = &sv;

khi đó các cách viết sau là tương đương:

sv1.ho_ten tương đương với p ->ho_ten hay (*p).ho_ten

Ví dụ 6.9: Viết chương trình nhập vào thông tin về một sinh viên gồm các thành phần họ tên, tuổi, điểm trung bình, địa chỉ. Sau đó in thông tin về sinh viên ra màn hình.

```
#include “stdio.h”
#include “conio.h”
void   main()
{
        typedef   struct
        {
                char   xa[20], huyen[20], tinh[20];
         } que_quan;
        struct   sinh_vien
        {
                char       ho_ten[30];
                int  tuoi;
                float diem;
                que_quan     dia_chi;
        } sv;
          float  x;
          printf(“Nhap thong tin ve sinh viên\n”);
          printf(“Nhap ho ten sinh viên\n”);
          fflush(stdin);
          gets(sv.ho_ten);
```

```
        printf("Nhap tuoi sinh vien\n");
        scanf("%d", &sv.tuoi);
        printf("Nhap diem trung binh sinh vien\n");
        scanf("%f",&sv.diem);
        printf("Nhap dia chi sinh vien\n");
        printf("Nhap xa:");          gets(sv.dia_chi.xa);
        printf("Nhap huyen:");       gets(sv.dia_chi.huyen);
        printf("Nhap tinh:");        gets(sv.dia_chi.tinh);
        printf("In thông tin ve sinh vien ra man hinh\n");
        printf("Ho ten \t tuoi \t  diem trung binh  \t dia chi");
        printf("%6s\t%d\t%f\t %s %s %s",sv.ho_ten, sv.tuoi,
        sv.diem_tb, sv.dia_chi.xa, sv.dia_chi.huyen, sv.dia_chi.tinh);
        getch();
}
```

6.1.5. Mảng cấu trúc

Là một mảng mà mỗi phần tử của mảng có kiểu là kiểu cấu trúc. Giả sử kiểu cấu trúc sinh_viên đã được định nghĩa ở tên. Khi đó khai báo:

```
struct   sinh_vien   sv, danh_sach[100];
```

sẽ cho một biến cấu trúc sv kiểu sinh_vien và một mảng cấu trúc danh_sach. Mảng danh_sach gồm 100 phần tử, mỗi phần tử là một cấu trúc kiểu sinh_vien.

Chú ý: không cho phép sử dụng phép toán lấy địa chỉ đối với các thành phần thực của mỗi phần tử trong mảng cấu trúc. Chẳng hạn không cho phép viết &danh_sách[i].diem nếu kiểu của diem là thực còn nếu kiểu của diem là nguyên thì cho phép.

Ví dụ 6.10: Viết chương trình nhập vào thông tin về một lớp gồm n sinh viên, mỗi sinh viên là một cấu trúc gồm các thành phần họ tên, tuổi, điểm trung bình, địa chỉ. Sau đó in thông tin về sinh viên ra màn hình.

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

```
void   main()
{
        struct   sinh_vien
        {
                char  ho_ten[30], dia_chi[30];
                int  tuoi;
                float điem;
        } lop[100];
        float  x;
        int  n,i;
        printf(“Nhap so sinh vien:”);
        scanf(“%d”,&n);
        for (i = 1;i<=n;i++)
        {
                printf(“Nhap thông tin ve sinh vien thu %d\n”,i);
                printf(“Nhap ho ten sinh vien\n”);
                fflush(stdin);
                gets(lop[i].ho_ten);
                printf(“Nhap tuoi sinh vien\n”);
                scanf(“%d”, &lop[i].tuoi);
                printf(“Nhap diem trung binh sinh vien\n”);
                 scanf(“%f”,&lớp[i].diem);
                printf(“Nhap dia chi sinh vien\n”);
                gets(lop[i].dia_chi);
        }
        printf(“In thông tin ve sinh vien ra man hinh\n”);
        printf(“ Ho ten \t Tuoi \t Diem \t Dia chi\n”);
```

```
        for (i = 1;i<=n;i++)
        {
            printf("%s \t %d\t  %f\t%s\n",  lop[i].ho_ten,  lop[i].tuoi,
            lop[i].điem, lop[i].dia_chi);
        }
        getch();
}
```

6.1.6. Khởi đầu cho một cấu trúc và phép gán cấu trúc

Có thể khởi đầu cho một cấu trúc bằng cách viết vào sau khai báo của chúng một danh sách các giá trị cho các thành phần .

Ví dụ 6.11: xét đoạn khai báo

```
struct  sinh_vien    sv ={"tran van A", 20, 6.5, "Ha noi"};
```

Xác định một cấu trúc có tên là sv kiểu là sinh_viên đã được định nghĩa ở ví dụ 6.1 và khởi đầu cho các thành phần của cấu trúc này. Như vậy sv.họ_ten là "tran van A", sv.tuoi = 20, sv.diem = 6.5 và sv.dia_chi = "Ha noi".

Có thể thực hiện phép gán trên các biến và phần tử mảng cấu trúc cùng kiển. Khi đó mỗi thành phần của cấu trúc này sẽ được gán cho thành phần tương ứng của cấu trúc kia. Chẳng hạn ta có thể viết sv1 = sv2, ở đây sv1, sv2 là các cấu trúc có cùng kiểu sinh_viên. Phép gán hai cấu trúc tỏ ra rất tiện lợi khi ta cần sắp xếp lại một mảng cấu trúc theo một trật tự mới.

Ví dụ 6.12: Đoạn chương trình sau sắp xếp danh sách sinh viên trong ví dụ trên theo thứ tự tăng dần của điểm trung bình

```
for (i=1;i<=n-1;i++)
for(j=i+1; j<=n; j++)
 if (lop[i].diem > lop[j].diem)
 {
        tg = lop[i];
        lop[i] = lop[j];
        lop[j] = tg;
 }
```

6.2. Kiểu Hợp

6.2.1. Định nghĩa kiểu hợp (union)

Cũng như cấu trúc, union gồm nhiều thành phần, nhưng chúng khác nhau ở chỗ: các thành phần của cấu trúc có những vùng nhớ khác nhau, còn các thành phần của union được cấp phát một vùng nhớ chung. Độ dài của union bằng độ dài của thành phần lớn nhất

6.2.2. Khai báo biến kiểu hợp

Việc định nghĩa một kiểu union, khai báo union, mảng union, con trỏ union và cách truy nhập đến các thành phần của union được thực hiện hoàn toàn giống như đối với cấu trúc. Một cấu trúc có thể có thành phần kiểu union và ngược lại các thành phần của union lại có thể là cấu trúc.

Ví dụ 6.13: Minh họa việc khai báo union

```
typedef  union
{
        unsigned  int    x;
        float  y
        char    ch[2];
}  data;
data   a,b;
```

ví dụ trên định nghĩa kiểu union data gồm ba thành phần là x, y và ch. Độ dài của data bằng độ dài của y và bằng 4. Sau đó khai báo các biến union là a và b.

phép gán a.x = 0xa1b2; sẽ gán một số hệ 16 cho x. Do x và ch cùng chiếm 2 byte đầu của union, nên sau phép gán trên ta cũng sẽ có a.ch[0] = 0xb2 và a.ch[1] = 0xa1. Như vậy ta đã dùng union để trích ra các byte của một số nguyên.

Ví dụ 6.14: minh họa khai báo một union có thành phần cấu trúc

```
struct  dia_chi
{
        int   so_nha;
        char    *pho;
}   ;
```

```
struct    ng
{
        int  ngay;
        int  thang;
        int  nam;
};
union    u
{
        struct   ng   date;
        struct   dia_chi    address;
}     diachi_ngaysinh;
```

Ví dụ 6.15: Viết chương trình nhập danh sách các thí sinh thi đại học của cả hai khối A và C. Sau đó in ra màn hình danh sách thí sinh thi khối A riêng và danh sách thí sinh thi khối C riêng.

```
#include”stdio.h”
#include”conio.h”
void    main()
{
        typedef   struct
        {
              float  toan, ly, hoa;
        } khoi_A;
        typedef   struct
        {
              float  van, su, dia;
        } khoi_C;
        typedef   struct
        {
```

```
        char  ho_ten[20], dia_chi[20], ten_khoi;
        union
        {
                khoi_A    kA;
                khoi_C     kC;
         }  khoi;
}  thi_sinh;
thi_sinh  ds[1000];
float  m1, m2, m3;
int   i, n;
printf("Nhap so sinh vien:");
scanf("%d",&n);
for (i=1; i <=n; i++)
{
        fflush(stdin);
        printf("Nhap thong tin ve thi sinh thu %d\n",i);
        printf("Nhap ho ten thi sinh\n");
        gets(ds[i].ho_ten);
        printf("Nhap dia chi thi sinh \n");
        gets(ds[i].dia_chi);
        printf("Nhap ten khoi cua thi sinh \n");
        fflush(stdin);
        ds[i].ten_khoi = getchar();
        if  (ds[i].tên_khoi == 'A')
       {
                printf("Nhap diem toan, ly, hoa:");
                scanf("%f%f%f",&m1,&m2,&m3);
```

```
                ds[i].khoi.kA.toan = m1;
                ds[i].khoi.kA.ly = m2;
                ds[i].khoi.kA.hoa = m3;
        }
        else   if  (ds[i].ten_khoi == 'C')
        {
                printf("Nhap diem van, su, dia:");
                scanf("%f%f%f",&m1,&m2,&m3);
                ds[i].khoi.kC.van = m1;
                ds[i].khoi.kC.su = m2;
                ds[i].khoi.kC.dia = m3;
        }
}
printf("Danh sach ket qua thi sinh thi khoi A\n");
printf(" Ho ten \t dia chi \t diem toan \t  diem ly \t diem hoa\n");
for (i=1; i <=n; i++)
{
        if  (ds[i].ten_khoi == 'A')
        printf("%-6s  \t %-7s \t %-9.2f   \t %-7.2f  \t
                %8.2f\n",ds[i].ho_ten,ds[i].dia_chi,ds[i].khoi.ka.toan,
                 ds[i].khoi.ka.ly, ds[i].khoi.ka.hoa);
}
printf("Danh sach ket qua thi sinh thi khoi C\n");
printf(" Ho ten \t dia chi \t diem van \t  diem su \t diem dia\n");
for (i=1; i <=n; i++)
{
        if  (ds[i].ten_khoi == 'C')
```

```
                printf("%-6s\t%-7s\t%-8.2f\t %-7.2f  \t
                    %-8.2f\n",ds[i].ho_ten,ds[i].dia_chi,ds[i].khoi.kc.van,
                    ds[i].khoi.kc.su, ds[i].khoi.kc.dia);
        }
    getch();
}
```

6.3. Cấu trúc tự trỏ và danh sách liên kết

6.3.1. Cấp phát bộ nhớ động

Giả sử ta cần quản lý một danh sách lớp gồm nhiều sinh viên. Khi viết chương trình ta chưa biết có bao nhiêu sinh viên. Nếu sử dụng mảng (cấp phát bộ nhớ tĩnh) để lưu trữ danh sách lớp thì ta phải khai báo tối đa số sinh viên có thể có. Như vậy sẽ có rất nhiều vùng nhớ được cấp phát mà không bao giờ dùng đến, điều này gây lãng phí vùng nhớ. Do đó, phần này minh họa phương pháp cấp phát bộ nhớ động. Số sinh viên sẽ đúng bằng số sinh viên có thực. Như vậy có bao nhiêu sinh viên thì sẽ có số vùng nhớ tương ứng đủ để lưu trữ từng ấy sinh viên.

Các hàm cấp phát bộ nhớ động thuộc thư viện alloc.h như sau:

- Hàm void *calloc(unsigned n, unsigned size); sẽ cấp phát một vùng nhớ có kích thước là (n * size) byte, tức là có n đối tượng, mỗi đối tượng có size byte. Nếu thành công thì hàm trả về địa chỉ đầu của vùng nhớ được cấp phát. Khi không đủ bộ nhớ hàm trả về giá trị NULL.

- Hàm void *malloc(unsigned n); thực hiện như hàm calloc nhưng chỉ cấp phát vùng nhớ có độ dài là n byte. Nếu thành công thì hàm trả về địa chỉ đầu của vùng nhớ được cấp phát. Khi không đủ bộ nhớ hàm trả về giá trị NULL.

- Hàm free(con_trỏ); để giải phóng vùng nhớ được cấp phát do con_trỏ trỏ tới

6.3.2. Cấu trúc tự trỏ và danh sách liên kết

6.3.2.1. Cấu trúc tự trỏ

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

Cách khai báo cấu trúc tự trỏ như sau:

Mẫu 1: typedef struct tên cấu trúc

```
{
        Danh sách các thành phần
        struct  tên cấu trúc    *tên con trỏ;
} tên kiểu cấu trúc;
```

Mẫu 2: typedef struct tên cấu trúc tên kiểu cấu trúc;

```
struct  tên cấu trúc
{
        Danh sách các thành phần
        Tên kiểu cấu trúc    *tên con trỏ;
} ;
```

Mẫu 3: struct tên cấu trúc

```
{
        Danh sách các thành phần
        struct  tên cấu trúc    *tên con trỏ;
};
typedef     tên cấu trúc     tên kiểu cấu trúc;
```

Ví dụ 6.16: Với chương trình về việc nhập thông tin về danh sách lớp sinh viên ta khai báo kiểu cấu trúc tự trỏ p_sinh_vien như sau

```
typedef  struct  sv
{
        char  ho_ten[20];
        char   dia_chi[20];
        struct  sv  *tiep;
} p_sinh_vien;
```

6.3.2.2. Danh sách liên kết

Cấu trúc tự trỏ được dùng để xây dựng danh sách liên kết. Danh sách liên kết là một danh sách gồm nhiều phần tử và các phần tử được móc nối (liên kết) lại với nhau. Mỗi phần tử có kiểu cấu trúc tự trỏ gồm hai thành phần: thành phần thứ nhất

chứa thông tin và thành phần thứ hai chứa địa chỉ của cấu trúc tiếp theo trong danh sách. Có thể có hai loại danh sách liên kết: danh sách liên kết thuận và danh sách liên kết ngược, được định nghĩa như sau:

1. Danh sách liên kết thuận. Đó là một nhóm các cấu trúc có các tính chất sau:

- Biết được địa chỉ cấu trúc đầu (nằm về phía đầu danh sách) đang được lưu trữ trong một con trỏ nào đó (thường đặt tên là p_dau), địa chỉ cấu trúc cuối (nằm về phía cuối danh sách) được lưu trữ trong một con trỏ nào đó (thường đặt tên là p_cuoi)

- Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp theo trong danh sách

- Cấu trúc cuối chứa hằng NULL.

Với danh sách liên kết này ta có thể lần lượt truy cập danh sách từ cấu trúc đầu tới cấu trúc cuối.

2. Danh sách liên kết ngược. Là một nhóm các cấu trúc cũng có các tính chất trên nhưng theo chiều ngược lại

- Biết địa chỉ cấu trúc cuối (nằm về phía đầu danh sách) được lưu trữ trong một con trỏ nào đó (đặt tên con trỏ là p_ds)

- Trong mỗi cấu trúc (trừ cấu trúc đầu) chứa địa chỉ của cấu trúc trước đó.

- Cấu trúc đầu (nằm về phía cuối danh sách) chứa hằng NULL

Với danh sách này, ta có thể lần lượt truy cập danh sách từ cấu trúc cuối tới cấu trúc đầu.

Ngoài ra ta có thể xây dựng các danh sách mà mỗi phần tử chứa hai địa chỉ: địa chỉ cấu trúc trước và địa chỉ cấu trúc sau. Với loại danh sách này ta có thể truy cập danh sách từ trên xuống dưới theo chiều thuận hoặc từ dưới lên trên theo chiều ngược.

6.3.3. Các phép toán trên danh sách liên kết

6.3.3.1. Khởi tạo danh sách.

*** Với danh sách liên kết thuận.**

Danh sách rỗng, do đó con trỏ p_dau có giá trị là hằng NULL, ta dùng phép gán: p_dau = NULL;

Sau khi đã khởi tạo danh sách, để thêm lần lượt từng phần tử vào trong danh sách ta thực hiện các bước sau:

1. Nhập các thông tin

2. Xin cấp phát vùng nhớ để chứa phần tử (mỗi phần tử là một cấu trúc có kiểu đã được định nghĩa), vùng nhớ này có độ lớn là độ lớn của kiểu cấu trúc đã định nghĩa trên. Và phần tử này do một con trỏ p cùng kiểu với kiểu của phần tử trỏ vào.

3. Gán giá trị của các thông tin đã nhập ở bước 1 vào các thành phần của phần tử do con trỏ p trỏ vào.

4. Nối phần tử này vào trong danh sách với chú ý rằng phần tử cần nối vào danh sách được con trỏ p trỏ vào. Khi nối phần tử vào ta chia hai trường hợp sau:

- Nối phần tử đầu tiên vào danh sách (trước khi nối phần tử đầu tiên vào danh sách thì danh sách đang rỗng, tức là con trỏ p_dau = NULL). Khi đó việc nối phần tử được thực hiện theo các bước như sau:

+ Vì phần tử này sẽ là phần tử đầu danh sách nên nó sẽ được con trỏ p_dau trỏ vào, tức là p_dau = p;

+ Trong danh sách hiện giờ chỉ có một phần tử do đó nó vừa là phần tử đầu tiên và cũng vừa là phần tử cuối cùng nên nó sẽ được con trỏ p_cuoi trỏ vào, tức là p_cuoi = p;

+ Theo định nghĩa danh sách liên kết thuận, phần tử cuối chứa hằng NULL, do đó p_cuoi -> tiep = NULL; (hoặc p -> tiep = NULL;)

- Nối phần tử thứ hai trở đi vào trong danh sách (lúc này con trỏ p_dau trỏ vào phần tử đầu tiên trong danh sách do đó p_dau != NULL). Giả sử trong danh sách đang có n phần từ (n >=1) và phần tử cuối cùng của danh sách do con trỏ p_cuoi trỏ vào. Khi đó việc nối phần tử tiếp theo được thực hiện theo các bước sau:

+ Nối phần tử này vào ngay sau phần tử cuối của danh sách. Tức là cho phần tử cuối của danh sách chứa địa chỉ của phần tử này. Ta dùng phép gán p_cuoi -> tiep = p;

+ Phần tử vừa được nối vào danh sách bây giờ trở thành phần tử cuối cùng trong danh sách, do đó con trỏ p_cuoi trỏ vào nó, tức là p_cuoi = p;

+ Theo định nghĩa danh sách liên kết thuận, phần tử cuối chứa hằng NULL, do đó p_cuoi -> tiep = NULL; (hoặc p -> tiep = NULL;)

*** Với danh sách liên kết ngược**

Danh sách rỗng, do đó con trỏ p_ds có giá trị là hằng NULL, ta dùng phép gán: p_ds = NULL;

Sau khi đã khởi tạo danh sách, để thêm lần lượt từng phần tử vào trong danh sách ta thực hiện các bước sau:

1. Nhập các thông tin

2. Xin cấp phát vùng nhớ để chứa phần tử (mỗi phần tử là một cấu trúc có kiểu đã được định nghĩa), vùng nhớ này có độ lớn là độ lớn của kiểu cấu trúc đã định nghĩa trên. Và phần tử này do một con trỏ p cùng kiểu với kiểu của phần tử trỏ vào.

3. Gán giá trị của các thông tin đã nhập ở bước 1 vào các thành phần của phần tử do con trỏ p trỏ vào.

4. Nối phần tử vào trong danh sách. Giả sử trong danh sách đang có n phần tử (n>=0) và phần tử cuối cùng đang được con trỏ p_ds trỏ vào, khi đó việc nối phần tử vào danh sách thực hiện như sau:

- Cho phần tử này chứa địa chỉ của phần tử cuối cùng danh sách, tức là p -> tiep = p_ds;
- Phần tử vừa được thêm vào bây giờ là phần tử cuối cùng danh sách, do đó p_ds = p;

6.3.3.2. Duyệt các phần tử trong danh sách

Để duyệt qua tất cả các phần tử của một danh sách ta dùng một con trỏ p chứa địa chỉ cấu trúc đang xét.

- Đầu tiên cho p = p_dau; (nếu là danh sách liên kết thuận) hay p = p_ds; (nếu là danh sách liên kết ngược)

- Để chuyển đến phần tử tiếp theo ta dùng phép gán p = p->tiep;

- Dấu hiệu để biết phần tử cuối là p->tiep = NULL

6.3.3.3. Tìm kiếm một phần tử trong danh sách

Để tìm xem trong danh sách có một phần tử cho trước x hay không ta dùng một con trỏ tên là p_tim

- Đầu tiên cho p_tim = p_dau; (nếu là danh sách liên kết thuận) hay p_tim = p_ds; (nếu là danh sách liên kết ngược)

- Kiểm tra điều kiện nếu (p_tim != NULL)&&(p_tim.thành phần != x) (so sánh thành phần của phần tử đang xét với giá trị x) có giá trị đúng thì chuyển đến phần tử tiếp theo bằng câu lệnh gán p = p_tiep.

- Việc tìm kiếm dừng lại khi hoặc p->tim = NULL tức là không có phần tử x trong danh sách hoặc thành phần của phần tử do con trỏ p_tìm trỏ vào bằng giá trị x (p_tim.thành_phần = x), tức là phần tử x có trong danh sách.

6.3.3.4. Xóa một phần tử khỏi danh sách

Ta thực hiện các bước sau:

- Thực hiện tìm kiếm xem trong danh sách có phần tử cần xóa hay không bằng phép toán tìm kiếm ở trên. Nếu không có thì kết thúc công việc, ngược lại nếu có phần tử thì ta thực hiện tiếp các công việc phía sau:

- Lưu trữ địa chỉ phần tử cần xóa vào một con trỏ, giả sử tên là con trỏ p. Có hai trường hợp xảy ra:

+ Phần tử cần xóa nằm ở đầu danh sách (tức là p = p_dau với danh sách liên kết thuận hoặc p = p_ds với danh sách liên kết ngược). Khi đó ta gán p_dau = p_dau -> tiep; (hoặc p_ds = p_ds -> tiep;)

+ Phần tử cần xóa nằm trong danh sách. Khi đó ta sửa để phần tử trước đó có địa chỉ của phần tử đứng sau phần tử mà ta cần xóa như sau:

* Đi đến phần tử đứng trước phần tử cần xóa trong danh sách, giả sử địa chỉ của phần tử này được lưu trong con trỏ q

* Cho con trỏ q->tiep lưu giữ địa chỉ của phần tử đứng sau phần tử cần xóa, tức là q->tiep = p->tiep;

- Giải phóng bộ nhớ của phần tử cần xóa.

6.3.3.5. Chèn một phần tử vào danh sách. Có hai kiểu chèn đó là chèn phần tử vào sau một phần tử trong danh sách và kiểu thứ hai là chèn phần tử vào trước một phần tử trong danh sách. Cụ thể như sau:

* Giả sử cần chèn phần tử A vào sau phần tử B trong danh sách, các bước chèn như sau:

- Cấp phát bộ nhớ và nhập thông tin của phần tử A, giả sử địa chỉ của A được lưu giữ bởi con trỏ q

- Thực hiện tìm kiếm xem trong danh sách có phần tử B hay không? Nếu không có B thì thông báo ra màn hình là không có phần tử B và kết thúc việc chèn sau.

Ngược lại, nếu có phần tử B trong danh sách và con trỏ p lưu giữ địa chỉ của phần tử B thì ta nối phần tử A vào trước phần tử đang đứng sau phần tử B trong danh sách bằng phép gán q -> tiep = p -> tiep (hay nói cách khác phần tử A chứa địa chỉ của phần tử đang đứng sau phần tử B trong danh sách) sau đó cho con trỏ p-> tiep lưu giữ địa chỉ của phần tử A, hay nói cách khác p->tiep = q;

* Giả sử cần chèn phần tử A vào trước phần tử B trong danh sách, các bước chèn như sau:

- Cấp phát bộ nhớ và nhập thông tin của phần tử A, giả sử địa chỉ của A được lưu giữ bởi con trỏ q

- Thực hiện tìm kiếm xem trong danh sách có phần tử B hay không? Nếu không có B thì thông báo ra màn hình là không có phần tử B và kết thúc việc chèn sau. Ngược lại, nếu có phần tử B trong danh sách và con trỏ p lưu giữ địa chỉ của B thì:

+ Đi đến phần tử đứng trước phần tử B trong danh sách, giả sử gọi đó là phần tử C và C do con trỏ r trỏ tới.

+ Khi đó để chèn A vào trước B tức là chèn A vào sau C, tức là ta thực hiện 2 phép gán q -> tiep = r -> tiep; (hoặc q -> tiep = p;) và r->tiep = q;

Ví dụ 6.17:Viết chương trình tạo một danh sách liên kết thuận của một lớp sinh viên sau đó thực hiện các công việc sau:

- In danh sách ra màn hình

- Chèn thêm một sinh viên vào sau một sinh viên có tên nhập từ bàn phím

- Xóa một sinh viên có tên nhập từ bàn phím ra khỏi danh sách

```
#include”stdio.h”
#include”`conio.h”
#include”alloc.h”
#include”string.h”
typedef   struct   sinh_viên
{
        char  ho_ten[20];
        float   diem_tb;
        struct    sinh_vien   *tiep;
```

```
} p_sv;
p_sv   *p_dau, *p_cuoi, *p;
void  tao_ds()
{
        char  ht[20], chon;
        int stt = 0; float  x;
        printf("Nhap danh sach sinh vien\n");
        p_dau = NULL;
        do {
              fflush(stdin);
              printf("Nhap thong tin ve sinh vien thu %d", ++stt);
              printf("Nhap ho ten:");
              gets(ht);
              printf("Nhap diem trung binh:");
              scanf("%f",&x);
              p = (p_sv *)malloc(sizeof(p_sv));
              strcpy(p ->ho_ten, ht);
              p -> diem_tb = x;
              if  (p_dau == NULL)
              {
                    p_dau = p;
                    p_cuoi = p;
             }
             else
             {
                    p_cuoi ->tiep = p;
                    p_cuoi = p;
```

```
                }
        p->tiep = NULL;
        printf("Co nhap nua khong?"):
        fflush(stdin);
        chon = getchar();
        }  while (chon != 'k' );
}
void  hien_ds()
{
         int   stt=0;
         printf("Danh sach sinh vien\n");
         printf("stt              Ho ten                    Diem trung binh\n");
         p = p_dau;
         while (p != NULL)
         {
                printf("%d        %s        %6.2f\n",++stt, p -> ho_ten,
                  p>diem_tb);
                p = p->tiep;
         }
        getch();
}
void   chen_pt()
{
         p_sv   *p_tim;    float   x;
         char    ht[20];
         p = (p_sv *)malloc(sizeof(p_sv));
         printf("Nhap ho ten sinh vien can chen:");
```

```
        fflush(stdin);
        gets(p->ho_ten);
        printf("Nhap diem trung binh:");
        scanf("%f",&x);
        p -> diem_tb = x;
        p ->tiep = NULL;
        printf("Muon chen sau sinh vien nao:");
        fflush(stdin);
        gets(ht);
        p_tim = p_dau;
        while  ((p_tim != NULL) && (strcmpi(p_tim->ho_ten,ht)))
                p_tim = p_tim ->tiep;
        if  (p_tim == NULL)
              printf("Khong tim thay vi tri can chen");
        else
        {
                if (p_tim -> tiep ==  NULL)   p_tim ->tiep = p;
                else
                {
                        p -> tiep = p_tim -> tiep;
                        p_tim -> tiep = p;
                }
        printf("Da chen xong");
      }
      getch();
}
void   xoa_pt()
```

```
{
        p_sv   *p_tim; *p_truoc;   char   ht[20];
        printf("Nhap ho ten sinh vien can xoa:");
        fflush(stdin);
        gets(ht);
        p_tìm = p_dau;
        while  ((p_tim != NULL) && (strcmpi(p_tim->ho_ten,ht)))
        {
              p_truoc = p_tim;
              p_tim = p_tim ->tiep;
        }
        if (p_tim == NULL)
              printf("Khong tim thay vi tri can xoa");
        else
        {
              if  (p_tim ->tiep == NULL)
                    p_truoc ->tiep = NULL;
              else
                    if  (p_tim == p_dau)    p_dau = p_tim -> tiep;
                    else    p_truoc -> tiep = p_tim ->tiep;
              free(p_tim);
              printf("Da xoa xong");
        }
        getch();
}
void  main();
{
```

```
char  chon;
do
{
        clrscr();
        printf("Danh sach chuc nang can thuc hien\n");
        printf("1. Tao danh sach\n");
        printf("2. In danh sach\n");
        printf("3. Chen phan tu vao trong danh sach\n");
        printf("4. Xoa phan tu trong danh sach\n");
        printf("5. Thoat\n");
        printf("Moi chon chuc nang");
        fflush(stdin);
        chon = getchar();
        switch (chon)
        {
              case 1:  tao_ds();  break;
              case 2:  in_ds();  break;
             case 3:  chen_pt();  break;
              case 4:  xoa_pt();  break;
        }
 }  while (chon !=5);
}
```

Ví dụ 6.18: Cũng giống như ví dụ trên nhưng làm việc với danh sách liên kết ngược.

Các hàm in danh sách, chèn phần tử vào danh sách, xóa một phần tử ra khỏi danh sách trong danh sách liên kết ngược cũng được thực hiện giống như đối với danh sách liên kết thuận, chỉ khác biệt ở hàm tạo danh sách.

```
void   tao_ds()
```

```
{
        char  ht[20], chon;
        float  x;
        printf("Nhap danh sach sinh vien\n");
        printf("Ket thuc nhap bang nhan phim K\n");
        p_ds = NULL;
        do
        {
                fflush(stdin);
                printf("Nhap ho ten:");
                gets(ht);
                printf("Nhap diem trung binh:");
                scanf("%f",&x);
                p = (p_sv *)malloc(sizeof(p_sv));
                strcpy(p ->ho_ten, ht);
                p -> diem_tb = x;
                p -> tiep = p_ds;
                p_ds = p;
                printf("\nCo nhap nua khong?");
                fflush(stdin);
                chon = getchar();
        }  while (chon != 'k' );
}
```

6.3.4. Ngăn xếp

Là một danh sách các phần tử nhưng phép thêm phần tử và loại bỏ phần tử chỉ được thực hiện ở một đầu của danh sách. Đầu này còn gọi là đỉnh ngăn xếp. Ngăn xếp hoạt động theo cơ chế LIFO (Last In First Out), tức là phần tử vào ngăn xếp

đầu tiên thì lại ra khỏi ngăn xếp cuối cùng và ngược lại, phần tử vào ngăn xếp cuối cùng thì ra đầu tiên.

Ví dụ 6.19: về việc sử dụng ngăn xếp trong bài toán chuyển một số n sang cơ số a bất kỳ.

```
#include”stdio.h”
#include”`conio.h”
#include”alloc.h”
typedef  struct   node
{
        int  so_du;
        struct  node  *tiep;
}  p_node;
p_node   *top, *p;
void  push(int   n)
{
          p = (p_node*)malloc(sizeof(p_node));
          p->so_du= n;
          p->tiep = top;
          top = p;
}
void  chuyen_co_so(int  so, int  co_so)
{
          top = NULL;
          while (so != 0)
          {
                  push(so % co_so);
                  so = so / co_so;
          }
```

```
}
void  pop(int  *m)
{
        p = top;
        *m = p->so_du;
        top = top -> tiep;
        free(p);
}
void   hien_ket_qua(int  so,int  co_so)
{
        int n;
        printf("\nso %d duoc doi sang co so %d la:", so, co_so};
        while (top != NULL)
        {
                pop(&n);
                if   (n <10)  printf("%d", n);
                else   printf("%c", n + 55);
        }
        getch();
}
void   main()
{
        int  n, a;
        printf("Nhap so va co so:");
        scanf("%d %d",&n, &a);
        chuyen_co_so(n, a);
        hien_ket_qua(n, a);
```

```
        getch();
}
```

6.3.5. Hàng đợi

Là một danh sách các phần tử trong đó phép toán thêm phần tử chỉ được thực hiện ở một đầu của danh sách (đuôi hàng) còn phép loại bỏ phần tử thực hiện ở đầu còn lại của danh sách (đầu hàng). Hàng đợi hoạt động theo cơ chế FIFO (First In First Out), tức là phần tử vào hàng đợi đầu tiên thì sẽ ra đầu tiên, ngược lại phần tử vào cuối cùng thì sẽ ra khỏi hàng đợi cuối cùng. Có thể cài đặt hàng đợi bằng danh sách liên kết thuận.

Ví dụ 6.20: về việc sử dụng hàng đợi trong bài toán chuyển n đĩa từ cột 1 sang cột 2

```
#include"stdio.h"
#include"conio.h"
#include"alloc.h"
typedef  struct   node
{
        int  dia, cot_nguon, cot_dich;
        struct  node  *tiep;
}  p_node;
p_node   *first, *last, *p;
void  them_pt(int   n, int c1, int c2)
{
          p = (p_node*)malloc(sizeof(p_node));
          p->dia = n;
          p->cot_nguon = c1;
          p ->cot_dich = c2;
          if  (first == NULL)
         {
                first = p;
```

```
                last = p;
        }
         else
        {
                last->tiep = p; last = p;
                 p -> tiep = NULL;
        }
}
void  chuyen_dia(int  n, int c1, int  c2, int  c3)
{
        if  (n == 1)    them_pt(n, c1, c2);
         else
         {
                chuyen_dia(n-1, c1, c3, c2);
                them_pt(n, c1, c2);
                chuyen_dia(n-1, c3, c2, c1);
           }
}
void  lay_pt(int  *n, int *c1, int  *c2)
{
         p = first;
         *n = p->dia;
         *c1 = p->cot_nguon;
         *c2 = p->cot_dich;
         first = first -> tiep;
         free(p);
}
```

```
void    hien_ket_qua(void)
{
        int   d, cn, cd;
         while (first != NULL)
         {
                lay_pt(&d, &cn, &cd);
                printf("\n chuyen dia thi %d tu cot %d sang cot %d", d,
                cn, cd);
         }
}
void   main()
{
          int  sd;
          first = NULL;
          printf("Nhap so dia:");
          scanf("%d",&sd);
          chuyen_dia(sd,1,2,3);
          hien_ket_qua();
          getch();
}
```

Bài tập cuối chương

Câu 1. Viết chương trình xây dựng một cấu trúc (ứng với phiếu điểm của thí sinh) gồm các thành phần: họ tên, quê quán, trường, tuổi, số báo danh, điểm thi. Trong đó họ tên lại là một cấu trúc gồm ba thành phần: họ, tên đệm và tên. Quê quán cũng là một cấu trúc gồm ba thành phần: xã, huyện, tỉnh. Điểm thi cũng là một cấu trúc gồm ba thành phần: toán, lý, hóa. Nhập số liệu từ một phiếu điểm cụ thể và lưu trữ vào các thành phần của cấu trúc nói trên sau đó in các số liệu ra màn hình.

Câu 2. Xây dựng một mảng cấu trúc mà mỗi thành phần của nó có kiểu như cấu trúc ở bài 1. Nhập số liệu của 20 phiếu điểm và lưu trữ vào mảng cấu trúc nói trên. Tìm kiếm và in ra các thí sinh có tổng số điểm ba môn lớn hơn 15,

Câu 3. Giả sử đã nhập số liệu của 20 phiếu điểm theo yêu cầu của câu 2. Hãy viết chương trình sắp xếp lại các phần tử của mảng cấu trúc theo thứ tự giảm dần của tổng số điểm, sau đó in danh sách thí sinh sau khi đã sắp xếp ra màn hình. Mỗi thí sinh in ra trên một dòng gồm các thông tin: Họ tên, quê quán, số báo danh, điểm toán, lý, hóa.

Câu 4. Viết chương trình nhập vào số liệu bán hàng với n mặt hàng có cấu trúc gồm các thành phần: Tên hang, đơn giá, số lượng, thành tiền (= số lượng * đơn giá). Sau đó hiện ra màn hình theo dạng:

SO LIEU BAN HANG

STT	Ten Hang	Don gia	So luong	Thanh tien
1	Sach	5	100	500
2	But	2	300	600

Câu 5. Dùng kết hợp struct và union để nhập vào danh sách các học sinh của một lớp học. Trong đó các thành phần chung là: Ho_ten, Que, Gioi tinh, Tong_diem

Các thành phần riêng ứng với giới tính là:

Nữ: Điểm hát, điểm múa.

Nam: Điểm thể dục, điểm tin

- Nhập dữ liệu cho các học sinh. Sau đó in ra danh sách học sinh theo dạng

DANH SACH HOC SINH NU

STT	Ho Ten	Que	Diem hat	Diem mua	Tong
. . .	. . .				

DANH SACH HOC SINH NAM

STT Ho Ten Que Diem thể dục Diem tin Tong

. . . . . .

Câu 6. Viết chương trình tạo một danh sách liên kết thuận để ghi danh sách các sinh viên có cấu trúc gồm các thành phần: Ho_ten, Tuoi, Diem TB.

- In danh sách ra màn hình theo dạng:

DANH SACH SINH VIEN

STT Ho ten Tuoi Diem TB

. . . . . .

- Chèn thêm một sinh viên (có tên nhập vào từ bàn phím) vào sau một sinh viên nào đó (có tên nhập vào từ bàn phím) rồi in danh sách theo dạng trên

- Xóa một sinh viên nào đó (có tên nhập từ bàn phím) ra khỏi danh sách rồi in danh sách theo dạng trên.

Câu 7. Viết chương trình tạo một danh sách liên kết ngược để ghi danh sách các sinh viên có cấu trúc gồm các thành phần: Ho_ten, Tuoi, Diem TB.

- In danh sách ra màn hình theo dạng:

DANH SACH SINH VIEN

STT Ho ten Tuoi Diem TB

. . . . . .

- Chèn thêm một sinh viên (có tên nhập vào từ bàn phím) vào trước một sinh viên nào đó (có tên nhập vào từ bàn phím) rồi in danh sách theo dạng trên.

- Xóa một sinh viên nào đó (có tên nhập từ bàn phím) ra khỏi danh sách rồi in danh sách theo dạng trên.

CHƯƠNG 7: THAO TÁC TRÊN CÁC TỆP TIN

7.1. Giới thiệu chung

Chương này trình bày các thao tác trên tệp như: tạo tệp mới, ghi dữ liệu từ bộ nhớ lên tệp, đọc dữ liệu từ tệp vào bộ nhớ,… Trong C các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia thành hai nhóm: cấp 1 và cấp 2. Mỗi hàm đều có thể truy xuất theo cả hai kiểu nhị phân và văn bản. Tuy nhiên việc chọn kiểu phù hợp cho mỗi hàm là cần thiết.

Các hàm cấp 1: Còn gọi là các hàm nhập/xuất hệ thống thực hiện việc đọc/ghi như DOS. Các hàm cấp 1 có đặc trưng sau:

- Không có dịch vụ nhập xuất riêng cho từng kiểu dữ liệu mà chỉ có dịch vụ đọc ghi một dãy các byte. Như vậy để ghi một số thực lên đĩa ta phải dùng dịch vụ ghi 4 byte, để ghi 10 số nguyên ta dùng dịch vụ ghi 20 byte

- Mỗi tệp có một số hiệu (còn gọi là danh số hay thẻ), các hàm cấp 1 làm việc với tệp thông qua số hiệu này.

Các hàm cấp 2: Được xây dựng từ các hàm cấp 1 nên dễ sử dụng và có nhiều khả năng hơn:

- Có dịch vụ truy xuất cho từng kiểu dữ liệu. Cụ thể sẽ có các hàm nhập xuất ký tự, chuỗi, số nguyên, số thực, cấu trúc,…

- C tự động cung cấp một vùng đệm. Mỗi lần đọc ghi thì thường tiến hành trên vùng đệm chứ không hẳn trên tệp, tức là khi ghi dữ liệu thì dữ liệu sẽ được đưa vào vùng đệm và khi nào đầy thì vùng đệm mới được đẩy lên đĩa. Còn khi đọc, thông tin được lấy từ vùng đệm và khi nào vùng đệm rỗng thì máy mới lấy dữ liệu từ đĩa chứa vào vùng đệm. Điều này sẽ làm giảm số lần nhập xuất trên đĩa và nâng cao tốc độ làm việc. Tuy nhiên, trong một số trường hợp ta phải làm sạch vùng đệm để tránh mất mát thông tin.

- Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

- Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1, nên trong chương trình C các hàm cấp 2 được ưa chuộng hơn. Vì các hàm cấp 1 ở mức độ sâu hơn, gần DOS hơn nên tốc độ truy nhập sẽ nhanh hơn. Trong khuôn khổ giáo trình, tác giả sẽ trình bày hệ thống truy xuất cấp 2.

7.2. Kiểu nhập xuất nhị phân và văn bản

Như chúng ta đã biết, một tệp tin (dù được xây dựng bằng cách nào) là một dãy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dãy chính là độ dài của tệp.

7.2.1. Kiểu nhập xuất nhị phân

7.2.1.1. Bảo toàn dữ liệu

Trong quá trình nhập xuất, dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân như trong bộ nhớ.

7.2.1.2. Mã kết thúc tệp

Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (End – Of – File, định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác không. Ở đây lý do chọn số -1 làm mã kết thúc tệp vì nếu chưa gặp cuối tệp thì sẽ đọc được một byte có giá trị từ 0 đến 255. Như vậy giá trị -1 sẽ không trùng với bất kỳ byte nào đọc được từ tệp.

7.2.2. Kiểu nhập xuất văn bản

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

7.2.2.1. Mã chuyển dòng.

Khi ghi dữ liệu lên tệp, một ký tự LF (mã 10) được chuyển thành hai ký tự CR (mã 13) và LF (mã 10).

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

Ví dụ 7.1: Xét hàm fputc(10, f); Nếu tệp f được mở để ghi theo kiểu nhị phân, thì hàm sẽ ghi lên tệp một ký tự mã 10, nhưng nếu tệp f được mở để ghi theo kiểu văn bản thì hàm sẽ ghi lên tệp hai mã là 13 và 10.

7.2.2.2. Mã kết thúc tệp.

Khi đọc một tệp f nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (số -1) và hàm feof(f) cho giá trị khác không (số 1).

Ví dụ 7.2: Giả sử ghi các mã 65 10 26 66 lên tệp f:

- Nếu ghi theo kiểu nhị phân thì nội dung của tệp f là: 65 10 26 66
- Nếu ghi theo kiểu văn bản thì nội dung của tệp f là: 65 13 10 26 66

Khi đọc tệp f trên thì:

- Nếu đọc theo kiểu nhị phân sẽ đọc được tất cả các mã.

- Nếu đọc theo kiểu văn bản thì chỉ đọc được các mã 65, 10, và 26 vì mã 26 được hiểu là mã kết thúc tệp.

7.3. Các hàm xử lý tệp cấp 2

Các hàm cấp 2 sử dụng cấu trúc FILE và mã kết thúc EOF, tất cả đều được khai báo và định nghĩa trong thư viện stdio.h. Mã EOF bằng -1 còn cấu trúc FILE gồm các thành phần dùng để quản lý tập tin như:

level: cho biết có còn dữ liệu trong vùng đệm không

bsize: độ lớn vùng đệm (mặc định là 512 byte)

flags: các cờ trạng thái.

Mặc dù mỗi hàm đều có thể dùng chung cho cả hai kiểu nhập xuất, nhưng việc lựa chọn kiểu để dùng cho mỗi hàm là cần thiết để tránh nhầm lẫn, sai sót. Dưới đây, chúng ta sẽ phân loại các hàm theo sự lựa chọn này.

7.3.1. Các hàm dùng chung cho cả hai kiểu nhập xuất

7.3.1.1. Hàm mở tệp

Dạng hàm:

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

trong đó đối thứ nhất là tên của tệp và đối thứ hai là kiểu truy nhập. Kiểu truy nhập có thể có các giá trị sau:

Bảng 0.1. Các kiểu truy nhập tệp

Kiểu	Ý nghĩa
”r” / “rt”	Mở 1 tệp để đọc theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi
“w” / “wt”	Mở 1 tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó sẽ bị xóa
“a” / “at”	Mở 1 tệp để ghi bổ xung theo kiểu văn bản. Tệp chưa tồn tại thì tạo tệp mới
“rb”	Mở 1 tệp để đọc theo kiểu nhị phân. Tệp cần tồn tại nếu

	không sẽ có lỗi
“wb”	Mở 1 tệp để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó sẽ bị xóa
“ab”	Mở 1 tệp để ghi bổ xung theo kiểu nhị phân. Tệp chưa tồn tại thì tạo tệp mới
“r+” / “r+t”	Mở 1 tệp để đọc/ghi theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi
“w+” / “w+t”	Mở 1 tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó sẽ bị xóa
“a+” / “a+t”	Mở 1 tệp để đọc/ghi bổ xung theo kiểu văn bản. Tệp chưa tồn tại thì tạo tệp mới
“r+b”	Mở 1 tệp để đọc/ghi theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi
“w+b”	Mở 1 tệp để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó sẽ bị xóa
“a+b”	Mở 1 tệp để đọc/ghi bổ xung theo kiểu nhị phân. Tệp chưa tồn tại thì tạo tệp mới

Công dụng của hàm: Dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Nếu có lỗi hàm trả về giá trị NULL.

Chú ý: trong các kiểu đọc/ghi, cần làm sạch vùng đệm trước khi chuyển từ đọc sang ghi và ngược lại.

7.3.1.2. Hàm đóng tệp

Dạng hàm: int fclose(FILE *fp); trong đó đối fp là con trỏ tương ứng với tệp cần đóng.

Công dụng: Dùng để đóng tệp, đóng tệp gồm các công việc sau:

- Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi)

- Xóa vùng đệm (khi đang đọc)

- Giải phóng biến fp để nó có thể dùng cho tệp khác. Nếu thành công hàm cho giá trị 0 ngược lại hàm cho EOF.

7.3.1.3. Hàm đóng tất cả các tệp đang mở

Dạng hàm: int fcloseall(void);

Công dụng: Dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đóng được, ngược lại hàm cho EOF.

7.3.1.4. Hàm làm sạch vùng đệm của tệp

Dạng hàm: int fflush(FILE *fp); với đối fp là con trỏ tệp.

Công dụng: Dùng để làm sạch vùng đệm của tệp fp. Nếu thành công hàm cho giá trị 0, ngược lại hàm cho EOF.

7.3.1.5. Hàm làm sạch vùng đệm của các tệp đang mở

Dạng hàm: int fflushall(void);

Công dụng: Dùng để làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đang mở, ngược lại hàm cho EOF.

7.3.1.6. Hàm kiểm tra lỗi thao tác tệp

Dạng hàm: int ferror(FILE *fp); với đối fp là con trỏ tệp.

Công dụng: Dùng để kiểm tra lỗi thao tác trên tệp fp. Hàm cho giá trị khác không nếu có lỗi, ngược lại nếu không có lỗi thì cho giá trị bằng không.

7.3.1.7. Hàm thông báo lỗi hệ thống

Dạng hàm: void perror(const char *s); với s là con trỏ trở tới một chuỗi ký tự.

Công dụng: Hàm in chuỗi s và thông báo lỗi

7.3.1.8. Hàm kiểm tra cuối tệp

Dạng hàm: int feof(FILE *fp); với fp là con trỏ tệp

Công dụng: Dùng để kiểm tra cuối tệp. Hàm cho giá trị khác không nếu gặp cuối tệp khi đọc, ngược lại hàm cho giá trị bằng không.

7.3.1.9. Hàm loại tệp trên đĩa

- Hàm unlink

Dạng hàm: int unlink(const char *tên_tệp) với đối tên_tệp là tên của tệp cần xóa.

Công dụng: Dùng để xóa một tệp có tên là tên_tệp trên đĩa. Nếu thành công hàm cho giá trị 0, ngược lại hàm cho giá trị EOF.

- Hàm remove

Dạng hàm: remove(const char *tên_tệp); với đối tên_tệp là tên của tệp cần xóa.

Công dụng: Dùng để xóa một tệp có tên là tên_tệp trên đĩa.

7.3.1.10. Hàm chuyển con trỏ về đầu tệp

Dạng hàm: void rewind(FILE *fp); với đối fp là con trỏ tệp

Công dụng: Chuyển con trỏ tệp về vị trí đầu tệp.

7.3.1.11. Hàm chuyển con trỏ đến vị trí bất kỳ

Dạng hàm: int fseek(FILE *fp, long sb, int xp); với đối fp là con trỏ tệp, đối sb là số byte cần di chuyển, và xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đấy, xp có thể nhận các giá trị sau:

Xp = SEEK_SET hay 0: xuất phát từ đầu tệp

Xp = SEEK_CUR hay 1:xuất phát từ vị trí hiện tại của con trỏ

Xp = SEEK_END hay 2: xuất phát từ cuối tệp.

Công dụng: Hàm di chuyển con trỏ tệp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối của sb. Chiều di chuyển về cuối tệp nếu sb dương, ngược lại sẽ di chuyển về đầu tệp. Khi thành công hàm trả về giá trị 0, khi có lỗi hàm trả về giá trị khác 0.

7.3.1.12. Hàm xem vị trí hiện tại của con trỏ tệp

Dạng hàm: long ftell(FILE *fp); với đối fp là con trỏ tệp

Công dụng: Khi thành công, hàm cho biết vị trí hiện tại của con trỏ tệp (byte thứ mấy trên tệp fp, số thứ tự của byte được tính bắt đầu từ 0). Khi có lỗi hàm trả về -1.

7.3.2. Các hàm nhập xuất ký tự

Các hàm nhập xuất ký tự dùng được cả trong kiểu nhập xuất nhị phân và nhập xuất văn bản.

7.3.2.1. Các hàm putc và fputc

Dạng hàm:

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

trong đó ch là một biến nguyên, fp là con trỏ tệp

Công dụng: Hai hàm trên có ý nghĩa như nhau. Cho phép ghi lên tệp fp một ký tự có mã bằng ch % 256 trong đó ch được xem là số nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, ngược lại hàm cho EOF.

Ví dụ 7.3: Giả sử ta ghi ký tự \n lên tệp f bằng câu lệnh putc('\n', f); thì:

- Nếu ghi tệp theo kiểu nhị phân, kết quả trên tệp có ký tự 10
- Nếu ghi theo kiểu văn bản, kết quả trên tệp có các ký tự mã 13 và 10.

7.3.2.2. Các hàm getc và fgetc

Dạng hàm:

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

với fp là con trỏ tệp.

Công dụng: Hai hàm trên có ý nghĩa như nhau. Cho phép đọc một ký tự từ tệp fp. Nếu thành công hàm cho mã của ký tự đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm cho EOF.

Ví dụ 7.4: Giả sử tệp f có nội dung như sau: 13 10

- Nếu mở tệp để đọc theo kiểu nhị phân ta có: ch1 = getc(f) = 13, ch2 = getc(f) = 10.
- Nếu mở tệp để đọc theo kiểu văn bản ta có: ch1 = getc(f) = 10, ch2 = getc(f) = -1 (gặp cuối tệp).

Ví dụ 7.5: Chương trình sao chép nội dung từ tệp f1 sang tệp f2

```
#include "stdio.h"
void   main()
{
        int   c;
```

```
        char   t1[20], t2[20];
        FILE  *f1, *f2;
        printf("\nTen tep nguon:");
        fflush(stdin);
        gets(t1);
        printf("\nTen tep dich:");
        fflush(stdin);
        gets(t2);
        f1 = fopen(t1, "rb");
        if (f1 == NULL)
        {
                printf("\n Tep %s khong ton tai",t1);
                getch();
                exit(1);
        }
        f2 = fopen(t1, "wb");
        while ((c=fgetc(f1) != EOF)  fputc(c, f2);
        fclose(f1);
        fclose(f2);
        printf("\nDa sao chep xong.");
        getch();
}
```

Chú ý: Nếu chương trình trên ta thay bằng kiểu nhập xuất văn bản thì chỉ các byte đứng trước mã 26 đầu tiên của tệp f1 được sao chép sang tệp f2.

7.3.3. Các hàm nhập xuất theo kiểu văn bản

Các hàm trong mục này chỉ nên dùng theo kiểu nhập xuất văn bản.

7.3.3.1. Hàm ghi dữ liệu lên tệp theo khuôn dạng fprintf

Dạng hàm:

int fprintf(FILE *fp, const char *cđk, danh sách đối);

Với fp là con trỏ tệp, cdk là hằng con trỏ kiểu char chứa địa chỉ của chuỗi điều khiển, danh sách đối là các đối mà giá trị của chúng cần ghi lên tệp. Trong đó chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm printf.

Công dụng: Giá trị của các đối được ghi lên tệp fp theo khuôn dạng xác định trong chuỗi điều khiển. Nếu thành công, hàm trả về một giá trị nguyên bằng số byte ghi lên tệp, ngược lại nếu có lỗi hàm cho EOF. Cách làm việc của hàm giống như hàm printf.

Ví dụ 7.6: Giả sử có khai báo sau: int x = 2, y = 10;

Câu lệnh fprintf(f,” Ma trận A \n %d__%d”, x, y); sẽ ghi lên tệp f nội dung sau:

Ma trận A

2 10

7.3.3.2. Hàm đọc dữ liệu từ tệp theo khuôn dạng fscanf

Dạng hàm:

int fscanf(FILE *fp, const char *cdk, danh sách đối);

Với fp là con trỏ tệp và cdk là hằng con trỏ kiểu char chứa địa chỉ của chuỗi điều khiển, danh sách đối là các đối dùng để chứa kết quả đọc được từ tệp. Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm scanf.

Công dụng: Đọc dữ liệu từ tệp fp, biến đổi theo khuôn dạng và lưu kết quả vào các đối. Hàm làm việc giống như hàm scanf. Hàm trả về giá trị bằng số trường được đọc.

Ví dụ 7.7: Giả sử có một dãy số nguyên ghi lên tệp “so_nguyen.sl”. Giữa hai số nguyên có ít nhất một khoảng trống. Sau chữ số cuối cùng là mã 26 hay cuối tệp, Yêu cầu đọc và in ra màn hình dãy số nói trên, mỗi số được in trên một dòng.

Chương trình được viết như sau:

```
#include”stdio.h”
void main()
```

```
{
        FILE *f;
        int  c;
        f = fopen(“so_nguyen.sl”, “r”);
        while  (!feof(f))
        {
                fscanf(f, “%d”, &c);
                printf(“\n%d”, c);
        }
        fclose(f);
        getch();
}
```

7.3.3.3. Hàm ghi một chuỗi ký tự lên tệp

Dạng hàm: int fputs(const char *s, FILE *fp); với đối s là hằng con trỏ kiểu char trỏ tới địa chỉ đầu của một chuỗi ký tự và fp là con trỏ tệp.

Công dụng: Ghi chuỗi s lên tệp fp (ký tự ‘\0’ của chuỗi không được ghi lên tệp). Khi thành công hàm trả về mã của ký tự cuối cùng được ghi lên tệp, khi có lỗi hàm cho EOF.

Ví dụ 7.8: Viết chương trình để nhập các dòng ký tự từ bàn phím và ghi lên tệp van_ban.

Chương trình được viết như sau

```
#include “stdio.h”
#include “conio.h”
void main()
{
        int  i = 0;
        char   dong_kt[256];
        FILE  *f;
```

```
        f = fopen("van_ban", "w");
        while (1)
        {
            ++i;
            printf("\nDong thu %d (Bam enter de ket thuc):", i);
            gets(dong_kt);
            if (dong_kt[0] = '\0')    break;
            if (i>1)  fputc(10, f);
            fputs(d, f);
        }
        fclose(f);
        getch();
}
```

7.3.3.4. Hàm đọc một chuỗi ký tự từ tệp

Dạng hàm: char * fgets(char *s, int n, FILE *fp); với s là con trỏ trỏ tới vùng nhớ đủ lớn để chứa chuỗi ký tự đọc từ tệp, n là số nguyên xác định độ dài cực đại của chuỗi cần đọc và fp là con trỏ tệp.

Công dụng: Hàm đọc một chuỗi ký tự từ tệp fp chứa vào vùng nhớ s. việc đọc kết thúc khi:

- Hoặc đã đọc được n -1 ký tự
- Hoặc gặp dấu xuống dòng (cặp mã 13 và 10). Khi đó mã 10 được đưa vào xâu kết s.
- Hoặc kết thúc tệp.

Chuỗi s sẽ được bổ sung thêm ký tự kết thúc chuỗi. Khi thành công hàm trả về địa chỉ của vùng chuỗi s, khi có lỗi hoặc gặp cuối tệp hàm cho giá trị NULL.

Ví dụ 7.9: Viết chương trình đọc các dòng ký tự trên tệp "van_ban" đã được tạo trong ví dụ 7.8 và in ra màn hình.

Chương trình được viết như sau:

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

```
#include “conio.h”
void main()
{
        int  i = 0;
        char   dong_kt[256];
        FILE  *f;
        f = fopen(“van_ban”, “r”);
        while (!feof(f))
        {
                ++i;
                fgets(dong_kt, 256, f);
                printf(“\nDong thu %d: %s”, i, dong_kt);
        }
        fclose(f);
        getch();
}
```

Ví dụ 7.10: Lập một tệp có tên là ma trận.C có nội dung như sau: dòng đầu tiên của tệp ghi giá trị số hàng và số cột của một ma trận A_{n*m}, dòng thứ hai ghi chuỗi “Ma trận A”, từ dòng thứ ba trở đi ghi lên tệp các phần tử của ma trận A, ghi trên n hàng, mỗi hàng có m phần tử.

```
#include”stdio.h”
#include”conio.h”
void  main()
{
        FILE  *fp;
        int  n, m, i, j, a[10][10];
        char  tb[50];
        fp = fopen(“ma tran.C”, “wt”);
```

```
        fprintf(fp,"Ma tran A\n");
        printf("\nNhap so hang, cot cua ma tran:");
        scanf("%d %d", &n, &m);
        fprintf(fp, "%d  %d", n, m);
        for(i=1; i<=n;i++)
        {
            for(j=1; j<=m;j++)
            {
                printf("\nNhap phan tu  = ");
                scanf("%d", &x);
                fprintf(fp,"%3d  ", x);
            }
            fprintf(fp,"\n");
        }
        fclose(fp);
        printf("\nDa lap tep xong."); getch();
}
```

Ví dụ 7.11: Giả sử có tệp ma trận.C như trên, viết chương trình đọc lại tệp sau đó ghi bổ sung vào cuối tệp ma trận chuyển vị của ma trận A_{n*m} như sau:

Dòng đầu của phần bổ sung ghi chuỗi "ma trận chuyển vị B_{m*n}". Các dòng tiếp theo ghi các phần tử của ma trận chuyển vị, ghi trên m hàng và mỗi hàng có n phần tử.

```
#include"stdio.h"
#include"conio.h"
void  main()
{
        FILE  *fp;
        int  n, m, i, j, a[10][10];
```

```
        char  tb[50];
        fp = fopen("ma tran.C", "rt");
        fgets(tb, 50, fp);
        fscanf(fp,"%d %d", &n, &m);
        fgets(tb, 50, fp);
        for(i=1; i<=n;i++)
        {
            for(j=1; j<=m;j++)
                fscanf(fp,"%d", &a[i][j]);
            fgets(tb, 50, f1);
        }
        fclose(fp);
        fp = fopen("ma tran.C", "at");
        fprintf(fp,"Ma tran chuyen vi\n");
        for(i=1; i<=m;i++)
        {
            for(j=1; j<=n;j++)
            fprintf(fp,"%-3d", a[j][i])
            fprintf(fp,"\n");
        }
        fclose(fp);
        getch();
    }
```

7.3.4. Các hàm nhập xuất theo kiểu nhị phân

Các hàm trong mục này chỉ nên dùng theo kiểu nhị phân

7.3.4.1. Hàm ghi một số nguyên lên tệp

Dạng hàm: int putw(int n, FILE *fp); với n là biến nguyên và fp là con trỏ tệp.

Công dụng: Ghi giá trị của n lên tệp fp dưới dạng 2 byte. Nếu thành công hàm trả về số nguyên được ghi, nếu có lỗi hàm cho EOF

7.3.4.2. Hàm đọc một số nguyên từ tệp

Dạng hàm: int getw(FILE *fp); với fp là con trỏ tệp

Công dụng: Đọc một số nguyên (2byte) từ tệp fp. Nếu thành công hàm trả về số nguyên đọc được. Nếu có lỗi hoặc gặp cuối tệp, hàm trả về EOF.

7.3.4.3. Hàm ghi các mẫu tin lên tệp

Dạng hàm:

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

với ptr là con trỏ không kiểu trỏ tới vùng nhớ chứa dữ liệu cần ghi, size là kích thước của mẫu tin theo byte, n là số mẫu tin cần ghi và fp là con trỏ tệp.

Công dụng: Ghi n mẫu tin, mỗi mẫu tin có kích thước là size byte từ vùng nhớ ptr lên tệp fp. Hàm trả về một giá trị bằng số mẫu tin thực sự được ghi.

7.3.4.4. Hàm đọc các mẫu tin từ tệp

Dạng hàm: int fread(void *ptr, int size, int n, FILE *fp); với ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu đọc được, size là kích thước của mẫu tin theo byte, n là số mẫu tin cần đọc và fp là con trỏ tệp.

Công dụng: Đọc n mẫu tin kích thước size byte từ tệp fp chứa vào vùng nhớ ptr. Hàm trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ 7.12: Viết chương trình làm việc với tệp chứa một danh sách sinh viên của một lớp, mỗi sinh viên gồm các thành phần: họ tên, điểm trung bình. Chương trình gồm các chức năng sau:

1. Tạo tệp
2. Bổ sung một sinh viên vào danh sách
3. Sửa chữa thông tin về một sinh viên trong danh sách
4. Xem và in danh sách ra màn hình
5. Kết thúc

```
#include”stdio.h”
#include”conio.h”
void tao_tep();
```

```
void   bo_sung();
void  sua_tep();
void  in_ra();
typedef   struct
{
        char ho_ten[30];
        float  diem_tb;
}  sinh_vien:
long   size = sizeof(sinh_vien);
FILE  *fp;  int   n = 0;
sinh vien   sv;
char chon;
void  main()
{
        do
        {
                clrscr();
                printf("1. Tao tep\n");
                printf("2. Bo sung vao tep\n");
                printf("3. Sua chua trong tep\n");
                printf("4. In noi dung tep\n");
                printf("5. Ket thuc\n");
                printf("Moi chon chuc nang\n");
                chon = getch();
                switch   (chon)
                {
                        case '1': tao_tep();      break;
```

```
                    case '2': bo_sung();    break;
                    case  '3':sua_tep();        break;
                    case  '4':in_ra();          break;
                }
          } while  (chon != '5');
        getch();
}
void   tao_tep()
{
          fp = fopen("dssv.sl","wb");
          do
          {
                printf("\nNhap sinh vien thu %d",++n);
                printf("\nNhap ho ten:");
                fflush(stdin);
                gets(sv.ho_ten);
                printf("\nNhap diem trung binh:");
                scanf("%f",&sv.diem_tb);
                fwrite(&sv, size, 1, fp);
                printf("\nCo nhap nua khong?");
                fflush(stdin); chon = getchar();
          } while (chon != 'k');
        fclose(fp);
}
void  bo_sung()
{
          fp = fopen("dssv.sl","ab");
```

```
        fseek(fp, 0, SEEK_END);
        n = ftell(fp)/size;
        do
        {
                printf("\nBo sung sinh vien thu %d",++n);
                printf("\nNhap ho ten:");
                fflush(stdin);
                gets(sv.ho_ten);
                printf("\nNhap diem trung binh:");
                scanf("%f",&sv.diem_tb);
                fwrite(&sv, size, 1, fp);
                printf("\nCo bo sung nua khong?");
                fflush(stdin); chon = getchar();
        } while  (chon != 'k');
        fclose(fp);
}
void  sua_tep()
{
        fp = fopen("dssv.sl","r+b");
        fseek(fp, 0, SEEK_END);
        n = ftell(fp)/size;
        do
        {
                printf("\nSua sinh vien thu:");
                scanf "%d", &i);
                fseek(fp,(i-1)*size,SEEK_SET);
                fread(&sv,size,1,fp);
```

```
            printf("\nHo ten sinh vien hien tai %s:",  sv.ho_ten);
            printf("\nDiem trung binh: %6.2f", sv.diem_tb);
            printf("\nNhap ho ten moi:");
            fflush(stdin);          gets(sv.ho_ten);
            printf("\nNhap diem trung binh moi:");
            scanf("%f",&sv.diem_tb);
            fseek(fp, -size, SEEK_CUR);
            fwrite(&sv, size, 1, fp);
            printf("\nCo sua nua khong");
            fflush(stdin);
            chon = getchar();
        } while (chon !='k');
        fclose(f);
        printf("\nDa sua xong.");
        getch();
}
void  in_ra()
{
          int   i = 0;
          fp = fopen("dssv.sl","rb");
          printf("\nDanh sach sinh vien\n");
          printf("\nSTT                Ho ten         Diem trung binh\n");
          while (fread(&sv, size, 1, fp) >0)
          printf("\n%d          %s              %6.2f",++i,
          sv.ho_ten,sv.diem_tb);
          fclose(fp);
          getch();
}
```

Bài tập cuối chương

Câu 1. Viết chương trình nhập một tệp có kiểu có tên là SO_NGUYEN.DAT chứa n (số n được nhập từ bàn phím và n <= 1000) số nguyên. Với các số nguyên được nhập vào từ bàn phím.

Câu 2. Viết chương trình đọc tệp có tên là SO_NGUYEN.DAT được lập từ câu 1. Sau đó in các số nguyên ra màn hình.

Câu 3. Viết chương trình nhập vào một tệp văn bản (với tên tệp được nhập từ bàn phím – và do người dùng lựa chọn) có chứa n (số n được nhập từ bàn phím và n <= 1000) số nguyên với yêu cầu: mỗi dòng của tệp chỉ chứa 10 số nguyên.

Câu 4. Viết chương trình đọc tệp được lập từ câu 3. Sau đó in các dữ liệu đọc được ra màn hình.

Câu 5. Viết chương trình tính tổng hai ma trận: $C_{nxm} = A_{nxm} + B_{nxm}$. Các ma trận A, B được nhập vào từ bàn phím. Ghi ba ma trận vào tệp CONG_MT.C (sử dụng kiểu nhập xuất văn bản).

Câu 6. Viết chương trình tính tích hai ma trận: $C_{nxm} = A_{nxp} \times B_{pxm}$. Trong đó dữ liệu vào: n, p, m và hai ma trận A, B được ghi trên tệp TICH_MT.C. Ghi bổ sung ma trận tích tính được vào cuối tệp này (sử dụng kiểu nhập xuất văn bản), rồi mở tệp ra xem kết quả.

Câu 7. Viết chương trình sử dụng kiểu nhập xuất nhị phân với tệp có tên là SO_LIEU.C để ghi lên tệp danh sách số liệu bán hàng có cấu trúc gồm các thành phần: Ten_hang, Don_gia, So_luong, Thanh_tien (= So_luong * Don_gia). Sau đó hiện nội dung tệp ra màn hình theo dạng:

SO LIEU BAN HANG

STT	Ten Hang	Don gia	So luong	Thanh tien
1	Sach	5	100	500
2	But	2	300	600
			Tong tien	1100

Câu 8. Viết chương trình sử dụng kiểu nhập xuất nhị phân với tệp có tên là DSACH.C để ghi lên tệp danh sách các sinh viên có cấu trúc gồm các thành phần: Ho_ten, Tuoi, Diem_TB.

- Hiện nội dung tệp ra màn hình theo dạng:

DANH SACH SINH VIEN

STT Ho ten Tuoi Diem TB

. . . . . .

- Bổ sung một sinh viên (có tên nhập vào từ bàn phím) rồi in danh sách ra màn hình theo dạng trên.

PHỤ LỤC 1

BẢNG MÃ ASCII

Bảng mã ASCII gồm 256 ký tự được phân bổ như sau:

- 32 ký tự đầu tiên là các ký tự điều khiển không in được như tự Enter (mã 13), ký tự ESC (mã 27),…

- Các mã ASCII 32 – 47, 58 – 64, 91 – 96, 123 – 127 là các ký tự đặc biệt như dấu chấm phẩy, dấu phẩy, dấu cách, dấu ngoặc, dấu móc, dấu hỏi,…

- Các mã ASCII 48 – 57 là 10 chữ số

- Các mã ASCII 65 – 90 là các chữ cái in hoa từ A – Z

- Các mã ASCII 97 – 122 là các chữ cái thường từ a – z

Lưu ý: chữ cái thường có mã ASCII lớn hơn 32 so với chữ hoa tương ứng.

- Các mã ASCII 128 – 255 là các ký tự đồ họa.

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
0	NUL	42	*	84	T	127	DEL
1	SOH	43	+	85	U		
2	STX	44	,	86	V		
3	ETX	45	-	87	W		
4	EOT	46	.	88	X		
5	ENQ	47	/	89	Y		
6	AK	48	0	90	Z		
7	E	49	1	91	[		
8	S	50	2	92	\		
9	HT	51	3	93	]		
10	LF	52	4	94	^		
11	VT	53	5	95	_		

12	FF	54	6	96	**		
13	CR	55	7	97	a		
14	SO	56	8	98	b		
15	SI	57	9	99	c		
16	DLE	58	:	100	d		
17	DC1	59	;	101	e		
18	DC2	60	<	102	f		
19	DC3	61	=	103	g		
20	DC4	62	>	104	h		
21	NAK	63	?	105	i		
22	SYN	64	@	106	j		
23	ETB	65	A	107	k		
24	CAN	66	B	108	l		
25	EM	67	C	109	m		
26	SUB	68	D	110	n		
27	ESC	69	E	111	o		
28	FS	70	F	112	p		
29	GS	71	G	113	q		
30	RS	72	H	114	r		
31	US	73	I	115	s		
32	SACE	74	J	116	t		
33	!	75	K	117	u		
34	“	76	L	118	v		
35	#	77	M	119	w		
36	$	78	N	120	x		

37	%	79	O	121	y		
38	&	80	P	123	z		
39	‘	81	Q	124	{		
40	(	82	R	125	|		
41	)	83	S	126	}		

PHỤ LỤC 2

TÓM TẮT MỘT SỐ HÀM CHUẨN CỦA TURBO C

1. Các hàm vào ra trên bàn phím màn hình (Chương 3)
2. Các hàm xử lý tệp (Chương 7)
3. Các hàm chuyển đổi dữ liệu

- int tolower(int c); Đổi từ chữ hoa sang chữ thường
- int toupper(int c); Đổi từ chữ thường sang chữ hoa
- double atof(char *s); Chuyển chuỗi s sang giá trị double
- int atoi(char *s); Chuyển chuỗi s sang giá trị int
- long atol(char *s); Chuyển chuỗi s sang giá trị long
- char *itoa(int x, char *s, int cs); Chuyển số nguyên x trong hệ đếm cơ số cs(8, 10, 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng nhớ s.
- char *ltoa(long x, char *s, int cs); Chuyển số kiểu long x trong hệ đếm cơ số cs(8, 10, 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng nhớ s.
- char *ultoa(unsigned long x, char *s, int cs); Chuyển số kiểu unsigned long x trong hệ đếm cơ số cs(8, 10, 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng nhớ s.

4. Các hàm kiểm tra ký tự

- int isalnum(int kt); Kiểm tra kt có phải là ký tự chữ cái hoặc chữ số?
- int isaalpha(int kt); Kiểm tra kt có phải là chữ cái?
- int iscntrl(int kt); Kiểm tra kt có phải là ký tự điều khiển?
- int isdigit(int kt); Kiểm tra kt có phải là chữ số?
- islower(int kt); Kiểm tra kt có phải là chữ cái thường?
- isupper(int kt); Kiểm tra kt có phải là chữ cái hoa?
- isspace(int kt); Kiểm tra kt có phải ký tự trống?
- isxdigit(int kt); Kiểm tra kt có phải là chữ số hệ 16

5. Các hàm xử lý chuỗi ký tự

- char *strcat(char *s_nhan, char *s); Bổ sung chuỗi s vào sau chuỗi s_nhan

- char *strchr(char *s, int kt); Tìm lần xuất hiện đầu tiên của kt trong s. Nếu tìm thấy hàm cho địa chỉ của ký tự tìm được trong s, ngược lại cho NULL

- char *strrchr(char *s, int kt); Tìm lần xuất hiện cuối cùng của kt trong s. Nếu tìm thấy hàm cho địa chỉ của ký tự tìm được trong s, ngược lại cho NULL

- int strcmp(char *s1, char *s2);So sánh hai chuỗi s1 và s2. Hàm cho giá trị âm nếu chuỗi s1 nhỏ hơn s2, giá trị 0 nếu s1 bằng s2, giá trị dương nếu s1 lớn hơn s2.

- int strsmpi(char *s1, char *s2); Hàm làm việc như strcmp nhưng không phân biệt chữ hoa với chữ thường khi so sánh.

- char strcpy(char *s_nhan, char *s_gui); Hàm sao chuỗi s_gui vào vùng s_nhan

- int strlen(char *s); Hâm cho độ dài chuỗi.

- char *strlwr(char *s); Hàm chuyển mọi chữ hoa trong s thành chữ thường.

- char *strncat(char *s_nhan, char *s, int n); Ghép n ký tự đầu tiên của chuỗi s vào chuỗi s_nhan

- char *strrev(char *s); Đảo ngược các ký tự trong chuỗi s. Hàm cho địa chỉ chuỗi đã đảo

- char *strstr(char *s, char *s_con); Hàm tìm sự xuất hiện đầu tiên của chuỗi s_con trong chuỗi s. Nếu thấy hàm cho địa chỉ chuỗi con trong s, ngược lại hàm cho NULL

- char *strupr(char *s); Hàm chuyển mọi chữ thường trong chuỗi s thành chữ hoa.

6. Các hàm toán học

- int abs(int x); Tính giá trị tuyệt đối của số nguyên x

- long labs(lont int x); Tính giá trị tuyệt đối của số nguyên dài x

- int rand(void); Cho một giá trị ngẫu nhiên từ 0 đến 32767

- int random(int n); Cho một giá trị ngẫu nhiên từ 0 đến n – 1

- void randomize(void); Khởi đầu bộ số ngẫu nhiên bằng giá trị ngẫu nhiên

- double acos(double x); Tính arc cosine của x

- double asin(double x); Tính arc sine của x

- double atan(double x); Tính arc tangent của x

- double cos(double x); Tính cosine của x

- double sin(double x); Tính sine của x

- double tan(double x); Tính tangent của x

- double exp(double x); Tính e mũ x

- double log(double x); Tính logarit tự nhiên của x

- double log10(double x); Tính logarit cơ số 10 của x

- double pow(double x, double y); Tính y mũ x

- double sqrt(double x); Tính căn bậc hai của x

7. Các hàm thời gian

- void gettime(struct time *t); Hàm nhận giờ hệ thống và đặt vào các thành phần của biến cấu trúc kiểu time do con trỏ t trỏ tới

- void settime(struct time *t); Hàm đặt lại giờ hệ thống theo các giá trị các thành phần của biến cấu trúc time do con trỏ t trỏ tới

- void getdate(struct time *d); Hàm nhận ngày hệ thống và đặt vào các thành phần của biến cấu trúc kiểu time do con trỏ d trỏ tới

- void setdate(struct time *d); Hàm đặt lại ngày hệ thống theo các giá trị các thành phần của biến cấu trúc time do con trỏ t trỏ tới

- long time(long *t); Hàm cho thời gian hiện tại theo giây bắt đầu tính từ 0 giờ 0 phút 0 giây (giờ GMT) ngày 01 – 01 – 1970

8. Các hàm cấp phát bộ nhớ

- Hàm calloc (chương 6)

- Hàm malloc (chương 6)

- Hàm free (chương 6)

- void realloc (void *ptr, unsigned size); Hàm thay đổi kích thước vùng nhớ đã cấp phát trước đó. Vùng nhớ mới có thể có địa chỉ khác so với vùng nhớ cũ. Phần dữ liệu trên vùng nhớ cũ được chuyển tới vùng nhớ mới. Khi thành công hàm trả về địa chỉ vùng nhớ mới, trái lại hàm cho NULL

9. Các hàm kiểm soát quá trình

- void abort(void); Làm cho chương trình kết thúc tức thời một cách không bình thường. Hàm không đẩy dữ liệu còn sót lại trong vùng ký ức đệm lên tệp và không đóng các tệp. Nó trả về giá trị 3 cho hệ điều hành. Nói chung không nên dùng hàm này để kết thúc chương trình trừ khi thấy cần thiết.

- void exit(int status); Làm cho chương trình kết thúc tức thời một cách không bình thường. Hàm sẽ đẩy dữ liệu còn sót lại trong vùng ký ức đệm lên tệp và đóng các tệp. Hàm chuyển giá trị status cho hệ điều hành. Nếu hàm trả về 0 thì chương trình kết thúc bình thường.

- int system(char *l_dos); Thực hiện câu lệnh DOS cho trong chuỗi l_dos. Khi thành công hàm trả về 0, khi có lỗi hàm trả về -1.

PHỤ LỤC 3

HƯỚNG DẪN SỬ DỤNG MÔI TRƯỜNG KẾT HỢP TURBO C

Ở chương I đã giới thiệu vài nét về việc áp dụng TURBO C để soạn thảo, dịch và thực hiện chương trình. Dưới đây sẽ trình bày các khả năng của TURBO C một cách đầy đủ hơn.

1. Trong môi trường TURBO C

1.1. Từ menu chính.

Để chọn một menu nào đó ta đưa con trỏ tới tên menu này và bấm phím ENTER

1.2. Từ trong menu

- Bấm chữ đầu của một tên chức năng (hoặc đưa con trỏ tới một chức năng nào đó và bấm ENTER) để thực hiện chức năng này.

- Bấm phím Esc để ra khỏi menu và trở về menu chính.

- Bấm phím Esc trong menu chính để trở về menu đang làm việc trước đó.

- Bấm phím F6 để chuyển về cửa sổ đang làm việc.

1.3. Từ bất kỳ vị trí nào trong TURBO C

- Bấm phím F1 để xem lời hướng dẫn về tình hình hiện tại.

- Bấm phím F10 để trở về menu chính.

- Bấm đồng thời phím Alt và chữ đầu trong tên của một menu để chọn menu đó. Chẳng hạn bấm Alt – E để mở cửa sổ Edit, bấm Alt – F để mở cửa sổ File, Alt – C để mở cửa sổ Compile,…

1.4. Tổng kết các cách bấm phím và tác dụng của chúng

Bấm phím	Tác dụng
F1	Hiện lên màn hình thông tin về tình trạng hiện tại
F2	Ghi tệp đang soạn thảo vào đĩa

F3	Nạp một tệp vào
F5	Đóng hoặc không đóng khung cho cửa sổ hiện tại
F6	Chuyển từ cửa sổ nọ sang cửa sổ kia
F7	Chuyển đến vị trí lỗi trước đó
F8	Chuyển đến vị trí lỗi sau
F9	Thực hiện "make"
F10	Trở về menu chính
Alt – F1	Cho hiện lại màn hình hướng dẫn vừa xem
Alt – F3	Lấy một tệp để nạp vào
Alt – F9	Dịch sang tệp OBJ (từ tệp đang soạn thảo)
Alt – F10	Hiện màn hình version chương trình
Alt – C	Vào menu Compile
Alt – D	Vào menu Debug
Alt – E	Vào menu Edit
Alt – O	Vào menu Option
Alt – R	Vào menu Run, cho chạy thử chương trình
Alt – F	Vào menu File
Alt – P	Vào menu Project
Alt – S	Vào menu Search

Alt – W	Vào menu Window
Alt – H	Vào menu Help
Alt – X	Thoát khỏi TURBO C

2. Xây dựng tệp mới

2.1. Một số lệnh của trình soạn thảo văn bản

- Di chuyển con chạy trong tệp đang soạn thảo bằng các phím mũi tên (lên, xuống, trái, phải).

- Chuyển con trỏ sang phải một từ bằng Ctrl - →

- Chuyển con trỏ sang trái một từ bằng Ctrl - ←

- Chuyển con trỏ về đầu dòng bằng Home

- Chuyển con trỏ về cuối dòng bằng End

- Chuyển con trỏ về đầu trang màn hình bằng Ctrl – Home

- Chuyển con trỏ về cuối trang màn hình bằng Ctrl - End

- Chuyển con trỏ lên màn hình trước bằng phím Page up

- Chuyển con trỏ xuống trang màn hình sau bằng phím Page down

- Chuyển con trỏ về đầu chương trình bằng Ctrl – Page up

- Chuyển con trỏ về cuối chương trình bằng Ctrl – Page down

- Xóa một dòng bằng Ctrl – Y

- Xóa một từ bằng Ctrl – T

- Đánh dấu đầu khối bằng Ctrl – K – B, đánh dấu cuối khối bằng Ctrl – K – K

- Chuyển khối bằng Ctrl – K – V

- Sao chép khối bằng Ctrl – K – C

- Xóa khối bằng Ctrl – K – Y

- Insert: chuyển giữa hai chế độ chèn và đè.

2.2. Tạo ra tệp gốc mới

Để tạo ra tệp gốc mới ta thường làm như sau: Từ menu chính chọn menu File, sau đó chọn chức năng New. Khi đó cửa sổ với tên NONAME.C sẽ được mở. Ta có thể soạn thảo một tệp mới bằng cách sử dụng các lệnh đã giới thiệu trong giáo trình.

2.3. Ghi tên tệp đang soạn thảo lên đĩa

Khi ta bấm phím F2 (từ bất kỳ vị trí nào trong TURBO C) hoặc khi ta chọn chức năng SAVE của menu File thì máy sẽ mở một hộp sáng trên màn hình và đợi. Ta cần nhập tên tệp và đường dẫn vào phần tên tệp sau đó ấn phím ENTER. Chẳng hạn như ta ấn: D:\BAI_TAP1.C. Khi đó tên tệp đang soạn thảo được ghi ra đĩa. Tên của tệp chính là BAI_TAP1.C và tệp được lưu ở ổ đĩa D.

Chú ý:

- khi ta không đưa vào dấu chấm và phần mở rộng của tên tệp thì máy sẽ xem tệp có đuôi là C.

- khi ta không đưa đường dẫn và thư mục thì tệp sẽ được lưu và thư mục chủ.

3. Một số menu trong môi trường kết hợp TURBO C

3.1. Menu File

Menu File có các chức năng sau:

- New: Báo hiệu cần tạo ra một tệp mới. Hệ thống sẽ chuyển vào chức năng soạn thảo. Tên tệp ngầm định là NONAME.C

- Open (F3): Mở các tệp được lưu trữ. Chọn một tệp bất kỳ để làm việc.

- Save (F2): Ghi tệp đang soạn thảo vào đĩa.

- Save as: Ghi tệp đang soạn thảo vào đĩa với một tên khác.

- Save all: Ghi tất cả các tệp đang soạn thảo vào đĩa.

- Change dir: Hiện nội dung thư mục hiện tại và cho phép đổi ổ đĩa hoặc thư mục.

- Print: In nội dung tệp.

- DOSS SHELL: Tạm thời rời bỏ môi trường kết hợp TURBO C và trở về DOS. Để quay trở lại môi trường kết hợp TURBO C ta bấm phím Exit.

- Quit (Alt – X): Ra khỏi TURBO C.

3.2. Menu Run

Menu Run có các chức năng sau:

- Run (Ctrl – F9): Khởi động quá trình tự dịch, liên kết và chạy chương trình thông qua các thông tin đã chuẩn bị sẵn trong chức năng Project name của menu Project. Nếu không dùng Project name thì chương trình có tên trong hộp sang Primary C file (menu Compile) được xét. Nếu hộp sáng này rỗng thì chương trình đang soạn thảo được xét.

- Program reset (Ctrl – F2): Lập lại chế độ thực hiện toàn bộ chương trình.

- Go to cusor (F4): Thực hiện chương trình cho đến dòng lệnh chứa con trỏ.

- Trace into (F7): Thực hiện từng lệnh một

- Step over (F8): Thực hiện từng câu lệnh và xem lời gọi hàm là một câu lệnh, như vậy sẽ nhẩy qua một hàm.

- Arguments: Xuất hiện hộp sáng để ta gõ vào các tham số dòng lệnh để chương trình chạy được chính xác hơn trong môi trường C.

3.3. Menu Comlile

Gồm các chức năng sau:

- Compile: Dịch chương trình gốc và liên kết với các hàm thư viện nhằm tạo ra tệp chương trình thực hiện có đuôi là EXE.

- Make: Dịch và liên kết để tạo thành tệp EXE. Các tệp nguồn được xác định như trong Compile.

- Link: Liên kết các tệp OBJ để tạo thành tệp chương trình thực hiện đuôi EXE.

- Build all: Dịch lại và liên kết để tạo thành tệp EXE. Các tệp nguồn được xác định như trong Compile.

- Information: hiển thị hộp sáng chứa thông tin về tệp chương trình sau khi được dịch như: tên tệp, tên thư mục, tổng số cảnh báo, tổng số lỗi,…

3.4. Menu Project

Gồm các chức năng sau:

- Open Project: Mở một project.

- Close Project: Đóng một project.

- Add Item: Thêm thành phần vào Project

- Delete Item: Xóa thành phần trong Project

3.5. Menu Option

Gồm các chức năng sau:

- Compiler: Dùng để chọn mô hình bộ nhớ và xác định độ dài cực đại của tên trong chương trình.

- Linker: Chọn phương thức liên kết

- Enviroment: Chọn môi trường.

- Directory: Đây là chức năng hay dùng đê thiết lập các thư mục liên quan tới quá trình dịch.

- Save: Ghi lại các thay đổi trong menu Option

TÀI LIỆU THAM KHẢO

[1] Đỗ Xuân Lôi, *Cấu trúc dữ liệu và giải thuật*, Nhà xuất bản Đại học Quốc gia Hà Nội, 2010.

[2] Nguyễn Xuân Huy, *Thuật toán*, Nhà xuất bản Khoa học và kỹ thuật.

[3] Nguyễn Thanh Thủy, *Nhập môn lập trình ngôn ngữ C*, Nhà xuất bản Khoa học và Kỹ thuật.

[4] Phạm Văn Ất, Kỹ thuật lập trình C cơ sở và nâng cao – Nhà xuất bản thống kê, 2003.

[5] Quách Tuấn Ngọc, Ngôn ngữ lập trình C, Nhà xuất bản Giáo dục, 1998.

[6] Nguyễn Đình Tê, Hoàng Đức Hải, Giáo trình lý thuyết và bài tập ngôn ngữ C, Nhà xuất bản Giáo dục, 1999.

[7] Brian W. Kernighan, Dennis M. Ritchie, *C Programming Language*,1988, Second Edition

[8] H.M. Deiteil & P.J. Deitel, *C: How to Program*, Prentice Hall International, 1994, Second Edition.