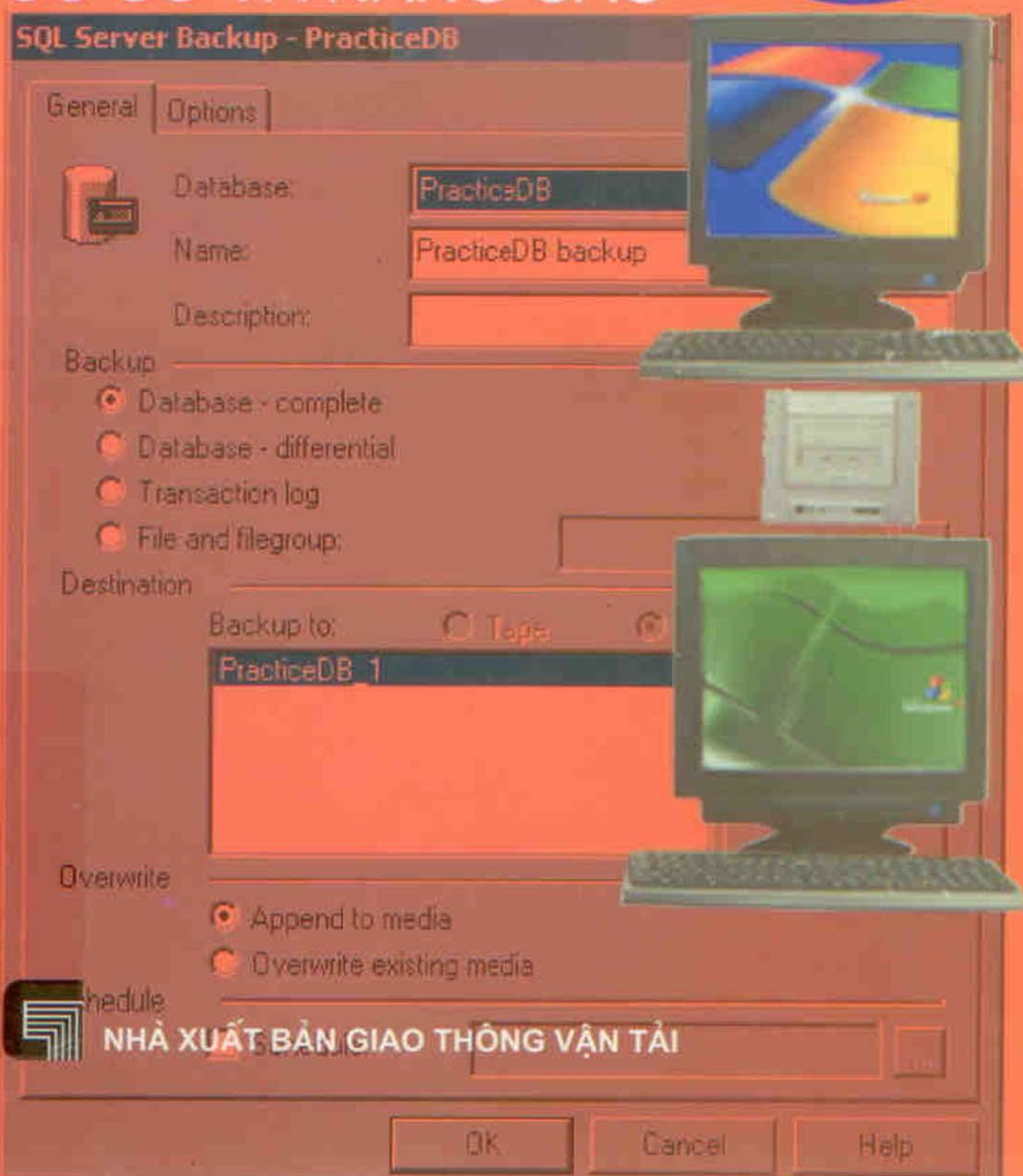
ẠM VĂN ẤT

Ỹ THUẬT LẬP TRÌNH C

CƠ SỞ VÀ NÂNG CAO



NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

GS. PHẠM VĂN ẤT

KỸ THUẬT LẬP TRÌNH C

CƠ SỞ VÀ NÂNG CAO

(Tái bản lần thứ 6 có sửa chữa, bổ sung)

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI
HÀ NỘI - 2006

GS. PHẠM VĂN ẤT

KỸ THUẬT LẬP TRÌNH C

CƠ SỞ VÀ NÂNG CAO

(Tái bản lần thứ 6 có sửa chữa, bổ sung)

NHÀ XUẤT BẢN
HÀ NỘI

LỜI NÓI ĐẦU*

Cuốn sách này là sự tiếp nối cuốn "Ngôn ngữ C - Lý thuyết và thực hành với 80 chương trình mẫu trong KHKT và Quản lý kinh tế " của tác giả.

Nội dung cuốn sách gồm hai phần: Cơ sở và nâng cao.

Phần cơ sở (10 chương đầu) dựa trên nền của cuốn trước nhưng thêm vào khá nhiều tư liệu mới cho đầy đủ như trong các chương. mục: Hàm đệ quy, con trỏ hàm, cấu trúc, danh sách móc nối, kỹ thuật đồ hoạ và tổ chức tệp.

Phần nâng cao (các chương còn lại và một vài vấn đề khó ở cuối chương 6, 7, 9, 10) là các tư liệu hoàn toàn mới chưa có ở cuốn sách trước. Bạn đọc có thể tìm thấy ở đây nhiều vấn đề bổ ích và lý thú như: Kỹ thuật tạo ảnh chuyển động, in ảnh từ màn hình đồ hoạ, chơi nhạc trên máy, kiến trúc bộ nhớ 8086 và cách truy cập trực tiếp vào bộ nhớ, sử dụng các chức năng sâu của DOS và BIOS, điều khiển chuột, cách lập hàm xử lý ngắt và chương trình thường trú, thay đổi chức năng các phím, lập trình theo thời gian thực và lập trình hướng sự kiện, thiết kế trò chơi đuổi bắt, tổ chức chương trình trên nhiều tệp, tạo chương trình COM, kết hợp giữa C và Assembler, xây dựng menu một mức, menu 2 mức trên môi trường văn bản và đồ hoạ, cách lập các hàm mà số đối (tham số hình thức) không cố định.

Trong sách đưa vào hơn 200 hàm chuẩn chọn lọc của Turbo C đủ để bạn đọc tra cứu và sử dụng.

Các vấn đề lý thuyết được minh hoạ trên nhiều chương trình chọn lọc đã thử nghiệm trên máy.

Sách gồm 18 chương và 13 phụ lục:

Chương 1 ngoài việc giới thiệu các khái niệm cơ bản 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.

---

* *Trong lần tái bản này có bổ sung thêm phụ lục 13 trình bày kỹ thuật bắt phím tổng quát và một số hàm tiện ích như: các hàm tạo menu, hàm vẽ đồ thị cho hàm một biến thực.*

Chương 2 trình bày các kiểu dữ liệu, cách biểu diễn các giá trị dữ liệu và cách tổ chức (lưu trữ) dữ liệu trong biến và mảng.

Chương 3 trình bày về các cách xử lý dữ liệu đơn giản nhờ các phép toán, biểu thức và câu lệnh gán.

Chương 4 trình bày các hàm vào ra dữ liệu trên bàn phím, màn hình và máy in.

Chương 5 trình bày về một lớp toán tử rất quan trọng dùng để thể hiện các thuật toán, đó là toán tử nhẩy goto, toán tử rẽ nhánh if, toán tử lựa chọn switch và các toán tử tạo lập chu trình (vòng lặp) for, while, do - while.

Chương 6 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 7 trình bày về một kiểu dữ liệu quan trọng là cấu trúc. Cũng sẽ nói về các hàm trên cấu trúc, cấu trúc tự trỏ và danh sách liên kết.

Chương 8 trình bày về việc quản lý màn hình văn bản và cách xây dựng cửa sổ. Một ví dụ hay ở đây là chương trình mô phỏng quy trình chuyển thấp trên màn hình mầu.

Chương 9 trình bày các hàm đồ hoạ để vẽ các hình cơ bản và kỹ thuật tạo ảnh chuyển động. Ở đây có nhiều ví dụ hay như chương trình vẽ tầu vũ trụ chuyển động, chương trình mô phỏng đồng hồ chạy theo thời gian thực, ...

Chương 10 trình bày 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,.....

Chương 11 trình bày về cách lưu trữ dữ liệu và tổ chức bộ nhớ của chương trình.

Chương 12 trình bày các chỉ thị tiền xử lý giúp việc biên soạn, biên dịch chương trình hiệu quả hơn.

Chương 13 trình bày cách sử dụng ngắt mềm của DOS và BIOS để quản lý trực tiếp các thiết bị như ổ đĩa, màn hình, bàn phím và chuột.

Chương 14 trình bày về kiến trúc bộ nhớ của 8086, địa chỉ phân đoạn, địa chỉ thực và cách truy nhập trực tiếp vào bộ nhớ. Ở đây cũng có một số ví dụ hay như các hàm đưa thông tin trực tiếp vào bộ nhớ màn hình.

Chương 15 trình bày quy tắc viết các hàm xử lý ngắt cứng và các lập trình thường trú. Đây là một trong những vấn đề khó nhưng được diễn đạt một cách giản dị, dễ hiểu và được minh hoạ bằng nhiều ví dụ thú vị, như các chương trình thường trú thông báo thời gian, các chương trình thường trú quản lý bàn phím (thay đổi chức năng một số phím, làm vô hiệu hoá một số hoặc toàn bộ bàn phím, tạo các autotext để làm tăng tốc độ soạn thảo văn bản).

Chương 16 sẽ minh hoạ quy tắc tạo âm thanh, âm nhạc trên nhiều ví dụ, trong đó có chương trình chơi bản nhạc Lambada quen thuộc.

Chương 17 đưa ra các khái niệm mới mẻ về lập trình theo thời gian thực, lập trình hướng sự kiện và kỹ thuật trò chơi. Một ví dụ lý thú ở đây là trò chơi bắn một tàu lạ lượn trên bầu trời. Qua chương này, người đọc có thể hiểu được cách sử dụng chuột, cách thiết kế và xây dựng các trò chơi đuổi bắt trên màn hình đồ hoạ.

Chương 18 trình bày cách sử dụng các hàm viết bằng Assembler trong C.

Phụ lục 1 trình bày quy tắc xuống dòng và sử dụng các kí tự trống khi viết chương trình.

Phụ lục 2 có thể dùngđể tra cứu các hàm chuẩn thường dùng của C

Phụ lục 3 trình bày các bảng mã ASCII và mã quét.

Phụ lục 4 hướng dẫn cách cài đặt Turbo C vào đĩa cứng.

Phụ lục 5 giới thiệu chung về môi trường kết hợp của C

Phụ lục 6 trình bày về cách sử dụng hệ soạn thảo C dùng để biên soạn chương trình gốc.

Phụ lục 7 trình bày các dùng menu Project để dịch chương trình viết trên nhiều tệp.

Phụ lục 8 hướng dẫn cách dùng trình biên dịch TCC để dịch (từ môi trường DOS) các chương trình lớn viết trên nhiều tệp. Phương pháp này cho phép biên dịch các chương trình rất lớn viết trên vài ngàn dòng lệnh.

Phụ lục 9 hướng dẫn phương pháp gỡ rối và chạy chương trình từng bước để dò tìm lỗi chương trình.

Phụ lục 10 trình bày 6 mô hình bộ nhớ của C. Cũng sẽ nói cách tạo tệp chương trình đuôi COM bằng cách dịch theo mô hình Tiny *trong chế độ dòng lệnh TCC (xem phụ lục 8). Cũng cần nói thêm, khi biên dịch thường nhận được các tệp chương trình đuôi EXE.*

Phụ lục 11 trình bày tóm tắt các hàm của Turbo C theo thứ tự ABC

Phụ lục 12 trình bầy cách xây dựng các hàm với số đối bất định, như các thủ tục writeln, readln của Pascal và các hàm printf, scanf của C. Công cụ chủ yếu được dùng là con trỏ và danh sách.

Phụ lục 13 trình bầy cách bắt một phím hoặc một tổ hợp phím bất kỳ (như tổ hợp 2 phím Ctrl End, tổ hợp 3 phím Ctrl Alt Del) và trình bầy một số hàm tiện ích thường dùng khi cài đặt chương trình như các hàm tạo menu trong chế độ văn bản và đồ hoạ, hàm vẽ đồ thị cho hàm có một biến số thực.

Khi viết chúng tôi đã cố gắng để giáo trình được hoàn chỉnh, song chắc chắn không tránh khỏi thiếu sót, vì vậy rất mong nhận được sự góp ý của đọc giả để giáo trình ngày một hoàn thiện hơn.

Hà Nội 10 - 2005

Tác giả

CHƯƠNG 1

CÁC KHÁI NIỆM CƠ BẢN

Trong chương này sẽ giới thiệu những thàng 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 và tên. Để có thể lập được một chương trình đầy đủ, chúng tôi cũng sẽ trình bầy đôi điều về câu lệnh gán, các câu lệnh vào ra, toán tử #include và những qui tắc cần lưu ý khi viết chương trình. Ngoài ra để giúp bạn đọc mau chóng tiếp cận với máy, chúng tôi sẽ giới thiệu vài chương trình đơn giản nhưng hoàn chỉnh và cách vận hành chúng trên máy để nhận được kết qủa cuối cùng. Tất cả những điều nói trên là bổ ích và đáng ghi nhớ vì chúng sẽ được thường xuyên sử dụng sau này. Đọc xong Chương 1 bạn có thể lập được một số chương trình đơn giản và biết cách thực hiện chương trình trên máy.

§1. TẬP KÝ TỰ DÙNG TRONG NGÔN NGỮ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để lập lên các từ (xem phụ lục 1). Đến lượt mình, các từ lại được liên kết theo một qui tắc nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và diễn đạt một thuật toán để giải một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau:

26 chữ cái hoa: A B C ... Z

26 chữ cái thường: a b c ... z

10 chữ số: 0 1 2 ... 9

Các ký hiệu toán học như: + - * / = ( )

Ký tự gạch nối: _ (chú ý phân biệt với dấu -)

Các ký hiệu đặc biệt khác như: . ,;: [] { } ?! \ & | % # $,...

Dấu cách (space) thực sự là một khoảng trống dùng để tách các từ. Ví dụ HA NOI gồm 6 ký tự, còn HANOI gồm 5 ký tự.

Chú ý: Khi viết chương trình ta không được sử dụng bất kỳ ký hiệu nào khác ngoài tập các ký tự nói trên.

Chẳng hạn khi giải phương trình bậc hai:

$ax^2 + bx + c = 0$

ta cần tính biệt thức:

$\Delta = b^2 - 4ac$

Ký tự) không cho phép dùng trong ngôn ngữ C, vì vậy ta phải dùng một cách ký hiệu khác như d hay delta.

§2. TỪ KHÓA

Từ khóa là những từ có một ý nghĩa hoàn toàn xác định. Chúng thường được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Sau đây là các khóa từ của TURBO C:

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

Ý nghĩa và cách sử dụng của chúng sẽ được lần lượt giới thiệu ở các mục sau. ở đây ta chỉ cần nhớ hai điều:

- Không được dùng từ khóa để đặt tên cho các hằng, biến, mảng, hàm, ...
- Từ khóa phải được viết bằng chữ thường. Chẳng hạn không được viết INT mà phải viết int.

§3. TÊN

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đối tượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc ,tên nhãn,... Tên được đặt theo quy tắc sau:

Tên là một dãy các ký tự: chữ, số và dấu gạch nối. Ký tự đầu của tên phải là chữ hoặc dấu gạch nối. Tên cần không được trùng với từ khóa (xem §2). Độ dài cực đại của tên mặc định là 32, nhưng có thể đặt lại một giá trị từ 1 đến 32 trong chức năng: Option - Compiler - Source - Identifier length trong môi trường phát triển kết hợp của C (phụ lục 5).

Các ví dụ đúng về tên:

a_1 BETA x1 delta_7 _x1

Các ví dụ sai về tên:

3XYZ_7 (Ký tự đầu tiên là số)

r#3 (sử dụng ký tự #)

f(x) (sử dụng các dấu ngoặc tròn)

case (trùng với từ khóa)

be ta (sử dụng khoảng trống)

X-1 (sử dụng dấu gạch ngang)

Chú ý: Trong các tên, chữ hoa và chữ thường được xem là khác nhau, như vậy tên AB khác tên ab. Trong C thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết các đôi tượng khác như biến, mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

§4. VÍ DỤ VỀ CHƯƠNG TRÌNH C

Dưới đây là các ví dụ nhằm minh hoạ cấu trúc của một chương trình C. Độc giả dễ dàng hiểu được mỗi câu lệnh qua các giải thích viết giữa các dấu /* */

Ví dụ 1. Viết chương trình cho hiện lên màn hình hai dòng chữ:

TURBO C HAN HANH

LAM QUEN VOI BAN

Dưới đây là hai chương trình cùng thực hiện yêu cầu đề ra.

```
/* Chương trình in 2 dòng chữ. Bản 1 */
#include "stdio.h" /* sử dụng thư viện vào ra chuẩn */
#include "conio.h"
void main () /* hàm chính */
  {
    /* Xuống dòng (\n) và in: TURBO C HAN HANH */
    printf("\nTURBO C HAN HANH");
    /* Xuống dòng và in: LAM QUEN VOI BAN */
    printf("\nLAM QUEN VOI BAN");
    getch(); /* Tạm dừng máy để xem kết quả */
  }
/* Chương trình in 2 dòng chữ. Bản 2 */
#include "stdio.h" /* sử dụng thư viện vào ra chuẩn */
#include "conio.h"
```

```
void main () /* hàm chính */
{
   /* Xuống dòng, in: TURBO C HAN HANH;
   lại xuống dòng rồi in: LAM QUEN VOI BAN */
   printf("\nTURBO C HAN HANH\nLAM QUEN VOI BAN");
   getch(); /* Tạm dừng máy để xem kết quả */
}
```

Nhận xét: Mỗi câu lệnh printf trong bản 1 in được một dòng. Câu lệnh printf trong bản 2 in được hai dòng.

Ví dụ 2. Chương trình dưới đây tính chu vi và diện tích hình tròn theo giá trị bán kính r nhập từ bàn phím.

```
#include "stdio.h"
#include "conio.h"
#include "math.h" /*Sử dụng thêm thư viện các hàm toán học*/
void main()
  {
    float r,cv,dt; /* Khai báo 3 biến thực */
    /* Đưa ra man hình thông báo về yêu cầu nhập số liệu */
    printf("\nBan kinh r= ");
    /* Nhập một giá trị thực đưa vào biến r */
    scanf("%f",&r);
    /* Tính chu vi và diên tích hình tròn
    Dùng hằng M_PI (PI) đã định nghĩa trong math.h
    */
    cv = 2*M_PI*r; dt = M_PI*r*r;
    /* In kết quả */
    printf("\nChu vi= %10.2f\nDien tich= %10.2f", cv,dt);
    getch(); /* Tạm dừng máy để xem kết quả */
  }
```

§5. MỘT SỐ QUI TẮC CẦN NHỚ KHI VIẾT CHƯƠNG TRÌNH

Bây giờ chúng ta vừa giải thích đôi điều về ví dụ trên,vừa rút ra một số chú ý khi viết chương trình.

Quy tắc đầu tiên cần nhớ là: mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải được kết thúc bằng dấu ;

Nhìn vào ví dụ 2 của §4 ta thấy:

+ Câu lệnh printf được viết trên 2 dòng.

+ Hai câu lệnh gán để tính cv và dt được viết trên một dòng.

Một điểm cần lưu ý ở đây là cách viết một hằng xâu ký tự (dãy ký tự đặt trong hai dấu nháy kép) trên nhiều dòng. Để báo cho TURBO C biết một chuỗi ký tự vẫn còn tiếp tục ở dòng dưới,ta đặt thêm dấu / trước khi xuống dòng. Ví du:

```
printf("\nTURBO C HAN HANH/
\nLAM QUEN VOI BAN");
```

Những điều nói trên nằm trong một nguyên lý tổng quát hơn sẽ trình bày trong phụ lục 1: Quy tắc sử dụng dấu cách giữa các từ và quy tắc xuống dòng khi viết chương trình.

Quy tắc thứ hai là quy tắc viết các lời giải thích. Các lời giải thích cần được đặt giữa dấu /* và dấu */ và có thể được viết:

- Trên một dòng.
- Trên nhiều dòng.
- Trên phần còn lại của một dòng.

Những lời giải thích không có tác dụng đối với sự làm việc của chương trình trên máy tính. Chúng chỉ có tác dụng đối với người đọc.

Qui tắc thứ 3 là qui tắc sử dụng các hàm chuẩn. Trong chương trình trên có dùng hàm chuẩn printf. Hàm này có trong tệp stdio.h (trong thư mục của C), vì vậy ở đầu chương trình cần viết:

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

Ta cũng chú ý rằng: cuối dòng này không có dấu; như cuối một câu lệnh (lệnh #include trình bầy trong Chương 12).

Tóm lại trước khi sử dụng một hàm cần biết nó nằm trên tệp nào và phải dùng toán tử #include để gắn tệp đó vào tệp chương trình của ta.

Qui tắc thứ 4 nói về cấu trúc của một chương trình. Một chương trình có thể chỉ có một hàm chính (main) như các ví dụ trên, hoặc có thể thêm vài hàm khác. Điều này sẽ được trình bày chi tiết trong Chương 6.

§6. KHAI BÁO VÀ TOÁN TỬ GÁN

Vấn đề khai báo sẽ nói kỹ trong chương 2. ở đây chúng ta chỉ cần biết vài điều sơ lược. Thứ nhất là: mọi biến trước khi sử dụng đều phải khai báo để xác định kiểu của nó. Để khai báo các biến nguyên (kiểu int) ta dùng từ khoá int. Đối với biến thực (kiểu float) ta dùng từ khoá float.

Ví dụ:

```
int a,b,c;   /* khai báo các biến a,b,c kiểu int */
float x,y,z; /* khai báo các biến x,y,z kiểu float */
```

Sự khác nhau giữa biến kiểu int và biến kiểu float là ở chỗ: biến kiểu int luôn luôn nhận giá trị nguyên trong qúa trình tính toán còn biến kiểu float có thể nhận cả các giá trị không nguyên.

Câu lệnh gán sẽ nói kỹ trong Chương 3. ở đây ta có thể hiểu toán tử gán có dạng:

```
b = bt;
```

Trong đó b là một biến, còn bt là một biểu thức toán học nào đó. Tác dụng của câu lệnh này là: trước tiên tính biểu thức bt và sau đó gán giá trị tính được cho biến b.

Ví dụ: sau khi thực hiện đoạn chương trình

```
float x;
x = 10.5;
x = 2*x - 2.5;
```

biến x sẽ nhận giá trị là 18.5.

Ta cần chú ý phân biệt toán tử gán với khái niệm đẳng thức trong toán học.

§7. ĐƯA KẾT QỦA LÊN MÀN HÌNH

Để đưa kết qủa ra màn hình ta dùng câu lệnh

```
printf (chuỗi điều khiển, bt1,bt2, ...,btk);
```

Ở đây bt1, bt2,..., btk là các biểu thức mà giá trị của chúng cần được đưa ra màn hình. Chuỗi điều khiển bao gồm ba loại ký tự:

- Ký tự điều khiển việc chuyển xuống đầu dòng tiếp theo.
- Các ký tự hiển thị.
- Các ký tự dùng để mô tả kiểu cách đưa ra của các biến, ta sẽ gọi chúng là các đặc tả. Mỗi biểu thức cần phải có một đặc tả tương ứng. Bây giờ sẽ giải thích từng loại ký tự.

- Ký tự điều khiển việc chuyển dòng là \n
- Các ký tự khác \n và khác các đặc tả là ký tự hiển thị và chúng sẽ được sao chép một cách nguyên xi ra màn hình.
- Bây giờ nói về các đặc tả. Đối với biểu thức nguyên có thể dùng đặc tả

%[fw]d

trong đó fw là một số nguyên xác định độ rộng tối thiểu dành cho trường ra (số vị trí tối thiểu trên màn hình dành cho một biến kiểu int).

Ở đây và sau này ta sử dụng qui ước: một thành phần đặt trong hai dấu [] là một thành phần không bắt buộc. Nghiã là nó có thể vắng mặt hay có mặt. Như vậy fw là một thành phần không bắt buộc.

Khi fw lớn hơn độ dài thực tế của trường ra (số ký tự của số nguyên, ví dụ số -345 có độ dài thực tế là 4, số 12467 có độ dài thực tế là 5), thì một số khoảng trống sẽ được bổ sung vào bên trái cho đủ fw vị trí.

Khi không có mặt fw hoặc khi fw nhỏ hơn hay bằng độ dài thực tế của trường ra, thì độ rộng trên màn hình dành cho trường ra sẽ bằng độ dài thực tế của nó.

Đối với biến thực có thể dùng đặc tả

%[fw][.pp]f

Ở đây pp là độ chính xác. Nói một cách cụ thể hơn: trên màn hình sẽ hiện lên một giá trị thực có pp chữ số sau dấu chấm thập phân. Nếu pp = 0, biến thực được đưa ra như một số nguyên (không có dấu chấm thập phân). Máy sẽ ngầm hiểu là pp bằng 6 khi nó vắng mặt (giá trị mặc định của pp là 6), fw là một số nguyên xác định độ rộng tối thiểu trên màn hình dành cho trường ra.

Khi fw lớn hơn độ dài thực tế của trường ra thì một số khoảng trống sẽ được bổ sung vào bên trái.

Khi fw vắng mặt hoặc khi fw nhỏ hơn hay bằng độ dài thực tế của trường ra thì độ rộng trên màn hình dành cho trường ra sẽ bằng độ dài thực tế của nó.

Độ dài thực tế của một biến thực bằng: số chữ số của phần nguyên cộng với pp, cộng với một vị trí dành cho dấu chấm thập phân (nếu pp>0), cộng với một vị trí dành cho dấu - (đối với số âm). Ví dụ độ dài thực tế của -23.46 bằng:

8 nếu pp = 4,

10 nếu pp vắng mặt,

3 nếu pp = 0.

Tất cả những điều nói trên được minh họa trong bảng sau:

Đặc tả	Giá trị của biến	Dạng đưa ra màn hình
%d	-456	|-456|
%d	456	|456|
%5d	456	| 456|
%5d	-456	| -456|
%8.0f	45.71	| 46|

%f	-45.63	|-45.630000|
%f	45.63	|45.630000|
%8.3f	-45.63	| -45.630|
%8.3f	45.6375	| 45.638|
%0.3f	-45.63	|-45.630|
%0.3f	45.63	|45.630|
%.2f	-0.345	|-0.35|
%.2f	0.345	|0.35|

Ở cột thứ ba ta sử dụng quy ước: số vị trí giữa hai dấu | | biểu thị số vị trí trên màn hình dành cho số được đưa ra.

Chú ý: Sau khi đọc những điều nói trên, một câu hỏi được đặt ra là làm thế nào để đưa ra các kí tự % ' " \

Câu trả lời như sau:

- Khi dấu % đứng ngoài kết cấu đặc tả thì nó được xem như ký tự thông thường, nghĩa là bản thân nó được đưa ra màn hình (máy in hoặc đĩa).

- Đối với các kí tự khác ta dùng thêm dấu \ đặt trước nó. Nói một cách cụ thể hơn:

Khi viết \' thì dấu ' được đưa ra.

Khi viết \" thì dấu " được đưa ra.

Khi viết \\ thì dấu \ được đưa ra.

Tất cả các quy tắc trình bầy trong mục này sẽ được minh họa trên các chương trình của §10.

§8. ĐƯA KẾT QỦA RA MÁY IN

Cách thức đưa kết quả ra máy in hoàn toàn tương tự như cách thức đưa ra màn hình. Sự khác nhau chỉ ở một vài chi tiết nhỏ như sau:

- Dùng lệnh fprintf thay cho lệnh printf.

- Đưa thêm tham số stdprn vào trước chuỗi điều khiển. Như vậy để đưa kết quả ra máy in ta dùng câu lệnh:

fprintf(stdprn, chuỗi điều khiển ,bt1,bt2,...,btk);

tham số stdprn chỉ ra rằng: thiết bị đưa ra là máy in, chuỗi điều khiển và các biểu thức bt1, bt2,...,btk có ý nghĩa hoàn toàn tương tự như đã trình bầy trong câu lệnh printf.

§9. VÀO SỐ LIỆU TỪ BÀN PHÍM

Để vào từ bàn phím hai giá trị kiểu int và ba giá trị kiểu float có thể dùng các câu lệnh sau:

```
int a,b;        /* khai báo 2 biến kiểu int */
float c,d,e;    /* khai báo 3 biến kiểu float */
scanf("%d%d%f%f%f",&a,&b,&c,&d,&e);
```

Đối với câu lệnh scanf cần chú ý các điểm sau:

- Không dùng tên biến như trong câu lệnh printf mà dùng địa chỉ của biến. Phép toán

& tên_biến

cho địa chỉ của biến.

- Mỗi biến ứng với một đặc tả. Như vậy số đặc tả bằng số biến.

- Dùng đặc tả %d đối với biến nguyên và %f đối với biến thực. Một cách tổng quát câu lệnh scanf có dạng:

scanf("t1t2...tk", &b1,&b2,...,&bk);

trong đó: b1, b2,..., bk là các biến (kiểu int và kiểu float) còn t1, t2,..., tk là các đặc tả tương ứng.

Sự hoạt động của câu lệnh scanf: Khi gặp câu lệnh này, máy sẽ dừng để đợi thao tác viên vào số liệu từ bàn phím. Trong ví dụ trên cần vào 5 giá trị, trong đó 2 giá trị đầu là nguyên và 3 giá trị sau là thực. Các giá trị cần được phân cách nhau bởi một hoặc vài khoảng trắng (ở đây khoảng trắng được hiểu là dấu cách hoặc dấu xuống dòng \n). Chẳng hạn để vào 5 giá trị: 25, -137, 45.3, 23.4, -12.5, thao tác viên có thể sử dụng một trong các cách bấm phím sau:

Cách 1 (hai giá trị nguyên trên một dòng, 3 giá trị thực trên dòng tiếp theo)

25 -137

45.3 23.4 -12.5

Cách 2 (mỗi giá trị trên một dòng)

25

-137 45.3 23.4

-12.5

Cách 3 (đặt cả 5 giá trị trên một dòng)

25 -137 45.3 23.4 -12.5

Cũng có thể sử dụng các cách thao tác khác như hai số nguyên đặt trên một dòng, 3 số thực trên 3 dòng,... Khi đó hiệu qủa của câu lệnh scanf là: gán giá trị 25 cho a, -137 cho b, 45.3 cho c, 23.4 cho d và -12.5 cho e.

Một điểm cần đặc biệt chú ý ở đây là: nếu trong câu lệnh printf ta có thể dùng tên biến hoặc biểu thức, thì trong câu lệnh scanf ta phải dùng địa chỉ của biến.

§10. MỘT VÀI CHƯƠNG TRÌNH ĐƠN GIẢN

Nhờ các hàm printf, fprintf và scanf ta có thể lập được những chương trình đơn giản nhưng hoàn chỉnh. Mục này xét 2 chương trình minh họa việc sử dụng các hàm nói trên.

Chương trình 1

```
/* chuong trinh tinh x luy thua y */
#include "stdio.h"
#include "math.h"
main ()
  {
    double x,y,z; /* khai bao ba bien kieu double */
    printf("\n vao x va y ");
    scanf("%lf%lf",&x,&y);    /* vao x,y tu ban phim */
    z=pow(x,y);               /* tinh x luy thua y va gan cho z */
                              /* in ket qua tren 3 dong */
    fprintf(stdprn,"\nx = %8.2lf\ny = %8.2lf\nz = %8.2lf",x,y,z);
  }
```

Chương trình 2

```
/* chuong trinh minh hoa cac kha nang dua ra */
#include "stdio.h"
main()
 {
   int a,c,t; float b,d;
   a = 123; c = -4685; t = 12;
   b = -45.855; d = 123.425;
   fprintf(stdprn,"\n\'Chuc cac ban may man \'\n");
   fprintf(stdprn,"\n\"Chuc cac ban may man \"");
   fprintf(stdprn,"\n\n\\Chuc cac ban may man\\\n\n");
   fprintf(stdprn,"\"Tong san luong hang nam tang %2d %\"",t);
     /* giua cac so dua ra khong co khoang cach */
   fprintf(stdprn,"\n\n%d%f%d%f",a,b,c,d);
     /* giua cac so dua ra co nhung khoang trong */
   fprintf(stdprn,"\n\n%6d%8.2f%7d%8.2f",a,b,c,d);
```

```
/* giua cac so co dat them cac ky tu khac */
fprintf(stdprn,"\n\na = %d, b = %0.2f, c=%d, d=%0.2f", a,b,c,d);
}
```

Kết quả thực hiện chương trình

'Chuc cac ban may man'

"Chuc cac ban may man"

\Chuc cac ban may man\

"Tong san luong hang nam tang 12 %"

123-45.855000-4685123.425000

123 -45.85 -4685 123.43

a = 123, b = -45.85, c = -4685, d = 123.43

§11. 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 kết qủa cuối cùng. Đây là phần việc có tính chất tác nghiệp ít đòi hỏi suy nghĩ và sáng tạo hơn so với công việc lập trình. Quá trình vận hành một chương trình 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 (Cách cài đặt sẽ trình bầy trong phụ lục 4).

1. Tạo tệp chương trình gốc.

Bấm các phím:

C:\TC>TC và Enter sẽ nhận được menu chính trên màn hình với nội dung sau:

File Edit Run Compile Project Option Debug Break/Watch

Menu chính gồm các menu: File, Edit,.... Ta cần sử dụng menu File. Muốn vậy cần bấm đồng thời hai phím Alt và F. Khi đó ta nhận được menu File với nội dung sau:

Load F3
Pick Alt-F3
New
Save F2
Write to

Directory

Change dir

OS shell

Quit Alt-X

Menu File gồm các chức năng: Load, Pick,.. Bây giờ ta cần dùng chức năng New để tạo một tệp chương trình mới. Muốn vậy, ta đưa con trỏ đến hộp sáng New và bấm Enter. Ta cũng có thể đạt được điều này bằng cách bấm phím N. 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 (đưa các dòng lệnh lên màn hình). Khi nạp sai ta có thể dịch chuyển con trỏ tới các vị trí (dòng và cột) cần thiết để sửa sai. Chương trình mặc dầu đã đưa lên màn hình nhưng nó vẫn chưa được ghi lên đĩa. Để ghi chương trình lên đĩa dưới dạng một tệp ta cần sử dụng chức năng Write to. Muốn vậy, đầu tiên cần bấm đồng thời hai phím alt và F để trở về menu File. Sau đó bấm phím W (hoặc dịch chuyển con trỏ đến hộp sáng Write to và bấm phím Enter). Chức năng Write to làm việc và trên màn hình xuất hiện hộp sáng:

New Name

Ta cần thông báo cho máy biết tên tệp chương trình gốc mà ta định đặt. Giả sử ta muốn tệp chương trình gốc có tên là THU1, khi đó ta lần lượt bấm 5 phím THU1 và ENTER. Chương trình sẽ được ghi lên đĩa dưới dạng một tệp có tên

THU1.C

Đến đây quá trình soạn thảo xem như hoàn thành.

Chú ý: Ta có thể sửa một tệp cũ hoặc tạo một tệp mới bằng cách dùng chức năng File-Load (chức năng load của menu File). Khi hỏi đến tên tệp, ta nạp vào tên tệp cần sửa hay tên tệp cần tạo. Nếu không nói rõ đuôi, trình soạn thảo sẽ hiểu là đuôi C.

2. Dịch chương trình.

Để dịch chương trình ta dùng menu Compile. Muốn vậy ta đồng thời bấm các phím alt và C. Ta sẽ nhận được menu Compile với các chức năng sau:

Compile to OBJ

Make EXE file Link EXE file

Build all

Primary C file

Get into

Ta cần dùng chức năng Primary C file để xác định tên tệp chương trình C cần dịch. Muốn vậy ta bấm phím P (hoặc đưa con trỏ đến hộp sáng Primary C file và bấm ENTER). Chức năng Primary C file thực hiện và trên màn hình xuất hiện hộp sáng:

```
┌──────── Primary File ────────┐
│ *.C                          │
└──────────────────────────────┘
```

Bây giờ ta cần thông báo cho máy biết tên tệp chương trình cần dịch. Chẳng hạn nếu muốn dịch chương trình THU1.C thì ta bấm các phím: T H U 1 và Enter. Tiếp đó ta dùng chức năng Make EXE file để dịch chương trình gốc và liên kết với các hàm thư viện nhằm tạo ra tệp chương trình thực hiện (đuôi EXE). Muốn vậy, ta bấm phím M (hoặc đưa con trỏ đến hộp sáng Make EXE file và bấm phím ENTER). Chức năng Make EXE file hoạt động, máy sẽ thực hiện việc dịch và liên kết. 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:

THU1.EXE

(gọi là tệp chương trình thực hiện). Đến đây quá trình dịch xem như hoàn thành.

Ta ra khỏi TC và trở về hệ thống bằng cách:

- Đầu tiên trở về menu File (Alt-F)

- Sau đó bấm phím Q (hoặc đưa con trỏ tới hộp sáng Quit Alt-X và bấm Enter).

Tóm lại sau khi thực hiện 2 bước vừa nêu trên ta nhận được tệp chương trình gốc THU1.C và tệp chương trình thực hiện là THU1.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.

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 THU1.EXE ta bấm các phím: T H U1 và Enter, khi đó chương trình bắt đầu làm việc. Nếu nạp đủ các số liệu cần thiết chương trình sẽ tính toán và đưa ra các kết qủa cuối cùng trên màn hình (máy in hoặc đĩa).

Chú ý:

+ Có thể chạy một chương trình (sau khi soạn thảo) trong môi trường Turbo C (không cần trở về DOS) bằng cách bấm đồng thời các phím Ctrl-F9.

+ Từ trong môi trương C có thể trở về DOS theo 2 cách:

Cách 1: Dùng chức năng File-Quit. Khi đó sẽ ra khỏi C để trở về DOS. Muốn trở lại C phải dùng trình TC (Bấm 3 phím T C Enter).

Cách 2: Dùng chức năng File-OS Shell. Khi đó chỉ tạm thời ra khỏi C. Muốn trở lại C ta bấm các phím EXIT và Enter.

BÀI TẬP CHƯƠNG 1

Bài 1. Thực hiện các chương trình viết trong các mục §4 và §10 trên máy. Đối chiếu kết qủa nêu trong sách với kết qủa nhận được.

Bài 2. Viết chương trình in một mẫu biểu đơn giản tự chọn, sau đó vận hành chương trình trên MTĐT để nhận được kết qủa do máy in ra.

Bài 3. Lập chương trình tính tích của 4 số thực được nạp từ bàn phím, sau đó thực hiện chương trình trên máy (dùng dấu * để biểu thị phép nhân).

Bài 4. Cho biết lương kỳ hai của cán bộ tính theo công thức:

$$lk2 = \frac{bl \times n}{26} - lk1$$

trong đó bl là bậc lương, n là số ngày công trong tháng, lk1 là các khoản tiền đã lĩnh ở kỳ một. Lập chương trình nhập bl, n, lk1 từ bàn phím, sau đó tính lk2 theo công thức trên. Thực hiện chương trình trên máy với các số liệu cụ thể (dùng dấu / để biểu thị phép chia, ví dụ a/b = a:b).

CHƯƠNG 2

HẰNG, BIẾN VÀ MẢNG

Trong C sử dụng các dạng thông tin (kiểu giá trị) sau: số nguyên (int), số thực hay số dấu phẩy động (float), số dấu phẩy động có độ chính xác kép (double) và ký tự (char). Hằng chính là một giá trị thông tin cụ thể. Biến và mảng là các đại lượng mang tin. Mỗi loại biến (mảng) có thể chứa một dạng thông tin nào đó, ví dụ biến kiểu int chứa được các số nguyên, biến kiểu float chứa được các số thực. Để có thể lưu trữ thông tin, biến và mảng được cấp phát bộ nhớ. Người ta lại chia biến (mảng) thành: biến (mảng) tự động, biến (mảng) ngoài, biến (mảng) tĩnh.

Biến (mảng) tự động chỉ tồn tại (được cấp phát bộ nhớ) khi chúng được sử dụng. Biến (mảng) ngoài và tĩnh tồn tại trong suốt thời gian làm việc của chương trình. Cách tổ chức như vậy vừa tiết kiệm bộ nhớ (vì cùng một khoảng nhớ lúc thì phân cho biến này, lúc thì phân cho biến khác), vừa cho phép sử dụng cùng một tên cho các đối tượng khác nhau mà không gây ra một sự nhầm lẫn nào.

§1. KIỂU DỮ LIỆU

Trong C sử dụng các kiểu dữ liệu:

- Ký tự (char)
- Số nguyên (int)
- Số dấu phẩy động độ chính xác đơn (float)
- Số dấu phẩy động độ chính xác kép (double)

1.1 Kiểu char. Một giá trị kiểu char chiếm một bye (8 bit) và biểu diễn được một ký tự thông qua bảng mã aSCii (xem phụ lục 3). Ví dụ:

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
B	066
a	097
b	098

Có hai kiểu char: là signed char và unsigned char. Kiểu thứ nhất biểu diễn một số nguyên từ -128 đếnđ 127, kiểu thứ hai có giá trị từ 0 đến 255. Bảng dưới đây cho kích cỡ và phạm vi biểu diễn của giá trị kiểu char.

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

Ví dụ sau minh hoạ sự khác nhau giữa 2 kiểu trên. Xét đoạn chương trình:

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

Khi đó thực chất:

```
ch1 = -56
ch1 + 56 = 0
ch2 + 56 = 256
```

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

Phân loại ký tự: Có thể chia 256 ký tự thành 3 nhóm:

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

+ Nhóm thứ hai là các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể đưa ra màn hình và máy in.

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

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

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

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

1.3. Kiểu dấu phẩy động. Trong C cho phép sử dụng 3 loại giá trị dấu phẩy động là float, double và long double. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây:

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	kích thước (byte)
float	3.4E-38 -> 3.4E+38	7-8	4
double	1.7E-308 -> 1.7E+308	15-16	8
long double	3.4E-4932 -> 1.1E4932	17-18	10

Giải thích: Máy có thể lưu trữ được số kiểu float có giá trị tuyệt đối từ 3.4E-38 đến 3.4e+38. Số có giá 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. HẰNG

Hằng là các đại lượng mà giá trị của nó không thay đổi trong qúa trình tính toán. Dưới đây trình bầy các loại hằng được sử dụng trong C.

2.1. Hằng dấu phẩy động (float và double) được viết theo 2 cách

Cách 1 (dạng thập phân): Số gồm phần nguyên, dấu chấm thập phân và phần phân. Ví dụ:

214.35 -4563.48 234.0

Chú ý: Phần nguyên hay phần phân có thể vắng mặt nhưng dấu chấm không thể thiếu, ví dụ cho phép viết:

.34 15.

Cách 2 (dạng khoa học hay dạng mũ): Số được tách thành 2 phần là định trị và phần bậc. Phần định trị là một số nguyên hoặc số thực dạng thập phân, phần bậc là một số nguyên. Hai phần này cách nhau bởi ký tự e hoặc E. Ví dụ:

123.456E-4	(biểu diễn giá trị 0.0123456)
0.12E3	(biểu diễn giá trị 120.0)
-49.5e-2	(biểu diễn giá trị -0.495)
1E8	(biểu diễn giá trị 100000000.0)

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

-45 4007 635...

Chú ý phân biệt 125 và 125.0. Số 125 là hằng nguyên còn 125.0 là hằng thực (dấu phẩy động).

2.3. Hằng long được viết theo cách:

-4893L hoặc -4893l

(thêm L hoặc l vào đuôi). Một số nguyên vượt ra ngoài miền xác định của int cũng được xem là hằng long. Ví dụ:

4563946L và 4563946

là hai hằng long có cùng giá trị.

2.4. Hằng int hệ 8 được viết theo cách:

0c1c2c3...

Ở đây ci là một số nguyên trong khoảng từ 0 đến 7. Hằng nguyên hệ 8 luôn luôn nhận giá trị dương. Ví dụ:

0345

là một hằng nguyên hệ 8. Giá trị của nó trong hệ 10 là

3*8*8 + 4*8 + 5 = 229

2.5. Hằng nguyên hệ 16. Trong hệ này sử dụng 16 ký tự: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. Để tránh nhầm lẫn giữa chữ số 11 và hai số 1 người ta dùng qui ước sau:

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

Hằng số hệ 16 có dạng

0xc1c2c3...hoặc 0Xc1c2c3...

trong đó ci là một chữ số hệ 16. Ví dụ:

0xa9 , 0Xa9 , 0xA9, 0XA9

là 4 hằng số hệ 16 như nhau. Giá trị của chúng trong hệ 10 là

10*16 + 9 = 169

2.6. Hằng ký tự là một ký tự riêng biệt được viết trong 2 dấu nháy đơn, ví dụ 'a'. Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97 (xem bảng mã ASCII). Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác. Ví dụ

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

Hằng ký tự còn có thể được viết theo cách

'\c1c2c3'

trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã aSCii của ký tự cần biểu diễn. Ví dụ, chữ a có mã hệ 10 là 97 đổi ra hệ 8 là 0141. Vậy hằng ký tự 'a' có thể viết dưới dạng '\141'. Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu \):

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

Chú ý:

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

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

-Lệnh printf("%c%c%c",7,7,7); phát ra 3 tiếng chuông.

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

"Ha noi"

"Hai phong"

"" /* 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 xoá \0 vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

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

2.8. Tên hằng. Hiệu qủa của toán tử

#define MaX 1000

là: Tất cả các tên MAX trong chương trình xuất hiện sau toán này đều được thay bằng 1000. Vì vậy ta thường gọi MAX là tên hằng (hay macro), nó biểu diễn số 1000. Một ví dụ khác, toán tử

#define Pi 3.141593

đặt tên cho hằng float 3.141593 là Pi.

2.9. Vài ứng dụng của các hằng ký tự. Các hằng kiểu int, long, float, double thường dùng trong tính toán, còn các hằng ký tự và hằng xâu ký tự thường dùng trong in ấn (đưa ra các dòng thông tin văn bản, lời giải thích, chỉ dẫn,...).

Chương trình dưới đây minh họa cách viết các hằng và cách đưa chúng ra màn hình (máy in hoặc đĩa).

```
/* Chuong trinh minh hoa cach dung cac hang */
    #include "stdio.h"
    main ()
    {
        frintf(stdprn,"\n hang dau phay dong\: %10.2f %10.2f %10.2f
        %10.2f", 14689.2e-4,-0.1256666e3,23468e-2,1e4);
        frintf(stdprn,"\n\nhang nguyen %10ld %10ld %6ld %6d %6d"
        ,456398461,45638946,35L,35,-123);
        frintf (stdprn,"\n\nhang nguyen he 8 va he 16:\
        %7d %7d %7d %7d %7d",
        0345,0xa9,0xa9,0Xa9,0Xa9);
        /* In hằng ký tự dùng đặc tả %[fw]c */
        frintf (stdprn,"%c %c %6c %6c %6c %6c %6c%",
        '\n','\n', 'a', '\141','\'','\"','\\');
        /* In hằng xâu ký tự dùng đặc tả %[fw ]s */
        frintf(stdprn,"\n\n%10c%s",' '," Chuc may man ");
    }
```

§3. KIỂU ENUM

3.1.Câu lệnh enum có thể viết theo 4 dạng sau:

```
enum  tk  {pt1, pt2, ... } tb1, tb2, ...;
enum  tk  {pt1, pt2, ... };
enum      {pt1, pt2, ... } tb1, tb2, ...;
enum      {pt1, pt2, ... };
```

Trong đó:

tk là tên kiểu enum (một kiểu dữ liệu mới),

pt1, pt2, ... là tên các phần tử,

tb1, tb2, ... là tên biến kiểu enum.

3.2. Tác dụng:

+ Câu lệnh thứ nhất có các chức năng sau:

a. Định nghĩa các macro (như kiểu #define) pt1, pt2, ... với các giá trị nguyên liên tiếp tính từ 0. Nói cụ thể hơn: pt1=0, pt2=1, ... Chức năng này tương đương các câu lệnh #define sau:

```
#define pt1 0
#define pt2 1
...
```

b. Định nghĩa kiểu enum có tên là tk. Sau này có thể dùng cụm từ enum tk để khai báo các biến enum theo mẫu:

```
enum tk x,y,z;
```

c. Khai báo các biến kiểu enum có tên là tb1, tb2, ...

+ Câu lệnh thứ hai có các chức a và b.

+ Câu lệnh thứ ba có các chức a và c.

+ Câu lệnh thứ tư chỉ có chức a.

3.3. Biến kiểu enum. 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. Mục đích chính của enum là tạo ra các macro , các kiểu và các biến gợi nhớ. Ví dụ để làm việc với các ngày trong tuần ta có thể dùng kiểu week_day và biến day như sau:

```
enum week_day {SUNDAY,MONDAY,TUESDAY,WEDSDAY
               THURSDAY,FRIDAY,SATURDAY} day;
```

3.4. Ví dụ: chương trình sau minh hoạ các lệnh có thể dùng với các kiểu, các macro và các biến tạo bởi enum.

```
#include <stdio.h>
main()
{
    enum {T0,T1,T2};
    enum day {cn,t2,t3,t4,t5,t6,t7} n1;
    enum day n2;
    int i,j=2000,k=T2;
    i=t7;
    n1=-1000;
    n2=j;
    printf("\n n1= %d n2= %d i= %d",n1,n2,i);
    printf("\nk= %d T1= %d ",k,T1);
}
```

4.1. Khai báo

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

```
type tên biến;
```

Ví dụ:

```
main ()
{
    int a,b,c;              /* khai báo ba biến kiểu int */
    long ad,bd,cd;          /* khai báo ba biến kiểu long */
    unsigned au,bu,cu;      /* khai báo ba biến kiểu unsigned */
    char beta,alfa;         /* khai báo hai biến kiểu char */
    float x,y;              /* khai báo hai biến kiểu float */
    double xd,yd;           /* khai báo hai biến kiểu double */
    .
    .
    .
}
```

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

4.2. Vị trí của các khai báo. Các khai báo cần đặt ngay sau dấu { đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác . Như vậy, sau một câu lệnh gán chẳng hạn thì không được khai báo nữa . Sau đây là một ví dụ sai về vị trí của khai báo

```
main ()
  {
    int a,b,c;
    a = 35;
    int d;        /* Vị trí của khai báo sai */
    .
    .
    .
  }
```

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

```
int     a, b = 20, c , d = 40;
float   e = -35.1 ,  x =23.0 , y, z, t = 36.1;
```

Tất nhiên điều này có thể đạt được nhờ toán tử gán và về thực tế hai cách này là tương đương. Vậy để đạt được ý định như ví dụ trên ta có thể dùng các câu lệnh:

```
int a,b,c,d;
float e,x,y,z,t;
b = 20; d = 40; e = -35.1; x = 23.0; t = 36.1;
```

4.4. Lấy địa chỉ của biến. Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ biến dùng trong một số hàm như hàm scanf. Để nhận địa chỉ biến ta dùng phép toán:

& tên_biến

§5. MẢNG

5.1. Khái niệm về mảng, cách khai báo

Mỗi biến chỉ có thể chứa một giá trị. Để chứa một dãy số hay một bảng số ta có thể dùng nhiều biến nhưng cách này không tiện lợi. Việc sử dụng mảng là cách tốt hơn nhiều trong những trường hợp như vậy .

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

- kiểu mảng (int, float double,...)
- tên mảng,
- số chiều và kích thước mỗi chiều.

Khái niệm về kiểu mảng và tên mảng cũng giống như khái niệm kiểu biến và tên biến , điều này đã nói ở các mục trên . Ta sẽ giải thích khái niệm số chiều và kích thước mỗi chiều thông qua các ví dụ sau. Các khai báo:

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

sẽ xác định 4 mảng: a, b, x và y. ý nghĩa của chúng như sau:

- Đối với mảng thứ nhất thì kiểu là int, tên là a, số chiều là một , kích thước bằng 10. Mảng có 10 phần tử được đánh số như sau: a[0],a[1],a[2],....,a[9]. Mỗi phần tử a[i] chứa được một giá trị kiểu int và mảng a có thể biểu diễn được một dãy 10 số nguyên .

- Đối với mảng thứ hai thì kiểu là int, tên là b số chiều là 2. kích thước của các chiều là 4 và 2. Mảng có 8 phần tử, chúng được đánh số và được sắp xếp như sau:

b[0][0]	b[0][1]
b[1][0]	b[1][1]
b[2][0]	b[2][1]
b[3][0]	b[3][1]

Mỗi phần tử b[i][j] chứa được một giá trị kiểu int, và mảng b có thể biểu diễn được một bảng số nguyên 4 dòng, 2 cột.

- Đối với mảng thứ ba thì kiểu là float, tên là x, số chiều là một, kích thước bằng 5. Mảng có 5 phần tử được đánh số như sau:

x[0],x[1],x[2],x[3],x[4]

Mỗi phần tử x[i] chứa được một giá trị kiểu float, và mảng x có thể biểu diễn được một dãy 5 số thực.

- Đối với mảng thứ 4 thì kiểu là float, tên là y, số chiều là 2, kích thước của các chiều là 3. Mảng có 9 phần tử, chúng được đánh số và được sắp xếp như sau:

y[0][0]	y[0][1]	y[0][2]
y[1][0]	y[1][1]	y[1][2]
y[2][0]	y[2][1]	y[2][2]

Mỗi phần tử y[i][j] chứa được một giá trị kiểu float, và mảng y có thể biểu diễn được một bảng số thực 3 dòng, 3 cột .

Chú ý:

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

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

5.2. Chỉ số mảng.

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

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

Khi đó:

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

Các cách viết sau là sai:

y[j] vì y là mảng hai chiều, cần hai chỉ số

b[i][j][1] vì b là mảng hai chiều, chỉ cần hai chỉ số

Chú ý về chỉ số:

+ Biểu thức dùng làm chỉ số có thể thực.

Khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng.

Ví dụ:

a[2.4] = a[2]

a[1.9] = a[1]

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

5.3. Lấy địa chỉ phần tử mảng.

Dưới đây khi nói mảng ta hiểu là mảng một chiều . Mảng có từ hai chiều trở lên ta nói mảng kèm số chiều (ví dụ mảng hai chiều,...). Có một vài hạn chế trên các mảng nhiều chiều. Chẳng hạn có thể lấy địa chỉ phần tử mảng một chiều, nhưng nói chung không cho phép lấy địa chỉ phần tử mảng nhiều chiều.

Như vậy máy sẽ chấp nhận phép tính:

&a[i]

nhưng không chấp nhận phép tính

&y[i][j]

5.4. Địa chỉ đầu của mảng.

Một chú ý quan trọng là: Tên mảng biểu thị địa chỉ đầu của mảng.

Như vậy ta có:

a = &a[0]

Nhiều ví dụ về mảng có thể tìm thấy trong *Chương 4*.

§6. ĐỊNH NGHĨA KIỂU BẰNG TYPEDEF

6.1. Công dụng. Từ khoá typedef dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ dùng để khai báo dữ liệu sau này. Nên chọn một tên vừa ngắn gọn vừa gợi nhớ.

6.2. Cách viết. Đặt từ khoá typedef vào trước một khai báo thông thường, khi đó tên dữ liệu (biến, mảng, cấu trúc,...) trở thành tên kiểu.

Ví dụ câu lệnh:

```
typedef int ng;
```

sẽ đặt tên kiểu int là ng. Sau đó có thể dùng ng (cũng như int) để khai báo các biến, mảng nguyên như sau:

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

Tương tự các câu lệnh:

```
typedef float mt50[50];
typedef int m_20_30[20][30];
typedef enum {T1,T2,T3} T;
```

sẽ:

+ Đặt tên kiểu mảng thực một chiều 50 phần tử là mt50.
+ Đặt tên kiểu mảng nguyên 2 chiều kích cỡ 20x30 là m_20_30
+ Đặt tên kiểu enum có 3 phần tử {T1,T2,T3} là T.

Sau này ta có thể dùng các khai báo:

```
mt50 x,y;        /* các mảng thực một chiều 50 phần tử */
m_20_30 a,b;     /* các mảng nguyên 2 chiều kích cỡ 20x30 */
T t;             /* t là biến enum */
```

Tóm lại: Chỉ cần thêm từ khoá typedef vào một khai báo ta sẽ nhận được một tên kiểu dữ liệu và ta có thể dùng tên này để khai báo các biến, mảng, cấu trúc,...

§7. KHỐI LỆNH

7.1. Định nghĩa. Khối lệnh là một dãy các câu lệnh được bao bởi các dấu { và }. Ví dụ:

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

7.2. Một chú ý quan trọng khi viết chương trình là:

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

7.3. Khai báo ở đầu khối lệnh.

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

```
{
    int a,b,c[50];
    float x,y,z,t[20][30];
    a=b=3; /* gán 3 cho a và b*/
    x = 5.7; y=a*x;
    z=b*x;
    printf ("\ny=%8.2f\nz=%8.2f",y,z);
}
```

7.4. Sự lồng nhau của các khối lệnh.

Bên trong một khối lệnh lại có thể viết các khối lệnh khác . Sự lồng nhau theo cách như vậy là không hạn chế. Ta thấy một điều thú vị là: Thân hàm cũng là một khối lệnh. Đó là khối lệnh cực đại theo nghĩa nó chứa nhiều khối lệnh khác bên trong và không khối lệnh nào chứa nó.

7.5. Phạm vi hoạt động của các biến và mảng.

Một điểm cần nhớ kỹ là: nếu ta quan niệm các biến và các mảng khai báo trong một khối lệnh sẽ tồn tại suốt thời gian làm việc của chương trình và được sử dụng trong toàn bộ chương trình, thì cách hiểu như vậy là không đúng. Thực chất của vấn đề như sau:

Khi máy bắt đầu làm việc với một khối lệnh thì các biến và các mảng khai báo bên trong nó mới được hình thành và được cấp phát bộ nhớ . Các biến này chỉ tồn tại trong thời gian máy làm việc bên trong khối lệnh và chúng sẽ lập tức biến mất ngay sau khi máy ra khỏi khối lệnh. Từ nguyên lý này có thể rút ra những kết luận quan trọng sau:

- Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra để sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó.

- Ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh.

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

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

7.6. Về các khối lệnh lồng nhau.

Giả sử cùng một tên biến hay tên mảng lại được khai báo ở cả hai khối lệnh lồng nhau theo cách

```
{
    int a;
    .
    .
    {
        int a;
        .
        .
    }
    .
    .
}
```

Khi đó làm thế nào để phân biệt được biến a trong và biến a ngoài. Về thực chất ở đây ta có hai đại lượng khác nhau nhưng có chung một tên là a. Máy sẽ cấp phát hai khoảng nhớ khác nhau cho hai biến này, phạm vi hoạt động và thời gian tồn tại của chúng cũng khác nhau. Biến a trong có phạm vi hoạt động tại các câu lệnh của khối lệnh trong và nó chỉ tồn tại trong thời gian máy làm việc với khối lệnh này. Còn phạm vi hoạt động của biến a ngoài bao gồm các câu lệnh bên trong khối lệnh ngoài nhưng không thuộc khối lệnh trong. Ta cũng lưu ý là: sự thay đổi giá trị của biến a ngoài không có ảnh hưởng gì tới biến a trong và ngược lại. Ví dụ xét đoạn chương trình:

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

trong chương trình trên có sử dụng hai câu lệnh printf hoàn toàn giống nhau về mặt hình thức, nhưng thực chất chúng sẽ cho các kết qủa khác nhau. Biến a ở câu lệnh printf thứ nhất có giá trị bằng 4 (biến a trong), còn biến a ở câu lệnh printf thứ hai có giá trị là 5 (biến a ngoài). Biến b có cùng một ý nghĩa như nhau trong cả hai câu lệnh, nó có giá trị bằng 6. Về mặt hình thức thì chương trình trên có hai biến là a và b. Nhưng thực chất tồn tại ba biến khác nhau: a ngoài, a trong và b.

Như vậy đoạn chương trình trên sẽ cho hiện trên màn hình hai dòng như sau:

```
a = 4    b = 6
a = 5    b = 6
```

Chương trình dưới đây minh họa các qui tắc vừa nêu trên.

```
/*chuong trinh minh hoa pham vi hoat dong
cua cac bien trong va ngoai khoi lenh */
    #include "stdio.h"
    main()
      {
        int a,b=20;
        int c,d=40;
        float e=-35.11;
        float x=23,y,z,t=36.1;
        a=c=10;
        y=z=a+b+c+d;
          {
            float y,z;
            y=z=e+x+t;
            fprintf(stdprn,"\n y trong=%8.2f\n z trong=%8.2f",y,z);
          }
        fprintf(stdprn,"\n\n y ngoai =%8.2f\n z ngoai=%8.2f",y,z);
      }
```

Kết quả thực hiện chương trình

```
y trong = 24.00
z trong = 24.00
y ngoai = 80.00
z ngoai = 80.00
```

§8. VÀI NÉT VỀ HÀM VÀ CHƯƠNG TRÌNH

Cấu trúc chương trình và hàm là một trong các vấn đề quan trọng của C, sẽ được bàn đến một cách tỉ mỉ trong chương 6. Ở đây chúng ta chỉ đưa ra một số qui tắc chung.

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

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

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

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

+ Chương trình C được tổ chức theo mẫu:

...

hàm 1

...

hàm 2

...

...

hàm m

Bên ngoài các hàm (ở vị trí ...) có thể đặt: Toán tử #include, toán tử #define, định nghĩa kiểu dữ liệu bằng typedef, khai báo biến ngoài, mảng ngoài, biến tĩnh ngoài và mảng tĩnh ngoài.

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

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

§9. BIẾN, MẢNG TỰ ĐỘNG

9.1. Định nghĩa. Các biến (mảng) khai báo bên trong thân của một hàm (kể cả hàm main) gọi là biến (mảng) tự động hay cục bộ.

Các đối của hàm cũng được xem là biến tự động.

Do thân hàm cũng là một khối lệnh, nên từ những qui định chung về khối lệnh đã nói trong §7, có thể rút ra những điều sau:

+ **Phạm vi hoạt động:** Các biến (mảng) tự động chỉ có tác dụng bên trong thân của hàm mà tại đó chúng được khai báo.

+ **Thời gian tồn tại:** Các biến (mảng) tự động của một hàm sẽ tồn tại (được cấp phát bộ nhớ) trong khoảng thời gian từ khi máy bắt đầu làm việc với hàm đến khi máy ra khỏi hàm.

+ **Nhận xét:**

Do chương trình bắt đầu làm việc từ câu lệnh đầu tiên của hàm main() và khi máy ra khỏi hàm main() thì chương trình kết thúc, nên các biến, mảng khai báo trong hàm main() sẽ tồn tại trong suốt thời gian làm việc của chương trình.

9.2. Khởi đầu:

Chỉ có thể áp dụng cơ chế khởi đầu (khởi đầu trong khai báo) cho biến tự động (xem ví dụ trong §4)

Muốn khởi đầu cho một mảng tự động ta phải sử dụng toán tử gán.

Ví dụ:

Đoạn chương trình

```
main()
{
    float a[4] = { 3.2, 1.5, 3.6, 2.8 };
}
```

sai ở chỗ đã sử dụng cơ chế khởi đầu cho mảng tự động a.

- Biến, mảng tự động chưa được khởi đầu thì giá trị của chúng là hoàn toàn là không xác định.

Ví dụ:

Đoạn chương trình

```
#include "stdio.h"
main()
{
    float a;
    int b[3];
    printf ("\n%8.2f %6d",a,b[1] );
}
```

là vô nghĩa và không thể làm việc được, vì nó thực hiện ý định đưa ra màn hình giá trị của biến a và phần tử mảng b[1] trong khi cả hai đều chưa được khởi đầu.

§10. BIẾN, MẢNG NGOÀI

+ **Định nghĩa.** Biến (mảng) khai báo bên ngoài các hàm gọi là biến (mảng) ngoài.

+ **Thời gian tồn tại.** Biến (mảng) ngoài sẽ tồn tại (được cấp phát bộ nhớ) trong suốt thời gian làm việc của chương trình.

+ **Phạm vi sử dụng.** Phạm vi hoạt động của biến (mảng) ngoài là từ vị trí khai báo của chúng cho đến cuối tệp chương trình. Như vậy, nếu một biến (mảng) ngoài được khai báo ở đầu chương trình (đứng trước tất cả các hàm) thì nó có thể sử dụng trong bất kỳ hàm nào miễn là hàm đó không có các biến tự động trùng tên với biến (mảng) ngoài này.

Chú ý: Nếu chương trình viết trên nhiều tệp và các tệp được dịch độc lập, thì phạm vi sử dụng của biến, mảng ngoài có thể mở rộng từ tệp này sang tệp khác bằng từ khoá extern (xem Chương 11).

+ **Các qui tắc về khởi đầu**

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

Ví dụ:

```
char sao = '*';
int a = 6*365;
long b 34*3*2467;
float x = 32.5;
float y [6] = { 3.2,0,5.1,23,0,41 };
int z[3][2] = {
               { 25,31 } ,
               { 46,54 },
               { 93,81 }
             };
main()
  {
    .
    .
    .
  }
```

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

Ví dụ:

```
float a[] = { 2.6,3,15 };
int t[][4] = {
                { 6,7,8,9 },
                { 3,12,4,14,}
             };
```

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

Ví dụ:

```
float beta[8] = { 6.3,12.5 };
int h[6][3] = {
                { 4,3,2 }, { 6,1,9 },
                { 0.5.2 }
              };
```

4/ Đối với mảng 2 chiều có thể khởi đầu theo các cách sau (Số giá trị khởi đầu trên mỗi hàng có thể khác nhau):

```
float a[][3]= {
                { 0 },
                { 2.5, 3.6, 8 },
                { -6.3 }
              };
int x[10][4]= {
                { 0 },
                { 1,2,3,4 },
                { 9,10 },
                {5,6,7,8}
              };
```

5/ Bộ khởi đầu của một mảng char có thể:

- hoặc là danh sách các hằng kí tự,
- hoặc là một hằng xâu kí tự.

Ví dụ:

```
char name[] = {'H','a', 'n','g','\0' };
char name[] = "Hang";
char name[10] = { 'H','a','n','g','\0' };
char name[10] = "Hang";
```

Chương trình sau đây minh họa các qui tắc khởi đầu nêu trên

```
#include "stdio.h"
int a = 41 , t[][3] =  {
                    { 25,30,40 },
                    { 145,83,10 }
                  };
float y[8] = { -45.8, 32.5 };
float x[10][2] = {
                { -125.3, 48.9 },
                { 145.6,83.5 }
              };
char n1[] = {'T','h','u','\0' };
char n2[] = "Thu";
char n3[10] = {'T','h','u','\0' };
char n4[10] = "Thu";
main ()
  {
    printf(stdprn,"\na = %6d, t(1,2) = %6d, t(1,1) = %6d",
          a,t[1][2], t[1][1] );
    printf(stdprn,"\n\x(1,1) = %8.2f, x(2,0) %8.2f",
          x[1][1], x[2][0] );
    printf(stdprn,"\n\n%8s%8s%8s%8s",n1,n2,n3,n4 );
  }
```

Kết qủa thực hiện chương trình

```
a = 41, t(1,2) = 10, t(1,1) = 83
x(1,1) = 83.50, x(0,2) = 0.00
Thu   Thu    Thu    Thu
```

§11. TOÁN TỬ sizeof

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

sizeof(kiểu dữ liệu)

sizeof(Đối tượng dữ liệu)

Kiểu dữ liệu có thể là kiểu chuẩn như int, float và kiểu được định nghĩa trong chương trình (bằng typedef, enum, struct, union).

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

Toán tử này thường dùng để xác định số phần tử của một mảng trong cơ chế khởi đầu, ví dụ:

```
double x[]={ 23.5, 78, 49.3 };
int n= sizeof(x)/sizeof(double);
```

Số phần tử của mảng x được lưu trong biến n.

§12. BIẾN TĨNH, MẢNG TĨNH

Để khai báo một biến (mảng) tĩnh ta viết thêm từ khóa static vào đằng trước, ví dụ:

```
static int a,b,c[10];
static float x,y[10][6];
```

Các khai báo dạng trên có thể đặt bên trong hoặc bên ngoài các hàm. Nếu đặt bên trong ta có các biến (mảng) tĩnh trong, đặt bên ngoài ta có các biến (mảng) tĩnh ngoài.

Các biến (mảng) tĩnh (trong và ngoài) giống biến (mảng) ngoài ở chỗ:

+ Chúng được cấp phát bộ nhớ trong suốt thời gian hoạt động của chương trình, do đó, giá trị của chúng được lưu trữ từ đầu đến cuối chương trình.

+ Chúng có thể được khởi đầu một lần khi dịch chương trình nhờ các biểu thức hằng. Các qui tắc khởi đầu đối với biến (mảng) ngoài áp dụng được đối với biến (mảng) tĩnh . Sự khác nhau giữa biến (mảng) ngoài với biến (mảng) tĩnh chỉ ở phạm vi hoạt động (sử dụng):

+ Các biến (mảng) tĩnh trong chỉ được sử dụng bên trong thân của hàm mà tại đó chúng được khai báo.

+ Phạm vi sử dụng của các biến (mảng) tĩnh ngoài được tính từ khi chúng khai báo đến cuối tệp gốc chứa chúng.

Trong trường hợp chương trình đặt trên một tệp, hoặc chương trình đặt trên nhiều tệp nhưng dùng toán tử #include để kết nối các tệp với nhau thì các biến (mảng) tĩnh ngoài có cùng phạm vi sử dụng như các biến (mảng) ngoài.

Hai chương trình dưới đây minh họa các điều đã nói về biến (mảng) tĩnh, chúng hoàn toàn tương tự như chương trình nêu trong §10. So sánh kết qủa của chương trình, chúng ta thấy được biến (mảng) ngoài và biến (mảng) tĩnh được khởi đầu theo những qui tắc giống nhau.

Chú ý: Biến, mảng tĩnh (trong và ngoài) sẽ nói kỹ hơn trong Chương 11.

/* chuong trinh minh hoa cach khoi dau bien, mang tinh ngoai */

```
#include "stdio.h"
static int a = 41, t[][3] = {
                        {25,30,40 },
                        {145,83,10 }
                      };
static float y[8] = {-45.8,32.5};
static float x[10][2] = {
                        {-125.3,48.9 },
                        { 145.6,83.5}
                      };
static char n1[] = { 'T','h','u','\0' };
static char n2 = "Thu";
static char n3[10] = 'T','h','u','\0' };
static char n4[10] = "Thu";
main ()
  {
     fprintf(stdprn,"\na = %6d, t(1,2) =%6d, t(1,1) =%6d",
            a,t[1][2],t[1][1] );
     fprintf(stdprn,"\n\nx(1,1) =%8.2f, x(2,0) =%8.2f",
            x[1][1],x[2][0] );
     fprintf(stdprn,"\n\n%8s%8s%8s%8s",n1,n2,n3,n4 );
  }
```

Kết quả thực hiện chương trình

```
a = 41, t(1,2) = 10, t(1,1) = 83
x(1,1) = 83.50, x(2,0) = 0.00
Thu    Thu    Thu    Thu
```

/* Chuong trinh minh hoa cach khoi dau cac mang tinh trong */

```
#include"stdio.h"
main()
  {
  static int a = 41, t[][3] = {
                        { 25,30,40 }, { 145,83,10 }
                      };
```

```
static float y[8] = {-45.8,32.5} };
    static float x[10][2] = {
                                {-125.3,48.9 },
                                { 145.6,83.5}
                            };
    static char n1[] = { 'T','h','u','\0' };
        static char n2[] = "Thu";
        static char n3[10] = {'T','h','u','\0' };
        static char n4[10] = "Thu";
        fprintf(stdprn,"\na =%6d, t(1,2) =%6d , t(1,1) =%6d",
                a,t[1][2], t[1][1]);
        fprintf(stdprn,"\n\ny(0) =%8.2f, y(1) =%8.2f, y(2) =%8.2f",
                y[0],y[1],y[2] );
        fprintf(stdprn,"\n\nx(1,1) =%8.2f, x(2,0) =%8.2f",
                x[1][1],x[2][0] );
        fprintf(stdprn,"\n\n%8s%8s%8s%8s",n1,n2,n3,n4 );
    }
```

Kết qủa thực hiện chương trình

```
a = 41, t(1,2) = 10, t(1,1) = 83
y(0) = -45.8, y(1) = 32.50, y(2) = 0.00
x(1,1) = 83.50, x(2,0) = 0.00
Thu     Thu     Thu     Thu
```

BÀI TẬP CHƯƠNG 2

Bài 1

- Tìm các chỗ sai trong chương trình sau

```
#include "stdio.h"
main()
  {
     fprintf(stdprn,"\na=%10.0f, b=%10d, c=%10ld, d=%10d",
     -3456,25e3,4635,456398461);
  }
```

- Sửa lại cho đúng rồi thực hiện trên máy.

Bài 2.

Viết chương trình in một dòng với nội dung sau:

N = 365

bằng cách sử dụng các loại hằng khác nhau (hằng số dấu phẩy động, hằng int, hằng long, hằng int hệ 8, hằng int hệ 16, hằng kí tự, hằng xâu kí tự).

Bài 3

Tìm các chỗ sai trong đoạn chương trình

```
float a[3], b = 2;
a[0] = 5;
a[10] = 4, a[b] = 7;
```

Bài 4

- Tìm các chỗ sai trong đoạn chương trình

```
{
    int a = 6;
    float b=5.3;
    {
        float x = a*b, y=a+b;
    }
    printf("\na=%10.2f, b=%10.2f, x=%10.2f, y=%10.2f ", a,b,x,y);
}
```

- Sửa chữa bổ sung để được một chương trình hoàn chỉnh sau đó thực hiện trên máy.

CHƯƠNG 3

BIỂU THỨC

Toán hạng có thể xem là một đại lượng có một giá trị nào đó. Theo nghĩa đó , toán hạng bao gồm hằng, biến, phần tử mảng và hàm. Biểu thức lập nên từ các toán hạng và các phép tính để tạo nên những giá trị mới. Biểu thức dùng để diễn đạt một công thức, một quy trình tính toán, vì vậy nó là thành phần không thể thiếu được trong chương trình.

Dưới đây sẽ giới thiệu các phép toán và cách viết các biểu thức. Ta sẽ thấy trong C có nhiều quan niệm rất mới về biểu thức.

§1. BIỂU THỨC

Biểu thức là một sự kết hợp giữa các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó. Khi viết biểu thức có thể và nên dùng các dấu ngoặc tròn để thể hiện đúng trình tự tính toán trong biểu thức. Mỗi biểu thức sẽ có một giá trị và nói chung cái gì có giá trị đều được xem là biểu thức. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức. Trong C đưa ra nhiều quan niệm mới về biểu thức như biểu thức gán, biểu thức điều kiện.

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

Biểu thức thường được dùng trong:

- vế phải của câu lệnh gán,
- làm tham số thực sự của hàm (như hàm printf),
- làm chỉ số,
- trong các toán tử if, switch, for, while, do while.

Sau đây là một ví dụ áp dụng biểu thức để tính diện tích tam giác:

```
p = (a + b + c)/2; /* Nửa chu vi */
s = sqrt(p*(p-a)*(p-b)*(p-c));
```

Chúng ta đã thấy có hai khái niệm chính tạo lên biểu thức là toán hạng và phép toán. Toán hạng gồm: hằng, biến, phần tử mảng và hàm. Hằng, biến và mảng đã xét ở các phần chương trước. Dưới đây sẽ nói đến phép toán và hàm.

§2. PHÉP TOÁN SỐ HỌC

Các phép toán hai ngôi số học là:

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

Có phép toán một ngôi - ví dụ -(a+b) nhưng không có phép +. Phép chia hai số nguyên sẽ chặt cụt phần phân, ví dụ:

11/3 = 3

Phép toán % cho phần dư của phép chia nguyên, nó không áp dụng được cho các giá trị kiểu float và double. Ví dụ:

11%3 = 2

Các phép toán + và - có cùng số ưu tiên, và nhỏ hơn số ưu tiên của * / và %. Ba phép toán này lại có số thứ tự ưu tiên nhỏ hơn phép trừ một ngôi . Các phép toán số học được thực hiện từ trái sang phải . Số ưu tiên và khả năng kết hợp của phép toán được chỉ ra trong §10.

Các phép toán số học dùng để viết các công thức toán học, ví dụ:

(a*a-b*b)/(x*x+y*y)

§3. CÁC PHÉP THAO TÁC BIT

Đây là các tác vụ thường gặp trong assembly (ít gặp trong ngôn ngữ bậc cao), nó cho phép xử lý đến từng bit của một số nguyên. Các tác vụ này (không dùng cho kiểu float và double) gồm:

Phép toán	Ý nghĩa	Ví dụ
&	Phép Và (AND) theo bit	a & b
|	Phép Hoặc (OR) theo bit	a| b
^	Phép Hoặc loại trừ (XOR) theo bit	a ^ b
<<	Dịch trái	a<< 4
>>	Dịch phải	a>> 4
~	Lấy phần bù theo bit	~a

Giải thích thêm:

1 & 1 = 1
1 & 0 = 0
0 & 1 = 0
0 & 0 = 0

1 | 1 = 1
1 | 0 = 1
0 | 1 = 1
0 | 0 = 0

1 ^ 1 = 0
1 ^ 0 = 1
0 ^ 1 = 1
0 ^ 0 = 0

a << n = a*2^N
a >> n = a/2^N

~1 = 0
~0 = 1

Chú ý về phép chuyển dịch: Cũng như assembly, C phân biệt:

+ Các phép dịch chuyển số học: Thực hiện trên giá trị int, bảo toàn bit dấu (bit cực trái).

+ Các phép dịch chuyển logic: Thực hiện trên các giá trị unsigned, bit dấu không đóng vai trò gì cả, cũng bị dịch chuyển như các bit khác.

Các ví dụ minh hoạ:

0xa1b6 & 0xff = 0xb6
0xa1b6 | 0xff = 0xa1ff
0xa1b6 ^ 0xffff = 0x5e49
0xa1b6 << 8 = 0xb600
0xa1b6 >> 8 = 0xa1
(-256) << 2 = -1024
(-256) >> 2 = -64
~0xa1b6 = 0x5e49

Các ví dụ áp dụng:

Ví dụ 1: Để xét xem bit thứ n (n>=0) của biến nguyên a bằng hay khác không ta có thể làm như sau:

```
b=a >> n;
if(0x01 & b)
printf("\n Khác không");
else
printf("\n Bằng không");
```

Ví dụ 2: Để trích byte thấp và byte cao của biến nguyên a có thể dùng đoạn chương trình sau:

```
b_thap = a & 0xff
b_cao = a >> 8
printf("\nbyte thấp= %x",b_thap); /* in dạng hệ 16 */
printf("\nbyte cao = %x",b_cao);
```

Ví dụ 3: Để xoá biến nguyên a có thể dùng lệnh

```
a = a^a;
```

§4. PHÉP TOÁN SO SÁNH VÀ LOGIC

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

4.1. Các phép toán so sánh cho trong bảng sau:

Phép toán	Ý nghĩa	Ví dụ
>	Có lớn hơn không?	a > b 3 > 7 có giá trị 0
>=	Có lớn hơn hay bằng không?	a >= b 8 >= 8 có giá trị 1
<	Có nhỏ hơn không?	a < b 9 < 9 có giá trị 0
<=	Có nhỏ hơn hay bằng không?	a <= b 3 <= 10 có giá trị 1
==	Có bằng nhau không?	a == b 3 == 9 có giá trị 0
!=	Có khác nhau không?	a != b 3 != 9 có giá trị 1

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

Các phép so sánh có số ưu tiên thấp hơn so với các phép toán số học, cho nên biểu thức:

i < n-1

được hiểu là:

i < (n-1)

4.2. Phép toán logic.

Trong C sử dụng ba phép toán logic:

- phép phủ định một ngôi !
- phép Và (AND) &&
- phép Hoặc (OR) ||

Ý nghĩa của chúng được cho trong các bảng:

a	!a
Khác không	0
Bằng không	1

a	b	a && b	a || b
Khác không	Khác không	1	1
Khác không	Bằng không	0	1
Bằng không	Khác không	0	1
Bằng không	Bằng không	0	0

Chú ý: a và b có thể nguyên hay thực.

Ví dụ:

3>7 có giá trị 0

7>3 có giá trị 1

3 && 7 có giá trị 1

!15.6 có giá trị 0

Các phép so sánh có số ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và || , cho nên các biểu thức như:

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

có thể viết một cách gọn hơn bằng cách bỏ đi dấu ngoặc

a<b && c>d

Dưới đây là bảng liệt kê các phép so sánh và logic theo thứ tự ưu tiên của chúng:

```
!
>    >=    <    <=
==   !=
&&   ||
```

Các phép toán so sánh và logic được dùng để thiết lập điều kiện rẽ nhánh trong toán tử if và điều kiện kết thúc chu trình trong các toán tử for, while và do while. Ở chương 5 sẽ có rất nhiều ví dụ về việc sử dụng các phép toán này.

§5. 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 gán một giá trị kiểu này cho một biến (hoặc phần tử mảng) kiểu kia. Điều này xẩy ra trong toán tử gán, trong việc truyền giá trị các tham số thực sự cho các đối (*Chương 6*), trong câu lệnh return (*Chương 6*).

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

(Type) (biểu thức)

Ví dụ:

(float)(a+b)

5.1. Chuyển đổi kiểu trong biểu thức.

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

Chẳng hạn:

- Giữa int và long thì int chuyển thành long.
- Giữa int và float thì int chuyển thành float.
- Giữa float và double thì float chuyển thành double.

Ví dụ:

1.5*(11/3)=4.5

1.5*11/3=5.5

(11/3)*1.5=4.5

5.2. Các phép chuyển đổi kiểu

Các phép chuyển đổi kiểu cũng được thực hiện thông qua phép gán. Giá trị của vế phải được chuyển sang kiểu của vế trái đó là kiểu của kết qủa . Kiểu int có thể được chuyển thành float. Kiểu float có thể chuyển thành int do chặt cụt phần thập phân. Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int bằng cách cắt bỏ một vài chữ số.

Ví dụ: nếu n là biến nguyên, thì sau khi thực hiện câu lệnh n = 15.6 nó sẽ có giá trị là 15

5.3. Ép kiểu.

Khi viết

(type)(biểu thức)

sẽ diễn ra sự biến đổi kiểu:

Kiểu của biểu thức được đổi thành kiểu Type theo các nguyên tắc nêu trên.

Ví dụ:

Phép toán

(int)a

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 rằng bản thân kiểu của a vẫn không bị thay đổi. Nói cách khác, a vẫn có kiểu float nhưng (int)a có kiểu int.

Đối với hàm toán học của thư viện chuẩn (xem phụ lục 2) 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. Cụ thể ta nên viết như sau:

sqrt((double)n)

Phép ép kiểu có cùng số ưu tiên như các toán tử một ngôi (xem bảng tóm tắt cuối chương).

Chú ý:

+ Muốn có giá trị chính xác trong phép chia 2 biến nguyên cần dùng phép ép kiểu:

((float)a)/b

+ Để đổi giá trị thực r sang nguyên, nên dùng:

(int)(r + 0.5)

+ Chú ý đến thứ tự ưu tiên:

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

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

§6. PHÉP TOÁN TĂNG GIẢM

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm -- sẽ trừ đi 1. Chẳng hạn nếu n đang có giá trị bằng 5 thì:

Sau phép tính ++n, n có giá trị 6

Sau phép tính --n, n có giá trị 4

Dấu phép toán ++ và -- có thể đứng trước hoặc sau toán hạng, như vậy có thể viết:

++n n++ --n n--

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

Ví dụ:

Nếu n bằng 5 thì câu lệnh

```
x = n++;
```

sẽ gán 5 cho x, còn câu lệnh

```
x = ++n;
```

sẽ gán 6 cho x. Trong cả hai trường hợp n đều trở thành 6.

Việc chọn phương án này hay phương án khác là tùy thuộc ngữ cảnh . Phép toán tăng giảm thường được sử dụng trong các toán tử for, while,... để tăng hay giảm giá trị cho các biến điều khiển. Rất nhiều ví dụ như vậy có thể tìm thấy trong các chương sau.

Cuối cùng, một điều cần lưu ý là không nên sử dụng toán tử tăng giảm quá tùy tiện trong các biểu thức, vì việc đó có thể dẫn đến các kết quả sai.

§7. CÂU LỆNH GÁN VÀ BIỂU THỨC GÁN

7.1. Câu lệnh gán. Các câu lệnh gán như:

```
i = i+2;
```

trong đó vế trái được lặp lại có thể viết gọn hơn như sau:

```
i += 2;
```

Toán tử gán dạng:

```
v += e;
```

(trong đó v là một biến hoặc phần tử mảng, e là biểu thức) có thể áp dụng cho các phép toán hai ngôi (như + - * / %).

Ví dụ câu lệnh:

x = x*(y+3);

có thể viết

x *= y+3;

7.2. 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 nó là kiểu của v. Nếu đặt dấu; vào sau biểu thức gán thì ta được toán tử gán:

v = e;

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác. Ví dụ, 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 8 cho a,b,c và d.

Xét thêm một ví dụ nữa. Sau khi thực hiện câu lệnh:

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

thì y có giá trị 2, x có giá trị 6 và z có giá trị 12.

§8. BIỂU THỨC ĐIỀU KIỆN

Biểu thức điều kiện là biểu thức có dạng:

e1 ? e2:e3

trong đó e1,e2,e3 là các biểu thức nào đó . Giá trị của biểu thức điều kiện bằng giá trị của e2 nếu e1 khác không (e1 đúng) và bằng giá trị của e3 nếu e1 bằng không (e1 sai). Kiểu của biểu thức điều kiện là kiểu cao nhất trong các kiểu của e2 và e3. Chẳng hạn , nếu kiểu của e2 là int, kiểu của e3 là float thì kiểu của biểu thức điều kiện là float.

Cần phải lưu ý rằng biểu thức điều kiện thực sự là một biểu thức và ta có thể dùng nó như bất kì biểu thức nào khác.

Ví dụ: Biểu thức

(a>b) ? a:b

cho giá trị cực đại của a và b. Các dấu ngoặc trong ví dụ trên có thể bỏ vì mức ưu tiên của ? và: là rất thấp chỉ trên phép gán.

Ta tiếp tục xét thêm vài ví dụ.

Câu lệnh:

```
s=a>b ? a:b;
```

sẽ gán giá trị cực đại của a và b cho biến s.

Câu lệnh:

```
printf("\n %8.2f",a<b ? a:b);
```

cho hiện lên màn hình giá trị cực tiểu của hai biến thực a và b.

§9. VÀI VÍ DỤ

Hai chương trình dưới đây nhằm minh họa:

- Qui tắc chuyển đổi kiểu giá trị và trình tự thực hiện các phép toán trong biểu thức.
- Kết qủa thực hiện của phép so sánh và logic
- Cách dùng các phép toán tăng giảm

Ở chương trình 1 ta đã không thận trọng trong khi dùng các phép toán tăng giảm nên dẫn đến kết quả sai. Chương trình 2 cho kết quả đúng.

Cũng cần chú ý rằng C cho phép viết

```
x-- - --x
(x--) - (--x)
x++ + ++x
(x++) + (++x)
```

nhưng không cho phép viết

```
x-----x x+++++x
```

Chương trình 1.

```
/*
Chuong trinh minh hoa:
   + viec chuyen kieu trong bieu thuc
   + su thuc hien cua cac phep toan logic
   + cach dung cac phep tang giam
*/
#include "stdio.h"
main()
```

```
{
 fprintf(stdprn,"\n1.5*(11/3)= %6.2f\n1.5*11/3= %6.2f",
         1.5*(11/3),1.5*11/3);
 fprintf(stdprn,"\n(11/3)*1.5= %6.2f\n11/3*1.5= %6.2f",
         (11/3)*1.5,11/3*1.5);
 fprintf(stdprn,"\n11/3+1.5= %6.2f",11/3+1.5);
 fprintf(stdprn,"\n\nGia tri cua quan he 3>7 la: %3d",3>7);
 fprintf(stdprn,"\n\nGia tri cua quan he 7>3 la: %3d",7>3);
 fprintf(stdprn<"\nGia tri cua phep toan 3&&7 la: %3d",3&&7);
 {
  int x = 10
  int y =11;
     /*Khong than trong khi dung phep tang giam, ket qua sai*/
  fprintf(stdprn,"\n\nKhi x=10 va y=11 thi cau lenh ");
  fprintf(stdprn,"\n fprintf(stdprn, \"\\n\\n%6d\\n%6d\\n%6d\",");
  fprintf(stdprn,"\n x--*++y, x-- - --y, x++ + ++y);");
  fprintf(stdprn,"\n\nse in ra: ( gia tri sai )");
  fprintf(stdprn,"\n\n%6d\n%6d\n%6d",x--*++y,x-- - --y,x++ + ++y);
 }
}
```

Kết quả thực hiện chương trình

```
1.5*(11/3) = 4.50
1.5*11/3 = 5.50
(11/3)*1.5 = 4.50
11/3*1.5 = 4.50 11/3+1.5 = 4.50
Gia tri cua quan he 3>7 la: 0 Gia tri cua quan he 7>3 la: 1
Gia tri cua phep toan 3&&7 la: 1
Khi x = 10 va y = 11 thi cau lenh
fprintf(stdprn,"\n\n%6d\n%6d\n%6d", se in ra: (gia tri sai)
   120
   0
   22
```

Chương trình 2.

```
/*
 Chuong trinh minh hoa cach dung phep tang giam mot cach than trong.
 Ket qua dung
*/
  #include "stdio.h"
      main()
        {
          int x = 10, y = 11;
          fprintf(stdprn,"\nKhi x = 10 va y =11 thi gia tri cua ");
          fprìnt(stdprn, "\n\n x-- * ++y\n x-- - --y\n x ++ + ++y");
          fprintf(stdprn,"\n\nse la: \n");
          fprintf(stdprn, "\n%d",x-- * ++y);
          fprintf(stdprn, "\n%d",x-- - --y);
          fprintf(stdprn, "\n%d",x++ + ++y);
        }
```

Kết quả thực hiện chương trình

```
Khi x = 10 và y =11 thi gia tri cua
x-- * ++y x-- - --y
x++ + ++y se la:
120
-2 20
```

§10. THỨ TỰ ƯU TIÊN CỦA CÁC PHÉP TOÁN

Các phép toán có độ ưu tiên khác nhau, điều này có nghĩa là trong cùng một biểu thức một số phép toán này được thực hiện trước một số phép toán khác. Thứ tự ưu tiên của các phép toán được trình bầy trong bảng sau.

Stt	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

Giải thích:

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

2) Đối với các phép toán cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại. Điều này chỉ ra trong cột "Trình tự kết hợp".

Ví dụ:

* --px = *(--px) (phải qua trái)

a>b?a:x?y:z = a>b?a:(x?y:z) (phải qua trái)

8/4*6 = (8/4)*6 (trái qua phải)

3) Để viết biểu thức một cách chính xác nên sử dụng các dấu ngoặc tròn.

4) Dưới đây là vài chỉ dẫn về các phép toán lạ:

Dòng 1:

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

. Dùng để biểu diễn thành phần của cấu trúc, ví dụ: ts.ht_ten

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

Dòng 2:

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

& Phép toán lấy địa chỉ, ví dụ &x

(type) là phép ép kiểu, ví dụ (float)(x+y)

Dòng 15:

, Toán tử phẩy thường dùng để viết một dãy biểu thức trong toán tử for (Chương 5)

BÀI TẬP CHƯƠNG 3

Bài 1

Lập chương trình giải phương trình bậc hai

$ax^2 + bx + c = 0$

(giả thiết delta không âm)

Bài 2

Xác định giá trị của các biểu thức

5.6 + 2.7 + 20/6 + 8.0

5.6 + 2.7 + 20/6.0 + 8.0

Bài 3

Hãy tìm các biểu thức đúng trong các biểu thức dưới đây

(i=j)++

i+j++

++(i+j) ++i+++j

Vào máy để kiểm tra các dự đoán

Bài 4

Lập chương trình xác định giá trị cực đại và giá trị cực tiểu của bốn số thực nhập từ bàn phím

Bài 5

Lập chương trình để:

- Nhập số nguyên n từ bàn phím (0<n<10)

- Xét xem n có phải là số nguyên tố hay không và in ra kết luận tương ứng.

Bài 6

Các câu lệnh sau:

```
int x,y;
printf("\nx=%6d y=%6d", x=-35.8, y=48.6);
```

sẽ làm gì?

CHƯƠNG 4

VÀO RA

Trong các chương trước chúng ta đã nhiều lần sử dụng các hàm printf, fprintf, scanf để tổ chức việc đưa vào và đưa ra. chương này sẽ giới thiệu lại các hàm trên một cách đầy đủ hơn. Ngoài ra sẽ bổ sung nhiều hàm mới như: nhập xuất chuỗi, nhập xuất ký tự, xoá màn hình, di chuyển con trỏ màn hình, kiểm tra bộ đệm bàn phím.

Tất cả các qui tắc vào ra sẽ được giải thích tỉ mỉ và được minh hoạ trên các ví dụ cụ thể. Các hàm trong các mục từ §1 đến §6 khai báo trong tệp stdio.h. Riêng các hàm trong §7 khai báo trong conio.h.

§1. HÀM printf

Hàm printf có khả năng chuyển dạng, tạo khuôn và đưa giá trị các đối ra màn hình. Dạng tổng quát của hàm như sau:

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

Ở đây đối dk là biến con trỏ kiểu char (xem *Chương 6*) chứa địa chỉ của chuỗi điều khiển.

1.1. Chuỗi điều khiển gồm ba loại ký tự:

+ Các ký tự điều khiển như:

\n Sang dòng mới

\f Sang trang mới

\b Lùi lại một vị trí

\t Dấu tab

+ Các đặc tả chuyển dạng và tạo khuôn (gọi tắt là đặc tả) cho các đối tương ứng (xem các điểm 2 và 3 dưới đây).

+ Các ký tự không phải là đặc tả cũng không phải là 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:

Cách viết	Ý nghĩa
\'	In ra dấu '
\"	In ra dấu "
\\	In ra dấu \

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

%[-][fw][.pp]ký tự chuyển dạng

Dấu % và ký tự chuyển dạng (Xem điểm 3 dưới đây) là những thành phần bắt buộc phải có. Các thành phần khác có thể vắng mặt.

a) Dấu trừ:

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

+ Khi có mặt dấu - thì mọi điều nói trên vẫn như vậy trừ ra là kết quả ra được dồn về bên trái và các vị trí dư thừa về bên phải (nếu có) luôn được lấp đầy bằng các khoảng trống. (Độ dài thực tế của xâu kí tự là số kí tự của xâu. Cách xác định độ dài thực tế của một số kiểu int, float hay double đã trình bày ở Chương 1, §7).

Ví dụ 1.

Trường ra	fw	dấu -	kết quả đưa ra
-2503	8	có	:-2503 :
-2503	08	có	:-2503 :
-2503	8	không	: -2503 :
-2503	08	không	:000-25.3:
"abcdef"	08	có	:abcdef :
"abcdef"	08	không	: abcdef:
"abcdef"	8	không	: abcdef:

b/ fw là một dãy số nguyên xác định độ rộng tối thiểu dành cho trường ra:

+ Khi fw lớn hơn độ dài thực tế của trường ra, thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số không và nội dung của trường ra sẽ được đẩy về bên phải hoặc bên trái. Điều này đã được giải thích ở trên.

+ Khi không có mặt fw hoặc khi fw nhỏ hơn hay bằng dộ dài thực tế của trường ra thì độ rộng (trên thiết bị ra) dành cho trường sẽ bằng độ dài thực tế của trường.

Ví dụ 2.

Trường ra	Fw	Kết quả đưa ra
"alphabeta"	3	:alphabeta:
432056	vắng mặt	:432056:
-13782	4	:-13782:

c) pp là dãy số nguyên. Tham số pp chỉ được sử dụng khi đối tương ứng là một xâu kí tự hoặc một giá trị kiểu float hay double.

+ Trong trường hợp đối tương ứng có giá trị kiểu float hay double, thì pp là độ chính xác của trường ra. Nói một cách cụ thể hơn giá trị in ra sẽ có pp chữ số sau dấu chấm thập phân. Khi vắng mặt pp thì độ chính xác được xem là 6.

+ Khi đối tương ứng là một xâu ký tự thì:

- Nếu pp nhỏ hơn độ dài của xâu thì chỉ pp ký tự đầu tiên của xâu được đưa ra.

- Nếu vắng mặt tham số pp hoặc nếu pp lớn hơn hay bằng độ dài của xâu thì cả xâu ký tự sẽ được đưa ra.

Ví dụ 3.

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

1.3. 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 tình trạng cùng một giá trị sẽ được đưa ra (in ra) theo các dạng khác nhau khi sử dụng các ký tự chuyển dạng khác nhau. Tuy nhiên, một điều cần chú ý ở đây là không được sử dụng các ký tự chuyển dạng một cách tuỳ tiện mà phải tuân theo những qui tắc định sẵn: với mỗi kiểu giá trị chỉ tương ứng với một hoặc một vài ký tự chuyển dạng nhất định. Bảng sau đây cho các thông tin về các ký tự chuyển dạng.

Bảng 1

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 0x đứng trước)
lx	long	Đối được coi là số hệ 16 không dấu (không có 0x đứng trước)
f	float hay double	Đối được chuyển sang dạng thập phân [-]m...m.n...n. Độ dài của dãy n...n bằng pp
e	float hay double	Đối được chuyển sang dạng thập phân [-]m.n...nE[+ hoặc -]xx Độ dài dãy n...n bằng (pp-1)
g	float hay double	Dùng %e hoặc %f tuỳ thuộc loại nào ngắn hơn. Không in ra các số 0 vô nghĩa.
s	xâu ký tự	Các ký tự xâu được in ra theo nguyên tắc đã nói trong 1.2.c

Ý nghĩa của các ký tự chuyển dạng được minh hoạ khá rõ ràng trong hai chương trình đầu của §4 bên dưới. Ở đây ta đưa ra một ví dụ đơn giản: Câu lệnh:

printf("%d %c %d %c",'A','A',66,66); in ra: 65 A 66 B

printf("%d %u",-1,-1); in ra: -1 65535

1.4. Danh sách các đối.

Các đối cần được phân cách nhau bởi dấu phẩy. Đối có thể là một hằng, một biến, một phần tử mảng, một lời gọi hàm (ví dụ: sin(x)) hay nói chung là một biểu thức bất kỳ. Giá trị của đối sẽ được chuyển dạng và in ra theo cách của đặc tả tương ứng. Một điều cần chú ý ở đây là mỗi đặc tả cần có một đối tương ứng. Khi mà một đặc tả không tìm thấy đối tương ứng hoặc khi kiểu giá trị của đối tương ứng không tương thích với ký tự chuyển dạng (như đã chỉ ra ở bảng 1) thì máy sẽ bị lẫn lộn và có thể đưa ra những kết quả vô nghĩa.

Số đối có thể lớn hơn số đặc tả, khi đó những đối cuối cùng không có đặc tả tương ứng sẽ không được in ra.

1.5. Hai chú ý quan trọng khi viết đặc tả chuyển dạng.

Chú ý 1. Theo định nghĩa, đặc tả chuyển dạng là một dãy kí tự mở đầu bằng dấu % và kết thúc bằng một kí tự chuyển dạng, ví dụ:

%d

%5d

%10.2f

%3s

Điều cần lưu ý ở đây là: mọi dãy ký tự không bắt đầu bằng dấu % hoặc không kết thúc bằng một kí tự chuyển dạng đều được xem là kí tự hiển thị. Ví dụ câu lệnh:

```
printf("\" nang suat tang: %d % \"\n\\d = %d\"",30,-50);
```

sẽ cho hiện lên màn hình nội dung:

```
" nang suat tang: 30 %"
\d = -50"
```

Chú ý 2. Xét đặc tả:

%10.2f

trong đặc tả này:

fw=10

pp=2

ký tự chuyển dạng là f

Điều cần lưu ý là tại vị trí của fw ta có thể đặt dấu * (thay fw bằng dấu *), khi đó fw được xác định bởi giá trị (nguyên) của đối tương ứng. Khả năng này cho phép thay đổi độ rộng dành cho trường ra một cách hết sức cơ động.

Ví dụ. Đoạn chương trình:

```
int n=8;
float x =25.6, y=-47.335;
printf ("\n%f\n%*.2f",x,n,y);
```

sẽ đưa lên màn hình các giá trị của x và y trên hai dòng khác nhau. Đặc tả chuyển dạng của y là:

%*.2f

Đối tương ứng với dấu * là n có giá trị bằng 8, vậy đặc tả của y trong trường hợp này thực chất là:

%8.2f

Điều đó có nghĩa máy sẽ dành ít nhất là 8 vị trí trên màn hình để chứa giá trị của y. Vậy trên màn hình ta sẽ nhận được hai dòng với nội dung sau:

25.600000

-47.34

1.6. Giá trị của hàm printf.

Khi thành công, hàm cho biết số ký tự (kể cả ký tự điều khiển) được đưa ra. Khi có lỗi, hàm có giá trị -1.

Ví dụ sau khi thực hiện các câu lệnh

```
int a=10, b=1245;
m = printf("\nA=%4d B=%d",a,b);
```

thì m=14.

1.7. Số nguyên hệ 10 có dấu và không dấu

Nội dung của điểm này có liên quan mật thiết đến các kí tự chuyển dạng d,ld, u và lu. Như đã biết, một số nguyên hệ 10 có dấu (kiểu int) được chứa trên 16 bit. Gọi x là giá trị chứa trên 15 bit cuối, khi đó:

$$0 <= x <= 2^{15} - 1 = 32767$$

+ Các số nguyên dương được biểu diễn như sau: bít đầu bằng 0, giá trị của số được biểu trên 15 bít cuối. Vậy trong máy có thể biểu diễn các số nguyên dương có giá trị từ 0 đến 32767.

+ Cách biểu diễn số nguyên âm trong máy như sau: Bít đầu bằng 1. Nếu gọi giá trị tuyệt đối của số là y, thì trong 15 bít cuối sẽ không biểu diễn y mà lại chứa phần bù của y, tức là chứa số:

$$x = 2^{15} - y = 32768 - y$$

Do:

$$0 <= x <= 32767$$

Nên suy ra:

$$1 <= y <= 32768$$

Vậy trong máy có thể biểu diễn được các số nguyên âm trong khoảng từ -32768 đến -1.

Khác với các số nguyên hệ 10 có dấu, số nguyên hệ 10 không dấu được hiểu là một số nguyên không âm và giá trị của nó được chứa trong cả 16 bit. Do đó, trong máy có thể biểu diễn được các số nguyên không dấu trong khoảng từ 0 đến 2^{16} -1 = 65535.

Bây giờ ta lý giải hiện tượng thú vị sau đây. Giả sử z là một biến nguyên có giá trị -1. Khi đó câu lệnh:

```
printf("%d",z);
```

tất nhiên sẽ cho màn hình số -1. Thế nhưng câu lệnh

```
printf("%u",z);
```

lại cho lên màn hình số 65535.

Tại sao vậy? Theo lý giải ở trên, cả 16 bit chứa số -1 đều có giá trị bằng 1. Trong câu lệnh thứ hai, ta dùng kí hiệu chuyển dạng u, do đó z được xem là một số nguyên hệ 10 không dấu và giá trị của nó được biểu diễn trên 16 bit, vì vậy khi đó giá trị của nó là

$$2^{16} - 1 = 65535$$

Dựa vào nguyên lý biểu diễn số trong máy như đã trình bầy ở trên, ta có thể rút ra những qui tắc sau:

+ Qui tắc thứ nhất: nếu dùng kí hiệu chuyển dạng u cho một đối nguyên kiểu int có giá trị -x (1 <= x<= 32768) thì trên thiết bị ra nhận được giá trị nguyên: 65536 - x

Qui tắc thứ hai: nếu dùng kí hiệu chuyển dạng lu cho một đối nguyên kiểu long có giá trị -x (1 <= x <= 2147483648) thì trên thiết bị ra nhận được giá trị nguyên 4294967296 - x

Tất cả những điều trình bầy trong mục này được minh hoạ một cách rõ ràng qua các chương trình 1 và 2 của §4 dưới đây.

§2 HÀM scanf

Hàm scanf là hàm có nhiều chức năng tương tự như hàm printf nhưng theo chiều ngược lại. Nó 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.

Hàm có dạng:

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

ở đây dk là biến con trỏ kiểu char chứa địa chỉ của chuỗi điều khiển.

2.1. Danh sách các đối. Mỗi đối trong danh sách 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. Các đối cần được phân cách nhau bởi dấu phẩy.

2.2. Chuỗi điều khiển gồm các ký tự đặc tả chuyển dạng. Mỗi đặc tả thường có một đối tương ứng.

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

%[*][d...d] ký tự chuyển dạng

a) Việc có mặt của dấu * nói lên rằng trường vào vẫn được dò đọc bình thường, nhưng giá trị của nó bị bỏ qua (không được lưu vào bộ nhớ). Như vậy, đặc tả chứa dấu * sẽ không có đối tương ứng (xem chương trình 9 của §4 bên dưới).

b) d...d là một dãy số xác định chiều dài cực đại của trường vào, ý nghĩa của nó sẽ được giải thích kỹ trong các mục sau.

2.4. Dòng vào và trường vào.

Dòng vào là một dãy ký tự liên tiếp nhau (bao gồm cả các khoảng trắng) trên thiết bị vào (ở đây, khoảng trắng được hiểu là dấu cách, dấu tab hoặc ký hiệu xuống dòng \n). Các khoảng trắng được xét đến trong ba trường hợp (sẽ nói dưới đây) còn trong tất cả các trường hợp khác chúng bị bỏ qua. Khi đó các khoảng trắng được xem là dấu phân cách giữa các trường vào, còn mỗi trường vào được xem là một dãy ký tự khác khoảng trắng. Chẳng hạn, trên dòng vào:

13.2 abc\t 25\n

gồm ba trường

13.2

abc

25

Độ dài của trường thứ nhất là 4, của trường thứ hai là 3 và của trường thứ ba bằng 2.

Bây giờ, ta trở lại giải thích ý nghĩa của tham số d...d. Ta chia ra hai trường hợp:

- *Trường hợp 1:* Nếu tham số d...d vắng mặt hoặc nếu giá trị của nó lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ trường vào sẽ được đọc, nội dung của nó được dịch và được gán cho địa chỉ tương ứng (nếu không có dấu *).

- *Trường hợp 2:* Nếu giá trị của d...d nhỏ hơn độ dài của trường vào tương ứng thì chỉ phần đầu của trường có kích cỡ bằng d...d được đọc, được dịch và được gán cho địa chỉ tương ứng. Phần còn lại của trường sẽ được xem xét bởi các đặc tả và đối tương ứng tiếp theo.

Ví dụ: Đoạn chương trình

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

Với dòng vào: 54.32e-1 25 12452348a

Sẽ gán:

5.432 cho x

25.0 cho y

124 cho a

xâu "523" có cả dấu kết thúc \0 cho ch

xâu "48a" có cả dấu kết thúc \0 cho ct

2.5. Ký tự chuyển dạng.

Ký tự chuyển dạng xác định cách thức dò đọc các ký tự trên dòng vào cũng như phương pháp chuyển dịch thông tin đọc được trước khi gán nó theo các địa chỉ tương ứng. Có thể nói cách dò đọc thứ nhất là đọc theo trường vào, khi đó các khoảng trắng bị bỏ qua, cách này áp dụng với hầu hết các trường hợp. Cách dò đọc thứ hai là đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét bình đẳng như các ký tự khác. Phương pháp dò đọc 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 và ý nghĩa của chúng.

c Vào một kí tự, đối tương ứng là con trỏ kí tự. Có xét kí tự khoảng trắng.

d Vào một giá trị kiểu int, đối tương ứng là con trỏ kiểu int, trường vào phải là số nguyên.

ld Vào một giá trị kiểu long, đối tương ứng là con trỏ kiểu long, trường vào phải là số nguyên.

o Vào một giá trị kiểu int hệ 8, đối tương ứng là con trỏ kiểu int, trường vào phải là số nguyên hệ 8 (xem chương trình 6 mục §4 dưới đây).

lo Vào một giá trị kiểu long hệ 8, đối tương ứng là con trỏ kiểu long, trường vào phải là số nguyên hệ 8. (xem chương trình 6 mục §4 dưới đây).

x Vào một giá trị kiểu int hệ 16,đối tương ứng là con trỏ kiểu int, trường vào phải là số nguyên hệ 16 (xem chương trình 6 mục §4 dưới đây).

lx Vào một giá trị kiểu long hệ 16, đối tương ứng là con trỏ kiểu long, trường vào phải là số nguyên hệ 16 (xem chương trình 6 mục §4 dưới đây).

f hay e Vào một giá trị kiểu float, đối tương ứng là con trỏ kiểu float, trường vào phải là số dấu phẩy động.

lf hay le Vào một giá trị kiểu double, đối tương ứng là con trỏ kiểu double, trường vào phải là số dấu phẩy động.

Vào một xâu kí tự, đối tương ứng là con trỏ kiểu char trỏ tới một vùng nhớ đủ lớn để nhận được xâu và dấu kết thúc \0 sẽ được tự động thêm vào, trường vào là dẫy kí tự bất kì không chứa các dấu cách và dấu xuống dòng \n.

[dãy kí tự] Các kí tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một kí tự không thuộc tập các kí tự đặt trong hai dấu []. Đối tương ứng phải là con trỏ kiểu char trỏ tới một vùng nhớ đủ lớn, trường vào là dãy kí tự bất kì (khoảng trắng được xét như ký tự khác).

[^dãy kí tự] Các kí tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một kí tự thuộc tập các kí tự đặt trong hai dấu []. Đối tương ứng phải là con trỏ kiểu char trỏ tới một vùng nhớ đủ lớn, trường vào là dãy kí tự bất kì (khoảng trắng được xét như ký tự khác).

Ví dụ dưới đây sẽ làm rõ ý nghĩa của hai loại kí tự chuyển dạng sau cùng. Đoạn chương trình:

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

Với dòng vào:

35 13145 XYZ 584235

Sẽ gán:

35 cho a

Xâu " 13145 " cho ch

Xâu "XYZ " cho ck

584 cho b

2.6. Giá trị của hàm. hàm cho một số nguyên bằng số giá trị nhận được (lưu vào bộ nhớ).

2.7. Vài điều cần lưu ý

a) Xét đoạn chương trình dùng để nhập (từ bàn phím) ba giá trị nguyên và gán cho ba biến a,b,c. Nó có thể vết như sau:

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

Để vào số liệu (từ bàn phím) có thể thao tác theo nhiều cách khác nhau:

Cách 1: Đưa ba số lên cùng một dòng, các số được phân cách bằng các dấu cách và dấu tab.

Cách 2: Đưa ba số lên ba dòng khác nhau.

Cách 3: Hai số đầu trên một dòng, số thứ ba trên dòng tiếp theo.

Cách 4: Số thứ nhất trên một dòng, hai số sau trên dòng tiếp theo.

Điều này là hoàn toàn dễ hiểu vì giữa các trường vào có thể đặt một số tuỳ ý các dấu cách, dấu tab và dấu xuống dòng.

b/ Việc ra khỏi hàm chuẩn scanf để trở về chương trình chứa lời gọi nó sẽ xảy ra khi:

- Hoặc xét hết các đặc tả.

- Hoặc xét hết các đối.

- Hoặc gặp một sự không tương thích giữa đặc tả và đối tương ứng.

Khi trở về chương trình, hàm scanf sẽ cho một giá trị nguyên bằng các trường vào đã được lưu vào bộ nhớ.

c/ Chú ý cuối cùng là: Để việc nhập số liệu được chính xác ta nên làm theo các yêu cầu sau:

- Số đối, số đặc tả và số trường vào phải bằng nhau.

- Giữa đối, đặc tả và trường vào cần có sự phù hợp như đã chỉ ra trong điểm 5.

§3. ĐƯA RA MÁY IN

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

fprintf(stdprn, const char *dk [,danh sách các đối]);

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

+ dk là biến con trỏ (kiểu char) chứa địa chỉ của chuỗi điều khiển.

Tất cả những điều đã nói ở mục §1 về chuỗi điều khiển và danh sách các đối vẫn còn đúng trong trường hợp này. Chỉ có một điểm khác biệt duy nhất là: Ở mục §1 thiết bị ra là màn hình, còn ở đây là máy in.

Ví dụ: Sau đây là đoạn chương trình in ma trận A cỡ 8x6 (8 hàng, 6 cột) Mỗi hàng của ma trận được in trên một dòng.

```
float a[8][6];
int ij;
fprintf(stdprn,"\n%20c MA TRAN A \n\n\n",32);
```

```
for (i=0;i<8,++i)
  {
     for (j=0;j<6;++j)
       fprintf(stdprn,"%10.2f",a[i][j]);
       fprintf(stdprn,"\n");
  }
```

§4. VÍ DỤ MINH HOẠ CÁC QUI TẮC VÀO RA

Các qui tắc trình bày trong các mục trên sẽ được minh hoạ qua 9 chương trình dưới đây.

Chương trình 1

```
/* Chương trình minh hoạ: Nếu dùng không đúng kí tự
   chuyển dạng thì kết quả in ra bị sai
*/
   #include "stdio.h"
   main()
    {
      int x=10;
      long y=3478925;
      printf("\nx=%10ld\ny=%10d",x,y);
    }
```

Kết quả thực hiện chương trình

```
x=361562122
y= 53
```

Chương trình 2

```
/* Chuong trinh minh hoa: Dạng in ra phụ thuộc vào đặc tả */
   #include "stdio.h"
   main()
    {
      int x=360, y=-1;
      long yl=-1l;
      char *vb = "programmer";
      double dx = 3.14159265;
```

```
    fprintf(stdprn,"\nx=%010d\nx=%-010d\nx=%10o\nx=%010o",
              x,x,x,x);
    fprintf (stdprn,"\nx=%-10d\nx=%-010x",x,x);
    fprintf(stdprn,"\n\ny=%10d\ny=%10u\ny=%10o\ny=%10x",
              y,y,y,y);
    fprintf(stdprn,"\n\nyl=%ld\nyl=%10lu\nyl=%10lo\nyl=%-10lx"
              ,yl,yl,yl,yl);
    fprintf (stdprn,"\n\ndx=%010f\ndx=%010.3f\ndx=%-010.3f",
              dx, dx,dx);
    fprintf (stdprn,"\n\ndx=%10.8f\ndx=%10.0f\ndx=%10g",dx,dx,dx);
    fprintf (stdprn,"\ndx=%10e\ndx=%10.2e",dx,dx);
    fprintf(stdprn,"\n\n%10s\n%10.7s\n%-10.7s\n%010.4s\n%10.0s",
              vb,vb,vb,vb,vb);
    fprintf (stdprn,"\n%0.3s\n%-010.3s",vb,vb);
  }
```

Kết quả thực hiện chương trình

```
x=0000000360
x=360
x= 550
x=0000000550
x=168 x=168

y= -1
y= 65535
y= 177777
y= ffff

yl=-1
yl=4294967295
yl=37777777777
yl=ffffffff

dx=003.141593 dx=000003.142
dx=3.142
dx=3.14159265
dx= 3
```

```
dx= 3.14159
dx= 3.14159e+00

programmer
  program
program
  prog
pro
pro
```

Chương trình 3

```
/*
Chuong trinh minh họa:
ket qua cua cau lenh scanf
ung voi cac dong vao khac nhau
*/

  #include "stdio.h"
  main()
   {
      int a,b;
      float x,y;
      char ch[10];
      scanf("%3d%4d%3f%f%2s",&a,&b,&x,&y,&ch);
      fprintf(stdprn,"\na=%10d\nb=%10d\nx=%8.2f\ny=%8.2f\n%s",
              a,b,x,y,ch);
   }
```

Với dòng vào

```
12345.2646123a5746
```

chương trình cho kết quả

```
a= 123
b= 45
x= 0.26
y=46123.00
a5
```

Với dòng vào

    12.34a123b456.789a90

chương trình cho kết quả

a= 12 b= -44

x= 0.00 y= 0.00

Với dòng vào

    123456789123a456

chương trình cho kết quả

a= 123 b= 4567

x= 891.00

y= 23.00

a4

Chương trình 4.

```
/*
Chuong trinh minh hoa:
Việc dùng sai đặc tả trong câu lệnh scanf
không bị phát hiện khi dịch nhưng kết quả sai
*/
  #include "stdio.h"
  main()
  {
    int a;
    long b;
    float x;
    double y;
    printf("\n vao a b x y");
    scanf("%ld%d%lf%f",&a,&b,&x,&y);
    printf("\na=%10d\nb=%10ld\nx=%8.2f\ny=%8.2f",a,b,x,y);
  }
```

Với dòng vào

    2346 447895 347 246

chương trình cho kết quả

a = 2346

b = 54679

x = 0.00

y = 0.00

Chương trình 5

```
/* Chương trình với đặc tả đúng */
   #include "stdio.h"
   main()
    {
      int a;
      long b;
      float x;
      double y;
      printf("\n vao a b x y");
      scanf("%d%ld%f%lf",&a,&b,&x,&y);
      printf("\na=%10d\nb=%10ld\nx=%8.2f\ny=%8.2f",a,b,x,y);
    }
```

Với dòng vào

2346 447895 347 264

chương trình cho kết quả

a = 2346

b = 447895 x = 347.00

y = 246.00

Chương trình 6.

```
/*
  Chương trình minh hoạ các đặc tả ứng với các loại số
*/
   #include "stdio.h"
   main()
    {
       int a,b,c;
       long x,y,z;
       double s,t,u,v;
       printf("\n Vao a,b,c: ");
       scanf("%d%o%x",&a,&b,&c);
       printf("\n Vao x,y,z: ");
       scanf("%ld%lo%lx",&x,&y,&z);
       printf("\n Vao s,t,u,v: ");
       scanf("%lf%le%lf%le",&s,&t,&u,&v);
```

```
        printf("\na=%4d\nb=%4d\nc=%4d",a,b,c);
        printf("\nx=%10ld\ny=%10ld\nz=%10ld",x,y,z);
        printf("\ns=%10.2e\nt=%10.2e\nu=%10.2g\nv=%10.2g",s,t,u,v);
    }
```

Với các dòng vào

74 112 4a

1048575 3777777 fffff

1e50 1E50 1E50 1e50

chương trình cho kết quả

a = 74

a = 74

c = 74

x = 1048575

y = 1048575

z = 1048575

s = 1.0e+50

t = 1.0e+50

u = 1e+50

v = 1e+50

Chương trình 7

```
/* Chuong trinh minh hoạ các đặc tả kí tự */
#include "stdio.h"
main()
 {
    char s1[10],s2[10],s3[20],s4[20];
    printf("\n Vao mot day ky tu ");
    scanf("%3s%s%[^,]%[123456789,]",s1,s2,s3,s4);
    printf("\n%s\n%s\n%s\n%s",s1,s2,s3,s4);
 }
```

Với dòng vào

abcdeft Pham van Ba,25 34a

chương trinh cho kết quả

abc

deft

Pham van Ba

,25

Chương trình 8

```
/* Chương trình thử các đặc tả số và ký tự */
  #include "stdio.h"
  main()
   {
    int a;
    long b;
    char ch[20],ck[20];
    printf("Nhap a, mot day ky tu va b: ");
    scanf("%d%[0123456789 ]%[^0123456789 ]%5ld",
          &a,ch,ck,&b);
    printf("\na = %3d\nb = %10ld\nch = %s\nck = %6s",
          a,b,ch,ck);
   }
```

Với dòng vào

35 13145xyz 58425367

chương trình cho kết quả

a = 35

b = 58425

ch = 13145

ck = xyz

Chương trình 9

```
/*
 Chuong trinh minh hoa đặc tả %* trong lệnh scanf
*/
  #include "stdio.h"
  main()
   {
    int a,b;
    float x,y;
    printf("\n Doc 6 so (3 int, 3 float) ");
    scanf("%d%*d%d%f%*f%f",&a,&b,&x,&y);
    printf("\na=%d\nb=%d\nx=%10.3e\ny=%10.6e", a,b,x,y);
   }
```

Với dòng vào

135 467 325 4538.26 1 6782

chương trình cho kết quả:

a = 135 b = 325

x = 4.54e+03

y = 6.78200e+03

§5. DÒNG VÀO STDIN VÀ CÁC HÀM scanf, gets, getchar

5.1. stdin là dòng vào chuẩn (bàn phím). Các hàm scanf, gets và getchar đều nhận dữ liệu từ stdin. Cần phân biệt 2 trường hợp:

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

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

5.2. Hàm gets nhập một chuỗi ký tự từ stdin.

+ Dạng hàm:

```
char *gets(char *s);
```

+ Đối: s là con trỏ (kiểu char) trỏ tới vùng nhớ chứa dãy ký tự nhận được.

+ Công dụng: Nhận dãy ký tự từ stdin cho đến khi gặp '\n'. Ký tự '\n' bị loại khỏi stdin nhưng không được đặt vào chuỗi.

Chuỗi được bổ sung thêm ký tự kết thúc '\0' và đặt vào vùng nhớ do s trỏ tới.

Hàm trả về địa chỉ chuỗi nhận được.

5.3. Hàm getchar nhận một ký tự từ stdin.

+ Dạng hàm:

```
int getchar(void);
```

+ Công dụng: Nhận một ký tự từ stdin. Hàm trả về ký tự nhận được.

5.4. Các chú ý về stdin.

+ Chú ý 1:

Xét các câu lệnh:

```
int ch;
ch=getchar();
```

- Nếu bấm phím

A ↵ (ký hiêu ↵ là Enter)

thì ch='A', ký tự '\n' vẫn còn lại stdin và nó sẽ làm trôi hàm getchar hoặc gets sau đó.

- Nếu chỉ bấm phím Enter thì ch='\n' và '\n' bị loại khỏi stdin.

+ Chú ý 2. Hàm scanf cũng để lại stdin ký tự '\n'. 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 điêu khiển như trong ví dụ sau (hoặc dùng hàm fflush trong 5.5):

```
char ht[25];
int t;
printf("\nTuoi: ");
scanf("%d%*c",&t); /* Khử '\n' trong dòng vào */
printf("\nHo ten: ");
gets(ht);
```

5.5. Làm sạch stdin.

Có thể làm sạch (xoá) stdin bằng cách dùng hàm fflush như sau:

```
fflush(stdin);
```

Dùng hàm này sẽ tránh được mọi hậu quả của các thao tác nhập số liệu trước đó. Ví dụ để dùng gets sau scanf ta có thể làm như sau:

```
char ht[25];
int t;
printf("\nTuoi: ");
scanf("%d",&t);
printf("\nHo ten: ");
fflush(stdin);     /* xoá stdin */
gets(ht);
```

5.6. Xét thêm một vài ví dụ.

Hai ví dụ sau minh hoạ thêm mối quan hệ giữa stdin và các hàm nhập số liệu.

Ví dụ 1:

Xét đoạn chương trình:

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

```
scanf("%d",&b);
```

Nếu bấm phím:

12 345 ↵ (↵ ký hiệu phím Enter)

thì trôi qua cả hai lệnh scanf và a=12, b=345.

Ví dụ 2:

Xét đoạn chương trình

```
int a;
char ch, ht[25];
scanf("%d",&a);
ch=getchar();
gets(ht);
```

Nếu bấm phím:

12ETran Van ↵

thì trôi qua cả 3 lệnh nhập số liệu (scanf, getchar, gets) và:

a=12 ch='E' ht="Tran Van "

§6. CÁC HÀM XUẤT KÝ TỰ puts VÀ putchar

Các hàm xuất printf, puts và putchar đều có tác dụng đưa dữ liệu lên dòng ra chuẩn stdout (màn hình).

6.1. Hàm puts:

Đưa một chuỗi ký tự ra stdout.

+ Dạng hàm:

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

+ Đối: s là con trỏ (kiểu char) trỏ tới vùng nhớ chứa chuỗi ký tự cần xuất ra stdout.

+ Công dụng:

Đưa chuỗi s và đưa thêm ký tự '\n' lên stdout. Khi thành công, hàm trả về ký tự cuối cùng được xuất (chính là '\n'). Khi có lỗi hàm trả về EOF.

+ Ví dụ:

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.

6.2. Hàm putchar:

Đưa một ký tự ra stdout.

+ Dạng hàm:

int putchar(int ch);

+ Đối:

ch chứa mã của ký tự cần xuất.

+ Công dụng:

Đưa ký tự ch lên stdout. Khi thành công hàm trả về ký tự được xuất (chính là ch). Khi có lỗi hàm cho EOF.

+ Ví dụ:

Câu lệnh

putchar('A');

sẽ in chữ A lên màn hình tại vị trí hiện tại của con trỏ.

Câu lệnh

putschar(7);

sẽ gõ một tiếng chuông

§7. CÁC HÀM VÀO RA TRÊN MÀN HÌNH, BÀN PHÍM

Trong mục này xét các hàm getch, getche, putch, kbhit, clrscr và gotoxy. Khác với các hàm trước khai báo trong stdio. h, các hàm ở mục này khai báo trong conio.h.

Các hàm getch và getche cho phép nhận một ký tự trực tiếp từ bộ đệm bàn phím.

Hàm putch dùng để đưa một ký tự lên cửa sổ văn bản trên màn hình.

Hàm kbhit cho biết trong bộ đệm bàn phím có còn ký tự nào hay không.

Hàm clrscr dùng để xoá màn hình.

Hàm gotoxy cho phép di chuyển con trỏ đến vị trí bất kỳ trên màn hình.

7.1. Hàm getch:

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

+ Dạng hàm:

int getch(void);

+ Công dụng:

- 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 Enter như trong các hàm nhập từ stdin). Ký tự vừa gõ không được hiện lên màn hình.

+ Hàm trả về ký tự nhận được.

7.2. Hàm getche:

Nhận một ký tự từ bộ đệm bàn phím, cho hiện lên màn hình.

+ Dạng hàm:

```
int getche(void);
```

+ Công dụng:

Hàm này làm việc giống như hàm getch, chỉ có một điểm khác là cho hiện ký tự được gõ lên màn hình.

7.3. Hàm putch:

Đưa một ký tự ra cửa sổ văn bản màn hình.

+ Dạng hàm:

```
int putch(int ch);
```

+ Đối: ch chứa 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 mầu xác định trong hàm textcolor. Đây là sự khác nhau với hàm putchar. Hàm putchar luôn hiển thị ký tự theo mầu trắng.

+ Hàm trả về ký tự được hiển thị.

7.4. Hàm kbhit:

Kiểm tra bộ đệm bàn phím.

+ Dạng hàm:

```
int kbhit(void);
```

+ Công dụng:

Hàm có giá trị khác không nếu bộ đệm bàn phím khác rỗng, có giá trị 0 nếu trái lại.

+ Chú ý:

Nếu bấm một phím thì ký tự tương ứng có thể bị gửi vào stdin hay vào bộ đệm. Làm sao để phân biệt được?.

Cách phân biệt như sau:

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

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

Dùng các hàm kbhit và getch có thể lấy ra tất cả các ký tự đang lưu trong bộ đệm và như vậy bộ đệm bị xoá (xem ví dụ mục 7.6 dưới đây).

7.5. Xoá màn hình và di chuyển con trỏ.

+ Để xoá màn hình ta dùng hàm:

    clrscr();

+ Để di chuyển con trỏ (màn hình) đến vị trí (x,y) ta dùng hàm

    gotoxy(x,y);

Ở đây x là số hiệu cột nhận giá trị từ 1 đến 80, y là dòng có giá trị từ 1 đến 25.

7.6. Vài ví dụ.

Dưới đây là một số chương trình minh hoạ cách dùng các hàm vừa nêu.

Ví dụ 1:

Nếu ta bấm một số phím trong khi chương trình đang làm việc thì các ký tự tương ứng được tạm thời đưa vào bộ đệm. Ta có thể dùng các hàm kbhit và getch (hay getche) để nhận lại các ký tự này, và như vậy ta có một cách để xoá bộ đệm. Sau đây là chương trình minh hoạ. Phần đầu chương trình sẽ cho phát 20 tiếng chuông. Trong lúc này ta có thể nhanh tay bấm một số phím. Phần tiếp chương trình sẽ nhận lại các phím đã bấm và cho hiện lên màn hình. Chương trình dùng toán tử while (xem Chương 5).

```
/*
  Khi máy đang reo chuông hãy bấm một số phím.
  Chương trình sẽ nhận các phím vừa bấm và hiện lên màn hình.
*/
      #include <conio.h>
      main()
        {
          int n,ch;
          clrscr();     /* Xoá màn hình */
          n=20;
          while(n>0) /* réo 20 tiếng chuông */
              {
                --n;
                putch(7);
              }
```

```
    while(kbhit()) /*Nhận các phím đã bấm khi máy reo chuông*/
      {
        ch=getch();
        putch(ch);
      }
  }
```

Ví dụ 2:

Dưới đây là chương trình báo thức, nó sẽ réo chuông cho đến khi bấm một phím bất kỳ. Trong chương trình có dùng toán tử if (xem Chương 5).

```
#include <conio.h>
main()
  {
    tiep:
    putch(7);              /* reo chuông */
    if (!kbhit()) goto tiep; /* Khi chưa bấm phím thì trở lại câu lệnh
                               có nhãn tiep để tiếp tục reo chuông */
  }
```

Ví dụ 3:

Chương trình dưới đây cũng réo chuông như trong ví dụ trên, nhưng chỉ kết thúc khi bấm phím ESC (mã 27).

```
#include <conio.h>
main()
  {
    tiep:
    putch(7);          /* reo chuông */
    if (!kbhit())      /* Khi chưa bấm phím thì trở lại câu lệnh
        goto tiep;        có nhãn tiep để tiếp tục reo chuông */
    if (getch()!=27)   /*
        goto tiep;       Khi có bấm phím thì dùng hàm getch để
  }                      nhận. Sau đó kiểm tra xem đó có phải là
                         phím ESC không. Nếu không phải, thì
                         nhảy trở lại câu lệnh có nhãn tiep để
                         tiếp tục réo chuông.
                       */
```

Bài 1. Tổ chức nhập một ma trận nguyên cấp 6x6 dưới dạng một bảng 6 dòng và 6 cột.

Bài 2. Viết chương trình tạo hình sau:

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

Bài 3. Nhập phần thực x và phần ảo y của một số phức, sau đó in ra theo mẫu:

(x,y)

với các yều cầu:

+ Số in ra có 2 chữ số sau dấu chấm thập phân

+ Kết quả in ra tạo thành một dãy ký tự liên tiếp nhau (không có khoảng trống)

Bài 4. Lập chương trình:

a) Đầu tiên in ra các dòng chữ

NGON NGU LAP TRINH

b) 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 các dòng chữ

TURBO C

Nếu bấm P thì chương trình in ra các dòng chữ

TURBO PASCAL

c) Bấm tiếp phím bất kỳ thì chương trình kết thúc.

CHƯƠNG 5

CÁC TOÁN TỬ ĐIỀU KHIỂN

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các toán tử điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể đang từ một câu lệnh này nhảy tới thực hiện một câu lệnh khác ở trước hoặc sau nó. Đường đi của máy trở nên linh hoạt hơn và nhờ vậy ta có thể viết chương trình một cách hiệu qủa hơn. Xét về mặt công dụng có thể chia các toán tử điều khiển thành ba nhóm chính:

+ Nhảy không điều kiện (goto)

+ Rẽ nhánh (if, switch)

+ Tổ chức chu trình (for, while,do while)

Ngoài ra còn có một số toán tử khác có chức năng bổ trợ như break, continue. Trong chương này sẽ giới thiệu cách viết và các nguyên tắc hoạt động của các toán tử nêu trên.Chúng ta sẽ thấy các toán tử điều khiển của C có những khả năng làm việc rất linh hoạt, phong phú và mạnh mẽ. Tất cả những điều này được giải thích tỉ mỉ và được minh họa rõ ràng trên nhiều chương trình hoàn chỉnh đã thử nghiệm trên máy.

§1. NHẮC LẠI KHÁI NIỆM VỀ CÂU LỆNH VÀ KHỐI LỆNH

Trong C mỗi câu lệnh có thể viết trên một hoặc nhiều dòng và được kết thúc bằng dấu chấm phẩy. Khái niệm về khối lệnh hay câu lệnh hợp thành đã trình bày ở chương 2. Ở đây ta chỉ nhắc lại vài điều.

- Khối lệnh là một dãy các câu lệnh đặt trong các dấu { } .

- Không được đặt dấu chấm phảy sau dấu ngoặc nhọn kết thúc khối.

- Khối lệnh tương đương với câu lệnh riêng lẻ về mặt cú pháp. Nói cách khác, chỗ nào đặt được một câu lệnh thì ở chỗ đó ta cũng có quyền viết một khối lệnh.

- Khi khối lệnh chỉ gồm một câu lệnh thì có thể bỏ các dấu ngoặc nhọn đầu và cuối. Nói cách khác có thể xem câu lệnh là trường hợp riêng của khối lệnh.

Dưới đây khi trình bầy các toán tử điều khiển như if, for, while, ..., ta dùng thuật ngữ "khối lệnh", nhưng mọi điều vẫn đúng nếu ta dùng "câu lệnh" (vì câu lệnh xem như trường hợp riêng của khối lệnh).

§2. TOÁN TỬ IF

Toán tử if cho phép lựa chọn một trong hai nhánh tùy thuộc vào sự bằng không hay khác không của một biểu thức, nó có hai cách viết sau:

if (biểu thức)
  khối lệnh 1;
else
  khối lệnh 2;
	if (biểu thức)
	  khối lệnh 1;

(Dạng 1)	(Dạng 2)

Chú ý: Biểu thức có thể nguyên hoặc thực.

Sự hoạt động của toán tử if dạng 1: Trước tiên máy sẽ xác định giá trị của biểu thức. Nếu biểu thức đúng (có giá trị khác không) máy sẽ thực hiện khối lệnh 1 sau đó nhảy tới các lệnh viết sau khối lệnh 2. Nếu biểu thức sai (có giá trị bằng 0) thì máy sẽ thực hiện khối lệnh 2 và sau đó thực hiện các lệnh viết sau nó.

Sự hoạt động của toán tử if dạng 2: Cũng như ở dạng 1, máy sẽ tính giá trị của biểu thức. Nếu biểu thức đúng thì máy thực hiện khối lệnh 1 và sau đó thực hiện các lệnh tiếp theo. Nếu biểu thức sai thì máy bỏ qua khối lệnh 1 mà chuyển đến các lệnh viết sau nó.

Ví dụ 1. Để tính max của hai biến a và b ta có thể sử dụng hai chương trình ứng với hai dạng khác nhau của toán tử if.

```
/* Chuong trinh tinh max cua hai so thuc , ban 1 */
#include "stdio.h"
    main()
    {
        float a,b,max;
        int k;
        tt: printf("\n Vao hai so a va b ");
        scanf("%f%f",&a,&b);  /*gia tri cua a va b nhan tu ban phim*/
        if (a>b) max = a;
        else max = b;
        printf("\n a = %8.2f\n b = %8.2f\n max = %8.2f",a,b,max);
        printf("Co tiep tuc khong - C/K");
        k=getch();
        if( k=='c' || k=='C') goto tt;
    }
```

```
/* Chuong trinh tinh max cua hai so thuc, ban 2 */
  include "stdio.h"
  main()
   {
     float a,b,max;
     int k;
     tt: printf("\n Vao hai so a va b ");
     scanf("%f%f",&a,&b);  /*gia tri cua a va b nhan tu ban phim*/
     max = a;
     if (b>max) max = b;
     printf("\n a = %8.2f\n b = %8.2f\n max = %8.2f",a,b,max);
     printf("Co tiep tuc khong - C/K");
     k=getch();
     if( k=='c'||k=='C') goto tt;
   }
```

Ví dụ 2:

Để tính max và min của 2 biến a, b, có thể dùng câu lệnh sau:

```
if (a>b)
   {
    max = a;
    min = b;
   }
else
   {
    max = b;
    min = a;
   }
```

Sự lồng nhau của toán tử if: Cho phép sử dụng các toán tử if lồng nhau. Điều đó có nghĩa là: các khối lệnh 1 và khối lệnh 2 (xem dạng 1 và dạng 2) lại có thể chứa các toán tử if khác. Trong trường hợp này nếu không sử dụng các dấu đóng mở khối thì rất dễ gây ra sự hiểu nhầm . Khi số từ khóa if bằng số từ khóa else:

```
if
else
    .
    .
    .
    if
    else
```

ta dễ dàng xác định được từng cặp if và else tương ứng. Vấn đề đặt ra: Máy sẽ sử lý như thế nào trong trường hợp số từ khóa else ít hơn số từ khóa if. Câu trả lời như sau: else được gắn với if không có else gần nhất trước đó.

Chẳng hạn, trong đoạn chương trình

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

thì else sẽ đi với if bên dưới. Để tránh nhầm lẫn ta nên sử dụng khối lệnh, khi đó đoạn chương trình trên có thể viết:

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

Chú ý cách viết chương trình có cấu trúc:

Để chương trình 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 tụt 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 thẳng cột.

Đoạn chương trình trên và ví dụ 2 đã minh hoạ cách viết chương trình có cấu trúc.

§3. else if

Khi muốn thực hiện một trong n quyết định ta có thể sử dụng toán tử if dưới dạng sau:

```
if (biểu thức 1)
     Khối lệnh 1;
else if (biểu thức 2 )
   Khối lệnh 2;
```

...

else if (biểu thức n-1);

    Khối lệnh n-1;

else

    Khối lệnh n;

Đối với toán tử này thì:

- Chỉ một trong n khối lệnh trên được thực hiện.

- Nếu biểu thức thứ i là biểu thức đầu tiên khác 0 (i = 1,.. n-1) thì câu lệnh i được thực hiện.

- Nếu cả n-1 biểu thức đều có giá trị bằng 0 thì câu lệnh n được thực hiện.

Ví dụ 1:

Giả sử để theo dõi trình độ của cán bộ ta dùng bảng mã sau:

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

Chương trình để từ mã suy ra trình độ học vấn có thể viết như sau:

```
/* Chuong trinh tim trinh do hoc van theo ma
 Bản 1 dùng else if */
        # include "stdio.h"
        main()
          {
            int ma;
            tt: printf ("\n ma = ");
            scanf("%d",&ma);
            fprintf(stdprn, "\n ma = %2d",ma);
            if(ma==1)
              fprintf(stdprn,"so cap\n");
            else if(ma == 2)
              fprintf(stdprn,"trung cap\n");
```

```
else if (ma == 3)
  fprintf(stdprn,"dai hoc\n");
else if (ma == 4)
  fprintf(stdprn,"cao hoc\n");
else if(ma == 5)
  fprintf(stdprn,"pho tien sy\n");
else if(ma == 6)
  fprintf(stdprn,"tien sy\n");
else
  fprintf(stdprn,"sai\n");
}
```

Ví dụ 2:

Chương trình giải phương trình bậc hai có thể viết như sau:

```
#include <stdio.h>
main()
  {
    float a,b,c,d;
    float x1,x2;
      printf("\nNhap a, b ,c: ");
      scanf("%f%f%f", &a,&b,&c);
      d = b*b - 4*a*c;
    if (d<0.0)
      printf("\nPhuong trinh vo nghiem");
    else
     if(d==0.0)
     printf("\nPhuong trinh co nghiem kep= %0.2f",-b/(2*a));
     else
      {
        x1= (-b - sqrt(d))/(2*a);
        x2= (-b + sqrt(d))/(2*a);
        printf("\nx1 = %0.2f x2 = %0.2f", x1, x2);
      }
  }
```

§4.TOÁN TỬ switch

Toán tử switch cho phép căn cứ vào giá trị của một biểu thức nguyên để chọn một trong nhiều cách nhảy. Nó có dạng sau:

```
switch (biểu thức nguyên)
{
    case n1:
        Các câu lệnh
    case n2:
        Các câu lệnh
    . . .
    case nk:
        Các câu lệnh
    [default:
        Các câu lệnh ]
}
```

Ở đây ni là các số nguyên, hằng kí tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch. Dưới đây, khi nói ra khỏi toán tử switch tức là ra khỏi thân của nó. Điều đó có nghĩa là máy sẽ nhảy tới câu lệnh ở ngay sau dấu } (dấu kết thúc thân của switch).

default là một thành phần không bắt buộc.

+ **Sự hoạt động của toán tử switch** phụ thuộc vào giá trị của biểu thức viết trong các dấu ngoặc tròn.

1/ Khi giá trị này bằng ni máy sẽ nhảy tới câu lệnh có nhãn case ni.

2/ Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay vắng mặt của default.

a. Khi có default, máy nhảy tới câu lệnh có nhãn default.

b. Khi không có default máy ra khỏi toán tử switch .

+ **Ra khỏi toán tử switch.** Máy sẽ ra khỏi toán tử switch khi nó gặp một câu lệnh break hoặc gặp dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể sử dụng câu lệnh goto trong thân toán tử switch để nhảy tới một câu lệnh bất kì bên ngoài switch. Khi toán tử switch nằm trong thân một hàm nào đó, ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (câu lệnh return trình bầy trong chương 6)

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

Chương trình tìm trình độ học vấn trong §3 có thể viết theo một dạng khác nhờ sử dụng toán tử switch như sau.

```
/*Chuong trinh tim trinh do hoc van theo ma
 Ban 2, dung switch */
        #include "stdio.h"
        main()
          {
            int ma;
            tt: printf("\n ma = ");  /* chon ma tu ban phim */
            scanf("%d",&ma);
            fprintf(stdprn,"\n ma = %2d",ma);
            switch(ma)
             {
              case 1: fprintf(stdprn," so cap \n");
                    break;
              case 2: fprintf(stdprn," trung cap \n");
                    break;
              case 3: fprintf(stdprn," dai hoc \n");
                    break;
              case 4: fprintf(stdprn," cao hoc \n");
                    break;
              case 5: fprintf(sdtprn," pho tien si \n");
                    break;
              case 6: fprintf(stdprn," tien sy \n");
                    break;
              default:
              fprintf(stdprn," sai\n");
             }
        }
```

§5. TOÁN TỬ GOTO VÀ NHÃN

Nhãn có thể viết như tên biến và có thêm dấu hai chấm (:) đứng sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình. Chẳng hạn, trong ví dụ:

```
tiep_tuc: s += a[i];
```

thì tiep_tuc là nhãn của câu lệnh gán s += a[i].

Toán tử goto có dạng:

```
goto nhan;
```

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

Khi dùng toán tử goto cần lưu ý các điểm sau:

- Câu lệnh goto và nhãn cần nằm trong một hàm. Điều đó nói lên rằng: toán tử 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 hàm này sang hàm khác.

- Không cho phép dùng toán tử goto để nhảy từ ngoài vào trong một khối lệnh. Tuy nhiên việc nhảy từ trong ra ngoài khối lệnh lại hoàn toàn hợp lệ. Chẳng hạn đoạn chương trình

```
goto alfa   /* Ví dụ sai */
      .
      .
      {
        .
        .
        alfa: printf("\nKet qua tinh toan ");
        .
        .
        .
      }
```

là sai vì nó thực hiện ý định nhảy từ ngoài vào trong một khối lệnh.

Khi sử dụng toán tử if và goto theo mẫu

```
lap_lai: câu lệnh 1;
         câu lệnh 2;
         .
         .
         câu lệnh n;
if (biểu thức) goto lap_lai;
```

ta sẽ nhận được một chu trình. Thân chu trình gồm n câu lệnh sẽ được thực hiện lặp đi lặp lại chừng nào mà biểu thức trong if vẫn còn đúng (có giá trị khác 0).

Xét một ví dụ: Giả sử cần lập chương trình tính tích vô hướng của hai véc tơ x và y. Các phần tử của x và y được đưa vào từ bàn phím. Dưới đây là chương trình nhập 2 véc tơ và tính tích vô hướng của chúng.

```
/*
 Chuong trinh tinh tich vo huong hai vec to, dung if va goto
*/
        #include "stdio.h"
        #define N 100 /* chieu cua vecto khong qua 100 */
        main()
          {
            float x[N],y[N],s;
            int n,i;
                /* so chieu thuc te cua cac vecto x va y */
            printf("\n chieu cua vecto = ");
            scanf("%d",&n);
                /* vao so lieu cho cac vecto x va y */
            i = 1;
            vaols:
            printf("\n x[%d] = ", i);
            scanf("%f",&x[i]);
            printf("\n y[%d] = ", i);
            scanf("%f",&y[i]);
            if ( ++i <= n )
                goto vaols;
                /* tinh tich vo huong */
            s = 0.0;
            i = 1;
            tvh:
            s + = x[i]*y[i];
            if( ++i <= n )
                goto tvh;
```

```
    /* in ket qua */
    i = 1;
    inkq:
    printf("\nx[%d]= %8.2f y[%d]= %8.2f",i,x[i],i,y[i]);
    if( ++i <= n )
        goto inkq:
    printf("\n\n tich vo huong = %8.2f", s);
}
```

Chương trình trên gồm ba chu trình ứng với ba phần việc: vào số liệu, tính tích vô hướng và in kết qủa. Bạn hãy viết một chương trình khác cho bài toán nêu trên nhưng chỉ dùng một chu trình.

§6. TOÁN TỬ for

Như đã chỉ ra ở mục §5: có thể tạo nên chu trình bằng cách dùng các toán tử if và goto. Toán tử for cho ta một cách khác xây dựng chu trình hiệu qủa hơn, nó có dạng sau:

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

    Khối lệnh; /* thân chu trình */

Như vậy toán tử for gồm ba biểu thức và thân for (thân chu trình for). Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khóa for. Bất kỳ biểu thức nào trong ba biểu thức nói trên đều có quyền vắng mặt nhưng phải giữ dấu chấm phẩy (;).

Thông thường biểu thức 1 là một toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức 3 là một toán tử gán dùng để thay đổi giá trị của biến điều khiển.

Sự hoạt động của toán tử for:

Toán tử for làm việc theo các bước sau:

1/ Xác định biểu thức 1.

2/ Xác định biểu thức 2.

3/ Tùy thuộc vào tính đúng, sai của biểu thức 2, máy sẽ lựa chọn một trong hai nhánh:

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

b- Nếu biểu thức 2 có giá trị khác 0 (đúng) máy sẽ thực hiện các câu lệnh trong thân for. Khi gặp dấu ngoặc đóng } cuối cùng của thân for hoặc gặp câu lệnh continue máy sẽ chuyển tới bước 4 (khởi đầu lại).

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

Nhận xét:

1/ Biểu thức 1 bao giờ cũng chỉ được tính một lần.

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

Vài chú ý:

1/ Khi biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần được thực hiện nhờ câu lệnh break, goto hoặc teturn (viết trong thân chu trình).

2/ Trong dấu ngoặc tròn sau từ khoá for gồm 3 phần phân cách nhau bởi dấu chấm phẩy. Trong mỗi phần không những có thể viết một biểu thức (như vừa nói ở trên) mà còn có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu thức trong mỗi phần sẽ được xác định từ trái sang phải.

Tính đúng sai của dãy biểu thức trong phần thứ hai được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy này.

3/ Bên trong thân của toán tử for ta lại có thể sử dụng các toán tử for khác. Bằng cách như vậy ta có thể xây dựng những chu trình lồng nhau (xem ví dụ 3 dưới đây).

4/ Khi gặp câu lệnh break (xem §9) trong thân for, máy sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này.

5/ Trong thân for có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ (xem ví dụ 3 dưới đây). Cũng có thể sử dụng toán tử return trong thân for để trở về một hàm nào đó. Dưới đây là một số ví dụ minh họa cách sử dụng toán tử for, if, switch và goto.

6/ Trong thân for có thể dùng câu lệnh continue (xem §10) để chuyển đến đầu một vòng lặp mới của chu trình. Nói rõ hơn: khi gặp continue thì máy sẽ bỏ qua các câu lệnh còn lại trong thân chu trình để chuyển đến xét biểu thức 3.

Ví dụ 1:

Các khía cạnh khác nhau của toán tử for được thể hiện trên 6 chương trình sau. Cả 6 chương trình này giải quyết cùng một bài toán: đảo ngược một dãy số.

```
/*
   Chuong trinh dao nguoc mot day so, Ban 1
*/
        #include "stdio.h"
        float x[] = { 63.2, -45.6, 70.1, 3.6, 14.5 };
        int n = sizeof(x)/sizeof(float);
        main()
          {
            int i,j;
            float c;
            for(i=0, j=n-1; i<j; ++i, --j)
              {
                c = x[i];
                x[i] = x[j];
                x[j] = c;
              }
            fprintf(stdprn, "\n Day ket qua \n \n");
            for( i=0; i<n; ++i )
            fprintf(stdprn,"%8.2f",x[i] );
          }
/*
   Chuong trinh dao nguoc mot day so, Ban 2
*/
        #include"stdio.h"
        float x[] = { 63.2, -45.6,70.1,3.6,14.5 };
        int n = sizeof(x)/sizeof(float);
        main()
          {
            int i,j;
            float c;
            for(i=0, j=n-1; i<j; c=x[i], x[i]=x[j], x[j]=c, ++i, --j);
              /* thân for là câu lệnh rỗng */
            fprintf(stdprn,"\n Day ket qua \n\n" );
            for(i= -1; ++i < n; )  /* vang mat bieu thuc 3 */
            fprintf(stdprn,"%8.2f",x[i] );
          }
```

```
/*
  Chuong trinh dao nguoc mot day so , Ban 3
*/
        #include "stdio.h"
        float x[] = { 63.2, -45.6, 70.1, 3.6, 14.5 };
        int n = sizeof(x)/sizeof(float);
        main()
          {
            int i = -1, j;
            float c;
            for (; n-1 - ++i, i<j; )  /* vang mat bieu thuc 1 va 3 */
             {
               c = x[i];
               x[i] = x[j];
               x[j] = c;
             }
            fprintf(stdprn,"\n Day ket qua \n");
            for( i=0; i-n; )      /* thay quan he i<n bang bieu thuc i-n */
            fprintf(stdprn,"%8.2f" , x[i++] );
          }
/*
  dao nguoc mot day so, Ban 4
*/
        #include"stdio.h"
        float x[] = { 63.2, -45.6, 70.1, 3.6, 14.5 };
        int n = sizeof(x)/sizeof(float);
        main()
          {
          int i = 0, j = n-1;
          float c;
          for(;; ) /* vang mat ca 3 bieu thuc */
             {
               c = x[i];
               x[i] = x[j];
               x[j] = c;
```

```
            if(++i >= --j)
                goto tt; /* ra khoi for bằng goto */
        }
        tt: fprintf(stdprn, "\n Day ket qua \n \n");
        for( i = -1; i++ < n-1; fprintf(stdprn,"%8.2f", x[i]) );
    }
```

```
/*
   dao nguoc mot day so, Ban 5
*/
    #include"stdio.h"
    float x[] = { 63.2, -45.6, 70.1, 3.6, 14.5 };
    int n = sizeof(x)/sizeof(float);
    main()
      {
        int i = 0, j = n-1;
        float c;
        for(;; ) /* vang mat ca 3 bieu thuc */
        {
          c = x[i];
          x[i] = x[j];
          x[j] = c;
          if( ++i >= --j)
            break;          /* ra khoi for bằng break */
        }
        fprintf(stdprn, "\n Day ket qua \n \n");
        for( i = -1; i++ < n-1; fprintf(stdprn,"%8.2f", x[i]) );
      }
```

```
/*
   Chuong trinh dao nguoc mot day so, Ban 6
*/
    #include"stdio.h"
    float x[] = { 63.2, -45.6, 70.1, 3.6, 14.5 };
    int n = sizeof(x)/sizeof(float);
    main()
      {
```

```
int i = 0;
int j = n-1;
float c;
for(;c = x[i], x[i] = x[j], x[j] = c , ++i < --j; );
fprintf(stdprn,"\n Day ket qua \n \n ");
for( i = 0; fprintf(stdprn, "%8.2f",x[i], ++i - n; );
}
```

Ví dụ 2:

Sử dụng bảng mã trong §3 và biết rằng trong mảng a chứa mã trình độ của 15 cán bộ. Cần thống kê:

- Số cán bộ có trình độ từ đại học trở lên
- Số cán bộ có trình độ từ cao học trở lên
- Số cán bộ có trình độ từ phó tiến sĩ trở lên
- Số cán bộ là tiến sĩ

Sử dụng toán tử for và switch có thể viết chương trình cho bài toán trên như sau.

```
/* Chuong trinh thong ke can bo theo trinh do */
#include"stdio.h"
int a[] = {6,1,4,3,2,5,1,6,4,5,2,3,1,3,5};
main()
{
    int i, trendh=0, trench=0, trenpts= 0, ts = 0;
    for(i=0; i<15; ++i)
    switch(a[i])
    {
        case 6: ts++;
        case 5: trenpts ++;
        case 4: trench ++;
        case 3: trendh ++;
    }
    printf("\nSo can bo tu dai hoc tro len la: %2d",trendh);
    printf("\nSo can bo tu cao hoc tro len la: %2d",trench);
    printf("\nSo can bo tu pts tro len la: %2d",trenpts);
    printf("\nSo can bo co hoc vi tien sy la: %2d",ts);
}
```

Ví dụ 3:

Chương trình tìm phần tử âm đầu tiên của một ma trận dưới đây sẽ minh họa cách sử dụng các chu trình lồng nhau và cách dùng goto, break để ra khỏi chu trình (Bản 1 dùng goto và bản 2 dùng break).

```
/*
Chuong trinh tim phan tu am dau tien cua ma tran Bản 1 dùng goto
*/
   #include "stdio.h"
   float a[3][4] = {
                  { 15,46,3.5,6,3 },
                  { 34,0,-25,35 } ,
                  { 1,-13,46,-38 }
                 };
   main()
      {
        int i,j;
        for( i=0; i<3; ++i )
        for (j=0; j<4; ++j )
        if (a[i][j]<0 ) goto tim_thay;
        printf("\n ma tran khong co phan tu am ");
        goto ket_thuc;
        tim_thay:
        printf("\nPhan tu am dau tien la a(%d,%d)= %8.2f",i+1, j+1,
              a[i][j]);
        ket_thuc: ; /*cau lenh rong */
      }
/*
Chuong trinh tim phan tu am dau tien cua ma tran Bản 2 dùng break
*/
      #include "stdio.h"
      float a[3][4] = {
                     { 15,46,3.5,6,3 },
                     { 34,0,-25,35 } ,
                     { 1,-13,46,-38 }
                    };
```

```
main()
{
  int i,j;
  for( i=0; i<3; ++i )
    {
      for (j=0; j<4; ++j )
      if (a[i][j]<0)
        break;    /* ra khỏi for j */
      if (j<4)
        break;    /* ra khỏi for i */
    }
  if (i<3 && j<4)
 printf("\nPhan tu am dau tien la a(%d,%d)= %8.2f",i+1,j+1,a[i][j]);
 else
 printf("\n ma tran khong co phan tu am ");
}
```

Ví dụ 4.

Chương trình dưới đây giải quyết bài toán tìm giá trị lớn nhất và nhỏ nhất trên mỗi hàng của ma trận. Trong chương trình, sử dụng các chu trình lồng nhau.

```
/*
   Chuong trinh tim max va min tren moi hang ma tran
*/
       #include "stdio.h"
       float a[3][4] = {
                   {15,46,3.5,6.3 },
                   { 341,0,-25,35 },
                   {1,13,46,31 }
                 };
       main()
         {
           int i,,cotmax [3],min[3];
           float max [3], min[3];
           for ( i=0; i<3; ++i )
```

```
    {
     max[i] = min[i] = a[i][0];
     cotmax[i] = cotmin[i] = 0;
     for( j=1; j<4; ++j )
     {
      if (max[i]< a[i][j])
        {
            max[i] = a[i][j];
            cotmax[i] = j;
        }
      if (min[i]>a[i][j])
        {
            min[i] = a[i][j];
            cotmin[i] = j;
        }
     }
    }
    for( i=0; i<3; ++i )
    printf("\nhang: %d max= a(%d,%d)= %5.2f\
    min= a(%d,%d)= %5.2f", i+1,i+1, cotmax[i]+1,
    max[i],i+1,cotmin[i]+1,min[i]);
}
```

§7. TOÁN TỬ while

Cũng giống như for, toán tử while dùng để xây dựng các chu trình, nó có dạng sau:

while (biểu thức)

Khối lệnh /* Thân chu trình */

Như vậy toán tử while gồm một biểu thức và thân while (thân chu trình). Thân while là một câu lệnh hoặc một khối lệnh.

Sự hoạt động của while:

Toán tử while làm việc theo các bước sau:

1/ Xác định giá trị của biểu thức (viết sau while)

2/ Tùy thuộc vào tính đúng sai của biểu thức này, máy sẽ lựa chọn một trong hai nhánh:

a/ Nếu biểu thức có giá trị 0 (sai), máy sẽ ra khỏi chu trình và chuyển tới câu lệnh sau thân while.

b/ Nếu biểu thức có giá trị khác 0 (đúng) máy sẽ thực hiện các câu lệnh trong thân while. Khi gặp dấu ngoặc nhọn đóng cuối cùng của thân while máy sẽ trở lại bước 1.

Nhận xét. Thân của 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 (sau while) đã sai.

Vài chú ý.

1/ Cũng giống như đối với for, trong các dấu ngoặc tròn sau while chẳng những có thể đặt một biểu thức mà còn có thể viết một dãy biểu thức phân cách nhau bởi dấu phẩy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

2/ Bên trong thân của một toán tử while lại có thể sử dụng các toán tử for hoặc while khác. Bằng cách đó ta có thể xây dựng được các chu trình lồng nhau.

3/ Khi gặp câu lệnh break (xem §9) trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

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

5/ Trong thân while có thể dùng câu lệnh continue (xem §10) để chuyển đến đầu một vòng lặp mới của chu trình. Nói rõ hơn: khi gặp continue thì máy sẽ bỏ qua các câu lệnh còn lại trong thân chu trình để trở lại bước tính và kiểm tra biểu thức (viết sau while).

Các ví dụ

Ví dụ 1:

Khi sử dụng các khả năng của toán tử while, chương trình tính tích vô hướng của hai véc tơ và y có thể viết theo các cách sau.

/* Chuong trinh tinh tich vo huong, **Ban 1** */

```
#include "stdio.h"
float x[] = {4,6,8,3.5},
y[] = {2.6,3.2,4,7};
main()
  {
```

```
        float s=0;
        int i = -1;
        while ( ++i<4 )
        s += x[i]*y[i];
        printf("\ntich vo huong = %8.2f",s);
      }
/*
  chuong trinh tinh tich vo huong, Ban 2
*/
    #include"stdio.h"
    float x[] = {4,6,8,3.5},
    y[] = {2.6,3.2,4,7};
    main()
      {
        float s=0;
        int i=0;
        while(1) /* bieu thuc luon luon dung */
          {
           s += x[i]*y[i];
           if (i++ >= 4)  goto xong;
          }
        xong: printf("\ntich vo huong = %8.2f",s);
      }
/*
  chuong trinh tinh tich vo huong, Ban 3
*/
    #include "stdio.h"
    float x[] = {4,6,8,3.5},
    y[] = {2.6,3.2,4,7};
    main()
      {
        float s=0; int i=0;
        while(s+=x[i]*y[i], ++i-5);
                   /* than while la cau lenh rong */
        printf("\ntich vo huong = %8.2f",s);
      }
```

Ví dụ 2:

Đoạn chương trình dưới đây sẽ cho số nguyên dương n nhỏ nhất sao cho 1 + 2 + ... + n > 10000.

```
int s=1, n=1;
while (s<=10000)
    s += ++n;
```

Ví dụ 3:

Cho dãy n số nguyên chứa trong mảng a (bắt đầu từ a[1]). Hãy xác định xem có tồn tại phần tử âm trong dãy hay không?. Nếu có, thì cho biết phần tử âm đầu tiên. Đoạn chương trình sau sẽ đáp ứng yêu cầu trên.

```
int i=1;
while (a[i]>=0 && i<=n) ++i;
  if (i<=n)
     printf("\n Phan tu am dau tien = a[%d] = %d", i,a[i]);
  else
     printf("\n Day khong co phan tu am");
```

§8.TOÁN TỬ do while

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

Toán tử do while có dạng sau:

```
do
  Khối lệnh  /* Thân chu trình */
while (biểu thức);
```

Ta nhận thấy, các thành phần của toán tử do while theo thứ tự gồm: từ khóa do, thân chu trình, từ khóa while, biểu thức và dấu chấm phẩy. Thân do while (cũng giống for và while) có thể là một câu lệnh riêng lẻ hoặc một khối lệnh.

Sự hoạt động của do while. Toán tử này làm việc theo các bước sau.

1/ Thực hiện các câu lệnh trong thân do while.

2/ Khi gặp dấu ngoặc nhọn cuối cùng của thân do while, máy sẽ xác định giá trị của biểu thức sau từ khóa while.

3/ Máy sẽ phân nhánh theo giá trị của biểu thức vừa nhận được - Nếu biểu thức có giá trị đúng (khác 0), máy sẽ trở lại bước 1 để tiếp tục thực hiện vòng mới của chu trình.

- Nếu biểu thức có giá trị sai (bằng 0), máy sẽ ra khỏi chu trình và chuyển tới câu lệnh đứng sau dấu chấm phảy đặt cuối toán tử do while.

Chú ý. Những điều cần lưu ý đối với toán tử while nêu trong §7 cũng đúng với toán tử do while.

Ví dụ minh hoạ

Ví dụ 1:

Để tính căn bậc hai của một số dương a có thể dùng công thức lặp:

x(0) = a

x(n+1) = (x(n)*x(n) + a) / (2*x(n)) , với n>=0

Quá trình lặp kết thúc khi

abs((x(n+1)-x(n)) / x(n))<0.00001

và khi đó x(n+1) được xem là giá trị gần đúng của căn a.

Thuật toán trên được chương trình hoá như sau.

```
/* Chuong trinh tinh can bac hai */
        #include "math.h"
        #include "stdio.h"
        main()
          {
            double a,xn,c;
            int ch;
            tt: printf("\n a = ");
            scanf("%lf",&a);    /* vao a tu ban phim */
            if (a<0)
              {
               printf("\n a<0 ? ");
               goto kt;
              }
            if (a == 0 )
              {
               xn = 0;
               goto kq;
              }
            /* Bat dau vao thuat toan */
            xn = a;
            do
```

```
        {
          c = xn;
          xn = ( xn*xn + a )/(2*xn);
        }
      while (fasb((xn-c)/c) >= 1e-5);
      kq: printf("\n a = %8.2f sqrt(a) = %8.2f",a,xn);
      printf("\nCo tiep tuc khong? - C/K");
      ch=getch();
      if ( ch=='c' || ch=='C' )
        goto tt;
        kt: ;
    }
```

Ví dụ 2:

Đoạn chương trình xác định phần tử âm đầu tiên trong ví dụ 3, §7 có thể viết lại bằng cách dùng do while như sau:

```
int i=0;
do
  ++i;  /* Thân chu trình là câu lệnh */
    while (a[i]>=0 && i<=n);
      if (i<=n)
        printf("\n Phan tu am dau tien = a[%d] = %d", i,a[i]);
      else
printf("\n Day khong co phan tu am");
```

§9. CÂU LỆNH break

Câu lệnh break cho phép ra khỏi for, while, do while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình (hoặc switch) bên trong nhất chứa nó. Như vậy, break cho ta khả năng ra khỏi một chu trình (từ một điểm bất kỳ bên trong chu trình) mà không dùng đến điều kiện kết thúc chu trình. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ở các mục trên đã có một vài ví dụ về việc sử dụng câu lệnh break. Bây giờ sẽ trình bầy một ví dụ khác.

Biết số nguyên dương n sẽ là nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn hai của n. Thuật toán trên sẽ được sử dụng trong đoạn chương trình dưới đây để kiểm tra tính nguyên tố của n.

```
int i,n, ng_to=1;
for (i=2; i <= sqrt(n); ++i)
if ((n mod i)==0)
   {
      ng_to=0;
      break;
   }
if (ng_to)
  printf("\n%d la nguyen to",n);
else
  printf("\n%d la hop so",n);
```

§10. CÂU LỆNH continue

Trái với câu lệnh break (dùng để ra khỏi chu trình) câu lệnh continue dùng để bắt đầu một vòng mới của chu trình bên trong nhất chứa nó. Nói một cách chính xác hơn:

- Khi gặp câu lệnh continue bên trong thân của một toán tử for, máy sẽ chuyển tới bước khởi đầu lại (bước 4 trong điểm " Sự hoạt động của for ").

- Khi gặp câu lệnh continue bên trong thân của while hoặc do while, máy sẽ chuyển tới xác định giá trị biểu thức (viết sau từ khóa while) và sau đó tiến hành kiểm tra điều kiện kết thúc chu trình.

Một điểm cần lưu ý là: continue chỉ áp dụng cho các chu trình chứ không áp dụng cho switch.

Ví dụ:

Giả sử cần viết chương trình để từ một ma trận a cho trước:

- Tính tổng các phần tử dương của a.
- Xác định số phần tử dương của a.
- Tìm cực đại của các phần tử dương của A.

Chương trình sẽ như sau.

```
/*
  Chuong trinh xu ly cac phan tu duong cua ma tran
*/
        #include "stdio.h"
```

```
float a[][4] = {
                 { 25,0,-3,5 },
                   { -6,4,0,-2 },
                   { 30,-4,7,-3 }
                 };

main()
   {
      int i,j,k = 0;
      float s = 0, max = 0;
      for( i = 0; i < 3; ++i )
      for( j = 0; j < 4; ++j )
       {
         if( a[i][j] <= 0)
            continue;
            s += a[i][j];
         if (max < a[i][j])
            max = a[i][j];
            ++k;
       }
      printf("\nSo phan tu duong la: %d", k );
      printf("\nTong cac phan tu duong la: %8.2f",s);
      printf("\nMax cac phan tu duong la: %8.2f",max);
   }
```

Bài 1. Lập chương trình giải hệ

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

các hệ số a, b, c, d, e, f nhận từ bàn phím. Yêu cầu xét tất cả các trường hợp có thể.

Bài 2. Lập chương trình để:

- Nhập một dãy số từ bàn phím.

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

Bài 3. Lập chương trình tính ex theo công thức xấp xỉ

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}$$

với độ chính xác 0.00001. Tức là n cần chọn sao cho:

$$\frac{xn}{n!} < 0.00001$$

Bài 4. Lập chương trình tính sin(x) với độ chính xác 0.0001 theo công thức

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Bài 5. Lập chương trình để:

- Vào bốn dãy số $a_1,\dots,a_n$; $b_1,\dots,b_n$; $c_1,\dots,c_n$; $d_1,\dots,d_n$.
- In kết qủa trên n dòng, mỗi dòng 5 giá trị theo mẫu sau

a_i b_i c_i d_i $\min(a_i,b_i,c_i,d_i)$ $\max(a_i,b_i,c_i,d_i)$

Bài 6. Lập chương trình tính tích phân

$$\int_0^1 \frac{\sin(x^2)}{e^x} dx$$

theo công thức Sim Sơn.

Bài 7. Lập chương trình tính tổ hợp chập m của n

$$C_n^m = \frac{n.(n-1)\dots(n-m+1)}{m!}$$

Bài 8. Lập chương trình để in một dãy n số thực $x_1,...,x_n$ trên nhiều dòng, mỗi dòng gồm m số (dòng cuối cùng có thể ít hơn).

- Các giá trị m,n và $x_1,...,x_n$ nhận từ bàn phím.

Bài 9. Nhập các hệ số $a_1,...,a_n$ từ bàn phím sau đó in hàm f(x) trên một dòng theo mẫu sau:

$f(x) = a_1x_1 + . . . + a_{nn}$

Bài 10. Lập chương trình tính

$$S = (a_1^2 + ... + a_n^2)^{0.5}$$

trong đó n và $a_1,...,a_n$ nhận từ bàn phím.

Bài 11. Cần có tổng số 200000 đ từ 3 loại giấy bạc 1000 đ, 2000đ và 5000 đ. Lập chương trình để tìm tất cả các phương án có thể.

Bài 12. Lập chương trình tìm phần tử âm cuối cùng của dãy $a_1,...,a_n$.

Bài 13. Cho hai dãy số:

$a_1,...,a_n$ và $b_1,...,b_m$

cả hai đều xếp theo thứ tự tăng. Lập chương trình để từ hai dãy trên xây dựng một dãy mới cũng theo thứ tự tăng.

Bài 14. Cho dãy số $a_1,...,a_n$. Lọc các số dương đưa vào mảng b, các số âm đưa vào mảng c.

Bài 15. Sắp xếp một dãy số theo thứ tự tăng dần.

Bài 16. Cho dãy số $a_1,...,a_n$. Lập chương trình in các số âm trên một dòng, các số dương trên dòng tiếp theo.

Bài 17. Cho hiện lên màn hình các kí tự có mã ASCII từ 33 đến 255.

CHƯƠNG 6

HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

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 phần việc nào đó và chương trình sẽ giải quyết cả bài toán trọn vẹn. Một trong các ưu điểm của C là nó cho phép tổ chức và sử dụng các hàm một cách đơn giản và hiệu qủa. Chương này sẽ giới thiệu các qui tắc xây dựng và sử dụng hàm. Tất cả các vấn đề lý thuyết sâu xa, phức tạp sẽ được minh họa giải thích trên các chương trình hoàn chỉnh đã thử nghiệm trên máy.

§1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

1.1. Xét bài toán đơn giản sau: Tìm giá trị lớn nhất của ba số mà giá trị của chúng được đưa vào từ bàn phím. Ta tổ chức chương trình thành hai hàm: hàm main và một hàm mà ta đặt tên là max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số giả định mà ta gọi là a, b, c. Nhiệm vụ của hàm main là đọc ba giá trị từ bàn phím, dùng hàm max3s để tính max của ba giá trị vừa đọc được, đưa kết quả ra màn hình và đọc tiếp ba giá trị khác nếu muốn. Chương trình được viết như sau.

```
#include "stdio.h"
float max3s(float a,float b,float c); /*Nguyên mẫu của hàm*/
main() /* bat dau ham main */
  {
    float x,y,z;
    int ch;
    tt: printf("\n vao ba so: ");
    scanf("%f%f%f", &x,&y,&z);
    printf("\nx= %0.2f\ny= %0.2f\nz= %0.2f\n max= %8.2f",
    x, y, z, max3s(x,y,z));
    printf("Co tiep tuc khong? - C/K");
    ch = getch();
    if (ch == 'c' || ch == 'C') goto tt;
  } /* ket thuc ham main */
```

```
/* Dòng đầu khai báo kiểu hàm, tên hàm, kiểu đối và tên đối */
float max3s(float a, float b, float c)
  {
    float max;  /* Biến cục bộ dùng trong thân hàm */
    max = a>b?a:b;
    return (max>c?max:c);  /* Giá trị hàm trả về */
  }  /* ket thuc ham max3s */
```

Chú ý:

+ Không nhất thiết phải khai báo nguyên mẫu (prototype) của hàm. Nhưng nên có, vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm (số đối không đúng) hay tự động việc chuyển dạng (ví dụ đổi từ int trong lời gọi hàm sang float là kiểu của đối).

+ Nguyên mẫu của hàm thực chất là dòng đầu của hàm và thêm vào dấu chấm phẩy. Tuy nhiên trong nguyên mẫu có thể bỏ tên các đối.

Qua chương trình này có thể rút ra nhiều khái niệm và qui tắc quan trọng liên quan đến việc xây dựng và sử dụng hàm.

+ Trước hết mỗi hàm phải có một tên. Ta có thể đặt cho mỗi hàm một tên bất kỳ theo các qui tắc nêu trong chương 1 mục 3. Lẽ dĩ nhiên trong cùng một chương trình, các hàm phải có tên khác nhau và mặc dù tên là tùy ý nhưng ta cũng nên đặt cho hàm những tên phù hợp với chức năng của nó.

+ Hàm thường có một vài đối. Ví dụ hàm max3s có ba đối là a, b, c. Cả ba đối này đều có kiểu giá trị float. Cũng có hàm không đối như hàm main chẳng hạn.

+ Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị của các đối. Hàm max3s cho giá trị lớn nhất của ba đối của nó. Giá trị của hàm có thể có kiểu int, float, double, ... Hàm max3s có giá trị kiểu float. Dưới đây sẽ thấy có những hàm không có giá trị.

Khi xây dựng và sử dụng hàm cần quán triệt các qui tắc sau:

1.2. Quy tắc xây dựng một hàm.

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

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

Đây là cách khai báo hàm theo kiểu ANSI (American National Standards Institute - Viện tiêu chuẩn quốc gia Mỹ) vì đó là đặc trưng của C được uỷ ban ANSI bổ sung. Các chương trình dịch C trước đó sử dụng cú pháp khác:

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

Tức là khai báo đối riêng ra. Phong cách cũ này được Turbo C chấp nhận và chúng ta sẽ gặp trong nhiều chương trình C cũ. Tuy nhiên phong cách mã hoá này nhiều lỗi hơn phong cách ANSI và nên tránh.

+ Thân hàm

Sau dòng tiêu đề là thân hàm. 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 đã đề ra cho hàm. Thân của hàm max3s là đoạn chương trình tính giá trị lớn nhất của ba đối a,b,c.

Trong thân hàm ta dùng thêm biến max. Cần phân biệt sự khác nhau giữa biến max và đối a,b,c. Biến max là biến cục bộ nó chỉ có tác dụng trong thân hàm và không có bất cứ một liên hệ gì đến các biến của các hàm khác trong chương trình. Trái lại các đối a, b, c lại được dùng để trao đổi dữ liệu giữa các hàm.

Trong thân hàm có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở những chỗ khác nhau và cũng có thể không sử dụng câu lệnh này. Dạng tổng quát của nó là:

return [biểu thức];

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

Trong ví dụ đang xét, giá trị của biểu thức này là giá trị lớn nhất của ba đối a,b,c.

1.3. Quy tắc hoạt động của hàm.

Trở lại ví dụ trên ta thấy tham số cuối của hàm printf là max3s(x,y,z), đó là lời gọi hàm max3s. Một cách tổng quát, lời gọi hàm có dạng sau:

tên_hàm ([danh sách tham số thực])

Một điều cần nhớ khi viết lời gọi hàm là: số tham số thực phải bằng số tham số hình thức (đối) và mỗi tham số thực phải có cùng kiểu giá trị như kiểu giá trị của đối tương ứng với nó.

Khi gặp một lời gọi hàm thì hàm bắt đầu được thực hiện. Nói cách khác, khi máy gặp một lời gọi hàm ở một chỗ nào đó của chương trình, thì máy sẽ tạm rời chỗ đó và chuyển đến hàm tương ứng. Qúa trình đó sẽ diễn ra theo trình tự 4 bước như sau:

a/ Cấp phát bộ nhớ cho các đối và các biến cục bộ.

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

c/ Thực hiện các câu lệnh trong thân hàm.

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

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

Trong ví dụ trên, giá trị của hàm max3s(x,y,z) sẽ được sử dụng trong câu lệnh printf. Nói cách khác, giá trị lớn nhất của ba số x, y, z sẽ được đưa ra màn hình.

1.4. Cấu trúc tổng quát của chương trình. Chương trình gồm nhiều hàm được viết theo trình tự sau:

+ Các #include

+ Các #define

+ Khai báo các đối tượng dữ liệu ngoài

(biến, mảng, cấu trúc, hợp,...)

+ Khai báo nguyên mẫu của các hàm

+ Hàm main

+ Định nghĩa các hàm

Chú ý: Hàm main có thể đặt sau hoặc xen vào giữa các hàm khác.

§2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

Trong mục này ta sẽ hệ thống hóa những điều đã nói rải rác ở mục §1 và bổ sung thêm vài điểm mới.

2.1. Liên quan đến hàm có các khái niệm sau.

- Tên hàm
- Kiểu giá trị của hàm
- Đối hay tham số hình thức
- Thân hàm
- Khai báo hàm (khai báo prototype)
- Lời gọi hàm
- Tham số thực

Đối với mỗi hàm bao giờ cũng có 2 giai đoạn khác nhau là: xây dựng hàm và sử dụng hàm.

2.2. Xây dựng hàm bao gồm các việc.

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

```
type ten_ham (khai báo các đối)
   {
      khai báo các biến cục bộ
      các câu lệnh
      [return [bieu thuc];]
   }
```

Chú ý 1:. Đối với các hàm không cho giá trị (như thủ tục trong Pascal) thì dùng kiểu void. Ví du hàm dùng để hiển thị giá trị lớn nhất của ba đối thực có thể viết như sau:

```
void ht_max_3s(float a,float b,float c)
   {
      float x;
      x = a>b?a:b;
      printf("\n Max= %0.2f",x>c?x:c);
   }
```

Chú ý 2: Hàm không đối thì ta cũng dùng void để khai báo đối, ví dụ hàm báo thức có thể viết như sau:

```
void bao_thuc(void)
   {
      int i;
      for(i=0; i<10; ++i) putch(7);
   }
```

Chú ý 3: Khi gặp một toán tử return có chứa biểu thức, thì giá trị của biểu thức sẽ được chuyển kiểu cho phù hợp với kiểu của hàm hàm trước khi nó được gán cho hàm.

2.3. Sử dụng hàm.

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

ten_ham([danh sách các tham số thực])

Ở đây ta cần lưu ý:

+ Số tham số thực phải bằng số đối.

+ Kiểu của tham số thực phải phù hợp với kiểu của đối tương ứng.

2.4. Nguyên tắc hoạt động của hàm.

-Điều này đã nói khá đầy đủ trong §1, ở đây ta nói thêm một số điều về các tham số thực, các đối và các biến cục bộ.

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

- Đối và biến cục bộ đếu là các biến tự động. Chúng được cấp bộ nhớ khi hàm được xét đến và chúng sẽ lập tức bị xoá trước khi ra khỏi hàm. Như vậy, không thể mang giá trị của đối ra khỏi hàm. Điều đó cũng có nghĩa là không thể sử dụng đối để làm thay đổi giá trị của bất kỳ một đại lượng nào ở bên ngoài hàm. Ta cũng cần chú ý rằng: đối hoặc biến cục bộ có thể trùng tên với bất kỳ đại lượng nào ở ngoài hàm mà không gây ra một nhầm lẫn nào cả. Nói cách khác, khi xây dựng một hàm ta có thể tự do sử dụng các đối và các biến cục bộ mà không sợ chúng có trùng tên với các đối tượng nào khác ở bên ngoài hàm hay không.

- Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gắn cho các đối. Như vậy, các đối chính là bản sao của các tham số thực. Hàm chỉ làm việc trên các đối, tức là chỉ làm việc trên các bản sao này. Các đối có thể bị biến đổi trong thân hàm, nhưng các tham số thực (bản chính) không hề bị thay đổi.

Xét ví dụ sau. Giả sử ta cần xây dựng một hàm dùng để hoán vị hai biến thực và ta viết hàm này như sau.

```
void hoan_vi(float x, float y)    /* ví dụ sai */
  {
    float z;
    z=x;
    x=y;
    y=z;
  }
```

Ta viết thêm đoạn chương trình để sử dụng hàm này.

```
main()
  {
    float x,y;
```

```
    x = 3.5;
    y = 7.2;
    hoan_vi(x,y);
    printf ("\n x= %0.2f y= %0.2f",x,y);
}
```

Bây giờ ta phân tích xem chương trình trên làm việc ra sao và thu được cái gì. Theo câu lệnh đầu tiên của hàm main() máy sẽ dành hai khoảng nhớ (mỗi khoảng 4 byte) cho các biến x và y. Hai câu lệnh tiếp theo là gán 3.5 cho x và 7.2 cho y. Câu lệnh sau đó là lời gọi hàm. Các biến x và y trong lời gọi hàm là các tham số thực. Theo câu lệnh này máy tạm thời rời khỏi hàm main() để chuyển đến hàm hoan_vi. Các đối x, y và biến cục bộ z được cung cấp khoảng nhớ. ở đây tuy có sự trùng tên giữa các biến x, y ở hàm main() và các đối x ,y ở hàm hoan_vi nhưng máy vẫn phân biệt được theo nguyên lý sau: - Biến x ở hàm main() và đối x ở hàm hoan_vi được bố trí tại các khoảng nhớ khác nhau.

- Biến x tồn tại trong suốt cả qúa trình làm việc của chường trình , còn đối x chỉ tồn tại khi máy làm việc bên trong hàm hoan_vi.

- Khi máy gặp tên x ở hàm main() nó hiểu đó là biến x, còn khi máy gặp tên x ở hàm hoan_vi thì nó hiểu đó là đối x.

- Đối y và biến y cũng được phân biệt theo cách như vậy .

Trở lại sự làm việc của máy: sau khi cấp phát khoảng nhớ cho các đối x và y, máy sẽ sao giá trị từ các tham số thực sang các đối tương ứng . Như vậy giá trị của biến x sẽ được gán cho đối x và giá trị của biến y sẽ được gán cho đối y. Kết qủa là đối x nhận được giá trị 3.5 và đối y nhận được giá trị 7.2 . Tiếp đó, máy sẽ thực hiện các câu lệnh trong thân hàm.

Khai báo

```
float z;
```

có tác dụng cung cấp một khoảng nhớ cho biến cục bộ z. Ba câu lệnh tiếp theo làm nhiệm vụ hoán vị hai đối x và y. Kết qủa là đối x nhận giá trị 7.2 còn đối y nhận giá trị 3.5. Sau đó là dấu } cuối cùng của hàm hoan_vi. Khi gặp dấu này máy trở lại hàm main() và thực hiện câu lệnh printf. Câu lệnh này cho hiện giá trị của các biến x và y lên màn hình. Rõ ràng hàm hoan_vi không làm thay đổi giá trị của các biến x và y, nên trên màn hình sẽ xuất hiện dòng sau:

```
x = 3.50 y = 7.20
```

Sau đó chương trình kết thúc.

Như vậy hàm hoan_vi(x,y) đã viết ở trên không đáp ứng được yêu cầu đề ra. Vấn đề tồn tại này sẽ được giải quyết trong mục sau nhờ sử dụng một khái niệm mới rất quan trọng là con trỏ.

§3. CON TRỎ VÀ ĐỊA CHỈ

3.1. Địa chỉ. Liên quan đến một biến ta đã có khái niệm:

- Tên biến
- Kiểu biến
- Giá trị của biến

Ví dụ câu lệnh

```
float alpha = 30.5;
```

xác định một biến có tên là alpha có kiểu float và có giá trị 30.5. Ta cũng đã biết, theo khai báo trên, máy sẽ cấp phát cho biến alpha một khoảng nhớ gồm 4 byte liên tiếp. Địa chỉ của biến là số thứ tự của byte đầu tiên trong một dãy các byte liên tiếp mà máy dành cho biến (các byte được đánh số từ 0).

Một điều cần chú ý là mặc dù địa chỉ của biến là một số nguyên nhưng không được đánh đồng nó với các số nguyên thông thường dùng trong các phép tính.

Rõ ràng 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
```

cho ta địa chỉ của biến x.

3.2. Con trỏ. Con trỏ là một biến dùng để chứa địa chỉ. Vì có nhiều loại địa chỉ nên cũng có nhiều kiểu con trỏ tương ứng. Con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu int. Tương tự, ta có con trỏ kiểu float, kiểu double,... Cũng như đối với bất kỳ một biến nào khác, một con trỏ cần khai báo trước khi sử dụng. Việc khai báo biến con trỏ được thực hiện theo mẫu sau:

```
type *tên_con_trỏ;
```

Ví dụ câu lệnh

```
int x,y,*px,*c;
```

khai báo hai biến kiểu int x,y và hai con trỏ kiểu int là px và c.

Tương tự câu lệnh

```
float *t, *d;
```

khai báo hai con trỏ kiểu float t và d.

Khi đã có các khai báo trên thì các câu lệnh

```
c= &y;
px = &x;
```

hoàn toàn xác định. Câu lệnh thứ nhất sẽ gán địa chỉ của y cho con trỏ c và câu lệnh thứ hai sẽ gán địa chỉ của biến x cho con trỏ px. Như vậy trong con trỏ c chứa địa chỉ của biến y và trong con trỏ px chứa địa chỉ của biến x. Chú ý rằng nếu viết

```
t = &y;
```

thì sẽ nhận được một câu lệnh sai: t là con trỏ kiểu float, nó chỉ chứa được địa chỉ của các biến float. Câu lệnh trên nhằm gán địa chỉ của biến nguyên y cho con trỏ t là không thể chấp nhận được.

Chú ý: Khi con trỏ px chứa địa chỉ của biến x thì ta nói px trỏ tới x.

3.3. Qui tắc sử dụng con trỏ trong các biểu thức. Ta có thể sử dụng tên con trỏ hoặc dạng khai báo của nó trong các biểu thức. Ví dụ đối với con trỏ px, ta có thể sử dụng các cách viết: px (tên con trỏ) và *px (dạng khai báo của con trỏ).

+ Sử dụng tên con trỏ. Con trỏ cũng là một biến nên khi tên của nó xuất hiện trong một biểu thức thì giá trị của nó sẽ được sử dụng trong biểu thức này. Chỉ có một điều cần lưu ý ở đây: giá trị của một con trỏ là địa chỉ của một biến nào đó. Khi tên con trỏ đứng ở bên trái của một toán tử gán thì giá trị của biểu thức bên phải (giá trị này phải là địa chỉ) được gán cho con trỏ.

Ta hãy xem các câu lệnh sau làm gì ?

```
float a, *p, *q;
p = &a;
q = p;
```

câu lệnh thứ nhất khai báo một biến kiểu float (biến a) và hai con trỏ p và q kiểu float. Câu lệnh thứ hai sẽ gán địa chỉ của biến a cho con trỏ p và câu lệnh thứ ba sẽ gán giá trị của p cho q. Két qủa là con trỏ q chứa địa chỉ của biến a.

Cũng giống như các biến khác, nội dung của con trỏ có thể thay đổi. Ta có thể sử dụng qui tắc này để biến đổi địa chỉ. Chẳng hạn, nếu con trỏ p chứa địa chỉ của phần tử mảng a[i] , thì sau khi thực hiện phép toán ++p nó sẽ chứa địa chỉ của phần tử a[i+1]. Vấn đề này sẽ còn được thảo luận trong các mục sau.

+ Sử dụng dạng khai báo của con trỏ. Như đã biết sau khi thực hiện các câu lệnh

```
float x, y, z,*px,*py;
px = &x; py = &y;
```

thì px trỏ tới x, py trỏ tới y. Bây giờ ta có thể bàn đến ý nghĩa của các cách viết

*px và *py

Điều này được phát biểu trong một nguyên lý rất ngắn gọn như sau: Nếu con trỏ px trỏ tới biến x thì các cách viết

x và *px

là tương đương trong mọi ngữ cảnh.

Theo nguyên lý này thì ba câu lệnh sau đều có hiệu lực như nhau

```
y = 3*x + z;
*py = 3*x + z;
*py = 3*(*px) + z;
```

Từ đây có thể rút ra một kết luận quan trong là: khi biết được địa chỉ của một biến thì chẳng những chúng ta có thể sử dụng giá trị của nó mà còn có thể gán cho nó một giá trị mới (làm thay đổi nội dung của nó). Điều này sẽ được áp dụng như một phương pháp chủ yếu để nhận kết quả của hàm thông qua đối.

3.4. Qui tắc về kiểu giá trị trong khai báo. Trong điểm 3 đã nêu hai cách sử dụng con trỏ: dùng tên và dùng dạng khai báo của nó. Như đã nói ở trên, việc sử dụng tên con trỏ cũng tuân theo các qui định chung về việc sử dụng tên các biến. Điều ta muốn nói ở đây là: qui tắc sử dụng dạng khai báo của con trỏ nêu trong điểm 3 nằm trong một qui tắc chung về khai báo. Ta sẽ trình bày qui tắc này thông qua một ví dụ cụ thể. Giả sử ta có khai báo

```
float a, b[7], *px;
```

khi đó mọi thành phần của nó đều cho giá trị kiểu float. Nói cách khác, khi:

a, b[i], *px

xuất hiện trong một biểu thức thì chúng luôn luôn cho một giá trị kiểu float. Qui tắc này có thể phát biểu một cách tổng quát hơn như sau: mọi thành phần của cùng một khai báo (biến, phần tử mảng, hàm, con trỏ) khi xuất hiện trong biểu thức đều cho cùng một kiểu giá trị.

3.5. Hàm có đối con trỏ. Điều đầu tiên cần ghi nhớ ở đây là: Nêu đối của hàm là con trỏ kiểu int (float, double,...) thì tham số thực tương ứng phải là địa chỉ của biến hoặc địa chỉ của phần tử mảng kiểu int (float, double,...). Khi đó địa chỉ của biến được truyền cho đối con trỏ tương ứng. Do đã biết địa chỉ của biến, nên ta có thể gán cho nó các giá trị mới bằng cách sử dụng các câu lệnh thích hợp trong thân hàm. Bây giờ bằng cách dùng đối con trỏ ta có thể xây dựng hàm hoán vị hai biến kiểu float như sau.

```
void hoan_vi (float *px, float *py) /* ví dụ đúng */
   {
      float z;
      z = *px;
      *px = *py;
      *py = z;
   }
#include "stdio.h"
main()
   {
      float a = 7.6,b = 13.5;
      hoan_vi(&a, &b);
      printf("\na = %0.2f b = %0.2f",a,b);
   }
```

Kết qủa thực hiện chương trình

a = 13.50 b = 7.60

Ta hãy xem hàm hoán vị làm việc thế nào. Như đã biết, chương trình bắt đầu từ câu lệnh đầu tiên trong hàm main(). Kết qủa là biến a nhận giá trị 7.6 và biến b nhận giá trị 13.5. Tiếp đó là lời gọi hàm hoan_vi. Máy sẽ gán giá trị của các tham số thực cho đối tương ứng. Như vậy địa chỉ của a được gán cho con trỏ px , địa chỉ của b được gán cho con trỏ py. Sau đó máy lần lượt xét đến các câu lệnh trong thân hàm. Câu lệnh thứ nhất sẽ cấp phát cho biến cục bộ z một khoảng nhớ 4 byte. Theo qui tắc về sử dụng con trỏ nêu trong điểm 3 thì 3 câu lệnh tiếp theo tương đương với các câu lệnh:

```
z = a;
a = b;
b = z;
```

Như vậy a sẽ nhận giá trị của b và ngược lại . Tiếp đó, máy trở về hàm main() và in ra những dòng kết qủa như đã chỉ ra ở trên.

3.6. Khi nào sử dụng đối con trỏ. Trong số các đối của hàm, ta có thể chia ra làm hai loại. Loại thứ nhất gồm các đối dùng để chứa các giá trị đã biết, ta gọi chúng là các đối vào. Loại thứ hai gồm các đối dùng để chứa các kết qủa mới nhận được, gọi là các đối ra. Ví dụ cần lập một hàm giải phương trình bậc hai ax2 + bx + c = 0. Đối với hàm này thì a, b, c là các đối vào, còn các nghiệm x1, x2 là các đối ra. Ngoài ra có thể thiết kế để hàm nhận giá trị bằng:

0 khi a = 0

1 khi a khác 0 và delta >= 0

-1 khi a khác 0 và delta < 0

Ta trở lại với câu hỏi khi nào sử dụng đối con trỏ? Câu trả lời như sau: các đối ra phải là con trỏ. Bảng sau đây chỉ ra mối quan hệ giữa tham số thực và đối tương ứng.

Tham số thực	Đối tương ứng
Giá trị kiểu int	Biến kiểu int
(float, double)	(float, double)
Địa chỉ kiểu int	Con trỏ kiểu int
(float, double)	(float, double)

Bây giờ việc xây dựng hàm giải phương trình bậc hai đã trở nên dễ dàng. Chương trình dưới đây sẽ minh hoạ các điều nói trên.

```
#include"stdio.h"
#include "math.h"
int ptb2(float a, float b, float c, float *x1, float *x2);
main()
  {
     int s,ch;
     float a,b,c,x1,x2;
     printf("\n Vao a,b,c");
     scanf("%f%f%f",&a,&b,&c);
     s = ptb2(a,b,c,&x1,&x2);
     if (s == 0)
        printf("\n a = 0 ");
     else if (s == -1)
        printf("\n delta < 0 ");
     else
        printf("\n x1 = %0.2f x2 = %0.2f",x1,x2 );
  }
/* Hàm giải phương trình bậc hai */
int ptb2(float a, float b, float c, float *x1, float *x2)
   {
     float delta;
     if (a == 0) return 0;
     delta = b*b-4*a*c;
```

```
    if (delta<0) return -1;
    *x1 = (-b - sqrt(delta))/(2*a);
    *x2 = (-b + sqrt(delta))/(2*a);
    return (1);
}
```

§4. CON TRỎ VÀ MẢNG MỘT CHIỀU

Trong C có mối quan hệ chặt chẽ giữa con trỏ và mảng: các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ. Trong mục này sẽ trình bầy các vấn đề có liên quan đến mảng và con trỏ.

4.1. Phép toán lấy địa chỉ áp dụng được cho các phần tử của mảng một chiều. Giả sử ta có khai báo

```
double b[20];
```

khi đó phép toán

```
&b[i]
```

với i trong khoảng [0,19] cho địa chỉ của phần tử b[i].

4.2. Tên mảng là một hằng địa chỉ. Như đã biết với khai báo

```
float a[10];
```

máy sẽ bố trí cho mảng a mười khoảng nhớ liên tiếp (mỗi khoảng nhớ 4 byte). Như vậy nếu biết địa chỉ của một phần tử nào đó của mảng a, thì dễ dàng suy ra địa chỉ của các phần tử khác. Một sự kiện mà ta định nói ở đây là: tên mảng là một hằng địa chỉ, nó chính là địa chỉ của phần tử đầu tiên của mảng. Như vậy, trong mọi ngữ cảnh:

a tương đương với &a[0]

a+i tương đương với &a[i]

*(a+i) tương đương với a[i]

4.3. Nếu con trỏ pa trỏ tới một phần tử a[k] nào đó thì:

pa+i trỏ tới phần tử thứ i sau a[k], tức là a[k+i]

pa-i trỏ tới phần tử thứ i trước a[k], tức là a[k-i]

*(pa+i) tương đương với pa[i]

Như vậy sau hai câu lệnh

```
float a[30], *p;
p = a;
```

thì bốn cách viết sau có tác dụng như nhau:

```
a[i]   *(a+i)   *(p+i)   p[i]
```

Bây giờ ta đưa ra một ví dụ minh họa các điều nêu trên.

Xét một bài toán đơn giản: vào từ bàn phím số liệu của các phần tử của một mảng và tính tổng của chúng. Dưới đây là bốn chương trình giải bài toán nêu trên.

```
/* vao so lieu cho mang va tinh tong, Ban 1 */
        #include "stdio.h"
        main()
          {
            float a[4],s;
            int i;
            for (i=0;i<4; ++i )
              {
                printf("\n a[%d] = ",i);
                scanf("%f",&a[i]);
              }
            s = 0;
            for (i=0; i<4; ++i )
            s += a[i];
            printf("\n tong = %8.2f",s);
          }
```

```
/* vao so lieu cho mang va tinh tong, Ban 2 */
        #include "stdio.h"
        main()
          {
            float a[4],s;
            int i;
            for (i=0; i<4; ++i )
            {
              printf("\n a[%d] = ",i);
              scanf("%f", a+i );
            }
            s = 0;
            for(i=0; i<4; ++i )
            s += a[i];
            printf("\n tong = %8.2f",s);
          }
```

/* vao so lieu cho mang va tinh tong , **Ban 3** */

```
#include "stdio.h"
main()
  {
    float a[4],s,*pa;
    int i;
    pa = a;
    for (i = 0; i<4; ++i )
      {
        printf("\n a[%d] = ",i);
        scanf("%f",&pa[i] );
      }
    s = 0;
    for(i=0; i<4; ++i )
    s += pa[i];
    printf("\n tong = %8.2f",s);
  }
```

/* vao so lieu cho mang va tinh tong , **Ban 4** */

```
#include "stdio.h"
main()
  {
    float a[4], s, *pa;
    int i;
    pa = a;
    for (i=0; i<4; ++i)
      {
        printf("\n a[%d] = ",i );
        scanf("%f", pa+i );
      }
    s = 0;
    for (i=0; i<4; ++i )
          s += *(pa+i);
    printf("\n tong = %8.2f",s);
  }
```

4.4. Nếu tham số thực là tên mảng a (một chiều) kiểu int (float double,...) thì đối pa tương ứng cần phải là một con trỏ kiểu int (float, double,...). Đối pa có thể khai báo kiểu con trỏ:

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

hoặc cũng có thể khai báo như một mảng hình thức:

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

Chú ý: Hai cách khai báo trên là tương đương.

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

*(pa+i) và pa[i]

Giả sử ta cần xây dựng một hàm để tính tổng các phần tử của một mảng kiểu double. Rõ ràng khi gọi tới hàm này cần sử dụng hai tham số thực: tên mảng chứa dãy và độ dài của dãy số. Do đó, hàm cần có hai đối: một đối là con trỏ kiểu double , một đối là biến kiểu int. Dưới đây là sáu cách viết khác nhau cho hàm này.

```
/*
chuong trinh su dung 6 ham
tinh tong cua day so
*/
        #include "stdio.h"
        /* Các nguyên mẫu của hàm */
            double sum1(double *a,int n);
            double sum2(double *a,int n);
            double sum3(double *a,int n);
            double sum4(double a[],int n);
            double sum5(double a[],int n);
            double sum6(double a[],int n);
```

```
main()
  {
    double b[4];
    b[0] = 465.2;
    b[1] = 1256.7;
    b[2] = 2.35e3;
    b[3] = 45e-2;
    printf("\n sum1 = %5.2 sum2 = %5.2",sum1(b,4),sum2(b,4));
    printf("\n sum3 = %5.2 sum4 = %5.2",sum3(b,4), sum4(b,4));
    printf("\n sum5 = %5.2 sum6 = %5.2",sum5(b,4),sum6(b,4));
  }
double sum1(double *a,int n) /* ban 1 */
  {
    double s = 0;
    int i;
    for (i=0; i<n; ++i)
          s += *(a+i);
    return s;
  }
double sum2(double *a,int n) /* ban 2 */
  {
    double s = 0;
    int i;
    for (i=0; i<n; ++i)
          s += a[i];
    return s;
  }
double sum3(double *a,int n)    /* ban 3 */
  {
    double s = 0;
    while (n--)
            s += *a++;
    return s;
  }
```

```
double sum4(double a[],int n) /* ban 4 */
  {
     double s = 0;
     int i;
     for (i=0; i<n; ++i)
          s += *(a+i);
     return s;
  }
double sum5(double a[],int n) /* ban 5 */
  {
     double s = 0;
     int i;
     for (i=0; i<n; ++i)
          s += a[i];
     return s;
  }
double sum6(double a[],int n) /* ban 6 */
  {
     double s = 0;
     while (n--)
          s += *a++;
     return s;
  }
```

4.5. Mảng, con trỏ và xâu ký tự . Như đã biết xâu kí tự là một dãy ký tự đặt trong hai dấu nháy kép, ví dụ

"Pham van Anh"

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

Điều muốn nói ở đây là: cũng giống như tên mảng, xâu kí tự là một hằng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo mes như một con trỏ kiểu char:

```
char *mes;
```

thì phép gán

```
mes = "Pham van Anh";
```

hoàn toàn có nghĩa . Sau khi thực hiện câu lệnh này trong con trỏ mes sẽ có địa chỉ đầu của mảng (kiểu char) đang chứa xâu kí tự bên phải. Khi đó các câu lệnh

```
puts("Pham van Anh");
```

và

```
puts(mes);
```

sẽ có cùng tác dụng là cho hiện lên màn hình dòng chữ

```
Pham van Anh
```

Mảng kiểu char thường dùng để chứa một dãy kí tự đọc vào bộ nhớ. Ví dụ , để nạp từ bàn phím tên của một người ta cần dùng một mảng kiểu char với độ dài 25. Các câu lệnh sau cho phép thực hiện điều này:

```
char t[25];
printf("\n Ho ten: ");
gets(t);
```

Bây giờ ta xem giữa mảng kiểu char và con trỏ kiểu char có những gì giống nhau và khác nhau. Để thấy được sự khác nhau của chúng ta đưa ra sự so sánh sau. Nếu các câu lệnh

```
char *mes, t[25];
mes = "Pham van anh";
gets(t);
```

là có nghĩa (đã giải thích ở trên), thì các câu lệnh

```
char *mes,t[15];
t = "Pham van anh";
gets(mes);
```

là không hợp lệ. Câu lệnh thứ hai sai ở chỗ: t là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác. Câu lệnh thứ ba đúng về ngữ pháp nhưng nó không thể thực hiện được. Thực vậy, mục đích của câu lệnh này là đọc từ bàn phím một dãy kí tự và lưu vào một vùng nhớ được con trỏ mes trỏ tới. Thế nhưng nội dung của con trỏ mes còn chưa xác định. Nếu mes đã trỏ tới một vùng nhớ nào đó thì câu lệnh này hoàn toàn có nghĩa. Chẳng hạn, sau khi thực hiện câu lệnh

```
mes = t;
```

thì các cách viết

```
gets(t); và gets(mes);
```

có tác dụng như nhau.

Việc xử lý mảng nhiều chiều phức tạp hơn so với mảng một chiều. Không phải mọi qui tắc đúng với mảng một chiều đều có thể đem ra áp dụng đối với mảng nhiều chiều. Dưới đây sẽ trình bầy cách sử dụng mảng nhiều chiều.

5.1. Phép toán lấy địa chỉ không dùng được đối với các phần tử của mảng nhiều chiều. Nói cách khác, trong nhiều trường hợp câu lệnh

&a[i][j]

là không hợp lệ và gây ra lỗi. (Đối với mảng hai chiều nguyên có thể dùng phép &a[i][j]).

Như vậy, chương trình sau đây với ý định vào số liệu cho một ma trận sẽ không làm việc được.

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

5.2. Phép cộng địa chỉ trong mảng hai chiều.

Như đã biết, mảng hai chiều a khai báo trong chương trình trên gồm sáu phần tử ứng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự sau (thứ tự hàng)

Phần tử	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
địa chỉ	1	2	3	4	5	6

Ta cũng đã biết tên mảng a biểu thị địa chỉ đầu tiên của mảng. Nhưng phép cộng địa chỉ ở đây phải hiểu như sau.

+ C quan niệm mảng hai chiều là mảng (một chiều) của mảng.

Như vậy với khai báo

float a[2][3];

thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của bảng). Vì vậy:

a   trỏ tới đầu hàng thứ nhất (phần tử a[0][0]),

a+1   trỏ tới đầu hàng thứ hai (phần tử a[1][0]).

...

5.3. Con trỏ và mảng hai chiều.

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta vẫn có thể dùng con trỏ theo cách sau:

```
float *pa, a[2][3];
/* Chú ý lệnh này */
pa = (float*) a;
```

Khi đó:

```
pa      trỏ tới a[0][0]
pa+1    trỏ tới a[0][1]
pa+2    trỏ tới a[0][2]
pa+3    trỏ tới a[1][0]
pa+4    trỏ tới a[1][1]
pa+5    trỏ tới a[1][2]
```

Chú ý: Phép gán

```
pa = a;
```

làm cho C băn khoăn, vì kiểu địa chỉ của pa và a khác nhau. pa là con trỏ float, còn a là địa chỉ kiểu float[3]. Khi dịch C sẽ đưa ra lời cảnh báo:

Suspicious pointer conversion in function main

Tuy vậy câu lệnh trên vẫn làm việc tốt.

Chương trình dưới đây minh hoạ cách dùng con trỏ để vào số liệu cho mảng 2 chiều.

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

Có thể cải tiến chương trình trên bằng cách đặt phép ép kiểu địa chỉ ngay trong hàm scanf như sau (không cần dùng con trỏ pa).

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

Trên ý tưởng này ta có thể viết chương trình vào số liệu cho ma trận thực cấp mxn như sau.

```
#include "stdio.h"
main()
  {
    float a[50][50];
    int m,n,i,j;
    printf("\n Vao m va n: ");
        scanf("%d%d",&m,&n);
    for (i=1; i<=m; ++i)
     for(j=1; j<=n; ++j)
     {
       printf("\n a[%d][%d]= ",i,j);
       scanf("%f", (float*)a + i*50 + j);
     }
  }
```

5.4. Sử dụng biến trung gian.

Ở trên trình bầy cách dùng con trỏ để vào số liệu cho mảng nhiều chiều. Một cách khác đơn giản hơn là dùng biến trung gian như sau:

Đọc một giá trị và chứa tạm vào một biến trung gian, sau đó ta gán biến này cho phần tử mảng.

Dưới đây là chương trình vào số liệu cho các ma trận a và b, sau đó tính tích của chúng. Kết qủa in ra theo thứ tự:

ma trận a

ma trận b

ma trận tích c

Các ma trận in ra dưới dạng bảng.

```
#include "stdio.h"
main()
   {
      float a[3][2],b[2][4],c[3][4],x;
      int i,j,k;
      /* vao ma tran a */
      for(i = 0; i < 3; ++i)
       for(j = 0; j < 2; ++j)
       {
         printf("\n a[%d][%d] = ",i,j );
         scanf("%f",&x);
         a[i][j] = x;
       }
   /* vao ma tran b */
   for(i=0;i<2;++i)
    for(j=0;j<4;++j)
    {
      printf("\n b[%d][%d]= ",i,j);
      scanf ("%f", &x);
      b[i][j] = x;
    }
  /* tinh c=a*b */
  for (i=0;i<3;++i)
   for (j=0;j<4;++j)
    {
      c[i][j] = 0;
      for( k=0; k<2; ++k )
      c[i][j] += a[i][k]*b[k][j];
    }
     /* in ma tran a */
 printf("\n MA TRAN A \n");
 for(i=0; i<3; ++i )
   for(j=0; j<2; ++j )
      printf("%8.2f%c",a[i][j],j==1?'\n':' ');
```

```
/* in ma tran b */
printf("\n MA TRAN B \n");
for(i=0; i<2; ++i )
  for(j=0; j<4; ++j )
    printf("%8.2f%c",b[i][j],j==3?'\n':' ');
  /* in ma tran tich */
printf("\n MA TRAN C = A*B \n");
  for(i=0;i<3;++i)
    for(j=0; j<4; ++j)
      printf("%8.2f%c",c[i][j],j==3?'\n':' ');
}
```

5.5. Tham số thực là tên mảng nhiều chiều.

Giả sử a là mảng hai chiều:

float a[60][50];

Vấn đề đặt ra ở đây là: Làm thế nào để có thể dùng tên mảng hai chiều a trong lời gọi hàm. Có hai cách:

Cách 1.

+ Dùng đối con trỏ kiểu float[50], khai báo theo một trong hai mẫu sau:

float (*pa)[50];

float pa[][50];

+ **Chú ý:** Mẫu 2 chỉ dùng để khai báo đối,

Mẫu 1 dùng để khai báo con trỏ kiểu float[50].

+ Trong thân hàm, để truy nhập đến phần tử a[i][j] ta dùng:

pa[i][j]

+ **Chú ý:** Theo cách này, hàm chỉ dùng được đối với các mảng hai chiều có 50 cột (số hàng không quan trọng).

Cách 2.

+ Dùng 2 đối:

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

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

+ Trong thân hàm, để truy nhập đến phần tử a[i][j] ta dùng công thức:

*(pa + i*N + j)

+ **Chú ý:** Theo cách 2, mảng hai chiều được quy về mảng một chiều. Việc sử lý trong thân hàm sẽ phức tạp hơn so với cách thứ nhất, nhưng vì không cố định bao nhiêu cột nên có thể dùng hàm cho bất kỳ mảng hai chiều nào.

Các ví dụ dưới đây sẽ minh hoạ hai cách nói trên.

Ví dụ 1: Chương trình dưới đây gồm hàm tính các tổng hàng của ma trận thực cấp mxn thiết kế theo cách 1. Hàm này sẽ được dùng trong hàm main.

```
#include "stdio.h"
void tong_h(float a[][50], int m, int n, int *th);
/* th trỏ đến vùng nhớ chứa các tổng hàng */
         main()
           {
             float b[30][50]; /* b phải có 50 cột */
             float h[30]; /* chứa các tổng hàng */
             float x;  int i,j,m,n;
             /* Vào số liệu */
             printf("\n Vao m va n: ");
                scanf("%d%d",&m,&n);
             for (i=1; i<=m; ++i)
              for(j=1; j<=n; ++j)
                {
                 printf("\n b[%d][%d]= ",i,j);
                 scanf("%f",&x); b[i][j] = x;
                }
          tong_h(b, m, n, h);     /* Tính các tổng hàng */
             /* In các tổng hàng */
          for (i=1; i<=m; ++i)
            printf("\n Tong hang %d = %0.2f",i,h[i]);
      }
void tong_h(float a[][50], int m, int n, int *th)
   {
      int i,j;
      for(i=1;i<=m;++i)
      {
        th[i]=0;
        for(j=1;j<=n;++j)
            th[i] += a[i][j];
      }
   }
```

Ví dụ 2:

Dưới đây là một chương trình gồm bốn hàm làm việc trên các ma trận vuông (số hàng bằng số cột). Hàm vao_mt dùng để nhập số liệu từ bàn phím cho một ma trận. Hàm ra_mt dùng để in một ma trận ra màn hình. Hàm nhan_mt dùng để nhân hai ma trận. Hàm cong_mt dùng để cộng hai ma trận. Hàm main sẽ sử dụng bốn hàm trên để nạp số liệu cho các ma trận a và b, tính c = a*b và d = a+b, sau đó đưa ra các ma trận a,b,c và d ra màn hình. Cả bốn hàm trên được viết theo cách 2. Đối N là số cột của mảng hai chiều, đối m là cấp của ma trận.

```
#include "stdio.h"
void vao_mt(float *a, int N, int m);
void ra_mt(float *a, int N, int m);
void nhan_mt(float *a, float *b, float *c, int N, int m);
void cong_mt(float *a, float *b, float *c, int N, int m);
main()
  {
  float a[20][20],b[20][20],c[20][20],d[20][20];
    vao_mt (a,20,5);
    vao_mt (b,20,5);
    nhan_mt(a,b,c,20,5);
    cong_mt(a,b,d,20,5);
    printf("\n Ma tran A\n ");
    ra_mt(a,20,5);
    printf("\n Ma tran B\n ");
    ra_mt(b,20,5);
    printf("\n Ma tran C\n "); ra_mt(c,20,5);
    printf("\n Ma tran D\n "); ra_mt(d,20,5);
  }
void vao_mt(float *a, int N, int m)
  {
    int i,j;
    for(i=1; i<=m; ++i)
     for(j=1; j<=m; ++j)
      {
       printf("\nphan tu (%d,%d) = ",i,j);
       scanf("%f", a + i*N +j);
      }
  }
```

```
void ra_mt(float *a, int N, int m)
  {
    int i,j;
    for(i=1; i<=m; ++i)
    {
      printf("\n");
      for(j=1; j<=m; ++j)
        printf("%8.2f", *(a + i*N + j) );
    }
  }
void nhan_mt(float *a, float *b, float *c, int N, int m)
  {
    int i,j,k;
    for(i=1; i<=m; ++i)
      for(j=1; j<=m; ++j)
       {
         *(c + i*N +j) = 0;
         for(k=1; k<=m; ++k)
           *(c+i*N +j) += (*(a+i*N+k))*(*(b+k*N+j));
       }
  }
void cong_mt(float *a, float *b, float *c, int N, int m)
  {
    int i,j;
    for(i=1; i<=m; ++i)
      for(j=1; j<=m; ++j)
          *(c+i*N +j) = *(a+i*N+j) + *(b+i*N+j);
  }
```

Nhận xét về chương trình: Khi dịch chương trình trên, trình biên sẽ phàn nàn tại các lời gọi hàm và đưa ra lời cảnh báo sau:

Suspicious pointer conversion in function main

Tại sao lại như vậy. Hãy xét lời gọi hàm

vao_mt (a,20,5);

Theo lệnh này, máy sẽ gán địa chỉ của a cho đối tương ứng.

Đối là con trỏ float, còn a là địa chỉ kiểu float[20]. Rõ ràng không có sự phù hợp về kiểu trong phép gán này.

Để làm cho C yên lòng có hai cách: hoặc dùng đối là con trỏ void (xem mục sau) hoặc dùng phép ép kiểu trong lời gọi như sau:

```
vao_mt ((float*)a,20,5);
```

Chú ý: Mặc dù có nhưng lời cảnh báo nhưng chương trình vẫn hoạt động đúng.

§6. KIỂU CON TRỎ, KIỂU ĐỊA CHỈ, CÁC PHÉP TOÁN TRÊN CON TRỎ

6.1. Kiểu con trỏ và kiểu địa chỉ.

Con trỏ dùng để lưu trữ địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán địa chỉ cho con trỏ chỉ diễn ra suôn sẻ khi kiểu địa chỉ phù hợp với kiểu con trỏ. Nếu không phù hợp, trình biên dịch C sẽ phàn nàn bằng lời cảnh báo:

Suspicious pointer conversion

Ví dụ theo khai báo:

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

thì

+ pa là con trỏ kiểu float,

+ pm là con trỏ kiểu float[30],

+ a là địa chỉ kiểu float[30].

Vì vậy phép gán

```
pa = a;
```

sẽ bị cảnh báo. Nhưng phép gán

```
pm = a;
```

thì hoàn toàn hợp lệ.

6.2. Các phép toán trên con trỏ.

Có 4 nhóm phép toán liên quan đến con trỏ và địa chỉ là: phép gán, phép tăng giảm địa chỉ, phép truy nhập bộ nhớ và so sánh. Kiểu con trỏ và kiểu địa chỉ có vaitrò quan trọng trong các phép toán nói trên.

+ Phép gán: Như đã nói ở trên, chỉ nên thực hiên phép gán các con trỏ cùng kiểu. Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu như ví dụ sau:

```
int x;
char *pc;
pc = (char*)(&x); /* ép kiểu */
```

+ **Phép tăng giảm địa chỉ.** Ta sẽ giải thích quy tắc tăng giảm địa chỉ qua các ví dụ.

Các câu lệnh

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

cho biết px là con trỏ float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ 4 byte (mỗi giá trị float chứa trong 4 byte), nên các phép tăng giảm địa chỉ được thực hiện trên 4 byte. Nói cách khác:

px + i trỏ tới phần tử x[10+i] px - i trỏ tới phần tử x[10-i]

Một ví dụ khác về phép tăng, giảm địa chỉ. Câu lệnh

float b[40][50];

xác định b là mảng gồm các dòng 50 phần tử thực. Kiểu địa chỉ của b là 50*4 = 200 byte. Do đó suy ra:

b	trỏ tới đầu dòng thứ nhất (phần tử b[0][0])
b+1	trỏ tới đầu dòng thứ hai (phần tử b[1][0])
b+5	trỏ tới đầu dòng thứ sáu (phần tử b[5][0])
...	

+ **Nguyên tắc truy nhập bộ nhớ như sau:** Con trỏ float truy nhập tới 4 byte, con trỏ int truy nhập 2 byte, con trỏ char truy nhập một byte. Ta minh hoạ điều này qua một ví dụ đơn giản. Giả sử đã có các khai báo:

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

Khi đó:

+ Nếu pf trỏ đến byte thứ 10001, thì *pf biểu thị vùng nhớ 4 byte liên tiếp từ byte 10001 đến byte 10004.

+ Nếu pi trỏ đến byte 10001 thì *pi biểu thị vùng nhớ 2 byte gồm các byte 10001 và 10002.

+ Nếu pc trỏ đến byte 10001 thì *pc biểu thị vùng nhớ 1 byte là byte 10001.

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

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

p1<p2 nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới,

p1=p2 nếu địa chỉ p1 trỏ tới bằng địa chỉ p2 trỏ tới.

p1>p2 nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

Dưới đây là đoạn chương trình tính tổng các số thực có dùng phép so sánh con trỏ:

```
float a[100], *p, *pcuoi, s=0.0;
int n;
pcuoi = a+n-1; /* Dịa chỉ cuối dẫy */
for (p=a; p<=pcuoi; ++p)
    s += *p;
```

6.3. Con trỏ kiểu void.

Con trỏ void được khai báo như sau:

```
void *tên_con_trỏ;
```

Đây là con trỏ đặc biệt (con trỏ không kiểu), nó có thể nhận bất kỳ địa chỉ kiểu nào. Chẳng hạn các câu lệnh sau là hợp lệ

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

+ **Chú ý:** Các phép tăng giảm địa chỉ, truy nhập bộ nhớ và so sánh không dùng trên con trỏ void.

+ **Cách dùng:** Con trỏ void thường dùng làm đối để nhận bất kỳ địa chỉ kiểu nào từ tham số thực. Trong thân hàm phải dùng phép ép kiểu để chuyển sang dạng địa chỉ cần xử lý. Ví dụ 2 dưới đây minh hoạ điều này.

6.4. Các ví dụ.

Ví dụ 1: Dùng con trỏ char dễ dàng tách các byte của một biến nguyên như sau. Giả sử đã có các lệnh:

```
unsigned int n = 0xABCD; /* Số nguyên hệ 16 */
char *pc;
pc = (char*)(&n);
```

Khi đó:

```
*pc = 0xAB (byte thứ nhất của n)
*(pc+1) = 0xCD (byte thư hai của n)
```

Ví dụ 2:

Ta đã nhận xét ví dụ 2 của §5 bị C cảnh báo khi biên dịch. Có thể khắc phục điều này bằng cách dùng đối con trỏ void trong các hàm. Chẳng hạn ta có thể sửa hàm cong_mt như sau:

void cong_mt(void *a, void *b, void *c, int N, int m);

```
void cong_mt(void *a, void *b, void *c, int N, int m)
{
    float *pa, *pb, *pc;
    int i,j;
    pa = (float*)a;
    pb = (float*)b;
    pc = (float*)c;
    for(i=1; i<=m; ++i)
      for(j=1; j<=m; ++j)
        *(pc+i*N +j) = *(pa+i*N+j) + *(pb+i*N+j);
}
```

Giải thích:

+ Vì đối là con trỏ void, nên nó có thể nhận được địa chỉ của các ma trận trong lời gọi hàm.

+ Song ta không thể sử dụng trực tiếp các đối con trỏ void trong thân hàm mà phải chuyển sang con trỏ float.

§7. MẢNG CON TRỎ

Mảng con trỏ là sự mở rộng khái niệm con trỏ. Mảng con trỏ là một mảng mà mỗi phần tử của nó có thể chứa được một địa chỉ nào đó. Cũng giống như con trỏ, mảng con trỏ có nhiều kiểu: mỗi phần tử của mảng con trỏ kiểu int sẽ chứa được các địa chỉ kiểu int. Tương tự, ta suy ra ý nghĩa của các mảng con trỏ kiểu char , float, ... Mảng con trỏ được khai báo theo mẫu

```
type *namearr[N];
```

trong đó type có thể là int, float, double, char, ..., namearr là tên của mảng, N là một hằng số nguyên xác định độ lớn của mảng.

Khi gặp khai báo trên máy sẽ cấp phát N khoảng nhớ liên tiếp cho N phần tử của mảng namearr. Ví dụ, câu lệnh

```
double *pa[100];
```

khai báo một mảng con trỏ kiểu double gồm một trăm phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double. Cũng như đối với các mảng khác , ta có thể khởi đầu cho các mảng con trỏ ngoài và tĩnh theo cách như đã trình bày ở chương 2.

Chú ý rằng bản thân mảng con trỏ không thể dùng để lưu trữ số liệu. Tuy nhiên mảng con trỏ cho phép sử dụng các mảng khác để lưu trữ số liệu một cách có hiệu qủa hơn theo cách: chia mảng thành các phần và ghi nhớ địa chỉ đầu của mỗi phần vào một phần tử của mảng con trỏ (xem ví dụ 2 dưới đây).

Cũng nên lưu ý rằng trước khi sử dụng một mảng con trỏ cần gán cho mỗi phần tử của nó một giá trị. 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.

Bây giờ ta đưa ra vài ví dụ về việc sử dụng mảng con trỏ.

Ví dụ 1:

Trong thực tế thường gặp bài toán tìm một đối tượng khi biết mã số của nó . Sau đây là một bài toán đơn giản loại này: Một tổ có 10 người, mã của mỗi người chính là số thứ tự. Ta cần lập một hàm để khi biết mã số của một tổ viên thì xác định được họ và tên của tổ viên đó. Danh sách của tổ được biểu thị bởi một mảng con trỏ tĩnh kiểu char. Họ tên các tổ viên được đưa vào theo cơ chế khởi đầu của mảng static. Chương trình gồm hàm tim_ng và hàm main. Hàm tim_ng có tác dụng tìm kiếm và in họ tên của một tổ viên theo mã của người đó. Hàm main sử dụng hàm tim_ng để in ra họ tên của những người ứng với các mã cụ thể. Dưới đây là chương trình cho bài toán này.

```
/* Chuong trinh tim nguoi khi biet ma */
/* Minh hoa cach khoi dau mang con tro static */
        #include <stdio.h>
        #include <ctype.h>
        void tim_ng(int ma);
        main()
         {
          int i;
          tt: printf("\n Can tim nguoi thu: ");
             scanf("%d",&i);
             tim_ng(i);
             printf("\n Co tiep tuc khong? - C/K");
             if (toupper(getch()) = 'C')
                goto tt;
          }
        void tim_ng(int ma)
           {
```

```
static char *ds[] = {
                     " Ma sai",
                     "Pham thu Huong",
                     "Pham thi Lan",
                     "Nong thi Thu",
                     "Nguyen tien Trong",
                     "Do thi Be",
                     "Le kim Long",
                     "Nguyen thanh Hung ",
                     "Tran van Thong",
                     "Pham quoc Khanh",
                     "Vu thi Binh"
                     };
printf("\n\n Ma %d", ma);
printf(": %s", (ma<1 || ma>10)?ds[0]:ds[ma]);
}
```

Ví dụ 2:

Trong ví dụ 2 ta xét bài toán tính khối lượng vận chuyển (tấn-km) cho một tổ lái xe gồm m người trong một ngày . Thông tin nhận được từ giấy đi đường của mỗi lái xe gồm quãng đường (km) và trọng tải (tấn) cho mỗi chuyến đi. Nếu một lái xe thực hiện ba chuyến đi thì số liệu của người đó gồm ba cặp số, chẳng hạn

25 km 4.5 tấn
50 km 5.2 tấn
30 km 3.6 tấn

Do số chuyến của mỗi lái xe khác nhau nên độ dài mảng số liệu của các lái xe nói chung là khác nhau. Số liệu của bài toán được tổ chức như sau:

Dùng mảng k kiểu int để lưu trữ số liệu về quãng đường và mảng t kiểu float để lưu trữ số liệu về trọng tải cho toàn đội. Như vậy nếu số liệu của các lái xe là (m = 6)

người 1		người 2		người 3		người 4		người 5		người 6	
km	tấn	km	tấn	km	tấn	km	tấn	km	tấn	km	tấn
25	4.5	100	5.0	40	4.5	80	6.2	45	5.5	35	4.2
50	5.2			60	3.8					55	3.5
30	3.6										

thì các mảng k và t sẽ có nội dung như sau:

k	k+1	k+2	k+3	k+4	k+5	k+6	k+7	k+8	k+9
25	50	30	100	40	60	80	45	35	55

t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9
4.5	5.2	3.6	5.0	4.5	3.8	6.2	5.5	4.2	3.5

Trong mảng k, phần số liệu của người thứ nhất bắt đầu từ k[0], của người thứ hai bắt đầu từ k[3]...Tương tự , ta có các phần tử t[0],t[3],... trong mảng t. Để lưu trữ địa chỉ của các phần tử này ta dùng hai mảng con trỏ pk và pt. Trong ví dụ đang xét, nội dung của hai mảng này như sau:

Pk	pk+1	pk+2	pk+3	pk+4	pk+5
&k[0]	&k[3]	&k[4]	&k[6]	&k[7]	&k[8]

pt	pt+1	pt+2	pt+3	pt+4	pt+5
&t[0]	&t[3]	&t[4]	&t[6]	&t[7]	&t[8]

Chương trình gồm: hàm vaosl dùng để vào số liệu cho từng lái xe . Số liệu của mỗi lái xe bao gồm các cặp số quãng đường và trọng tải . Mỗi cặp số ghi trên một dòng . Phần số liệu của mỗi lái xe kết thúc bằng một cặp hai số không. Chẳng hạn, đối với một lái xe có hai chuyến hàng thì số liệu được nạp vào như sau

47 4.6 ↵

55 3.8 ↵

0 0 ↵

Hàm vaosl sẽ đọc số liệu từ bàn phím và xây dựng các mảng k,t,pk và pt theo số liệu thực tế nhận được . Các mảng này có thể sử dụng trong tất cả các hàm của chương trình vì chúng là các mảng ngoài.

Hàm rasln dùng để in số liệu của lái xe thứ i, ở đây i xem như một số nguyên cho trước trong khoảng từ 1 đến m.

Hàm klvcn dùng để tính khối lượng vận chuyển (tấn-km) của lái xe thứ i.

Hàm main sẽ dùng các hàm trên để vào số liệu và in ra khối lượng vận chuyển của từng lái xe trong tổ .

Để cơ động trong việc mở rộng hoặc thu hẹp các mảng k,t, pk, pt ta đã dùng toán tử #define để xác định kích cỡ của chúng. Dưới đây là bản chương trình.

```
/* Chuong trinh tinh khoi luong van chuyen cua lai xe */
        #include "stdio.h"
        #define N 1000/* so chuyen di trong 1 ngay khong qua 1000 */
```

```
#define L 20 /* so lai xe khong qua 20 */
int k[N];       /* km */
float t[N];     /* tan */
int *pk[L];     /*mang con tro pk */
float *pt[L];   /* mang con tro pt */
char *thongb = "\n khong co lai xe thu %d ";
void vaosl (int m) /* so lai xe la m */
  {
    int i, * pkk;
    float *ptt;
    pkk = pk[0] = k;
    ptt = pt[0] = t;
    for(i=1;i<=m;++i)
      {
        printf("\n so lieu cua lai xe %d ",i);
        doc: scanf("%d%f",pkk,ptt);
        if (*pkk!= 0)
        {
          ptt++;
          pkk++;
          goto doc;
        }
        pk[i] = pkk;
        pt[i] = ptt;
      }
  }
void rasln(int m,int i)
/* so lai xe thuc te la m, can in so lieu cua nguoi i, 1 <= i <= n */
  {
    int *pkk;
    float *ptt;
    if( i < 1 || i > m )
      {
        printf(thongb,i); return;
      }
```

```
    pkk = pk[ i-1 ];
    ptt = pt[i-1];
    printf("\n\n so lieu lai xe %d\n",i);
    while ( pkk < pk[i] )
      printf("\n %4d km %8.2f tan",*pkk++, *ptt++);
  }
 float klvcn(int m, int i)
     /* so lai xe la m */
     /* tinh khoi luong van chuyen cua lai xe i */
   {
     int *pkk;
     float *ptt, s = 0.0;
     if( i < 1 || i > m )
     {
       printf(thongb,i);
       return 0.0;
     }
     pkk = pk[i-1];
     ptt = pt[i-1];
     while ( pkk < pk[i])
     s += ( *pkk++)*( *ptt++);
     return s;
   }
main()
  {
    int m,i;
    printf("\n so lai xe = "); scanf("%d", &m);
    vaosl(m);
     /* in so lieu va khoi luong van chuyen của cả tổ */
    for( i = 1; i < = m; ++i )
     {
      rasln(m,i);
      printf("\n khoi luong van chuyen=%8.2f km/tan",
            klvcn(m,i));
     }
  }
```

§8. CON TRỎ TỚI HÀM

8.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm. Ta trình bầy quy tắc khai báo thông qua các ví dụ.

+ Tác dụng của câu lệnh:

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

là khai báo:

- f là con trỏ hàm kiểu float có đối float,
- mf là mảng con trỏ hàm kiểu float có đối int (mảng có 50 phần tử).

+ Tác dụng câu lệnh:

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

là khai báo:

- g là con trỏ hàm kiểu double có các đối int và double, - mg là mảng con trỏ hàm kiểu double có các đối double và float (mảng có 30 phần tử).

8.2. Tác dụng của con trỏ hàm.

Con trỏ hàm dùng để chứa địa chỉ của hàm. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm.

Các ví dụ sau minh hoạ điều này.

Ví dụ 1:

Vừa khai báo vừa gán.

```
#include <stdio.h>
double fmax(double x, double y) /* Hàm tính max */
  {
    return (x>y?x:y);
  }
/* Khai báo và gán tên hàm cho con trỏ hàm */
double (*pf)(double, double) = fmax;
/* Sử dụng con trỏ hàm */
main()
  {
    printf("\nmax= %f", pf(4.5, 78.9));
  }
```

Ví dụ 2:

Gán sau khi khai báo.

```
#include <stdio.h>
double fmax(double x, double y) /* Hàm tính max */
  {
    return (x>y?x:y);
  }
/* Khai báo con trỏ hàm */
double (*pf)(double, double);
main()
  {
      /* Gán tên hàm cho con trỏ hàm */
    pf = fmax;
      /* Sử dụng con trỏ hàm */
    printf("\nmax= %f",pf(4.5,78.9));
  }
```

8.3. Đối con trỏ hàm.

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

+ Cách dùng con trỏ hàm trong thân hàm. Điều này có thể diễn đạt như sau.

Nếu đối được khai báo:

double (*f)(double, int)

thì trong thân hàm ta có thể dùng các cách viết sau để xác định giá trị của hàm (do con trỏ f trỏ tới):

f(x,m) (f)(x,m) (*f)(x,m)

ở đây x là biến double, m là biến int.

8.4. Các ví dụ.

Ví dụ 1: Chương trình dưới đây sẽ:

- Lập hàm tính tích phân của f(x) trên đoạn [a,b] theo phương pháp hình thang bằng cách chia [a,b] thành 1000 khoảng có độ dài như nhau.

- Dùng hàm trên để tính:

S1 = Tích phân trên [0,PI/2] của sin(x)

S2 = Tích phân trên [0,PI/2] của cos(x)

S3 = Tích phân trên [0,1.0] của exp(x)

S4 = Tích phân trên [-1.2,3.5] của

g(x) = (exp(x)-2*sin(x*x))/(1+pow(x,4));

```
/* Chương trình tính tích phân */
        #include "stdio.h"
        #include "math.h"
        double tp(double (*f)(double), double a, double b);
        double g(double);
        double tp(double (*f)(double), double a, double b)
          {
            int i, n=1000;
            double s, h=(b-a)/n;
            s= (f(a)+f(b))/2;
            for(i=1; i<n; ++i) s += f(a+i*h);
            return h*s;
          }
        double g(double x)
          {
            double s;
            s = (exp(x)-2*sin(x*x))/(1+pow(x,4));
            return s;
          }
        main()
          {
            printf("\nS1= %f",tp(sin,0,M_PI/2));
            printf("\nS2= %f",tp(cos,0,M_PI/2));
            printf("\nS3= %f",tp(exp,0,1.0));
            printf("\nS4= %f",tp(g,-1.2,3.5));
          }
```

Ví dụ 2: Chương trình dưới đây sẽ:

- Xây dựng hàm sắp xếp dãy n đối tượng đặt trong vùng nhớ do con trỏ buf (kiểu void) trỏ tới. Độ dài của đối tượng là size byte. Tiêu chuẩn sắp xếp cho theo hàm ss.

- Dùng hàm trên để sắp xếp dãy số thực theo thứ tự tăng dần.

```
/*
Sap xep tong quat:
n doi tuong
chieu dai doi tuong la size
dia chi dau buf
theo tieu chuan ss la mot ham
*/
    #include "stdio.h"
    #include "math.h"
    #include "mem.h"
    #include "alloc.h"
    void sort(void *buf,int size,int n,int (*ss)(void*,void*));
      int tang(void *u,void *v);
      int tang(void *u,void *v)
        {
          return ( *((float*)u) <= *((float*)v));
        }
    void sort(void *buf,int size,int n,int (*ss)(void*,void*))
      {
      void *tg; char *p; int i,j;
        p=(char*)buf;
        tg = (char*)malloc(size);
        for(i=0;i<n-1;++i)
         for(j=i+1;j<n;++j)
          if(!ss(p+i*size,p+j*size))
          {
           memcpy(tg,p+i*size,size);
           memcpy(p+i*size,p+j*size,size);
           memcpy(p+j*size,tg,size);
          }
      }
   float x[]={20, 25,10, 5,15};
   main()
      {
```

```
int j;
sort(x,4,5,tang);
clrscr();
for(j=0; j<5; ++j)
printf("%10.2f",x[j]);
}
```

Ví dụ 3:

Chương trình dưới đây minh hoạ cách dùng mảng con trỏ hàm để lập bảng giá trị cho các hàm: x*x, sin(x), cos(x), exp(x) và sqrt(x). Biến x chạy từ 1.0 đến 10.0 theo bước 0.5.

```
#include "stdio.h"
#include "math.h"
double bp(double x) /* Hàm tính x*x */
  {
    return x*x;
  }
main()
  {
  int j; double x=1.0;
  typedef double(*ham)(double); /* Khai bao mang con tro ham */
    ham f[6];
      /* Có thể khai báo mảng con trỏ hàm như sau:
           double (*f[6])(double);  */
      /* Gán tên hàm cho các phần tử mảng con trỏ hàm */
    f[1]=bp;
    f[2]=sin;
    f[3]=cos;
    f[4]=exp;
    f[5]=sqrt;
          /* Lập bảng giá trị */
    while(x<=10.0)
    {
     printf("\n");
     for(j=1; j<=5; ++j)
      printf("%10.2f",f[j](x));  x+=0.5;
    }
  }
```

Ví dụ 4: Minh hoạ cách dùng con trỏ hàm để thực hiện các hàm của DOS. Dùng một con trỏ hàm trỏ tới hàm khởi động lại (Restart) của DOS đặt tại địa chỉ phân đoạn 0xFFFF : 0x0000. Sau đó dùng chính con trỏ hàm này để gọi hàm Restart của DOS. Chương trình dưới đây minh hoạ điều này. Chương trình yêu cầu nhập mật khẩu, nếu nhập sai sẽ cho máy tính khởi động lại.

```
/* Chương trình kiểm tra mật khẩu */
        #include <stdio.h>
        #include <conio.h>
        #include <dos.h>
        #include <ctype.h>
        char mk[]="ABC";
        void far (*f)(void);
        main()
          {
            char ch; int i;
            int dung = 1;
            clrscr();
            printf("\n Vao mat khau (3 ky tu): ");
            i=0;
            while(i<3)
              {
                ch=getch();
                if(toupper(ch)!=mk[i])
                  {
                   dung=0;  break;
                  }
                   ++i;
                 }
                 if(!dung)
                   {
                     printf("\n Sai Mat khau");   getch();
                     f = MK_FP(0xFFFF,0x0000);  f();
                   }
               else
             printf("\n Dung Mat khau");  getch();
          }
```

9.1. Khái niêm chung về đệ quy.

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

Ta cũng chú ý rằng: 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, nghĩa là: lần ra đầu tiên ứng với lần vào cuối cùng và lần ra khỏi hàm cuối cùng ứng với lần đầu tiên gọi tới hàm.

Để minh họa những điều nói trên, ta đưa ra một ví dụ đơn giản. Giả sử ta cần viết một chương trình tính n giai thừa. Khi không dùng phương pháp đệ quy thì hàm này có thể viết như sau:

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

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

n!= 1 nếu n = 0

n!= n*(n-1)! nếu n > 0

Dựa vào công thức trên ta có thể xây dựng hàm để tính n! một cách đệ quy như sau

```
/* Ham tinh n! theo de quy *
        long gtdq(int n)
          {
            if (n==0||n==1)
            return 1;
            else
              return (n*gtdq(n-1));
          }
```

Ta hãy giải thích cơ chế hoạt động của hàm gtdq khi nó được gọi từ hàm main() dưới đây:

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

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

n * gtdq(n-1)

Để tính biểu thức trên cần gọi tới chính hàm gtdq. Lần gọi thứ hai được thực hiện. Máy sẽ tạo ra đối n mới, ta gọi nó là n thứ hai. Giá trị của n-1 ở đây vẫn hiểu là n thứ nhất, được truyền cho n thứ hai. Như vậy , n thứ hai có giá trị 2. Bây giờ trong thân hàm n được hiểu là n thứ hai. Điều kiện n==1 || n==0 vẫn chưa được thỏa mãn nên máy lại tính biểu thức

n * gtdq(n-1)

Cần phải gọi tới hàm gtdq lần thứ ba. Máy sẽ tạo ra đối n mới , ta gọi nó là n thứ ba. Giá trị n-1, ở đây n hiểu là n thứ hai được truyền cho n thứ ba. Vậy n thứ ba có giá trị 1. Trong thân hàm n được hiểu là n thứ ba. Điều kiện n==1 đúng và máy thực hiện câu lệnh

return 1

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

n * gtdq(1)

n hiểu là n thứ hai, nên n=2. Biểu thức trên có giá trị 2*1 = 2. Theo câu lệnh return, máy sẽ thực hiện lần ra khỏi thứ hai. Kết qủa là: đối n thứ hai được giải phóng, hàm cho giá trị gtdq(2)=2, máy trở về với biểu thức

n * gtdq(2)

n hiểu là n thứ nhất, nên n = 3. Biểu thức trền cố giá trị 3.2.1 = 6. Câu lệnh return thực hiện lần ra thứ ba. Ta để ý rằng lần ra thứ ba ứng với lần vào thứ nhất. Mấy sẽ giải phóng n thứ nhất, trở về hàm main() và cho giá trị gtdq(3) = 6. Giá trị này được sử dụng trong câu lệnh printf trong hàm main(). Do vậy trên màn hình hiên ra kết qủa sau

3! = 6

Nhận xét: Qua ví dụ trên ta thấy một hàm đệ quy dùng nhiều vùng nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy với một bài toán có cách giải lặp (không đệ quy) thì nên chọn cách này. Song tồn tại nhiều bài toán chỉ có thể giải bằng đệ quy.

9.2. Cách dùng đệ quy.

+ Bài toán nào có thể dùng đệ quy

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

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

- Trong trường hợp tổng quát, bài toán có thể 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 nhận thấy bài toán tính n! nêu trên thể hiên rất rõ các đặc điểm này.

+ Cách xây dựng hàm đệ quy

Hàm đệ quy thường được viết theo thuật toán sau:

```
if (trường hợp suy biến)
  {
    trình bầy cách giải bài toán
    (giả định đã có cách giải)
  }
else /* trường hợp tổng quát */
  {
    gọi đệ quy tới hàm (đang lập) với
    giá trị khác của tham số
  }
```

9.3. Các ví dụ. Mục này sẽ trình bầy một số ví dụ nhằm minh hoạ cách xây dựng hàm đệ quy nêu trong 9.2.

Ví dụ 1: Xét bài toán tìm ước số chung lớn nhất của hai số nguyên dương x và y.

+ Trường hợp suy biến là trường hợp x=y. Khi đó

usc(x,y) = x.

+ Trường hợp chung x khác y có thể đệ quy như sau:

usc(x,y) = usc(x-y,y) nếu x>y

usc(x,y) = usc(x,y-x) nếu x<y

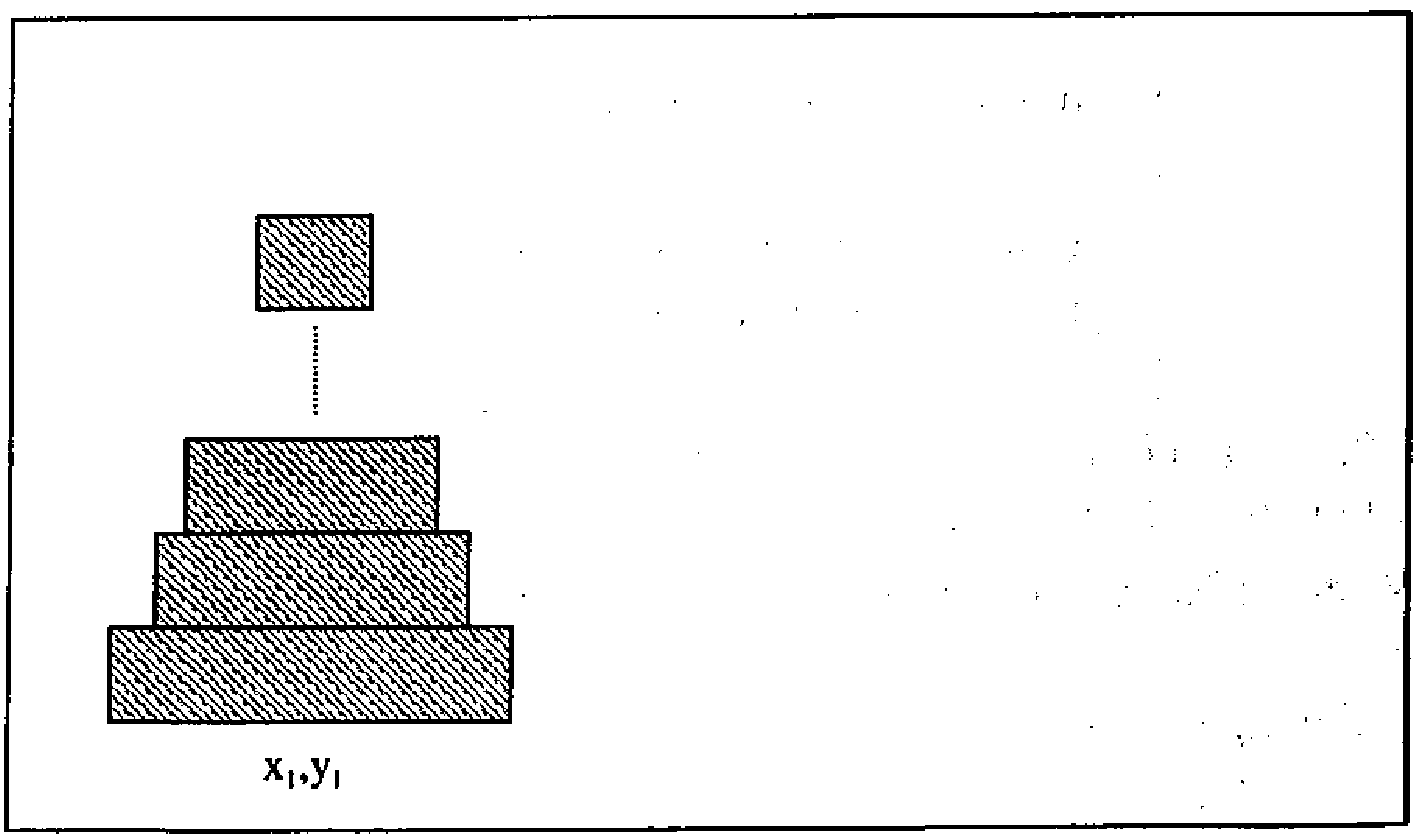
Hàm tìm ước số chung lớn nhất có thể viết như sau:

```
int usc(int x, int y)
{
    if(x==y)
        return x;
    else if(x>y)
        return usc(x-y,y);
    else
        return usc(x,y-x);
}
```

Ví dụ 2. Bài toán tháp Hà Nội được đặt ra như sau: Có một tháp m tầng đang đặt tại vị trí (x1, y1), cần chuyển đến vị trí mới (x2, y2). Khi chuyển cho phép dùng vị trí trung gian (x3, y3).

Để cho rõ ta quy ước như sau:

- Số tầng m lớn hơn hoặc bằng 1.
- Số thứ tự tầng đánh từ 1 đến m từ đỉnh xuống đáy.
- Mỗi tầng được biểu thị bởi một hình chữ nhật có chiều cao là một đơn vị ký tự màn hình, và chiều rộng là một số lẻ đơn vị ký tự. Tầng dưới rộng hơn tầng trên.
- Toạ độ hiểu là toạ độ màn hình ở chế độ văn bản.
- Tâm tháp là tâm của tầng cuối. Như vậy (x1,y1) là tâm tháp ban đầu.



Yêu cầu xây dựng quy trình chuyển tháp theo nhiều bước. Mỗi bước chuyển một tầng từ vị trí này đến vị trí khác. Ngoài ra không được đặt tầng lớn đè lên trên tầng nhỏ.

Thuật toán chuyển tháp có thể diễn đạt như sau:

+ Trường hợp suy biến: m=1, khi đó chỉ cần chuyển tầng 1 từ (x1,y1) đến (x2,y2).

+ Trường hợp tổng quát m>1 có thể giải quyết đệ quy như sau:

- Chuyển tháp m-1 tầng từ (x1, y1-1) đến (x3,y3), dùng (x2,y2) làm vị trí trung gian.

- Chuyển tầng m từ (x1,y1) đến (x2,y2).

- Chuyển tháp m-1 tầng từ (x3, y3) đến (x2,y2-1), dùng (x1,y1) làm vị trí trung gian.

Theo thuật toán này có lập hàm chuyển tháp và chương trình như sau. Chương trình sẽ in ra quy trình vận chuyển. Trong chương 8 sẽ mô phỏng việc chuyển tầng trên màn hình mầu và chương trình sẽ tạo nên một bộ phim tả cảnh chuyển tháp.

```
/* Thap Ha noi */
        #include "stdio.h"
        #include "conio.h"
        void cthap(int m, int x1,int y1, int x2, int y2, int x3, int y3);
        void cthap(int m, int x1,int y1, int x2, int y2, int x3, int y3)
          {
            if (m<1)
                return;
            else
                 if(m==1)
                printf("\nChuyen tang 1 tu (%d,%d) den (%d,%d)",
                x1,y1,x2,y2);
             else
                {
                  cthap(m-1,x1,y1-1,x3,y3,x2,y2);
                  printf("\nChuyen tang %d tu (%d,%d) den (%d,%d)",
                  m, x1,y1,x2,y2);
                  cthap(m-1,x3,y3,x2,y2-1,x1,y1);
                }
          }
```

```
main()
{
    int m;
    printf("\nSo tang: ");
    scanf("%d",&m);
    cthap(m,10,24,30,24,50,24);
}
```

Ví dụ 3. Các vòng for lồng nhau và bài toán tổ hợp.

Bài toán tổ chức m chu trình lồng nhau có đặc trưng đệ quy. Thực vậy tại chu trình k<m sẽ gọi tới chu trình k+1. Tại chu trình m (trường hợp suy biến) sẽ thực hiện một điều gì đó tuỳ theo yêu cầu của mỗi bài toán. Xét hàm tạo một vòng lặp.

Hàm này có 3 đối là:

- Số hiệu vòng lặp k,
- Cận dưới của biến điều khiển d,
- Cận trên của biến điều khiển t.

Hàm có thể xây dựng theo lược đồ sau:

```
void chu_trinh(int k, int d, int t)
{
    int i; /* Biến điều khiển */
    for (i=d; i<=t; ++i)
     if (k==m)
       {
       /* làm điều gì đó */
       }
     else
       {
         /* Gọi đệ quy, chú ý d(i) và t(i) phụ thuộc vào i
         */
         chu_trinh(k+1,d(i),t(i));
       }
}
```

Bây giờ xét bài toán Liệt kê các tổ hợp chập m của n số: 1,2,...,n. Một tổ hợp $i_1, i_2, ..., i_m$ cần thoả mãn điều:

$1 <= i_1 < i_2 < ... < i_m <= n$

Ta sẽ dùng mảng ngoài cs[20] để chứa tổ hợp theo cách:

cs[1] chứa phần tử thứ nhất i_1

cs[2] chứa phần tử thứ hai i_2

.....

Khi m=3 bài toán này được giải quyết bằng 3 vòng for lồng nhau như sau:

```
k=1;
for (i1=1; i1<=n-2; ++i1)
  {
    cs[k]= i1;
    ++k;
    for (i2=i1+1; i2<=n-1; ++i2)
      {
        cs[k]= i2;
        ++k;
        for (i3=i2+1; i3<=n; ++i3)
          {
            cs[k]= i3;
            /* Do k=3, in cs[1],...,cs[k] */
          }
      }
  }
```

Trong trường hợp tổ hợp chập m, ta phải tổ chức m chu trình lồng nhau. Muốn vậy phải dùng hàm:

```
void chu_trinh(int k, int d, int t)
```

Với bài toán tổ hợp thì:

+ Khi k=1, cận dưới d=1.

+ Tai vòng lặp k cận trên bằng n-m+k, vì vậy có thể bỏ đối t.

+ d(i) = i+1.

Từ phân tích trên dễ dàng viết được hàm chu_trinh cho bài toán tổ hợp như sau:

```
#include "stdio.h"
int m,n,cs[20];
void chu_trinh(int k, int d)
  {
    int i;        /* Biến điều khiển */
```

```
        for (i=d; i<= n-m+k; ++i)
         {
           cs[k]=i;
           if (k==m)
             {
               /* in một tổ hợp */
               int j;
               printf("\n");
               for(j=1;j<=k;++j)
                 printf("%3d",cs[j]);
             }
          else
         chu_trinh(k+1,i+1);
        }
     }
```

Hàm chu_trinh sẽ được áp dụng trong main như sau:

```
/* Liet ke cac to hop */
       #include "conio.h"
       main()
         {
           clrscr();
           printf("\n Vao m va n (m<=n) ");
           scanf("%d%d",&m,&n);
           chu_trinh(1,1);
           getch();
         }
```

Ví dụ 4: Một bài toán ứng dụng tổ hợp. Cho dãy n số thực, hãy tìm bộ m số có tích lớn nhất. Một bộ m số chính là một tổ hợp chập m của n số, vì vậy đây chính là bài toán duyệt các tổ hợp. Khi nhận được một tổ hợp ta cần tính tích của chúng và so sánh với các giá trị trước đó để chọn tích lớn nhất. Ta đưa vào các biến, mảng ngoài sau:

- Mảng thực x[20] chứa dãy số thực,
- Mảng nguyên cs[20] chứa các tổ hợp được duyệt,
- Mảng nguyên csmax[20] chứa tổ hợp cần tìm (có tích lớn nhất),
- Biến thực max chứa tích lớn nhất.

Hàm chu_trinh được sửa đôi chút: Khi k=m (tức là nhận được một tổ hợp) thì tính tích của chúng và so sánh với phương án cũ để tìm tích lớn hơn.

Phương án ban đầu chính là tổ hợp gồm các chỉ số 1,...,m được xây dựng trong hàm main.

Dưới đây là chương trình tìm bộ m phần tử (trong dãy n số) có tích lớn nhất.

```
/*
tim bo m phan tu (trong n so) co tich lon nhat
*/
        #include "stdio.h"
        #include "conio.h"
        int m,n,cs[20],csmax[20];
        float max,x[20];
        void chu_trinh(int k, int d)
          {
            int i;
            for (i=d; i<=n-m+k; ++i)
              {
                cs[k]=i;
                if (k==m)
                  {
                    int j;
                    float s=1.0;
                    for(j=1;j<=k;++j)
                      s *= x[cs[j]];
                    if (s>max)
                    {
                      max=s;
                      for(j=1;j<=k;++j)
                        csmax[j] = cs[j];
                    }
                  }
              else
                chu_trinh(k+1,i+1);
            }
          }
```

```
main()
  {
    int j;
    clrscr();
    printf("\n Vao m va n (m<=n) ");
    scanf("%d%d",&m,&n);
    for(j=1;j<=n;++j)
      {
        printf("\nx[%d]= ",j);
        scanf("%f",&x[j]);
      }
    max=1;
    for(j=1;j<=m;++j)
    {
        csmax[j] = j;
        max *= x[j];
    }
    chu_trinh(1,1);
    for(j=1;j<=m;++j)
      {
        printf("\nx[%d]= %f ",csmax[j],x[csmax[j]]);
      }
    printf("\nTich= %f ",max);
  }
```

Ví dụ 5:

Duyệt trên cây và bài toán biểu diễn tổng.

Bài toán duyệt trên cây có thể giải theo đệ quy như sau.

Xuất phát từ một đỉnh nào đó ta thực hiện 2 động tác:

- Xét đỉnh này.
- Đi đến các đỉnh kề nó bằng phép gọi đệ quy.

Bằng thủ tục trên có thể đi qua tất cả các đỉnh của một cây. Ta minh hoạ điều này bằng việc xét bài toán phân tích số n nguyên dương thành tổng các số nguyên dương nhỏ hơn hơn n. Đỉnh xuất phát là n. Từ đỉnh này có thể đi đến các đỉnh gồm một cặp 2 số có tổng bằng n, các đỉnh đó là

(i,n-i) , với i=1,...,n/2.

Tại mỗi đỉnh 2 số lại có thể đi đến các đỉnh 3 số. Một cách tổng quát, từ đỉnh k số

(a1,...,ak-1,ak)

có thể đi đến các đỉnh (k+1) số sau:

(a1,...,ak-1,i,ak - i), với i=ak-1,...,ak/2

Nhận xét: mỗi dãy số tại đỉnh là đơn điệu không giảm.

Dễ dàng chứng minh mọi tổng đều ứng với một đỉnh nào đó thực vậy xét tổng (k+1) số:

n = t1 + t2 + ... + tk + tk+1

(dãy ti không giảm). Rõ ràng tổng này có thể suy ra từ đỉnh gồm k số:

(t1,...,tk-1,tk + tk+1)

Như vậy mỗi tổng (k+1) số đều có thể suy ra từ đỉnh k số. Do tập các tổng một số là đầy đủ (chỉ gồm n), nên lần lượt suy ra các tập k số cũng đầy đủ với mọi k>1. Đó là điều cần chứng minh.

Ta dùng mảng ngoài a[100] để chứa bộ số của đỉnh. Với đỉnh xuất phát thì:

k=1 và a[1]=n

Vì cần dùng đến a[k-1], nên đưa thêm a[0] = 1. Ta xây dựng hàm duyệt_đỉnh k với nội dung sau:

1. Xét đỉnh k>1: in các số a[1],...,a[k] dưới dạng tổng

   a[1] + a[2] + ... + a[k]

   (không xét đỉnh với k=1)

2. Đi đến các đỉnh (k+1) số:

   (a1,...,ak-1,i,ak - i) , với i=ak-1,...,ak/2

Hàm

void duyet_dinh(int k);

trong chương trình dưới đây lập theo thuật toán này.

```
/*
   phan tich n = tong cac so nho hon n
   duyệt trên các đỉnh của cây bằng thủ tục đệ quy
*/
      #include "stdio.h"
      #include "conio.h"
      int a[100];
```

```
/* Da biet a[0],...,a[k] */
    void duyet_dinh(int k)
      {
        int x,i;
          /* xet dinh k >1 */
        if(k>1)
        for(i=1;i<=k;++i) printf("%c%d",i>1?'+':'\n',a[i]);
          /* đi đến các đỉnh (k+1) số */
        x=a[k];
        for(i=a[k-1];i<= x/2;++i)
          {
            a[k]=i; a[k+1]=x-i; duyet_dinh(k+1);
          }
      }
    main()
      {
        int n;
        clrscr();
        printf("\n n="); scanf("%d",&n); a[0]=1; a[1]=n;
        duyet_dinh(1); getch();
      }
```

§10. ĐỐI DÒNG LỆNH

Ta đã quen dùng hàm main không đối dạng

```
    main()
      {
        .
        .
        .
      }
```

hay

```
    void main(void)
      {
        .
        .
        .
      }
```

Mục này giới thiệu cách dùng hàm main có đối. C cho phép đưa vào hàm main hai đối: đối thứ nhất là biến kiểu int, đối thứ hai là mảng con trỏ kiểu char. Tất cả đối do người lập trình tự đặt. Dạng của hàm main có đối là

```
main (int n, char *a[])
  {
    .
    .
    .
  }
```

giá trị của n và a được nhận từ bàn phím khi khởi động chương trình. Để minh họa điều này ta giả sử tệp chương trình thực hiện có tên là

PNAME.EXE

Như đã biết, nếu chương trình chứa hàm main không đối thì việc khởi động chương trình được thực hiện bằng cách gọi tên của nó. Nói cách khác, ta bấm các phím:

PNAME VÀ ENTER

Trên màn hình sẽ hiện lên tên của chương trình

D:\TC>PNAME

(giả sử PNAME nằm ở thư mục TC của ổ D). Nếu chương trình PNAME chứa hàm main có đối thì khi khởi động ta cần đưa thêm vào một vài tham số. Mỗi tham số được xem là một xâu ký tự. Các tham số đặt trên cùng một dòng với tên chương trình. Sau khi đã đưa lên màn hình tham số cuối cùng ta bấm phím ENTER.

Xét vài trường hợp cụ thể .

Ví dụ 1: Giả sử ta cho hiện lên màn hình dòng tham số

D:\TC>PNAME hom nay la ngay 2 thang 9

Trong trường hợp này ta đưa vào 7 tham số, các tham số cách nhau bởi ít nhất một dấu cách, khi đó n nhận giá trị 8 còn dãy tham số sẽ được ghi nhận vào mảng a. Nói một cách chính xác hơn, nội dung các phần tử của mảng a là:

a[0]="D:\TC>PNAME"		a[1] = "hom"
a[2] = "nay"	a[3] = "la"	a[4] = "ngay"
a[5]="2"	a[6] = "thang"	a[7] = "9"

Ví dụ 2: Giả sử ta cho hiện lên màn hình dòng tham số

D:\TC>PNAME 123 -456 34

thì n và a nhận các giá trị:

n = 4

a[0] = "D:\TC>PNAME"   a[1] ="123"

a[2] = "-456"   a[3] = "34"

Một cách tổng quát nếu xem tên chương trình là tham số đầu tiên, thì đối n sẽ chứa số tham số, còn các phần tử của mảng a sẽ chứa địa chỉ của các tham số đưa vào. Trong thân của hàm main ta có thể sử dụng n và a để xử lý các tham số nhận được từ bàn phím. Dưới đây là hai chương trình minh họa cách dùng các đối dòng lệnh. Chương trình thứ nhất chỉ đơn giản là sử dụng các đối (n và a) để in các tham số nhận được khi thao tác chương trình. Chương trình thứ hai sẽ xem các tham số như các số thực và tính tổng của chúng.

Chương trình 1

```
/* Chuong trinh in cac tham so dong lenh
Ten tep chuong trinh thuc hien la INTSDL */
        #include "stdio.h"
        void main(int n, char **a)
          {
            int i;
            printf("\n Ten chuong trinh: %s",a[0]);
            printf("\n Cac tham so nhan duoc:");
            for (i=1; i<n; ++i) printf("\n%s",a[i]);
          }
```

Với thao tác

D:\TC>INTSDL hom nay la ngay 2 thang 9

Chương trình cho kết qủa sau

```
Ten chuong trinh: D:\TC>INTSDL.EXE
Cac tham so nhan duoc:
hom nay
la
ngay
2
thang
9
```

Chương trình 2

```
/* Chuong trinh tinh tong cua day so, moi so la mot tham so dong lenh
   Ten cua tep chuong trinh thuc hien la SUM */
        #include <stdio.h>
        #include <stdlib.h>
        main(int pc, char *pv[])
```

```
{
    float x, s=0.0;  int i;
    if(pc > 1)
      {
      printf("\nKet qua");
      for(i=1; i<pc; ++i)
        {
          x=atof(pv[i]); /* Đổi từ chuỗi sang float */
          printf("\n%0.2f",x);  s += x;
        }
      }
    printf("\nTong la: %0.2f\n",s);
}
```

Với thao tác

D:\TC>SUM 1.5 3.6 -97 5.1

chương trình cho kết qủa sau:

```
Ket qua
1.50 3.60
-97.00
5.10
Tong la: -86.80
```

BÀI TẬP CHƯƠNG 6

Viết hàm thực hiện các bài toán dưới đây.

Bài 1. Giải hệ phương trình bậc nhất

$ax + by = c$

$dx + ey = f$

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

Bài 2. Tính đa thức cấp n: $f(x) = a_0x^n + ... + a_{n-1}x + a_n$

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

Bài 3. Tính cực đại và cực tiểu của một dãy số.

Bài 4. Tính giá trị trung bình và độ lệch tiêu chuẩn của một đại lượng ngẫu nhiên theo công thức

$$s = \frac{x^1 + ... + x^n}{n}$$

$$d^2 = \frac{(x_1 - s)\sup 2 + ... + (x_n - s)^2}{n}$$

Trong đó x1,...,xn là dãy quan sát nhận được.

Bài 5. Nhân ma trận A cấp mxn với véc tơ X cấp n.

Bài 6. Giải phương trình AX = B

bằng phương pháp khử Gauss (trong đó A là ma trận vuông cấp n, b là véc tơ cấp n).

Bài 7. Hoán vị một ma trận chữ nhật A cho trước.

Bài 8. Tìm nghịch đảo của một ma trận vuông bằng phương pháp Jordance.

Bài 9. Tính tích phân của hàm f(x) trên đoạn [a,b].

Bài 10. Xây dựng hàm h(x) là max của các hàm f(x) và g(x) trên đoạn [a,b].

Bài 11. Giải phương trình trùng phương

$ax^4 + bx^2 + c = 0$

Bài 12. Tìm tọa độ giao điểm của hai đường thẳng AB và CD khi biết tọa độ của các điểm A,B,C,D.

Bài 13. Xây dựng các hàm số sau đây bằng phương pháp đệ qui

$f(x,n) = x^n$

$s(n) = (2n)!!$

$p(n) = 1^3 + 2^3 + \ldots + n^3$

Bài 14. Tính các giá trị nhỏ nhất và lớn nhất của hàm f(x) trên đoạn [a,b].

Bài 15. Tìm một nghiệm của phương trình f(x)=0 trên đoạn [a,b]. Giả thiết hàm f(x) liên tục trên [a,b] và f(a)f(b) < 0.

Bài 16. 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 lớn nhất (dùng đệ quy).

yêu cầu đối với các bài toán trên:

- Xây dựng các hàm ứng với các bài toán đề ra.
- Viết thêm hàm main để sử dụng các hàm này.
- Chạy máy theo các dữ liệu tự chọn.

CHƯƠNG 7

CẤU TRÚC

Để lưu trữ và xử lý thông tin trong máy tính ta có các biến và các mảng. Mỗi biến chứa được một giá trị. Mảng có thể xem là tập hợp nhiều biến có cùng một kiểu giá trị và được biểu thị bằng một tên. 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. Cấu trúc là một tập hợp các biến, các mảng và được biểu thị bởi một tên duy nhất. Khái niệm cấu trúc trong C có những nét tương tự như khái niệm bản ghi (record) trong PASCAL hay FOXPRO. Một ví dụ truyền thống về cấu trúc là phiếu ghi lương: mỗi công nhân được miêu tả bởi một tập hợp các thuộc tính như tên, địa chỉ, ngày sinh, bậc lương,... Một vài trong các thuộc tính này lại có thể là cấu trúc: tên có thể có nhiều thành phần, địa chỉ và thậm chí ngày sinh cũng vậy.

§1. KIỂU CẤU TRÚC

Trước khi xây dựng một hoặc một số cấu trúc có cùng một kiểu ta cần phải mô tả kiểu của nó. Điều này cũng tương tự như việc phải thiết kế một kiểu nhà trước khi xây dựng những căn nhà thực sự ở các địa điểm khác nhau. Khi định nghĩa một kiểu cấu trúc cần chỉ ra: tên của kiểu cấu trúc và các thành phần của nó. Việc này được thực hiện theo mẫu sau:

Mẫu 1.1

```
struct tên_kiểu_cấu_trúc
  {
   Khai báo các thành phần của nó
  };
```

trong đó: struct là từ khóa, tên_kiểu_cấu_trúc là một tên bất kỳ do người lập trình tự đặt theo các quy tắc nêu trong Chương 1.

Thành phần của cấu trúc có thể là: biến, mảng, nhóm bit (§4), hợp (§12) hoặc một cấu trúc khác mà kiểu của nó đã định nghĩa từ trước.

Ví dụ 1.1: Đoạn chương trình

```
struct ngay {
            int ngay_thu;
            char ten_thang[10];
            int nam;
          };
```

mô tả một kiểu cấu trúc có tên là ngay gồm ba thành phần: biến nguyên ngay_thu, mảng ten_thang và biến nam.

Ví dụ 1.2: Đoạn chương trình

```
struct nhan_cong
  {
    char ten [15];
    char dia_chi [20];
    double bac_luong;
    struct ngay ngay_sinh;
    struct ngay ngay_vao_co_quan;
  };
```

thiết kế một kiểu cấu trúc có tên là nhan_cong gồm năm thành phần. Hai thành phần đầu là các mảng ten và dia_chi (kiểu char). Thành phần thứ ba là một biến kiểu double (biến bac_luong). Hai thành phần còn lại là các cấu trúc ngay_sinh và ngay_vao_co_quan, cả hai cấu trúc này xây dựng theo kiểu cấu trúc ngay định nghĩa trong ví dụ 1.1.

Chú ý 1.1: Có thể dùng toán tử typedef để định nghĩa các kiểu cấu trúc ngay và nhan_cong ở trên như sau:

```
typedef struct
  {
    int ngay_thu;
    char ten_thang[10];
    int nam;
  } ngay;
typedef struct
  {
    char ten [15]; char dia_chi [20];
    double bac_luong;
    ngay ngay_sinh;
    ngay ngay_vao_co_quan;
  } nhan_cong;
```

§2. KHAI BÁO THEO MỘT KIỂU CẤU TRÚC ĐÃ ĐỊNH NGHĨA

Việc làm này nhằm xây dựng những cấu trúc thực sự theo các kiểu đã thiết kế ở các phần trên. Về mặt ngữ pháp điều này hoàn toàn giống như việc khai báo các biến và các mảng. Khi khai báo một biến cần chỉ ra kiểu và tên của nó. Ví dụ khai báo:

```
float x , y;
```

cho ta hai biến kiểu float với các tên là x và y. Ta cũng sẽ làm y hệt như vậy khi khai báo biến cấu trúc. Giả sử ta đã có các kiểu cấu trúc ngay và nhan_cong như trong các ví dụ 1.1 và 1.2. Khi đó ta có thể xét các ví dụ sau.

Ví dụ 2.1: Khai báo

```
struct ngay ngay_di, ngay_den;
```

sẽ cho ta hai biến cấu trúc với tên là ngay_di và ngay_den. Cả hai đều được xây dựng theo kiểu ngay (xem ví dụ 1.1).

Ví dụ 2.2: Khai báo

```
struct nhan_cong nguoi_a,nguoi_b;
```

sẽ cho ta hai cấu trúc với tên là nguoi_a và nguoi_b. Cả hai hai đều được xây dựng theo kiểu nhan_cong (xem ví dụ 1.2).

Một cách tổng quát, việc khai báo cấu trúc được thực hiện theo mẫu:

Mẫu 2.1.

struct tên kiểu cấu trúc danh sách tên cấu trúc;

Sau đây là một vài điều cần lưu ý .

Nhận xét 2.1. Các biến cấu trúc được khai báo theo mẫu trên sẽ được cấp phát bộ nhớ một cách đầy đủ cho tất cả các thành phần của nó.

Nhận xét 2.2. Việc khai báo có thể thực hiện đồng thời với việc định nghĩa kiểu cấu trúc. Muốn vậy chỉ cần đặt danh sách tên biến cấu trúc cần khai báo vào sau dấu } cuối cùng trong mẫu 1.1.

Nói cách khác: Để vừa thiết kế kiểu vừa khai báo các biến cấu trúc ta sử dụng mẫu sau:

Mẫu 2.2.

```
struct tên_kiểu_cấu_trúc
  {
    các thành phần của cấu trúc
  } danh sách tên biến cấu trúc;
```

Ví dụ 2.3: Các cấu trúc ngay_di và ngay_den trong ví dụ 2.1 có thể được xây dựng theo cách:

```
struct ngay
  {
     int ngay_thu;
     char ten_thang [10];
     int nam;
  } ngay_di, ngay_den;
```

Ví dụ 2.4: Các cấu trúc nguoi_a, nguoi_b trong ví dụ 2.2 có thể xây dựng theo mẫu sau (giả sử kiểu ngay đã được mô tả như ở ví dụ 1.1)

```
struct nhan_cong
  {
    char ten [15];
    char dia_chi [20];
    double bac_luong;
    struct ngay ngay_sinh;
    struct ngay ngay_vao_co_quan;
  } nguoi_a, nguoi_b;
```

Nhận xét 2.3. Khi vừa định nghĩa kiểu (cấu trúc) vừa khai báo cấu trúc như trong các ví dụ 2.3 và 2.4, ta có thể không cần đến tên kiểu cấu trúc. Nói cách khác các cấu trúc có thể được khai báo theo mẫu sau.

Mẫu 2.3

```
struct
  {
   các thành phần của cấu trúc
  } danh sách tên cấu trúc;
```

Ví dụ 2.5: Các cấu trúc ngay_di và ngay_den trong ví dụ 2.3 được khai báo theo cách:

```
struct
  {
    int ngay_thu;
    char ten_thang[10];
    in nam;
  } ngay_di, ngay_den;
```

Cũng cần nói thêm sự khác nhau giữa mẫu 2.2 và mẫu 2.3. Điều này sẽ trở nên hoàn toàn rõ ràng khi so sánh ví dụ 2.3 và ví dụ 2.5. ở ví dụ 2.3 ngoài việc xây dựng được các cấu trúc ngay_di, ngay_den ta còn cho ra đời được kiểu cấu trúc ngay. Kiểu ngay có thể sử dụng để khai báo các cấu trúc khác. ở ví dụ 2.5 ta chỉ đạt được mục đích đầu của ví dụ 2.3.

Chú ý: Nếu dùng typedef để định nghĩa kiểu cấu trúc, thì khi khai báo ta chỉ cần dùng tên kiểu (bỏ từ khóa struct). Ví dụ nếu kiểu cấu trúc ngay được định nghĩa như trong chú ý 1.1 của $1, thì các cấu trúc ngay_di và ngay_den trong ví dụ 2.1 có thể khai bao như sau:

```
ngay ngay_di, ngay_den;
```

Chúng ta đã khá quen biết việc sử dụng các biến, các phần tử mảng và tên mảng trong các câu lệnh. Ta cũng đã biết các thành phần cơ bản của một cấu trúc là biến và mảng, nên một lẽ tự nhiên và cũng là một quy tắc cần ghi nhớ là việc xử lý một cấu trúc bao giờ cũng phải được thực hiện thông qua các thành phần cơ bản của nó. Để truy nhập tới một thành phần cơ bản (biến hoặc mảng) của một cấu trúc ta sử dụng một trong cách viết sau:

Mẫu 3.1

tên_cấu_trúc.tên_thành_phần

tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần

tên_cấu_trúc.tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần

....

Cách viết thứ nhất của mẫu 3.1 được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc. Ví dụ biến ngay_thu, biến nam và mảng ten_thang là các thành phần trực tiếp của cấu trúc ngay_di (xem ví dụ 2.3). Mảng ten, mảng dia_chi và biến bac_luong là các thành phần trực tiếp của cấu trúc nguoi_b (xem ví dụ 2.4). Các cách viết còn lại của mẫu 3.1 được sử dụng khi biến hoặc mảng là thành phần trực tiếp của một cấu trúc mà bản thân cấu trúc này lại là thành phần của một cấu trúc lớn hơn.

Ví dụ 3.1: Ta xét vài phép toán trên các thành phần của cấu trúc nguoi_a và nguoi_b (xem ví dụ 2.4).

Câu lệnh

```
printf ( "%s", nguoi_a.ten );
```

sẽ đưa tên của nguoi_a lên màn hình.

Khi thực hiện câu lệnh

```
s = nguoi_a.bac_luong + nguoi_b.bac_luong;
```

thì ở biến s sẽ nhận được tổng bậc lương của nguoi_a và nguoi_b.

Câu lệnh

```
printf("%d",nguoi_a.ngay_sinh.nam);
```

sẽ cho lên màn hình năm sinh của nguoi_a.

Câu lệnh

```
printf ("%d",nguoi_b.ngay_vao_co_quan.nam);
```

sẽ cho lên màn hình năm vào cơ quan của nguoi_b.

Chú ý 3.1. Có thể sử dụng phép toán lấy địa chỉ đối với thành phần cấu trúc để nhập số liệu trực tiếp vào thành phần cấu trúc. Ví dụ có thể viết:

```
scanf("%d",&nguoi_b.ngay_vao_co_quan.nam);
```

Nhưng đối với các thành phần không nguyên, việc làm trên có thể dẫn đến treo máy (xem ví dụ 9.1, §9 dưới đây). Vì vậy trước tiên nên nhập số liệu vào một biến trung gian, sau đó mới gán cho thành phần cấu trúc. Cách làm như sau:

```
int x;
scanf("%d",&x);
nguoi_b.ngay_vao_co_quan.nam = x;
```

Chú ý 3.2. Để tránh dài dòng khi làm việc với các thành phần cấu trúc ta có thể dùng lệnh #define. Ví dụ câu lệnh scanf trong chú ý trên có viết theo cách sau:

```
#define p nguoi_b.ngay_vao_co_quan
...
scanf("%d",&p.nam);
```

Để kết thúc mục này, ta đưa ra một ví dụ đơn giản về tổ chức thông tin cán bộ. Giả sử dữ liệu của mỗi cán bộ gồm:

+ Ngày tháng năm sinh.

+ Ngày tháng năm vào cơ quan.

+ Bậc lương.

yêu cầu viết một chương trình để:

+ Xây dựng cấu trúc cho cán bộ.

+ Vào số liệu của một cán bộ.

+ Đưa số liệu đó ra máy in.

Dưới đây là một chương trình và kết quả thực hiện của nó.

```
/* Chuong trinh minh hoa:
   + Cach xay dung cau truc
   + Cach vao ra cho cac thanh phan cua cau truc */
      #include "stdio.h"
      typedef struct
        {
          int ng;
          char t[10];
          int nam;
      }date;
      typedef struct
          {
            date dates;
            date datecq;
            float luong;
          }person;
```

```
main()
  {
   person p;
   printf("\nNgay sinh: ngay thang nam ");
   / *25 hai 1945 */
   scanf("%d%s%d",&p.dates.ng,p.dates.t,&p.dates.nam);
   printf("\nNgay vao co quan: ngay thang nam ");
   /* 30 tu 1970*/
   scanf("%d%s%d",&p.datecq.ng,p.datecq.t,&p.datecq.nam);
   printf("\n tien luong= ");
   /* 550.00 */
   scanf("%f",&p.luong );
   fprintf(stdprn,"\n%2d %3s %d",
   p.dates.ng,p.dates.t,p.dates.nam);
   fprintf(stdprn,"\n %2d %3s %d",
   p.datecq.ng,p.datecq.t,p.datecq.nam);
   fprintf(stdprn,"\nluong =%7.2f",p.luong);
  }
```

Kết quả thực hiện chương trình

```
25  hai  1945
30  tu   1970
luong = 550.00
```

§4.THÀNH PHẦN KIỂU FIELD (NHÓM BIT)

Các thành phần nguyên (unsigned hoặc signed) với miền giá trị nhỏ (như tuổi biến thiên từ 1 đến 100) có thể khai báo kiểu nhóm bit theo mẫu sau:

```
struct date
  {
   int a:8;
   int b:6;
   int c:8;
   int d:2;
  } x;
```

Khi đó sizeof(struct date) = 3 (3 byte). Khi dùng kiểu field cần lưu ý các điểm sau:

1. Kiểu của field phải là int (unsigned hoặc signed).

2. Độ dài của mỗi field không qúa 16 bit.

3. Khi muốn bỏ qua một số bit thì ta bỏ trống tên trường.

Ví dụ khi viết

```
int:8;
int:x;
```

tức là bỏ qua 8 bit thấp, x chiếm 8 bit cao của một word.

4. Không cho phép lấy địa chỉ thành phần kiểu field.

5. Không thể xây dựng các mảng kiểu field.

6. Không thể trả về từ hàm bằng một thành phần kiểu field.

Chẳng hạn không cho phép viết:

```
return x.d;
```

mà phải dùng thủ thuật sau:

```
int t = x.d;
return t;
```

Ứng dụng của nhóm bit: Nhóm bit thường được ứng dụng để:

+ Tiết kiệm bộ nhớ.

+ Dùng trong union (xem §12) để lấy ra các bit của một từ.

Ví dụ:

```
union
  {
  struct
    {
      unsigned a1;
      unsigned a2;
    } s;
  struct
    {
       unsigned n1:1;
       unsigned:15;
       unsigned n2:1;
       unsigned:7;
       unsigned n3:8;
    }f;
  } u;
```

Khi đó:

u.f.n1 la bit 0 của u.s.a1

u.f.n2 la bit 0 của u.s.a2

u.f.n3 la byte cao của u.s.a2

§5. MẢNG CẤU TRÚC

Như đã biết ở các mục trên: khi sử dụng một kiểu giá trị (kiểu int chẳng hạn) ta có thể khai báo các biến và các mảng kiểu int. Ví dụ khai báo:

```
int a,b,c[10];
```

cho ta hai biến nguyên a,b và mảng nguyên c.

Hoàn toàn tương tự như vậy: Có thể sử dụng một kiểu cấu trúc đã mô tả để khai báo các cấu trúc và các mảng cấu trúc.

Ví dụ 5.1: Giả sử kiểu cấu trúc nhan_cong đã định nghĩan trong §2, khi đó khai báo:

```
struct nhan_cong nguoi_a,nguoi_b,to_1[10],to_2[20];
```

sẽ cho:

+ hai biến cấu trúc nguoi_a và nguoi_b.

+ hai mảng cấu trúc to_1 và to_2.

Mảng to_1 có 10 phần tử và mảng to_2 có 20 phần tử. Mỗi phần tử của chúng là một cấu trúc kiểu nhan_cong.

Ví dụ 5.2: Đoạn chương trình sau sẽ tính tổng lương của 10 người ở to_1.

```
double s=0;
for(i=0; i<10; ++i)
s += to_1[i].bac_luong;
```

Ghi chú: Không cho phép sử dụng phép toán lấy địa chỉ đối với các thành phần của mảng cấu trúc. Chẳng hạn không cho phép viết:

```
&to_1[i].bac_luong
```

(nếu kiểu của bac_luong là nguyên thì được)

§6. KHỞI ĐẦU CHO MỘT CẤU TRÚC

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

Ví dụ 6.1: Đoạn chương trình

```
struct date
        {
            int day; int month;
            int year;
            int yearday;
            char * month_name;
        };
```

struct date dd = { 4,7,1990,120," December" };

xác định một cấu trúc (theo kiểu date) có tên là dd và khởi đầu cho các thành phần của nó. Như vậy: nội dung của dd.day là 4, của dd.month là 7, của dd.year là 1990, của dd.yearday là 120 và của dd.month_name là "December".

Chú ý nếu month_name được khai báo kiểu mảng ký tự như

```
char month_name[20];
```

thì các khởi đầu trên vẫn đúng.

Ví dụ 6.2. Đoạn chương trình

```
struct month
    {
        int number;
        char * name;
    } year[12] = {
                    { 1,"january" },
                    { 2,"february" },
                    . . .
                    { 12,"december" }
                 };
```

xác định và khởi đầu một mảng cấu trúc có tên là year bao gồm 12 phần tử. Vì mỗi phần tử của mảng lại là một cấu trúc (kiểu month) nên để khởi đầu cho mảng year, một cách hợp lý hơn cả là sử dụng 12 bộ khởi đầu cho 12 cấu trúc tương ứng.

Chẳng hạn

```
{ 1,"january" }
```

là bộ khởi đầu cho phần tử thứ nhất của mảng year.

Cũng như đối với các mảng khác, khi khởi đầu một mảng cấu trúc ngoài (hoặc tĩnh) ta không cần chỉ ra kích cỡ của nó. Lúc đó kích cỡ của các mảng được khởi đầu sẽ được xác định một cách chính xác (khi dịch chương trình) thông qua số các bộ khởi đầu. Như vậy, đoạn chương trình trong ví dụ 6.2 có thể viết một cách khác như sau

```
struct month
    {
     int number;
     char * name;
    } year[] = {
                { 1,"january" },
                { 2,"february" }'
                . . .
                { 12,"december" }
               };
```

Để xác định số phần tử của year ta có thể dùng toán tử sizeof theo cách sau:

```
int n = sizeof(year)/sizeof(struct month);
```

Nhận xét cuối cùng ở mục này là: những gì đã nói về việc khởi đầu một mảng thông thường vẫn còn đúng đối với mảng cấu trúc, đó là:

+ Chỉ cho phép khởi đầu các cấu trúc và mảng cấu trúc ngoài (tĩnh). Chúng sẽ nhận giá trị 0 nếu không được khởi đầu.

+ Nếu kích thước của mảng cấu trúc cần khởi đầu đã được khai báo là n thì số bộ khởi đầu (m) cần không vượt qúa n. Mỗi bộ khởi đầu cho giá trị của một phần tử mảng cấu trúc. Khi m < n thì chỉ có m phần tử đầu của mảng được khởi đầu, (n-m) phần tử còn lại nhận giá trị 0.

+ Việc khởi đầu được thực hiện một lần khi dịch chương trình.

Ví dụ khi dịch đoạn chương trình

```
struct
  {
    float a;
    int b;
    char * c;
  }
d[10]={
      { 7.1,5,"alpha" },
      { -10.6,8,"beta"}
     };
```

chỉ hai phần tử đầu của mảng cấu trúc d được khởi đầu. Trong ví dụ này n = 10 và m = 2.

§7. PHÉP GÁN CẤU TRÚC

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

1. Gán hai biến (cấu trúc) cho nhau.
2. Gán biến cho phần tử mảng (cấu trúc)
3. Gán phần tử mảng cho biến.
4. Gán hai phần tử mảng cho nhau.

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

Đoạn chương trình sau minh họa cách dùng phép gán cấu trúc để sắp xếp n thí sinh theo thứ tự giảm của tổng điểm.

```
struct thi_sinh
    {
      char ht[25]; /* họ tện */
      float td; /* tổng điểm */
    } tg,ts[1000];
int i,j,n;
   for(i=1;i<=n-1;++i)
   for(j=i+1;j<=n;++j)
      if(ts[i].td<ts[j].td)
        {
         tg=ts[i];
         ts[i]=ts[j];
         ts[j]=tg;
        }
```

§8. CON TRỎ CẤU TRÚC VÀ ĐỊA CHỈ CẤU TRÚC

8.1. Con trỏ và địa chỉ.

Xét các kiểu cấu trúc ngay va nhan_cong (§2):

```
struct ngay {
              int ngay_thu;
              char ten_thang[10];
              int nam;
            };
struct nhan_cong
       {
              char ten [15];
              char dia_chi [20];
              double bac_luong;
              struct ngay ngay_sinh;
              struct ngay ngay_vao_co_quan;
       };
```

Khi đó con trỏ cấu trúc kiểu nhan_cong có thể được khai báo cùng với biến và mảng (cấu trúc) như sau:

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

Trong khai báo trên thì:

+ p, p1 và p2 là con trỏ cấu trúc

+ nc1 và nc2 là biến cấu trúc

+ ds là mảng cấu trúc

Con trỏ cấu trúc dùng để lưu trữ địa chỉ của biến cấu trúc và mảng cấu trúc, vì vậy các phép gán sau đây là hợp lệ:

```
p1=&nc1; /* Gửi địa chỉ nc1 vào p1 */
p2=&ds[2]; /* Gửi địa chỉ ds[2] vao p2 */
p = ds; /* Gửi địa chỉ ds[0] vào p */
```

8.2. Truy nhập thông qua con trỏ. Có thể truy nhập đến các thành phần thông qua con trỏ theo một trong hai cách sau:

Cách 1:

Tên_con_trỏ->tên_thành_phần

Cách 2:

(*Tên_con_trỏ).tên_thành_phần

Chẳng hạn, sau khi thực hiện các phép gán địa chỉ trong 8.1 thì các cách viết sau là tương đương:

nc1.ngay_sinh.nam và p1->ngay_sinh.nam

ds[2].ngay_sinh.ten_thang và (p2*).ngay_sinh.ten_thang

8.3. Dùng phép gán thông qua con trỏ. Giả thiết đã có các lệnh:

```
p1=&nc1; /* Gửi địa chỉ nc1 vào p1 */
p2=&ds[2]; /* Gửi địa chỉ ds[2] vao p2 */
```

khi đó trong các phép gán cấu trúc:

+ Có thể dùng *p1 thay cho nc1

+ Có thể dùng *p2 thay cho ds[2]

Như vậy cách viết

```
ds[4]=nc1;      ds[2]= nc2;
```

tương đương với

```
ds[4]=*p1;      *p2=nc2;
```

8.4. Phép cộng địa chỉ. Sau các phép gán

```
p=ds;
p2=&ds[2];
```

thì p trỏ tới ds[0] va p2 trỏ tới ds[2]. Ta có thể dùng các phép cộng, trừ địa chỉ để làm cho p và p2 trỏ tới các thành phần bất kỳ nào khác. Ví dụ sau các câu lệnh

```
p=p+10; p2=p2-2;
```

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

8.5. Con trỏ và mảng. Giả sử con trỏ p trỏ tới đầu mảng ds, khi đó hai điều sau là đáng ghi nhớ.

8.5.1. Có thể truy nhập tới các thành phần bằng 3 cách sau:

ds[i].thành_phần

p[i].thành_phần

(p+i)->thành_phần

Ví du ba cách viết dưới đây có ý nghĩa như nhau:

ds[i].ngay_sinh.nam

p[i].ngay_sinh.nam

(p+i)->ngay_sinh.nam

8.5.2. Khi sử dụng cả cấu trúc (chẳng hạn trong phép gán cấu trúc) thì các cách viết sau là tương đương:

ds[i] p[i] *(p+i)

§9. HÀM TRÊN CÁC CẤU TRÚC

9.1. Đối của hàm có thể là:

+ Biến cấu trúc. Khi đó tham số thực tương ứng là một giá trị cấu trúc.

+ Con trỏ cấu trúc. Khi đó tham số thực tương ứng là địa chỉ của biến cấu trúc.

+ Mảng cấu trúc hình thức hoặc con trỏ cấu trúc. Khi đó tham số thực tương ứng là tên mảng cấu trúc.

9.2. Ham có thể trả về giá trị:

+ Giá trị cấu trúc. + Con trỏ cấu trúc.

9.3. Các ví du. Những điều nói trong 9.1 và 9.2 được minh họa trên 2 ví dụ sau.

Ví dụ 9.1. Xét kiểu cấu trúc person gồm 3 thành phần:

+ ht (họ tên) kiểu mảng char

+ ns (năm sinh) kiểu struct date

+ bl (bậc lương) kiểu float

Ta đưa vào 6 hàm thao tác trên kiểu person.

+ Hàm

```
person *ptim(char *ht,person h[],int n);
```

có tác dụng tìm trong danh sách n nhân viên lưu trong mảng h, người có tên cho trong ht. Hàm trả về con trỏ trỏ tới người tìm được hoặc trả về NuLL nếu không tìm thấy.

+ Hàm

person tim(char *ht,person h[],int n);

cũng có tác dụng tìm kiếm như hàm trên, nhưng nó trả về một cấu trúc chứa thông tin người tìm được. Các thông tin này sẽ bằng không nếu không tìm thấy.

+ Hàm

void hv(person *p1, person *p2);

dùng để hoán vị hai cấu trúc.

+ Hàm

void sapxep(person *p,int n);

có tác dụng sắp xếp n phần tử cấu trúc chứa trong p theo thứ tự tăng của năm sinh.Trong hàm sapxep có dùng tới hàm hv.

+ Hàm

void vao(person *p);

dùng để nhập dữ liệu cho một đối tượng kiểu person. Một chú ý quan trọng là: Nếu trong hàm vao ta dùng câu lệnh

scanf("%f%*c",&h.bl);

để nhập trực tiếp vào thành phần h.bl thì bị treo máy.

+ Hàm

void in(person p);

dùng để in một cấu trúc.

Cách thức làm việc của chương trình như sau: Đầu tiên là phần nhập số liệu, rồi đến sắp xếp, sau đó sẽ in danh sách nhân viên đã sắp xếp. Cuối cùng đến các mục tìm kiếm theo hàm ptim và hàm tim. Dưới đây là chương trình nguồn.

```
#include "stdio.h"
#include "conio.h"
#include "string.h"
struct date
    {
        int ngay,thang,nam;
    };
typedef struct
    {
        char ht[25];
        struct date ns;
        float bl;
    } person;
```

```
person *ptim(char *ht,person h[],int n);
person tim(char *ht,person h[],int n);
void hv( person *p1, person *p2);
void sapxep( person *p,int n);
void vao( person *p);
void in( person p);
person tim(char *ht,person h[],int n)
  {
    int i; person ps;
    ps.ns.ngay=ps.ns.thang=ps.ns.nam=0;
    ps.bl=0.0;
    ps.ht[0]=0;
    for(i=1;i<=n;++i)
          if (strcmp(ht,h[i].ht)==0)
            return(h[i]);
    return(ps);
  }
person *ptim(char *ht,person h[],int n)
  {
    int i;
    for(i=1;i<=n;++i)
          if (strcmp(ht,h[i].ht)==0)
            return(&h[i]);
    return(NuLL);
  }
void hv( person *p1, person *p2)
  {
    person h;
    h=*p1;
    *p1=*p2;
    *p2=h;
  }
void vao( person *p)
  {
    person h;
    float bl;
    printf("\nHo ten ");
    gets(h.ht);
    printf("\nNgay thang nam sinh: ");
```

```
        scanf("%d%d%d%%*c",&h.ns.ngay,&h.ns.thang,&h.ns.nam);
        printf("\nBac luong: ");
        scanf("%f%*c",&bl); h.bl=bl;
        *p=h;
    }
    void in( person p)
        {
            printf("\nHo ten %s Sinh ngay %d thang %d nam %d \
            Bac luong %0.1f", p.ht,p.ns.ngay,p.ns.thang,p.ns.nam,p.bl);
        }
    void sapxep( person *p,int n)
        {
            int i,j;
            for(i=1;i<=n-1;++i)
             for(j=i+1;j<=n;++j)
                  if(p[i].ns.nam>p[j].ns.nam) hv(&p[i],&p[j]);
        }
    main()
        {
            person *p,ds[100];
            int n,i,j;
            char ht[40];
            /* vao so lieu */
            clrscr();
            printf("\nSo nguoi n = ");
              scanf("%d%*c",&n);
            for(i=1;i<=n;++i)
              vao(&ds[i]);
                  /* sap xep theo chieu tang cua ns(nam sinh) */
              sapxep(ds,n);
                  /* in danh sach sau khi sap xep */
               for(i=1;i<=n;++i)
                 in(ds[i]);
                  /* tim kiem theo ho ten (dung ham ptim) */
            while(1)
               {
                  printf("\nHo ten nguoi can tim \(bam enter de ket thuc) ");
```

```
                gets(ht);
                if(ht[0]==0)
                   break;
                if( (p=ptim(ht,ds,n))==NuLL)
                      printf("\n Khong tim thay");
                else in(*p);
             }
             /* tim kiem theo ho ten (dung ham tim) */
          while(1)
             {
                printf("\nHo ten nguoi can tim \(bam enter de ket thuc) ");
                gets(ht);
                if(ht[0]==0)
                   break;
                if( tim(ht,ds,n).ht[0]==0)
                      printf("\n Khong tim thay");
                else
                   in(tim(ht,ds,n));
             }
       } /* Ket thuc */
```

Ví dụ 9.2: Xét kiểu cấu trúc sophuc (số phức) gồm 2 thành phần:

+ biến x kiểu float biểu thị phần thực

+ biến y kiểu float biểu thị phần ảo

Ta đưa vào hai hàm thao tác trên kiểu sophuc.

+ Hàm

sophuc cong(sophuc u,sophuc v);

dùng để cộng các giá trị phức u, v. Giá trị của hàm là cấu trúc chứa tổng **u + v**.

+ Hàm

void insp(sophuc u);

dùng để in cấu trúc u. Một điểm đáng lưu ý ở đây là: Có thể dùng hàm này để in trực tiếp tổng của hai giá trị phức cho bởi hàm tổng.

```
#include "stdio.h"
#include "conio.h"
typedef struct
      {
        float x,y;
      } sophuc;
```

```
sophuc cong(sophuc u,sophuc v);
void insp(sophuc u);
    sophuc cong(sophuc u,sophuc v)
        {
          sophuc w;
          w.x=u.x+v.x;
          w.y=u.y+v.y;
          return w;
        }
    void insp(sophuc u)
        {
          printf("(%0.2f,%0.2f)",u.x,u.y);
        }
    main()
      {
         sophuc u,v;
         u.x=6.5;  u.y=-3.6;
         v.x=2.8;v.y=12.1;
         printf("\nso phuc u = ");
         insp(u);
         printf("\nso phuc v = ");
         insp(v);
         printf("\ntong = ");
         insp(cong(u,v));
         getch();
      }
```

§10. CẤP PHÁT BỘ NHỚ ĐỘNG

Giả sử ta cần quản lý các viện. Mỗi viện có nhiều phòng và mỗi phòng có nhiều nhân viên. Khi thiết kế chương trình chúng ta chưa biết có bao nhiêu viện, chưa biết mỗi viện có bao nhiêu phòng và cũng chưa biết mỗi phòng có bao nhiêu nhân viên. Nếu sử dụng mảng (cấp phát bộ nhớ tĩnh), thì ta phải sử dụng số viện tối ta. Mỗi viện phải xem như có cùng một số phòng và mỗi phòng phải xem như có số nhân viên bằng nhau. Như vậy sẽ có rất nhiều vùng nhớ được cấp phát mà không bao giờ dùng đến. Chương trình dưới đây minh họa phương pháp cấp phát bộ nhớ động. Số viện sẽ đúng bằng số viện cần quản lý. Mỗi viện sẽ được cấp phát một vùng nhớ vừa đủ để chứa số phòng thực sự của nó, và mỗi phòng cũng có một vùng nhớ ứng với số nhân viên hiện có của nó.

Nhân viên được mô tả bằng cấu trúc person có 2 thành phần là ht (họ tên), và ns (năm sinh). Phòng được mô tả bởi cấu trúc ppp có 3 thành phần là tenphong (tên phòng), sonhanvien (số nhân viên trong phòng) và nhanvien (con trỏ kiểu person trỏ tới vùng nhớ chứa thông tin các nhân viên trong phòng). Viện được mô tả bằng cấu trúc vvv gồm 3 thành phần là tenvien (tên viện), sophong (số phòng của viện) và phong (con trỏ kiểu ppp trỏ tới vùng nhớ chứa thông tin của các phòng trong viện)

Chương trình gồm hai phần:

+ Phần đầu gồm các thao tác nhập liệu và cấp phát bộ nhớ đặt xen kẽ nhau. Nhập liệu để biết cần cấp phát bao nhiêu. Cấp phát để có vùng nhớ chứa thông tin nhập vào sau đó.

+ Phần tiếp theo đơn giản chỉ là in ra màn hình các thông tin vừa nhập vào.

Dưới đây là chương trình.

```
#include "stdio.h"
#include "conio.h"
#include "alloc.h"
typedef struct
    {
      char ht[25];  /* ho ten */
      int ns;       /* nam sinh */
    } person;
typedef struct
    {
      char tenphong[30];
      int sonhanvien;
      person *nhanvien;
    } ppp;
typedef struct
    {
      char tenvien[30];
      int sophong;
      ppp *phong;
    } vvv;
vvv *vien;
int sovien;
main()
  {
    int i,j,k,ngay,thang,nam,sophong,sonhanvien,ns;
```

```
/* Vao so lieu va cap phat bo nho, so vien */
clrscr();
printf("\nSo vien: ");
scanf("%d%*c",&sovien);
vien=(vvv*) malloc( (sovien+1)*sizeof(vvv));
    /* Vao so lieu tung vien */
for(i=1;i<=sovien;++i)
  {
     printf("\nTen vien thu %d: ",i);
     gets(vien[i].tenvien);
     printf("\nSo phong cua vien %s: ",vien[i].tenvien);
     scanf("%d%*c",&sophong); vien[i].sophong=sophong;
     vien[i].phong=(ppp*)malloc((sophong+1)*sizeof(ppp));
     for(j=1;j<=sophong;++j)
       {
         printf("\n Ten phong %d cua vien %s: ", j,vien[i].tenvien);
         gets(vien[i].phong[j].tenphong);
         printf("\n So nhan vien cua phong %s vien %s: ",
               vien[i].phong[j].tenphong,vien[i].tenvien);
         scanf("%d%*c",&sonhanvien);
               vien[i].phong[j].sonhanvien=sonhanvien;
         printf("\n Vien %s Phong %s",vien[i].tenvien,
               vien[i].phong[j].tenphong);
         for(k=1;k<=sonhanvien;++k)
          {
            printf("\nHo ten nhan vien %d: ",k);
            gets(vien[i].phong[j].nhanvien[k].ht);
            printf("\nNam sinh nhan vien %d: ",k);
            scanf("%d%*c",&ns); vien[i].phong[j].nhanvien[k].ns=ns;
          }
       }
  }
/* Dua ra man hinh */
clrscr();
for(i=1;i<=sovien;++i)
   {
```

```
    printf("\nVien %s co %d phong ", vien[i].tenvien,vien[i].sophong);
    for(j=1;j<=vien[i].sophong;++j)
      {
        printf("\nPhong %s vien %s co %d nhan vien: ",
        vien[i].phong[j].tenphong, vien[i].tenvien,
        vien[i].phong[j].sonhanvien);
        for(k=1;k<=vien[i].phong[j].sonhanvien;++k)
          {
            printf("\nHo ten %s sinh nam %d",
            vien[i].phong[j].nhanvien[k].ht,
            vien[i].phong[j].nhanvien[k].ns);
          }
      }
  }
  getch();
}
```

§11. CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT

Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa gọi là cấu trúc tự trỏ. Dưới đây trình bầy 3 cách định nghĩa cấu trúc tự trỏ person

Cách 1:

```
typedef struct pp
    {
      char ht[25];        /* ho ten */
      char qq[20];        /* que quan */
      int tuoi;
      struct pp *tiep;
    } person;
```

Cách 2:

```
typedef struct pp person;
    struct pp
    {
      char ht[25];
      char qq[20];
      int tuoi;
      person *tiep;
    };
```

Cách 3:

```
struct pp
      {
        char ht[25];
        char qq[20];
        int tuoi;
        struct pp *tiep;
      };
typedef pp person;
```

Cấu trúc tự trỏ được dùng để xây dựng danh sách liên kết (móc nối), đó là một nhóm các cấu trúc có tính chất sau:

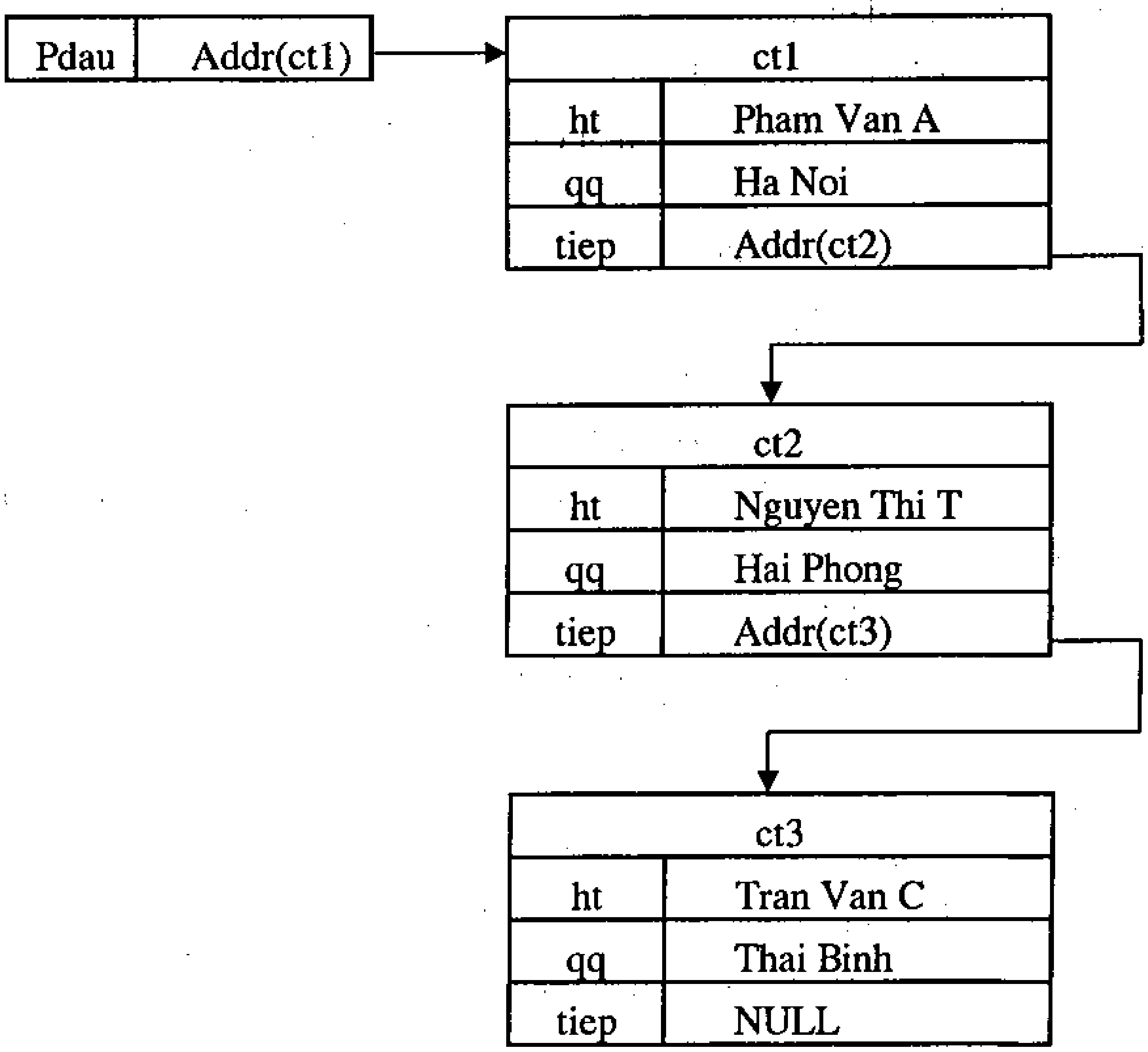
+ Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó (giả sử pdau).

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

+ Cấu trúc cuối chứa hằng NuLL.

Danh sách có 3 tính chất trên gọi là danh sách móc nối theo chiều thuận. Với danh sách này, ta có thể lần lượt truy nhập từ cấu trúc đầu tới cấu trúc cuối theo chiều từ trên xuống dưới.

Hình vẽ dưới đây minh họa một danh sách liên kết thuận.



Tương tự, danh sách liên kết theo chiều ngược cũng có 3 tính chất trên nhưng theo chiều ngược lại:

+ Biết địa chỉ cấu trúc cuối.

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

+ Cấu trúc đầu chứa hằng NuLL.

Với danh sách này, ta có thể lần lượt truy nhập từ cấu trúc cuối tới cấu trúc đầu theo chiều từ dưới lên trên.

Ngoài ra 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 nhập từ trên xuống dưới theo chiều thuận hoặc từ dưới lên trên theo chiều ngược.

Dưới đây là 2 chương trình minh họa cách dùng danh sách móc nối.

Chương trình 1 gồm các phần:

+ Nhập một số người và chứa vào bộ nhớ dưới dạng danh sách móc nối thuận (lập danh sách mới).

+ in danh sách móc nối ra màn hình.

+ Tìm kiếm trên danh sách móc nối.

+ Xóa một người khỏi danh sách

+ Bổ sung vào cuối danh sách

+ Chèn một người vào giữa danh sách

Trước khi xem chương trình, hãy tìm hiểu các thủ thuật làm việc trên danh sách móc nối.

+ Để tạo danh sách mới cần thực hiện các khâu sau:

- Cấp phát bộ nhớ cho một cấu trúc.
- Nhập một người vào vùng nhớ vừa cấp.
- Gán địa chỉ người sau cho thành phần con trỏ của người trước.

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

- Đầu tiên cho p = pdau
- Để chuyển đến người tiếp theo ta dùng phép gán:

p = p->tiep;

- Dấu hiệu để biết đang xét người cuối là:

p->tiep == NuLL

+ Để loại một người khỏi danh sách cần:

- Lưu trữ địa chỉ người cần loại vào một con trỏ (dùng để giải phóng bộ nhớ của người cần loại)

- Sửa để người trước đó có địa chỉ của người đứng sau người mà ta định loại.
- Giải phóng bộ nhớ của người cần loại.

+ Để bổ sung (hoặc chèn) cần
  - Cấp phát bộ nhớ và nhập bổ sung.
  - Sửa thành phần con trỏ trong các cấu trúc có liên quan để đảm bảo mỗi người chứa địa chỉ người tiếp theo.

Chương trình dưới đây sẽ minh họa các kỹ thuật nói trên. Ngoài ra chúng ta còn thấy phương pháp cấp phát bộ nhớ và các cách truy nhập tới thành phần cấu trúc thông qua con trỏ.

```
/* Chuong trinh 1 */
        #include "stdio.h"
        #include "alloc.h"
        #include "conio.h"
        #include "string.h"
        typedef struct pp
          {
            char ht[25];            /* ho ten */
            char qq[20];            /* que quan */
            int tuoi;
            struct pp *tiep;
          } person;
        main()
          {
            int t;
            char ht[25],qq[20];
            person *pdau,*p,*p1;
            clrscr();
              /*
              Vao mot so nguoi va luu tru trong bo
              nho duoi dang mot danh sach lien ket
              */
            pdau=NuLL;
            while(1)
              {
               printf("\nHo ten (bam enter ket thuc vao so lieu): ");
               gets(ht);
```

```
        if (ht[0]==0)
            break;
        if (pdau==NuLL)
          {
            pdau=(person*)malloc(sizeof(person));
            p=pdau;
          }
        else
          {
            p->tiep=(person*)malloc(sizeof(person));
            p=p->tiep;
          }
        strcpy(p->ht,ht);
        printf("\nQue quan: ");
          gets(p->qq);
        printf("\nTuoi: ");
          scanf("%d%*c",&t);
        p->tuoi=t;
        p->tiep = NuLL;
    }
/*
  Dua danh sach lien ket ra man hinh
  biet con tro pdau tro toi dau danh sach
*/
  p=pdau;
  while(p!=NuLL)
    {
      printf("\nHo ten %-25s Que %-30s Tuoi %d",
      (*p).ht,(*p).qq,(*p).tuoi);
      p = p->tiep;
    }
/* Tim kiem theo que quan */
  while(1)
    {
      printf("\nQue (bam enter ket thuc tim kiem): ");
        gets(qq);
      if (qq[0]==0)
          break;
```

```
        /* duyet tu dau danh sach va in
          ra man hinh nhung nguoi tim duoc
        */
        p=pdau;
        while(p!=NuLL)
          {
            if(strcmp(p->qq,qq)==0)
              printf("\nHo ten %-25s Que %-30s Tuoi %d",
              (*p).ht,(*p).qq,(*p).tuoi);
            p=p->tiep;
          }
      }
printf("\nLoai phan tu dau danh sach");
      /* Loai phan tu dau danh sach */
if (pdau!=NuLL)
    {
      p=pdau;
      pdau=p->tiep; /* pdau tro toi nguoi thu hai */
      free(p);       /* giai phong vung nho cua nguoi dau */
    }
printf("\nBo sung mot nguoi vao cuoi danh sach ");
                    /* Bo sung mot nguoi vao cuoi danh sach */
if(pdau==NuLL)      /* Danh sach rong */
    {
      pdau=(person*)malloc(sizeof(person));
      p=pdau;
    }
else
    {
          /* tim dia chi cuoi va dat vao p */
      p=pdau;
      while(p->tiep!=NuLL)
            p=p->tiep;
          /* cap phat vung nho va nhap bo sung */
      p->tiep=(person*)malloc(sizeof(person));
      p=p->tiep;
    }
```

```
/* Nhap bo sung */
printf("\nHo ten: ");
gets(p->ht);
printf("\nQue quan: ");
gets(p->qq);
printf("\nTuoi: ");
scanf("%d%*c",&t);
p->tuoi=t;
p->tiep=NuLL;
printf("\nChen them vao truoc nguoi thu hai");
   /* Chen them mot nguoi vao truoc nguoi thu hai
   gia thiet co it nhat hai nguoi, truoc tien cap phat
   vung nho cho p de chua nguoi bo sung */
p=(person*)malloc(sizeof(person));
printf("\nHo ten: ");
gets(p->ht);
printf("\nQue quan: ");
gets(p->qq);
printf("\nTuoi: ");
scanf("%d%*c",&t); p->tuoi=t;
/* ket noi */
p1=pdau->tiep;       /* p1 chua dia chi nguoi thu 2 */
pdau->tiep=p;        /* sua de nguoi dau chua dia chi
                        nguoi moi bo sung */
p->tiep=p1;          /* nguoi moi bo sung chua dia chi
                        nguoi thu 2 trong danh sach cu */
                     /* Dua danh sach moi ra man hinh
                     biet con tro pdau tro toi dau danh sach */
p=pdau;
while(p!=NuLL)
   {
      printf("\nHo ten %-25s Que %-30s Tuoi %d",
      (*p).ht,(*p).qq,(*p).tuoi);
      p = p->tiep;
   }
   getch();
}
```

Chương trình 2 cũng nhập một số người và tạo thành một danh sách liên kết, nhưng phức tạp hơn chương trình 1 ở chỗ: Người vừa nhập được chèn vào vị trí thích hợp để danh sách được sắp theo chiều tăng của tuổi. Như vậy ta có quy trình để vừa nhập liệu vừa sắp xếp một cách đồng thời. Chương trình dùng hai hàm. Hàm

void vao_sl(char *ht,person **p);

dùng để gán ht (họ tên) vào p->ht, gán NuLL cho p->tiep và nhập quê quán gán cho p->qq, nhập tuổi gán cho p->tuoi. Hàm

void bo_sung(person *pdau,person ng,person **pchen,int *vt);

dùng để xác định xem cấu trúc ng cần chèn vào đâu. Nếu vt = 0 thì ng cần chèn vào đầu danh sách. Còn nếu vt = 1 thì ng cần chèn vào ngay sau phần tử được pchen trỏ tới.

Chương trình 2 chẳng những minh họa các thủ thuật làm việc trên danh sách móc nối mà còn cho những ví dụ hay về việc dùng các hàm có đối con trỏ.

```
/* Chuong trinh 2 */
#include "stdio.h" #include "alloc.h" #include "conio.h"
#include "string.h"
typedef struct pp
    {
      char ht[25]; /* ho ten */
      char qq[20]; /* que quan */
      int tuoi;
      struct pp *tiep;
    } person;
void bo_sung(person *pdau,person ng,person **pchen,int *vt);
  /*
    y nghia ham bo_sung:
    Biet:
    + pdau tro toi dau danh sach (gia thiet pdau!= NuLL)
    + ng la cau truc chua nguoi vua nhap
    Ham cho biet:
    + Neu vt=0 chen vao dau danh sach
    + Neu vt=1 chen vao sau cau truc do pchen tro toi */
void vao_sl(char *ht,person **p);
  /* Ham vao_sl dung de:
  + Cap phat vung nho cho p
  + Gan ht vao p->ht
  + Vao cac so lieu khac nhu que quan va tuoi */
```

```
void bo_sung(person *pdau,person ng,person **pchen,int *vt)
  {
    person *p=pdau,*p1;
    if (ng.tuoi < p->tuoi)
     {
       *vt=0;
       return;
     }
     else
       while (1)
        {
           p1=p->tiep;
           if ((p1==NuLL)||(p1!=NuLL && ng.tuoi<p1->tuoi))
             {
               *pchen=p;
               *vt=1;
               return;
             }
        p=p1;
       }
  }
  void vao_sl(char *ht,person **p)
   {
     int t;
     person *pp;
     pp= (person*)malloc(sizeof(person));
     strcpy(pp->ht,ht);
     printf("\nQue quan: ");
     gets(pp->qq);
     printf("\nTuoi: ");
     scanf("%d%*c",&t);
     pp->tuoi=t;
     pp->tiep=NuLL;
     *p=pp;
   }
```

```
main()
{
    int vt;
    char ht[25];
    person *pdau,*p,*ptg,*pchen;
    clrscr();
            /* Vao mot so nguoi va luu tru trong bo nho
               duoi dang mot danh sach lien ket */
    pdau=NuLL;
    while(1)
     {
        printf("\nHo ten (bam enter ket thuc vao so lieu): ");
        gets(ht);
        if (ht[0]==0)  break;
        if (pdau==NuLL)
         vao_sl(ht,&pdau);
        else
           {
             vao_sl(ht,&ptg);
             bo_sung(pdau,*ptg,&pchen,&vt);
             if(vt==0)   /* Chen vao dau danh sach */
                {
                  ptg->tiep = pdau;  pdau = ptg;
                }
           else
             {    /* chen vao sau pchen */
                ptg->tiep = pchen->tiep;  pchen->tiep= ptg;
             }
           }
     }
   /* Dua danh sach lien ket ra man hinh biet
   con tro pdau tro toi dau danh sach */
    p=pdau;
    while(p!=NuLL)
      {
        printf("\nHo ten %-25s Que %-30s Tuoi %d",
               (*p).ht,(*p).qq,(*p).tuoi);
       p=p->tiep;
      }
    getch();
}
```

§12. KIỂU UNION (HỢP)

12.1. union là gì. 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.

12.2. Khai báo. Việc định nghĩa một kiểu union, việc khai báo biến union, mảng union, con trỏ union, cũng như cách truy nhập đến các thành phần union được thực hiện hoàn toàn tương tự như đối với cấu trúc. Một cấu trúc có thể có thành phần kiểu union, ngược lại các thành phần của union lại có thể là cấu trúc. Dưới đây là hai ví dụ về khai báo union.

12.3. Các ví dụ về hợp.

Ví dụ 1:

```
typedef union
    {
      unsigned int ival;
      float fval;
      unsigned char ch[2];
    } val;
val  a,b,x[10];
```

Ví dụ trên định nghĩa kiểu union val gồm 3 thành phần là ival, fval và ch. Độ dài của val bằng độ dài của fval và bằng 4. Tiếp đó khai báo các biến union a, b và mảng union x

Phép gán:

a.ival = 0xa1b2;

sẽ gán một số hệ 16 cho ival. Do ival và ch cùng chiếm 2 byte đầu của union, nên sau câu lệnh trên thì ta 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 2: Dưới đây minh họa cách khai báo một union có thành phần cấu trúc.

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

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

Ta kết thúc mục này bằng một chương trình minh họa cách dùng union và cấu trúc để tạo ra các biến có thể lưu trữ nhiều kiểu dữ liệu khác nhau (như trong foxpro). Biến cấu trúc kiểu stdata gồm thành phần type để nhận biết kiểu và thành phần v (union) để lưu trữ giá trị. Type có thể nhận một trong các ký tự sau:

'C' (Chuỗi ký tự)

'N' (Số thực float)

'L' (Logic với các giá trị T t y y F f N n)

'D' (Kiểu date dd/mm/yy)

Trong chương trình dùng hai hàm:

Hàm:

```
stdata store_data(char type,char *data);
```

dùng để biến đổi dữ liệu cho bởi type và data thành một trong bốn kiểu nêu trên và chứa vào một cấu trúc kiểu stdata. Cấu trúc này chính là giá trị trả về của hàm.

Hàm:

```
void print_data(stdata x);
```

dùng để in ra màn hình giữ liệu đang lưu trữ trong biến x (kiểu stdata). Chương trình gồm các nội dung:

+ Vào dữ liệu dạng (type,data) từ bàn phím

+ Đổi từ dạng (type,data) ra một trong các kiểu chuỗi ký tự, số thực, logic và date. Kết qủa được lưu trong một biến kiểu stdata

+ in ra mành hình dư liệu đang lưu trong biến kiểu stdata.

Dưới đây là chương trình.

```
#include "stdio.h"
#include "conio.h"
#include "ctype.h"
#include "string.h"
#include "stdlib.h"
typedef struct
    { unsigned day: 5;     /* ngay */
      unsigned month: 4;  /* thang */
      unsigned year: 7;     /* nam */
    } date;
```

```
typedef struct
        {
          char type;
          union
             {
                char *cval;  /* Gia tri kieu char */
                float fval;   /* Gia tri kieu float */
                date dval;   /* Gia tri kieu date */
                int lval;     /* Gia tri kieu logic */
             } v;
        } stdata;
/* Các hàm */
stdata store_data(char type,char *data);
void print_data(stdata x);
stdata store_data(char type,char *data)
   {
       stdata x;  char *p;
       switch(x.type=type)
         {
            case 'C':
            x.v.cval=strdup(data); break;
            case 'N':
            x.v.fval=atof(data); break;
            case 'D':
                 /* data = "dd/mm/yy"
                 neu trai lai xem la 1-1-95 */
            if( (p=strchr(data,'/'))!= NuLL)
               {
                 *p++ = '\0';
                 x.v.dval.day = atoi(data);       /* Ngay */
                 data=p;                          /* data = "mm/yy" */
                 if( (p=strchr(data,'/'))!= NuLL)
                    {
                       *p++ = '\0';
                       x.v.dval.month = atoi(data);
                       x.v.dval.year = atoi(p); break;
                    }
           }
```

```
        /* Truong hop data khong co dang dd/mm/yy */
          x.v.dval.day = 1; x.v.dval.month = 1;
          x.v.dval.year=95;
          break; case 'L':
           /* data phai la mot trong cac chu y y N n hoac T t F f */
          if ( strchr("yyTt",data[0])!= NuLL )
          x.v.lval=1;        /* TRue */
          else x.v.lval=0;   /* FaLSe */
        }
        return x;
    }
void print_data(stdata x)
    {
        char *p;
        printf("\nDu lieu duoc luu tru duoi dang: ");
        switch (x.type)
           {
             case 'C': printf("Character: %s",x.v.cval); break;
             case 'N': printf("Numeric: %.2f",x.v.fval); break;
             case 'D': printf("Date: %02u/%02u/%02u",
             x.v.dval.day,x.v.dval.month,x.v.dval.year); break;
             case 'L': printf("Logic: %s", x.v.lval? "TRue": "FaLSe" );
           }
        }
        main()
         {
             char buf[256];  char type;  stdata x;
             tt:
             printf("\nHay nhap du lieu: "); gets(buf);
             printf("\nKieu: C(char) N(numeric) D(date) L(logic) ");
             type = getche();
                type = toupper(type);   /* Doi ra chu hoa */
                              /* Neu type khong hop le cho bang C */
             if ( strchr("CNDL",type)==NuLL) type='C';
                x = store_data (type,buf); /* Luu tru du lieu vao cau truc x */
                print_data(x);             /* in du lieu tu x */
             printf("\nCo tiep tuc? - C/K ");
             if (toupper(getch())=='C') goto tt;
         }
```

BÀI TẬP CHƯƠNG 7

Bài 1.

1. 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 và tỉnh. Điểm thi là một cấu trúc gồm ba thành phần: toán, lý, hóa (điểm chấm chính xác đến 1/4).

2. Đọc 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 giấy.

Bài 2.

1. Xây dựng một mảng cấu trúc mà mỗi thành phần của nó có kiểu như cấu trúc ở bài một .

2. Nhập số liệu của 20 phiếu điểm và lưu trữ vào mảng cấu trúc nói trên.

3. Tìm kiếm và in ra các thí sinh có tổng số điểm ba môn lớn hơn 15.

Bài 3.

Giả sử đã nhập số liệu của 20 phiếu điểm theo như yêu cầu của bài 2. Hãy lập 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 (theo thứ tự nói trên). Mỗi thí sinh sẽ in trên một dòng gồm các thông tin:

- Họ tên
- Quê quán
- Số báo danh
- Điểm toán, lý, hóa.

Bài 4. Một hội ái hữu quản lý hội viên của mình như sau: Mỗi hội viên có hai thông tin chung là họ tên và địa chỉ. Ai đang có vợ thì khai thêm họ tên vợ và ngày cưới. Ai có người yêu thì khai họ tên và số điên thoại của người yêu. Hãy lập chương trình để:

1. Nhập n hội viên.

2. In thiếp mời các hội viên chưa có gia đình.

(dùng union)

Bài 5.

Trong một trường trung học, hoc sinh bắt buộc phải học 3 môn toán, lý và hoá. Ngoài ra học sinh nam học thêm môn kỹ thuật còn học sinh nữ học thêm môn nữ công. Viết chương trình:

1. Nhập họ tên, giới tính và điểm của n học sinh.
2. In số liệu về học sinh nam trước rồi đến các học sinh nữ.

(dùng union)

Bài 6.

Nhập danh sách n học sinh với các thuộc tính: họ tên, năm sinh và tổng điểm. Sắp xếp danh sách theo thứ tự giảm của tổng điểm. Khi tổng điểm như nhau thì học sinh có năm sinh nhỏ hơn được xếp trước. In danh sách học sinh đã sắp xếp sao cho tất cả các chữ cái của họ tên chuyển thành chữ hoa.

Bài 7.

Cho một danh sách liên kết gồm các cấu trúc kiểu ts được định nghĩa như sau:

```
struct ts /* Kiểu thí sinh */
  {
    char ht[26];          /* Họ tên */
    float td;             /* Tổng điểm */
    struct ts *tiep;      /* Trỏ tới thí sinh sau */
  };
```

Cho biết con trỏ pdau trỏ tới đầu danh sách. Lập đoạn chương trình thực hiên các yêu cầu:

1. Bổ sung một thí sinh vào cuối danh sách.

2. Lập một danh sách liên kết mới gồm các thí sinh trúng tuyển (cho biết điểm chuẩn bằng 15).

Bài 8. Cho một danh sách liên kết gồm các cấu trúc kiểu hs:

```
struct hs /* Kiểu học sinh */
  {
    char ht[26];    /* Họ tên */
    int ns;         /* Năm sinh */
    struct hs *tiep; /* Trỏ tới học sinh sau */
  };
```

Cho biết con trỏ pdau trỏ tới đầu danh sách. Lập đoạn chương trình thực hiên các yêu cầu:

1. In các học sinh có năm sinh từ 1972 trở lại đây.

2. Xoá khỏi danh sách các học sinh sinh năm 1974.

Bài 9.

Định nghĩa kiểu cấu trúc tự trỏ hoc_sinh gồm ba thành phần: họ tên, năm sinh và con trỏ kiểu hoc_sinh. Sau đó thực hiện các yêu cầu:

1. Nhập từ bàn phím một số học sinh và chứa vào một danh sách móc nối. Kết thúc việc nhập bằng cách bấm Enter khi chương trình đề nghị vào họ tên của học sinh tiếp theo.

2. In ra màn hình danh sách học sinh theo thứ tự nhập vào từ bàn phím.

3. Như câu 2 nhưng yêu cầu in ra theo thứ tự tăng dần của tuổi (chỉ dùng danh sách móc nối, không dùng mảng).

Bài 10.

Định nghĩa kiểu cấu trúc mô tả đa thức, sau đó viết các hàm vào đa thức, in đa thức, cộng đa thức và nhân đa thức. áp dụng trong hàm main() để thực hiện các việc:

1. Vào từ bàn phím ba đa thức P_1, P_2 và P_3.

Tính đa thức P theo công thức:

$$P = (P_1 + P_2)^2 + P_3$$

In P_1, P_2, P_3 và P.

2. Vào từ bàn phím một dãy n đa thức, sau đó in chúng lên màn hình theo thứ tự giảm của bậc.

CHƯƠNG 8

QUẢN LÝ MÀN HÌNH VÀ CỬA SỔ

Trong chương này sẽ giới thiệu một số hàm dùng để quản lý màn hình màu và tạo cửa sổ. Bằng cách tạo nên các cửa sổ với màu sắc khác nhau sẽ làm cho màn hình trở nên đẹp đẽ sáng sủa hơn, điều này giúp cho việc tổ chức số liệu và tổ chức hội thoại người máy trở nên thuận tiện hơn. Tất cả các hàm quản lý màn hình giới thiệu dưới đây đều khai báo trong tệp conio.h.

§1. CHỌN KIỂU MÀN HÌNH VĂN BẢN

1.1. Trước khi sử dụng các hàm quản lý màn hình ta phải dùng hàm textmode để chọn kiểu màn hình văn bản. Hàm có dạng:

```
void textmode(int mode);
```

Ở đây mode là một biến nguyên dùng để xác định mốt văn bản định sử dụng. Dưới đây là các giá trị mà biến mode có thể nhận và nghĩa của chúng.

Bảng 1

Ký hiệu tượng trưng	**Giá trị số**	**Mốt văn bản video**
LASTMODE	-1	Mốt văn bản trước đó
BW40	0	Đen trắng 40 cột
C40	1	16 màu, 40 cột
BW80	2	Đen trắng 80 cột
C80	3	16 màu, 80 cột
MONO	7	Màn hình đơn sắc, 80 cột

Chú ý 1. Các ký hiệu tượng trưng chính là các hằng đã được định nghĩa trong tệp conio.h.

Chú ý 2. Độc lập với các mốt văn bản được chọn, màn hình luôn luôn gồm 25 hàng.

1.2. Ví dụ.

Ví dụ 1: Câu lệnh

```
textmode(C40);
```

sẽ chọn mốt hiển thị văn bản 16 màu, 40 cột. Như vậy, màn hình sẽ gồm 25 hàng và 40 cột. Mã của các màu được cho trong bảng 2.

Ví dụ 2: Câu lệnh

textmode(C80);

sẽ chọn kiểu hiện thị văn bản 16 màu, 80 cột. Màn hình sẽ gồm 25 hàng và 80 cột. Mỗi dòng trên màn hình chứa được 80 ký tự.

Ta sẽ thấy các ký tự ở mốt C40 có bề ngang lớn hơn 2 lần các ký tự ở mốt C80.

1.3. Cách xác định vị trí trên màn hình. Hệ trục toạ độ trên màn hình nhận điểm ở góc trên bên trái làm điểm gốc, trục hoành là trục nằm ngang chạy từ trái sang phải, trục tung chạy từ trên xuống dưới. Như vậy, đối với màn hình 25 hàng 80 cột thì điểm (1,1) là điểm ở góc trên bên trái, điểm (80,1) là điểm ở góc trên bên phải, điểm (1,25) là điểm ở góc dưới bên trái và điểm (80,25) là điểm ở góc dưới bên phải.

§2. ĐẶT MẦU NỀN VÀ MẦU CHỮ

Để đặt màu nền (màu cửa sổ) ta dùng hàm

void textbackground(int color);

Để đặt màu chữ ta dùng hàm

void textcolor(int color);

Trong cả hai trường hợp color là một biến nguyên chứa mã của màu (xem bảng sau)

Ghi chú: Muốn tạo chữ nhấp nháy ta cộng thêm 128 vào tham số mầu, ví dụ lệnh textcolor(4+128) sẽ tạo dòng chữ nhấp nháp mầu đỏ.

Bảng 2.

Ký hiệu tượng trưng	**Giá trị số** (mã màu)	**Màu chữ hay** màu nền
BLACK (đen)	0	Cả hai
BLUE (xanh da trời)	1	Cả hai
GREEN(xanh lá cây)	2	Cả hai
CYAN (xanh lơ)	3	Cả hai
RED (đỏ)	4	Cả hai
MAGENTA (tím)	5	Cả hai
BROWN (nâu)	6	Cả hai
LIGHTGRAY (xám nhạt)	7	Cả hai
DACKGRAY (xám sẫm)	8	Chỉ cho mầu chữ

LIGHTGBLUE (xanh nhạt)	9	Chỉ cho mầu chữ
LIGHTGREEN (xanh nhạt)	10	Chỉ cho mầu chữ
LIGHTCYAN (xanh nhạt)	11	Chỉ cho mầu chữ
LIGHTRED (đỏ nhạt)	12	Chỉ cho mầu chữ
LIGHTMAGENTA (tím nhạt)	13	Chỉ cho mầu chữ
YELLOW (vàng)	14	Chỉ cho mầu chữ
WHITE (trắng)	15	Chỉ cho mầu chữ

Chú ý 1. Các ký hiệu tượng trưng chính là các hằng được định nghĩa trong tệp conio.h và trong tệp graphis.h . Chúng cũng chính là tên của các mầu (viết theo tiếng Anh).

Chú ý 2. Các hàm trên không làm thay đổi mầu sắc của các cửa sổ và các dòng chữ đã có từ trước trên màn hình.

Chú ý 3. Một câu lệnh textbackground xác định một màu, đó sẽ là mầu của tất cả các cửa sổ được xây dựng sau câu lệnh này và trước một câu lệnh textbackground khác. Tương tự một câu lệnh textcolor xác định một mầu , đó sẽ là màu của các dòng chữ được đưa lên màn hình bằng các câu lệnh cprintf và cscanf viết sau câu lệnh textcolor này và trước một câu lệnh textcolor khác.

Ví dụ 1: Các cửa sổ được xây dựng sau câu lệnh

textbakground(RED);

và trước một câu lệnh textbackground khác sẽ có màu đỏ.

Ví dụ 2: Các dòng ký tự được đưa lên màn hình bằng các câu lệnh cprintf và cscanf viết sau câu lệnh

textcolor(WHITE);

và trước một câu lệnh textcolor khác sẽ có mầu trắng.

§3. XÂY DỰNG CỬA SỔ VÀ SỬ DỤNG CỬA SỔ

Hàm

void window(int xt,int yt,int xd,int yd);

sẽ có hai tác dụng:

1 - Xây dựng một cửa sổ trên màn hình (hình chữ nhật có các cạnh song song với các cạnh của màn hình) trong đó điểm ở góc trên bên trái có toạ độ là (xt,yt) và điểm ở góc dưới bên phải có toạ độ là (xd,yd).

2 - Đưa con trỏ về vị trí góc trên bên trái của cửa sổ vừa xây dựng.

Chú ý . Các biến xt,yt,xd,yd cần thoả mãn điều kiện

xd >= xt và yd >= yt

Ví dụ 1. Ta hãy xem hai câu lệnh

```
textbackground(RED);
window(5,5,35,20);
```

có tác dụng gì ? . Câu lệnh thứ nhất xác định màu nền mới là màu đỏ. Các cửa sổ xây dựng sau câu lệnh này sẽ có màu đỏ. Câu lệnh thứ hai xác định một cửa sổ có điểm trên bên trái là (5,5) và điểm dưới bên phải là (35,20). Thế thì sau câu lệnh thứ hai ta có nhận ngay được một cửa sổ đỏ hay không. Câu trả lời là không. Câu lệnh window không có chức năng xoá bỏ những gì đã có trong phạm vi cửa sổ mới mở. Muốn có màu đỏ trên cửa sổ này ta phải xoá đi tất cả những gì đang có trên cửa sổ đó. Hàm

```
void clrscr(void);
```

đảm nhiệm chức năng này.

Ví dụ 2: Giả sử ta muốn có một màn hình cỡ 25 x 80 (25 hàng 80 cột) màu CYAN. Giữa màn hình là một cửa sổ màu RED. Trong cửa sổ có hai dòng chữ. Dòng chữ bên trên là

Chuc mung nam moi

màu YELLOW và dòng chữ dưới là

Happy new year

màu WHITE.

Chương trình dưới đây sẽ đáp ứng được yêu cầu nêu trên.

```
#include "conio.h"
#include "stdio.h"
main()
  {
    /* Chon mode van ban 16 mau 80 cot */
    textmode(C80);
    /* Lam cho man hinh mau CYAN */
    textbackground(CYAN);
    window(1,1,80,25);
    clrscr();
  /* Tao cua so mau RED */
   textbackground(RED);
```

```
        window(20,8,60,18);
        clrscr();
        /* Dòng chữ màu vàng */
        textcolor(YELLOW);
        cprintf("\n%10c Chúc mừng năm mới",' ');
        /* Dòng chữ màu trắng */
        textcolor(WHITE);
        gotoxy(1,4);
        cprintf("\n\n%10c Happy new year",' ');
    }
```

Ví dụ 3: Xét chương trình

```
#include <stdio.h>
#include <conio.h>
main()
  {
    float r,c,pi=3.14;
    textmode(C40);    /* Chọn mode văn bản 16 màu 40 cot */
        /* Làm cho màn hình có màu GREEN */
    textbackground(GREEN); window(1,1,40,25);
    clrscr();
        /* Tạo cửa sổ màu MAGENTA */
    textbackground(MAGENTA); window(10,8,30,18);
    clrscr();
        /* Cho hiện dòng chữ " Ban kinh = " màu YELLOW */
    textcolor(YELLOW);
    cprintf("\n\n Ban kinh = ");
        /* Chọn bán kính từ bàn phím, lay r = 5 */
        /* So 5 mau RED se hien len man hinh */
    textcolor(RED);  cscanf("%f",&r);
        /* Tinh chu vi */
    c = 2*pi*r;
    textcolor(WHITE);  /* Cho hiện lên dòng kết quả màu WHITE */
    gotoxy(1,4);  cprintf("\n\nChu vi= %f6.2",c);
  }
```

§4. CÁC HÀM printf, scanf, cprintf, cscanf

Khi màn hình đang ở chế độ văn bản ta có thể sử dụng các hàm printf, scanf, cprintf và cscanf để vào số liệu từ bàn phím và đưa kết quả ra màn hình. Điều này đã chỉ ra trong các ví dụ 2 và 3 của §3, ở đây ta muốn nói đến vài điều khác nhau giữa các hàm nói trên.

1. Khi sử dụng các hàm cprintf và cscanf thì các dòng ký tự hiện lên màn hình sẽ có màu xác định bởi câu lệnh textcolor gần nhất. Còn khi sử dụng các hàm printf và scanf thì qui tắc này sẽ không còn đúng nữa. Nói chung các hàm cprintf và cscanf sẽ cho một màn hình đẹp hơn so với các hàm printf và scanf.

2. Khi dùng hàm scanf để nhập số liệu ta có thể tự do sửa chữa khi bấm sai bằng cách sử dụng các phím chức năng. Nhưng điều này lại bị hạn chế khi dùng hàm cscanf.

Để thấy rõ sự khác nhau vừa nêu, cách tốt nhất là hãy thay các hàm cprintf và cscanf bằng các hàm prinf và scanf trong các chương trình của §3, sau đó thực hiện chúng trên máy và so sánh các kết quả nhận được trên màn hình.

3.Sự khác nhau giữa printf và cprintf: Ngoài sự khác nhau về mầu hiển thị như đã nói trên, chúng còn khác nhau về phạm vi hiển thị.

+ Hàm printf có tính toàn màn hình không phụ thuộc vào cửa sổ hiện tại. Như vậy các dòng thông tin do hàm đưa ra có thể nằm ngoài cửa sổ nếu như cửa sổ quá hẹp.

+ Hàm cprintf chỉ làm việc trên cửa sổ hiện tại. Như vậy các dòng thông tin do hàm đưa ra chỉ có thể nằm trong cửa sổ và nếu cửa sổ không đủ chỗ chứa thì một số dòng sẽ bị mất.

§5. CÁC HÀM KHÁC

1. Hàm clrscr: Xoá cửa sổ hiện hành.

+ Dạng hàm:

```
void clrscr(void);
```

+ Công dụng: xoá cửa sổ hiên tại và đưa con trỏ đến góc trên bên trái của cửa sổ. Sau hàm này cửa sổ sẽ có mầu xác định bởi hàm textbackground. Hàm này đã được giới thiệu và sử dụng trong các chương trình ở §3.

2. Hàm clreol: Xoá đến cuối dòng trong cửa sổ.

+ Dạng hàm:

```
void clreol(void);
```

+ Công dụng: xoá mọi ký tự đứng sau con chạy cho tới cuối dòng trong cửa sổ mà không làm thay đổi vị trí của con chạy.

3. Hàm delline: Xoá một dòng trong cửa sổ.

+ Dạng hàm:

```
void delline(void);
```

+ Công dụng: xoá dòng của cửa sổ đang chứa con trỏ.

4. Hàm gotoxy: Di chuyển con trỏ trong phạm vi cửa sổ.

+ Dạng hàm:

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

+ Công dụng: dịch chuyển con chạy đến vị trí (x,y) trong cửa sổ hiện tại. Chú ý toạ độ (x,y) không phải là toạ độ màn hình mà là toạ độ trong cửa sổ, ví dụ điểm (1,1) là điểm trên trái của cửa sổ.

+ **Chú ý:** Để đưa con chạy tới một vị trí bất kỳ của màn hình ta có thể sử dụng chức năng thứ hai của hàm window (xem §3).

Ví dụ câu lệnh

```
window (1,24,1,24);
```

sẽ đưa con chạy tới vị trí (1,24) của màn hình.

5. Hàm wherex: Cho vị trí ngang của con trỏ trong cửa sổ.

+ Dạng hàm:

```
int wherex(void);
```

+ Công dụng: cho vị trí ngang (x) của con trỏ trong cửa sổ hiện hành.

6. Hàm wherey: Cho vị trí dọc của con trỏ trong cửa sổ.

+ Dạng hàm:

```
int wherey(void);
```

+ Công dụng: cho vị trí doc (y) của con trỏ trong cửa sổ hiện hành.

7. Hàm gettextinfo: Cho thông tin về kiểu hiển thị văn bản.

+ Dạng hàm:

```
void gettextinfo(struct text_info *r);
```

+ Đối: r là con trỏ trỏ tới địa chỉ của một biến cấu trúc kiểu text_info được định nghĩa trong conio.h như sau:

```
struct text_info
        {
```

```
    unsigned char winleft, wintop;
    unsigned char winright, winbottom;
    unsigned char attribute, normattr;
    unsigned char currmode;
    unsigned char screenheight;
    unsigned char screenwidth;
    unsigned char curx, cury;
};
```

+ Công dụng: Gửi các thông tin có liên quan đến kiểu hiển thị màn hình văn bản đang sử dụng vào các thành phần của biến cấu trúc do con trỏ r trỏ tới. Các thông tin này thương dùng để khôi phục kiểu màn hình văn bản ban đầu.

§6. VÀI VÍ DỤ

Ví dụ 1: Chương trình dưới đây sẽ vẽ lên màn hình hai cửa sổ. Thông tin về mỗi cửa sổ được đưa vào từ bàn phím gồm: màu, toạ độ của điểm trên bên trái và toạ độ của điểm dưới bên phải. Làm theo sự hướng dẫn của chương trình ta có thể tiếp tục vẽ hay kết thúc chương trình.

```
#define mh(x) textbackground(x)
#include "stdio.h"
#include "conio.h" #include "ctype.h"
main()
  {
    int m,xt,xd,yt,yd,sg,xt2,yt2,xd2,yd2,m2;
    /*Chon mot man hinh 16 mau, co 25 x 80 */
    textmode(C80);
    /* Man hinh mau CYAN chu mau WHITE */
    mh(3);
    textcolor(15);
    window(1,1,80,25);
    clrscr();
    /* Dua con tro ve vi tri (1,1) chuan bi
    nap thong tin ve hai cua so */
    tt:
    window(1,1,50,10);
```

```
clrscr();
cprintf(" cua so thu 1: mau xt,yt xd,yd");
cscanf("%d%d%d%d%d", &m,&xt,&yt,&xd,&yd);
cprintf("\n\n\n cua so thu 2: mau xt,yt xd,yd \n");
cscanf("%d%d%d%d%d",&m2,&xt2,&yt2,&xd2,&yd2);
/* Mo cua so thu 1 */
mh(m);
window(xt,yt,xd,yd);
clrscr();
        /* Dua con tro ve vi tri (1,22)
Tiep tuc hay ket thuc */
window(1,22,80,22);
clrscr();
cprintf("Co tiep tuc khong - C/K");
sg=getch();
if(toupper(sg)=='C') goto tt;
        /* Tro ve mot den trang 80 cot
    Xoa man hinh, dua con tro ve vi tri (1,1) */
textmode(2); window(1,1,80,25);
clrscr();
}
```

Ví dụ 2: Xét bài toán vào số liệu của một danh sách cán bộ. Số liệu được tổ chức dưới dạng mảng cấu trúc. Thông tin về mỗi người gồm: họ tên, tuổi và lương. Để cho việc nhập số liệu được tiện lợi ta xây dựng trên màn hình bốn cửa sổ. Cửa sổ thứ nhất màu đỏ trong có ghi tổng số cán bộ. Cửa sổ thứ hai màu tím sẽ chứa mã (số thứ tự) của cán bộ đang xét. Cửa sổ thứ ba màu nâu dùng để ghi họ tên . Cửa sổ thứ tư màu nâu dùng để ghi tuổi và bậc lương. Sau khi vào số liệu sẽ khôi phục kiểu màn hình văn bản ban đầu rồi in dữ liệu ra màn hình. Chương trình dưới đây đáp ứng được các yêu cầu đề ra.

```
#include"conio.h"
#include"stdio.h"
main()
  {
  struct text_info h;
     struct person
```

```
{
    char ht[25];
    int tuoi;
    float luong;
  } p[100];
int n,i,t; float l;
/* lưu kiểu hiển thị văn bản hiện tại */
gettextinfo(&h);
/* chon mot man hinh 16 mau 25 x 80 */
textmode(C80);
/* Man hinh mau CYAN, chu mau WHITE */
textbackground(CYAN);
textcolor(WHITE);
window(1,1,80,25);
clrscr();
/* Cua so thu 1 mau RED trong do ghi so nguoi */
textbackground(RED);
window(1,1,30,4);
clrscr();
cprintf("\n so nguoi = ? ");
scanf("%d",&n);
for(i=0;i<n;++i)
  {
   /* Cua so thu 2 mau MAGENTA trong
   do ghi nguoi đang xét */
   textbackground(MAGENTA);
   window(1,6,30,9);
   clrscr();
   cprintf("\n nguoi thu %d",i+1);
   /* Cua so thu 3 mau BROWN trong do ghi ho ten */
   textbackground(BROWN);
   window(1,12,38,15);
   clrscr();
   cprintf("\n ho ten: ");
```

```
          fflush(stdin); gets(p[i].ht);
          /* Cua so thu 4, mau BROWN trong do ghi tuoi va luong*/
          window(1,17,30,20);
          clrscr();
          cprintf("\nTuoi va luong: ");
          scanf("%d%f",&t,&l);
          p[i].tuoi=t;
          p[i].luong=l;
      }         /* Ket thuc qua trinh vao so lieu */
     /* Khôi phục kiểu màn hình văn bản ban đầu */
       textmode(h.currmode);
       textbackground( h.attribute/16);
       textcolor( h.attribute%16);
       clrscr();
              /* in dữ liệu */
       for(i=0;i<n;++i)
         {
           printf("\n Ho ten: %s",p[i].ht);
           printf("\ntuoi: %3d\nLuong: %8.2f",
           p[i].tuoi,p[i].luong);
         }
    }
```

Ví dụ 3: Chương trình trong ví dụ 2 sử dụng mốt văn bản C80. Chương trình dưới đây cũng giải quyết bài toán đặt ra trong ví dụ 2 nhưng sử dụng mốt văn bản C40. Ta sẽ thấy các ký tự trong mốt C40 to hơn khoảng hai lần so với các ký tự trong mốt C80.

```
#include"conio.h"
#include"stdio.h"
main()
  {
     struct text_info h;
     struct person
        {
           char ht[25];
           int tuoi;
           float luong;
        } p[100]; int n,i,t;
```

```
    float l;
    /* Lưu mốt văn bản hiện tại */
    gettextinfo(&h);
    /* chon mot man hinh 16 mau 25 x 80 */
    textmode(C40);
    /* Man hinh mau CYAN, chu mau WHITE */
    textbackground(CYAN);
    textcolor(WHITE);
    window(1,1,40,25);
    clrscr();
    /* Cua so thu 1 mau RED trong do ghi so nguoi */
    textbackground(RED);
    window(1,1,30,4);
    clrscr();
    cprintf("\n so nguoi = ? ");
    scanf("%d",&n);
for(i=0;i<n;++i)
  {
        /* Cua so thu 2 mau MAGENTA trong
              do ghi nguoi đang xét */
    textbackground(MAGENTA);
    window(1,6,30,9);
    clrscr();
    cprintf("\n nguoi thu %d",i+1);
     /* Cua so thu 3 mau BROWN trong do ghi ho ten */
    textbackground(BROWN);
    window(1,12,38,15);
    clrscr();
    cprintf("\n ho ten: ");
    fflush(stdin); gets(p[i].ht);
    /* Cua so thu 4, mau BROWN trong do ghi tuoi va luong*/
    window(1,17,30,20);
    clrscr();
    cprintf("\nTuoi va luong: ");
    scanf("%d%f",&t,&l);
```

```
        p[i].tuoi=t;
        p[i].luong=l;
      }       /* Ket thuc qua trinh vao so lieu */
    /* Khôi phục mốt văn bản ban đầu */
  textmode(h.currmode);
  textbackground(h.attribute/16);
  textcolor(h.attribute%16);
  clrscr();
        /* in dữ liệu */
  for(i=0;i<n;++i)
    {
      printf("\n Ho ten: %s",p[i].ht);
      printf("\ntuoi: %3d\nLuong: %8.2f",
      p[i].tuoi,p[i].luong);
    }
}
```

§7. BÀI TOÁN THÁP HÀ NỘI

Chương trình trong ví dụ 2 mục §9.3 chương 6 cho quy trình chuyển tháp dưới dạng một văn bản hướng dẫn. Chương trình dưới đây sẽ mô phỏng bằng hình ảnh quá trình chuyển tháp. Ta se nhìn thấy các tầng được di chuyển như thế nào và tổng số là bao nhiêu lần di chuyển. Các tầng được thể hiện theo các mầu khác nhau.

+ Tổ chức dữ liệu:

- Dùng mảng a để chứa độ dài các tầng (còn gọi là khối), số khối m<=12.
- Dùng mảng mau để chứa mã mầu của các tầng.
- Dùng biến t để chứa tốc độ di chuyển.

+ Các hàm được sử dụng:

- tao_khoi(int x,int y,int k) dùng để tạo khối thứ k tại vị trí (x,y).
- xoa_khoi(int x,int y,int k) dùng để xoá khối thứ k tại vị trí (x,y).
- dung_thap(int x,int y,int m) dùng để dựng tháp m khối (tầng) tại vị trí (x,y), đó cũng là vị trí của tầng m.
- chuyen_khoi(int x1,int y1,int x2,int y2,int k) dùng để chuyển khối thứ k từ (x1,y1) đến (x2,y2).
- chuyen_thap(int x1,int y1,int x2,int y2,int x3,int y3,int m) dùng để chuyển tháp m tầng từ vị trí (x1,y1) đến (x2,y2) dùng vị trí trung gian (x3,y3).

+ Các sử dụng chương trình: Chương trình yêu cầu nạp vào từ bàn phím số tầng m và tốc độ chuyển động t (t càng nhỏ thì chuyển động càng nhanh). Sau khi nạp số liêu ta sẽ thấy hiện lên một tháp và máy tạm dừng. Bấm Enter thì bắt đầu di chuyển các tầng. Có thông báo số lần di chuyển.

```
/* Thap Ha Noi
   Minh hoa:
   1. De quy
   2. Lam mat con tro bang cach cho mau nen = mau chu
   3. Dung bien static cuc bo de theo roi so lan chuyen khoi */
      #include "stdio.h"
      #include "conio.h"
      #include "dos.h"
      #include "stdlib.h"
      #define di_chuyen(x,y,k) tao_khoi(x,y,k);  delay(t);
      xoa_khoi(x,y,k)
      int t;         /* Toc do di chuyen */
      int a[]={0,3,5,7,9,11,13,15,17,19,21,23,25};  /*Do dai khoi */
      int mau[]={0,1,2,3,4,5,6,7,9,10,11,12,13};     /*Mau cua khoi */
      int mau_nen=0;
      void tao_khoi(int x,int y,int k); void xoa_khoi(int x,int y,int k);
      void dung_thap(int x,int y,int m);
      void chuyen_khoi(int x1,int y1,int x2,int y2,int k);
      void chuyen_thap(int x1,int y1,int x2,int y2,int x3,int y3,int m);
      void tao_khoi(int x,int y,int k)
        {
          int h=a[k]/2;
          window(x-h,y,x+h,y);
          textbackground(mau[k]);
          textcolor(mau[k]);clrscr();
        }
      void xoa_khoi(int x, int y, int k)
        {
          int h=a[k]/2;
          window(x-h,y,x+h,y);
          textbackground(mau_nen); clrscr();
        }
```

```
void dung_thap(int x,int y,int m)
  {
    int i;
    for(i=m;i>=1;--i) tao_khoi(x,y-m+i,i);
  }
void chuyen_khoi(int x1,int y1,int x2,int y2,int k)
  {
  /* Dua vao bien static de xac dinh so lan chuyen khoi*/
      static sl_chuyen=0; /* Lenh nay chi thuc hien mot lan
      khi dich */
    int x,y;
    sl_chuyen++;window(1,1,24,1);
    textbackground(BLUE);clrscr();
    textcolor(WHITE);cprintf("Lan di chuyen thu: ");
    textcolor(YELLOW+BLINK);cprintf("%4d",sl_chuyen);
    for(y=y1;y>=4;--y)              { di_chuyen(x1,y,k);}
     if(x1<x2)
       for(x=x1;x<=x2;++x)   { di_chuyen(x,4,k);}
     else
       for(x=x1;x>=x2;--x)   { di_chuyen(x,4,k);}
       for(y=4;y<=y2;++y)   { di_chuyen(x2,y,k);}
       tao_khoi(x2,y2,k);
  }
void chuyen_thap(int x1,int y1,int x2,int y2,int x3, int y3,int m)
  {
    if (m<1) return;
    else if(m==1)
    chuyen_khoi(x1,y1,x2,y2,1);
    else
     {
        chuyen_thap(x1,y1-1,x3,y3,x2,y2,m-1);
        chuyen_khoi(x1,y1,x2,y2,m);
        chuyen_thap(x3,y3,x2,y2-1,x1,y1,m-1);
     }
  }
```

```
/* HAM MAIN() */
main()
  {
    int m;
    int x1,y1,x2,y2,x3,y3;
    x1=13;
    x2=40;
    x3=67;
    y1=y2=y3=24;
    textmode(C80);
    clrscr();
    printf("\n So tang cua Thap m (1<m<13): ");
    scanf("%d",&m);
    if(m<2||m>12) exit(1);
    printf("\n Toc do di chuyen t (0<1<1000): ");
    scanf("%d", &t);
    textbackground(mau_nen); clrscr();
    dung_thap(x1,y1,m);  getch();
    chuyen_thap(x1,y1,x2,y2,x3,y3,m); getch();
    window(1,1,80,25);
    textbackground(mau_nen);
    textcolor(WHITE); clrscr();
    system("cls");
  }
```

BÀI TẬP CHƯƠNG 8

Bài 1.

Thực hiện các chương trình đã nêu trên máy. Đối chiếu những điều nói trong sách với các kết quả nhận được.

Bài 2.

Trong các chương trình nói trên hãy thay hàm scanf bằng hàm cscanf và ngược lại. Thực hiện các chương trình nhận được trên máy. Từ đó rút ra ưu điểm và nhược điểm của mỗi hàm.

CHƯƠNG 9

ĐỒ HỌA

Trong chương này sẽ giới thiệu các hàm dùng để vẽ các đường và hình cơ bản như đường tròn, cung elip, hình quạt, đường gãy khúc, hình đa giác, đường thẳng, đường chữ nhật, hình chữ nhật, hình hộp chữ nhật, ... Ngoài ra còn đề cập tới các vấn đề rất lý thú khác như: Xử lý văn bản trên màn hình đồ họa, cửa sổ và kỹ thuật tạo ảnh di động. Các hàm đồ họa được khai báo tệp graphics.h.

§1. KHÁI NIỆM ĐỒ HỌA

Để hiểu kỹ thuật lập trình đồ họa, đầu tiên phải hiểu các yếu tố cơ bản của đồ họa. Từ trước đến nay chúng ta chủ yếu làm việc với kiểu văn bản. Nghĩa là màn hình được thiết lập để hiển thị 25 dòng, mỗi dòng có thể chứa 80 ký tự. Trong kiểu văn bản, các ký tự hiển thị trên màn hình đã được phần cứng của máy PC ấn định trước và ta không thể nào thay đổi được kích thước, kiểu chữ.

Ở màn hình đồ họa, ta có thể xử lý đến từng chấm nhỏ (pixel) trên màn hình và do vậy muốn vẽ bất kỳ thứ gì cũng được. Sự bài trí và số pixel trên màn hình được gọi là độ phân giải (resolution). Do mỗi kiểu màn hình đồ họa có một cách xử lý đồ họa riêng nên TURBO C cung cấp một tệp tin điều khiển riêng cho từng kiểu đồ họa. Bảng 1 cho thấy các kiểu đồ họa và các tệp tin điều khiển chúng.

Ngoài các tệp có đuôi BGI chứa chương trình điều khiển đồ họa, TURBO C còn cung cấp các tệp tin đuôi CHR chứa các Font chữ để vẽ các kiểu chữ khác nhau trên màn hình đồ họa. Đó là các tệp:

GOTH.CHR
LITT.CHR
SANS.CHR
TRIP.CHR

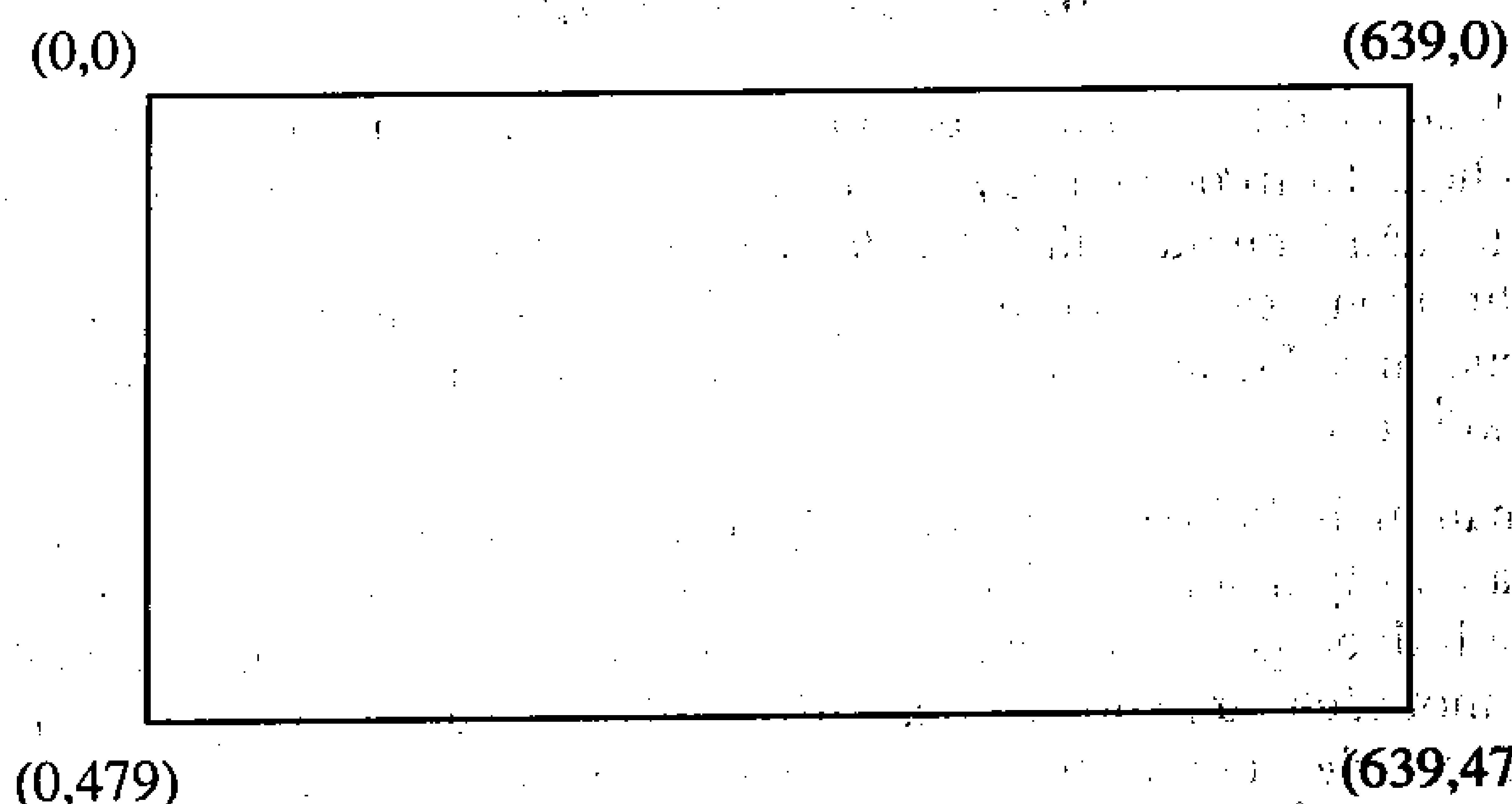
Bảng 1. Các tệp tin điều khiển đồ họa của TURBO C

Tên tệp tin	**Kiểu màn hình đồ họa**
ATT.BGI	ATT & T6300 (400 dòng)
CGA.BGI	IBMCGA, MCGA và các máy tương thích
EGAVGA.BGI	IBM EGA, VGA và các máy tương thích

HERC.BGI	Hercules monochrome và các máy tương thích
IBM8514.BGI	IBM 8514 và các máy tương thích
PC3270.BGI	IBM 3270 PC

Màn hình đồ họa gồm nhiều điểm ảnh được sắp xếp trên các đường thẳng nằm ngang và thẳng đứng. Điều này đúng cho tất cả các kiểu màn hình đồ họa của máy tính. Khác biệt chủ yếu giữa chúng là kích thước và số các điểm ảnh. Trong kiểu CGA (độ phân giải thấp), điểm ảnh có kích thước lớn, chiều ngang có 320 điểm ảnh, còn theo chiều đứng có 200 điểm ảnh. Màn hình VGA có độ phân giải cao hơn: Điểm ảnh nhỏ hơn, trên mỗi hàng có 640 điểm ảnh và trên mỗi cột có 480 điểm ảnh. Điểm ảnh càng nhỏ thì số điểm ảnh trên màn hình càng nhiều và chất lượng đồ họa càng cao.

Mỗi kiểu đồ họa dùng một hệ tọa độ riêng. Hệ tọa độ cho màn hình VGA là 640 x 480 (hình 1)



Hình 1. Hệ tọa độ VGA

Nhờ hệ tọa độ này, ta có thể tác động hay tham chiếu đến bất kỳ điểm ảnh nào trên màn hình đồ họa.

Nếu dùng màn hình CGA thì góc phải dưới có tọa độ (319, 199). Độc lập với kiểu đồ họa đang sử dụng, các hàm getmaxx và getmaxy bao giờ cũng cho tọa độ x và y lớn nhất trong kiểu đồ họa đang dùng.

Một chương trình đồ họa thường gồm các phần sau:

- Khởi động hệ thống đồ họa.
- Xác định mầu nền (mầu màn hình), mầu đường vẽ, mầu tô và kiểu (mẫu) tô.
- Vẽ, tô mầu các hình mà ta mong muốn.
- Các thao tác đồ họa khác như cho hiện các dòng chữ...
- Đóng hệ thống đồ họa để trở về mode văn bản.

§2. KHỞI ĐỘNG HỆ ĐỒ HỌA

Mục đích của việc khởi động hệ thống đồ họa là xác định thiết bị đồ họa (màn hình) và mốt đồ họa sẽ sử dụng trong chương trình. Để làm điều này ta dùng hàm

```
void initgraph(int *graphdriver, int *graphmode, char *driverpath);
```

trong đó: driverpath là đường dẫn của thư mục chứa các tệp tin điều khiển đồ họa, graphdriver, graphmode cho biết màn hình và mốt đồ họa sẽ sử dụng trong chương trình. Bảng 2 cho các giá trị khả dĩ của graphdriver và graphmode.

Ví dụ 1: Giả sử máy tính của ta có màn hình EGA, các tệp tin đồ họa chứa trong thư mục C: \TC, khi đó ta có thể khởi động hệ thống đồ họa như sau:

```
#include "graphics.h"
void main()
  {
    int mh=EGA, mode= EGALO;
    initgraph(&mh, &mode, "C:\TC");
    . . .
  }
```

Bảng 2. Các giá trị khả dĩ của graphdriver, graphmode

graphdriver	graphmode	Độ phân giải
Detect (0)		
CGA (1)	CGAC0 (0)	320 x 200
	CGAC1 (1)	320 x 200
	CGAC2 (2)	320 x 200
	CGAC3 (3)	320 x 200
	CGAHi (4)	640 x 200
MCGA (2)	MCGA0 (0)	320 x 200
	MCGA1 (1)	320 x 200
	MCGA2 (2)	320 x 200
	MCGA3 (3)	320 x 200
	MCGAMed (4)	640 x 200
	MCGAHi (5)	640 x 480
EGA (3)	EGALO (0)	640 x 200
	EGAHi (1)	640 x 350

EGA64 (4)	EGA64LO (0)	640 x 200
	EGA64Hi (1)	640 x 350
EGAMONO (5)	EGAMONOHi (0)	640 x 350
VGA (9)	VGALO (0)	640 x 200
	VGAMED (1)	640 x 350
	VGAHI (2)	640 x 480
HERCMONO (7)	HERCMONOHI	720 x 348
ATT400 (8)	ATT400C0 (0)	320 x 200
	ATT400C1 (1)	320 x 200
	ATT400C2 (2)	320 x 200
	ATT400C3 (3)	320 x 200
	ATT400MED (4)	640 x 400
	ATT400HI (5)	640 x 400
PC3270 (10)	PC3270HI (0)	720 x 350
IBM8514 (6)	IBM8514LO (0)	640 x 480, 256 mầu
	IBM8514HI (1)	1024 x 768, 256 mầu

Chú ý 1: Bảng 2 cho các hằng và giá trị của chúng mà các biến graphdriver, graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0... Khi lập trình ta có thể dùng tên hằng hoặc giá trị tương ứng của chúng. Chẳng hạn các phép gán trong ví dụ 1 có thể viết theo một cách khác tương đương như sau:

```
mh=3;
mode=0;
```

Chú ý 2: Bảng 2 cho thấy độ phân giải phụ thuộc cả vào màn hình và mode. Ví dụ trong màn hình EGA nếu dùng mode EGALO thì độ phân giải là 640 x 200, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 199. Nếu cũng màn hình EGA mà dùng mode EGAHI thì độ phân giải là 640x 350, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 349.

Chú ý 3: Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver hằng DETECT hay giá trị 0. Khi đó kết qủa của hàm initgraph sẽ là:

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

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

Như vậy việc dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ họa của màn hình hiện có theo mode có độ phân giải cao nhất, mà còn giúp ta xác định chính xác kiểu màn hình đang sử dụng.

Ví dụ 2: Chương trình dưới đây xác định kiểu màn hình đang sử dụng.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
void main()
  {
    int mh=0, mode= 0;
    initgraph(&mh, &mode, "");
    closegraph();
    printf("\n Giá trị số của màn hình là: %d", mh);
    getch();
  }
```

Nếu chương trình cho kết quả:

Giá trị số của màn hình là: 3

thì ta có thể khẳng định loại màn hình đang dùng là EGA.

Chú ý 4: Nếu chuỗi dùng để xác định driverpath là một chuỗi rỗng (như trong ví dụ 2) thì chương trình dịch sẽ tìm các tệp điều khiển đồ họa trên thư mục chủ (thư mục hiện hành).

§3. LỖI ĐỒ HỌA

Khi khởi động hệ thống đồ họa nếu máy không tìm thấy các chương trình điều khiển đồ họa thì sẽ phát sinh lỗi đồ họa và việc khởi động coi như không thành. Lỗi đồ họa còn phát sinh khi dùng các hàm. Trong mọi trường hợp, hàm graphresult cho biết có lỗi hay không lỗi và đó là lỗi gì. Bảng 3 cho các mã lỗi mà hàm này phát hiện được. Ta có thể dùng hàm grapherrormsg với mã lỗi do hàm graphresult trả về để biết được đó là lỗi gì, ví dụ:

```
int maloi;
maloi = graphresult();
printf("\nLỗi đồ họa là: %s", grapherrormsg(maloi));
```

Bảng 3. Các mã lỗi của Graphresult

Hằng	Trị	Lỗi phát hiện
grOk	0	Không có lỗi

grNoInitGraph	-1	Chưa khởi động hệ đồ họa
grNotDetected	-2	Không có phần cứng đồ họa
grFileNotFound	-3	Không tìm thấy trình điều khiển đồ họa
grInvalidDriver	-4	Trình điều khiển không hợp lệ
grNoLoadMem	-5	Không đủ RAM cho đồ họa
grNoScanMem	-6	Vượt vùng RAM trong Scan fill
grNoFloodMem	-7	Vượt vùng RAM trong flood fill
grFontNoFound	-8	Không tìm thấy tập tin Font
grNoFontMem	-9	Không đủ RAM để nạp Font
grInvalidMode	-10	Kiểu đồ họa không hợp lệ cho trình điều khiển
grError	-11	Lỗi đồ họa tổng quát
grIOerror	-12	Lỗi đồ họa vào ra
grInvalidFont	-13	Tập tin Font không hợp lệ
grInvalidFontNum	-14	Số hiệu Font không hợp lệ

§4. MẦU VÀ MẪU

Dưới đây là các hàm để chọn mầu và mẫu.

1. Để chọn mầu nền ta sử dụng hàm

void setbkcolor(int color);

2. Để chọn mầu đường vẽ ta dùng hàm

void setcolor(int color);

3. Để chọn mẫu (kiểu) tô và mầu tô ta dùng hàm

void setfillstyle(int pattern, int color);

Trong cả 3 trường hợp color xác định mã của mầu. Các giá trị khả dĩ của color cho trong bảng 4, pattern xác định mã của mẫu tô (xem bảng 5).

Mẫu tô và mầu tô sẽ được sử dụng trong các hàm pieslice, fillpoly, bar, bar3d và floodfill (xem §5 dưới đây).

4. Chọn giải mầu: Để thay đổi giải mầu đã được định nghĩa trong bảng 4 ta dùng hàm

void setpalette(int colornum, int color);

Ví dụ câu lệnh

setpalette(0, Lightcyan);

biến mầu đầu tiên trong bảng mầu thành xanh lơ nhạt (Lightcyan). Các mầu khác không bị ảnh hưởng.

Bảng 4. Các giá trị khả dĩ của color

Tên hằng	Giá trị số	Mầu hiển thị
BLACK	0	đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIHGTGRAY	7	Xám nhạt
DARKGRAY	8	Xám sẫm
LIGHTBLUE	9	Xanh da trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	15	Trắng

5. Để nhận giải mầu hiện hành ta dùng hàm

```
void getpalette (struct palettetype *palette);
```

ở đây palettetype là kiểu đã định nghĩa trước như sau:

```
#define MAXCOLORS 15
struct palettetype
   {
     unsigned char size;
     unsigned char colors[MAXCOLORS+1];
   };
```

ở đây: size là số lượng mầu trong palette, colors là mảng chứa mầu với chỉ số mảng chạy từ 0 đến size - 1.

Bảng 5. Các giá trị khả dĩ của pattern

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

6. Hàm getcolor trả về mầu đã xác định trước đó bằng hàm setcolor.

7. Hàm getbkcolor trả về mầu đã xác định trước đó bằng hàm setbkcolor.

8. Hàm getmaxcolor trả về số lượng mầu cực đại thuộc giải mầu hiện đang có hiệu lực. Trên 256 K EGA, hàm getmaxcolor luôn cho giá trị 15.

§5. VẼ VÀ TÔ MẦU

Có thể chia các đường và hình thành bốn nhóm chính:

- Đường tròn và hình tròn
- Đường gấp khúc và hình đa giác
- Đường thẳng
- Hình chữ nhật

5.1. Đường tròn và hình tròn

Nhóm này gồm cung tròn, đường tròn, cung ellipse và hình quạt.

1. Cung tròn: Để vẽ một cung tròn ta dùng hàm

void arc(int x, int y, int gd, int gc, int r);

ở đây:

(x, y) là tọa độ của tâm cung tròn

r là bán kính

gd là góc đầu

gc là góc cuối

Chú ý: Trong tất cả các hàm dưới đây, góc tính theo độ và có giá trị từ 0 đến 360.

2. Đường tròn: Để vẽ một đường tròn ta dùng hàm

void circle(int x, int y, int r);

ở đây:

(x, y) là tọa độ của tâm

r là bán kính đường tròn

3. Cung ellipse: Để vẽ một cung Ellipse ta dùng hàm

void ellipse(int x,int y,int gd,int gc,int xr,int yr);

ở đây:

(x, y) là tọa độ của tâm cung Ellipse

gd là góc đầu

gc là góc cuối

xr là bán trục ngang

yr là bán trục đứng

4. Hình quạt: Để vẽ và tô màu một hình quạt ta dùng hàm

void pieslice(int x,int y,int gd,int gc,int r);

ở đây:

(x,y) là tọa độ tâm hình quạt

gd là góc đầu

gc là góc cuối

r là bán kính

Ví dụ 1: Chương trình dưới đây sẽ vẽ: Một cung tròn ở góc phần tư thứ nhất, một cung ellipse ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
#include <graphics.h>
#include <conio.h>
main()
  {
   int mh, mode;
```

```
        /* Khởi động đồ họa, màn hình EGA, mode EGALO */
mh=EGA;
mode=EGALO;
initgraph(&mh, &mode,"C:\\TC\\BGI");
        /* Mầu nền Green, mầu đường vẽ
        White, mầu tô Red, kiểu tô SlashFill */
  setbkcolor (GREEN);
  setcolor (WHITE);
  setfillstyle (SLASH_FILL, RED);
            /* Vẽ: một cung tròn ở góc phần tư thứ nhất,
            một cung Ellipse ở góc phần tư thứ ba,
            một đường tròn, một quạt tròn */
  arc(160, 50, 0, 90, 45);
  ellipse(480, 50, 180, 270, 150, 45);
  circle(160, 150, 45);
  pieslice(480, 150, 90, 360, 45);
  getch();
            /* Kết thúc chế độ đồ họa */
  closegraph();
}
```

5.2. Đường gấp khúc và đa giác

1. Muốn vẽ một đường gấp khúc đi qua n điểm: (x1,y1), ..., (xn,yn) thì trước hết ta phải đưa các tọa độ vào một mảng a nào đó kiểu int. Nói một cách chính xác hơn, cần gán x1 cho a[0], y1 cho a[1], x2 cho a[2], y2 cho a[3],... Sau đó ta viết lời gọi hàm:

    drawpoly(n, a);

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

2. Giả sử a là mảng đã nói trong điểm 1, khi đó lời gọi hàm

    fillpoly(n, a);

sẽ có tác dụng vẽ và tô mầu một đa giác có đỉnh là các điểm

(x1, y1), ... ,(xn, yn).

Ví dụ 2: Chương trình dưới đây sẽ vẽ một đường gấp khúc và hai hình tam giác.

```
#include <graphics.h>
#include <conio.h>
        /* Xây dựng các mảng chứa tọa độ các đỉnh */
int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
        /* Mầu nền CYAN, mầu đường vẽ
    YELLOW, mầu tô MAGENTA, mẫu tô SolidFill */
    setbkcolor (CYAN);
    Setcolor (YELLOW);
    setfillstyle (SOLID_FILL, MAGENTA);
        /* Đường gấp khúc */
    drawpoly (3, poly1);
        /* Hình đa giác */
    fillpoly (3, poly2);
        /* Hình đa giác */
    fillpoly(4, poly3);
    getch();
    closegraph();
  }
```

5.3. Đường thẳng

1. Hàm

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

vẽ đường thẳng nối hai điểm (x1, y1) và (x2, y2) nhưng không làm thay đổi vị trí con trỏ.

2. Hàm

void lineto(int x,int y);

vẽ đường thẳng từ điểm hiện tại tới điểm (x, y) và chuyển con trỏ đến điểm (x, y).

3. Hàm

void linerel(int dx,int dy);

vẽ một đường thẳng từ vị trí hiện tại (x, y) của con trỏ đến điểm (x + dx, y + dy). Con trỏ được di chuyển đến vị trí mới.

4. Hàm

void moveto(int x,int y);

sẽ di chuyển con trỏ tới vị trí (x, y).

Ví dụ 3: Chương trình dưới đây tạo lên một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau: (20,20), (620, 20), (620, 180), (20, 180) và (320, 100).

```
#include <graphics.h>
#include <conio.h>
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
    setcolor(YELLOW);
    moveto(320,100);
    line(20,20,620,20);
    linerel(-300,80);
    lineto(620,180);
    lineto(620,20);
    getch();
    closegraph();
  }
```

5.4. Hình chữ nhật

1. Hàm

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

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

2. Hàm

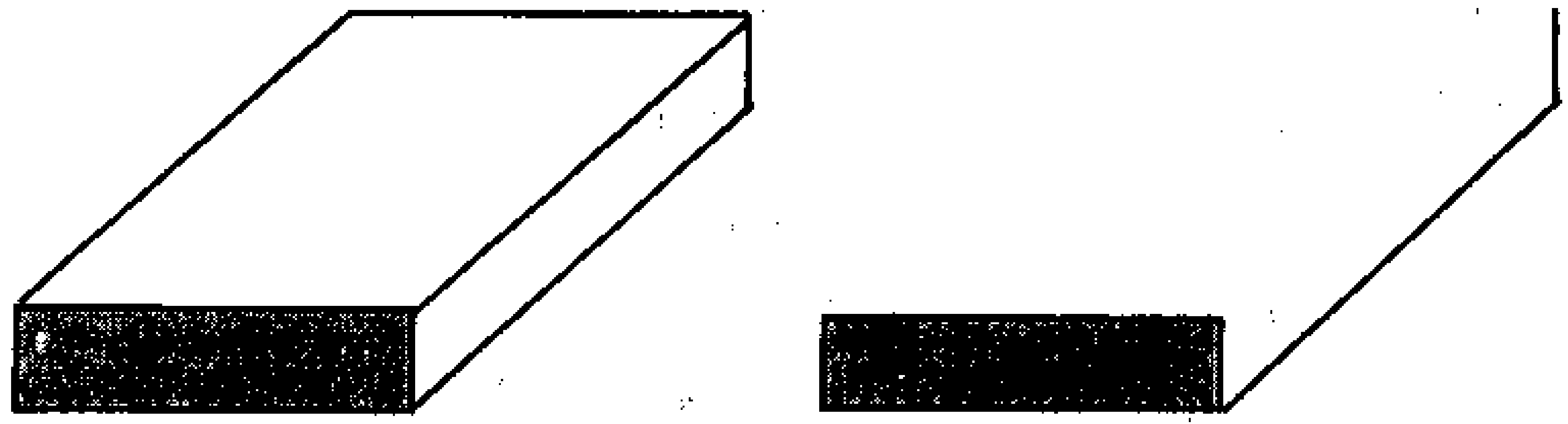
void bar(int x1,int y1,int x2,int y2);

sẽ vẽ và tô mầu một hình chữ nhật. Các giá trị x1, y1, x2 và y2 có ý nghĩa như đã nói trong điểm 1.

3. Hàm

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

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các tọa độ x1,y1,x2,y2 (như đã nói trong điểm 2). Hình chữ nhật này được tô mầu. Tham số depth ấn định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số top cho biết khối hộp có nắp hay không: Khi top <>0 là có nắp, top = 0 là không nắp (xem hình vẽ).



top<>0 top=0

Ví dụ 4: Chương trình dưới đây sẽ vẽ một đường chữ nhật, một hình chữ nhật và một khối hộp chữ nhật có nắp.

```
#include <graphics.h>
#include <conio.h>
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL,YELLOW);
    rectangle(5,5,300,160);
    bar(5,175,300,340);
    bar3d(320,100,500,340,100,1);
    getch();
    closegraph();
  }
```

§6. CHỌN KIỂU ĐƯỜNG

Mục này giới thiệu 3 hàm

1. Hàm

void setlinestyle(int linestyle,int pattern,int thickness);

tác động đến nét vẽ của các thủ tục line, lineto, rectange, drawpoly, circle,... Hàm này cho phép ta ấn định 3 yếu tố của đường thẳng là dạng, bề dầy và mẫu tự tạo.

+ Dạng đường được xác định bởi tham số linestyle. Sau đây là các giá trị khả dĩ của linestyle và dạng đường thẳng tương ứng.

SOLID_LINE = 0	Nét liền
DOTTED_LINE = 1	Nét chấm
CENTER_LINE = 2	Nét chấm gạch
DASHED_LINE = 3	Nét gạch
USERBIT_LINE = 4	Mẫu tự tạo

+ Bề dầy được xác định bởi tham số thickness. Tham số này có thể là:

NORM_WIDTH = 1	Bề dầy bình thường
THICK_WIDTH = 3	Bề dầy gấp ba

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

```
int pattern= 0x1010;
setlinestyle(USERBIT_LINE, pattern, NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010 hay trong hệ 2 là

0001 0000 0001 0000

Chỗ nào có bit 1 điểm ảnh sẽ sáng, bit 0 làm tắt điểm ảnh.

2. Để nhận các giá trị hiện hành của 3 yếu tố trên ta dùng hàm:

void getlinesettings(struct linesettingstype *lineinfo);

với kiểu linesettingstype đã được định nghĩa trước như sau:

```
struct linesettingstype
    {
        int linestyle;
        unsigned int upattern;
        int thickness;
    };
```

Ví dụ 1: Chương trình dưới đây minh họa cách dùng các hàm setlinestyle và getlinesettings để vẽ đường thẳng.

```
/* Kiểu đường */
        #include <graphics.h>
        #include <conio.h>
        main()
            {
```

```
    struct linesettingstype kieu_cu;
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk)
        exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    line(0,0,100,0);
            /* Lưu lại kiểu hiện tại */
    getlinesettings(kieu_cu);
            /* Thiết lập kiểu mới */
    setlinestyle(DOTTED_LINE,0,THICK_WIDTH);
    line(0,0,100,10);
            /* Phục hồi kiểu cũ */
    setlinestyle(kieu_cu.linestyle,kieu_cu.upattern,
                 kieu_cu.thickness);
    Line(0,20,100,20);
    getch();
    closegraph();
}
```

3. Hàm

void setwritemode(int writemode);

sẽ thiết lập kiểu thể hiện đường thẳng cho các hàm line, drawpoly, linerel, lineto, rectangle. Kiểu thể hiện được xác định bởi tham số writemode:

- Nếu writemode bằng COPY_PUT = 0, thì đường thẳng được viết đè lên dòng đang có trên màn hình.

- Nếu writemode bằng XOR_PUT = 1, thì mầu của đường thẳng định vẽ sẽ kết hợp với mầu của từng chấm điểm của đường hiện có trên màn hình theo phép toán XOR (Chương 3, §3) để tạo lên một đường thẳng mới.

Một ứng dụng của XOR_PUT là: Khi thiết lập kiểu writemode bằng XOR_PUT rồi vẽ lại đường thẳng cùng mầu thì sẽ xóa đường thẳng cũ và trả về màn hình nguyên thủy.

Chương trình dưới đây minh họa cách dùng hàm setwritemode. Khi thực hiện ta sẽ thấy hình chữ nhật thu nhỏ dần vào tâm màn hình.

Ví dụ 2:

```
/* Thu hình; */
        #include <graphics.h>
        #include <conio.h>
        main()
          {
            struct linesettingstype kieu_cu;
            int mh=0, mode=0, x1, y1, x2, y2;
            initgraph(&mh, &mode, "");
            if (graphresult!= grOk)
                exit(1);
            setbkcolor(GREEN);
            setcolor(RED);
            setfillstyle(CLOSE_DOT_FILL, YELLOW);
            x1=0;
            y1=0;
            x2=getmaxx();
            y2=getmaxy();
            setwritemode(XOR_PUT);
                /* Vẽ hình chữ nhật */
            tt: rectangle(x1,y1,x2,y2);
            if ( (x1+1)<(x2-1) && (y1+1)<(y2-1) )
              {     /* xóa hình chữ nhật */
                rectangle(x1,y1,x2,y2);
                x1=x1+1;
                y1=y1+1;
                    /* co hình chữ nhật */
                x2=x2-1;
                y2=y2-1;
                goto tt;
              }
              /* Trở về overwrite mode */
            setwritemode(COPY_PUT);
            closegraph();
          }
```

§7. CỬA SỔ (VIEWPORT)

1. Viewport

Là một vùng chữ nhật trên màn hình đồ họa tựa như window trong textmode. Để thiết lập viewport ta dùng hàm

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

trong đó (x1,y1) là tọa độ góc trên bên trái và (x2,y2) là tọa độ góc dưới bên phải. Bốn giá trị này phải thỏa mãn:

0 <= x1 <= x2

0 <= y1 <= y2

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

clip = 1 không cho phép vẽ ra ngoài viewport

clip = 0 Cho phép vẽ ra ngoài viewport.

Ví dụ câu lệnh

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

sẽ thiết lập viewport. Sau khi lập viewport ta có hệ tọa độ mới mà góc trên bên trái của viewport sẽ có tọa độ (0,0).

2. Để nhận viewport hiện hành ta dùng hàm

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

ở đây kiểu viewporttype đã được định nghĩa như sau:

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

3. Để xóa viewport ta dùng hàm

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

4. Để xóa màn hình và đưa con chạy về tọa độ (0,0) của màn hình ta dùng hàm

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

Chú ý: Câu lệnh này sẽ xóa mọi thứ trên màn hình.

5. Tọa độ âm dương:

Nhờ sử dụng Viewport có thể viết các chương trình đồ họa theo tọa độ âm dương. Muốn vậy ta thiết lập viewport sao cho tâm (điểm giữa) màn hình là góc trên bên trái của viewport và cho clip = 0 để có thể vẽ ra ngoài giới hạn của viewport. Sau đây là đoạn chương trình thực hiện công việc trên

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

Như thế màn hình sẽ được chia làm 4 phần với tọa độ âm dương như sau:

Phần tư trái trên x âm, y âm

Phần tư trái dưới x âm, y dương

Phần tư phải trên x dương, y âm

Phần tư phải dưới x dương, y dương

Chương trình dưới đây vẽ đồ thị hàm sin(x) trong hệ trục tọa độ âm dương. Hoành độ x lấy các giá trị từ -4*PI đến 4*PI. Trong chương trình có dùng hai hàm mớilà: outtextxy và putpixel (xem các mục sau).

Ví dụ 1:

```
/* Đồ thị hàm sin; */
    #include <graphics.h>
    #include <conio.h>
    #include <math.h>
    #define SCALEX 20
    #define SCALEY 60
    main()
      {
        int mh=0, mode=0, x, y, i;
        initgraph(&mh, &mode, "");
        if (graphresult!= grOk)
          exit(1);
        setviewport(getmaxx()/,getmaxy()/2, getmaxx(),g etmaxy(),0);
              /* Kẻ hệ trục tọa độ */
        setcolor(BLUE);
        line(-(getmaxx()/2),0,getmaxx()/2,0);
        line(0,-(getmaxy()/2),0,getmaxy()/2);
        settextjustify(1,1);
        setcolor(RED);
        outtextxy(0,0,"(0,0)");
        for (i=-400;i<=400;++i)
```

```
    {
      x=round(2*M_PI*i*SCALEX/200);
      y=round(sin(2*M_PI*i/200)*SCALEY);
      putpixel(x,y,YELLOW);
    }
  getch();
  closegraph();
}
```

Ví dụ 1 tạo lên một đồ thị từ các chấm điểm. Bây giờ ta sửa ví dụ 1 đôi chút: giữ nguyên từ đầu đến outtextxy, thay phần cuối bởi đoạn chương trình dưới đây. Ta sẽ được đồ thị liên tục gồm các đoạn thẳng ghép lại với nhau.

Ví dụ 2:

```
/* Phần đầu giống ví dụ 1 */
   setcolor(YELLOW);
   for (i=-400;i<=400;++i)
     {
       x=round(2*M_PI*i*SCALEX/200);
       y=round(sin(2*M_PI*i/200)*SCALEY);
       if(i= -400)
           moveto(x,y);
       else
           lineto(x,y);
     }
   getch();
   closegraph();
}
```

§8. TÔ ĐIỂM, TÔ MIỀN

1. Hàm

void putpixel(int x, int y, int color);

sẽ tô điểm (x,y) theo mầu xác định bởi color.

2. Hàm

unsigned getpixel(int x, int y);

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y).

Chú ý: nếu điểm này chưa được tô mầu bởi các hàm vẽ hoặc putpixel (mà chỉ mới được tạo mầu nền bởi setbkcolor) thì hàm cho giá trị bằng 0. Vì vậy có thể dùng hàm này theo mẫu dưới đây để xác định các nét vẽ trên màn hình đồ hoạ và vẽ ra giấy.

```
if(getpixel(x,y)!=0)
  {
     /* Điểm (x,y) được vẽ , đặt một chấm điểm ra giấy */
  }
```

3. Tô miền. Để tô mầu cho một miền nào đó trên màn hình ta dùng hàm:

void floodfill(int x, int y, int border);

ở đây:

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

tham số border chứa mã của một mầu.

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

a) Khi trên màn hình có một đường (cong hoặc gấp khúc) khép kín mà mã mầu của nó bằng giá trị của border thì:

+ Miền giới hạn bởi đường kín sẽ được tô mầu nếu điểm gieo (x,y) nằm bên trong miền này.

+ Nếu (x,y) nằm bên ngoài thì phần màn hình bên ngoài miền đóng nói trên được tô màu.

b) Khi trên màn hình không có một đường nào như vậy, thì cả màn hình được tô màu.

Ví dụ 1: Chương trình dưới đây sẽ vẽ một đường tròn đỏ trên màn hình xanh. Tọa độ (x,y) của điểm gieo được nạp vào từ bàn phím. Tùy thuộc vào giá trị cụ thể của x, y, chương trình sẽ tô mầu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include <graphics.h>
#include <stdio.h>
main()
  {
     int mh=0, mode=0, x, y;
     initgraph(&mh, &mode, "");
     if (graphresult!= grOk)
          exit(1);
```

```
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(11,YELLOW);
    circle(320,100,50);
    moveto(1,150);
    outtext(" Toa do diem gieo x,y ");
    scanf("%d%d",&x,&y);
    flooddfill(x,y,RED);
    getch();
    closegraph();
  }
```

Ví dụ 2: Minh họa cách dùng hàm Putpixel và hàm getpixel để vẽ các điểm ảnh và sau đó xóa các điểm ảnh. Muốn kết thúc chương trình bấm ESC

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
int seed = 1962;      /* Nhận cho bộ tạo số ngẫu nhiên */
int numpts = 2000;  /* Vẽ 2000 chấm điểm */
int ESC = 27;
void putpixelplay(void);
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    if (graphresult()!= grOk)
      {
        exit(1);
      }
    putpixelplay();
    closegraph();
}
void putpixelplay(void)
    {
     int i,x,y,color,xmax,ymax,maxcolor,ch;
     struct viewporttype v;
```

```
        getviewsettings(&v);
        xmax=(v.right - v.left -1);
        ymax=(v.bottom - v.top -1);
        maxcolor=getmaxcolor();
        while (!kbhit())
          {
            /* Vẽ các chấm điểm một cách ngẫu nhiên */
            srand(seed);
            i=0;
            while(i<=numpts)
              {
                ++i;
                x=random(xmax)+1;
                y=random(ymax)+1;
                color=random(maxcolor);
                putpixel(x,y,color);
              }
            /* Xóa các điểm ảnh */
            srand(seed);
            i=0;
            while(i<=numpts)
              {
                ++i;
                x= random(xmax) + 1;
                y= random(ymax) + 1;
                color=random(maxcolor);
                putpixel(x,y,0);
              }
            if(kbhit())
              {
                ch=getch();
                if(ch==ESC)
                   break;
              }
          }
} /* Kết thúc hàm putpixelplay */
```

§9. XỬ LÝ VĂN BẢN TRÊN MÀN HÌNH ĐỒ HOẠ

9.1. Hiển thị văn bản trên màn hình đồ hoạ

Hàm

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

sẽ cho hiện chuỗi ký tự (do s trỏ tới) tại vị trí hiện tại của con trỏ.

Hàm

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

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

Ví dụ 1: Hai cách sau đây sẽ cho cùng kết quả

```
outtextxy (100,100,"Chao ban");
```

và

```
moveto (100,100);
outtext ("Chao ban");
```

Chú ý: Trong mốt đồ họa vẫn cho phép hàm vào dữ liệu scanf và các hàm bắt phím getch, kbhit.

9.2. Fonts

Như đã nêu các Fonts nằm trong các tệp tin .CHR trên đĩa. Các Font này cho các kích thước và kiểu chữ khác nhau sẽ hiện thị trên màn hình đồ hoạ bằng các hàm outtext, outtextxy. Để chọn và nạp Font chúng ta dùng hàm:

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

Chú ý: hàm chỉ có tác dụng nếu tồn tại các tệp .CHR

+ Tham số direction có thể nhận một trong hai giá trị:

HORIZ_DIR = 0

VERT_DIR = 1

Nếu direction = HORIZ_DIR, văn bản sẽ hiển thị theo chiều nằm ngang từ trái sang phải. Nếu direction = VERT_DIR, văn bản sẽ hiển thị theo chiều đứng từ dưới lên trên.

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

- Nếu charsize = 1, Font được thể hiện trong hình chữ nhật 8*8 pixel.

- Nếu charsize = 2, Font được thể hiện trong hình chữ nhật 16*16 pixel.

...

- Nếu charsize = 10, Font được thể hiện trong hình chữ nhật 80*80 pixel.

+ Tham số font dùng để chọn kiểu chữ và nhận một trong các giá trị sau:

DEFAULT_FONT = 0
TRIPLEX_FONT = 1
SMALL_FONT = 2
SANS_SERIF_FONT = 3
GOTHIC_FONT = 4

Các giá trị do settextstyle thiết lập sẽ dữ nguyên cho đến khi gọi một settextstyle mới.

Ví dụ 2:

```
settextstyle (3,VERT_DIR,2);
outtextxy (50,50," HELLO ");
```

9.3. Vị trí hiển thị

Hàm settextjustify quy định vị trí văn bản hiển thị (trong các hàm outtext và outtextxy) so với điểm mốc (x,y).

Hàm này có dạng

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

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

LEFT_TEXT = 0 (Điểm mốc bên trái văn bản)
CENTER_TEXT = 1 (Điểm mốc ở giữa văn bản theo chiều ngang)
RIGHT_TEXT = 2 (Điểm mốc bên phải văn bản)

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

BOTTOM_TEXT = 0 (Điểm mốc ở đáy văn bản)
CENTER_TEXT = 1 (Điểm mốc ở giữa văn bản theo chiều dọc)
TOP_TEXT = 2 (Điểm mốc ở đỉnh văn bản)

Ví dụ 3:

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

Kết quả là điểm (100,100) sẽ nằm giữa chữ B.

9.4. Bề rộng và bề cao của văn bản

Hàm

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

trả về chiều cao (theo pixel) của chuỗi do s trỏ tới. Với 8*8 bit map Font và hệ số khuyếch đại chữ là 1 thì

textheight ("H") = 8

Ví dụ 4: Đoạn chương trình dưới đây sẽ cho hiện 5 dòng chữ.

```
#include <graphics.h>
#include <conio.h>
main()
  {
    int mh=0,mode=0,y,size;
    initgraph(&mh,&mode,"");
    y=10;
    settextjustify(0,0);
    for(size=1; size<=5; ++size)
      {
        settextstyle(0,0,size);
        outtextxy(0,y,"GRAPHICS");
        y += textheight("GRAPHICS") + 10;
      }
    getch();
    closegraph();
  }
```

Hàm

void textwidth(char *s);

sẽ dựa vào chiều dài của chuỗi, kích thước Font chữ, hệ số khuyếch đại chữ để trả về bề rộng (theo pixel) của chuỗi do s trỏ tới.

Ví dụ 5: Trong chương trình dưới đây sẽ lập các hàm vào ra trên màn hình đồ hoạ.

```
#include <graphics.h>
#include <conio.h>
#define Enter 13
#define Lmargin 10
void text_write(int *x,int *y,char *s);
void text_writeln(int *x,int *y,char *s);
void text_read(int *x,int *y,char *s);
void text_write(int *x,int *y,char *s)
  {
    outtextxy(*x,*y,s);  *x += textwidth(s);
  }
```

```
void text_writeln(int *x,int *y,char *s)
   {
     outtextxy(*x,*y,s);
     *x=Lmargin;
     *y += textheight(s)+5;
   }
void text_read(int *x,int *y,char *s)
   {
     int i=0;
     char ch[2];
     ch[1]=0;
     while(1)
      {
        ch[0]=getch();
        if(ch[0]==Enter) break;
        text_write(x,y,ch);
        s[i]=ch[0]; ++i;
      }
     s[i]=0;
   }
main()
   {
     int mh=0,mode=0,x,y,xmax,ymax;
     char name[25];
     initgraph(&mh,&mode,"");
     settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
     x=Lmargin;
     y=100;
     text_write (&x,&y,"cho ten của bạn: ");
     text_read (&x,&y,name);
     text_writeln (&x,&y,"" );
     text_write(&x,&y,"chào bạn ");
     text_write(&x,&y,name);
     getch();
     closegraph();
   }
```

§10. CẮT HÌNH, DÁN HÌNH VÀ TẠO ẢNH CHUYỂN ĐỘNG

1. Hàm

unsigned imagesize(int x1,int y1,int x2,int y2)

trả về số byte cần thiết để lưu trữ ảnh trong phạm vi hình chữ nhật (x1,y1,x2,y2).

2. Hàm

#include <alloc.h>

void *malloc(unsigned n);

trả về con trỏ trỏ tới một vùng nhớ n byte mới được cấp phát.

3. Hàm

void getimage(int x1,int y1,int x2,int y2,void *bitmap);

sẽ chép các điểm ảnh của hình chữ nhật (x1,y1,x2,y2) và các thông tin về bề rộng, cao của hình chữ nhật vào vùng nhớ do bitmap trỏ tới. Vùng nhớ và biến bitmap cho bởi hàm malloc. Độ lớn của vùng nhớ được xác định bằng hàm imagesize.

4. Hàm

void putimage(int x,int y,void *bitmap,int copymode);

dùng để sao ảnh lưu trong vùng nhớ bitmap ra màn hình tại vị trí (x,y). Tham số copymode xác định kiểu sao chép ảnh, nó có thể nhận các giá trị sau:

COPY_PUT = 0	Sao chép nguyên bản
XOR_PUT = 1	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép XOR
OR_PUT = 2	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép OR
AND_PUT = 3	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép AND
NOT_PUT = 4	ảnh xuất hiện trên màn hình theo dạng đảo ngược (phép NOT) với ảnh trong bitmap.

Nhận xét: Nếu dùng mode XOR_PUT để sao hình, rồi lặp lại đúng câu lệnh đó thì hình sẽ bị xoá và màn hình trở lại như cũ. Kỹ thuật này dùng để tạo lên các hình ảnh chuyển động.

Ví dụ 1: Chương trình dưới đây minh hoạ cách dùng imagesize, malloc, getimage và putimage.

```
#include <alloc.h>
#include <graphics.h>
main()
   {
      int mh=0,mode=0;
      char *p;
      unsigend size;
      initgraph (&mh,&mode,"");
      bar(0,0,getmaxx(),getmaxy());
      size = imagesize(10,20,30,40);
      p=(char*)malloc(size); /* p trỏ tới vùng nhớ size
                                          byte mới được cấp phát */
      getimage (10,20,30,40,p); getch();
      cleardevice();
      putimage (100,100,p,COPY_PUT);
      getch();
      closegraph();
   }
```

5. Tạo ảnh chuyển động

Nguyên tắc tạo ảnh chuyển động giống như phim hoạt hình:

- Vẽ một hình (trong chuỗi hình mô tả chuyển động)
- Delay
- Xoá hình đó
- Vẽ hình kế theo
- Delay

A) Vẽ hình

Cách 1: Vẽ lại một ảnh nhưng tại các vị trí khác nhau.

Cách 2: Lưu ảnh vào một vùng nhớ rồi đưa ảnh ra màn hình tại các vị trí khác nhau.

B) Xóa ảnh

Cách 1: Dùng hàm cleardevice

Cách 2: Dùng hàm putimage (mode XOR_PUT) để xếp chồng lên ảnh cần xoá.

Cách 3: Lưu trạng thái màn hình vào một chỗ nào đó. Vẽ một hình ảnh. Đưa trạng thái cũ màn hình ra xếp đè lên ảnh vừa vẽ.

Kỹ thuật tạo ảnh chuyển động được minh hoạ trong các chương trình của §11.

Chương trình 1: Đầu tiên vẽ bầu trời đầu sao. Sau đó từng chùm pháo hoa được bắn lên bầu trời. Khi bấm phím Enter thì việc bắn pháo hoa kết thúc, ta nhận lại bầu trời đầy sao. Bấm tiếp Enter thì kết thúc chương trình.

```
/* Bắn pháo hoa trên bầu trời đầy sao */
        #include <graphics.h>
        #include <conio.h>
        #include <stdlib.h>
        #include <alloc.h>
        main()
          {
            int x[101],y[101];
            int mh=0,mode=0,i,n;
            char *p[101];
            initgraph(&mh,&mode,"");
            if(graphresult()!=0)
                exit(1);
            setcolor(RED);
                /* Vẽ bầu trời đầy sao */
            for(i=1;i<=1000;++i)
              {
                putpixel(random(getmaxx()),
                random(getmaxy()),random(getmaxcolor()));
              }
                    /*Lưu hiện trạng 100 hình chữ nhật
                      trên màn hình để khôi phục*/
            for(i=1;i<=100;++i)
              {
                x[i]=random(getmaxx())-10;
                y[i]=random(getmaxy())-10;
                if(x[i]<0) x[i]=0;
                 if(y[i]<0) y[i]=0;
                n=imagesize(x[i],y[i],x[i]+10,y[i]+10);
```

```
        p[i]=(char*)malloc(n);
        getimage(x[i],y[i],x[i]+10,y[i]+10,p[i]);
      }
      /* Chu trình bắn pháo hoa */
    do
      {
              /* Đưa 100 quả pháo lên màn hình
                 tại các vị trí quy định */
        for(i=1;i<=100;++i)
          {
            setfillstyle(SOLID_FILL,i%15+1);
            pieslice(x[i]+5,y[i]+5,0,360,5);
          }
        delay(500);
              /*Xoá chùm pháo hoa vừa bắn
              bằng cách khôi phục màn hình*/
        for(i=100;i>=1;--i)
        putimage(x[i],y[i],p[i],COPY_PUT);
        delay(500);
      }
      while(!kbhit());
       getch();
       getch();
    closegraph();
  }
```

Chương trình 2: Vẽ đồng hồ có 3 kim giờ, phút và giây. Đồng hồ chạy đúng theo giờ hệ thống. Muốn kết thúc chương trình bấm ↵.

```
/*
 Đồng hồ
*/
        #include <graphics.h>
        #include <conio.h>
        #include <math.h>
        #include <dos.h>
```

```
/*
Hàm kẻ đoạn thẳng từ tâm đồng hồ theo độ, chiều dài,
độ dầy và mầu
*/
        void ke(int ddo, unsigned dai, unsigned day,unsigned mau);
        /* Kẻ kim giây khi biết số giây */
        void ke_giay(unsigned giay);
        /* Kẻ kim phút khi biết số phút */
        void ke_phut(unsigned phut);
        /* Kẻ kim giờ khi biết số giờ */
        void ke_gio(unsigned gio, unsigned phut);
        void chay_kim_giay(void);
        void chay_kim_phut(void);
        void chay_kim_gio(void);
        int x0,y0,rgio,rphut,rgiay,mgio,mphut,mgiay;
        unsigned phutgioht,gioht,phutht,giayht;
        void ke(int ddo, unsigned dai, unsigned day,unsigned mau)
            {
              unsigned x,y; float goc;
              while (ddo>=360)
                    ddo=ddo-360;
                    goc=(M_PI/180)*ddo;
                    x=x0+ (int)(dai*cos(goc)+0.5);
                    y=y0- (int)(dai*sin(goc)+0.5);
                    setcolor(mau);
                    setlinestyle(0,0,day);
                    line(x0,y0,x,y);
            }
/* Hàm ke kim giay*/
        void ke_giay(unsigned giay)
            {
              int ddo;
              ddo = (90 - 6*giay);
              ke(ddo,rgiay,1,mgiay);
            }
```

```
/* Hàm ke kim phut*/
        void ke_phut(unsigned phut)
          {
            int ddo;
            ddo= (90-6*phut);
            ke(ddo,rphut,3,mphut);
          }
/* Hàm ke kim gio*/
        void ke_gio(unsigned gio, unsigned phut)
          {
            int ddo;
            ddo = 360 + 90 - 30*(gio%12) - (phut+1)/2;
            ke(ddo,rgio,3,mgio);
          }
/* Hàm chỉnh giây hiện tại và làm chuyển động kim giây */
        void chay_kim_giay(void)
          {
            unsigned giay; struct time t;
            gettime(&t);
            giay=t.ti_sec;
            if (giay!=giayht)
              {
                ke_giay(giayht);
                giayht=giay;
                ke_giay(giayht);
              }
          }
/* Hàm chỉnh phút hiện tại và làm chuyển động kim phút */
        void chay_kim_phut(void)
          {
            unsigned phut;
            struct time t;
            gettime(&t);
```

```
            phut=t.ti_min;
            if(phut!=phutht)
              {
                  ke_phut(phutht);
                  phutht=phut;
                  ke_phut(phutht);
              }
        }
/* Hàm chỉnh giờ phút hiện tại và làm chuyển động kim giờ */
    void chay_kim_gio(void)
      {
         unsigned h,gio,phut,sophut,sophutht;
         struct time t;
         gettime(&t);
         gio=t.ti_hour;
         phut=t.ti_min;
         sophut = gio*60+phut;
         sophutht = gioht*60+phutgioht;
         if( sophut<sophutht)
               sophut=sophut+ 12*60;
               h=sophut-sophutht;
         if(h>=12)
         {
           ke_gio(gioht,phutgioht);
           phutgioht=phut;
           gioht=gio;
           ke_gio(gioht,phutgioht);
         }
      }
main()
  {
     struct time t;
     char*dso[]={"","12","1","2","3","4","5","6","7","8","9","10", 11"};
           int i, mh=0, mode=0,r,x,y;
```

```
float goc;
initgraph(&mh,&mode,"");
x0=(getmaxx()/2)-1;
y0=(getmaxy()/2)-1;
r=y0-2;
rgiay = r-10;
rphut=r-50;
rgio=r-90;
mgiay= BROWN;
mphut=RED;      /* mgio:=magenta;*/
mgio=YELLOW;

      /* Vẽ chu vi đồng hồ */
setcolor(BLUE);
setlinestyle(0,0,3);
circle(x0,y0,r);
setfillstyle(1,YELLOW);
floodfill(0,0,BLUE);
setfillstyle(1,WHITE);
floodfill(x0,y0,BLUE);
setlinestyle(0,0,1);
circle(x0,y0,10);
setfillstyle(1,GREEN);
floodfill(x0,y0,BLUE);
settextjustify(1,1);
setcolor(MAGENTA);
outtextxy(x0,y0+120,"IBM-JIMIKO");
      /* Ghi chữ số */
settextstyle(3,0,3);
settextjustify(1,1);
setcolor(BLUE);
for(i=1;i<=12;++i)
  {
```

```
        goc=(2*M_PI+M_PI/2) - (i-1)*(M_PI/6);
        x = x0+ (int)(rphut*cos(goc)+0.5);
        y = y0- (int)(rphut*sin(goc)+0.5);
        outtextxy(x,y,dso[i]);
      }
    /* Xác định thời điểm đầu */
   gettime(&t);
   gioht=t.ti_hour;
   phutht=t.ti_min;
   giayht=t.ti_sec;
   phutgioht=phutht;
   setwritemode(XOR_PUT);
        /* Ve kim gio,phut,giay */
   ke_gio(gioht,phutgioht);
   ke_phut(phutht);
   ke_giay(giayht);
        /* Làm chuyển động các kim */
   do
     {
       chay_kim_giay();
       chay_kim_phut();
       chay_kim_gio();
     }
   while(!kbhit());
   closegraph();
 }
```

Chương trình 3: Vẽ một con tầu vũ trụ bay trên bầu trời đầy sao theo quỹ đạo ellipse. Trong khi tầu chuyển động thì các ngôi sao thay nhau nhấp nháy

```
/* Tầu vũ trụ chuyển động trên bầu trời đầy sao nhấp nháy */
      #include <graphics.h>
      #include <stdio.h>
      #include <conio.h>
      #include <alloc.h>
      #include <stdlib.h>
```

```
#include <math.h>
          /* Khai báo các hàm trong chương trình */
void tau_cd(void);          /* tầu chuyển động */
void nhap_nhay_bt(void); /* sao nhấp nháy trên bầu trời */
void main(void);            /* hàm main */
            /* Khai báo các biến mảng ngoài */
int a,b,x,y,x0,y0;
int mh=0,mode=0,n,i;
float goc,xt,yt;
char *p;
int xx[1001],yy[1001];
 /* Hàm main */
void main(void)
  {
    initgraph(&mh,&mode,"");
    if(graphresult()!=0)
        exit(1);
          /* Vẽ tầu vũ trụ */
    setcolor(RED);
    ellipse(100,50,0,360,20,8);
    ellipse (100,46,190,357,20,6);
    line(107,44,110,38);
    circle(110,38,2);
    line(93,44,90,38);
    circle(90,38,2);
    setfillstyle(SOLID_FILL,BLUE);
    floodfill(101,54,RED);
    setfillstyle(SOLID_FILL,MAGENTA);
    floodfill(94,45,RED);
        /* Lưu ảnh của tầu vũ trụ vào bộ nhớ */
    n=imagesize(79,36,121,59);
    p=(char*)malloc(n);
    getimage(79,36,121,59,p);
```

```
/*
Vẽ bầu trời đầy sao và lưu vị trí của chúng
vào các mảng xx, yy để phục vụ hàm nhap_nhay_bt
*/
cleardevice();
for(i=1;i<=1000;++i)
  {
    xx[i]=random(getmaxx());
    yy[i]=random(getmaxy());
    putpixel(xx[i],yy[i],random(getmaxcolor()));
  }
    /* Xác định giá trị ban đầu cho các biến
    dùng để điều khiển chuyển động tầu */
goc= 2*M_PI + M_PI/2;
x0= (getmaxx() - 42)/2;
y0= (getmaxy() - 25)/2;
a=x0;
b=y0;
    /* chu trình tầu vũ trụ chuyển động và
          các ngôi sao nhấp nháy*/
do
  {
    tau_cd();
    nhap_nhay_bt();
  }
while(!kbhit());
getch();
closegraph();
}
}
void tau_cd(void)
{
  xt=a*cos(goc)+x0;
  yt=-b*sin(goc)+y0;
```

```
        x=(int)(xt+0.5);
        y=(int)(yt+0.5);
                /* Đặt tầu vũ trụ lên màn hình */
        putimage(x,y,p,XOR_PUT);
        delay(500);
                /* Xóa */
        putimage(x,y,p,XOR_PUT);
                /* Thay đổi góc để làm cho tầu chuyển động */
        goc -= M_PI/30;
        if (goc<M_PI/2)
            goc=2*M_PI+M_PI/2;
    }
void nhap_nhay_bt(void)
    {
        static i=1;   /* Lệnh này thực hiện một lần khi dịch */
        int j;
            /* Cho nhấp nháy bằng cách đổi mầu 50 ngôi sao */
        for(j=1;j<=50;++j)
          {
            putpixel(xx[i],yy[i],random(getmaxcolor()));
            ++i;
            if(i>1000)
             i=1;
          }
    }
```

§12. IN ẢNH TỪ MÀN HÌNH ĐỒ HOẠ

Hàm in_anh dưới đây sẽ in ảnh trong miền chữ nhật (xt, yt, xd, yd) của màn hình đồ hoạ ra giấy trên các máy in LQ1070, LQ1170 và FX1050.

```
void in_anh(int dd,int xt,int yt,int xd,int yd);
```

Tham số dd là độ đậm của nét in. Thực chất dd là số lần in lại. Bình thường chon dd=1. Nếu muốn in rõ hơn ta chọn dd bằng 2 hay 3.

Trong hàm in_anh có dùng hàm tao_mau, nó được mô tả như sau:

```
int tao_mau(int k,int x,int y);
```

Hàm này sẽ dò trên k chấm điểm theo chiều dọc bắt đầu từ toạ độ (x,y) trên màn hình để biết xem chấm điểm nào đã tô mầu. Hàm sẽ trả về một giá trị nguyên tạo bởi các bit 1 (ứng với điểm đã tô mầu) và 0 (ứng với điểm chưa tô mầu).

Hàm in_anh sẽ dùng hàm tao_mau để duyệt trên miền chữ nhật (xt,yt,xd,yd). Mỗi lần duyệt sẽ nhận được một mẫu các chấm điểm (giá trị nguyên) và mẫu này được in ra giấy.

Dưới đây là nội dung của 2 hàm nói trên.

```
/* in ảnh c*/
        #include  "stdio.h"
        #include  "graphics.h"
        int tao_mau(int k,int x,int y);
        void in_anh(int dd,int xt,int yt,int xd,int yd);
         int tao_mau(int k,int x,int y)
           {
              int c=0,i;
              for(i=0;i<k;++i)
              if(getpixel(x,y+i))
                   c =c | (128>>i);
              return c;
           }
        void in_anh(int dd,int xt,int yt,int xd,int yd)
        {
               /*dd - so lan in lai mot dong */
        char c,ch1;
        int scot, m, mm, k, dong, cot, i, j, n1, n2;
        dong=(yd-yt+1)/6;
        mm=(yd-yt+1) % 6;
        cot=xd-xt+1;
        for(i=0;i<=dong;++i)
           {
           if(i<dong)
             m=6;
           else
             m=mm;
           if(m>0)
             {
```

```
scot=0;
for(j=0;j < cot;++j)
if(tao_mau(m,xt+j,yt+i*6))
   scot=j+1;
if(scot)
  {
   n1=scot % 256;
   n2= scot/256;
   for(k=0;k<dd;++k)
    {
     fprintf(stdprn,"%c%c%c%c%c%c",13,27,'*',0,n1,n2);
         /*LQ*/
     for(j=0;j < scot;++j)
      {
         /*bat phim*/
       if (kbhit())
        {
         if((ch1=getch())==0)
           getch();
         if(ch1==27)
           goto ket;
        }
           c=tao_mau(m,xt+j,yt+i*6);
           fprintf(stdprn,"%c",c);
        }
      }
    }
   fprintf(stdprn,"%c%c%c",27,'A',m);
   fprintf(stdprn,"\n");
  }
 }
ket: fprintf(stdprn,"%c%c",27,'@');
}
```

13.1. Cách sử dụng 256 mầu

Các chương trình đồ hoạ bên trên chỉ sử dụng được 16 mầu. Muốn sử dụng 256 mầu, cần làm như sau:

+ Trong thư mục làm việc (chứa tệp chương trình đồ hoạ đang xây dựng) cần có trình điều khiển đồ hoạ 256 mầu (tệp SVGA256.BGI)

+ Việc khởi động hệ đồ hoạ cần tiến hành theo mẫu:

```
int   mh, mode;
mh = installuserdriver("SVGA256", NULL);
mode = 16;  /* Độ phân giải cao */
initgraph(&mh, &mode,"");
```

13.2. Ví dụ

Chương trình sau minh hoạ cách dùng 256 mầu để vẽ 260 hình chữ nhật có 256 mầu khác nhau.

```
#include  "conio.h"
#include   "graphics.h"
void main()
 {
   int mh, mode, k, i, j;
   mh = installuserdriver("SVGA256", NULL);
   mode = 16;
   initgraph(&mh, &mode, "");
   k=0;
   for (i=1; i<=13; ++i)
   for(j=1; j<=20; ++j)
     {
      ++k ;
      setfillstyle(1,k);
      bar(5+(j-1)*30,20 +(i-1)*25,5 +(j-1)*30 +25,20+ (i-1)*25+20);
     }
    getch();
   closegraph();
 }
```

BÀI TẬP CHƯƠNG 9

Bài 1. Lập chương trình vẽ một tam giác nội tiếp trong hình tròn, hình tròn lại nội tiếp trong ellipse. Sau đó tô các mầu khác nhau cho các miền khác nhau tạo nên từ các hình nói trên.

Bài 2. Lập chương trình vẽ đồ thị của hàm số y = f(x) trên đoạn [a,b].

Bài 3. Vẽ một bánh xe lăn trên đường thẳng.

Bài 4. Viết chương trình vẽ một đồng bạc quay quanh trục thẳng đứng.

Bài 5. Viết chương trình cho phép vẽ trên màn hình bằng các phím mũi tên. ở mỗi vị trí chương trình sẽ chấm một điểm và từ đó có thể tạo các đường thẳng và cong.

Bài 6. Viết chương trình cho phép nhận và trình bầy ký tự trên màn hình đồ hoạ. Lưu ý nếu gõ sai có thể dùng phím BackSpace để xoá ký tự gõ nhầm.

Bài 7. Viết chương trình cho phép chèn và huỷ ký tự của một văn bản ở mode đồ hoạ.

Bài 8. Lập chương trình vẽ một quả cầu dao động ở một đầu giây. Một đầu giây buộc cố định.

CHƯƠNG 10

THAO TÁC TRÊN CÁC TỆP TIN

Chương này trình bày 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,... . 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 (dù cấp 1 hay cấp 2) đều có thể truy xuất theo cả hai kiểu nhị phân và văn bản (xem §2). 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. Đặc trưng cơ bản của chúng là:

+ Không có dịch vụ nhập xuất riêng cho từng kiểu dữ liệu mà chỉ có dịch vụ đọc ghi một dẫy 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, tiếng Anh là handle). Các hàm cấp 1 làm việc với tệp thông qua số hiệu tệp.

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. Chẳng hạn khi ghi một số nguyên thì số được đưa vào vùng đệm và khi nào đầy thì vùng đệm mới được đẩy lên đĩa. Khi đọc, thông tin được lấy từ vùng đệm, và chỉ khi vùng đệm đã trống rỗng thì máy mới lấy dữ liệu từ đĩa chứa vào vùng đệm. Việc sử dụng vùng đệm sẽ giảm số lần nhập xuất trên đĩa và nâng cao tốc độ làm việc. Tuy vậy trong một số trương hợp ta phải nhớ tới tác nghiệp vét vùng đệm để đề phòng 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 chuông hơn. Mặt khác vì các hàm cấp 1 ở mức độ sâu hơn, gần DOS hơn nên tốc độ truy nhập sẽ nhanh hơn.

Dưới đây trong các mục §2 - §8 sẽ trình bầy hệ thống truy xuất cấp 2. Trong các mục còn lại sẽ xét đến các hàm vào/ra cấp 1.

§1. KIỂU NHẬP XUẤT NHỊ PHÂN VÀ VĂN BẢN

Trước khi giới thiệu các kiểu xuất nhập nhị phân và văn bản chúng ta cũng cần biết cấu trúc chung của một tệp tin trên đĩa. Một tệp tin (dù nó được xây dựng bằng cách nào) đơn giản chỉ 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.

1.1. Kiểu nhị phân.

1.1.1. 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.

1.1.2. Mã kết thúc tệp.

Trong khi đọc nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác không. Tại sao lại chọn số -1 làm mã kết thúc tệp? Lý do rất đơn giản: 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.

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

1.2.1. Mã chuyển dòng:

- Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF.

_ Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF.

Ví dụ xét hàm

```
fputc(10,fp);
```

Nếu tệp fp mở theo kiểu nhị phân, hàm sẽ ghi lên tệp một ký tự mã 10. Nhưng nếu fp mở theo kiểu văn bản thì hàm ghi lên tệp hai mã là 13 và 10.

Điều này cốt để phù hợp với DOS, vì các dòng trên tệp văn bản của DOS được kết thúc bởi hai mã 13 và 10.

1.2.2. Mã kết thúc tệp.

Trong khi đọc nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (số -1) và hàm feof(fp) cho giá trị khác không (số 1). Điều này cốt để phù hợp với một số hệ soạn thảo (như C, Pascal). Tệp soạn thảo trong các hệ này kết thúc bởi mã 26.

1.2.3. Tệp văn bản của DOS. Kiểu văn bản trong C tạo ra các tệp văn bản có cấu trúc như các tệp văn bản chuẩn của DOS. Các hệ soạn thảo của C, Pascal, Foxpro, Sidekick cũng tạo ra các tệp như vậy. Như vậy có thể dùng tệp văn bản để nhận kết quả từ một chương trình C dùng cho chương trình Pascal. Một ứng dụng khác của kiểu nhập xuất văn bản là: Các kết quả của chương trình có thể đưa ra một tệp văn bản, rồi in tệp này ở bất kỳ chỗ nào khác có máy in.

1.3. Ví dụ minh hoạ.

Chương trình sau tạo 2 tệp có tên là vb và np. Trong chương trình dùng các hàm:

+ fopen để mở tệp,
+ fputc để ghi một ký tự lên tệp,
+ fclose để đóng tệp.

```
#include <stdio.h>
main()
  {
  FILE *fvb, *fnp; /* Khai báo 2 biến con trỏ tệp */
                /* Mở tệp vb để ghi theo kiểu văn bản
                   gán con trỏ tệp cho biến fvb */
  fvb=fopen("vb","wt");
               /* Mở tệp np để ghi theo kiểu nhị phân
                  gán con trỏ tệp cho biến fnp */
    fnp=fopen("np","wb");
    /* Ghi các ký tự lên tệp fvb */
    fputc('A',fvb);
    fputc(26,fvb);
    fputc(10,fvb);
    fputc('B',fvb);
    /* Ghi các ký tự lên tệp fnp */
    fputc('A',fnp);
    fputc(26,fnp);
    fputc(10,fnp);
    fputc('B',fnp);
    fclose(fvb); /* Đóng tệp fvb */
    fclose(fnp);  /* Đóng tệp fnp */
  }
```

Kết quả:

+ Tệp vb có các ký tự ứng với các mã: 65 26 13 10 66

+ Tệp np có các ký tự ứng với các mã: 65 26 10 66

Chú ý: Nếu dùng kiểu văn bản để đọc tệp vb hay tệp np, thì ta chỉ nhận được một ký tự đầu (mã 65) vì khi gặp ký tự thứ hai (mã 26) thì ta nhận được mã kết thúc EOF.

1.4. Một ví dụ khác. Trong 1.3 đã nhắc nhở rằng: muốn đọc tất cả các ký tự của tệp, ta cần dùng hàm fgetc theo kiểu nhị phân. Ta xét thêm một ví dụ nữa để thấy việc chọn kiểu là cần thiết. Xét chương trình:

```
#include <stdio.h>
main()
  {
    FILE f;        /* Khai báo biến con trỏ tệp */
                   /* Mở tệp sl để ghi theo kiểu văn bản
                      Gắn tệp sl với con trỏ f */
    f=fopen("sl","wt");
          /* Ghi 3 dòng lên tệp f */
    fprintf(f,"%2d\n%2d\n%2d",56,7,8);
          /* Đóng tệp */
     fclose(f); /* Đóng tệp f */
  }
```

Hàm fprintf đưa kết quả ra tệp theo cách như hàm printf. Vì tệp f mở theo kiểu văn bản nên ký tự xuống dòng '\n' được ghi thành 2 mã 13 và 10. Kết quả là 10 ký tự ứng với các mã sau được ghi lên tệp:

53 54 13 10 32 55 13 10 32 56

(53 là mã của chữ số 5, 32 là mã của khoảng trống,...)

Cấu trúc trên của tệp sl phù hợp với DOS, vì vậy khi dùng lệnh type (của DOS) có thể cho hiện hay in nội dung của tệp sl thành 3 dòng.

Bây giờ nếu ta mở tệp sl theo kiểu nhị phân bằng cách dùng câu lệnh:

f=fopen("sl","wb");

thì tệp sl sẽ gồm 8 mã sau:

53 54 10 32 55 10 32 56

Cấu trúc này không phù hợp với DOS và khi dùng lệnh TYPE sẽ không nhân được 3 dòng mong muốn.

§2. GIỚI THIỆU CHUNG VỀ CÁC HÀM XỬ LÝ TỆP

Các hàm cấp 2 sử dụng cấu trúc FILE và mã kết thúc EOF, tất cả đều được khai báo và định nghĩa trong <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 cả hai kiểu, nhưng như đã nói ở trên, 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.

2.1. Các hàm dùng chung cho cả hai kiểu.

1. fopen dùng để mở tệp. Trước khi làm việc với tệp cần phải mở nó.
2. fclose dùng để đóng tệp. Cần đóng tệp trước khi kết thúc chương trình hay khi không làm việc với tệp nữa.
3. fcloseall dùng để đóng tất cả các tệp đang mở.
4. fflush dùng để làm sạch vùng đệm của tệp.
5. fflushall dùng để làm sạch vùng đệm của các tệp đang mở.
6. ferror cho biết có lỗi (khác 0) hay không lỗi (bằng 0).
7. perror thông báo lỗi trên màn hình (khi biết có lỗi).
8. feof cho biết đã gặp cuối tệp hay chưa.
9. unlink và remove dùng để loại tệp trên đĩa.
10. rewind dùng để chuyển con trỏ chỉ vị về đầu tệp.
11. fseek dùng để di chuyển con trỏ chỉ vị đến vị trí bất kỳ trên tệp (hàm này chỉ nên dùng theo kiểu nhị phân).
12. ftell cho biết vị trí hiện tại của con trỏ chỉ vị.

2.2. Các hàm nhập xuất ký tự. Các hàm này có thể dùng trong kiểu nhị phân cũng như văn bản.

13. putc và fputc dùng để ghi ký tự lên tệp.
14. getc và fgetc dùng để đọc ký tự từ tệp.

2.3. Các hàm nhập xuất theo kiểu văn bản. Các hàm này chỉ nên dùng theo kiểu văn bản.

15. fprintf dùng để ghi dữ liệu theo khuôn dạng lên tệp.
16. fscanf dùng để đọc dữ liệu từ tệp theo khuôn dạng.

17. fputs dùng để ghi một chuỗi ký tự lên tệp.

18. fgets dùng để đọc một dẫy ký tự từ tệp.

2.4. Các hàm nhập xuất theo kiểu nhị phân. Các hàm này chỉ nên dùng theo kiểu nhị phân.

19. putw dùng để ghi một số nguyên (2 byte) lên tệp.

20. getw dùng để đọc một số nguyên (2 byte) từ tệp.

21. fwrite dùng để ghi một số mẫu tin lên tệp.

22. fread dùng để đọc một số mẫu tin từ tệp.

§3. ĐÓNG MỞ TỆP, XOÁ VÙNG ĐỆM VÀ KIỂM TRA LỖI

Các hàm trong mục này dùng chung cho cả hai kiểu nhị phân và văn bản.

3.1.Hàm fopen: Mở tệp.

+ Dạng hàm:

```
FILE *fopen(const char *tên_tệp, const char *kiểu);
```

+ Các đối:

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Kiểu có thể có các giá trị sau:

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

"w+" "w+t"	Mở một tệp mới để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

+ **Công dụng:** Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp 2 sẽ làm việc với tệp thông qua con trỏ này. 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 hoặc từ ghi sang đọc. Các hàm fflush và hàm di chuyển đầu từ đều làm được chuyện này.

3.2. Hàm fclose: đóng tệp.

+ **Dạng hàm:**

int fclose(FILE *fp);

+ **Đối:**

fp là con trỏ tương ứng với tệp cần đóng.

+ **Công dụng:** Hàm dùng để đóng tệp. Nội dung đóng tệp gồm:

- Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi)
- Xoá 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, trái lại hàm cho EOF.

3.3. Hàm fcloseall: đóng các tệp đang mở.

+ **Dạng hàm:**

int fcloseall(void);

+ **Công dụng:** Hàm dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đóng được, trái lại hàm cho EOF.

3.4. Hàm fflush: làm sạch vùng đệm.

+ **Dạng hàm:**

int fflush(FILE *fp);

+ **Đối:**

fp là con trỏ tệp.

+ **Công dụng:** Hàm dùng làm sạch vùng đệm của tệp fp. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

3.5. Hàm fflushall: làm sạch vùng đệm.

+ **Dạng hàm:**

int fflushall(void);

+ **Công dụng:** Hàm dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đang mở, trái lại hàm cho EOF.

3.6. Hàm ferror; kiểm tra lỗi.

+ **Dạng hàm:**

int ferror(FILE *fp);

+ **Đối:**

fp là con trỏ tệp.

+ **Công dụng:** Hàm dùng để kiểm tra lỗi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không lỗi, trái lại hàm cho giá trị khác không.

3.7. Hàm perror: thông báo lỗi hệ thống.

+ **Dạng hàm:**

void perror(const char *s);

+ **Đối:**

s là con trỏ trỏ tới một chuỗi ký tự.

+ **Công dụng:** Hàm in chuỗi s và thông báo lỗi.

3.8.Hàm feof: kiểm tra cuối tệp.

+ **Dạng hàm:**

int feof(FILE *fp);

+ **Đối:** fp là con trỏ tệp.

+ **Công dụng:** Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc,trái lại hàm cho giá trị 0.

3.9.Hàm unlink: xoá tệp.

+ **Dạng hàm:**

int unlink(const char *tên_tệp);

+ **Đối:** là tên tệp cần xoá.

+ **Công dụng:** Hàm dùng để xoá một tệp trên đĩa. Nếu thành công hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

3.10. Hàm remove: xoá tệp.

+ **Dạng hàm:**

```
remove(const char *tên_tệp);
```

+ **Đối:** là tên tệp cần xoá.

+ **Công dụng:** Hàm dùng để xoá một tệp trên đĩa. Nó hàm macro gọi tới unlink.

Ví dụ: Đoạn chương trình sau sẽ mở một tệp và kiểm tra lỗi.

```
FILE *fp;  /* Biến con trỏ tệp */
           /* Mở tệp so_lieu để đọc theo kiểu nhị phân.
              Nếu thành công con trỏ tệp so_lieu gán cho biến fp.
              Các hàm sẽ làm việc với tệp so_lieu thông qua fp.
              Vì vậy ta có nói tệp so_lieu hay tệp fp.
              */
fp = fopen("so_lieu","rb");
              /* Kiểm tra lỗi */
if(fp==NULL)
perror("Lỗi khi mở tệp so_lieu: ");
```

§4. NHẬP XUẤT KÝ TỰ

Các hàm nhập xuất ký tự dùng được cả trong kiểu nhị phân và ký tự, nhưng tác dụng có những điểm khác nhau.

4.1. Các hàm putc và fputc.

+ **Dạng hàm:**

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

+ **Đối:** ch là một giá trị nguyên, fp là con trỏ tệp.

+ **Công dụng:** Hàm ghi lên tệp fp một ký tự có mã bằng:

m= ch % 256

trong đó ch được xem là số nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại hàm cho EOF.

Ví dụ câu lệnh:

```
putc(-1,fp);
```

sẽ ghi lên tệp fp mã 255, vì dạng không dấu của -1 là 65535.

+ Ghi chú:

- Hai hàm trên có ý nghiã như nhau.

- Trong kiểu văn bản, nếu m=10 thì hàm sẽ ghi lên tệp hai mã 13 và 10.

4.2.Các hàm getc và fgetc.

+ Dạng hàm:

```
int getc(FILE *fp);
int fgetc(FILE *fp);
```

+ Đối: fp là con trỏ tệp.

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

+ Ghi chú:

- Hai hàm trên có ý nghiã như nhau.

- Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10.

- Trong kiểu văn bản, khi gặp mã 26 thì hàm không trả về 26 mà trả về EOF.

4.3. Các ví dụ minh hoạ.

Ví dụ 1: Chương trình sao tệp.

```
/*SAO TEP Dùng mode nhị phân và getc, putc */
    #include "stdio.h"
    main()
      {
        int c;
        char t1[14],t2[14];
        FILE *f1,*f2;
        printf("\n TEP NGUON: ");
        gets(t1);
        printf("\n TEP DICH: ");
        gets(t2);
        f1=fopen(t1,"rb");
        if(f1==NULL)
```

```
    {
        printf("\n TEP %s khong ton tai",t1);
        getch();
        exit(1);
    }
    f2=fopen(t2,"wb");
    /* Việc sao tệp được thực hiện ở đoạn này */
    while ((c=fgetc(f1))!=EOF)
        fputc(c,f2);
    fclose(f1);
    fclose(f2);
}
```

Chương trình thực hiện sao tệp theo thuật toán sau:

1. Đọc một ký tự của tệp f1, kết quả đặt vào biến c.

2. Nếu c bằng EOF thì kết thúc.

Nếu c khác EOF thì ghi c vào tệp f2 rồi trở lại 1.

Nhận xét 1. Nếu trong 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 sang tệp f2.

Nhận xét 2. Nếu dùng hàm feof và thuật toán:

1. Nếu feof(f1) khác không thì kết thúc, trái lại chuyển xuống 2.

2. Đọc một ký tự từ tệp f1, ghi lên tệp f2, rồi trở lại 1, thì nhận được đoạn chương trình:

```
while (!feof(f1))
fputc(fgetc(f1),f2);
```

Nhưng đoạn chương trình sau không thật đúng. Tệp f2 sẽ dài hơn tệp f1 đúng một byte có giá trị 255. Lý do cũng dễ giải thích. Giả sử tệp f1 có đúng một ký tự mã 65, khi đó thuật toán sẽ diễn ra như sau:

1. Đầu từ đang trỏ vào ký tự A nên feof(f1) = 0, chuyển xuống 2.

2. Đọc ký tự A của f1 và ghi lên f2, trở lại 1.

3. Đầu đọc đặt ở cuối tệp f1 nhưng chưa có thao tác đọc nên feof(f1) vẫn bằng 0, chuyển xuống 2.

4. Đọc một ký tự của f1. Khi đó ta nhận được -1. Ghi -1 lên f2 thì mã 255 sẽ được ghi. Ngoài ra, do khi đọc từ f1 gặp phải cuối tệp nên lúc này feof(f1) khác 0.

Đến đây thuật toán kết thúc.

Ví dụ 2: Chương trình cho hiên lên màn hình các ký tự của tệp. Trên mỗi dòng đặt 10 ký tự. Mỗi ký tự sẽ được thể hiện bởi 2 thông số là mã và dạng ký tự. Sau khi cho hiện 24 dòng thì chương trình tạm dừng. Bấm phím bất kỳ để xem tiếp.

```
/*DOC TEP*/
        #include "stdio.h"
        #include "conio.h"
        main()
          {
            int c,i,k;
            char tep[14];
            FILE *fp;
            printf("\n TEP NGUON: ");  gets(tep);
            fp=fopen(tep,"rb");
            if(fp==NULL)
              {
                printf("\n TEP %s khong ton tai",tep);
                getch(); exit(1);
              }
            i=0; k=0; clrscr();
          while ((c=fgetc(fp))!=EOF)
            {   /* In mã */
              printf("%-4d",c);
              if (c<32)  c=46;
                printf("%-4c",c);  ++i;    /* In ký tự */
              if(i==10)
                {
                  i=0;  ++k;
                }
              if(k==24)
                {
                  getch();  k=0;   clrscr();
                }
            }
           fclose(fp);
```

§5. CÁC HÀM NHẬP XUẤT THEO KIỂU VĂN BẢN

Các hàm trong mục này chỉ nên dòng theo kiểu văn bản.

5.1 Hàm fprintf: ghi dữ liệu theo khuôn dạng.

+ Dạng hàm:

int fprintf(FILE *fp,const char *dk,...);

+ Đối:

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

... là danh sách các đối mà giá trị của chúng cần ghi lên tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm printf.

+ Công dụng: Giá trị các đối được ghi lên tệp fp theo khuôn dạng xác định trong chuỗi điều khiển dk. Nếu thành công, hàm trả về một giá trị nguyên bằng số byte ghi lên tệp. Khi có lỗi hàm cho EOF. Hàm làm việc giống như printf.

+ Ví dụ: Xét chương trình:

```
#include <stdio.h>
main()
  {
     FILE *f;
     int i;
     f=fopen("text","wt");
     fprintf(f,"Cac dong");
     for(i=1;i<=2;++i)
     fprintf(f,"\nDong%2d",i);
     fclose(f);
  }
```

Chương trình trên sẽ tạo ra tệp văn bản text gồm 3 dòng với nội dung như sau:

Cac dong

Dong 1 Dong 2

5.2.Hàm scanf: Đọc dữ liệu từ tệp theo khuôn dạng

Dạng hàm:

int fscanf(FILE *fp, const char *dk, ...);

+ **Đối:**

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

... là danh sách các đối 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 (đặc tả) trong dk và lưu kết quả vào các đối. Hàm làm việc giống như scanf. Hàm trả về một giá trị bằng số trường được đọc.

+ **Ví dụ 1:** Giả sử có tệp kiểu văn bản "da_giac.sl" chứa thông tin về một đa giác. Tệp gồm n+1 dòng với nội dung sau:

Dòng 1:	n	(số đỉnh)
Dòng 2:	x1 y1	(toạ độ đỉnh 1)
Dòng 3:	x2 y2	(toạ độ đỉnh 2)
. . .		
Dòng n+1:	xn yn	(toạ độ đỉnh n)

Chương trình sau sẽ đọc số dỉnh và toạ độ các đỉnh từ tệp "da_giac.sl"

```
#include <stdio.h>
main()
  {
    FILE *f; int i,n;
    float x[50],y[50];
    f=fopen("da_giac.sl","rt");
    fscanf(f,"%d",&n);
    for(i=1;i<=n;++i)
    fscanf(f,"%f%f", &x[i], &y[i]);
    fclose(f);
  }
```

+ **Ví dụ 2:** Giả sử có một dãy số nguyên ghi trên tệp "sl.sl". Giữa hai số nguyên có ít nhất một khoảng trống hay các dấu xuống dòng. Yêu cầu đọc và in ra màn hình dãy số nói trên.

Ta phân biệt hai trường hợp:

1. Sau chữ số cuối cùng là mã 26 hay cuối tệp.

2. Sau chữ số cuối cùng có ít nhất một khoảng trống hay các dấu xuống dòng.

Dưới đây là hai chương trình ứng với hai trường hợp nêu trên.

/* Đọc dãy số theo trường hợp 1 */

```
#include <stdio.h>
main()
  {
    FILE *f; int c;
    f=fopen("sl.sl","r");
    while(!feof(f))
     {
       fscanf(f,"%d",&c);
       printf("\n%d",c);
     }
    fclose(f);
    getch();
  }
```

/* Đọc dãy số theo trường hợp 2 */

```
#include <stdio.h>
main()
  {
    FILE *f; int c;
    f=fopen("sl.sl","r");
    while(1)
     {
       fscanf(f,"%d",&c);
       if(feof(f)) break;
       printf("\n%d",c);
     }
    fclose(f);
    getch();
  }
```

5.3. Hàm fputs: ghi một chuỗi ký tự lên tệp.

+ Dạng hàm:

int fputs(const char *s,FILE *fp);

+ Đối:

s là con trỏ trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu '\0'.

fp là con trỏ tệp.

+ **Công dụng:** Ghi chuỗi s lên tệp fp (dấu '\0' không ghi lên tệp). Khi thành công, hàm trả về ký tự cuối cùng được ghi lên tệp. Khi có lỗi hàm cho EOF.

+ **Ví dụ:** Chương trình sau sẽ nhập các dòng ký tự từ bàn phím và ghi lên tệp "dong_vb".

```
/* ghi các dòng văn bản lên tệp */
    #include <stdio.h>
    #include <conio.h>
    main()
      {
        int i=0;
        char d[256];
        FILE *f;
        f=fopen("dong_vb","w");
        clrscr();
        while(1)
          {
            ++i;
            printf("\nDong %d (Bam Enter ket thuc): ",i);
            gets(d);
            if(d[0]=='\0')
              break;
            if(i>1)
              fputc(10,f);
              fputs(d,f);
          }
        fclose(f);
      }
```

5.4. Hamf fgets: đọc một dãy ký tự từ tệp.

+ Dạng hàm:

char *fgets(char *s, int n, FILE *fp);

+ Đối:

s là con trỏ (kiểu char)trỏ tới một vùng nhớ đủ lớn để chứa chuỗi ký tự đọc từ tệp.

n là số nguyên xác định độ dài cực đại của dãy cần đọc.

fp là con trỏ tệp.

+ Công dụng: Đọc một dãy ký tự từ tệp fp chứa vào vùng nhớ s.

Việc đọc kết thúc khi:

- Hoặc đã đọc n-1 ký tự.

- Hoặc gặp dấu xuống dòng (Cặp mã 13 10). Khi đó mã 10 được đưa vào xâu kết quả.

- Hoặc kết thúc tệp.

Xâu kết quả sẽ được bổ sung thêm dấu hiệu kết thúc chuỗi '\0'.

Khi thành công, hàm trả địa chỉ vùng nhận kết quả. Khi có lỗi hoặc gặp cuối tệp, hàm cho giá trị NULL.

+ Ví dụ: Chương trình sau sẽ đọc các dòng ký tự trên tệp "dong_vb" (xem 5.3) và in ra màn hình.

```
/* Đọc các dòng ký tự
   trên tệp văn bản */
        #include <stdio.h>
        #include <conio.h>
        main()
          {
            int i=0;
            char d[256];
            FILE *f;
            f=fopen("dong_vb","r");
            clrscr();
            while(!feof(f))
              {
                ++i;
                fgets(d,256,f);
                printf("Dong %d: %s",i,d);
              }
            fclose(f);
            getch();
          }
```

§6. TỆP VĂN BẢN VÀ CÁC THIẾT BỊ CHUẨN

Có thể dùng các hàm nhập xuất văn bản (theo kiểu văn bản) trên các thiết bị chuẩn. C đã định nghĩa các tệp tin và con trỏ tệp ứng với các thiết bị chuẩn như sau:

Tệp	**Con trỏ**	**Thiết bị**
In	stdin	Thiết bị vào chuẩn (bàn phím)
out	stdout	Thiết bị ra chuẩn (màn hình)
err	stderr	Thiệt bị lỗi chuẩn (màn hình)
prn	stdprn	Thiết bị in chuẩn (máy in)

Khi chương trình C bắt đầu làm việc thì các tệp này được tự động mở, vì vậy có thể dùng các con trỏ nêu trên để nhập xuất trên các thiết bị chuẩn. Dưới đây là một chương trình minh hoạ:

```
/* Chương trình minh hoạ cách dùng các con trỏ tệp tin chuẩn */
      #include "stdio.h"
      #include "conio.h"
      main()
       {
          char ht[25];
          float diem;
          int ns;
          printf("\nHo ten: ");
          fgets(ht,25,stdin);
          printf("\ndiem va ns: ");
          fscanf(stdin,"%f%d",&diem,&ns);
          fputs(ht,stderr);
          fprintf(stdout,"Diem %f ns %d",diem,ns);
       }
```

§7. CÁC HÀM NHẬP XUẤT THEO KIỂU NHỊ PHÂN

Các hàm trong mục này chỉ nên dùng theo kiểu nhị phân.

7.1. Hàm putw: ghi một số nguyên.

+ Dạng hàm:

int putw(int n, FILE *fp);

+ Đối:

n là giá trị nguyên,

fp là con trỏ tệp.

+ **Công dụng:** Ghi giá trị n lên tệp fp dưới dạng 2 byte. Nếu thành công, hàm trả về số nguyên được ghi. Khi có lỗi hàm trả về EOF.

7.2. Hàm getw: đọc một số nguyên.

+ **Dạng hàm:**

int getw(FILE *fp);

+ **Đối:**

fp là con trỏ tệp.

+ **Công dụng:** Đọc một số nguyên (2 byte) từ tệp fp. Nếu thành công, hàm trả về số nguyên đọc được. Khi có lỗi hoặc gặp cuối tệp, hàm trả về EOF.

+ **Ví dụ:** Chương trình sau minh hoạ cách dùng các hàm putw và getw. Đầu tiên ghi một dãy số nguyên lên tệp "so_ng", sau đó đọc các số nguyên từ tệp này và in ra màn hình.

```
/*  Chương trình ghi, đọc số nguyên
    minh hoạ cách dùng các hàm: putw và getw */
#include "stdio.h"
#include "conio.h"
main()
  {
    FILE *f;
    int i;
            /* Ghi các số nguyên */
    f=fopen("so_ng","wb");
    for(i=1000;i<=1010;++i)
    putw(i,f);
    fclose(f);
      /* Đọc các số nguyên */
    clrscr();
    f=fopen("so_ng","rb");
    while((i=getw(f))!=EOF)
    printf("\n%d",i);
    fclose(f);
  }
```

7.3. Hàm fwrite: ghi các mẫu tin lên tệp.

+ Dạng hàm:

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

+ Đối:

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi,

size là kích thước của mẫu tin theo byte,

n là số mẫu tin cần ghi,

fp là con trỏ tệp.

+ Công dụng: Ghi n mẫu tin kích thước size byte từ vùng nhớ ptr lên tệp fp. Hàm trả về một giá trị bằng số mẫu tin thực sự ghi được.

7.4. Hàm fread: đọc các mẫu tin từ tệp.

+ Dạng hàm:

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

+ Đối:

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,
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.

Các ví dụ minh hoạ fwrite, fread.

Các hàm fread, fwrite thường dùng để đọc ghi các đối tượng có cùng độ lớn như cấu trúc, số thực,... . Dưới đây là 3 ví dụ minh hoạ.

Ví dụ 1: Chương trình dưới đây thực hiện việc sao tệp bằng các hàm fread và fwrite.

```
/* SAO TEP dung fread, fwrite */
  #include "stdio.h"
  main()
    {
      int n;
      char t1[14],t2[14],c[1000];
      FILE *f1,*f2;
      printf("\n TEP NGUON: ");
      gets(t1);
```

```
        printf("\n TEP DICH: ");
        gets(t2);
        f1=fopen(t1,"rb");
        if(f1==NULL)
          {
            printf("\n TEP %s khong ton tai",t1);
            getch();
            exit(1);
          }
        f2=fopen(t2,"wb");
        while ( (n=fread(c,1,1000,f1))>0 )
        fwrite(c,1,n,f2);
        fclose(f1);
        fclose(f2);
      }
```

Ví dụ 2: Chương trình dưới đây minh hoạ cách dùng các hàm fread và fwrite để ghi đọc một dãy n phần tử (nguyên, thực, cấu trúc).

```
/* Dùng fread và fwrite để ghi đọc một dãy n phần tử */
      #include "stdio.h"
      #include "conio.h"
      typedef struct
            {
              char ht[25];
              int ns;
            } hs;
/* Khai báo và khởi đầu mảng cấu trúc ds */
      hs ds[50]={
                  {"",0},
                  {"Pham Thu Huong",1976},
                  {"Nguyen Viet Dung",1978},
                  {"Thanh Tam",1974}
               };
      main()
        {
          FILE *fp;
          float x[50]; int a[50];
```

```
int i;
      /* Gán giá trị cho a[i] và x[i] */
for(i=1;i<=10;++i)
    a[i]=100*i;
for(i=1;i<=5;++i)
    x[i]=1.0/i;
      /* Ghi mảng a, x và ds từ phần tử có chỉ số 1 */
fp=fopen("mang.sl","wb");
fwrite(a+1,sizeof(int),10,fp);
fwrite(x+1,sizeof(float),5,fp);
fwrite(ds+1,sizeof(hs),3,fp);
fclose(fp);
      /* Đọc từ tệp và đưa ra mảng từ phần tử chỉ số 11 */
fp=fopen("mang.sl","rb");
fread(a+11,sizeof(int),10,fp);
fread(x+11,sizeof(float),5,fp);
fread(ds+11,sizeof(hs),3,fp);
fclose(fp);
      /* In */
for(i=11;i<=20;++i)
    printf("\n%d", a[i]);
for(i=11;i<=15;++i)
    printf("\n%f", x[i]);
for(i=11; i<=13; ++i)
    printf("\n%s %d", ds[i].ht,ds[i].ns);
}
```

Ví dụ 3: Chương trình dưới đây minh hoạ cách dùng các hàm fread và fwrite để ghi đọc các cấu trúc (chưa biết rõ số lượng là bao nhiêu)

```
    /* đọc/ghi cấu trúc bằng fread fwrite */
#include "stdio.h"
#include "conio.h"
typedef struct
    {
     char ht[25];
     int ns;
    } hs;
```

```
main()
{
    FILE *fp;
    hs h;
        /* Nhập số liệu từ bàn phím rồi ghi lên tệp */
    fp=fopen("hs.sl","wb");
    while(1)
      {
        printf("\nHo ten (Bấm Enter kết thúc): ");
        gets(h.ht);
        if(h.ht[0]=='\0')
          break;
        printf("\nNam sinh ");
        scanf("%d%*c",&h.ns);
        fwrite(&h,sizeof(hs),1,fp);
      }
    fclose(fp);
          /* Đọc dữ liệu từ tệp và in ra màn hình */
    fp=fopen("hs.sl","rb");
    while( fread(&h,sizeof(hs),1,fp)>0)
    printf("\n%s %d",h.ht,h.ns);
    fclose(fp);
}
```

§8. NHẬP XUẤT NGẪU NHIÊN

Mỗi tệp khi đang mở có một con trỏ chỉ vị (File position locator) dùng để xác định vị trí đọc ghi trên tệp. Khi mở tệp tin để đọc hay ghi, con trỏ chỉ vị luôn luôn ở đầu tệp tin (byte 0). Nhưng nếu tệp được mở theo mode "a" thì con trỏ chỉ vị ở cuối tệp để ghi thêm dữ liệu vào tệp này. Việc nhập xuất dữ liệu được bắt đầu từ vị trí hiện tại của con trỏ chỉ vị và sau khi hoàn thành thì con trỏ này dịch chuyển một số byte chính bằng số byte đã đọc hay ghi. Như vậy việc nhập xuất được tiến hành tuần tự theo chiều từ đầu đến cuối tệp tin.

Trong mục này sẽ nói cách di chuyển con trỏ chỉ vị đến vị trí mong muốn, nhờ đó ta có thể nhập xuất tại bất kỳ chỗ nào trên tệp tin. Cách làm này gọi là nhập xuất ngẫu nhiên (Random Access I/O) rất tiện lợi cho việc sữa chữa, bổ sung dữ liêu trực tiếp trên tệp.

8.1. Hàm rewind: chuyển con trỏ chỉ vị về đầu tệp.

+ Dạng hàm:

void rewind(FILE *fp);

+ Đối:

fp là con trỏ tệp.

+ Công dụng: Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu tệp.

8.2.Hàm fseek: di chuyển con trỏ chỉ vị đến vị trí mong muốn.

+ Dạng hàm:

int fseek(FILE *fp, long sb, int xp);

+ Đối:

fp là con trỏ tệp,
sb là số byte cần di chuyển,
xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ 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 chỉ vị.
xp = SEEK_END hay 2: Xuất phát từ cuối tệp.

+ Công dụng: Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối cua sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

+ Chú ý: Không nên dùng fseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

8.3. Hàm ftell: cho biết vị trí hiện tại của con trỏ chỉ vị.

+ Dạng hàm:

long ftell(FILE *fp);

+ Đối:

fp là con trỏ tệp.

+ Công dụng: Khi thành công hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp fp). Số thứ tự của byte được tính từ 0. Khi có lỗi hàm trả về -1L.

+ **Minh hoạ:** Các ví dụ sau sẽ giải thích thêm cách làm việc của hàm fseek và ftell.

Giả sử tệp fp có 3 ký tự lần lượt là A, B và C.

1. Sau câu lệnh

    fseek(fp,0,SEEK_END);

    thì con trỏ chỉ vị ở cuối tệp và ftell(fp) = 3.

2. Sau câu lệnh

    fseek(fp,-1,SEEK_END);

    thì con trỏ chỉ vị đặt tại C và ftell(fp) = 2.

3. Sau câu lệnh

    fseek(fp,0,SEEK_SET);

    thì con trỏ chỉ vị đặt tại A và ftell(fp) = 0.

Hai ví dụ minh hoạ.

+ **Ví dụ 1:** Chương trình dưới đây dùng các hàm fseek và ftell để xác định độ dài của tệp bất kỳ.

```
/* Dung fseek, ftell xac dinh do dai tep tin */
#include "stdio.h"
#include "conio.h"
main()
  {
    FILE *fp;
    long n;
    char tep[14];
    clrscr();
    printf("\nTen tep: "); gets(tep);
    fp=fopen(tep,"rb");
    if(fp==NULL)
      {
        printf("\nTen %s khong ton tai ",tep);
        exit(1);
      }
    fseek(fp,0,SEEK_END);
    n=ftell(fp);
    fclose(fp);
    printf("\nDo dai cua tep %s la %ld byte",tep,n);
  }
```

+ **Ví dụ 2:** Chương trình dưới đây dùng các hàm fseek, ftell, fread và fwrite để tổ chức một hệ quản trị dữ liệu với 4 chức năng: tạo mới, bổ sung, sửa chữa và xem dữ liệu.

```
/*
   Dùng các mode wb, ab, rb, r+b và các hàm f
   write, fread, fseek, ftell để tổ chức cơ sở dữ liệu
*/
      #include "conio.h"
      #include "stdio.h"
      void taomoi(void);
      void bosung(void);
      void suachua(void);
      void xem(void);
      typedef struct
            {
              char ht[25];
              int bl;
              float nc;
            } person;
      long size = sizeof(person);
      main()
        {
          int sg;
          while (1)
            {
              clrscr();
              printf("\n1.Tao moi");
              printf("\n2.Bo sung");
              printf("\n3.Sua chua");
              printf("\n4.Xem du lieu");
              printf("\n5.Ket thuc");
              sg=getch();
              if(sg=='1')
                 taomoi();
              else if (sg=='2')
                 bosung();
```

```
            else if (sg=='3')
                suachua();
            else
              if (sg=='4')
                xem();
            else
              {
                clrscr();
                exit(1);
              }
          }
      }
void taomoi(void)
  {
    person cb;
    FILE *fp;
    int n=0;
    fp=fopen("dscb.sl","wb");
    while(1)
      {
        clrscr();
        printf("\nNguoi thu %d\n\n",n+1);
        printf("\nHo ten (Bam Enter ket thuc): ");
        gets(cb.ht);
        if(cb.ht[0]==0) break;
        printf("\nBac luong: ");
        scanf("%d",&cb.bl);
        printf("\nNgay cong: ");
        scanf("%f%*c",&cb.nc);
        ++n;
        fwrite(&cb,size,1,fp);
      }
    fclose(fp);
  }
void bosung(void)
  {
```

```
        person cb;
        FILE *fp;
        long n;
        fp=fopen("dscb.sl","ab");
    /* Xac dinh n = so ban ghi */
    fseek(fp,0,SEEK_END);
    n=ftell(fp)/size;
    while(1)
        {
          clrscr();
          printf("\nNguoi thu %ld\n\n",n+1);
          printf("\nHo ten (Bam Enter ket thuc): ");
          gets(cb.ht);
          if(cb.ht[0]==0) break;
          printf("\nBac luong: ");
          scanf("%d",&cb.bl);
          printf("\nNgay cong: ");
          scanf("%f%*c",&cb.nc);
          ++n;
          fwrite(&cb,size,1,fp);
        }
       fclose(fp);
    }
void suachua(void)
    {
       person cb;
       FILE *fp; long i,n;
       if( (fp=fopen("dscb.sl","r+b"))==NULL)
         {
           printf("\nChua co du lieu");
           return;
         }
       /* Xac dinh n = so ban ghi */
    fseek(fp,0,SEEK_END);
    n=ftell(fp)/size;
```

```
while(1)
    {
        clrscr();
        printf("\nSua nguoi thu (n<=%ld)",n);
        scanf("%ld%*c",&i);
        if(i>0&&i<=n)
            {
                fseek(fp,(i-1)*size,SEEK_SET);
                fread(&cb,size,1,fp);
                printf("\nSo lieu hien tai:");
                printf("\nHo ten: %s",cb.ht);
                printf("\nBac luong: %d",cb.bl);
                printf("\nNgay cong: %0.2f",cb.nc);
                printf("\n\nSua la:");
                printf("\nHo ten (Bam Enter ket thuc): ");
                gets(cb.ht);
                if(cb.ht[0]==0) break;
                printf("\nBac luong: ");
                scanf("%d",&cb.bl);
                printf("\nNgay cong: ");
                scanf("%f%*c",&cb.nc);
                printf("\nCo Ghi sua ? - C/K");
                if(toupper(getch())=='C')
                    {
                        fseek(fp,m,SEEK_SET);
                        fwrite(&cb,size,1,fp);
                    }
            }
    }
    fclose(fp);
}
void xem(void)
{
    person cb;
    FILE *fp;
    int i=0;
```

```
    if( (fp=fopen("dscb.sl","rb"))==NULL)
      {
        printf("\nChua co du lieu");
        return;
      }
    clrscr();
    while( fread(&cb,size,1,fp)>0 )
      {
        ++i;
        printf("\n\nNguoi %d",i);
        printf("\nHo ten: %s",cb.ht);
        printf("\nBac luong: %d",cb.bl);
        printf("\nNgay cong: %0.2f",cb.nc);
      }
    fclose(fp);
    getch();
}
```

§9. CÁC HÀM NHẬP XUẤT CẤP 1

9.1. Các tệp tiêu đề và biến chuẩn.

Để sử dụng các hàm cấp 1, ta cần tới các tệp tiêu đề sau:

io.h (chứa các nguyên mẫu của các hàm cấp 1)
fcntl.h (chứa định nghĩa quyền truy nhập (access))
sys/stat.h (Chứa định nghĩa thuộc tính (amode))
dos.h (chứa định nghĩa thuộc tính (attribute) theo DOS)

Ngoài ra còn cần đến biến chuẩn của Turbo C _fmode (định nghĩa trong fcntl.h và stdlib.h) để xác định kiểu nhập xuất (nhị phân hay văn bản).

9.2. Tóm tắt các hàm.

1. Hàm creat dùng để tạo một tệp mới.
2. Hàm _creat dùng để tao một tệp mới theo kiểu nhị phân.
3. Hàm open dùng để mở tệp mới hoặc tệp đã tồn tại để nhập xuất.
4. Hàm _open dùng để mở tệp đã tồn tại để nhập xuất.
5. Các close và _close dùng để đóng tệp.
6. Hàm chmod dùng để thay đổi thuộc tinh của tệp.
7. Hàm _chmode thay đổi thuộc tính tệp theo kiểu DOS.

8. Hàm perror dung để thông báo lỗi (stdio.h).

9. Hàm write dùng để ghi một dãy các byte.

10. Hàm read dùng để đọc một dãy các byte trên tệp.

11. Hàm lseek dùng để di chuyển con trỏ chỉ vị.

9.3. UNIX và DOS.

Do tính lịch sử, vào giai đoạn ban đầu, ngôn ngữ C được phát triển dưới hệ điều hành UNIX. Ngay nay C đã phát triển khá mạnh trên các hệ khác như DOS, OS/2,... Để duy trì tính kế thừa và khả năng tương thích của các chương trình viết bằng C trong hệ UNIX, Turbo C vẫn đưa ra các hàm nhập xuất tệp cấp 1 làm việc kiểu UNIX. Các đối trong các hàm open, creat, chmod có liên quan đến UNIX. Để tiện dùng trong DOS, Turbo C cũng đưa ra một số hàm tương tự như _open, _creat, _cdmod,... Sự khác nhau của chúng như sau:

1. Kiểu truy nhập mặc định của creat và open là văn bản, của _creat và _open là nhị phân.

2. Thuộc tính tệp trong các hàm creat và chmod là các hằng có đặc trưng UNIX định nghĩa trong <sys/stat.h>. Thuộc tính tệp trong các hàm _creat và _chmod là các hằng định nghĩa trong <dos.h>

§10. TẠO TỆP, ĐÓNG MỞ TỆP VÀ KIỂM TRA LỖI

10.1. Hàm creat: Xây dựng tệp mới.

+ Dạng hàm:

int creat(const char *fname, int amode);

+ Đối:

Đối thứ nhất là tên tệp,

Đối thứ hai là thuộc tính tệp. Các giá trị thường dùng của amode (định nghĩa trong sys/stat.h):

S_IREAD Tệp chỉ đọc. Sau này không thể xoá,sửa,bổ sung.

S_IWRITE Tệp để ghi. Sau này có thể xoá, sửa, bổ sung.

+ Công dụng: Xây dựng tệp mới có tên cho bởi đối thứ nhất và có thuộc tính cho bởi amode. Trong trường hợp tệp đã tồn tại:

- Nếu tệp để ghi, thì nó bị xoá,

- Nếu tệp chỉ đọc, thì bị lỗi.

Khi có lỗi, hàm trả về -1.

Khi thành công, hàm trả về số hiệu tệp (handle). Số hiệu này có thể dùng trong hàm write để ghi thông tin lên tệp (ngay cả khi amode = S_IREAD). Kiểu ghi được xác định bởi biến _fmode (định nghĩa trong fcntl.h và stdlib.h). Giá trị mặc định của _fmode là O_TEXT (kiểu văn bản). Muốn ghi theo kiểu nhị phân thì ta cần xác định lại _fmode (trước khi dùng hàm creat) bằng câu lệnh: _fmode = O_BINARY;

+ **Ví dụ về creat:** Chương trình sau dùng creat để tạo tệp mới và ghi lên đó 5 số nguyên. Tệp có thuộc tính chỉ đọc nên nó được bảo vệ. Sau này nếu muốn sửa, ghi bổ sung hay xoá ta cần thay đổi thuộc tính của nó bằng hàm chmod hay _chmod.

```
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
int a[5]={10,2,10,4,5};
main()
  {
    int fd;
    /* Tạo tệp mới sl.sl để ghi theo kiểu nhị phân */
    _fmode = O_BINARY;
    fd=creat("sl.sl",S_IREAD);
            /* Ghi 5 số nguyên lên tệp */
    write(fd,a,10);
    close(fd);    /* Đóng tệp */
  }
```

10.2. Hàm _creat: Xây dựng tệp mới.

+ **Dạng hàm:**

int _creat(const char *fname, int attrib);

+ **Đối:**

Đối thứ nhất là tên tệp,

Đối thứ hai là thuộc tính tệp. Các giá trị thường dùng của attrib (định nghĩa trong dos.h):

FA_RDONLY Tệp chỉ đọc, không thể xoá hoặc sửa chữa.

FA_ARCH Tệp để ghi.

FA_HIDDEN Tệp ẩn, không hiện trong lệnh DIR của DOS.

Chú ý: Giá trị của attrib có thê là một tổ hợp (cónghĩa) của các giá trị trên, ví dụ nếu attrib = FA_HIDDEN | FA_ARCH

thì thuộc tính tệp là: ẩn và có thể ghi.

+ Công dụng:

Xây dựng tệp mới có tên cho bởi đối thứ nhất và có thuộc tính cho bởi attrib. Trong trường hợp tệp đã tồn tại:

- Nếu tệp để ghi, thì nó bị xoá,
- Nếu tệp chỉ đọc, thì bị lỗi.

Khi có lỗi, hàm trả về -1.

Khi thành công, hàm trả về số hiệu tệp (handle). Số hiệu này có thể dùng trong hàm write để ghi thông tin lên tệp (ngay cả khi attrib = FA_RDONLY). Kiểu ghi luôn luôn là nhị phân.

+ Ví dụ về _creat: Chương trình sau dùng _creat để tạo tệp mới và ghi lên đó 5 số nguyên. Tệp có thuộc tính để ghi nên sau này có thể sửa, ghi bổ sung hay xoá.

```
#include <io.h>
#include <dos.h>
int a[5]={10,2,10,4,5};
main()
  {
    int fd;
          /* Tạo tệp mới sl.sl để ghi
             theo kiểu nhị phân */
    fd=_creat("sl.sl",FA_ARCH);
          /* Ghi 5 số nguyên lên tệp */
    write(fd,a,10);
    close(fd);    /* Đóng tệp */
  }
```

10.3. Hàm open:

Mở tệp đã có hoặc xây dựng tệp mới để đọc ghi

+ Dạng hàm:

int open(const char *fname, int access, [, unsigned amode]);

+ Đối:

- Đối thứ nhất fname xác định tên tệp cần mở,

- Đối thứ hai access xác định quyền và kiểu truy nhập, nó có thể nhận từ các giá trị sau (định nghĩa trong fcntl.h)

Giá trị	Ý nghĩa
O_RDONLY	Đọc
O_WRONLY	Ghi
O_RDWR	Đọc/ghi
O_APPEND	Ghi bổ sung
O_TRUNC	Xoá tệp nếu nó tồn tại (giữ ngyên thuộc tính)
O_TEXT	Kiểu nhập xuất văn bản
O_BINARY	Kiểu nhập xuất nhị phân
O_CREAT	Nếu tệp chưa có thì nó được tạo. (Khi đó cần đưa thêm đối thứ ba).

Chú ý 1: Đối thứ hai access có thể là một tổ hợp có nghĩa bất kỳ của các giá trị nêu trên (dùng phép |), ví dụ

O_WRONLY|O_TRUNC|O_BINARY

Chú ý 2: Nếu trong đối access không mô tả rõ O_TEXT hay O_BINARY thì kiểu truy nhập được xác định theo giá trị của biến _fmode (xem hàm creat)

- Đối thứ ba amode (xem hàm creat) dùng để xác định thuộc tính của tệp mới được tạo, nó thường nhận hai giá trị sau:

S_IREAD Tệp chỉ đọc, không thể xoá hoặc sửa chữa.

S_IWRITE Tệp có thể ghi.

+ Công dụng:

1. Không có O_CREAT: Mở một tệp đã tồn tại để truy nhập theo cách xác định bởi access. Lỗi xẩy ra khi:

- Tệp không tồn tại.
- Mở để ghi một tệp chỉ đoc.

2. Có O_CREAT: Nếu tệp chưa có thì nó được tạo. Lỗi xẩy ra khi:

- Mở để ghi một tệp chỉ đọc.

+ Giá trị hàm:

Khi thành công hàm trả về số hiệu tệp, nó sẽ dùng trong các hàm để truy nhập tới tệp. Khi có lỗi hàm cho giá trị -1.

+ **Ví dụ về open:**

Câu lệnh:

```
fd=open("sl.sl",O_WRONLY|O_BINARY|O_TRUNC|O_CREAT,
    S_IWRITE);
```

sẽ mở tệp sl.sl để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó bị xoá. Nếu tệp chưa có thì nó được xây dựng.

Câu lệnh:

```
fd=open("sl.sl",O_RDONLY|O_BINARY);
```

sẽ mở tệp đã tồn tại sl.sl để đọc theo kiểu nhị phân.

10.4. Hàm _open: Mở tệp đã tồn tại để đọc ghi.

+ **Dạng hàm:**

```
int open(const char *fname, int oflag);
```

+ **Đối:**

- Đối thứ nhất fname xác định tên tệp cần mở,
- Đối thứ hai oflags xác định quyền và kiểu truy nhập, nó có ý nghĩa như đối access trong hàm open, ngoại trừ 2 điểm sau: kiểu mặc định là O_BINARY (chứ không phải O_TEXT) và phương án O_CREAT không có tác dụng tạo tệp mới.

+ **Công dụng:** Mở một tệp đã tồn tại để truy nhập theo cách xác định bởi oflags. Lỗi xẩy ra khi:

- Tệp không tồn tại.
- Mở để ghi một tệp chỉ đọc.

+ **Giá trị hàm:**

Khi thành công hàm trả về số hiệu tệp, nó sẽ dùng trong các hàm để truy nhập tới tệp. Khi có lỗi hàm cho giá trị -1.

+ **Ví dụ về open:**

Câu lệnh:

```
fd=_open("sl.sl",O_WRONLY|O_TRUNC);
```

sẽ mở và xoá tệp sl.sl (đã tồn tại) để ghi lại từ đầu (theo kiểu nhị phân).

Câu lệnh:

```
fd=_open("sl.sl",O_RDONLY);
```

sẽ mở tệp đã tồn tại sl.sl để đọc theo kiểu nhị phân.

10.5. Các hàm close và close: đóng tệp.

+ Dạng hàm:

int close(int fd);

int close(int fd);

+ Đối:

fd là số hiệu tệp nhận được trong các hàm creat, _creat open và _open.

+ Công dụng: Đóng tệp có số hiệu fd. Số hiệu fd được giải phóng và nó có thể dùng cho tệp khác.

Khi thành công hàm trả về số 0. Khi có lỗi hàm cho -1.

10.6. Hàm chmod: thay đổi thuộc tính tệp.

+ Dạng hàm:

int chmod(const char *fname, int amode);

+ Đối:

Đối fname xác định tên tệp cần đổi thuộc tính.

Đối aodme cho thuộc tín mới của tệp. Các giá trị của amode ở đây giống như amode trong hàm creat:

S_IREAD Tệp chỉ đọc,

S_WRITE Tệp để ghi.

+ Công dụng: Đổi thuộc tính tệp fname theo amode.

Hàm cho giá trị 0 nếu thành công và -1 nếu có lỗi.

10.7. Hàm chmod: Nhận hay đổi thuộc tính tệp theo kiểu DOS.

+ Dạng hàm:

int _chmod(const char *fname, int func [,int attrib]);

+ Đối:

Đối fname xác định tên tệp cần nhận hay đổi thuộc tính.

Nếu func = 0, hàm cho biết thuộc tính của tệp, và không cần dùng đối thứ ba.

Nếu func = 1, thay đổi thuộc tính, dùng đối attrib để xác định thuộc tính mới. Các giá trị của attrib ở đây giống như attrib trong _creat:

FA_RDONLY Tệp chỉ đọc,

FA_ARCH Tệp để ghi.

FA_HIDDEN Tệp ẩn.

+ **Công dụng:** Cho biết thuộc tính của tệp fname (func=0) hoặc đổi thuộc tính tệp theo attrib (func=1).

Hàm cho giá trị 0 nếu thành công và -1 nếu có lỗi.

10.8.Hàm perror: thông báo lỗi hệ thống.

(Hàm này đã trình bầy trong §3.7)

+ **Dạng hàm:**

```
#include <stdio.h>
void perror(const char *s);
```

+ **Đối:**

s là con trỏ trỏ tới một chuỗi ký tự.

+ **Công dụng:** Hàm in chuỗi s và thông báo lỗi.

§11. CÁC HÀM WRITE, READ VÀ LSEEK

Để nhập xuất dữ liệu trên tệp, hệ thống cấp 1 cung cấp 3 hàm: write, read và lseek, chúng làm việc giống như các hàm fwrite, fread và fseek. Các hàm này nên dùng với kiểu nhập xuất nhị phân.

11.1. Hàm write: ghi một dãy các byte lên tệp.

+ **Dạng hàm:**

```
unsigned write(int fd, void *ptr, unsigned n);
```

+ **Đối:**

fd là số hiệu tệp,

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu,

n là số byte.

+ **Công dụng:** Ghi n byte từ vùng nhớ ptr lên tệp có số hiệu fd. Khi thành công hàm trả về một số bằng số byte ghi được.

Khi có lỗi hàm trả về -1.

+ **Nhận xét:** Khi giá trị trả về của hàm khác n là có lỗi.

11.2. Hàm read: đọc một dãy các byte từ tệp.

+ **Dạng hàm:**

```
unsigned read(int fd, void *ptr, unsigned n);
```

+ **Đối:**

fd là số hiệu tệp,

ptr là con trỏ trỏ tới vùng nhớ nhận dữ liệu đọc được,

n là số byte.

+ Công dụng:

Đọc n byte từ tệp có số hiệu fd chứa vào vùng nhớ ptr. Khi thành công hàm trả về một số bằng số byte đọc được. Khi có lỗi hàm trả về -1.

+ Nhận xét:

Khi gặp cuối tệp thì giá trị trả về của hàm có thể nhỏ hơn n.

11.3. Hàm lseek: di chuyển con trỏ chỉ vị.

+ Dạng hàm:

long lseek(int fd, long sb, int xp);

+ Đối:

fd là số hiệu tệp,

sb là số byte cần di chuyển,

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 chỉ vị.

xp = SEEK_END hay 2: Xuất phát từ cuối tệp.

+ Công dụng:

Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về một giá trị kiểu long bằng vị trí mới của con trỏ chỉ vị (số thứ tự của byte mà con trỏ chỉ vị trỏ tới). Khi có lỗi hàm trả về -1L.

+ Chú ý:

Không nên dùng lseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

+ Ví dụ:

Để xác định độ dài của tệp có số hiệu fd ta có thể dùng hàm lseek như sau:

```
long do_dai;
do_dai = lseek(fd,0,2);
```

Ví dụ 1: Chương trình sao tệp.

```
/*
Sao Tệp
Dùng các hàm _creat _open read write theo kiểu nhị phân
*/
        #include "dos.h"
        #include "io.h"
        #include "fcntl.h"
        #include "stdio.h"
        main()
          {
            int n;
            char t1[14],t2[14],c[1000];
            int fd1,fd2;
            printf("\n Tệp nguồn: ");
            gets(t1);
            printf("\n Tệp đích: ");
            gets(t2);
            fd1=_open(t1,O_RDONLY);
            if(fd1==-1)
              {
                printf("\n Tệp %s không tồn tại",t1);
                exit(1);
              }
            fd2=_creat(t2,FA_ARCH);
            if(fd2==-1)
              {
                printf("\n Loi tao tep %s ",t2);
                exit(1);
              }
                /* Thực hiện sao tệp */
            while ( (n=read(fd1,c,1000)) > 0)
                  write(fd2,c,n);
            close(fd1);
            close(fd2);
          }
```

Ví dụ 2: Chương trình thay đổi thuộc tính tệp.

```
/* Đổi thuộc tính, dùng hàm _chmod */
    #include <stdio.h>
    #include <conio.h>
    #include <io.h>
    #include <fcntl.h>
    main()
      {
        char tep[14];
        int x,attrib;
        printf("\nTep: ");
        gets(tep);
        printf("\nChọn thuộc tính: \
            0-Để ghi 1-Chỉ đọc (Cộng thêm 2 thành ẩn) ");
        x=getche() - '0';
        if(x==0)
            attrib=FA_ARCH;
        else
            attrib=FA_RDONLY;
        if (x>=2)
            attrib=attrib|FA_HIDDEN;
        if(_chmod(tep,1,attrib)==-1)
             perror("\nLỗi:");
        else
            printf("\nĐã thay đổi thuộc tính");
      }
```

Ví dụ 3: Xây dựng hệ quản trị dữ liệu như ví dụ 2 mục §8. Dùng các hàm creat open write read lseek (kiểu nhị phân).

```
/* Dùng các hàm cấp 1: creat open write, read, lseek
để tổ chức cơ sở dữ liệu */
    #include "conio.h"
    #include "stdio.h"
    #include "sys/stat.h"
    #include "fcntl.h"
    #include "io.h"
```

```
void taomoi(void);
void bosung(void);
void suachua(void);
void xem(void);
typedef struct
    {
      char ht[25];
      int bl;
      float nc;
    } person;
long size = sizeof(person);
main()
  {
     int sg;
     while (1)
      {
       clrscr();
         printf("\n1.Tao moi");
         printf("\n2.Bo sung");
         printf("\n3.Sua chua");
         printf("\n4.Xem du lieu");
         printf("\n5.Ket thuc");
         sg=getch();
         if(sg=='1')
           taomoi();
         else
          if (sg=='2')  bosung();
           else
            if (sg=='3')  suachua();
            else
             if (sg=='4')  xem();
          else
            {
             clrscr(); exit(1);
            }
       }
  }
```

```
void taomoi(void)
  {
    person cb;
    int fd,n=0;
    _fmode=O_BINARY;
    fd=creat("dscb.sl",S_IWRITE);
    if(fd==-1)
      {
      perror("\nLoi tao tep dscb.sl");
      exit(1);
      }
    while(1)
      {
      clrscr();
          printf("\nNguoi thu %d\n\n",n+1);
          printf("\nHo ten (Bam Enter ket thuc): ");
          gets(cb.ht);
          if(cb.ht[0]==0)
             break;
          printf("\nBac luong: ");
          scanf("%d",&cb.bl);
          printf("\nNgay cong: ");
          scanf("%f%*c",&cb.nc);
          ++n;
          write(fd,&cb,size);
    }
   close(fd);
  }
void bosung(void)
  {
    person cb;
    int fd; long n;
    fd=open("dscb.sl",O_BINARY|O_WRONLY|O_APPEND
            |O_CREAT, S_IWRITE);
    if(fd==-1)
```

```
{
  perror("\nLoi tao tep dscb.sl");
  exit(1);
}
/* Xac dinh n = so ban ghi */
n=lseek(fd,0,2)/size;
while(1)
  {
    clrscr();
    printf("\nNguoi thu %ld\n\n",n+1);
    printf("\nHo ten (Bam Enter ket thuc): ");
    gets(cb.ht);
    if(cb.ht[0]==0)
      break;
    printf("\nBac luong: ");
    scanf("%d",&cb.bl);
    printf("\nNgay cong: ");
    scanf("%f%*c",&cb.nc);
    ++n;
    write(fd,&cb,size);
  }
close(fd);
}
void suachua(void)
{
  person cb;
  int fd; long i,n;
  fd=open("dscb.sl",O_BINARY|O_RDWR);
  if( fd==-1)
    {
      printf("\nChua co du lieu");
      return;
    }
  /* Xac dinh n = so ban ghi */
  n=lseek(fd,0,2)/size;
```

```
while(1)
 {
   clrscr();
   printf("\nSua nguoi thu (n<=%ld)",n);
   scanf("%ld%*c",&i);
   if(i>0&&i<=n)
     {
      lseek(fd,(i-1)*size,SEEK_SET);
      read(fd,&cb,size);
      printf("\nSo lieu hien tai:");
      printf("\nHo ten: %s",cb.ht);
      printf("\nBac luong: %d",cb.bl);
      printf("\nNgay cong: %0.2f",cb.nc);
      printf("\n\nSua la:");
      printf("\nHo ten (Bam Enter ket thuc): ");
      gets(cb.ht);
      if(cb.ht[0]==0) break;
      printf("\nBac luong: ");
      scanf("%d",&cb.bl);
      printf("\nNgay cong: ");
      scanf("%f%*c",&cb.nc);
      printf("\nCo Ghi sua ? - C/K");
      if(toupper(getch())=='C')
        {
          lseek(fd,m,SEEK_SET);
          write(fd,&cb,size);
        }
     }
 }
 close(fd);
}
void xem(void)
{
  person cb;
  int fd,i=0;
```

```
    fd=open("dscb.sl",O_BINARY|O_RDONLY);
    if( fd==-1)
      {
        printf("\nChua co du lieu");
        return;
      }
    clrscr();
    while( read(fd,&cb,size)>0 )
      {
       ++i;
        printf("\n\nNguoi %d",i);
        printf("\nHo ten: %s",cb.ht);
        printf("\nBac luong: %d",cb.bl);
        printf("\nNgay cong: %0.2f",cb.nc);
      }
    close(fd);
    getch();
}
```

BÀI TẬP CHƯƠNG 10

Bài 1. Lập chương trình xem và sửa nội dung của một tệp bất kỳ (tên tệp nhập vào từ bàn phím). Chương trình cho hiện lên màn hình từng trang 200 ký tự của tệp. Mỗi ký tự sẽ hiện mã ASCII và dạng ký tự nếu có. Có thể dùng các phím mũi tên để di chuyển đến mã cần sửa. Bấm Enter để xem/sửa trang tiếp theo. Bấm ESC để kết thúc chương trình.

Bài 2. Cho biết thông tin cố định về lương của mỗi cán bộ trong đơn vị X gồm:

- Họ tên
- Số thứ tự
- Bậc lương

Số thứ tự được đánh số liên tiếp từ 1,2,...

Yêu cầu:

a) Xây dựng một tệp (gọi là tệp LUONG) để lưu trữ thông tin cố định về lương của đơn vị X.

b) Sử dụng tệp LUONG để tính lương cho một người khi biết số thứ tự và số ngày công của người đó. Hai số liệu này nhập vào từ bàn phím. Lương tính theo công thức:

lương = (bậc lương * ngày công)/26

Bài 3. Sao chép các dữ liệu của tệp LUONG (trong bài 2) sang một tệp mới (gọi là tệp L_MOI) theo thứ tự giảm dần của bậc lương, in nội dung tệp L_MOI ra màn hình.

Bài 4. Lập chương trình thực hiện các việc sau:

+ Xây dựng tệp MA_TRAN và ghi lên đó các phần tử của hai ma trận vuông cùng cấp A và B.

+ Dùng tệp MA_TRAN để tính ma trận tích C = A*B, sau đó ghi bổ sung ma trân C lên tệp MA_TRAN.

+ Đọc các ma trận A, B và C từ tệp MA_TRAN vào bộ nhớ rồi in chúng ra màn hình.

Bài 5. Xét mảng cấu trúc

```
struct
    {
      char ht[25];      /* họ tên */
      char qq[30];      /* quê quán */
      int tuoi;
    } person[100];
```

Mỗi phần tử của mảng cấu trúc person chứa thông tin về một người. Hãy lập chương trình thực hiện các yêu cầu sau:

+ Vào số liệu của n người và ghi lên tệp NHAN_SU

+ Đọc thông tin từ tệp NHAN_SU và chứa vào mảng person

+ Ghi danh sách nhân viên từ mảng person vào tệp NS_MOI theo thứ tự tăng của tuổi.

CHƯƠNG 11

LƯU TRỮ DỮ LIỆU VÀ TỔ CHỨC BỘ NHỚ CHƯƠNG TRÌNH

Như đã biết trước khi sử dụng một đối tượng dữ liệu (biến, mảng, cấu trúc, union,...), thì cần phải khai báo nó.

Vị trí khai báo (bên trong hoặc bên ngoài hàm) quyết định hai vấn đề then chốt sau đây:

+ Đối tượng dữ liệu được lưu giữ trong phần cố định (vùng cấp phát tĩnh) hay phần tạm thời (vùng ngăn xếp) của bộ nhớ chương trình.

+ Phạm vi sử dụng của đối tượng dữ liệu là cục bộ (bên trong một khối, một hàm) hay toàn bộ (từ vị trí khai báo đến cuối tệp tin của chương trình).

Chương này sẽ trình bầy các từ khoá đặt trước các khai báo cho một đối tượng để làm rõ thêm cách thức lưu trữ và khả năng truy nhập của đối tượng. Đó là các từ khoá: auto, extern, static, register, const và volatile. Ngoài ra cũng sẽ đề cập cách tổ chức bộ nhớ cho chương trình.

Để chon gọn khi trình bầy ta sẽ chỉ nói đến biến. Nếu không giải thích gì thêm thì điều đó cũng đúng cho mọi đối tượng dữ liệu.

§1. BỘ NHỚ CHƯƠNG TRÌNH

Bộ nhớ chương trình được chia lam 4 phần là (theo thứ tự từ địa chỉ thấp đến địa chỉ cao):

+ Vùng mã lệnh (chứa mã lệnh và hằng).

+ Vùng cấp phát tĩnh (chứa các đối tượng ngoài và tĩnh).

+ Vùng cấp phát động (heap).

+ Vùng ngăn xếp (chứa các đối tượng cục bộ).

Hai vùng đầu có độ lớn cố định trong suốt thời gian chương trình làm việc.

Vùng cấp phát động lưu trữ các đối tượng (biến, cấu trúc, mảng,...) được cấp phát bằng hàm malloc. Vì vậy mỗi khi thực hiện hàm này, thì vùng heap nở ra. Ngược lại khi thực hiện hàm free (để giải phóng một vùng nhớ trên heap), thì không gian sử dụng trên heap bị thu lại.

Vùng ngăn xếp dùng để lưu trữ các đối tượng cục bộ. Mỗi khi chương trình thực hiện một khối lệnh hoặc một hàm, thì các đối tượng khai báo trong khối hoặc hàm đó được lưu trữ trên ngăn xếp và ngăn xếp phình ra. Khi chương trình ra khỏi một hàm hay một khối lệnh, thì các đối tượng cục bộ của hàm hay khối đó bị lấy ra khỏi ngăn xếp, và ngăn xếp bị thu nhỏ lại.

Như vậy độ lớn sử dụng của vùng heap và ngăn xếp luôn thay đổi trong thời gian chương trình làm việc.

Hình vẽ dưới đây minh hoạ tổ chức bộ nhớ của chương trình:

+ Các vùng mã lệnh và cấp phát tĩnh có độ lớn cố định.

+ Giữa ngăn xếp và heap là một vùng tự do. Cả ngăn xếp và haep đều có quyền dùng vùng tự do này khi cần thiết. Ví dụ khi cần lưu trữ 3 biến cục bộ (kiểu int), thì ngăn xếp sẽ được mở rộng 6 byte (lấy từ vùng tự do). Khi giải phóng 3 biến nói trên thì ngăn xếp trả lại 6 byte cho vùng tự do. Ngăn xếp bị thu nhỏ lại, còn vùng tự do thì rộng ra. Cách phát triển và thu nhỏ vùng haep cũng tương tự như vậy.

Địa chỉ cao

Vùng ngăn xếp, phát triển xuống
Vùng nhớ tự do còn chưa được dùng
Vùng heap phát triển lên
Vùng cấp phát tĩnh
Vùng mã lệnh

Địa chỉ thấp

§2. TỪ KHOÁ AUTO

Từ khoá auto có đặt trước một khai báo biến bên trong hàm để chỉ rõ tính cục bộ của các đối tượng được khai báo. Vì các biến khai báo trong một hàm đương nhiên là cục bộ, nên từ khoá này là không cần thiết và rất ít được dùng.

Ví dụ:

```
#include <stdio.h>
main()
  {
    auto int i=10;
    printf("\n i= %d",i);
  }
```

§3. BIẾN NGOÀI VÀ TỪ KHOÁ EXTERN

3.1.Biến cục bộ. Biến khai báo bên trong một hàm hoặc một khối là biến cục bộ. Nó được lưu trữ trong ngăn xếp khi chương trình xét đến hàm hoặc khối chứa nó. Phạm vi sử dụng của biến cục bộ là bên trong hàm hoặc khối mà nó khai báo. Khi chưa được khởi đầu thì giá trị của biến cục bộ là chưa xác định.

3.2.Biến ngoài. Biến khai báo bên ngoài các hàm gọi là biến ngoài hay biến toàn bộ. Nó có một địa chỉ cố định (tại vùng cấp phát tĩnh) trong suốt thời gian chương trình làm việc. Khi chưa được khởi đầu thì giá trị của toàn bộ bằng không. Phạm vi sử dụng của biến ngoài là từ vị trí khai báo của nó cho đến cuối tệp tin chương trình. Nó cũng có thể sử dụng cho các tệp khác (khi chương trình gồm nhiều tệp) nhờ khai báo extern viết trên các tệp này.

Ví dụ sau minh hoạ phạm vi sử dụng của biến ngoài đối với chương trình trên một tệp tin. Phạm vi sử dụng của biến ngoài dd là trong các hàm in_1 và in_2, vì vậy chương trình làm việc tốt. Nếu sử dụng trực tiếp dd trong hàm main() thì chương trình sẽ báo lỗi.

```
#include <stdio.h>
void in_1(void);
void in_2(void);
main()
  {
   in_1(); in_2();
  }
  double dd=89.55;
  void in_1(void)
    {
     printf("\ndd= %0.2f",dd);
    }
  void in_2(void)
    {
     printf("\n2*dd= %0.2f",2*dd);
    }
```

3.3.Từ khoá extern. Từ khoá extern sử dụng khi một chương trình viết trên nhiều tệp và các tệp này được dịch độc lập sau đó mới liên kết với nhau để tạo thành tệp chương trình khả thi (đuôi exe). Có hai cách làm như vậy. Cách thứ nhất là dùng menu Project trong môi trường phát triển kết hợp của

TURBO C. Cách thứ hai là dùng chương trình TCC.EXE từ DOS.

Ta minh hoạ vấn đề nêu trên qua ví dụ sau. Giả sử chương trình được viết trên hai tệp là tep1.c và tep2.c. Trên tep1.c ta khai báo biến toàn bộ b_chung (kiểu int) và ta muốn sử dụng nó trong tep2.c. Làm sao để dịch và liên kết hai tệp nói trên thành một chương trình khả thi. Ta thử đưa ra vài phương án và xem chúng làm việc ra sao.

Phương án 1: Dịch đồng thời bằng cách dùng #include. Cách thức làm như sau:

+ Thêm vào đầu tệp tep2.c dòng lệnh:

```
#include "tep1.c"
```

+ Dịch tep2.c theo cách thông thường (ví dụ dùng chức năng Make Exe File của menu Compile).

Đây là cách đơn giản mà ta thường dùng. Đáng tiếc là theo cách này thì các tệp tep1.c và tep2.c được dịch một cách đồng thời như thể dịch một tệp nguồn ghép 2 tệp này với nhau. Cách làm này ngoài tốn thời gian phải dịch lại cả hai tệp nguồn, còn có nhược điểm rất cơ bản là không thể dịch được nếu tep1.c và tep2.c có kích thước lớn. Trong thực tế chúng tôi đã gặp nhiều trường hợp như vậy.

Phương án 2: Dịch độc lập. Đầu tiên ta dịch tep1.c để tao ra tệp đích tep1.obj. Điều này sẽ diễn ra thuận lợi. Tiếp đó ta dịch tep2.c để tạo tep2.obj. Chương trình dịch sẽ báo lỗi vì trong tệp này có sử dụng biến b_chung nhưng lại chưa khai báo. Nếu ta bổ sung vào tep2.c khai báo:

```
int b_chung;
```

thì việc dịch chót lọt và ta nhận được hai mô đun chương trình: tep1.obj và tep2.obj.

Bước tiếp theo là liên kết hai mô đun trên để tạo thành chương trình khả thi. Bước này không ổn, vì chương trình dịch sẽ gặp hai biến b_chung và nó sẽ báo lỗi:

```
Linker Error: b_chung defined in module tep1.c
              is duplicated in module tep2.c
```

Đây chính là lúc cần dùng từ khoá extern. Để tep2.c có thể sử dụng biến b_chung đã khai báo ở tep1.c (khai báo ngoài) ta chỉ cần thêm vào tep2.c một khai báo biến b_chung như đã làm ở tep1.c nhưng thêm vào từ khoá extern như sau:

```
extern int b_chung;
```

Khai báo extern nói trên báo cho chương trình dịch biết biến b_chung đã được định nghĩa ở đâu đó, do đó không cần cấp phát bộ nhớ cho nó một lần nữa. Việc dịch tep2.c sang tệp tep2.obj sẽ tiến triển tốt. Ngoài ra khi liên kết các mô đun tep1.obj và tep2.obj sẽ mắc lỗi khai báo hai lần biến b_chung và như vậy mọi việc đều ổn thoả.

Dưới đây là một ví du đơn giản về cách dùng extern.

Chương trình của ta gồm hai tệp với nội dung sau:

```
/* tệp b1.c */
        int a[5]={1,2,3,8,5};
        float x=10.5;
        int imax(int x,int y)
          {
           return (x>y?x:y);
          }
```

Tệp này chứa:

+ Mảng ngoài a đã có giá trị khởi đầu.

+ Biến ngoài x = 10.5.

+ Hàm imax cho max của hai số nguyên.

Tệp thứ hai (b2.c) sẽ dùng các biến, mảng và hàm của tệp thứ nhất, nội dung của nó như sau:

```
/* Tệp b2.c */
        #include <stdio.h>
        extern int imax(int x,int y);
        extern int a[5];
        void in_x(void);
        void in_xx(void);
        main()
          {
            int i,max; max= a[0];
            for(i=1;i<5;++i)
                max=imax(max,a[i]);
            printf("\nmax= %d",max);
            in_x();
            in_xx();
              {
                extern float x;
              }
             printf("\nx= %0.2f",x);
          }
        void in_x(void)
          {
            printf("\nx= %0.2f",x);
          }
```

```
void in_xx(void)
   {
      printf("\n2*x= %0.2f",2*x);
   }
```

3.4. Vị trí khai báo và phạm vi sử dụng của biến extern.

+ Vị trí khai báo extern:

- Có thể ngoài các hàm.
- Có thể bên trong một hàm hoặc một khối lệnh.

+ Phạm vi sử dụng của biến extern: Dù khai báo ở đâu thì phạm vi sử dụng của biến extern (trên tệp chứa khai báo này) được tính từ vị trí khai báo cho đến cuối tệp. Rõ ràng đây là điều khác thường so với nhưng gì đã nói về phạm vi sử dụng của một biến.

Vận dụng nguyên lý này cho tệp b2.c ta nhận thấy:

+ Mảng a có thể sử dụng trên toàn tệp b2.c.

+ Biến x có thể xuất hiện (sử dụng trực tiếp) trong các hàm in_x, in_xx và phần hàm main phía dưới khối chứa khai báo extern biến x. Nếu trên khối này ta dùng biến x thì chương trình dịch sẽ báo lỗi.

§4. TỪ KHOÁ STATIC

4.1.Công dụng. Từ khoá static dùng để khai báo các biến tĩnh trong (cục bộ) và các biến tĩnh ngoài (toàn bộ). Ví dụ:

```
#include "stdio.h"
static int size;        /* Biến tĩnh ngoài */
static float aa[100];  /* Mảng tĩnh ngoài */
void dem(void)
   {
      int static dd=1;  /* Biến tĩnh trong */
      ...
   }
```

4.2. Biến tĩnh trong.

+ Bộ nhớ. Biến tĩnh trong được lưu trữ trong vùng cấp phát tĩnh, nó tồn tại trong suốt thời gian chương trình làm việc. Về điểm này nó giống như biến ngoài.

+ Giá trị khởi đầu. Biến tĩnh trong được khởi đầu một lần khi dịch theo giá trị ghi trong khai báo dạng:

```
static int so_thu_tu = 1;
```

Nếu không cho giá trị khởi đầu thì giá trị khởi đầu mặc định của biến tĩnh trong bằng không. Về điểm này biến tĩnh trong cũng giống như biến ngoài.

+ **Phạm vi sử dụng** của biến tĩnh trong chỉ giới hạn bên trong hàm mà nó khai báo. Tuy nhiên giá trị của nó vẫn được giữ khi ra khỏi hàm và giá trị này có thể sử dụng mỗi khi hàm được thực hiên trở lại.

+ **Ví dụ:** Ta lập hàm đánh số trang. Hàm có hai câu lệnh: in ra số hiệu trang hiện tại và tăng số hiệu trang lên một đơn vị để chuẩn bị cho lần in sau. Trong ngữ cảnh này, hay hơn cả là dùng một biến tĩnh để biểu thị số hiệu trang.

Chương trình có thể viết như sau:

```
#include "stdio.h"
void danh_so_trang(void);
void danh_so_trang(void)
   {
      int so_hieu_trang = 1;
      printf("\n Trang: %d",so_hieu_trang);  so_hieu_trang ++;
   }
main()
   {
      int i;
      for(i=0;i<10;++i)  danh_so_trang();
   }
```

4.3. Biến tĩnh ngoài.

+ **Bộ nhớ.** Biến tĩnh ngoài được lưu trữ trong vùng cấp phát tĩnh, nó tồn tại trong suốt thời gian chương trình làm việc. Về điểm này nó giống như biến ngoài và biến tĩnh trong.

+ **Giá trị khởi đầu.** Biến tĩnh ngoài được khởi đầu một lần khi dịch theo giá trị ghi trong khai báo dạng:

```
static int so_thu_tu = 1;
```

Nếu không cho giá trị khởi đầu thì giá trị khởi đầu mặc định của biến tĩnh ngoài bằng không. Về điểm này biến tĩnh ngoài cũng giống như biến ngoài và biến tĩnh trong.

+ **Phạm vi sử dụng** của biến tĩnh ngoài được tính từ khi khai báo cho đến cuối tệp chương trình. Phạm vi sử dụng của nó không thể mở rộng sang các tệp khác bằng khai báo extern như đối với biến ngoài. Thậm chí ta có thể dùng các biến trùng tên với biến tĩnh ngoài trên các tệp khác như ví dụ dưới đây. Ta cũng nhận thấy rằng khi chương trình viết trên một tệp thì biến ngoài và biến tĩnh ngoài có tác dụng như nhau.

+ Ví dụ: Ta tách chương trình trong 4.2 thành 2 tệp là t1.c và t2.c. Tệp t1.c có nội dung sau:

```
/* Tệp t1.c */
        #include "stdio.h"
        static int so_hieu_trang = 1;
        void danh_so_trang(void);
        void danh_so_trang(void)
          {
            printf("\nTrang: %d",so_hieu_trang);  so_hieu_trang++;
          }
```

Hàm danh_so_trang dùng biến tĩnh ngoài so_hieu_trang để tăng số hiệu trang lên một đơn vị sau mỗi lần in.

Nội dung tệp t2.c như sau:

```
        int so_hieu_trang;
        main()
          {
             for(so_hieu_trang=0; so_hieu_trang<5;++so_hieu_trang )
             danh_so_trang();  getch();
          }
```

Trong tệp này ta cố ý dùng biến ngoài so_hieu_trang trùng tên với biến tĩnh ngoài của t1.c, nhưng bộ dịch vẫn phân biệt được đó là hai biến khác nhau và chương trình vẫn hoạt động tốt.

§5. TỪ KHOÁ register

Từ khoá register có thể dùng để khai báo các biến cục bộ và các đối kiểu int hoặc char theo mẫu:

```
      register int i,j;
      register char ch;
```

Các biến register được lưu trữ trong các thanh ghi SI hoặc DI nếu có thể. Khi các thanh ghi này bận thì các biến register được lưu trữ vào ngăn xếp như các biến cục bộ thông thường.

Biến register thường được sử dụng làm biến điều khiển để tăng tốc độ thực hiện của các vòng lặp. Ví dụ trong hàm tính xn ta đưa vào đối register n như sau:

```
        double pow(double x, register int n)
          {  double s=1.0;
             for(; n; --n)  s *= x; return s;
          }
```

Chú ý: Ta có thể khai báo nhiều biến register nhưng không quá 2 trong chúng được lưu trữ trong các thanh ghi. Chương trình dưới đây khai báo đến 4 biến register, nó vẫn hoạt động bình thường.

```
#include "stdio.h"
main()
  { register int i,j,k,t;   t=0;
    for(i=1;i<=5;++i)
    for(j=1;j<=5;++j)
    for(k=1;k<=5;++k)  t++;
    printf("\ni= %d j= %d k= %d Tong= %d",i,j,k,t);
  }
```

§6. TỪ KHOÁ const

6.1. Công dụng.

Từ khoá const có hai tác dụng:

+ Dùng để khai báo và khởi đầu giá trị cho các biến trong và biến ngoài mà sau này giá trị của nó không cho phép thay đổi bởi các lệnh trong chương trình. Ta gọi chúng là các đối tượng hằng.

+ Dùng để khai báo các đối con trỏ mà trong thân hàm ta không được phép làm thay đổi giá trị các đối tượng được trỏ tới bởi các đối này.

Các điều nói trên được minh hoạ qua hai ví dụ sau.

Ví dụ 1. Khi biên dịch chương trình:

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

ta nhận được thông báo lỗi sau:

cannot modify a const object in function main

Ví dụ 2. Khi biên dịch hàm:

```
void tang_stt(const int *stt)
  {
    *stt = *stt + 1;        /* Sai */
  }
```

ta nhận được thông báo lỗi:

cannot modify a const object in function tang_stt

6.2. Cách dùng.

1. Từ khoá const thường dùng để tạo ra các hằng có kiểu sử dụng trong chương trình.

2. Có thể dùng từ khoá const trong các hàm để khai báo các đối mảng mà giá trị của nó không bị thay đổi trong thân hàm. Ví dụ xet hàm tính tích của 2 ma trận vuông a và b cấp n. Để a và b không bị thay đổi trong thân hàm ta cần khai báo chúng là các đối mảng const như sau:

```
void nhan_mt(const int a[20][20], const int b[20][20], int c[20][20], int n)
        {
          int i,j,k;
          for(i=1;i<=n;++i)
           for(j=1;j<=n;++j)
           {  c[i][j]=0;
              for(k=1; k<=n;++k)
                 c[i][j] += a[i][k]*b[k][j];
           }
        }
```

§7. TỪ KHOÁ volatile

Khi khai

```
volatile int t_time;
```

dùng để báo cho Turbo C biết giá trị của biến nguyên t_time có thể bị thay đổi theo một cách nào đó không được mô tả rõ trong chương trình, chẳng hạn t_time có thể thay đổi bởi các ngắt của DOS hay ROM BIOS. Biến volatile thường là các biến toàn bộ dùng trong các thủ tục ngắt. Trong chương 15 có đưa ra các vị dụ về việc sử dụng kiểu biến này.

BÀI TẬP CHƯƠNG 11

Bài 1. Trình bầy phương án tổ chức chương trình trên nhiều tệp bằng cách dùng từ khoá extern.

Bài 2. Nêu cách dùng các đối tượng dữ liệu sau: ngoài, tĩnh ngoài và tĩnh trong.

Bài 3. Khi nào thì dùng khai báo tĩnh đối với hàm.

Bài 4. Khi nào thì dùng khai báo const cho đối của hàm.

CHƯƠNG 12

CÁC CHỈ THỊ TIỀN XỬ LÝ

Các chỉ thị tiền xử lý không phải là các lệnh của ngôn ngữ C mà là các lệnh giúp cho việc soạn thảo chương trình nguồn C trước khi biên dịch. Có thể nói về vai trò của các chỉ thị tiền xử lý như sau: Khi dịch một chương trinh C thì không phải chính bản chương trình nguồn mà ta soạn thảo được dịch. Trước khi dịch, các lệnh tiền xử lý sẽ chỉnh lý bản gốc, sau đó bản chỉnh lý này sẽ được dịch. Có ba cách chỉnh lý được dùng là:

+ Phép thay thế (lệnh #define)

+ Phép chèn tệp (lệnh #include)

+ Phép lựa chọn các dòng lệnh để biên dịch (các lệnh #if)

Các chỉ thị tiền xử lý giúp chúng ta viết chương trình ngắn gọn hơn và tổ chức biên dịch, gỡ rối chương trinh linh hoạt, hiệu quả hơn.

§1. CHỈ THỊ #define ĐƠN GIẢN

Chỉ thị #define cho phép tạo ra các macro thay thế đơn giản và các macro thay thế theo đối (như thể các hàm). Trong mục này bàn đến macro đơn giản.

1.1. Cách viết. Chỉ thị:

#define tên dãy_ký_tự

có tác dụng thay thế tên bằng dãy_ký_tự đứng sau nó. Thông thường văn bản thay thế là phần còn lại của dòng. Một định nghĩa dài có thể được tiếp tục ở dòng sau bằng cách đặt dấu \ vào cuối dòng trước. Phạm vi của tên được định nghĩa bởi #define là từ lúc nó được định nghĩa cho tới cuối tệp gốc. Có thể định nghĩa lại một tên (đã định nghĩa ở trên) và từ thời điểm này trở đi nó có ý nghĩa mới. Trước khi định nghĩa lại một tên, ta cần giải phóng nó bằng chỉ thị:

#undef tên

Nếu thiếu thủ tục giải phóng trước khi định nghĩa lại, chúng ta sẽ nhận được lời cảnh báo (warning) khi dịch chương trình. Một tên đã định nghĩa có thể sử dụng trong các định nghĩa sau đó.

Một điều cần chú ý là: Phép thế không thực hiện cho các hằng chuỗi ký tự (được bao bởi hai dấu nháy kép). Chẳng hạn nếu ALPHA là một tên đã được định nghĩa (bởi #define), thì sẽ không có việc thay thế nào trong câu lệnh:

```
printf("\n ALPHA ");
```

Để minh hoạ các điều nói trên, ta xét chương trình:

```
#include "stdio.h"
#define in printf
#define N 100
main()
  {
    int M = 200;
    in("\nN=%d M=%d",N,M);
    #define M 300
    in("\nN=%d M=%d",N,M);
    #undef M
    #define M (N + 400)
    in("\nN=%d M=%d",N,M);
  }
```

Trong suốt chương trình trên:

+ in luôn luôn được thay bằng printf

+ N (ở ngoài các dấu nháy kép) luôn luôn được thay bằng 100. Còn M thì được xử lý khác nhau. Trong lệnh in thứ nhất, M được xem là biến nguyên có giá trị 100. Trong lệnh in thứ hai, M được thay bằng 300, còn trong lệnh in thứ ba, nó được thay bằng biểu thức (N + 400). Cũng cần chú ý là không có việc thay thế đối với các N và M trong các dấu nháy kép.

Chương trình trên sau khi dịch và chạy cho 3 dòng kết quả sau:

```
N=100 M=200
N=100 M=300
N=100 M=500
```

Có thể dùng #define để định nghĩa một tên hàm (như hàm printf trong ví dụ trên), một biểu thức, một đoạn chương trình bằng một tên. Bằng cách đó chương trình nguồn được viết ngắn gọn và dễ hiểu hơn. Để tránh xẩy ra những sai sót đáng tiếc, chúng ta cần thực hiện những lưu ý sau khi dùng #define để định nghĩa các chuỗi dài.

Chú ý 1.1. Khi định nghĩa một biểu thức ta nên đặt nó trong các dấu ngoặc tròn. Chẳng hạn nếu viết:

```
#define length 3 + 5
```

thì định nghĩa này làm việc tốt cho các câu lệnh đơn giản như:

```
size = length;
```

nhưng lại hỏng trong câu lệnh:

```
double_size = 2*length;
```

Nếu đặt biểu thức 3 + 5 trong các dấu ngoặc tròn, tức là nếu dùng định nghĩa:

```
#define length (3 + 5)
```

thì ta có thể dùng length ở bất kỳ chỗ nào cũng vẫn đúng.

Chú ý 1.2. Khi định nghĩa một đoạn chương trình gồm nhiều câu lệnh, ta nên viết dưới dạng khối lệnh. Hay nói cách khác: Đặt đoạn chương trình trong các dấu { và }

Ví dụ nếu định nghĩa:

```
#define KT { printf("\nKet thuc"); exit(1); }
```

rồi sau đó dùng KT để kiểm tra giá trị của biến x theo mẫu:

```
if (x > MAX_VALUE) KT
```

thì chương trình chỉ kết thúc khi giá trị của x vượt ra ngoài phạm vi cho phép. Tuy nhiên nếu bỏ các dấu { và } trong định nghĩa trên thì câu lệnh if sẽ trở thành:

```
if (x > MAX_VALUE) printf("\nKet thuc"); exit(1);
```

và do đó chương trình sẽ luôn luôn kết thúc ngay cả khi giá trị của biến x trong khoảng cho phép.

Một tác dụng nữa của #define là có thể đưa vào C các phong cách viết mới. Chẳng hạn ta có thể định nghĩa các từ khoá và các hàm chuẩn bằng các tên mới theo nghĩa tiếng Việt Bằng cách đó ta có thể Việt Nam hoá ngôn ngữ C và dễ dàng phổ biến C cho người chưa biết tiếng Anh. Những người yêu thích PASCAL có thể đưa vào các định nghĩa:

```
#define { begin
#define } end
```

rồi sau đó dùng begin và end để tổ chức khối lệnh.

§2. CHỈ THỊ #define CÓ ĐỐI

Có thể dùng #define để định nghĩa các macro có đối tương tự như các hàm. Khi đó văn bản thay thế sẽ phụ thuộc vào cách gọi tới macro. Chẳng hạn ta có thể định nghĩa một macro để tính max của hai giá trị như sau:

```
#define max(A,B) (A)>(B)?(A):(B)
```

Khi đó dòng:

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

được thay bằng câu lệnh:

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

Dòng:

```
y = max(x*y,1+y+x);
```

được thay bằng câu lệnh:

```
y = (x*y)>(1+y+x)?(x*y):(1+y+x);
```

Như vậy nhờ các macro có đối ta có thể xây dựng được các hàm đơn giản để sử dụng trong chương trình. Đối của macro không ứng với một kiểu (dữ liệu) nhất định, vì vậy macro max(A,B) định nghĩa ở trên có thể dùng cho cả giá trị nguyên cũng như thực.

Khi dùng macro có đối cần quán triệt hai chú ý sau.

Chú ý 2.1. Giữa tên macro (tên hàm) và dấu ngoặc mở không được đặt các khoảng trống. Chẳng hạn định nghĩa sau là hợp lệ:

```
#define max( A , B ) (A)>(B)?(A):(B)
```

Nhưng định nghĩa dưới đây là sai:

```
#define max (A,B) (A)>(B)?(A):(B)
```

vì có khoảng trống giữa dấu (và tên macro max.

Chú ý 2.2. Khi viết biểu thức thay thế, các đối hình thức (như A và B trong ví dụ trên) cần được bao quanh bởi các dấu ngoặc đơn. Để minh hoạ điều này, ta thử bỏ các dấu ngoặc đơn trong định nghĩa macro tich(a,b) (tích của a và b) và khảo sát hiệu quả của chương trình sau:

```
#include "stdio.h"
#define tich(a,b) a*b
main()
  {
    int x=10;
    int y=20,z=5;
    printf("\nTich= %d",tich(x+y,z));
  }
```

Khi biên dịch chương trình này, trình biên dịch sẽ thay

```
tich(x+y,z)
```

bằng:

```
x+y*z
```

Như vậy ta sẽ không nhận được tích (x+y)*z như ta mong muốn.

§3. CHỈ THỊ BAO HÀM TỆP #include

Lệnh #include chỉ thị cho bộ tiền xử lý nhận nội dung của tệp khác và đặt chèn vào tệp chương trình nguồn đang xét.

Cách viết. Chỉ thị này có thể viết theo các cách sau:

Cách 1.

#include <[đường_dẫn]tên_tệp>

Cách 2.

#include "[đường_dẫn]tên_tệp"

Các ví dụ về cách viết.

Ví dụ 3.1: (cách 1, có cả đường dẫn)

#include <a:\tv\cac_ham.c>

Ví dụ 3.2: (cách 2, có cả đường dẫn)

#include "a:\tv\cac_ham.c"

Ví dụ 3.3: (cách 1, không có đường dẫn, chỉ có tên tệp)

#include <vao_sl.c>

Ví dụ 3.4: (cách 2, không có đường dẫn, chỉ có tên tệp)

#include "vao_sl.c"

Tác dụng. Trước khi dịch, Chương trình dịch sẽ tìm tệp theo tên và đường dẫn ghi trong lệnh #include. Nếu tìm thấy thì nội dung của tệp này được gọi ra và chèn vào tệp nguồn đang xét tại đúng vị trí của #include. Nếu không tìm thấy, thì chương trình dịch sẽ thông báo lỗi:

Unable to open include file '[đường_dẫn]tên_tệp'

Cách thức làm việc của hai dạng #include chỉ khác nhau khi không có thông tin về đương dẫn. Khi đó #include dạng 1 sẽ tìm tệp trong thư muc INCLUDE của TURBO C. Còn #include dang 2 thì trước tiên tìm tệp trong thư mục chủ, nếu không thấy thì tiếp tục tìm trong thư muc INCLUDE. Như vậy các ví dụ 3.1 và 3.2 đều có tác dụng như nhau là tìm tệp cac_ham.c trong thư muc tv trên ổ A. Trong ví dụ 3.3 tệp vao_sl.c chỉ được tìm trong thư muc INCLUDE. Trong ví dụ 3.4 tệp vao_sl.c được lần lượt tìm trong thư mục chủ và thư mục INCLUDE.

Các #include lồng nhau. C cho phép các #include lồng nhau. Điều này được giải thích như sau. Giả sử trong tệp gốc aaa (tệp đưa ra để dịch) có chứa chỉ thị

#include "bbb"

Mặt khác trong tệp bbb lại chứa chỉ thị

```
#include "ccc"
```

Khi đó điều gì sẽ xẩy ra trước khi tệp aaa được dịch?.

Đầu tiên dòng

```
#include "ccc"
```

trong tệp bbb được thay bằng nội dung của tệp ccc. Ta nhận được tệp bbb mới. Tệp mới này lại được đặt xen vào tệp aaa tại vị trí của dòng lệnh

```
#include "bbb"
```

Như vậy trước khi dịch, cả hai tệp ccc và bbb đều được ghép vào tệp aaa.

Cách thức sử dụng. Có thể nêu ra 3 công dụng quan trọng của chỉ thị #include.

Công dụng 1. Tổ chức chương trình trên nhiều tệp

Một chương trình lớn thường gồm nhiều hàm. Mỗi hàm do một nhóm người xây dựng, nên các hàm thường được chứa trên các tệp khác nhau. Chỉ thị #include cho phép ghép các hàm trên các tệp khác nhau vào tệp gốc để tạo thành một chương trình hoàn chỉnh. Chương trình này được dịch và thực hiện theo các quy tắc như đối với chương trình viết trên một tệp. Như vậy mặc dầu trên thực tế chương trình được chứa trên nhiều tệp khác nhau, nhưng nhờ #include ta vẫn có thể xem như chương trình viết trên một tệp.

Công dung 2. Tổ chức thư viện.

Trong quá trình làm việc chúng ta thường xây dựng được các hàm hữu ích. Ta soan thảo các hàm này trên một hoặc nhiều tệp. Chẳng hạn các hàm về ma trận ta đưa vào tệp ma_tran.tv, các hàm đồ hoạ đưa vào tệp do_hoa.tv, các hàm vào số liệu đưa vào tệp vao_sl.tv. Nếu trong một chương trình nào đó ta cần đến các hàm vào số liệu và ma trận, thì ta chỉ cần đưa vào đầu chương trình này các chỉ thị:

```
#include "vao_sl.tv"
#include "ma_tran.tv"
```

Công dụng 3. Sử dụng các tệp tiêu đề.

Trên các tệp tiêu đề của TURBO C chứa khai báo các hàm chuẩn, định nghĩa các hằng và các kiểu dữ liệu. Khi muốn sử dụng một hàm, một hằng hay một kiểu dữ liệu nào đó, ta phải dùng chỉ thị #include để kết nối một tệp tiêu đề tương ứng. Chẳng hạn, nếu trong chương trình có dùng đến các hàm vao ra và các hàm toán học, thì ở đầu chương trình cần đưa vào các chỉ thị:

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

Kết nối hai lần.

Giả sử chương trình của ta cần đến các tệp module_a và module_b. Trong cả hai tệp trên đều chứa chỉ thị:

```
#include "defs.h"
```

Như vậy nếu ta dùng các chỉ thị:

```
#include "module_a"
#include "module_b"
```

thì tệp defs.h được kết nối hai lần vào tệp chương trình. Điều này có thể gây ra lỗi khi dịch, vì sẽ có những đối tượng được khai báo lại. Để tránh lỗi ta cần sử dụng các chỉ thị #if để bỏ qua không dịch các phiên bản được kết nối trùng lặp. Điều này sẽ nói tới trong mục §6 dưới đây.

§4. CÁC CHỈ THỊ BIÊN DỊCH CÓ ĐIỀU KIỆN #if

4.1. Chỉ thị #if dạng1:

```
#if biểu_thức_hằng
    Đoạn chương trình
#endif
```

Tác dụng của chỉ thị này như sau: Nếu biểu_thức_hằng là đúng thì TURBO C sẽ biên dịch đoạn chương trình giữa #if và #endif, trái lại đoạn này sẽ bị bỏ qua.

Biểu_thức_hằng là biểu thức mà các toán hạng trong đó đều là các hằng. Các tên đã được định nghĩa bởi #define cũng được xem là các hằng.

4.2. Chỉ thị #if dạng2:

```
#if biểu_thức_hằng
    Đoạn 1
#else
    Đoạn 2
#endif
```

Tác dụng của chỉ thị này như sau:

Nếu biểu_thức_hằng là đúng thì TURBO C sẽ biên dịch đoạn 1, trái lại sẽ biên dịch đoạn 2

Ví dụ 4.1. Xét chương trình:

```
#include "stdio.h"
#define MAX 100
main()
  {
    #if MAX > 200
      printf("\nBien dich voi MAX > 200");
    #else
      printf("\nBien dich voi MAX <= 200");
    #endif
  }
```

Chương trình này định nghĩa MAX bằng 100. Như vậy điều kiện

MAX > 200

là sai, vì vậy câu lệnh printf thứ hai được biên dịch và sau khi chạy, chương trình đưa ra thông báo:

Bien dich voi MAX <= 200

Chú ý 4.1. Các đoạn chương trình trong #if lại có thể chứa các chỉ thị tiền xử lý. Như vậy chúng ta có thể sử dụng các #if lồng nhau như trong các ví dụ sau.

Ví dụ 4.2.

```
#define MAX 100
#if MAX == 100
  #undef MAX
  #define MAX 200
  #define MIN -200
#endif
```

Ví dụ 4.3.

```
#if MAX > 200
  #if VERSION
    int port = 198;
  #else
    int port = 200
  #endif
  #else
  char buffer[100];
#endif
```

4.3. #if dạng 3.

Chỉ thị #if dạng 3 dùng để chọn một trong nhiều đoạn chương trình để dịch, nó được viết như sau:

```
#if biểu_thức_hằng_1
  Đoạn 1
#elif biểu_thức_hằng_2
  Đoạn 2
#elif biểu_thức_hằng_3
  Đoạn 3
    .
    .
    .
#elif biểu_thức_hằng_n
  Đoạn n
   [ #else
     Đoạn n+1 ]
#endif
```

Ví dụ 4.4: Đoạn chương trình dưới đây sử dụng giá trị của ACTIVE_COUNTRY để định nghĩa dấu hiệu của tiền tệ.

```
#define US 0
#define ENGLAND 1
#define FRANCE 2
#define ACTIVE_COUNTRY US
   #if ACTIVE_COUNTRY == US
      char money[] = "dollar"
   #elif ACTIVE_COUNTRY == ENGLAND
      char money[] = "pound"
   #else
      char money[] = "franc"
   #endif
```

§5. CÁC CHỈ THỊ BIÊN DỊCH CÓ ĐIỀU KIỆN #ifdef VÀ #ifndef

Một phương pháp biên dịch có điều kiện khác là sử dụng chỉ thị #ifdef và #ifndef, chúng có nghĩa là: "Nếu đã định nghĩa" và "Nếu chưa được định nghĩa". Các chỉ thị này có thể dùng theo các dạng sau:

Dạng 5.1.

```
#ifdef tên_macro
  Đoạn chương trình
#endif
```

Dạng 5.2.

```
#ifdef tên_macro
  Đoạn 1
#else
  Đoạn 2
#endif
```

Dạng 5.3.

```
#ifndef tên_macro
  Đoạn chương trình
#endif
```

Dạng 5.4.

```
#ifndef tên_macro
  Đoạn 1
#else
  Đoạn 2
#endif
```

Ý nghĩa dạng 5.1. Nếu tên_macro đã được định nghĩa (bởi #define) thì trình biên dịch C sẽ dịch đoạn chương trình nằm giữa #ifdef và #endif, trái lại đoạn này bị bỏ qua.

Ý nghĩa dạng 5.2. Nếu tên_macro đã được định nghĩa (bởi #define) thì trình biên dịch C sẽ dịch đoạn 1, trái dịch đoạn 2.

Ý nghĩa dạng 5.3. Nếu tên_macro chưa được định nghĩa thì trình biên dịch C sẽ dịch đoạn chương trình nằm giữa #ifndef và #endif, trái lại đoạn này bị bỏ qua.

Ý nghĩa dạng 5.4. Nếu tên_macro chưa được định nghĩa thì trình biên dịch C sẽ dịch đoạn 1, trái lại sẽ dịch đoạn 2.

Chú ý 5.1. Các đoạn chương trình trong #ifdef và #ifndef có thể chứa các chỉ thị tiền xử lý. Do đó chúng ta có thể sử dụng các phương án lồng nhau đối với #ifdef và #ifndef như thể đã làm đối với #if.

Ví dụ 5.1. Xét chương trình:

```
#include "stdio.h"
#define TED 10
main()
  {
    #ifdef TED
        printf("\nTED");
    #else
        printf("\nNO TED");
    #endif
    #ifndef TOP
        printf("\nNO TOP");
    #endif
  }
```

Do TED đã được định nghĩa nên câu lệnh printf thứ nhất được dịch. Do TOP chưa được định nghĩa nên câu lệnh printf cuối được dịch. Như vậy chương trình sẽ in ra các dòng sau:

```
TED
NO TOP
```

Ta có thể sử dụng các chỉ thị này để dịch chương trình theo một trong hai chế độ: Chế độ gỡ rối (in các kết quả trung gian để kiểm tra) và chế độ không gỡ rối (khi chương trình đã làm việc tốt) theo sơ đồ sau:

```
#define DEBUG
#ifdef DEBEG
    /* in kết quả trung gian */
#endif
  .
  .
  .
#ifdef DEBEG
    /* in kết quả trung gian */
#endif
  .
  .
  .
```

Khi không muốn gỡ rối thì bỏ chỉ thị #define DEBUG hoặc thay nó bằng chỉ thị: #undef DEBUG.

Ngoài cách định nghĩa một macro bằng #define, còn cho phép định nghĩa macro từ dòng lệnh khi dịch chương trình ở mức DOS bằng trình TCC.

Lựa chọn

TCC -Dsymbol

sẽ xác định macro symbol (xem như nó đã định nghĩa).

Lựa chon:

TCC -Usymbol

sẽ khiến symbol trở nên không xác định.

§6. TỔ CHỨC CÁC TỆP THƯ VIỆN

Mục này sẽ bàn đến cách tổ chức các tệp thư viên (như các tệp tiêu đề chuẩn của C hoặc các tệp thư viên riêng của mỗi người) sao cho chúng có thể được kết nối nhiều lần vào một chương trình nguồn mà không gây lỗi. Một tệp như vậy (giả sử tệp defs.h) sẽ được tổ chức như sau:

```
    /* Cấu trúc tệp defs.h */
#ifndef _DEFS_H_ #define _DEFS_H_
    /* Nội dung thực sự của tệp defs.h */
#endif
```

Dễ dàng thấy rằng mặc dù tệp defs.h được chèn vào tệp nguồn tại nhiều chỗ, nhưng nó chỉ được biên dịch khi gặp lần đầu (lúc đó _DEFS_H_ còn chưa được xác định). Các phiên bản sau của defs.h đều bị bỏ qua (vì _DEFS_H_ đã được định nghĩa trong lần đầu gặp tệp defs.h).

§7.CHỈ THỊ #error

Chỉ thị #error sẽ dừng biên dịch chương trình và in ra một thông báo lỗi. Chỉ thị này có thể sử dụng để kiểm tra tính nhất quán của các hằng. Ví dụ:

```
#if (NAME_TABLE > MAX_TABLE) ||
    (ID_TABLE > MAX_TABLE)
#error MAX_TABLE too small. Make it bigger and recompile
#endif
```

BÀI TẬP CHƯƠNG 12

Bài 1. Hàm xây dựng bởi #define khác hàm thông thường ở những điểm gì?

Bài 2. Dùng #include có thể dịch các chương trình lớn viết trên nhiều tệp được không?

Bài 3. Nêu công dụng của các chỉ thị biên dịch có điều kiện.

CHƯƠNG 13

SỬ DỤNG NGẮT TRONG C

Chương này sẽ trình bầy cách sử dụng các ngắt của DOS và ROM BIOS để can thiệp trực tiếp vào bộ nhớ và các thiết bị ngoại vi như ổ đĩa, màn hình, bàn phím. Công cụ sử dụng ở đây là các hàm, các macro, các cấu trúc và hợp định nghĩa trong dos.h.

§1. KIẾN TRÚC CỦA 8086

Để tận dụng PC, ta cần phải hiểu biết kiến trúc bộ vi xử lý 8086 và tổ chức bộ nhớ.

Các bộ vi xử lý 8086, 8088 và 80286 là các bộ vi xử lí 16 bit, chúng khác nhau một chút ít về tập lệnh cũng như đường dữ liệu. Tuy nhiên dưới góc độ lập trình, có thể coi chúng là tương đương.

1.1.Các thanh ghi. 8086 có 14 thanh ghi chia làm 4 nhóm như sau:

+ Các thanh ghi chung (nháp) AX, BX, CX, DX

AX (Accumulator: Tích luỹ)

BX (Base: Cơ sở)

CX (Count: Đếm)

DX (Data: Dữ liệu

+ Các thanh ghi đoạn CS, DS, SS, ES

CS (Code Segment: Đoạn lệnh)

DS (Data Segment: Đoạn dữ liệu)

SS (Stack Segment: Đoạn ngăn xếp)

ES (Extra Segment: Đoạn thêm)

+ Các thanh ghi độ lệch (offset) IP, SP, BP, SI, DI

IP (Instruction Pointer: Con trỏ lệnh)

SP (Stack Pointer: Con trỏ ngăn xếp)

BP (Base Pointer: Con trỏ cơ sở)

SI (Source Index: Chỉ số nguồn)

DI (Destination Index: Chỉ số đích)

+ Thanh ghi cờ FLAGS.

Các thanh ghi chung dùng để chứa kết quả tính toán hiện thời. Chúng có thể được dùng như một thanh ghi 16 bit hoặc hai thanh ghi 8 bit. Ví dụ AX là 16 bit, AH là nửa trên và AL là nửa dưới của AX.

Các thanh ghi còn lại chỉ có thể được sử dụng như các thanh ghi 16 bit.

Bộ quản lý bộ nhớ dùng các thanh ghi đoạn CS, DS, SS và ES để chuyển một địa chỉ kiểu đoạn:độ lệch (segment:offset) thành địa chỉ bộ nhớ thực.

Thanh ghi IP (Instruction Pointer) chứa địa chỉ lệnh cần thực hiện.

Bốn thanh ghi SI, DI, BP và SP được dùng cho các mục đích đặc biệt như dịch chuyển xâu ký tự, truy nhập mảng và các thao tác ngăn xếp.

Thanh ghi cờ FLAGS chứa 9 cờ (mỗi cờ chiếm một bit) được phân bố như sau:

Bit:	15	11	10	9	8	7	6	4	2	0
Cờ:		OF	DF	IF	TF	SF	ZF	AF	PF	CF

Có thể chia các cờ thành hai nhóm: 6 cờ trạng thái để ghi nhận trạng thái của bộ vi xử lý và 3 cờ điều khiển dùng để hướng dẫn cho một số lệnh của 8088.

Các cờ trạng thái:

+ CF Carry Flag — Cờ nhớ chỉ phép số học có nhớ.
+ OF Overflow Flag — Cờ tràn chỉ phép toán số học bị tràn.
+ ZF Zero Flag — Cờ zero chỉ kết quả bằng không.
+ SF Sign Flag — Cờ dấu chỉ kết quả âm.
+ PF Parity Flag — Cờ chẵn lẻ chỉ số chẵn các bit.
+ AF Auxiliary Flag — Cờ nhớ phụ chỉ sự điều chỉnh trong các phép toán trên các số BCD (Binary-Coded Decimal).

Các cờ điều khiển.

+ DF Direction Flag — Cờ định hướng điều khiển hướng tiến lùi trong các phép lặp (dùng SI hay DI).
+ IF Interrupt Flag — Cờ ngắt điều khiển khi nào cho phép ngắt
+ TF Trap Flag — Cờ bẫy điều khiển việc chạy từng lệnh.

1.2. Tổ chức bộ nhớ. 8086 sử dụng kiến trúc bộ nhớ phân đoạn (mỗi đoạn có độ lớn từ 1 đến 64 KB). Theo cách này, mỗi địa chỉ bộ nhớ gồm hai phần (đoạn, độ lệch) và được viết theo mẫu: SEGMENT:OFFSET. Ví dụ đối với địa chỉ 3000:0056 thì đoạn là 3000 và độ lệch là 0056. Địa chỉ thực được tính theo công thức:

SEGMENT*16 + OFFSET

Như vậy ứng với địa chỉ phân đoạn 0x3000:0x0056 thì địa chỉ thực là: 0x30056

Bộ nhớ của một chương trình có thể chia thành 3 đoạn. Đoạn mã chứa các chỉ thị và các hằng, nó được trỏ tới bởi thanh ghi CS. Đoạn dữ liệu chứa tất cả các biến tĩnh, biến toàn bộ và được trỏ tới bởi thanh ghi DS. Đoạn ngăn xếp chứa các biến cục bộ, được trỏ tới bởi thanh ghi SS. Các thanh ghi này có thể trỏ tới các vùng khác nhau của bộ nhớ hoặc cùng trỏ tới một vùng bộ nhớ nào đó (đối với các chương trình nhỏ nằm gọn trong một đoạn). Ngoài ra chương trình có thể truy nhập tới vùng nhớ thứ 4 (vùng phụ được trỏ tới bởi thanh ghi ES) để làm việc.

1.3. Chuyển đổi địa chỉ. Để chuyển từ địa chỉ thực sang địa chỉ phân đoạn ta dùng các macro sau:

```
unsigned FP_SEG(địa_chỉ_thực)
unsigned FP_OFF(địa_chỉ_thực)
```

Để chuyển ngược lại ta dùng macro

```
void far *MK_FP(đoạn,độ_lệch)
```

Ví dụ sau khi thực hiện các câu lệnh:

```
char buf[100];
unsigned ds,dx;
ds = FP_SEG(buf); dx = FP_OFF(buf);
```

thì ds:dx chứa địa chỉ của mảng buf.

Ngược lại, giả sử tại es:bx chứa địa chỉ của một vùng nhớ nào đó. Để thu được địa chỉ thực của vùng nhớ này tại con trỏ pchar ta làm như sau:

```
char *pchar;
pchar = (char*)MK_FP(es,bx);
```

§2. CÁC NGẮT CỦA 8086

Ngắt là gì. Có hai loại ngắt: ngắt cứng và ngắt mềm. Mặc dù có cấu trúc tương tự và cùng chia nhau bảng véc tơ ngắt, chúng có các khác nhau cơ bản.

Các ngắt cứng xuất hiện khi các thiết bị ngoại vi cần được phục vụ. Ví dụ khi ta bấm một phím thì bàn phím gây nên một ngắt. Các ngắt này định hướng theo sự kiện: phím bị ấn, việc đọc đĩa kết thúc, máy in xong một ký tự,... . Các ngắt cứng có thể xuất hiện bất kỳ lúc nào trong khi chương trình đang làm việc.

Các ngắt mềm xuất hiện khi chương trình thực hiện một lệnh ngắt (INT). Có thể xem ngắt mềm như một dạng đặc biệt của hàm.

Bảng véc tơ ngắt. Có tất cả 256 ngắt (cả cứng lẫn mềm) đánh số từ 0 đến 255. Mỗi ngắt chứa 4 byte trong vùng nhớ 1024 byte từ địa chỉ 0000:0000 đến địa chỉ 0000:0400 (hệ 16). Vùng nhớ này gọi là bảng véc tơ ngắt và dùng để lưu trữ địa chỉ của các thủ tục xử lý ngắt.

Ngắt mềm. Ngắt mềm thường được sử dụng để quản lý các thiết bị nhoại vi như đĩa, màn hình, bàn phím, máy in,... Ví dụ để gửi một ký tự ra màn hình ta có thể dùng chức năng 6 của ngắt 0x21, gọi tắt là ngắt (0x21,6).

Ta hãy xem cách thực hiện một ngắt mềm. Khi gặp một lệnh ngắt (INT), DOS nhận địa chỉ của thủ tục xử lí ngắt từ bảng véc tơ ngắt và thủ tục này được thực hiện để giải quyết các nhiệm vụ do ngắt đề ra. Một điều cần nhớ là: việc truyền dữ liệu giữa chương trình và thủ tục xử lí ngắt được thực hiện thông qua các thanh ghi.

§3. SỬ DỤNG NGẮT MỀM TRONG C

3.1. Thủ tục sử dụng ngắt mềm. Mỗi ngắt mềm được xác định bởi một cặp hai số nguyên hệ 16 là số hiệu và chức năng. Để gọi một ngắt ta tiến hành theo các bước sau:

+ Đặt chức năng vào thanh ghi AH.

+ Xác định các tham số đầu vào và đưa chúng vào các thanh ghi theo yêu cầu của mỗi ngắt.

+ Thực hiện câu lệnh gọi ngắt theo số hiệu ngắt.

+ Lấy kết quả do ngắt sản ra (tham số đầu ra) từ các thanh ghi.

3.2. Phương pháp thứ nhất. Phương pháp này dùng hàm:

```
void geninterrupt(int sh_ngat);
```

và các giả thanh ghi (pseudo register):

```
_AX _AH _AL
_BX _BH _BL
_CX _CH _CL
_DX _DH _DL
_CS _DS _ES _SS
_SP _BP _SI _DI
_FLAGS
```

Các ký hiệu trên được sử dụng như biến nguyên để truy nhập tới các thanh ghi tương ứng của bộ vi xử lí.

Ví dụ để thực hiện ngắt (0x21,6) (đưa một ký tự lên màn hình) ta làm như sau:

```
_AH = 6;            /* Đưa số hiệu chức năng vao thanh ghi AH */
_AL = 65;           /* Đưa mã ký tự (chữ A) vào thanh ghi AL */
geninterrupt(0x21); /* Gọi ngắt 0x21 */
```

Chú ý.

+ Giá trị các giả thanh ghi tự động biến đổi theo các thanh ghi, vì vậy không thể dùng chúng với mục đích lưu trữ.

+ Trước khi gán một giá trị nào đó vào giả thanh ghi, ta nên lưu trữ tình trạng hiện thời của nó vào một biến để sau này có thể khôi phục lại. Nếu không làm như vậy có thể dẫn tới treo máy. Điều này được minh hoạ trong các ví dụ của §3 (xem thêm ví dụ 7 trong §4.5).

3.3. Phương pháp thứ hai. Phương pháp này dùng các cấu trúc, các hợp (union) và các hàm sau.

+ **Các cấu trúc và hợp:**

```
struct WORDREGS /* Các thanh ghi chung 16 bit */
    {
     unsigned int ax,bx,cx,dx,si,di,cflag,flags;
    };
struct BYTEREGS /* Các thanh ghi chung 8 bit */
    {
     unsigned int al,ah,bl,bh,cl,ch,dl,dh;
    };
struct SREGS /* Các thanh ghi đoạn */
    {
     unsigned int es,cs,ss,ds;
    };
union REGS /* Hợp giữa các thanh ghi 16 bit và 8 bit */
    {
     struct WORDREGS x;
     struct BYTEREGS h;
    };
struct REGPACK
    {
     unsigned r_ax,r_bx,r_cx,r_dx;
     unsigned r_bp,r_si,r_di,r_ds,r_es,r_flags;
    };
```

+ Các hàm:

Hàm

int int86(int sh_ngat, union REGS *v, union REGS *r);

thực hiên ngắt mềm sh_ngat. Các tham số đầu vào được nạp vào các thanh ghi thông qua union v. Tham số đầu ra (kết quả) nhận được từ union r. Hàm cho lại giá trị của thanh ghi trạng thái STATUS.

Hàm

int int86x(int sh_ngat, union REGS *v,

union REGS *r, struct SREGS *s);

có tác dụng cũng như hàm int 86 nhưng tham số đầu vào được xác định thông qua union v và cấu trúc s, kết quả nhận từ union r và cấu trúc s.

Hàm

void segread(struct SREGS *s);

đặt nội dung các thanh ghi đoạn vào cấu trúc s. Hàm này được dùng để chuẩn bị cho hàm int86x.

Chú ý. Ngắt không yêu cầu đưa tham số vào các thanh ghi đoạn thì dùng hàm int86. Ngắt đòi hỏi đưa tham số vào các thanh ghi đoạn thì phải dùng hàm int86x.

Hàm

int intdos(union REGS *v, union REGS *r);

thực hiện ngắt 0x21 của DOS. Các tham số v và r được sử dụng như trong hàm int86.

Hàm

int intdosx(union REGS *v, union REGS *r, struct SREGS *s);

thực hiện ngắt 0x21 của DOS. Các tham số v, r và s được sử dụng như trong hàm int86x.

Hàm

void intr(int sh_ngat, struct REGPACK *reg);

thực hiện ngắt sh_ngat. Các tham số (đầu vào và đầu ra) đều được xác định thông qua cấu trúc reg.

Ví dụ để dùng hàm int86 thực hiên ngắt (0x21,6) (đưa một ký tự lên màn hình) ta làm như sau:

```
union REGS v,r; /* Khai báo hai biến kiểu REGS
biến v dùng để chứa tham số đầu vào
biến r dùng để chứa tham số đầu ra */
```

```
v.h.ah = 6;  /*Đưa số hiệu chức năng vao thanh ghi AH */
v.h.al = 65;  /*Đưa mã ký tự (chữ A) vào thanh ghi AL */
int86(0x21, &v, &r);         /* Gọi ngắt 0x21 */
```

3.4. Thêm vài ví dụ minh hoạ.

Xét ngắt (0x21,0x0A)

+ Tác dụng: Đọc một dãy ký tự từ bàn phím.

+ Tham số đầu vào: DS:DX chứa địa chỉ vùng nhận. Byte đầu tiên của vùng đó chứa số ký tự tối đa được nhận.

+ Tham số đầu ra: Byte thứ hai của vùng nhận chứa số ký tự đọc được. Dẫy ký tự nhập vào được lưu trữ từ byte thứ ba của vùng nhận.

Ví dụ 1 dưới đây minh hoạ cách dùng ngắt (0x21,0x0A) bằng hàm geninterrupt (phương pháp một) để xây dựng hàm nhập một chuỗi có độ dài tối đa n ký tự. Hàm trả về một giá trị bằng số ký tự thực sự đọc được. Chuỗi nhập vào được chứa trong một vùng nhớ (ít nhất n+2 byte) do con trỏ buf trỏ tới. Ví dụ 2 sau đó sẽ viết lại hàm vào chuỗi nhưng dùng hàm int86x (phương pháp hai).

Ví dụ 1:

```
#include <dos.h>
int vao_chuoi(char *buf, int n)
  {
    int m,i;
    buf[0] = n;
    _DS = FP_SEG(buf);
    _DX = FP_OFF(buf);
    _AH = 0x0A;
    geninterrupt(0x21);
    m = buf[1]; /* Số ký tự đọc được */
               /* Sắp lại các ký tự đọc
                  đuợc thành một chuỗi */
    for(i=0; i<m; ++i)
    buf[i] = buf[i+2];
    buf[m] = 0; /* Dấu hiệu kết thúc dẫy */
    return m;
  }
```

Ví dụ 2.

```
#include <dos.h>
int vao_chuoi(char *buf, int n)
 {
   int m,i;
   union REGS v,r;
   struct SREGS s;
   buf[0] = n;
   s.ds = FP_SEG(buf);
   v.x.dx = FP_OFF(buf);
   v.h.ah = 0x0A;
   int86x(0x21,&v,&r,&s);
   m = buf[1];     /* Số ký tự đọc được */
       /* Sắp lại các ký tự đọc đuợc thành một chuỗi */
     for(i=0; i<m; ++i)
     buf[i] = buf[i+2];
     buf[m] = 0;                  /* Dấu hiệu kết thúc dẫy */
     return m;
 }
```

3.5. So sánh các phương pháp.

Phương pháp thứ nhất sử dụng các giả thanh ghi, chúng luôn luôn thay đổi theo các thanh ghi tương ứng. Hơn thế nữa khi can thiệp vào giả thanh ghi thì sẽ làm thay đổi các thanh ghi và có thể bị treo máy. Vì vậy khi sử dụng phương pháp thứ nhất phải hết sức thận trọng. Các hàm trong phương pháp thứ hai sẽ truyền dữ liệu từ các cấu trúc vào các thanh ghi và lấy kết quả mhận được từ thanh ghi đặt vào cấu trúc. Chúng ta sẽ làm việc gián tiếp trên các cấu trúc chứ không làm việc trực tiếp trên thanh ghi. Do đó cách làm này thoải mái và an toàn hơn.

§4. QUẢN LÝ TỆP VÀ THƯ MỤC

Mặc dù C đã đưa ra các hàm chuẩn để thực hiện việc quản lý tệp như open, creat, read, write, close, ... hoặc fopen, fread, fwrite, ... , việc nghiên cứu các ngắt của DOS sẽ cho phép ta hiểu rõ hơn về chúng. Sau đây là các chức năng quản lý tệp của DOS từ version 2.00 trở lên.

4.1. Mã lỗi. Khi thực hiện các ngắt nêu trong 4.2, mã lỗi được đặt trong AX có ý nghĩa sau:

1 - Số hiệu chức năng không hợp lệ.
2 - Không thấy tệp tin.
3 - Không thấy đường dẫn.
4 - Hết thẻ file (số hiệu tệp).
5 - Không truy nhập được.
6 - Thẻ file không hợp lệ.
7 - Khối điều khiển bộ nhớ MCB bị phá huỷ.
8 - Không đủ bộ nhớ.
9 - Địa chỉ khối nhớ không hợp lệ.
A - Môi trường không hợp lệ.
B - Khuôn dạng không hợp lệ.
C - Mã truy nhập không hợp lệ.
D - Dữ liệu không hợp lệ.
E - Không dùng.
F - ổ đĩa không hợp lệ.
10 - Định xoá thư mục hiện hành.
11 - Không cùng thiết bị.
12 - Hết tệp tin để tìm.

4.2. Các ngắt.

1. Tạo tệp.

+ Ngắt: (0x21,0x3C)

+ Vào: DS:DX chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần tạo.

CX chứa thuộc tính tệp (xem mục 4.3, CX = 0 đối với tệp đọc ghi bình thường)

+ Ra: - Nếu cờ nhớ CF=0 (trường cflag của struct WORDREGS) là thành công và AX chứa số hiệu tệp vừa được xây dựng. số hiệu này có thể dùng trong thao tác ghi tệp (xem ngắt (0x21,0x40) dưới đây).

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 3, 4, 5.

+ **Chú ý:** Nếu tệp đã tồn tại thì nó bị xoá.

2. Mở tệp.

+ Ngắt: (0x21,0x3D)

+ Vào: DS:DX chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần mở.

AL chứa mốt mở (0: đọc, 1: ghi, 2: đọc/ghi).

+ Ra: - Nếu cờ nhớ CF=0 (trường cflag của struct WORDREGS) là thành công và AX chứa số hiệu tệp vừa được xây dựng.

Số hiệu tệp này lớn hơn 4 (vì DOS đã sử dụng các số hiệu từ 0 đến 4) và sẽ được dùng cho các thao tác trên tệp sau này.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 2, 3, 4, 5 và 12.

+ **Chú ý:**

- Chức năng này cho phép mở mọi loại tệp kể cả các tệp bị che dấu.

- Tệp phải tồn tại, nếu trái lại mã lỗi = 2.

- Nếu mở một tệp đã tồn tại theo mốt ghi (mốt 1), thì tệp vẫn không bị xoá. Chỉ các phần tử (trên tệp) bị ghi đè mới bị thay đổi. Các phần tử không bị ghi đè vẫn có nội dung cũ.

3. Đóng tệp.

+ Ngắt: (0x21,0x3E)

+ Vào: BX chứa số hiệu tệp cần đóng.

+ Các lỗi có thể gặp: 6.

4. Đọc tệp.

+ Ngắt: (0x21,0x3F)

+ Vào: BX chứa số hiệu tệp.

CX chứa số byte cần đọc.

DS:DX chứa địa chỉ vùng nhớ nhận dữ liệu.

+ Ra: - Nếu cờ nhớ CF=0 là thành công và AX chứa số byte đọc được.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 5, 6.

5. Ghi lên tệp.

+ Ngắt: (0x21,0x40)

+ Vào: BX chứa số hiệu tệp.

CX chứa số byte cần ghi.

DS:DX chứa địa chỉ vùng dữ liệu cần ghi lên tệp.

+ Ra: - Nếu cờ nhớ CF=0 là thành công và AX chứa số byte ghi được.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 5, 6.

6. Xoá tệp.

+ Ngắt: (0x21,0x41)

+ Vào: DS:DX chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần xoá. + Lỗi: Nếu CF=0 là thành công.

Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 2, 5.

7. Di chuyển con trỏ tệp.

+ Ngắt: (0x21,0x42)

+ Vào: BX chứa số hiệu tệp.

CX:DX chứa số byte cần chuyển qua n.

AL chứa một số nguyên xác định phương pháp di chuyển

- Nếu AL=0, con trỏ dời tới byte thứ n kể từ đầu tệp (n cần lớn hơn hay bằng 0)

- Nếu AL=1, con trỏ dời tới byte thứ n kể từ vị trí hiện tại (con trỏ dịch về cuối tệp nếu n>0, về đầu tệp nếu n<0).

- Nếu AL=2, con trỏ di chuyển về phía đầu tệp qua n byte kể từ cuối tệp (n cần lớn hơn hay bằng 0).

+ Ra: DX:AX chứa vị trí mới của con trỏ tệp. Nếu gọi vị trí này là m thì m được xác định như sau:

$$m = DX*65536 + AX$$

+ **Chú ý** CX và DX được xác định theo n như sau:

$$CX = n / 65536 \quad DX = n \% 65536$$

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 1, 6.

8. Thay đổi hoặc nhận biết thuộc tính tệp.

+ Ngắt: (0x21,0x43)

+ Vào: DS:DX chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần thay đổi hoặc nhận biết thuộc tính.

- Nếu AL=1, thuộc tính của tệp sẽ được thay đổi theo nội dung của CX.

- Nếu AL=0, thuộc tính hiện tại của tệp sẽ được chuyển vào CX.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 2, 3, 5.

9. Đổi tên tệp.

+ Ngắt: (0x21,0x56)

+ Vào: DS:DX chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần đổi tên.

ES:DI chứa địa chỉ dẫy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp mới.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 2, 3, 5, 11.

10. Thay đổi hoặc nhận biết ngày giờ liên kết với tệp.

+ Ngắt: (0x21,0x57)

+ Vào: BX chứa số hiệu tệp

- Nếu AL=1, thay đổi ngày tháng theo nội dung DX và giờ theo CX.
- Nếu AL=0, ngày giờ hiện tại của tệp được chuyển vào DX và CX.

(Khuôn dạng ngày giờ sẽ trình bầy trong mục 4.4 dưới đây)

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 1, 6.

11. Tạo thư mục.

+ Ngắt: (0x21,0x39)

+ Vào: DS:DX chứa địa chỉ dãy ký tự biểu thị ổ đĩa, đường dẫn và tên thư mục cần tạo.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 3, 5.

12. Xoá thư mục.

+ Ngắt: (0x21,0x3A)

+ Vào: DS:DX chứa địa chỉ dãy ký tự biểu thị ổ đĩa, đường dẫn và tên thư mục cần xoá.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 5 (thư mục khác rỗng).

13. Chuyển thư mục.

+ Ngắt: (0x21,0x3B)

+ Vào: DS:DX chứa địa chỉ dãy ký tự biểu thị ổ đĩa, đường dẫn và tên thư mục cần chuyển.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 3.

14. Lấy tên thư mục hiện hành.

+ Ngắt: (0x21,0x47)

+ Vào: DL chứa số hiệu ổ đĩa (0:ổchủ 1:ổ A 2:ổ B ...)

+ Ra: DS:SI chứa địa chỉ vùng 64 byte chứa tên thư mục hiện hành.

+ Lỗi: - Nếu cờ nhớ CF=0 là thành công.

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 15 (ổ đĩa không hợp lệ).

15. Tìm tệp đầu tiên.

+ Ngắt: (0x21,0x4E)

+ Vào: DS:DX chứa địa chỉ dãy ký tự biểu thị ổ đĩa, đường dẫn và tên tệp cần tìm.

CX chứa thuộc tính tệp cần tìm.

+ Ra: - Nếu cờ nhớ CF=0 là thành công và DOS sẽ đưa ra 43 byte thông tin về tệp tìm được vào DTA (vùng chuyển dữ liệu đĩa) hiện hành. Các thông tin được phân bố trong DTA như sau:

offset (hex)	**chiều dài** (byte)	**Nội dung thông tin**
0	1	Tên ổ đĩa
1	11	Đặc tả về tên tệp tin cần tìm
C	1	Thuộc tính tệp tin cần tìm
D	2	Số hiệu điểm vào thư mục của tệp tin tìm được
F	2	Số hiệu cluster đầu của thư mục cha
11	4	Dự trữ
15	1	Thuộc tính tệp tìm được
16	2	Thời gian của tệp tìm được
18	2	Ngày tháng năm của tệp tìm được
1A	4	Kích thước tệp tin tìm được
1E	13	Tên tệp tin tìm được

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 2 (không thấy tệp tin), 18 (hết tệp tin để tìm).

16. Tìm tệp tin thoả mãn kế tiếp.

+ Ngắt: (0x21,0x4F)

+ Công dụng: Tiếp tục công việc tìm kiếm các tập tin thoả mãn vừa được chức năng 4E khởi đầu hoặc tiếp theo chức năng 4F trước đó. Nó dựa vào thông tin chứa trong 13 byte đầu của DTA để tìm kiếm.

+ Ra: - Nếu cờ nhớ CF=0 là thành công và tên tệp tìm được sẽ chứa trong vùng DTA hiện hành (offset 0x1E).

- Nếu CF=1 là có lỗi và AX chứa mã lỗi. Các lỗi có thể gặp là 18 (hết tệp tin để tìm).

17. Khởi đầu DTA.

+ Ngắt: (0x21,0x1A)

+ Vào: DS:DX chứa địa chỉ vùng nhớ 128 byte làm DTA (DOS dùng vùng này cho việc vào ra tệp tin). DTA mặc định dài 128 byte có sẵn trong PSP tại offset 80 hex.

18. Lấy địa chỉ DTA.

+ Ngắt: (0x21,0x2F)

+ Ra: ES:BX chứa địa chỉ vùng DTA hiện hành.

4.3. Thuộc tính của tệp. Các bit của byte thuộc tính:

B7B6B5B4B3B2B1B0

có ý nghĩa như sau:

B0=0 - Tệp có thể đọc và ghi.

B0=1 - Tệp chỉ đọc, không sửa đổi hoặc xoá được.

B1=0 - Tệp hiển thị được.

B1=1 - Tệp không hiển thị trong lệnh DIR.

B2=0 - Tệp thông thường.

B2=1 - Tệp hệ thống không xuất hiện trong lệnh DIR.

B3=0 - Tệp thông thương.

B3=1 - Nhãn chứa tên đĩa trong 11 ký tự đầu tiên.

B4=0 - Tệp thông thường.

B4=1 - Thư mục con.

Ví dụ nếu byte thuộc tính có giá trị 0 thì tất cả các bit bằng 0 và đó là tệp thông thường có thể đọc và ghi lên đó. Nếu byte thuộc tính có giá trị 2 thì bit B1=1, các bit khác bằng 0, suy ra tệp có thể dọc và ghi nhưng không hiển thị.

4.4. Ngày giờ liên kết với tệp.

Ngày tháng năm chứa trong hai byte theo khuôn dạng sau (khuôn dạng ngày):

NNNNNNNTTTTNNNNN (Năm, Tháng, Ngày)

Năm chứa trong 7 bit, tháng chứa trong 4 bit và ngày trong 5 bit. Số năm bắt đầu tính từ 1980. Điều đó có nghĩa là để có năm chính xác ta cần cộng thêm 1980 vào giá trị nhận được trong 5 bit của trường năm.

Giờ phút giây chứa trong hai byte theo khuôn dạng sau (khuôn dạng giờ):

GGGGGPPPPPPGGGGG

Giờ chứa trong 5 bit, phút chứa trong 6 bit và giây trong 5 bit.

Để hiểu rõ hơn về các khuôn dạng trên hãy xem ví dụ dưới đây.

4.5. Các ví dụ minh họa

Ví dụ 1: Chương trình in ngày, tháng, năm, giờ, phút và giây liên kết với tệp.

```
/* Tim ngay gio lien ket tep */
        #include "stdio.h"
        #include "dos.h"
        main()
          {
            struct SREGS s;
            union REGS v,r;
            char ten_tep[64];
            union
            {
              int ngay_gio;
              struct
              {
                unsigned ngay:5;
                unsigned thang:4;
                unsigned nam:7;
              } ng;
            struct
              {
                unsigned giay:5;
                unsigned phut:6;
                unsigned gio:5;
              } g;
        } u;
          printf("\nTen tep can lay ngay gio: ");
          gets(ten_tep);
                /* Mo tep */
          v.h.ah = 0x3D;
          v.h.al = 0;
          s.ds = FP_SEG(ten_tep);
          v.x.dx = FP_OFF(ten_tep);
          int86x(0x21,&v,&r,&s);
          if (r.x.cflag!=0)
```

```
printf("\nLoi khi mo tep %s",ten_tep);
        /* Lay ngay gio */
v.x.bx = r.x.ax;
v.h.ah = 0;
v.h.ah=0x57;
int86(0x21,&v,&r);
        /* In ngay thang nam */
  u.ngay_gio=r.x.dx;
  printf("\nNgay %u thang %u nam %u",
  u.ng.ngay,u.ng.thang,u.ng.nam+1980);
         /* In gio phut giay */
  u.ngay_gio=r.x.cx;
  printf("\nGio %u phut %u giay %u",u.g.gio,u.g.phut,u.g.giay);
}
```

Ví du 2: Chương trình quản lý dữ liệu học sinh gồm các chức năng nhập mới, bổ sung, sửa và xem. Chương trình gồm các hàm: tạo tệp, mở tệp, đóng tệp, ghi, đọc, di chuyển con trỏ tệp đến vị trí mong muốn, di chuyển đến cuối tệp, xác định độ lớn của tệp.

```
/* Doc, ghi tep minh hoa cac ngat quan ly tep */
        #include <stdio.h>
        #include <conio.h>
        #include <dos.h>
        #include <stdlib.h>
        int tao_tep(char *ten_tep);
        int mo_tep(char *ten_tep,int mode);
        void dong_tep(int sh_tep);
        void ghi_tep(int sh_tep,void *buf,unsigned len);
        int doc_tep(int sh_tep,void *buf,unsigned len);
        void di_chuyen_ct(int sh_tep,unsigned long vt);
        void cuoi_tep(int sh_tep);
        unsigned long do_dai_tep(int sh_tep);
                /* Ham tra ve so hieu tep */
        int tao_tep(char *ten_tep)
          {
           union REGS v,r;
```

```
    struct SREGS s;
     v.h.ah = 0x3c;
     v.x.cx = 0;
     s.ds = FP_SEG(ten_tep);
     v.x.dx = FP_OFF(ten_tep);
     int86x(0x21,&v,&r,&s);
     if (r.x.cflag!= 0)
       {
        printf("\nLoi khi tao tep");
        getch();
        exit(1);
       }
     return r.x.ax;
  }
/* Ham tra ve so hieu tep, mode: 0-doc 1-ghi 2-doc/ghi */
int mo_tep(char *ten_tep,int mode)
  {
     union REGS v,r;
     struct SREGS s;
     v.h.ah = 0x3d;
     v.h.al = mode;
     s.ds = FP_SEG(ten_tep);
     v.x.dx = FP_OFF(ten_tep);
     int86x(0x21,&v,&r,&s);
     if (r.x.cflag!= 0)
       {
        printf("\nLoi khi mo tep");
        getch();
        exit(1);
       }
     return r.x.ax;
  }
void dong_tep(int sh_tep)
  {
     union REGS v,r;
     v.h.ah = 0x3e;
```

```
        v.x.bx = sh_tep;
        int86(0x21,&v,&r);
        if (r.x.cflag!= 0)
          {
            printf("\nLoi khi dong tep");
            getch();
            exit(1);
          }
      }
    void ghi_tep(int sh_tep,void *buf,unsigned len)
      {
        union REGS v,r; struct SREGS s;
        v.h.ah = 0x40;
        v.x.bx = sh_tep;
        v.x.cx = len;
        s.ds = FP_SEG(buf);
        v.x.dx = FP_OFF(buf);
        int86x(0x21,&v,&r,&s);
        if (r.x.cflag!= 0)
          {
            printf("\nLoi khi ghi tep");
            getch();
            exit(1);
          }
      }
/* Hàm cho số byte đọc được */
    int doc_tep(int sh_tep,void *buf,unsigned len)
      {
        union REGS v,r;
        struct SREGS s;
        v.h.ah = 0x3f;
        v.x.bx = sh_tep;
        v.x.cx = len;
        s.ds = FP_SEG(buf);
        v.x.dx = FP_OFF(buf);
        int86x(0x21,&v,&r,&s);
```

```
    if (r.x.cflag!= 0)
      {
        printf("\nLoi khi ghi tep");
        getch();
        exit(1);
      }
    return r.x.ax;
  }
void di_chuyen_ct(int sh_tep,unsigned long vt)
  {
    union REGS v,r;
    v.h.ah = 0x42;
    v.h.al = 0;
    v.x.bx = sh_tep;
    v.x.cx = vt/0x10000;
    v.x.dx = vt%0x10000;
    int86(0x21,&v,&r);
    if (r.x.cflag!= 0)
      {
        printf("\nLoi khi di chuyen");
        getch();
        exit(1);
      }
  }
unsigned long do_dai_tep(int sh_tep)
  {
    union REGS v,r;
    unsigned dx,ax;
    unsigned long m;
    /* Xac dinh vi tri hien tai */
    v.h.ah = 0x42;
    v.h.al = 1;
    v.x.bx = sh_tep;
    v.x.cx = 0;
```

```
        v.x.dx = 0;
        int86(0x21,&v,&r);
        dx=r.x.dx;
        ax=r.x.ax;
          /* Dua con tro den cuoi tep */
        v.h.ah = 0x42;
        v.h.al = 2;
        v.x.bx = sh_tep;
        v.x.cx = 0;
        v.x.dx = 0;
        int86(0x21,&v,&r);
          /* Xac dinh do dai tep */
        m = r.x.dx*0x10000 + r.x.ax;
          /* Dua con tro ve vi tri cu */
        v.h.ah = 0x42;
        v.h.al = 0;
        v.x.bx = sh_tep;
        v.x.cx = dx;
        v.x.dx = ax;
        return m;
    }
void cuoi_tep(int sh_tep)
    {
        union REGS v,r;
        v.h.ah = 0x42;
        v.h.al = 2;
        v.x.bx = sh_tep;
        v.x.cx = 0;
        v.x.dx = 0;
        int86(0x21,&v,&r);
    }
   typedef struct
        {
         char ht[25];
         int tuoi;
        } hs;
```

```
main()
 {
  hs h;
  char *ten_tep="hoc_sinh.sl";
  int sg,sh_tep;
  unsigned int n,sh_rec,len=sizeof(hs);
  while(1)
   {
    clrscr();
    printf("\n1.Tao tep moi de ghi ");
    printf("\n2.Ghi bo sung ");
    printf("\n3.Xem toan bo du lieu");
    printf("\n4.Xem sua tung ngnoi ");
    printf("\n5.Ket thuc ");
    sg=getch();
    clrscr();
    if(sg=='1')
     {
      sh_tep=tao_tep(ten_tep);
      sh_rec=0;
      while(1)
       {
        sh_rec++;
        printf("\nHo ten (Enter k/thuc) hs %u:", sh_rec);
        gets(h.ht);
        if(h.ht[0]==0)
           break;
        printf("\n Tuoi: ");
        scanf("%d%*c",&h.tuoi);
        ghi_tep(sh_tep,&h,len);
       }
  dong_tep(sh_tep);
 }
else if(sg=='2')
```

```
{
  sh_tep=mo_tep(ten_tep,1);
  cuoi_tep(sh_tep);
  n = do_dai_tep(sh_tep)/len;
   printf("\nDa vao so lieu %u hoc sinh",n);
   sh_rec=n;
   while(1)
     {
        sh_rec++;
        printf("\nHo ten (Enter k/thuc) hs %u: ", sh_rec);
         gets(h.ht);
         if(h.ht[0]==0)
             break;
         printf("\n Tuoi: ");
         scanf("%d%*c",&h.tuoi);
         ghi_tep(sh_tep,&h,len);
       }
     dong_tep(sh_tep);
   }
else if(sg=='3')
 {
   sh_tep=mo_tep(ten_tep,0);
   sh_rec=1;
   while(doc_tep(sh_tep,&h,len)>0)
   {
     printf("\nHoc sinh %u Ho ten %s Tuoi %d", sh_rec, h.ht, h.tuoi);
      ++sh_rec;
   }
   dong_tep(sh_tep);
   getch();
 }
else if(sg=='4')
 {
   sh_tep=mo_tep(ten_tep,2);
```

```
        n = do_dai_tep(sh_tep)/len;
        while(1)
        {
            printf("\nDa vao so lieu %u hoc sinh",n);
            printf("\nSua hoc sinh (Bam 0 - k/thuc): ");
            scanf("%u%*c",&sh_rec);
            if(sh_rec<=0 || sh_rec>n) break;
            di_chuyen_ct(sh_tep,(sh_rec-1)*len);
            doc_tep(sh_tep,&h,len);
            printf("\nHo ten %s Tuoi %d",h.ht,h.tuoi);
            printf("\nHo ten (Bam Enter k/thuc) hs %u: ", sh_rec);
            gets(h.ht);
            if(h.ht[0]==0) break;
            printf("\n Tuoi: ");
            scanf("%d%*c",&h.tuoi);
            di_chuyen_ct(sh_tep,(sh_rec-1)*len);
            ghi_tep(sh_tep,&h,len);
        }
        dong_tep(sh_tep);
      }
      else break;
    }
  }
```

Ví dụ 3: Che dấu tệp.

```
/* Che dau tep */
#include "stdio.h"
#include "dos.h"
main()
{
  struct SREGS s;
  union REGS v,r;
  char ten_tep[64];
  printf("\nTen tep can che dau: ");
  gets(ten_tep);
  v.h.ah = 0x43;
```

```
    v.h.al = 1;
    v.x.cx = 2;
    s.ds = FP_SEG(ten_tep);
    v.x.dx = FP_OFF(ten_tep);
    int86x(0x21,&v,&r,&s);
    if (r.x.ax==2)
      printf("\nTep %s khong ton tai",ten_tep);
    else
      printf("\nTep %s da bi che dau",ten_tep);
}
```

Ví dụ 4: Tìm thư mục hiện hành.

```
    /* Tim thu muc hien hanh */
#include "stdio.h"
#include "dos.h"
#include "conio.h"
main()
  {
    unsigned seg,off;
    char far *p;
    int disk_num;
    clrscr();
    printf("\nSo hieu o dia: ");
    scanf("%i",&disk_num);
    _DL=disk_num;
    _AH=0x47;
    geninterrupt(0x21);
    seg=_DS; off=_SI;
    p=(char far*)MK_FP(seg,off);
    printf("\n Thu muc hien hanh: %s",p);
    getch();
  }
```

Ví dụ 5: Tìm tệp, đặt DTA.

```
/* Tim tep, dat DTA */
    #include "stdio.h"
    #include "dos.h"
```

```
#include "conio.h"
main()
  {
    union REGS v,r;
    struct SREGS s;
    unsigned char ten[65],dta[128];
    int sotep=0;
          /* Dat DTA */
    s.ds=FP_SEG(dta);
    v.x.dx=FP_OFF(dta);
    v.h.ah=0x1A;
    int86x(0x21,&v,&r,&s);
          /* Tim tep dau */
    clrscr();
    printf("\nTen tep can tim: ");
    gets(ten);
    s.ds =FP_SEG(ten);
    v.x.dx =FP_OFF(ten);
    v.x.cx=0;
    v.h.ah =0x4E;
    int86x(0x21,&v,&r,&s);
          /* In tep tim duoc */
    if(r.x.ax!=2 && r.x.ax!=18)
      {
        printf("\n%s",dta+0x1E);
        ++sotep;
      }
    else
         printf("\nKhong co tep nao nhu vay");
              /* Tim tiep */
    while(1)
      {
        v.h.ah =0x4F;
        int86(0x21,&v,&r);
        if(r.x.ax==18)
```

```
                break;
            sotep++;
            printf("\n%s",dta+0x1E);
            if (sotep % 20 == 0)
              {
                getch();
                clrscr();
              }
        }
    printf("\nTong so co %d tep tim duoc",sotep );
    getch();
  }
```

Ví dụ 6: Tìm tệp, lấy địa chỉ DTA.

```
/* Tim tep, lay dia chi DTA */
   #include "stdio.h"
   #include "dos.h"
   #include "conio.h"
   main()
     {
       union REGS v,r; struct SREGS s;
       unsigned char ten[65],dta[128];
       char far *p;
       int sotep=0;
       /* Lay dia chi DTA */
     v.h.ah=0x2F;
     int86x(0x21,&v,&r,&s);
     p = (char*)MK_FP(s.es,r.x.bx);
     /* Tim tep dau */
     clrscr();
     printf("\nTen tep can tim: "); gets(ten);
   s.ds =FP_SEG(ten); v.x.dx =FP_OFF(ten);
   v.x.cx=0;
   v.h.ah =0x4E;
   int86x(0x21,&v,&r,&s);
```

```
/* In tep tim duoc */
if(r.x.ax!=2 && r.x.ax!=18)
  {
     printf("\n%s",p+0x1E);  ++sotep;
  }
else
printf("\nKhong co tep nao nhu vay");
/* Tim tiep */
while(1)
   {
     v.h.ah =0x4F;
     int86(0x21,&v,&r);
     if(r.x.ax==18) break;
     sotep++;
     printf("\n%s",p+0x1E);
     if (sotep % 20 == 0)
       {
         getch();  clrscr();
       }
   }
  printf("\nTong so co %d tep tim duoc",sotep );
  getch();
}
```

Ví dụ 7: Mở tệp theo phương pháp 1 (dùng các giả thanh ghi và hàm geninterrupt.

```
/* mo tep, dung gia thanh ghi */
#include "stdio.h"
#include "dos.h"
main()
  {
    unsigned ds,ax;
    union
      {
        int flags;
        struct
```

```
            {
             unsigned cf:1;
             unsigned:15;
            } f;
      }u;
     char ten_tep[64];
     printf("\nTen tep: ");
     gets(ten_tep);
     /* Luu ds */
     ds=_DS;
     _DS=FP_SEG(ten_tep);
     _DX = FP_OFF(ten_tep);
     _AH = 0x3d;
     _AL = 0;
     geninterrupt(0x21);
       /* Khoi phuc ds */
     _DS=ds;
       /* Lay so hieu tep */
     ax=_AX;
       /* Kiểm tra cờ CF */
     u.flags = _FLAGS;
     if(u.f.cf==1)
       printf("\n Loi");
     else
       printf("\n Thanh cong, so hieu tep: %d",ax);
     getch();
}
```

§5. CÁC CHỨC NĂNG QUẢN LÝ DĨA CỦA DOS VÀ ROM-BIOS

Trong mục này sẽ trình bầy một số chức năng của DOS (ngắt 0x21) và ROM-BIOS (ngắt 0x13) dùng để quản lý đĩa.

5.1. Chọn ổ đĩa chủ.

+ Ngắt: (0x21,0xE)

+ Vào: DL chứa số hiệu ổ đĩa cần chọn (0=A 1=B 2=C ...)

5.2. Xác định ổ đĩa chủ.

+ Ngắt: (0x21,0x19)

+ Ra: AL chứa số hiệu ổ chủ (0=A 1=B 2=C ...)

5.3. Nhận các thông tin về ổ đĩa chủ.

+ Ngắt: (0x21,0x1B)

+ Ra: AL chứa số sector trên một đơn vị cấp phát (allocation unit hay cluster). Nếu đĩa bị lỗi thì AL=0xFF.

CX chứa kích thước một sector.

DX chứa số cluster trên đĩa.

5.4. Nhận các thông tin về một ổ đĩa nào đó.

+ Ngắt: (0x21,0x1C)

+ Vào: DL chứa số hiệu ổ đĩa cần nhận thông tin (0=A 1=B 2=C ...).

+ Ra: Như chức năng 0x1B nhưng đó là các thông tin về ổ đĩa được chọn trong DL.

5. Nhận thông tin về vùng trống trên đĩa.

+ Ngắt: (0x21,0x36)

+ Vào: DL chứa số hiệu ổ đĩa cần nhận thông tin (0=ổ chủ 1=A 2=B 3=C ...)

+ Ra: BX chứa số cluster tự do.

DX chứa tổng số cluster trên đĩa.

AX chứa số sector trong cluster. Nếu có lỗi ở đĩa thì AX=0xFFFF.

5.6. Lấy trạng thái đĩa.

+ Ngắt: (0x13,1)

+ Vào: DL chứa số hiệu ổ đĩa cần lấy trạng thái (0=A, 1=B 0x80 = đĩa cứng một, 0x81 = đĩa cứng hai).

+ Ra: AH chứa byte trạng thái, ý nghĩa của nó như sau (hệ 16)

AH=00 - thành công.

AH=01 - lệnh không hợp lệ.

AH=02 - không tìm thấy dấu địa chỉ trên đĩa.

AH=03 - muốn ghi lên đĩa mềm đã dán băng chống ghi.

AH=04 - không tìm thấy sector.

AH=05 - không tái lập được đĩa cứng.

AH=06 - đĩa mềm đã lấy ra.

AH=07 - bảng tham số đĩa cứng bị hỏng.

AH=08 - lỗi DMA (đĩa mềm).

AH=09 - DMA ở ngoài phạm vi 64K.

AH=0A - cờ sector bị lỗi.

AH=10 - lỗi CRC hay ECC.

AH=11 - ECC đã điều chỉnh dữ liệu sai (đĩa cứng).

AH=20 - lỗi do bộ điều khiển đĩa.

AH=40 - không tìm thấy track.

AH=80 - vượt quá thời hạn cho phép.

AH=AA - ổ đĩa không sẵn sàng (đĩa cứng).

AH=BB - lỗi không xác định (đĩa cứng).

AH=CC - lỗi lúc ghi (đĩa cứng).

AH=E0 - lỗi thanh ghi trạng thái (đĩa cứng).

AH=FF - thao tác dò thất bại (đĩa cứng).

5.7. Đọc sector đĩa.

+ Ngắt: (0x13,2)

+ Công dụng: Đọc một hay nhiều sector cùng trên một track.

+ Vào: DL chứa số hiệu ổ đĩa (0=A, 1=B, 0x80 = đĩa cứng một, 0x81 = đĩa cứng hai).

DH chứa số hiệu mặt hay số hiệu đầu đọc/ghi (DH=0,..)

CH chứa số hiệu track hay cylinder (CH=0,...).

CL chứa số hiệu sector (CL=1,...).

AL chứa số lượng sector cần đọc.

ES:BX chứa địa chỉ vùng nhận thông tin đọc được.

+ Ra: - Nếu cờ CF=0 là không lỗi và AH=0, AL chứa số sector đọc được.

- Nếu CF=1, có lỗi và AH chứa byte trạng thái đĩa (xem chức năng lấy trạng thái đĩa).

5.8. Ghi sector đĩa.

+ Ngắt: (0x13,3)

+ Công dụng:Ghi lên một hay nhiều sector cùng trên một track

+ Vào: DL chứa số hiệu ổ đĩa (0=A, 1=B, 0x80 = đĩa cứng một, 0x81 = đĩa cứng hai).

DH chứa số hiệu mặt hay số hiệu đầu đọc/ghi (DH=0,..)

CH chứa số hiệu track hay cylinder (CH=0,...).

CL chứa số hiệu sector (CL=1,...).

AL chứa số lượng sector cần đọc.

ES:BX chứa địa chỉ vùng thông tin muốn ghi.

+ Ra: - Nếu cờ CF=0 là không lỗi và AH=0, AL chứa số sector ghi được.

- Nếu CF=1, có lỗi và AH chứa byte trạng thái đĩa (xem chức năng lấy trạng thái đĩa).

5.9. Kiểm tra sector đĩa.

+ Ngắt: (0x13,4)

+ Công dụng: Kiểm tra một hay nhiều sector cùng trên một track.

+ Vào: DL chứa số hiệu ổ đĩa (0=A, 1=B, 0x80 = đĩa cứng một, 0x81 = đĩa cứng hai).

DH chứa số hiệu mặt hay số hiệu đầu đọc/ghi (DH=0,..)

CH chứa số hiệu track hay cylinder (CH=0,...).

CL chứa số hiệu sector (CL=1,...).

AL chứa số lượng sector cần kiểm tra.

+ Ra: - Nếu cờ CF=0 là không lỗi và AH=0, AL chứa số sector đọc kiểm tra.

- Nếu CF=1, có lỗi và AH chứa byte trạng thái đĩa (xem chức năng lấy trạng thái đĩa).

5.10. Xác định kiểu ổ đĩa.

+ Ngắt: (0x13,15)

+ Vào: DL chứa số hiệu ổ đĩa (0=A, 1=B, 0x80 = đĩa cứng một, 0x81 = đĩa cứng hai).

+ Ra: - Nếu cờ CF=0 là không lỗi và AH cho biết kiểu đĩa. AH=0 là không có ổ đĩa. AH=1 là kiểu ổ đĩa mềm 360K. AH=2 là kiểu đĩa mềm 1.2M hay 1.44M. AH=3 là ổ đĩa cứng và khi đó CX:DX được xem như số nguyên 4 byte cho biết tổng số sector trên đĩa cứng.

- Nếu CF=1, có lỗi (số hiệu đĩa cho sai).

Các ví dụ minh hoạ.

Ví dụ 1: Xác định các thông tin về đĩa.

```
/* Xac dinh thong tin ve dia */
#include "stdio.h"
#include "dos.h"
main()
  {
    union REGS v,r;
    unsigned int sh_dia;
```

```
printf("\nSo hieu o dia (0=hien hanh 1=A 2=B 3=C 4=D): ");
scanf("%u",&sh_dia);
v.h.ah = 0x1c;
v.h.dl = sh_dia;
int86(0x21,&v,&r);
if (r.h.ah==0xFF)
     printf("\nLoi ");
else
  {
    printf("\nSo sector tren cluster %u",r.h.al);
    printf("\nKich thuoc mot sector %u",r.x.cx);
    printf("\nSo cluster tren dia %u",r.x.dx);
  }
}
```

Ví dụ 2: Xác định không gian trống trên đĩa.

```
/* Xac dinh khong gian trong tren dia */
   #include "stdio.h"
   #include "dos.h"
   main()
     {
       union REGS v,r; long unsigned int a,b,c,d;
       unsigned int sh_dia;
       printf("\nSo hieu o dia (0=o chu, 1=A, 2=B, 3=C, 4=D): ");
       scanf("%u",&sh_dia);
       v.h.ah = 0x36;
       v.h.dl = sh_dia;
       int86(0x21,&v,&r);
       if (r.x.ax==0xFFFF)
            printf("\nLoi ");
       else
         {
           a=r.x.dx; b=r.x.bx; c=r.x.ax; d=r.x.cx;
           printf("\nTong so byte tren dia %9lu",a*c*d),
           printf("\nTong so byte tu do %9lu",b*c*d);
         }
      getch();
     }
```

Ví dụ 3: Xác định loại ổ đĩa.

```
/* Xac dinh loai o dia */
    #include "stdio.h"
    #include "dos.h"
    main()
      {
        union REGS v,r;
        unsigned int sh_dia;
        printf("\nSo hieu o dia (0=A 128=Dia cung): ");
        scanf("%u",&sh_dia);
        v.h.ah = 0x15; v.h.dl = sh_dia;
        int86(0x13,&v,&r);
        if (r.x.cflag!=0)
              printf("\nLoai ");
        else
          {
            printf("\nLoai dia: %d",r.h.ah);
            if(r.h.ah>2)
              printf("\nDung luong dia cung: %lu sector",
                    r.x.cx*0x10000+r.x.dx);
          }
      }
```

§6. CÁC CHỨC NĂNG QUẢN LÝ MÀN HÌNH.

Trong mục này sẽ trình bầy các chức năng quản lý màn hình của ROM-BIOS (ngắt 0x10) và của DOS (ngắt 0x21).

6.1.Sơ lược về màn hình và cách biểu diễn ký tự.

Bộ nhớ màn hình bắt đầu từ địa chỉ:

0xB800:0x0000

Khi đưa một ký tự vào vùng bộ nhớ màn hình, thì nó sẽ hiện lên màn hình. Mỗi ký tự trên màn hình chiếm 2 byte trong bộ nhớ màn hình: byte đầu chứa mã ASCII, byte thứ hai biểu diễn mầu hiển thị gọi là byte thuộc tính. Các bit của byte thuộc tính

B7B6B5B4B3B2B1B0

được phân làm 3 nhóm:

+ Nhóm 1 gồm bit B7 biểu thị sự nhấp nháy. Nếu B7=0 thì ký tự không nhấp nháy. Nếu B7=1 thì ký tự sẽ nhấp nháy.

+ Nhóm 2 gồm các bit B6B5B4 biểu thị 8 mầu nền của ký tự.

+ Nhóm 3 gồm các bit B3B2B1B0 biểu thị 16 mầu ký tự.

Mỗi trang màn hình gồm 80x25 ký tự, do đó cần 80x25x2 = 4000 byte. Thực tế mỗi trang được phân bổ 4096 = 0x1000 byte. Như vậy:

+ Trang màn hình thứ nhất bắt đầu từ địa chỉ 0xB800:0x0000

+ Trang màn hình thứ hai bắt đầu từ địa chỉ 0xB800:0x1000

+ Trang màn hình thứ ba bắt đầu từ địa chỉ 0xB800:0x2000

+ Trang màn hình thứ tư bắt đầu từ địa chỉ 0xB800:0x3000

6.2. Các chức năng quản lý màn hình của ngắt 0x10.

1. Chọn mốt màn hình văn bản.

+ Ngắt: (0x10,0)

+ Vào: AL=0, khổ 40x25 đen/trắng.

AL=1, khổ 40x25 mầu

AL=2, khổ 80x25 đen/trắng.

AL=3, khổ 80x25 mầu.

+ **Chú ý:** Đối với màn hình mầu như VGA không có đen/trắng. khi đó:

- AL=0 và AL=1 có tác dụng như nhau
- AL=2 và AL=3 có tác dụng như nhau

2. Xác định lại kích thước con trỏ.

+ Ngắt: (0x10,1)

+ Vào: CH chứa dòng đầu của con trỏ.

CL chứa dòng cuối của con trỏ.

0 < CH <= CL < 15.

+ **Chú ý:** Đối với con trỏ chuẩn thì CH=13, CL=14

Con trỏ có kích thước lớn nhất khi CH=1, CL=14.

3. Định vị con trỏ màn hình.

+ Ngắt: (0x10,2)

+ Vào: BH chứa số hiệu trang (tính từ 0). DH chứa số hiệu dòng (tính từ 0). DL chứa số hiệu cột (tính từ 0).

4. Xác định vị trí hiện tại của con trỏ màn hình.

+ Ngắt: (0x10,3)

+ Vào: BH chứa số hiệu trang.

+ Ra: DH,DL cho biết vị trí con trỏ của trang BH. DH chứa số hiệu dòng, DL chứa số hiệu cột.

CH,CL cho biết kích thước con trỏ.

5. Chọn trang hiển thị.

+ Ngắt: (0x10,5)

+ Vào: AL chứa số hiệu trang cần hiển thị.

6. Đẩy từ dưới lên (trọng phạm vị một cửa sổ).

+ Ngắt: (0x10,6)

+ Vào: CH, CL chứa vị trí (dòng,cột) của điểm trên/trái của cửa sổ.

DH, DL chứa vị trí của điểm dưới/phải của cửa sổ.

AL chứa số dòng cần đẩy. Nếu AL=0 là đẩy toàn bộ, khi đó cả cửa sổ bị xoá.

BH chứa thuộc tính (mầu) của miền bị xoá.

+ **Chú ý:** Nếu đẩy m dòng thì m dòng ở đáy cửa sổ trở thành trống.

7. Đẩy từ trên xuống (trong phạm vi một cửa sổ).

+ Ngắt: (0x10,7)

+ Vào: Như chức năng 6.

+ Chú ý: Nếu đẩy m dòng thì m dòng ở đỉnh cửa sổ trở thành trống.

8. Đọc ký tự và thuộc tính tại vị trí con trỏ.

+ Ngắt: (0x10,8)

+ Vào: BH chứa số hiệu trang.

+ Ra: AL chứa mã ký tự, BL chứa thuộc tính.

9. Hiện ký tự và thuộc tính tại vị trí con trỏ.

+ Ngắt: (0x10,9)

+ Vào: BH chứa số hiệu trang.

AL chứa mã ký tự, BL chứa thuộc tính.

CX chứa số lần lặp lại của ký tự.

+ **Chú ý:** Không di chuyển con trỏ.

10. Hiện ký tự vị trí con trỏ.

+ Ngắt: (0x10,0xA)

+ Vào: BH chứa số hiệu trang.

AL chứa mã ký tự.

CX chứa số lần lặp lại của ký tự.

+ **Chú ý:** Không thay đổi thuộc tính và không di chuyển con trỏ

11. Xác định mốt vidio hiện tại.

+ Ngắt: (0x10,0xF)

+ Ra: AL chứa mốt và BH chứa số hiệu trang hiện tại.

12. Hiện một ký tự lên màn hình.

+ Ngắt: (0x21,2)

+ Vào: DL chứa mã ký tự cần hiển thị

13. Hiện lên màn hình một chuỗi ký tự kết thúc bởi dấu $.

+ Ngắt: (0x21,9)

+ Vào: DS:DX chứa địa chỉ chuỗi ký tự.

§7. CÁC CHỨC NĂNG QUẢN LÝ BÀN PHÍM.

Trong mục này sẽ trình bầy các chức năng quản lý bàn phím của DOS (ngắt 0x21) và của ROM-BIOS (ngắt 0x16). Các chức năng này phân biệt với nhau theo ba tiêu chuẩn:

+ Có đợi nhận ký tự gõ vào hay không?

+ Có cho hiện lên màn hình ký tự được gõ hay không?

+ Có bị tác động bởi tổ hợp phím Ctrl-Break hay không?

Nếu bị tác động thì chương trình sẽ kết thúc và trở về DOS khi gõ Ctrl-Break. Nếu không bị tác động thì các chức năng sẽ nhận được mã của Ctrl-Break (mã 3) như một ký tự thông thường khác.

7.1. Đợi nhận một ký tự từ bàn phím, có hiện lên màn hình.

+ Ngắt: (0x21,1)

+ Ra: AL chứa mã ký tự được gõ từ bàn phím.

+ Chú ý: chức năng này có bị tác động bởi Ctrl-Break.

7.2.Vào/ra trực tiếp một ký tự từ bàn phím/màn hình.

+ Ngắt: (0x21,6)

+ Vào: - Nếu DL=0xFF thì AL sẵn sàng nhận một ký. Khi đã có ký tự (đã bấm phím) thì AL chứa mã ký tự được gõ, trái lại (chưa bấm phím) thì AL=0.

- Nếu DL#0xFF thì đây chính là ký tự cần đưa ra màn hình.

+ **Chú ý:** Chức năng 6 không đợi đọc từ bàn phím, nó không hiện ký tự đã đọc.

Chức năng này không bị tác động bởi Ctrl-Break.

7.3. Đợi nhận một ký tự từ bàn phím, không hiện lên màn hình.

+ Ngắt: (0x21,7)

+ Ra: AL chứa mã ký tự được gõ từ bàn phím.

+ Chú ý: chức năng này không bị tác động bởi Ctrl-Break.

7.4. Đợi nhận một ký tự từ bàn phím, không hiện lên màn hình.

+ Ngắt: (0x21,8)

+ Ra: AL chứa mã ký tự được gõ từ bàn phím.

+ Chú ý: chức năng này có bị tác động bởi Ctrl-Break.

7.5. Đợi nhận một xâu ký tự từ bàn phím, có hiện lên màn hình.

+ Ngắt: (0x21,0xA)

+ Vào: DS:DX chứa địa chỉ vùng nhận. Byte đầu chứa một số nguyên n cho biết số ký tự tối đa được nhận là n-1.

+ Ra: Byte thứ hai của vùng nhận chứa số ký tự đọc được.

Xâu ký tự nhập vào được đặt từ vị trí thứ ba.

+ **Chú ý:** chức năng này có bị tác động bởi Ctrl-Break.

7.6. Kiểm tra trạng thái bàn phím.

+ Ngắt: (0x21,0xB)

+ Ra: AL=0xFF nếu đã có sẵn một ký tự, trái lại AL=0.

7.7. Xoá vùng đệm bàn phím và thực hiện vào/ra ký tự.

+ Ngắt: (0x21,0xC)

+ Tác dụng: Xoá vùng đệm bàn phím và thực hiện một trong các chức năng 1, 6 và 7. Trong AL đặt số hiệu chức năng cần thực hiện.

7.8. Đợi nhận một ký tự từ bàn phím, không hiện lên màn hình.

+ Ngắt: (0x16,0)

+ Ra: AL chứa mã chính (mã ASCII đối với phím thông thường hoặc bằng 0 đối với phím đặc biệt như phím chức năng).

AH chứa mã phụ đối với phím đặc biệt.

Ví dụ nếu gõ phím A thì AL=65, AH không xác định,

nếu gõ PgUp thì AL=0 , AH=73.

+ **Chú ý:** chức năng này không bị tác động bởi Ctrl-Break.

7.9. Báo cho biết đã có ký tự sẵn sàng hay chưa.

+ Ngắt: (0x16,1)

+ Ra: - Nếu cờ zero ZF=1 là không có ký tự sẵn sàng (vùng đệm bàn phím của ROM-BIOS) trống.

- Nếu cờ zero ZF=0 là có ký tự sẵn sàng, ký tự này có thể được lấy ra nhờ phục vụ 0.

+ **Chú ý:** Có thể dùng các phục vụ 0 và 1 để xây dựng đoạn chương trình xoá vùng đệm bàn phím. Các phục vụ này có ưu điểm (so với các phục vụ khác) ở chỗ sử lý được cả các ký tự thông thường cũng như đặc biệt.

Các ví dụ minh hoạ.

Ví dụ 1: Hàm đọc một chuỗi có kiểm soát số ký tự tối đa được phép nhập.

```
/* Đọc một dãy tối đa n-1 ký tự đưa vào vùng nhớ do p trỏ tới
Hàm cho lại số ký tự đọc được
*/
        int doc_day_kt(char *p,int n)
          {
            union REGS v,r;
            struct SREGS s;
            int i,m;
            v.h.ah=0xA;
            s.ds=FP_SEG(p);
            v.x.dx=FP_OFF(p);
            p[0]=n;
            int86x(0x21,&v,&r,&s);
              /* Chỉnh lý để:
                + Chuỗi ký tự bắt đầu từ p[0]
                + Có dấu hiệu kết thúc chuỗi */
            m=p[1];
            for(i=0;i<m;++i)
            p[i]=p[i+2];
            p[m]=0;
            return m;
          }
```

Ví dụ 2: Hàm bắt phím tổng quát, nhận được cả ký tự thông thường (mã đơn) và ký tự đặc biệt (mã kép).

```
/* Đọc ký tự bất kỳ.
Với ký tự đơn, mã của nó đặt trong c1.
```

Với ký tự kép, c1=0, c2 chứa mã phụ của nó. */

```
void bat_phim(int *c1,int *c2)
  {
    union REGS v,r;
    *c1 = *c2 = 0;
    v.h.ah = 0;
    int86(0x16,&v,&r);
    if (r.h.al!=0)
      *c1=r.h.al;
    else
      *c2=r.h.ah;
  }
```

Ví dụ 3:

Hàm kiểm tra bàn phím. Hàm cho giá trị 1 nếu có ký sẵn sàng, trái lại hàm cho giá trị 0.

```
int kt_bp(void)
  {
    union REGS v,r;
    union
      {
        int flags;
        struct
          {
            unsigned:6;
            unsigned zf:1;
            unsigned:9;
          } f;
      }u;
    v.h.ah = 1;
    int86(0x16,&v,&r);
    u.flags = r.x.flags;
    return (!u.f.zf);
  }
```

Ví dụ 4: Hàm xoá vùng đệm bàn phím có dùng đến các hàm kt_bp và bat_phim.

```
void xoa_bp(void)
  { int c1=0, c2=0;
    while (kt_bp())
    bat_phim(&c1,&c2);
  }
```

§8. CÁC CHỨC NĂNG ĐIỀU KHIỂN CHUỘT.

Có thể điều khiển chuột nhờ các chức năng của ngắt 0x33. Điều này sẽ trình bầy trong mục §4 chương 17. Có một chú ý ở đây là số hiệu chức năng của ngắt 0x33 phải đặt vào thanh ghi AX chứ không phải đặt vào AH như đối với các ngắt đã nói ở trên.

BÀI TẬP CHƯƠNG 13

Bài 1. Xây dựng chương trình tiện ích quản lý tệp gồm các chức năng: hiển thị danh sách các tệp (như lệnh DIR), sao chép, dịch chuyển (move), xoá tệp, khôi phục tệp mới xoá.

Bài 2. Viết chương trình lưu trữ và phục hồi boot sector và boot master.

CHƯƠNG 14

TRUY NHẬP TRỰC TIẾP VÀO BỘ NHỚ

§1. CÁC HÀM TRUY NHẬP THEO ĐỊA CHỈ PHÂN ĐOẠN

1.1. Hàm pokeb: Gửi một ký tự vào bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

void pokeb(unsigned seg, unsigned off, char value);

+ Công dụng: Gửi giá trị ký tự value vào bộ nhớ tại địa chỉ phân đoạn seg:off

1.2. Hàm peekb: Nhận một ký tự từ bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

char peekb(unsigned seg, unsigned off);

+ Công dụng: Nhận một byte tại địa chỉ phân đoạn seg:off

1.3. Hàm poke: Gửi một số nguyên vào bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

void poke(unsigned seg, unsigned off, int value);

+ Công dụng: Gửi giá trị nguyên value vào bộ nhớ tại địa chỉ phân đoạn seg:off

1.4. Hàm peek: Nhận một số nguyên từ bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

int peekb(unsigned seg, unsigned off);

+ Công dụng: Nhận một word tại địa chỉ phân đoạn seg:off

1.5. Hàm movedata: Sao các byte.

+ Nguyên mẫu trong mem.h như sau:

```
void movedata(unsigned seg_gui, unsigned off_gui,
              unsigned seg_nhan, unsigned off_nhan, int n);
```

+ Công dụng: Sao n byte từ seg_gui:off_gui đến seg_nhan:off_nhan

§2. ĐỔI TỪ ĐỊA CHỈ PHÂN ĐOẠN SANG ĐỊA CHỈ THỰC

2.1. Để chuyển từ địa chỉ thực sang địa chỉ phân đoạn ta dùng các macro:

unsigned FP_SEG(địa_chỉ_thực) unsigned FP_OFF(địa_chỉ_thực)

2.2.Để chuyển từ địa chỉ phân đoạn sang địa chỉ thực ta dùng macro:

void far *MK_FP(seg,off)

Ví dụ 1. Sau khi thực hiện các câu lệnh:

```
char buf[100];
unsigned ds,dx;
ds = FP_SEG(buf); dx = FP_OFF(buf);
```

thì ds:dx chứa địa chỉ của mảng buf.

Ví dụ 2. Sau khi thực hiện các câu lệnh:

```
char *pchar;
pchar = (char*)MK_FP(0xb800:0);
```

thì pchar trỏ tới địa chỉ đầu của bộ nhớ màn hình. Khi đó ta có thể sử dụng các lệnh gán để truy nhập trực tiếp tới bộ nhớ màn hình.

§3. CÁC VÍ DỤ MINH HOẠ

Mục này giới thiệu hai chương trình. Chương trình thứ nhất minh hoạ cách thâm nhập trực tiếp vào bộ nhớ màn hình. Chương trình thứ hai cho biết địa chỉ của các thủ tục xử lý ngắt lưu trữ trong bộ nhớ từ địa chỉ 0000:0000 đến 0000:0400 (hệ 16).

Ví dụ 1. Chương trình dưới đây gồm hàm main() và hai hàm sau:

1. Hàm cuaso

```
void cuaso(int dongt,int cott,int dongd,int cotd,int maucs);
```

thiết lập một cửa sổ mầu có toạ độ góc trên-trái là (dongt, cott) và góc dưới-phải là (dongd,cotd). Mầu cho bởi tham số maucs. Ở đây sử dụng hàm pokeb và địa chỉ phân đoạn.

2. Hàm duarmh

```
void duarmh(char *day,int dong,int cotd,int cotc,int m_nen, int m_chu);
```

sẽ đưa ra màn hình một dẫy ký tự (chứa trong dãy) tại dòng dong, từ cột cotd đến cotc. Mầu nền cho bởi m_nen, mầu chữ cho bởi m_chữ. ở đây sử dụng toán tử gán trên địa chỉ thực.

Trong hàm main() sẽ sử dụng các hàm cuaso và duarmh để tạo hai cửa sổ và viết hai dòng chữ trên trang màn hình thứ hai (từ dòng 26 đến dòng 50).

```
/* chương trình minh hoạ
   cách truy nhập trực tiếp vào bộ
   nhớ của màn hình */
   #include "dos.h"
   #include "conio.h"
   void duarmh(char *day,int dong,int cotd,int cotc,int m_nen, int m_chu);
   void cuaso(int dongt,int cott,int dongd,int cotd,int maucs);
```

```
main()
 {
  cuaso(26,1,50,80,BLUE);
  duarmh("Chuc mung nam moi",28,30,50,MAGENTA,WHITE);
  cuaso(30,20,46,60,RED);
  duarmh("Chuc mung nam moi",40,30,50,MAGENTA,YELLOW);
  getch();
 }
void cuaso(int dongt,int cott,int dongd,int cotd,int maucs)
     /* Dung dia phan doan */
 {
  int i,j,p,t,dt,dd,mau;
  union REGS v,r;
        /* Xac dinh thuoc tinh mau */
  mau = (maucs << 4)+maucs;
     /* Xac dinh trang man hinh t
       va cac chi so dong tren dt,
       dong duoi dd trong trang t  */
  t=(dongt-1)/25;
  dt=(dongt-1)-t*25;
  dd=(dongd-1)-t*25;
        /* Chon t la trang hien thi */
  v.h.ah=5;
  v.h.al=t;
  int86(0x10,&v,&r);
       /*
         Dua cac khoang trong (ma 32) va thuoc tinh mau
         vao cac vi tri thich hop cua bo nho man hinh
       */
  for(i=dt;i<=dd;++i)
    {
      p=t*4096+i*160+(cott-1)*2;
      for(j=0;j<=cotd-cott;++j)
        {
          pokeb(0xb800,p+2*j,32);
          pokeb(0xb800,p+2*j+1,mau);
        }
    }
 }
```

```
void duarmh(char *day,int dong,int cotd,int cotc,int m_nen,,int m_chu)
/* Dung dia chi thuc */
  {
    int i,p,t,d,kt,mau;
    char *buf;
    union REGS v,r;
      /* Lay dia chi thuc cua bo nho man hinh */
   buf=(char*)MK_FP(0xb800,0);
      /*  Xac dinh thuoc tinh mau */
   mau = (m_nen << 4)+m_chu;
      /*
      Xac dinh trang man hinh t
      va cac chi so dong d trong trang t
      */
   t=(dong-1)/25;
   d=dong-1-t*25;
      /* Chon t la trang hien thi */
   v.h.ah=5;
   v.h.al=t;
   int86(0x10,&v,&r);
   p=t*4096+d*160+(cotd-1)*2;
      /*
        Dua cac ky tu va thuoc tinh
        mau vao cac vi tri thich hop
        cua bo nho man hinh
      */
   for(i=0;i<=cotc-cotd;++i)
    {
      if((kt=day[i])==0)
         break;
      buf[p+2*i]=kt;
      buf[p+2*i+1]=mau;
    }
 }
```

Ví dụ 2. Chương trình cho biết địa chỉ của thủ tục xử lý ngắt n (giá trị n nhập vào từ bàn phím). Số hiệu của ngắt được tính từ 0, nhưng n được đánh số từ 1.

```
/*   Xac dinh dia chi cac thu tuc ngat */
    #include "dos.h"
    #include "conio.h"
    #include "stdio.h"
    main()
      {
      unsigned char far *p;     /*p se tro toi bang vecto ngat*/
      int n;                     /* n - so hieu ngat, n=1,2,... */
      int k;                   /* vi tri cua ngat n trong bang vecto ngat */
      unsigned seg,off;
                          /* p tro toi bang vecto ngat */
      p=(unsigned char far*)MK_FP(0,0);
      clrscr();
      while(1)
        {
          printf("\n So hieu ngat (Bam 0 - Ket thuc): ");
          scanf("%d",&n);
          if(n==0)
            break;
          k=(n-1)*4;
          off=p[k]+256*p[k+1];
          seg=p[k+2]+256*p[k+3];
          printf("\nDia chi %x:%x",seg,off);
        }
      }
```

Ví dụ 3. Thời gian hệ thống (tính theo đơn vị tích tắc) chứa trong 4 byte từ địa chỉ 0x0:0x46C. Biết 1 giờ = 65543 tích tắc, 1 phút = 1092 tích tắc và 1 giây = 18 tích tắc. Chương trình dưới đây sẽ lấy thời gian chứa trong địa chỉ 0:0x46C và đổi ra giờ, phút, giây.

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
void main()
 {
   unsigned long far *t_time;
   int gio, phut, giay;
```

```
    unsigned long  du;
    t_time = (unsigned long far*)MK_FP(0,0x46C);
    gio = (int) (*t_time / 65543);
    du =  *t_time % 65543;
    phut = (int) (du / 1092);
    du = du % 1092;
    giay = (int) (du / 18);
    printf("\nBây giờ là %d giờ %d phút %d giây",gio,phut,giay);
}
```

BÀI TẬP CHƯƠNG 14

Bài 1. Dùng cách truy nhập trực tiếp bộ nhớ màn hình để xây dựng các hàm sử lý màn hình văn bản.

Bài 2. Dùng cách truy nhập trực tiếp bộ nhớ màn hình để xây dựng các hàm đồ hoạ cơ sở như: tô điểm, tô miền, vẽ các đường và hình cơ bản (biết địa chỉ đầu của vùng nhớ ứng với màn hình đồ hoạ là 0xA000:0x0000).

CHƯƠNG 15

HÀM XỬ LÝ NGẮT VÀ CHƯƠNG TRÌNH THƯỜNG TRÚ

Từ trước tới nay chúng ta luôn luôn làm việc với các chương trình tuần tự. Điều đó có nghĩa là lệnh nọ nối sau lệnh kia theo một trật tự nào đó. Để thực sự sử dụng sức mạnh của máy tính, chúng ta cần hiểu lập trình ở mức độ ngắt. Đây là dạng khó nhất và phức tạp nhất trong lập trình. Sửa lỗi chương trình dùng ngắt có thể rất phập phù vì vấn đề có thể xuất hiện rất ngẫu nhiên.

Trong chương này chúng ta sẽ bàn đến các quy tắc để thiết lập các hàm xử lý ngắt cứng và cách sử dụng chúng trong chương trình. Bằng cách lập trình thận trọng, có thể giảm tối đa khả năng mắc lỗi và tạo ra các chương trình đáng tin cậy hơn.

§1. KHÁI NIỆM VỀ HÀM XỬ LÝ NGẮT CỨNG

1.1. Ngắt cứng.

Ngắt cứng được tạo bởi các thiết bị vào ra khi chúng cần được phục vụ. Thí dụ bàn phím tạo ngắt khi một phím được ấn. Đây là cách để bàn phím báo cho bộ vi xử lý là nó cần được chú ý.

Các dịch vụ ngắt cứng của 8086 giống như ngắt mềm. Mỗi ngắt có một thủ tục dịch vụ riêng mà địa chỉ được cất trong bảng từ 0000:0000 đến 0000:0400 (hệ 16).

Ngắt cứng có thể xẩy ra bất kỳ lúc nào trong khi thực hiện chương trình. Vì lẽ này phải cất giữ toàn bộ các thanh ghi và cờ trạng thái trong khi hoạt động. Không làm tốt điều này có thể xẩy ra các hiện tượng lạ lùng khi chạy chương trình.

1.2. Hàm xử lý ngắt cứng.

Khi xuất hiện ngắt cứng số hiệu n thì chương trình tạm thời gác lại, máy sẽ tìm (trong bảng véc tơ ngắt) địa chỉ thủ tục ngắt n để thực hiện. Nếu có cách thay địa chỉ thủ tục dịch vụ ngắt n bằng địa chỉ của một hàm do chúng ta lập ra, thì hàm của chúng ta sẽ được thực hiện mỗi khi xẩy ra ngắt n. Hàm được xây theo cách này gọi là hàm xử lý ngắt (cứng).

§2. QUY TẮC VIẾT CHƯƠNG TRÌNH XỬ LÝ NGẮT

Trong các mục 2.1 - 2.3 sẽ trình bầy các quy tắc xây dựng hàm xử lý ngắt. Hai mục sau đó sẽ nói về cấu trúc chương trình có sử dụng các hàm xử lý ngắt.

2.1. Khai báo.

Từ khoá interrupt dùng để khai báo hàm xử lý ngắt, ví dụ hàm xử lý ngắt đồng hồ có thể khai báo như sau:

void interrupt ngat_dong_ho(void);

Khi gặp khai báo interrupt, một mã đặc biệt được sinh ra để cất tất cả các thanh ghi và cờ.

2.2. Tham số. Trong hàm xử lý ngắt có thể đưa vào các đối. Việc trao đổi thông tin giữa chương trình và hàm được thực hiện thông qua các đối theo các quy tắc sau:

1. Các đối sẽ nhận giá trị của các thanh ghi ở thời điểm xẩy ra ngắt (khi bắt đầu thực hiện hàm).
2. Giá trị của các đối có thể (nhưng rất hạn chế và thận trọng) thay đổi bằng các câu lệnh trong thân hàm.
3. Khi thoát khỏi hàm giá trị của các đối sẽ gán trả lại cho các thanh ghi.

Đối (nếu có) của hàm xử lý ngắt cần được khai báo theo mẫu sau:

void interrupt ngat_ban_phim(int bp,int di,int si,int ds, int es,int dx,int cx,int bx,int ax,int ip, int cs,int flags);

Còn một lưu ý nữa khi dùng đối. Mặc dù khai báo khá nhiều đối, song trong thân hàm thường chỉ dùng một ít trong chúng. Điều này sẽ gây ra nhiều cảnh báo (warning) khi dịch. Để tránh các cảnh báo khó chịu này, ta cần dùng các lệnh tiền xử lý sau (đặt đầu và cuối hàm):

```
#pragma warn - par      /* Tắt kiểm tra tham số */
#pragma warn . par      /* Bật kiểm tra tham số */
```

2.3. Thân hàm. Các quy tắc sau cần được quán triệt trong thân hàm:

1. Không được sử dụng các dịch vụ của DOS (như các hàm vào ra).
2. Không được sử dụng các phép tính dấu phẩy động.
3. Nói chung chỉ nên dùng các phép gán, phép tính trên số nguyên, ký tự và các phép logic.

2.4. Mã tới hạn và các hàm disable, enable. Có những đoạn mã (đoạn chương trình) đòi hỏi điều kiện nghiêm ngặt sau: trong thời gian thực hiện đoạn mã này không cho phép xuất hiện ngắt cứng. Ta gọi đó là các đoạn mã tới hạn. Việc xẩy ra ngắt trong đoạn mã tới hạn có thể làm cho chương trình hoạt động rối loạn hay thiếu chính xác. Để viết các đoạn mã tới hạn ta cần dùng hai hàm sau:

```
#include <dos.h>
void disable(void);
void enable(void);
```

Hàm disable tắt hệ thống ngắt cứng, còn hàm enable thì bật nó lên. Các ngắt chỉ nên tắt trong khoảng thời gian rất ngắn.

Mọi ngắt xuất hiện khi hệ thống ngắt bị tắt sẽ phải chờ cho đến khi nó được bật trở lại.

Đoạn mã tới hạn được viết theo mẫu:

```
disable();
        các câu lệnh
enable();
```

2.5. Các hàm getvect và setvect.

Để chương trình gọi tới một hàm xử lý ngắt, trước đó cần đặt địa chỉ của nó đè lên địa chỉ của thủ tục dịch vụ ngắt chuẩn tương ứng. Điều này được thực hiện nhờ hai hàm sau:

```
void interrupt (*getvect(int sh_ngat))();
void setvect(int sh_ngat, void interrupt(*con_tro_ham)());
```

+ Hàm getvect dùng để nhận địa chỉ của một thủ tục dịch vụ ngắt chuẩn với mục đích khôi phục sau này.

+ Hàm setvect dùng để đặt địa chỉ của hàm xử lý ngắt đè lên địa chỉ của thủ tục dịch vụ ngắt chuẩn của ngắt có số hiệu sh_ngat.

Cấu trúc của một chương trình xử lý ngắt như sau:

```
            /* Khai báo con trỏ hàm để lưu địa chỉ thủ tục ngắt chuẩn */
void interrupt (*dos_int)(void);
                  /* Khai báo hàm xử lý ngắt */
void interrupt (*user_int)(void);
                  /* xây dựng hàm xử lý ngắt */
main()
  {
     /* Lưu địa chỉ của thủ tục ngắt chuẩn */
    dos_int = getvect(8);
       /* Cài đặt địa chỉ mới */
    disable();
    setvect(8,user_int);
    enable();
         .
         /* Thực hiện hàm xử lý ngắt */
         .
         /* Khôi phục địa chỉ */
    disable();
    setvect(8,dos_int);
    enable();
  }
```

Nhận xét: Hàm setvect sẽ làm thay đổi địa chỉ trong bảng véc tơ ngắt. Rõ ràng khi đang thay đổi địa chỉ thì phải tắt các ngắt. Do vậy, các đoạn cài đặt địa chỉ mới và khôi phục địa chỉ là các đoạn mã tới hạn. Sau này chúng ta còn gặp một vài đoạn mã tới hạn khác.

§3. CÁC VÍ DỤ VỀ CHƯƠNG TRÌNH XỬ LÝ NGẮT

Mục này xét ba chương trình xử lý ngắt:

+ Dùng ngắt số 8 (đồng hồ) để tính thời gian gõ 5 dòng ký tự.

+ Dùng ngắt 0x15 (ngắt bàn phím) để phát hiện mã quét của các phím.

+ Dùng ngắt 0x15 để thay đổi chức năng các phím (ví dụ khi gõ a thì hiện b, khi gõ c thì hiện d, ...).

Chương trình 1.

Đồng hồ hệ thống thực hiện ngắt số 8 ở mức 65536 lần trong một giờ (tích tắc). Một phút bằng 1092.38 và một giây bằng 18.21 tích tắc. Ta sẽ lập hàm xử lý ngắt số 8 để xác định thời gian gõ mỗi dòng và tổng thời gian gõ 5 dòng ký tự. Ta dùng biến ngoài ltime để đếm thời gian mỗi dòng và biến ttime cho tổng thời gian 5 dòng. Hàm của ta không dùng đối và đơn giản như sau:

```
void interrupt clock_interrupt()
  {
     ltime++;
     ttime++;
     (*dos_interrupt)();
  }
```

Có một vấn đề ở đây là DOS đã có sẵn thủ tục sử lý ngắt thời gian và sẽ không làm việc nếu ta bỏ qua nó. Cách giải quyết ở đây là cho hàm ngắt của ta gọi lại thủ tục này.

Tuy nhiên nếu áp dụng công thức này cho ngắt 0x15 (xem 2 chương trình tiếp theo) thì lại dẫn đến treo máy.

Dưới đây là chương trình:

```
/*
   Sử dụng ngắt số 8
    để tính thời gian gõ 5 dòng ký tự
*/
      #include "stdio.h"
      #include "dos.h"
      #include "conio.h"
```

```
int ltime=0,ttime =0,times[5];
void interrupt (*dos_interrupt)();
void interrupt clock_interrupt()
  {
  ltime++;
  ttime++; (*dos_interrupt)();
  }
main()
  {
    char line[80];int i,total,n=5;
    /* Nhận ngắt chuẩn */
    dos_interrupt = getvect(8);
        /* Cài ngắt mới */
    disable();
    setvect(8,clock_interrupt);
    enable();
    clrscr();
    printf("\nHay vao 5 dong van ban bat ky");
    for(i=0;i< n;++i)
      {
       printf("dong %d\n",i+1);
       fgets(line,sizeof(line),stdin);
           /*
            chú ý đến đoạn mã tới hạn này để bảo đảm tổng
            thời gian gõ các dòng  bằng thời gian tổng ttime
           */
       disable();
       times[i]=ltime;
       ltime=0;
       enable();
      }
   total =ttime;
   for(i=0;i<n;++i)
     printf("\ndong %d tg %d",i+1,times[i]);
     printf("\ntong %d",total);
        /* Khôi phục ngắt chuẩn */
   disable();
   setvect(8,dos_interrupt);
   enable();
  }
```

Chương trình 2: In mã ASCII và mã quét.

Trên các máy PC và các hệ sau này, BIOS có thủ tục ngắt bàn phím. Mỗi lần ấn phím, một ngắt 15 được tạo ra với AH được đặt bằng 0x4F và AL chứa mã quét của phím. Ngắt này được thiết kế để bộ phận dịch bàn phím dùng.

Ta sẽ lập hàm sử lý ngắt 15 để gửi giá trị của AL ra biến ngoài axx. Giá trị của axx được in ra trong hàm main. Bằng cách này có thể phát hiện mã quét của các phím. Trong thân của hàm sẽ dùng thanh gi AX. Để có được giá trị của các thanh ghi trong thân hàm ta cần dùng tới các đối. Nội dung các thanh ghi được gửi vào các đối trước khi thực hiện hàm. Chúng ta sẽ làm không thay đổi giá trị các đối trong thân hàm, do đó sẽ không làm thay đổi chức năng các phím. Hàm này được viết đơn giản như sau:

```
#pragma warn - par
    void interrupt new_keyboard(INTERRUPT_REGS)
      {
        if((ax>>8)==KBD_FUNCTION)
        axx= 0xff&ax;
      }
#pragma warn . par
```

Dưới đây là chương trình cho biết mã quét của các ký tự.

```
/*
Công dụng: Dùng ngắt bàn phím 0x15
            để nhận và in mã quét của các ký tự
Cách dùng: Khi bấm một phím, chương trình in ra mã
            ASCII và mã quét của phím này
Thoát khỏi chương trình bằng cách bấm ESC
Ghi chú: Minh hoạ cách viết chương trình xử lý ngắt
*/
#include "dos.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
#define KBD_INT 0x15
#define KBD_FUNCTION 0x4F
#define u_int unsigned int
#define INTERRUPT_REGS u_int bp,u_int di,u_int si,u_int ds,\
        u_int es,u_int dx,u_int cx, u_int bx, u_int ax,\
        u_int ip,u_int cs,u_int flags
```

```
#define INTERRUPT_PARAM bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flags
void interrupt new_keyboard(INTERRUPT_REGS);
void interrupt (*old_keyboard)(INTERRUPT_REGS);
static unsigned int axx;
main(void)
  {
     unsigned char ch;
     old_keyboard = getvect(KBD_INT);
     disable();
     setvect(KBD_INT,new_keyboard);
     enable();
     clrscr();
     while(1)
       {
          printf("\n Bấm một ký tự (ESC - Kết thúc): ");
          ch=getch();
          if(ch==27)
              break;
          printf("\n Ma ASC %u ma quyet %x", ch,axx);
       }
     disable();
     setvect(KBD_INT,old_keyboard);
     enable();
  }
#pragma warn - par
void interrupt new_keyboard(INTERRUPT_REGS)
  {
     if((ax>>8)==KBD_FUNCTION)
     axx= 0xff&ax;
  }
#pragma warn . par
```

Chương trình 3: Thay đổi chức năng các phím.

Ta nêu một trường hợp cần thay đổi chức năng các phím. Giả sử phím 1 (mã quét 0x02) bị hỏng trong khi đó ta thường xuyên phải dùng đến nó. Ta hãy chọn một phím ít dùng như phím ~ (mã quét 0x29, ở bên trái phím 1) và đổi chức năng để nó trở thành 1. Nói cách khác khi bấm phím ~ thì trên màn hình hiện số 1.

Muốn vậy ta lập hàm sử lý ngắt 15 để nhận giá trị các thanh ghi AH, AL và biến đổi theo quy tắc sau: Nếu AH = 0x5F và AL = 0x29 (bấm phím ~) thì đổi AL = 0x02 (phím 1). Hàm này được viết đơn giản như sau:

```
#pragma warn - par
void interrupt new_keyboard(INTERRUPT_REGS)
  {
    if ( (ax>>8)==KBD_FUNCTION && (ax&0xFF)==MA_NGA)
    ax= ax&0xFF00 | MA_1;
  }
#pragma warn . par
```

Dưới đây là chương trình đổi phím ~ thành phím 1.

```
        /* Công dụng: Gõ ~ thì hiện 1 */
#include "dos.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
#define MA_NGA 0x29
#define MA_1 0x02
#define KBD_INT 0x15
#define KBD_FUNCTION 0x4F
#define u_int unsigned int
#define INTERRUPT_REGS u_int bp, u_int di, u_int si,\
        u_int ds, u_int es, u_int dx,u_int cx, u_int bx, u_int ax,\
        u_int ip, u_int cs,u_int flags
#define INTERRUPT_PARAM bp,di,si,ds,es,dx,cx,bx,ax,ip,cs, flags
void interrupt new_keyboard(INTERRUPT_REGS);
void interrupt (*old_keyboard)(INTERRUPT_REGS);
#pragma warn - par
void interrupt new_keyboard(INTERRUPT_REGS)
  {
    if ( (ax>>8)==KBD_FUNCTION && (ax&0xFF)==MA_NGA)
    ax= ax&0xFF00 | MA_1;
  }
#pragma warn . par

void main(void)
  {
    old_keyboard = getvect(KBD_INT);
```

```
    disable();
    setvect(KBD_INT,new_keyboard);
    enable();
    clrscr();
    printf("\n Go mot day ky tu (ket thuc Enter):");
    while(1)
      if(getche()==13)
           break;
    disable();
    setvect(KBD_INT,old_keyboard);
    enable();
}
```

§4. LẬP TRÌNH THƯỜNG TRÚ

Trong các mục trước đã xét các chương trình xử lý ngắt không thường trú (gọi vắn tắt là chương trình không thường trú). Mục này sẽ nói đến các chương trình xử lý ngắt thường trú, gọi cho gọn là chương trình thường trú (kết thúc và thường trú - Terminate and Stay Resident - TSR).

4.1. Đặc điểm của chương trình TSR.

4.1.1. Chiếm dụng ngắt.

+ Đối với chương trình không thường trú, các ngắt chuẩn chỉ bị chiếm dụng trong thời gian chương trình làm việc.

Chúng được khôi phục trước khi kết thúc chương trình.

+ Đối với chương trình TSR thì các ngắt chuẩn sẽ bị chiếm dụng vĩnh viễn (chỉ khi khởi động lại máy chúng mới được khôi phục).

4.1.2. Thực hiện chương trình.

+ Chương trình không thường trú được thực hiện ở mức DOS bằng cách gõ tên của nó.

+ Chương trình TSR sẽ được thực hiện (trong bất kỳ môi trường nào) mỗi khi xuất hiện ngắt chuẩn (đang bị chiếm dụng bởi chương trình). Nếu dùng ngắt 8 (đồng hồ) thì cứ sau một tích tắc (khoảng 1/18 giây) chương trình lại được kích hoạt. Nếu dùng ngắt 0x15 (bàn phím) thì chương trình được kích hoạt mỗi khi bấm phím.

4.1.3. Vai trò của hàm ngắt và hàm main.

+ Trong chương trình không thường trú, hàm ngắt thường dùng để lấy thông từ các thanh ghi chứa vào các biến ngoài để cho hàm main xử lý.

+ Trong hàm TSR hàm ngắt phải làm tất cả mọi việc. Hàm main chỉ làm nhiệm vụ chiếm dụng ngắt và thường trú.

4.2. Kỹ thuật thường trú.

4.2.1. Làm giảm kích thước.

Chương trình thường trú chiếm dụng lâu dài một vùng nhớ, vì vậy trước khi đặt vào bộ nhớ ta cần làm giảm độ lớn của nó. Turbo C cung cấp hai biến đặc biệt kiểu unsigned int là _stklen và _heaplen để làm chuyện này. Biến _stklen dùng để quy định kích thước vùng ngăn xếp còn biến _heaplen dùng để quy định kích thước vùng heap. Độ lớn mặc định của ngăn xếp là 4K byte của heap là toàn bộ vùng nhớ có thể cung cấp (ứng với _heaplen=0). Ta nên đặt vùng nhớ tối thiểu cho ngăn xếp là 1K byte và heap là 1 word (2 byte) bằng các câu lệnh:

```
unsigned int _heaplen = 2;
unsigned int _stklen = 1024;
```

4.2.2. Hàm keep.

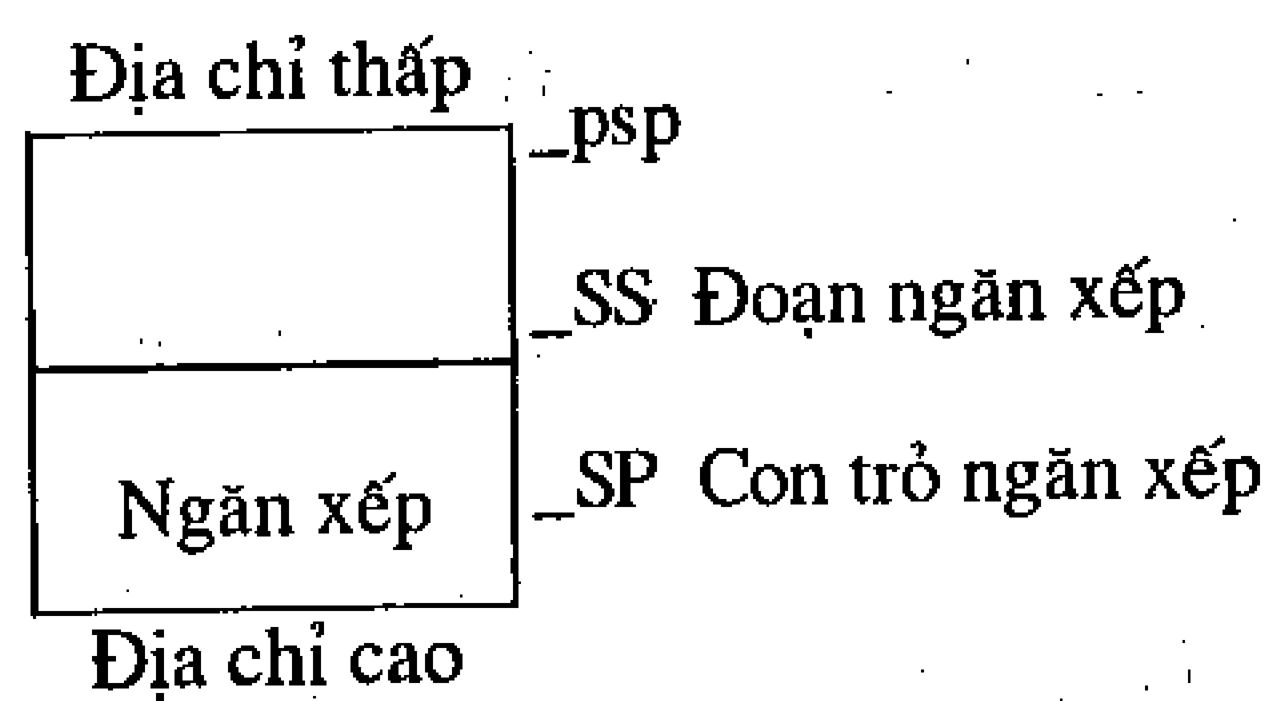
Để kết thúc và thường trú chương trình ta dùng hàm:

```
#include <dos.h>
    void keep(unsigned char status,unsigned size);
```

Hàm này có hai hai đối là mã kết thúc (status) và kích thước chương trình tính theo 16 byte. Mã kết thúc là dễ, ta dùng mã 0 (kết thúc bình thường). Dưới đây sẽ bàn đến cách tính kích thước chương.

4.2.3. Kích thước chương trình.

Xét bản đồ bộ nhớ (cho mọi mô hình bộ nhớ) chúng ta thấy đoạn ngăn xếp là ở cuối và thỉnh thoảng được nối tiếp bởi đoạn bộ nhớ phân chia động. Vì vùng đó là không dùng đến trong các chương trình thường trú nên đoạn ngăn xếp là đoạn cần dùng cuối cùng (hình vẽ).



Biến đặc biệt _psp chỉ đến tiền tố đoạn chương trình (program segment prefix - PSP) của chương trình này. Đó chính là địa chỉ đầu của chúng ta.

Vậy cần trữ từ chỗ _psp:0000 đến _SP:_SS cộng với một giá trị nào đó để chắc là có đủ chỗ cho chương trình. Kết quả là tham số size (tính theo 16 byte) của hàm keep có thể tính như sau:

size = _SS - _psp + _SP/16 + 50

Công thức này đúng đối với mọi mô hình bộ nhớ.

4.3. Tóm tắt các lệnh cần dùng để thường trú chương trình.

Để một chương trình xử lý ngắt trở thành thường trú, ta dùng các câu lệnh sau:

```
unsigned int _stklen=1024, _heaplen=2;
unsigned int size = _SS - _psp + _SP/16 + 50;
keep(0,size);
```

Trong đó:

+ Lệnh thứ nhất đặt trên (bên ngoài) hàm main.
+ Lệnh thứ hai đặt ở đầu (bên trong) hàm main.
+ Lệnh thứ ba đặt ở cuối (bên trong) hàm main.

§5. CÁC VÍ DỤ VỀ CHƯƠNG TRÌNH THƯỜNG TRÚ

Trong mục này trình bầy hai chương trình thường trú sau:

+ *Chương trình 1* được cải biên từ chương trình xử lý ngắt thứ 3 của §3. Sau khi thực hiện chương trình này, nó sẽ thương trú trong bộ nhớ và sau đó (dù ở mức nào) cứ mỗi khi bấm phím ~ thì nhận được số 1.

+ *Chương trình 2* chiếm dụng ngắt 0x15. Mỗi khi bấm phím F12 thì chương trình này được kích hoạt và nó cho hiện lên góc trái thông báo giờ như sau:

Time: giờ phút giây

Bấm tiếp F12 thì thông báo trên biến mất và ta nhận lại trạng thái cũ của màn hình.

Chương trình 1.

```
/* Chương trình thường trú 1
   Công dụng: Gõ ~ thì hiện 1
*/
#include "dos.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
#define MA_NGA 0x29
#define MA_1 0x02
#define KBD_INT 0x15
#define KBD_FUNCTION 0x4F
#define u_int unsigned int
#define INTERRUPT_REGS u_int bp,u_int di,u_int si,u_int ds,\
        u_int es,u_int dx,u_int cx,u_int bx,u_int ax,u_int ip,\
        u_int cs,u_int flags
```

```
#define INTERRUPT_PARAM bp,di,si,ds,es,dx,cx,bx,ax,ip,cs, flags
        void interrupt new_keyboard(INTERRUPT_REGS);
        void interrupt (*old_keyboard)(INTERRUPT_REGS);
        unsigned int _stklen=1024, _heaplen=2;
        #pragma warn - par
        void interrupt new_keyboard(INTERRUPT_REGS)
          {
            if ( (ax>>8)==KBD_FUNCTION && (ax&0xFF)==MA_NGA)
            ax= ax&0xFF00 | MA_1;
          }
        #pragma warn . par
        void main(void)
          {
            unsigned int size = _SS - _psp + _SP/16 + 50;
            disable();
            setvect(KBD_INT,new_keyboard);
            enable();
            keep(0,size);
          }
```

Chương trình 2. sử dụng hàm ngắt:

```
void interrupt new_keyboard(INTERRUPT_REGS);
```

Hàm này chiếm dụng ngắt 0x15 của bàn phím. Phím kích hoạt là F12. Khi bấm một phím thì hàm new_keybord được thực hiện. Hàm này sẽ kiểm tra xem có phải phím F12 hay không. Nếu đúng là F12 thì hàm lại kiểm tra xem đã có thông báo giờ chưa?. Nếu chưa, thì hàm cho hiện thông báo giờ. Trái lại, hàm xoá thông báo giờ đi. Hàm dùng biến ngoài

```
int hien_tb_gio;
```

để theo dõi điều này. Nó có giá trị -1 nếu chưa hiện thông báo, trái lại nó có giá trị 1.

Trong trường hợp cần đưa ra thông báo giờ, hàm thực hiện các việc sau:

+ Lưu vùng nhớ màn hình vào mảng buf_luu để khôi phục sau này.

+ Lấy thông tin về thời gian trong vùng nhớ 4 byte từ địa chỉ 0:0x46C của BIOS và quy ra giờ, phút, giây.

+ Đổi giờ, phút, giây ra ký tự và chuyển vào bộ nhớ màn hình.

Trong trường hợp xoá thông báo giờ, hàm sẽ khôi phục lại trạng thái cũ màn hình bị thông báo viết đè lên.

Dưới đây là chương trình.

/*Chương trình thường trú
Công dụng: Bấm F12 thì hiện hoặc xoá thông báo giờ
Hàm dùng: Ngắt 0x15, bộ nhớ màn hình và vùng nhớ thấp của ROM BIOS*/

```
#include "dos.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
#define VT 132                    /* Vị trí đầu thông báo */
#define KBD_INT 0x15
#define KBD_FUNCTION 0x4F
#define F12 0x58
#define u_int unsigned int
#define INTERRUPT_REGS u_int bp, u_int di, u_int si, u_int\
ds, u_int es, u_int dx,u_int cx, u_int bx, u_int ax, \
u_int ip,u_int cs, u_int flags
#define INTERRUPT_PARAM bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flags
            /* Lấy địa chỉ vùng nhớ màn hình và ROM BIOS */
char far *p_mh=(char far*)MK_FP(0xB800,0);
unsigned long far *t_time=(unsigned long far*)MK_FP(0,0x46C);
unsigned int _heaplen=2;
unsigned int _stklen=1024;
int hien_tb_gio = -1;
unsigned long du;                 /* Biến trung gian */
int gio,phut,giay;
char buf_time[28]={'T',47,'I',47,'M',47,'E',47,':',47,' ',47,' ',47,' ',47,' ',47,
' ',47,' ',47,' ',47, ' ',47,' ',47};
char buf_luu[28];
void interrupt new_keyboard(INTERRUPT_REGS);
#pragma warn - par
void interrupt new_keyboard(INTERRUPT_REGS)
  {
     static int ah,al,i;
     ah=ax>>8; al=ax&0xff;
     if(ah==KBD_FUNCTION && al==F12)
       {
         hien_tb_gio = -hien_tb_gio;
         if(hien_tb_gio==1)
```

```
            {
            for(i=0;i<28;++i) /* Luu trang thai man hinh */
            buf_luu[i]=p_mh[VT+i];
            /* Xac dinh gio, phut, giay */
            gio = (int) (*t_time / 65543);
            du = *t_time % 65543;
            phut= (int)(du / 1092);
            du = du % 1092;
            giay=(int)(du/18);
            /* Doi ra ky tu, dua vao mang buf_time */
            buf_time[12]=gio/10+48;
            buf_time[14]=gio%10+48;
            buf_time[18]=phut/10 +48;
            buf_time[20]=phut%10+48;
            buf_time[24]=giay/10+48;
            buf_time[26]=giay%10+48;
            /* Dua thong bao thoi gian ra man hinh */
            for(i=0;i<28;++i)
            p_mh[VT+i]=buf_time[i];
          }
        else
            /* Khoi phuc man hinh */
        for(i=0;i<28;++i)
        p_mh[VT+i]=buf_luu[i];
      }
  }
#pragma warn . par
void main(void)
  {
    unsigned keep_size;
    disable();
    setvect(KBD_INT,new_keyboard);  enable();
    keep_size = _SS - _psp +(_SP/16) +50; keep(0,keep_size);
  }
```

BÀI TẬP CHƯƠNG 15

Bài 1. Viết chương trình thường trú để xác định thời gian hoạt động của các chương trình khác.

Bài 2. Viết chương trình thương trú để máy dạo nhạc trong khi các chương trình khác đang làm việc.

CHƯƠNG 16

ÂM THANH, ÂM NHẠC

§1. TẠO ÂM THANH TRONG TURBO C

Loa của máy PC làm việc theo nguyên lý chuyển mạch bật/tắt. Trong trường hợp bật sẽ tạo âm thanh ở một tần số nào đó, còn trường hợp chuyển mạch tắt sẽ làm tắt âm thanh.

Công cụ cơ bản của Turbo C để xử lý loa là các hàm sound, nosound và delay. Tất cả các hàm này đều khai báo trong dos.h. Sound nhận một tham biến là tần số của tone cần phát ra. Ví dụ nốt đô trung có tần số 512 Hertz trong khi nốt đô ở quãng tám thấp hơn có tần số 256 Hertz và nốt đô ở quãng tám cao hơn có tần số 1024 Hertz.

Đi kèm hàm sound là hàm nosound. Mỗi khi ta bật loa để phát một tone nào đó, loa sẽ tiếp tục phát tone cho đến khi dùng nosound để tắt nó đi. Để ấn định thời gian phát tone, có thể dùng hàm delay hay sleep.

Hàm

```
void delay(int n);
```

làm cho máy tạm dừng trong n miligiây, còn hàm

```
void sleep(int n);
```

buộc máy tạm dừng trong n giây.

Mẫu tiêu biểu để tạo âm thanh là:

```
sound (256);
delay (100);
nosound();
```

§2. VÍ DỤ VỀ CÁCH TẠO ÂM THANH

Bằng cách sử dụng khéo léo các hàm sound, delay và nosound, có thể tạo ra các âm thanh kỳ thú. Sau đây là vài ví dụ.

Ví dụ 1: Tiếng Bíp. Để phát tiếng Bíp ta dùng tần số 150 Hertz trong khoảng thời gian 400 miligiây. Hãy xem chương trình sau:

```
/* Tạo tiếng Bíp */
#include <dos.h>
```

```
#include <conio.h>
#include <stdio.h>
void beep(void);
void beep(void)
  {
   sound (150);
   delay (400);
   nosound();
  }
main()
  {
    int n;
    clrscr();
    printf("\n cho một số từ 1 đến 5 ");
    scanf("%d",&n);
    if (n>5 || n<1)
      {
       beep();
       printf("\n Bạn vào sai ");
      }
    else
    printf("\n Bạn vào đúng ");
    printf("\n Gõ Enter ");
    getch();
  }
```

Ví dụ 2: BUZZ (tiếng rù rù). Để tạo tiếng rù rù ta dùng tần số 100 Hertz trong khoảng thời gian 30 miligiây như chương trình sau:

```
/* Tạo tiếng rù rù */
#include <dos.h>
#include <conio.h>
#include <stdio.h>
void buzz(void);
void buzz(void)
  {
    int n;
    for(n=1;n<=30;++i)
      {
```

```
            sound(100); delay(30);
            nosound(); delay(30);
        }
    }
main()
    {
     buzz();
    }
```

Ví dụ 3: Tiếng banh nẩy. Để tạo tiếng banh nẩy ta thay đổi giữa hai tone thấp và cao như quả bóng nẩy lên rồi lại rơi xuống. Mỗi lần banh nẩy sẽ giảm khoảng thời gian giữa hai tone này để cho tiếng banh ngày càng gần nhau. Chương trình dưới đây sẽ minh hoạ điều này.

```
/* Tiếng banh nẩy */
 #include <dos.h>
 #include <conio.h>
 #include <stdio.h>
 void ball(int high, int low);
 void ball(int high, int low)
    {
       int n;
       n=20;
       while(n>=1)
         {
          sound(low-2*n);
          delay((500*n)/20);
          nosound();
          delay (100);
          sound (high);
          delay ((500*n)/15);
          nosound();
          delay (150);
          n=n-2;
         }
    }
```

```
main()
  {
  ball (350,200);
  }
```

Ví dụ 4: Tiếng bom nổ. Hàm bombs dưới đây có 3 tham biến là high, low và t. High và low cho tần số cao thấp. Tham biến t dùng để kiểm soát thời khoảng của tone.

```
/* Tạo tiếng bom nổ */
    #include <dos.h>
    #include <conio.h>
    #include <stdio.h>
    void bombs (int high, int low, int t);
    void bombs (int high, int low, int t)
      {
        int n=low;
        while(n <= high)
          {
            sound (n);
            delay((t/n)*75);
            n=n+10;
          }
        for(n=1;n<=3;++i)
          {
            nosound();
            sound (40);
            delay (500);
            nosound();
            delay (100);
          }
        sound (40);
        delay (3000);
        nosound();
      }
```

```
main()
  {
   bombs (1200,200,500);
  }
```

Ví dụ 5: Tiếng báo động. Chương trình dưới đây tạo ra tiếng báo động cho đến khi gõ phím Enter mới chấm dứt.

```
/* Tiếng báo động */
 #include <dos.h>
 #include <conio.h>
 #include <stdio.h>
 void baodong (int high, int low);
 void baodong (int high, int low)
   {
     char ch;
     clrcsr();
     printf("\ngõ phím bất kỳ để chấm dứt báo động ");
     do
       {
        sound (high);
        delay (400);
        sound (low);
        delay (400);
       }
     while(!kbhit());
     ch=getch();
     nosound();
   }
main()
  {
   baodong (359,200);
  }
```

§3. ÂM NHẠC

Có thể dùng hàm sound với tần số của nốt nhạc để tạo nên giai điệu âm nhạc. Chương trình dưới đây gồm các hàm tạo các nốt nhạc cơ bản như đồ, rê, mi, fa, son,... Sau đó ghép chúng với nhau để tạo nên bản nhạc Lambada quen thuộc.

```
/* Lambada */
        #include <dos.h>
        #include <conio.h>
        #include <stdio.h>
        void t(void)
          {
           delay (800);
           nosound();
          }
        void d(void)
          {
           delay (400);
           nosound();
          }
        void m(void)
          {
           delay (200);
           nosound();
          }
        void mm(void)
          {
           delay (100);
           nosound();
          }
        void o(void)
          {
           delay (1600);
           nosound();
          }
        void cn (int i)
          {
           sound ((int) (i*1.5));
          }
```

```
void si(void) { cn (490); }
void do1(void) { cn (520); }
void re(void){ cn(584);}
void re1(void) { cn (294);}
void mi(void) { cn (656);}
void mi1(void) { cn (328);}
void fa(void) { cn (346);}
void son(void) { cn (394);}
void la(void) { cn (435);}
void lambada(void)
  {
    clrscr();
    printf("\nLambada");
    printf("\n Bấm Enter: Đợi chơi hết bản nhạc ");
    printf("\n mới kết thúc chương trình ");
    do
     {
       mi();d();mi();d();mi();m();re();
       m();do1();m();si();m();
       la();d();la();m();do1();m();
       si();m();la();m();
       son();m();la();m();mi1();d();mi1();
       m();re1();m();mi1();d();
       delay(500);
       mi();d();mi();d();mi();m();
       re();m();do1();m();si();m();
       la();d();la();m();
       do1();m();si();m();la();m();
       son();m();la();m();mi1();
       d();mi1();m();
       re1();m();mi1();d();
       delay(500);
       re();d();re();m();re();m();fa();o();
       fa();m();la();m();mi();d();
```

```
          re();m();do1();m();fa();m();
          fa();m();la();m();
          do1();m();mi();m();la();m();
          son();m();son();m();
          son();d();la();m();
          son();m();la();d();
          delay(500);
          re();d();re();m();re();m();
          fa();d();fa();m();
          la();m();mi1();d();
          re();m();do1();m();
          fa();m();fa();m();
          la();m();do1();m();
          si();m();la();m();
          son();m();son();m();
          son();d();re();
          m();si();m();do1();m();
          do1();m();si();m();la();d();
          delay(500);
      }
       while(!kbhit());
       nosound();
   }
main()
   {
    lambada();
   }
```

BÀI TẬP CHƯƠNG 16

Bài 1. Viết chương trình tạo ra tiếng chim hót.

Bài 2. Viết chương trình chơi một bản nhạc mà bạn yêu thích (dùng các nốt nhạc cơ bản trong chương trình Lambada).

CHƯƠNG 17

LẬP TRÌNH THEO THỜI GIAN, THEO SỰ KIỆN VÀ TRÒ CHƠI

§1. LẬP TRÌNH THEO THỜI GIAN THỰC

Chương trình theo thời gian là chương trình phải hoàn thành công việc của nó trong một thời gian giới hạn nào đó.

Các việc đặc trưng cho lập trình thời gian thực bao gồm thu thập dữ liệu, xử lý thông tin đến, điều khiển quá trình và các trò chơi. Thu nhập dữ liệu có thể rất nhạy với thời gian. Ví dụ nếu ta có một thiết bị dùng để gửi nhiệt độ của một lò nung từng 5 giây một, ta có thể bị mất dữ liệu nếu chương trình xử lý yêu cầu đề ra trong quá 5 giây. Dữ liệu đưa vào là một sự kiện phải được xử lý trong thời gian thực. Trong trò chơi ta sẽ gặp nhiều sự kiện phải xử lý theo thời gian thực. Chẳng hạn sau mỗi chu kỳ T giây phải cho chuyển động một đối tượng nào đó. Để lấy thời gian hệ thống có thể dùng hàm:

```
#include <dos>
void gettime(struct time *t);
```

trong đó t là con trỏ trỏ tới một biến cấu trúc kiểu time đã được định nghĩa như sau:

```
struct time
  {
    unsigned char ti_hour; /* Giờ */
    unsigned char ti_min; /* Phút */
    unsigned char ti_sec;  /* Giây */
    unsigned char ti_hunt;        /* Phần trăm giây (hunt)*/
  };
```

Hàm nhận giờ hệ thống và lưu vào vùng nhớ do t trỏ tới.

Để nhận thời gian và qui ra đơn vị phần trăm giây ta lập hàm sau:

```
unsigned long int tg_giay_100(void)
  {
    static struct time ti;
    unsigned long int t;
    gettime(&ti);
```

```
        /* Doi ra do vi thoi gian 1/100 giay */
        t=((ti.ti_hour*60+ti.ti_min)*
        60+ti.ti_sec)*100+ti.ti_hund;
        return t;
    }
```

Hàm sẽ được sử dụng trong tất cả các chương trình của chương này.

Có thể phân biệt 2 dạng xử lý khác nhau:

Dạng 1: Cần xử lý một đối tượng sau mỗi khoảng thời gian T nào đó. Ví dụ trong trò chơi tìm trẻ lạc (xem $3), thì cứ sau 25 hunt (1/100 giây) lại phải cho đứa trẻ di động.

Dạng 2: Cần thực hiện một việc nào đó trong khoảng thời gian T. Ví dụ máy chỉ đợi trong khoảng thời gian 50 hunt để người sử dụng bấm một phím. quá khoảng thời gian trên, máy sẽ chuyển sang việc khác.

Các bài toán trên có thể thực hiện theo các mẫu sau.

Dạng 1

```
t_bd = tg_giay_100();          /* Thời điểm đầu chu kỳ xử lý */
while(1)
  {
    t_ht = tg_giay_100();  /* Thời điểm hiện tại */
    if(t_ht-t_bd > chu_kỳ_xử_lý)  /*Đã đến chu kỳ xử lý*/
      {
        t_bd = t_ht;              /* Thời điểm đầu chu kỳ mới */
        Xử lý (phục vụ) đối tượng
      }
  }
```

Dạng 2

```
t_bd = tg_giay_100();          /* Thời điểm bắt đầu đợi */
while(1)
  {
    Nếu bấm phím
      {
        Xử lý phím nhận được
        Thoát khỏi while (dùng break)
      }
    t_ht = tg_giay_100();           /* Thời điểm hiện tại */
    if(t_ht - t_bd > Thời gian chờ) /* Quá thời chờ */
```

```
{
    Làm một điều gì đó
    Thoát khỏi while (dùng break)
}
}
```

Các chương trình trò chơi trong §3 chứa các mẫu dạng 1. Ví dụ sau đây minh hoạ một bài toán xử lý sự kiện thời gian thực dạng 2. Chương trình mô phỏng trò chơi kiểm tra trình độ bấm phím. Máy hiện lên màn hình một chữ cái và người chơi phải bấm theo đúng chữ đó. Máy chỉ đợi trong một khoảng thời gian ngắn rồi lại hiện ký tự tiếp theo. Một lỗi sẽ được tính nếu không bấm kịp hoặc bấm sai. Số chữ cái xuất hiện và mức chơi do người chơi tự chọn ở đầu chương trình. Có bốn mức như sau:

STT	**Mức**	**Thời gian đợi**
1	dễ	300
2	trung bình	200
3	khó	100
4	nhanh	50

Cũng cần nói thêm vai trò của các hàm ngẫu nhiên trong trò chơi là không thể thiếu được và ta sẽ gặp nó trong suốt chương này.

Chương trình dưới đây dùng hàm random để sinh ra các chữ cái một cách ngẫu nhiên như sau:

```
ch = toupper(random(26)+65);
```

Dưới đây là bản chương trình.

```
/*
Trò chơi kiểm tra bấm phím
*/
    #include "stdio.h"
    #include "dos.h"
    #include "stdlib.h"
    #include "time.h"
    #include "conio.h"
    unsigned long int tg_giay_100(void)
      {
        static struct time ti;
        unsigned long int t;
```

```
    gettime(&ti);
        /* Doi ra do vi thoi gian 1/100 giay */
    t=((ti.ti_hour*60+ti.ti_min)*60+ti.ti_sec)*100+ti.ti_hund;
    return t;
  }
main()
 {
   int n; /* Số chữ cái */
   int tg_doi; /* Thời gian đợi */
   int i;
   int s_loi=0; /* Số lỗi */
   int hang=10;      /* Dòng chữ hiện lên hàng 10 */
   char ch,ch1;
   unsigned long int t_bd, t_ht;
   clrscr();
   printf("\nCHUONG TRINH KIEM TRA TOC DO BAM PHIM ");
   printf("\n Chon so chu kiem tra (tu 5 den 40) ");
   scanf("%d",&n);
   if(n<5) n=5;
   else
   if(n>40) n=40;
   printf("\n\n 1.Muc de");
   printf("\n 2.Muc trung binh");
   printf("\n 3.Muc kho");
   printf("\n 4.Muc nhanh");
   printf("\n ban chon mot trong cac muc tren (tu 1 den 4) ");
   i=getch();
   switch (i)
     {
      case '1': tg_doi=300;break;
      case '2': tg_doi=200;break;
      case '3': tg_doi=100;break;
      case '4': tg_doi=50; break;
      default: tg_doi=300;
     }
```

```
clrscr();
gotoxy(20,4);
printf("CHUONG TRINH KIEM TRA TOC DO BAM PHIM");
randomize();
for(i=1;i<=n;++i)
    {
        ch= toupper(random(26)+65);
        gotoxy(20+i,hang);
        putchar(ch);
        t_bd=tg_giay_100();
        while(1)
          {
            t_ht=tg_giay_100();
            if(t_ht-t_bd>tg_doi)
              {
                s_loi++; gotoxy(20+i,hang+2);
                putchar('^'); break;
              }
           if(kbhit())
              {
                ch1=toupper(getch());
                gotoxy(20+i,hang+1);
                putchar(ch1);
                if(ch1!=ch)
                  {
                    s_loi++;
                    gotoxy(20+i,hang+2);
                    putchar('^');
                  }
                break;
              }
          }
    }
  gotoxy(20,15);
  printf("So loi %d\n",s_loi);
  getch();
}
```

§2. LẬP TRÌNH THEO HƯỚNG SỰ KIỆN

Việc thiết kế các chương trình hướng sự kiện hoàn toàn khác với các chương trình tuần tự đơn giản. Các chương trình mà ta đã thấy cho đến nay là các chương trình tuần tự. Nghĩa là các sự kiện được xử lý một cách tuần tự: xét sự kiện A xong mới xét đến B, tiếp theo B là C và v. v...

Các chương trình hướng sự kiện phản ứng với các sự kiện bên ngoài như:

+ Con chuột dịch chuyển (sự kiện chuột)

+ Đã đến lúc cần quan tâm đến một đối tượng (sự kiện thời gian)

+ Một phím được ấn (sự kiện bàn phím)

Các sự kiện được quan tâm đồng thời. Chương trình được tổ chức thành vòng lặp và trên mỗi lần lặp ta đều duyệt qua tất cả các sự kiện để biết sự kiện nào cần phục vụ. Điều này có thể thực hiện trong C theo mẫu sau:

```
while (1) /* Vòng lặp không dừng */
  {
     Nếu chuột chuyển động thì xử lý chuột
     Nếu phím được ấn thì xử lý bàn phím
     Nếu đã đến lúc cần phục vụ sự kiên A thì phục vụ A Nếu đã đến lúc
     cần phục vụ sự kiên B thì phục vụ B
     ...
  }
```

Trong §3 sẽ dùng thuật toán trên để tổ chức các trò chơi đuổi bắt.

§3. TRÒ CHƠI

Mục này trình bầy hai chương trình trò chơi, chương trình 1 dùng màn hình văn bản, chương trình 2 dùng màn hình đồ hoạ.

Chương trình 1: Tìm trẻ lạc. Nội dung như sau: một đứa trẻ (chữ T) di động theo hành trình ngẫu nhiên trên màn hình. Người chơi sử dụng các phím mũi tên để chuyển động đuổi theo đứa trẻ. Trên màn hình vị trí người chơi biểu thị bởi chữ M.

Chu kỳ di động của đứa trẻ là 25 hunt. Có thể đứa trẻ di động hơn nhanh, khó bắt. Muốn cho dễ hơn có thể sửa chu kỳ di động thành 50 hay 100.

Tổ chức chương trình

1. Phần khởi tạo gồm các nội dung:

+ Phân bố màn hình thành 2 khu vực

- Khu vực thông báo
- khu vực di động của các đối tượng

+ Hiện tên trò chơi và các thông báo liên quan
+ Xác định thời điểm bắt đầu cuộc chơi
+ Xác định vị trí ban đầu của đứa trẻ (T) và người mẹ (M)
+ Xác định chu kỳ di động của đứa trẻ

2. Trò chơi thực hiện theo vòng lặp như sau:

```
while(1) /* vòng lặp vô hạn */
  {
    1. Sự kiện bàn phím (hành trình người mẹ)
    q - break
    mũi tên - di chuyển M phím khác - không xét
    2. Kiểm tra xem gặp nhau chưa?
    gặp nhau - phát âm thanh, hiện thông báo, break
    chưa - tiếp tục
    3. Sự kiện thời gian (hành trình ngẫu nhiên của trẻ)
    nếu đến chu kỳ di động - cho di động
    trái lại - không làm gì cả
    4. Thông báo diễn biến trò chơi
  }
```

```
/* Hai nguoi tim nhau */
    #include "stdio.h"
    #include "conio.h"
    #include "time.h"
    #include "dos.h"
    #include "stdlib.h"
    unsigned long int tg_giay_100(void)
      {
        static struct time ti;
        unsigned long int t;
        gettime(&ti);
            /* Doi ra do vi thoi gian 1/100 giay */
        t=((ti.ti_hour*60+ti.ti_min)*60+ti.ti_sec)*100+ti.ti_hund;
        return t;
      }
```

```
#define PERIOD 25        /* chu ky chuyen dong */
main()
 {
        /* Vi tri ban dau cua me va con */
  int me_x_c=5,me_y_c=5, con_x_c=40,con_y_c=12;
  int me_x_m=5,me_y_m=5, con_x_m=40,con_y_m=12;
  int c1,c2;                     /* Dung de bat phim */
  unsigned long int tg=0;  /* Thoi gian da troi qua */
  unsigned long int t_bd_choi;
  unsigned long int t_d_ck;
  unsigned long int t_ht;
       /* Xac dinh thoi diem ban dau */
  t_bd_choi=t_d_ck= tg_giay_100();
   clrscr();
   gotoxy(me_x_m,me_y_m); putch('M');
   gotoxy(con_x_m,con_y_m); putch('T');
   gotoxy(4,2);
   printf("Tro choi tim tre lac - Bam Q ket thuc");
   while(1)
    {
      /* Su kien ban phim */
    if(kbhit())
     {
      if( (c1=getch())==0)
          c2=getch();
      if(c1=='q'||c1=='Q')
          break;
      if(c1==0)
       {
        if(c2==77)
          me_x_m++;               /* phai */
        else
          if(c2==75) me_x_m--;    /*trai*/
          else
           if(c2==80) me_y_m++;   /*duoi*/
          else
           if(c2==72) me_y_m--;   /*tren*/
```

```
            /* Chinh ly */
            if(me_x_m<1) me_x_m=1;
            if(me_x_m>80) me_x_m=80;
            if(me_y_m<5) me_y_m=5;
            if(me_y_m>25)
              me_y_m=25;
                   /* Di chuyen */
            if((me_x_m!=me_x_c)||(me_y_m!=me_y_c) )
              {
               gotoxy(me_x_c,me_y_c);
               putch(' ');
               gotoxy(me_x_m,me_y_m);
               putch('M');
               me_x_c=me_x_m;
               me_y_c=me_y_m;
           }
       }
    }
/* Kiem tra gap nhau ? */
   if(me_x_m==con_x_m && me_y_m==con_y_m)
    {
      sound(200);
      delay(100);
      puts(" Da tim thay ");
      break;
    }
      /* Su kien thoi gian ( hanh trinh dua tre ) */
    t_ht = tg_giay_100();
    tg=t_ht - t_bd_choi;
    if(t_ht - t_d_ck>PERIOD)
    {
     t_d_ck=t_ht;
     con_y_m = random(21) + 5;
     con_x_m = random(80) + 1;
     gotoxy(con_x_c,con_y_c);
     putch(' ');
```

```
        gotoxy(con_x_m,con_y_m);
        putch('T');
        con_x_c=con_x_m; con_y_c=con_y_m;
      }
        /* thong bao thoi gian */
      gotoxy(44,2);
      printf("Thoi gian da troi qua: %lu giay",tg/100);
    }   /*Ket thuc tro choi*/
    getch();
    clrscr();
  }
```

Chương trình 2: Bắn tầu lạ. Nội dung như sau: Trên bầu trời đầy sao có một tầu lạ chuyển động ngang lượn đi lượn lại từ trái sang phải rồi lại từ phải qua trái. Tốc độ chuyển động lúc nhanh lúc chậm không quy luật (dùng hàm ngẫu nhiên). Người chơi có thể di động một bệ súng (đặt ở đáy màn hình) sang phải hay trái nhờ các phím mũi tên tương ứng và có thể dùng phím mũi tên lên để bắn. Đạn di chuyển theo quỹ đạo thẳng từ bệ súng lên trên với tốc độ nhanh hơn tầu (nhưng cũng lúc nhanh lúc chậm). Lúc viên đạn đang bay thì không có quyền bắn tiếp. Khi đạn chạm tầu sẽ được cộng một điểm và một âm thanh được phát ra. Trò chơi kết thúc khi bấm Q.

Các hàm dùng trong chương trình

Đây là chương trình khá phức tạp sử dụng màn hình đồ hoạ. Trong chương trình sử dụng các hàm sau:

1. unsigned long int tg_giay_100(void);

   Dùng để lấy thời gian hệ thống và quy ra hunt.

2. Void kt_va_cham(int x,int y,int *xd,int *yd, int *ban);

   Dùng để kiểm tra va chạm giữa tầu và đạn.

3. Void ve_bt(void);

   Dùng để vẽ bầu trời đầy sao.

4. Void tao_tau(void);

   Dùng để vẽ tầu và lưu vào bộ nhớ.

5. Void tao_dan(void);

   Dùng để vẽ đạn và lưu vào bộ nhớ.

6. Void tao_be(void);

   Dùng để vẽ bệ súng và lưu vào bộ nhớ.

7. Void tau_cd(int *x, int *y, int *ch_cd);

   Dùng để làm tầu chuyển động.

8. Void dan_cd(int *x, int *y, int *ban);

   Dùng để làm đạn chuyển động.

Các sự kiện cần quan tâm trong vòng lặp

1. Sự kiện bàn phím theo dõi việc bắn
2. Sự kiện bàn phím theo dõi việc di động bệ súng
3. Sự kiện thời gian theo dõi việc chuyển động của đạn
4. Sự kiện thời gian theo dõi việc chuyển động của tầu

Chương trình:

```
/* Bắn tầu lạ */
    #include <graphics.h>
    #include <stdio.h>
    #include <conio.h>
    #include <alloc.h>
    #include <stdlib.h>
    #include <math.h>
    #include <time.h>
    #include <dos.h>
    #define CK_TAU 6
    #define CK_DAN 4
    #define BUOC_CD 10
    char *ptau, *pdan, *pbe;
    int xmax, ymax;
    int diem=0;
    static char s_diem[5];
    unsigned long int tg_giay_100(void);
    void kt_va_cham(int x,int y,int *xd,int *yd, int *ban);
    void ve_bt(void);
    void tao_tau(void);
    void tao_dan(void);
    void tao_be(void);
    void tau_cd(int *x, int *y, int *ch_cd);
    void dan_cd(int *x, int *y, int *ban);
    unsigned long int tg_giay_100(void)
```

```
{
   static struct time ti;
   unsigned long int t;
   gettime(&ti);
        /* Doi ra do vi thoi gian 1/100 giay */
   t=((ti.ti_hour*60+ti.ti_min)*60+ti.ti_sec)*100+ti.ti_hund;
   return t;
}
void kt_va_cham(int x,int y,int *xd,int *yd, int *ban)
   {
   if( abs((*xd+5)-(x+20))<20 && abs((*yd+5)-(y+12))<12)
      {
        sound(100); delay(100);
        setcolor(0);
        outtextxy(500,50,s_diem);
        setcolor(YELLOW); diem++;
        itoa(diem,s_diem,10);
        outtextxy(500,50,s_diem);
        *ban =1;
        putimage(*xd,*yd,pdan ,XOR_PUT);
        *xd=*yd=0; nosound();
      }
   }
 void tau_cd(int *x, int *y, int *ch_cd)
   {
    int xx,yy,s; int b;
      xx=*x; yy=*y;
      b=random(BUOC_CD)+1;
           /* Xóa */
      putimage(xx,yy,ptau,XOR_PUT);
      s= xx + (*ch_cd)*b;
      if(s+40>xmax || s<0) *ch_cd = -(*ch_cd);
      xx += (*ch_cd)*b;
      putimage(xx,yy,ptau,XOR_PUT);
      *x=xx; *y=yy;
    }
```

```
void dan_cd(int *x, int *y, int *ban)
  {
    int b,xx,yy,s;
    b=random(BUOC_CD)+1;
    xx=*x; yy=*y;
        /* Xóa */
    putimage(xx,yy,pdan ,XOR_PUT);
    s= yy - b;
    if(s < ymax/2) *ban = 1;       /* Cho phep ban tiep */
    else
     {
       yy = s;
       putimage(xx,yy,pdan,XOR_PUT);
       *x=xx; *y=yy;
     }
  }
void ve_bt(void)
  {
    int i,x,y;
    /* Vẽ bầu trời đầy sao */
    cleardevice();
    for(i=1;i<=1000;++i)
     {
       x=random(getmaxx());
       y=random(getmaxy());
       putpixel(x,y,random(getmaxcolor()));
     }
  }
void tao_dan(void)
  {
    unsigned n;
    setcolor(YELLOW);
    cleardevice();
    circle(5,5,5);
    setfillstyle(SOLID_FILL,YELLOW);
```

```
        floodfill(5,5,YELLOW);
        n=imagesize(0,0,10,10);
        pdan =(char*)malloc(n);
        getimage(0,0,10,10,pdan );
    }
void tao_be(void)
    {
        unsigned n;
        setcolor(BLUE);
        cleardevice();
        setfillstyle(SOLID_FILL,BLUE);
        bar(0,0,30,20);
        n=imagesize(0,0,30,20);
        pbe =(char*)malloc(n);
        getimage(0,0,30,20,pbe);
    }
void tao_tau(void)
    {
        unsigned n;
        setcolor(RED);
        /* Vẽ tầu vũ trụ */
        ellipse(100,50,0,360,20,8);
        ellipse (100,46,190,357,20,6);
        line(107,44,110,38);
        circle(110,38,2);
        line(93,44,90,38);
        circle(90,38,2);
        setfillstyle(SOLID_FILL,BLUE);
        floodfill(101,54,RED);
        setfillstyle(SOLID_FILL,MAGENTA);
        floodfill(94,45,RED);
            /* Lưu ảnh của tầu vũ trụ vào biến */
        n=imagesize(79,36,121,59);
        ptau=(char*)malloc(n);
        getimage(79,36,121,59,ptau);
        cleardevice();
    }
```

```
void main(void)
  {
    unsigned long int t_bd_tau,t_ht_tau,t_bd_dan,t_ht_dan;
    int ch,ch1;
    int mh=0,mode=0, ban=1;
    int xd, yd, xb, yb;
    int x=0, ch_tau=1, y;
    initgraph(&mh,&mode,"");
    if(graphresult()!=0) exit(1);
    xmax=getmaxx();
    ymax=getmaxy();
    y=ymax/2;
    tao_tau();
    tao_dan();
    tao_be();
    ve_bt();
    xb=xmax/2-15;
    yb=ymax-15;
    randomize();
    putimage(xb,yb,pbe,XOR_PUT);
         /* Thong bao diem */
         settextstyle(0,0,3);
    setcolor(MAGENTA);
    outtextxy(5,50,"Q - Quit");
    setcolor(YELLOW);
    outtextxy(380,50,"DIEM: ");
    itoa(diem,s_diem,10);
    outtextxy(500,50,s_diem);
    t_bd_tau=tg_giay_100();
    putimage(x,y,ptau,XOR_PUT);
 while(1)
   {
     /* Su kien ban phim */
    if(kbhit())
    {
    if( (ch=getch()) == 0)
```

```
        ch1=getch();
      if(ch=='q'||ch=='Q')
        break;
      else
      if(ch==0&&(ch1==77||ch1==75)) /* be -> */
        {
          putimage(xb,yb,pbe,XOR_PUT);
          if (ch1==77)
            {
              xb += 15;
              if(xb>xmax-30) xb=xmax-30;
            }
          else
            {
              xb -= 15;
              if(xb<0) xb=0;
            }
         putimage(xb,yb,pbe,XOR_PUT);
    }
else
if(ch==0&&ch1==72&&ban==1)
  {
     ban=0;
     t_bd_dan=tg_giay_100();
     xd=xb+10;
     yd= ymax-10;
     putimage(xd,yd,pdan,XOR_PUT);
  }
}
       /* Su kien thoi gian cua tau */
t_ht_tau=tg_giay_100();
if(t_ht_tau-t_bd_tau> 1+random(CK_TAU))
   {
     t_bd_tau=t_ht_tau;
     tau_cd(&x,&y,&ch_tau);
   }
```

```
  /* Su kien thoi gian cua dan */
 if(ban==0)
  {
   t_ht_dan=tg_giay_100();
   if(t_ht_dan-t_bd_dan>1+random(CK_DAN))
    {
      sound(200);
      t_bd_dan=t_ht_dan;
      dan_cd(&xd,&yd,&ban);
      nosound();
      if (ban==0)
      kt_va_cham(x,y,&xd,&yd,&ban);
    }
  }
 }     /* ket thuc while */
 getch();
 closegraph();
}
```

§4. ĐIỀU KHIỂN CHUỘT

4.1. Toạ độ micki của chuột

Không phụ thuộc vào chế độ văn bản hay đồ hoạ, toạ độ micki của chuột như sau:

0 <= x_mouse <= 632

0 <= y_mouse <= 192

Để đổi sang toạ độ văn bản (80 cột 25 hàng) ta dùng công thức:

x_text = x_mouse/8 + 1

y_text = y_mouse/8 + 1

Để đổi sang toạ độ đồ hoạ có thể dùng công thức:

x_graph = (x_mouse*getmaxx())/632;

y_graph = (y_mouse*getmaxy())/192;

4.2. Ngắt 0x33 dùng để điều khiển chuột

Để điều khiển chuột ta sử dụng các chức năng của ngắt 0x33. Chú ý là số hiệu chức năng của ngắt 0x33 phải đặt vào thanh ghi AX chứ không phải đặt vào AH như các ngắt đã xét từ trước đến nay. Dưới đây là các chức năng điều khiển chuột của ngắt 0x33.

1. Chức năng 0: Khởi tạo chuột.

Đầu ra:

AX = 0 nếu chưa cài đặt chuột

AX = -1 nếu đã cài đặt chuột

BX = số núm (thường là 2)

2. Chức năng 1: Làm hiện con trỏ chuột.

3. Chức năng 2: Làm ẩn con trỏ chuột.

4. Chức năng 3: cho biết trạng thái núm và vị trí con trỏ.

Đầu ra:

BX cho trạng thái các núm:

Bit 0 = 0 nếu núm trái nhả

Bit 0 = 1 nếu núm trái ấn

Bit 1 = 0 nếu núm phải nhả

Bit 1 = 1 nếu núm phải ấn

CX = x_mouse: vị trí ngang hiện tại của con trỏ chuột

DX = y_mouse: vị trí dọc hiện tại của con trỏ chuột

5. Chức năng 4: Di chuyển con trỏ chuột.

Đầu vào:

CX = vị trí ngang mới của con trỏ chuột

DX = vị trí dọc mới của con trỏ chuột

6. Chức năng 5: Cho biết trạng thái và số lần ấn của một núm từ khi gọi chức năng này.

Đầu vào:

BX = núm cần hỏi (0 - trái, 1 - phải, 2 - giữa)

Đầu ra:

AX cho trạng thái của núm:

bit 0 cho trạng thái của núm trái

bit 1 cho trạng thái của núm phải

bit 2 cho trạng thái của núm giữa

BX = số lần ấn

CX = vị trí ngang (lần ấn cuối cùng)

DX = vị trí dọc (lần ấn cuối cùng)

7. Chức năng 6: Cho biết trạng thái và số lần nhả của một núm từ khi gọi chức năng này.

Đầu vào:

BX = núm cần hỏi (0 - trái, 1 - phải, 2 - giữa)

Đầu ra:

AX cho trạng thái của núm:

bit 0 cho trạng thái của núm trái

bit 1 cho trạng thái của núm phải

bit 2 cho trạng thái của núm giữa

BX = số lần nhả

CX = vị trí ngang (lần nhả cuối cùng)

DX = vị trí dọc (lần nhả cuối cùng)

8. ***Chức năng 7:*** quy định phạm vi di chuyển ngang của con trỏ chuột.

Đầu vào:

CX = cột trái

DX = cột phải

9. ***Chức năng 8:*** quy định phạm vi di chuyển dọc của con trỏ chuột.

Đầu vào:

CX = dòng trên

DX = dòng dưới

4.3. Ví dụ minh hoạ.

Trong chương trình tìm trẻ lạc ở §3, cho phép dùng bàn phím để di chuyển vị trí của người mẹ (M). Bây giờ ta thay bàn phím bằng chuột, có nghĩa là dùng chuột để điều khiển hành trình của M. Ta cũng đổi tên trò chơi là mèo đuổi chuột, nhưng bản chất cũng giống như trò chơi mẹ tìm con.

```
/*
Tro choi meo chuot
*/
    #include "stdio.h"
    #include "dos.h"
    #include "conio.h"
    #include "stdlib.h"
    #define INT_MOUSE 0x33 /* Ngat chuot */
    #define MOUSE_INIT 0x00 /* khoi tao chuot */
    #define MOUSE_SHOW 0x01 /* hien con tro chuot */
    #define MOUSE_GET_POSITION 0x03 /* nhan vi tri chuot */
    #define MOUSE_Y_LIMIT 0x08 /* Gioi han chuot theo y */
    #define INT_VIDEO 0x10 /* Ngat video */
```

```
#define CURSOR_SET 0x01 /* dat kich thuoc con tro */
#define CURSOR_GET 0x03 /* nhan biet vi tri con tro */
#define BELL 7 /* Chuong */
#define PERIOD 25 /* Thoi gian giua 2 chuyen dong chuot */
unsigned long int tg_giay_100(void);
main()
  {
     int mouse_status; /* Trang thai chuot */
     char ch;
     unsigned long int t_bd, t_ht;
     int cat_x,cat_y,mouse_x,mouse_y;
     int raw_mouse_x,raw_mouse_y; /*Vi tri chuot dang tho */
     int cursor_size;
     int old_mouse_x, old_mouse_y;
     /*Khoi tao chuot*/
     _AX=MOUSE_INIT;
     geninterrupt(INT_MOUSE);
     mouse_status=_AX;
     if (mouse_status!= -1)
       {
         fprintf(stderr,"Khong co CT DK chuot\n");
         exit(1);
       }
     _AX=MOUSE_Y_LIMIT; _CX=5*8; _DX=25*8;
     geninterrupt(INT_MOUSE);  clrscr();
     puts("Tro choi meo chuot - Bam Q de ket thuc");
     _AH=CURSOR_GET;
     geninterrupt(INT_VIDEO);
     cursor_size=_CX;
     _AH=CURSOR_SET;
     _CX=0x0f0f; /* Tat con chay */
     geninterrupt(INT_VIDEO);
     old_mouse_x=old_mouse_y=-1;
     cat_x=random(80)+1; cat_y=random(24)+1;
     gotoxy(cat_x,cat_y); putch('C');
     t_bd = tg_giay_100();
```

```
        /* Cho su kien xuat hien */
while(1)
   {  /* Su kien phim */
   if(kbhit())
     {
      ch=getch();
      if(toupper(ch)=='Q')   /* Bam Q ket thuc */
        break;
      else putch(BELL);
     }
       /*Su kien chuot */
      _AX=MOUSE_GET_POSITION;
      geninterrupt(INT_MOUSE);
      raw_mouse_x=_CX;
      raw_mouse_y=_DX;
              /* Sua vi tri chuot */
      mouse_x= raw_mouse_x/8 +1;
      mouse_y= raw_mouse_y/8 +1;
      if((mouse_x!=old_mouse_x) || (mouse_y!=old_mouse_y))
       {
                /*Chuyen dong chuot */
         if(old_mouse_x!=-1)
         {
          gotoxy(old_mouse_x,old_mouse_y);
          putch(' '); /*Xoa chuot*/
         }
        gotoxy(mouse_x,mouse_y); putch('M');
              /* Luu vi tri chuot */
        old_mouse_x=mouse_x;
        old_mouse_y=mouse_y;
       }
             /* Kiem tra da bat duoc chua ? */
if (mouse_x==cat_x && mouse_y==cat_y)
 {
  gotoxy(50,1);
  cputs("Bat duoc roi!"); break;
 }
```

```
        /* Su kien thoi gian */
        t_ht = tg_giay_100();
        if (t_ht - t_bd > PERIOD )
          {
             gotoxy(cat_x,cat_y); putch(' '); /*Xoa meo cu */
             cat_x = random(80) + 1;
             cat_y = random(21) + 5;
             gotoxy(cat_x,cat_y); putch('C');
             t_bd=t_ht;
          }
        } /* while */
        _AH=CURSOR_SET;
        _CX=cursor_size; /* Khoi phuc con chay */
        geninterrupt(INT_VIDEO);
    }
unsigned long int tg_giay_100(void)
    {
        static struct time ti;
        unsigned long int t;
        gettime(&ti);
                      /* Doi ra do vi thoi gian 1/100 giay */
        t=((ti.ti_hour*60+ti.ti_min)*60+ti.ti_sec)*100+ti.ti_hund;
        return t;
    }
```

BÀI TẬP CHƯƠNG 17

Bài 1. Hãy cải tiến chương trình 2, mục §3 để khi đạn chạm phải tầu thì tầu bị hất tung lên trên.

Bài 2. Hãy cải tiến chương trình 2, mục §3 để tầu bay lượn lúc cao lúc thấp trên bầu trời.

Bài 3. Lập chương trình để cho một chiếc xe chuyển động và tránh các vật cản trên đường đi của nó.

Bài 4. Hãy lập chương trình mô phỏng trò chơi đẩy bóng như sau. Bóng được ném vào gôn từ nhiều phía. Người chơi dùng chuột hay các phím di chuyển bàn đỡ để đẩy bóng ra ngoài gôn.

CHƯƠNG 18

GIAO DIỆN GIỮA C VÀ ASSEMBLER

Trong chương này sẽ trình bầy quy tắc xây dựng các hàm bằng assembler nhưng lại dùng trong chương trình C. Việc này phụ thuộc khá nhiều vào mô hình bộ nhớ dùng trong C. Các điều trình bầy dưới đây dùng cho mô hình large (xem phụ lục 10). Trước khi vào chương này, đọc giả phải xem qua chương 13 và phải có những hiểu biết nhất định về assembler. Để dịch các hàm assembler, đọc giả cần một thư mục chứa các tệp masm.exe, exe2bin.exe và link. exe (giả định đó là thư mục D:\ASM).

§1. XÂY DỰNG HÀM TRONG ASSEMBLER

1.1. Cấu trúc của hàm trong assembler

Trong assembler, hàm viết dưới dạng một thủ tục theo mẫu sau:

```
    public tên_hàm
tên_hàm proc far
    các câu lệnh
    ret
tên_hàm endp
```

Chú ý:

+ Tên hàm sẽ dùng để gọi trong chương trình C

+ Cần thêm dấu gạch dưới vào trước tên hàm. Ví dụ nếu trong C ta định dùng hàm nhap_sl thì trong assembler ta phải đặt tên hàm là _nhap_sl

1.2. Truyền tham số giữa hàm và chương trình C

Các tham số từ lời gọi của trình C được đưa vào ngăn xếp dưới dạng số liệu hoặc địa chỉ và được bố trí theo thứ tự sau (giả định BP đang trỏ đến đỉnh hiện thời của ngăn xếp):

Offset	Nội dung
BP + 0	Đỉnh ngăn xếp của chương trình gọi
BP + 2	Địa chỉ trở về (offset và đoạn)
BP + 6	Tham số thứ nhất
..	Tham số thứ hai
..	...

Ví dụ với lời gọi

int a,b; float x;

sub(b,&a,x);

thì các tham số được phân bố như sau:

offset	Nội dung
BP + 6	Giá trị của b (2 byte)
BP + 8	Địa chỉ của a (4 byte)
BP + 12	Giá trị của x (4 byte)

1.3. Tham số địa chỉ

Địa chỉ chiếm 4 byte trong đó 2 byte đầu là offset và 2 byte sau là đoạn. Chẳng hạn trong ví dụ trên, địa chỉ của biến a chiếm 4 byte từ BP + 8 đến BP + 11 thì offset nằm trong các byte BP + 8 và BP + 9 còn đoạn nằm trong các byte BP + 10 và BP + 11.

1.4. Cách lấy các tham số

Để lấy các tham số từ ngăn xếp ta dùng các thanh ghi và các lệnh của assembler. Dưới đây là đoạn chương trình lấy tham số từ lời gọi của ví dụ trên:

```
push    bp
mov     bp, sp           ; bp trỏ đến đỉnh ngăn xếp
mov     ax, [bp+6]       ; Đưa giá trị b vào thanh ghi ax
mov     bx, [bp+8]
mov     cx, [bp+10]      ; cx:bx là địa chỉ của biến a
```

1.5. Kết quả trả về của hàm

Khi kết thúc, hàm trả về giá trị của thanh ghi ax. Nếu không quan tâm đến giá trị này ta cũng nên gán số 0 vào ax để đảm bảo tính nhất quán của C.

Các ví dụ dưới đây sẽ minh hoạ hàm với các dạng tham số khác nhau: giá trị nguyên, địa chỉ nguyên, giá trị thực, địa chỉ thực, địa chỉ ký tự.

§2. SỬ DỤNG HÀM ASSEMBLER TRONG C

Chúng ta sẽ trình bầy cách dùng hàm assembler trong C thông qua một ví dụ gồm 4 phần sau:

+ Tệp h1.asm chứa hàm gan_sn viết trong assembler. Hàm này được mô tả trong C như sau:

int gan_sn(int *y,int x);

Hàm dùng để gán giá trị của x vào vùng nhớ do y trỏ tới. Hàm trả về giá trị x.

+ Cách dich tệp h1.asm để nhận được tệp h1.obj

+ Tệp ct1.c là một chương trình C dùng hàm gan_sn (hàm này viết bằng assembler)

+ Cách dịch và liên kết các tệp ct1.c , h1.obj để tạo thành tệp chương trình thực hiện ct1.exe

2.1. Nội dung tệp h1.asm

```
; int gan_sn(int *y,int x)
pgroup      group       prog
  prog      segment byte public 'CODE'
            assume cs:prog
            public _gan_sn
  _gan_sn   proc far
  push      bp          ; lưu bp vào ngăn xếp
  mov       bp,sp       ; bp trỏ tới đỉnh ngăn xếp
  mov       ax,[bp+10]  ; chuyển x vào ax
  mov       bx,[bp+6]
  mov       dx,[bp+8]   ; dx:bx chứa địa chỉ do y trỏ tới
  push      ds          ; lưu ds trên ngăn xếp
  mov       ds,dx       ; ds:bx chứa địa chỉ do y trỏ tới
  mov       [bx],ax     ; chuyển x vào địa chỉ do y trỏ tới
  pop       ds          ; khôi phục ds pop bp; khôi phục bp
  ret
  _gan_sn   endp
  prog      ends
            end
```

2.2. Dịch tệp h1.asm để nhận tệp h1,obj.

Dịch trong thư mục D:\ASM theo dòng lệnh sau:

D:\ASM> masm h1 /mx/zi;

2.3. Nội dung tệp ct1.c

```
/* Dùng hàm gan_sn trong tệp h1.obj */
    #include "stdio.h"
    #include "conio.h"
```

```
extern int gan_sn(int *, int);
main()
  {
    int x=100,y,z;
    y=gan_sn(&z,x);
    printf("\n y= %d z= %d",y,z);
    getch();
  }
```

4. Dịch và liên kết tệp ct1.c với h1.obj

Trình tự thao tác như sau:

+ Đưa các tệp trên vào thư mục của Turbo C
+ Vào môi trường Turbo C
+ Tạo tệp ct1.prj gồm 2 dòng sau:
  ct1.c
  h1.obj
+ Dùng chức năng Project Name của menu Project để chọn tệp ct1.prj
+ Dùng chức năng Compiler của menu Option để chọn mô hình bộ nhớ Large
+ Bấm F9

Kết quả sẽ nhận được tệp ct1.exe trong thư mục của Turbo C.

§3. THÊM MỘT VÍ DỤ VỀ THAM SỐ NGUYÊN

Trong §2 đã xét hàm gan_sn có 2 tham số là địa chỉ nguyên và giá trị nguyên.

Trong mục này sẽ đưa thêm một ví dụ về hàm có tham số nguyên là hàm xác định dạng con trỏ màn hình. Hàm có 2 đối nguyên là d_dau (dòng đầu) và d_cuoi (dòng cuối). Khi xây dựng hàm ta dùng chức năng 1 của ngắt 10 (xem mục §6, Chương 13)

3.1. Tệp h2.asm chứa hàm con_tro

```
        ; void con_tro(int d_dau, int d_cuoi)
pgroup  group prog
prog    segment byte public 'CODE'
        assume cs:prog
        public _con_tro
```

```
_con_tro    proc    far
            push    bp
            mov     bp,sp
            mov     ch,[bp+6]
            mov     cl,[bp+8]
            mov     ah,1
            int     10h
            pop     bp
            ret
_con_tro    endp
prog        ends
            end
```

3.2. Tệp ct2.c là chương trình C dùng hàm con_tro

```
/*
 ct2.c chương trình C dùng hàm con_tro
*/
  #include "stdio.h"
  void con_tro(int d_dau, int d_cuoi);
  main()
    {
      int d,c;
      printf("\n Dong dau, dong cuoi cua con tro: ");
      scanf("%d%d",&d,&c);
      con_tro(d,c);
      getch();
    }
```

§4. THAM SỐ LÀ GIÁ TRỊ THỰC VÀ ĐỊA CHỈ THỰC

Trong mục này sẽ xây dựng hàm

```
void gan_st(float *y,float x);
```

Hàm có 2 đối là địa chỉ thực và giá trị thực. Công dụng của hàm là gán giá trị của x vào địa chỉ do y trỏ tới. Hàm này minh hoạ cách lấy địa chỉ của biến thực và cách gán một số thực (chứa trong 4 byte) vào một địa chỉ trong bộ nhớ.

4.1. Tệp h3.asm chứa hàm gan_st

```
; void gan_st(float *y,float x)
pgroup      group prog
prog        segment byte public 'CODE'
            assume cs:prog
            public _gan_st
_gan_st     proc    far
            push    bp             ;lưu bp vào ngăn xếp
mov         bp,sp                  ;bp trỏ tới đỉnh ngăn xếp
mov         ax,[bp+10];
mov         cx,[bp+12]             ; chuyển x vào ax và cx
mov         bx,[bp+6]
mov         dx,[bp+8]              ; dx:bx chứa tham số địa chỉ y
push        ds                     ; lưu ds vào ngăn xếp
mov         ds,dx                  ; ds:bx chứa tham số địa chỉ y
mov         [bx],ax
mov         [bx+2],cx              ; gán x vào địa chỉ do y trỏ tới
pop         ds                     ; khôi phục ds
pop         bp                     ; khôi phục bp
mov         ax,0
ret
_gan_st     endp
prog        ends
            end
```

4.2. Tệp ct3.c là chương trình C dùng hàm gan_st

```
/* ct3.c: dùng hàm gan_st */
 #include "stdio.h"
 extern void gan_st(float *, float);
 main()
   {
      float x=100.55,z;
      gan_st(&z,x/2);
      printf("\n x= %f z= %f ",x,z); getch();
   }
```

§5. THAM SỐ LÀ ĐỊA CHỈ KÝ TỰ

Trong mục này sẽ xây dựng hàm

void hien_kt(char *st);

Hàm có đối là địa chỉ ký tự. Công dụng của hàm là hiện lên màn hình chuỗi ký tự st kết thúc bởi dấu dolar. Hàm minh hoạ cách lấy địa chỉ của chuỗi. Trong hàm có sử dụng chức năng 9 của ngắt 0x21 (xem mục §6, Chương 13)

5.1. Tệp h4.asm chứa hàm hien_kt

```
pgroup      group prog
prog        segment byte public 'CODE'
            assume cs:prog
            public _hien_kt
_hien_kt    proc  far
            push  bp
            mov   bp,sp
            mov   dx,[bp+6]
            mov   ax,[bp+8]
            push  ds
            mov   ds,ax
            mov   ah,9
            int   21h
            pop   ds
            pop   bp
            ret
_hien_kt    endp
prog        ends
            end
```

5.2. Tệp ct4.c là chương trình C dùng hàm hien_kt

```
#include "stdio.h"
#include "string.h"
void hien_kt(char *st);
main()
```

```
{
  float tt[3000];
  char x[100],y,*c;
  printf("\nHai phong\n");
  c="HA NOI $";
  strcpy(x,c);
  hien_kt(c);
  hien_kt ("\n\015NAM HA\015\n$"); /* \n\015 - xuống dòng */
  hien_kt(x);
  getch();
}
```

§6. THƯ VIỆN CÁC HÀM TRONG ASSEMBLER

Trong mục này sẽ trình bầy cách xây dựng một tệp assembler (ta gọi là tệp thu_vien.asm) gồm các hàm sau:

gan_sn (xem §2)

con_tro (xem §3)

gan_st (xem §4)

hien_kt (xem §5)

ch_dong Hàm không đối có tác dụng chuyển con trỏ xuống đầu dòng tiếp theo

(dùng chức năng 2 ngắt 0x21)

chao_ban Hàm không đối đưa ra màn hình 2 dòng chữ

Turbo C

chao ban

(dùng chức năng 9 ngắt 0x21)

Nội dung trình bầy gồm 3 vấn đề:

+ Tệp thu_vien.asm

+ Cách dịch tệp thu_vien.asm để được tệp thu_vien.obj

+ Cách dùng các hàm trong unit thu_vien.obj

6.1. Tệp thu_vien.asm

```
; Tệp thu_vien.asm
pgroup group prog; bắt đầu một unit
```

```
; Đoạn dữ liệu
      data          segment
      tb1           db 'Turbo C',13,10,'$'
      tb2           db 'chao ban',13,10,'$'
      data          ends; kết thúc đoạn dữ liệu
; Đoạn mã (chứa các thủ tục)
prog          segment byte public 'CODE'
              assume cs:prog
; hàm gan_sn
              public  _gan_sn
_gan_sn       proc    far
              push    bp              ; lưu bp vào ngăn xếp
              mov     bp,sp           ; bp trỏ tới đỉnh ngăn xếp
              mov     ax,[bp+10]      ; chuyển x vào ax
              mov     bx,[bp+6]
              mov     dx,[bp+8]       ; dx:bx chứa địa chỉ do y trỏ tới
              push    ds              ; lưu ds trên ngăn xếp
              mov     ds,dx           ; ds:bx chứa địa chỉ do y trỏ tới
              mov     [bx],ax         ; chuyển x vào địa chỉ do y trỏ tới
              pop     ds              ; khôi phục ds
              pop     bp              ;khôi phục bp
              ret
_gan_sn       endp
; hàm con_tro
              public  _con_tro
_con_tro      proc    far
              push    bp
              mov     bp,sp
              mov     ch,[bp+6]
              mov     cl,[bp+8]
              mov     ah,1
              int     10h
```

```
                pop     bp
                ret
_con_tro        endp
; hàm gan_st
                public  _gan_st
_gan_st         proc    far
                push    bp              ;lưu bp vào ngăn xếp
                mov     bp,sp           ;bp trỏ tới đỉnh ngăn xếp
                mov     ax,[bp+10];
                mov     cx,[bp+12]      ; chuyển x vào ax và cx
                mov     bx,[bp+6]
                mov     dx,[bp+8]       ; dx:bx chứa tham số địa chỉ y
                push    ds              ; lưu ds vào ngăn xếp
                mov     ds,dx           ; ds:bx chứa tham số địa chỉ y
                mov     [bx],ax
                mov     [bx+2],cx       ; gán x vào địa chỉ do y trỏ tới
                pop     ds              ; khôi phục ds
                pop     bp              ; khôi phục bp
                mov     ax,0
                ret
_gan_st         endp
; hàm hien_kt
                public  _hien_kt
_hien_kt        proc    far
                push    bp
                mov     bp,sp
                mov     dx,[bp+6]
                mov     ax,[bp+8]
                push    ds
                mov     ds,ax
                mov     ah,9
                int     21h
                pop     ds
```

```
        pop     bp
        ret
_hien_kt        endp
; hàm ch_dong
        public  _ch_dong
_ch_dong        proc    far
        mov     ah,2
        mov     dl,13
        int     21h
        mov     dl,10
        int     21h
        ret
_ch_dong        endp
; hàm chao_ban
        public  _chao_ban
_chao_ban       proc    far
        push    ds
        mov     ax,data
        mov     ds,ax
        mov     dx,offset tb1
        mov     ah,9
        int     21h
        mov     dx,offset tb2
        int     21h
        pop     ds
        ret
_chao_ban       endp
prog    ends    ; kết thúc đoạn mã (chứa các thủ tục)
        end     ; kết thúc unit
```

6.2. Dịch tệp thu_vien.asm để được tệp thu_vien.obj

Có thể dịch tệp thu_vien.asm theo cách dịch tệp h1.asm trình bầy trong §2 (điểm 2) như sau:

D:\ASM> masm thu_vien /mx/zi;

6.3. Sử dụng các hàm trong tệp thu_vien.obj

Trong chương trình C có thể dùng các hàm của unit thu_vien.obj. Để dịch một tệp chương trình C như vậy (gọi là tệp ct.c) ta có thể làm theo cách đã trình bầy trong §2 (điểm 4) như sau:

+ Đưa các tệp ct.c và thu_vien.obj vào thư mục của Turbo C

+ Vào môi trường Turbo C

+ Tạo tệp ct.prj gồm 2 dòng sau:

    ct.c

    thu_vien.obj

+ Dùng chức năng Project Name của menu Project để chọn tệp ct.prj

+ Dùng chức năng Compiler của menu Option để chọn mô hình bộ nhớ Large

+ Bấm F9

Kết quả sẽ nhận được tệp ct.exe trong thư mục của Turbo C.

BÀI TẬP CHƯƠNG 18

Bài 1. Việc tổ chức giao diện giữa C và assembler có ích lợi gì?

Bài 2. Viết một số hàm đơn giản bằng hợp ngữ (assembler) và nêu cách dùng chúng trong chương trình C.

PHỤ LỤC 1

QUI TẮC XUỐNG DÒNG VÀ SỬ DỤNG CÁC KHOẢNG TRỐNG KHI VIẾT CHƯƠNG TRÌNH

Một câu hỏi thường đặt ra khi viết chương trình là:

- Khi nào thì được phép xuống dòng
- Khi nào thì hai kí tự phải viết liền nhau, khi nào có quyền đặt dấu cách giữa chúng.

Chẳng hạn dòng

```
y = a + b - c;
```

có thể viết trên ba dòng theo cách

```
y = a
        + b -c
              ;
```

hay không ? Hay biểu thức sin(x) có bắt buộc các kí tự phải viết liền nhau không? Có quyền viết một cách thoải mái như

```
sin ( x )
```

hoặc

```
sin (
     x
       )
```

hay không?

Mục đích của phụ lục 1 là giải đáp các câu hỏi nêu trên. Trước hết ta cần hiểu thế nào là một từ.

+ **Định nghĩa.** Từ là một dãy kí tự viết liền nhau. Các kí tự trong một từ cần các khoảng trống (trừ ra đối với hằng xâu kí tự).

Ví dụ trên dòng

* +,+= ([.; main alfa **:

có 11 từ.

+ **Các loại từ.** Khi viết chương trình ta sử dụng các loại từ dưới đây;

1/ Các hằng. Ví dụ:

"a" -145.2 27 "name"

Chú ý: Trong hằng xâu ký tự cho phép sử dụng các dấu cách

Ví dụ:

" Pham thu Huong"

2/ Các tên (tên hằng, tên biến, tên mảng, ...). Ví dụ

ax by luong hoan_vi sum

3/ Các từ khoá . Ví dụ

goto if switch

4/ Các dấu phép tính (xem *Chương 3*). Ví dụ

( ] . & ++ % == ?: * =

Chú ý: Đối với các phép tính ++ += -= ==!= *= ... hai ký tự phải viết liền nhau. Chẳng hạn khi gặp câu lệnh

s + = a;

máy sẽ thông báo sai (vì các ký tự + và = không viết liền nhau)

5/ Một số dấu chức năng khác như

; } {:

+ Qui tắc sử dụng các khoảng trắng: Giữa các từ có thể đặt một số bất kỳ các khoảng trắng. Cũng như trên, ở đây ta hiểu khoảng trắng là dấu cách hoặc dấu xuống dòng \n.

Qui tắc này cho phép ta viết các dòng chương trình hết sức thoải mái. Chẳng hạn dòng chương trình

```
for ( i=0;i<2;++i )
```

có thể viết trên nhiều dòng như sau:

```
for ( i=0;i<
        2; ++
                i )
```

Chú ý 1. Không được bẻ gãy một từ trên nhiều dòng. Chẳng hạn máy sẽ không chấp nhận đoạn chương trình:

```
dou
ble x, y;
printf ("\n x=%8.2f
y=%8.2f",x,y );
```

vì từ khoá double và hằng xâu ký tự "\n x=%8.2f y=%8.2f" đã bị bẻ gãy trên hai dòng.

Chú ý 2. Riêng đối với #include và #define, qui tắc nói trên không hoàn toàn đúng. Giữa dấu # từ khoá include và tên tệp có thể đặt một số bất kỳ các khoảng trống nhưng ba từ nói trên phải đặt trên một dòng. Đối với #define cũng như vậy (xem Chương 12).

+ Dưới đây là 3 ví dụ minh hoạ các qui tắc nêu trên. Mỗi ví dụ sẽ cho hai bản khác nhau của cùng một chương trình. Bản thứ nhất thể hiện một cách viết tương đối chuẩn mực. Bản thứ hai trình bày một cách viết chương trình hết sức tự do thoải mái, xuống dòng và sử dụng các khoảng trống một các tuỳ hứng.

Ví dụ 1, bản 1.

```
/*
Vao so lieu cho cac thanh phan cua mang cau truc
Ban 1 (cach viet chuan muc)
*/
#include"stdio.h"
main()
  {
    struct date
      {
        int ng;
        char th[10];
        int nam;
      };
    struct
      {
        struct date ngsinh;
        struct date ngthue;
        float luong;
      } p[100];
    int ngs,ngt,i,nams,nam;
    float ll;
    printf("\nvao sl 2 nguoi\n");
    for (i=0;i<2;++i);
      {
               scanf("%d%s%d%d%s%d%f",&ngs,p[i].ngsinh.t
           h, &nams, &ngt, p[i].ngthue.th,&namt,&ll);
        p[i].ngsinh.ng=ngs;
        p[i].ngsinh.nam=nams;
        p[i].ngthue.ng=ngt;
```

```
p[i].ngthue.nam=namt;
p[i].luong=ll;
}
for(i=0;i<2;++i);
    fprintf(stdprn,"\n%6d%10s%6d\n%6d%10s%6d\n%8.2f",
    p[i].ngsinh.ng,p[i].ngsinh.th,p[i].ngsinh.nam,[i].ngthue.ng,
    p[i].ngthue.th, p[i].ngthue.nam, p[i]. luong);
}
```

Ví dụ 1, bản 2.

```
/*
Vao so lieu cho các thanh phan cua mang cau truc
Ban hai (xuong dong tuy hung)
*/
#include "stdio.h"
main
  (
   )
{struct date
{ int ng;
char th
[
10
];
int nam
;
};
struct
{ struct date ngsinh
;
struct
date
ngthue
;
float luong;
```

```
} p [
100
] ;
int ngs,ngt,i
,nams,namt
float ll;
printf("\n vao sl 2 nguoi \n");
for
   {
   i=0;i
   <2;++i
   ){
   scanf("%d%s%d%d%s%d%f",&
   ngs,p[
   i] .ngsinh .th,&nams,&
   ngt,p
   [ i ]
   .ngthue.th,&namt,&ll
   ) ;
   p[i]
   .ngsinh.ng
   =
   ngs; p[i].ngsinh.nam=nams
   ;p[i].ngthue.ng=
   ngt;p[i] .ngthue.nam=namt;
   p[i].luong=ll;}
   for (i=0;i<
   2;++
   i)
   fprintf(stdprn,"\n%6d%10s%6d\n%6d%10s%6d\n%8.2f",
   p[i].ngsinh.ng,p[i].ngsinh.th,p[i].ngsinh.nam,
   p[i].ngthue.ng,p[i].ngthue.th,p[i].ngthue.nam,p[i].luong);
}
```

Ví dụ 2, bản 1.

```
/*
Chuong trinh tinh x luy thua y
Ban 1 (cach viet chuan muc)
*/
#include "stdio.h"
#include "math.h"
main()
  {
    double x,y,z; /* khai bao ba bien kieu double */
    printf("\n vao x va y");
    scanf("%lf%lf",&x,&y); /* vao x,y tu ban phim */
    z=pow(x,y); /* tinh x luy thua y va gan cho z */
    /* in ket qua tren ba dong */
    fprintf(stdprn,"\nx=%8.2f\ny=%8.2f\nz=%8.2f",x,y,z);
  }
```

Ví dụ 2, bản 2.

```
/*
Chuong trinh x luy thua y
ban 2 (xuong dong tuy hung)
*/
#include "stdio.h"
#include "math.h"
main
(
)
  {
  double
  x,
  y,z; /* khai bao ba bien kieu double */
  printf
  ("\n vao x va y"
  );
```

```
scanf("%lf%lf",&
x,&y);/* vao x,y tu ban phim */
z=pow(x
,y); /* tinh x luy thua y va gan cho z */
/* in ket qua tren ba dong */
fprintf(stdprn,
,"\nx=%8.2f\ny=%8.2f\nz=%8.2f",x,y,z);
}
```

Ví dụ 3, bản 1.

```
/*
  Chuong trinh tinh tong gia tri tuyet doi cac phan
  tu cua mang a va tinh max cac phan tu cua mang a
  Ban 1 (cach viet chuan muc)
*/
#include "stdio.h"
float a[]= {3,-4,15,16 };
main()
  {
    int i=0;
    float r,max=0,sum=0;
    for(;;++i)
      {
        r=fabs((double)a[i]);
        sum +=r;
        max=(max>a[i])?max:a[i];
        if(i==3) goto kt;
      }
    kt: fprintf(stdprn,"\nsum=%6.2f max=%6.2f",sum,max);
  }
```

Ví dụ 3, bản 2.

```
/*
  Chuong trinh tinh tong gia tri tuyet doi cac phan
  tu cua mang a va tinh max cac phan tu cua mang a
  Ban 2 (xuong dong tuy hung)
*/
  #include "stdio.h"
```

```
float a[
]
= { 3
,-4,15,6
} ;
main()
   {
   int i = 0; float r,max=0,sum=0;
   for (;;++
   i)
   { r=fabs((double
   )
   a[i]);
   sum
   +=
   r ;
   max=(max>
   a[i]
   )?
   max
   :
   a[i]
   ;
   if(i
   ==
   3
   ) goto
   kt ;} kt
   :
   fprintf(stdprn,"\nsum=%6.2f max=%6.2f",
   sum,max
   )
```

PHỤ LỤC 2

TÓM TẮT CÁC HÀM CHUẨN CỦA TURBO C

§1. PHÂN LOẠI HÀM

Để tiện việc tra cứu ta chia các hàm chuẩn của Turbo C thành các nhóm sau:

1. Các hàm vào ra trên bàn phím, màn hình và máy in
2. Các hàm xử lý tệp
3. Các hàm quản lý màn hình văn bản
4. Các hàm đồ hoạ
5. Các hàm xử lý ngắt
6. Các hàm tạo âm thanh và tạm dừng máy
7. Các hàm chuyển đổi dữ liệu
8. Các hàm kiểm tra ký tự
9. Các hàm xử lý chuỗi
10. Các hàm thao tác bộ đệm
11. Các hàm toán học
12. Các hàm thời gian
13. Các hàm cấp phát bộ nhớ
14. Các hàm kiểm soát thư mục
15. Các hàm kiểm soát quá trình

Các hàm nhóm 1 đã trình bầy trong chương 4

Các hàm nhóm 2 đã trình bầy trong chương 10

Các hàm nhóm 3 đã trình bầy trong chương 8

Các hàm nhóm 4 đã trình bầy trong chương 9

Các hàm nhóm 5 đã trình bầy trong các chương 13 và 15

Các hàm nhóm 6 đã trình bầy trong chương 16

Dưới đây sẽ trình bầy các hàm còn lại.

Để việc mô tả các hàm được gọn nhẹ ta sẽ sử dụng các thuật ngữ sau:

+ Một vùng nhớ do con trỏ ps trỏ tới (địa chỉ đầu) gọi là vùng nhớ ps hay vùng ps.

+ Chuỗi là một dẫy ký tự kết thúc bởi '\0'

+ Một chuỗi chứa trong vùng ps gọi là chuỗi ps

§2. CÁC HÀM CHUYỂN ĐỔI DỮ LIỆU

1. Chuyển đổi tổng quát

```
#include <stdio.h>
int sprintf(char *s, char *dk, danh sách đối);
```

Công dụng: Hàm làm việc giống như hàm quen biết printf, nhưng thay việc đưa ra màn hình bằng việc đưa chuỗi kết quả ra vùng nhớ s. Có thể dùng hàm này để thực hiện rất nhiều phép chuyển đổi như:

+ Đổi từ giá trị số (các kiểu) ra ký tự,

+ Tạo một chuỗi mới bằng cách ghép nhiều chuỗi đã biết,

+ Tạo một chuỗi mới từ các chuỗi và các số.

Hàm rất hữu ích khi cần in các thông báo đồ hoạ dưới dạng chuỗi.

Ví dụ để ghép họ, đệm và tên thành một chuỗi (ht) ta làm như sau:

```
#include <stdio.h>
main()
  {
    char ho[10], dem[10], ten[10];
    char ht[30];
    /* Ghép họ, đệm và tên */
    sprintf(ht,"%s %s %s",ho,dem,ten);
    ...
  }
```

Các hàm sau khai báo trong ctype.h

2. tolower

```
int tolower(int c);
```

Đổi từ chữ hoa sang chữ thường

3. toupper

```
int toupper(int c);
```

Đổi từ chữ thường sang chữ hoa

Các hàm dưới đây khai báo trong stdlib.h

4. atof

```
double atof(char *s);
```

Chuyển chuỗi s sang giá trị double.

5. atoi

```
int atoi(char *s);
```

Chuyển chuỗi s sang giá trị int.

6. atol

long atol(char *s);

Chuyển chuỗi s sang giá trị long.

7. itoa

char *itoa(int x, char *s, int cs);

Chuyển số nguyên x trọng hệ đếm cơ số cs (cs=10: hệ 10, cs=8: hệ 8, cs=16: hệ 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng s.

8. ltoa

char *ltoa(long x, char *s, int cs);

Chuyển số kiểu long x trọng hệ đếm cơ số cs (cs=10: hệ 10, cs=8: hệ 8, cs=16: hệ 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng s.

9. ultoa

char *ultoa(unsigned long x, char *s, int cs);

Chuyển số kiểu unsigned long x trọng hệ đếm cơ số cs (cs=10: hệ 10, cs=8: hệ 8, cs=16: hệ 16) sang chuỗi và lưu vào vùng nhớ s. Hàm trả về địa chỉ của vùng s.

10. ecvt chuyển double sang chuỗi chỉ gồm các chữ số.

char *ecvt(double x, int n, int *vt_cham, int *dau);

Vào: + x là giá trị double cần chuyển

+ n là số chữ số cần lưu giữ

Ra: + hàm trả về chuỗi kết quả gồm n chữ số (không có dấu - để biểu thị số âm, không có dấu chấm ngăn cách phần nguyên và phân).

+ vị trí của dấu chấm thập phân được gửi vào biến do vt_cham trỏ tới.

+ biến do con trỏ dau trỏ tới sẽ nhận 0 nếu x dương và nhận 1 nếu x âm.

(Xem ví dụ dưới đây).

11. fcvt chuyển double sang chuỗi chỉ gồm các chữ số.

char *fcvt(double x, int n, int *vt_cham, int *dau);

Vào: + x là giá trị double cần chuyển

+ n là số chữ số sau dấu chấm thập phân cần lưu giữ

Ra: + hàm trả về chuỗi kết quả gồm các chữ số (không có dấu - để biểu thị số âm, không có dấu chấm ngăn cách phần nguyên và phân). Chuỗi phải cho đầy đủ thông tin để xác định được n chữ số sau dấu chấm thập phân.

+ vị trí của dấu chấm thập phân được gửi vào biến do vt_cham trỏ tới.

+ biến do con trỏ dau trỏ tới sẽ nhận 0 nếu x dương và nhận 1 nếu x âm.

(Xem ví dụ dưới đây).

12. gcvt chuyển double sang chuỗi có cả dấu chấm thập phân và dấu - cho số âm.

```
char *gcvt(double x, int n, char *s);
```

Vào: + x là giá trị double cần chuyển

+ n là số chữ số có nghĩa cần lưu giữ

Ra: + chuỗi kết quả (ở dạng thập phân hoặc mũ) chứa trong vùng s.

+ hàm trả về địa chỉ vùng s.

(Xem ví dụ dưới đây).

Chương trình sau minh hoạ 3 hàm cuối.

```
#include "stdlib.h"
#include "stdio.h"
main()
  {
    char *p,s[20]; double x;
    int cham, dau;
    tt:
    printf("\n Nhap so thuc (0 - ket thuc): ");
    scanf("%lf",&x);
    if(x!=0.0)
     {
      p=ecvt(x,6,&cham,&dau);
      printf("\nSo: %s cham: %d dau: %d",p,cham,dau);
      p=fcvt(x,2,&cham,&dau);
      printf("\nSo: %s cham: %d dau: %d",p,cham,dau);
      gcvt(x,4,s);
      printf("\nSo: %s",s);
      getch(); goto tt;
     }
  }
```

Chương trình cho kết quả sau:

+ Nếu nhập x = -78.456789 thì

```
So: 784568    cham: 2    dau: 1
So: 7846      cham: 2    dau: 1
So: -78.46
```

+ Nếu nhập x = -0.0456789 thì

```
So: 456789    cham: -1   dau: 1
```

So: 5 cham: -1 dau: 1
So: -0.04568

+ Nếu nhập x = 345.6789 thì

So: 345679 cham: 3 dau: 0
So: 34568 cham: 3 dau: 0
So: 345.7

+ Nếu nhập x = 100000000 thì

So: 100000 cham: 9 dau: 0
So: 10000000000 cham: 9 dau: 0
So: 1e+08

§3. CÁC HÀM KIỂM TRA KÝ TỰ

Các hàm dưới đây (khai báo trong ctype.h) dùng để kiểm tra một ký tự có thuộc một tập ký tự nào đó không? Nếu thuộc thì hàm trả về giá trị khác không, trái lại hàm trả về 0.

1. isalnum Kiểm tra kt có phải là ký tự alphanumeric (chữ cái hoặc chữ số)?

int isalnum(int kt);

2. isalpha Kiểm tra kt có phải là chữ cái?

int isalpha(int kt);

3. iscntrl Kiểm tra kt có phải là ký tự điều khiển (mã từ 0 đến 0x1F hoặc mã bằng 0x7F (DEL))?

int iscntrl(int kt);

4. isdigit Kiểm tra kt có phải chữ số?

int isdigit(int kt);

5. isgraph Kiểm tra kt có phải là ký tự in được và khác ký tự trống (mã từ 0x21 đến 0x7E)?

int isgraph(int kt);

6. islower Kiểm tra kt có phải là chữ cái thường (từ a đến z)?

int islower(int kt);

7. isprintf Kiểm tra kt có phải là ký tự in được kể cả ký tự trống (mã từ 0x20 đến 0x7E)?

int isprint(int kt);

8. ispunct Kiểm tra kt có phải là ký tự dấu chấm câu (ký tự in được khác alphanumeric và khác trống)?

int ispunct(int kt);

9. isspace Kiểm tra kt có phải là ký tự trống?

int isspace(int kt);

10. isupper Kiểm tra kt có phải là chữ hoa (từ A đến Z)?

int isupper(int kt);

11. isxdigit Kiểm tra kt có phải là chữ số hệ 16 (gồm chữ số, các chữ từ a đến f, các chữ từ A đến F)?

int isxdigit(int kt);

§4. CÁC HÀM XỬ LÝ CHUỖI KÝ TỰ

Các hàm này khai báo trong string.h

1. strcat Ghép chuỗi

char *strcat(char *s_nhan, char *s);

Công dụng: bổ sung chuỗi s vào sau chuỗi s_nhan.

2. strchr Tìm lần xuất hiện đầu tiên của ký tự trong chuỗi

char *strchr(char *s, int kt);

Công dụng: Tìm lần xuất hiện đầu tiên của kt trong s. Nếu tìm thấy hàm cho địa chỉ của ký tự tìm được trong s, trái lại hàm cho NULL.

3. strrchr Tìm lần xuất hiện cuối cùng của ký tự trong chuỗi

char *strrchr(char *s, int kt);

Công dụng: Tìm lần xuất hiện cuối cùng của kt trong s. Nếu tìm thấy hàm cho địa chỉ của ký tự tìm được trong s, trái lại hàm cho NULL.

4. strcmp So sánh 2 chuỗi

int strcmp(char *s1, char *s2)

Hàm cho:

giá trị âm nếu chuỗi s1 nhỏ hơn s2,

giá trị 0 nếu chuỗi s1 bằng s2,

giá trị dương nếu chuỗi s1 lớn hơn s2,

5. strcmpi So sánh 2 chuỗi không phân biệt chữ hoa chữ thường

int strcmpi(char *s1, char *s2)

Hàm làm việc như strcmp nhưng không phân biệt chữ hoa với chữ thường khi so sánh.

6. strcpy Sao chuỗi

char strcpy(char *s_nhan, char *s_gui);

Hàm sao chuỗi s_gui vào vùng s_nhan.

7. strcspn Tìm độ dài đoạn đầu của chuỗi

int strcspn(char *s, char *s_con);

Hàm cho độ dài đoạn đầu (lớn nhất có thể) của s, mà mọi ký tự của đoạn không có mặt trong chuỗi s_con.

8. strspn Tìm độ dài đoạn đầu của chuỗi

int strspn(char *s, char *s_con);

Hàm cho độ dài đoạn đầu (lớn nhất có thể) của s, mà mọi ký tự của đoạn đều có mặt trong chuỗi s_con.

9. strdup Gấp đôi một chuỗi

char *strdup(char *s);

Công dụng: Hàm cấp một vùng nhớ để đặt 2 lần chuỗi s. Nếu thành công hàm cho địa chỉ vùng này, trái lại hàm cho NULL.

10. stricmp So sánh 2 chuỗi không phân biệt chữ hoa và thường

int stricmp(char *s1, char *s2);

Hàm này làm việc giống như strcmpi

11. strlen Cho độ dài của chuỗi

int strlen(char *s);

12. strlwr Chuyển chữ hoa thành chữ thường

char *strlwr(char *s);

Công dụng: Mọi chữ hoa trong s được chuyển thành chữ thường.

13. strncat Ghép thêm n ký tự vào chuỗi

char *strncat(char *s_nhan, char *s, int n);

Công dụng: Ghép n ký tự đầu tiên của s vào s_nhan

14. strncmp So sánh n ký tự đầu của 2 chuỗi

int strncmp(char *s1, char *s2, int n);

Hàm tiến hành so sánh n ký tự đầu tiên của s1 và s2.

15. strnicmp So sánh n ký tự đầu của 2 chuỗi không phân biệt chữ hoa chữ thường

int strnicmp(char *s1, char *s2, int n);

16. strncpy Sao n ký tự

char *strncpy(char *s_nhan, char *s_gui, int n);

Hàm sao n ký tự đầu của s_gui sang s_nhan.

17. strnset Gán n lần một ký tự cho một chuỗi.

char *strnset(char *s, int c, int n);

Công dụng: Ký tự c sẽ được gán n lần cho s. Kết quả s là một chuỗi gồm n ký tự đều là c.

18. strpbrk Tìm sự xuất hiện đầu tiên

char *strpbrk(char *s1, char *s2);

Công dụng: Tìm sự xuất hiện đầu tiên của một ký tự của s2 trong s1. Nếu có xuất hiện, hàm cho địa chỉ của ký tự tìm thấy trong s1. Trái lại hàm cho NULL.

19. strrev Đảo ngược chuỗi ký tự

char *strrev(char *s);

Công dụng: Đảo ngược các ký tự trong chuỗi s. Hàm cho địa chỉ chuỗi đã đảo.

20. strset Đặt một ký tự vào mọi vị trí của chuỗi

char *strset(char *s, int kt);

Công dụng: Thay mọi ký tự trong s bằng kt.

21. strstr Tìm sự xuất hiện đầu tiên

char *strstr(char *s, char *s_con);

Công dụng: Tìm sự xuất hiện đầu tiên của s_con trong s. Nếu thấy, hàm cho địa chỉ chuỗi con (trùng với s_con) trong s. Trái lại hàm cho NULL.

22. strupr Chuyển chữ thường thành chữ hoa

char *strupr(char *s);

Công dụng: Mọi chữ thường trong s được chuyển thành chữ hoa.

§5. CÁC HÀM THAO TÁC BỘ ĐỆM

Các hàm này khai báo trong memory.h hoặc string.h.

1. memccpy

char *memccpy(char *nhan,char *gui,int kt,unsigned n);

Sao các ký tự từ vùng gui đến vùng nhan. Việc sao kết thúc khi gặp ký tự kt hoặc đã sao đủ n ký tự.

Nếu kt được sao thì hàm trả về địa chỉ của byte đứng ngay sau kt trong vùng gui, trái lại hàm trả về NULL.

2. memchr

char *memchr(char *s, int kt, unsigned n);

Tìm ký tự kt trong n byte đầu của vùng s. Nếu tìm thấy hàm trả về địa chỉ của byte chứa ký tự kt đầu tiên trong s, trái lại hàm trả về NULL

3. memcmp

int memcmp(char *s1, char *s2, unsigned n);

So sánh n byte trong các vùng s1 và s2 và cho kết quả sau:

giá trị âm nếu s1 nhỏ hơn s2
giá trị 0 nếu s1 trùng với s2
giá trị dương nếu s1 lớn hơn s2

4. memicmp

int memicmp(char *s1, char *s2, unsigned n);

Hàm này thực hiện giống như hàm memcmp nhưng không phân biệt chữ hoa và chữ thường trong khi so sánh.

5. memcpy

char *memcpy(char *nhan, char *gui, unsigned n);

Sao n ký tự từ vùng gui sang vùng nhan. Hàm cho lại địa chỉ vùng nhan.

6. memset

char *memset(char *nhan, int kt, unsigned n);

Hàm gửi ký tự kt vào n byte đầu của vùng nhan. Hàm cho lại địa chỉ vùng nhan.

7. movedata

void movedata(unsigned seg_g, unsigned off_g,
unsigned seg_n,unsigned off_n,unsigned n);

Sao n byte từ địa chỉ phân đoạn seg_g:off_g đến địa chỉ seg_n:off_n

§6. CÁC HÀM TOÁN HỌC

Các hàm sau khai báo trong stdlib.h

1. abs Tính giá trị tuyệt đối của số nguyên x

int abs(int x);

2. labs Tính giá trị tuyệt đối của số nguyên dài x

long int labs(long int x);

3. rand Cho một giá trị ngẫu nhiên từ 0 đến 32767

int rand(void);

4. random Cho một giá trị ngẫu nhiên từ 0 đến n-1

int random(int n);

5. srand Khởi đầu bộ số ngẫu nhiên bằng giá trị seed

void srand(unsigned seed);

6. randomize Khởi đầu bộ số ngẫu nhiên bằng giá trị ngẫu nhiên

void randomize(void);

Chú ý: Hàm này dùng hàm time trong <time.h>

Các hàm sau khai báo trong math.h

7. acos Tính arc cosine của x

double acos(double x);

8. asin Tính arc sine của x

double asin(double x);

9. atan Tính arc tangent của x

double atan(double x);

10. atan2 Tính arc tangent của y/x

double atan2(double y, double x);

11. cabs Tính giá trị tuyệt đối (mô đun) của số phức z

double cabs(struct complex z);

Cấu trúc complex đã định nghĩa trong math.h như sau:

```
struct complex
  {
   double x,y;
  };
```

12. ceil Xác định số nguyên (kiểu double) bé nhất trên x

double ceil(double x);

13. cos Tính cosine của x double cos(double x);

14. cosh Tính cosine hyperbolic của x

double cosh(double x);

15. exp Tính e mũ x

double exp(double x);

16. fabs Tính giá trị tuyệt đối của x

double fabs(double x);

17. floor Tìm số nguyên (kiểu double) lớn nhất dưới x

double floor(double x);

18. fmode Tìm phần dư dấu phẩy động của y/x

double fmode(double y, double x);

19. log Tính logarit tự nhiên của x

double log(double x);

20. log10 Tính logarit sơ số 10 của x

double log10(double x);

21. pow Tính y mũ x

double pow(double y, double x);

22. sin Tính sine của x

double sin(double x);

23. sinh Tính sine hyperbolic của x

double sinh(double x);

24. sqrt Tính căn bậc hai của x

```
double sqrt(double x);
```

25. tan Tính tangent của x

```
double tan(double x);
```

26. tanh Tính tangent hyperbolic của x

```
double tanh(double x);
```

§7. CÁC HÀM THỜI GIAN

Các hàm này khai báo trong time.h. Các cấu trúc liên quan khai báo trong dos.h

1. gettime Nhận giờ hệ thống

```
void gettime(struct time *t);
```

Kiểu cấu trúc time định nghĩa trong dos.h như sau

```
struct time
  {
  unsigned ti_hour; /* Giờ */
  unsigned ti_min;  /* Phút */
  unsigned ti_sec;        /* Giây */
  unsigned ti_hund;       /* Miligiây */
  };
```

Công dụng: Hàm nhận giờ hệ thống và đặt vào các thành phần của một biến cấu trúc kiểu time do con trỏ t trỏ tới.

2. settime Thiết lập giờ hệ thống

```
void settime(struct time *t);
```

Công dụng: Hàm đặt lại giờ hệ thống theo giá trị các thành phần của một biến cấu trúc kiểu time do con trỏ t trỏ tới.

3. getdate Nhận ngày hệ thống

```
void getdate(struct date *d);
```

Kiểu cấu trúc date được định nghĩa trong dos.h như sau

```
struct date
  {
    int da_year;          /* Năm */
    char da_mon;  * Tháng */
    char da_day;          /* Ngày */
  };
```

Công dụng: Hàm nhận ngày hệ thống và đặt vào các thành phần của một biến cấu trúc kiểu date do con trỏ d trỏ tới.

4. setdate Thiết lập ngày hệ thống

void setdate(struct date *d);

Công dụng: Hàm đặt lại ngày hệ thống theo giá trị các thành phần của một biến cấu trúc kiểu date do con trỏ d trỏ tới.

5. time Nhận thời gian hệ thống quy ra giây

long time(long *t);

Công dụng: 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

Các hàm này khai báo trong alloc.h

1. calloc Cấp phát vùng nhớ cho n đối tượng kích cỡ size byte

void *calloc(unsigned n, unsigned size);

Công dụng: cấp phát vùng nhớ n*size byte. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp, khi không đủ bộ nhớ để cấp phát hàm trả về NULL.

2. malloc Cấp phát vùng nhớ cho n byte

void *malloc(unsigned n);

Công dụng: cấp phát vùng nhớ n byte. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp, khi không đủ bộ nhớ để cấp phát hàm trả về NULL.

3. free Giải phóng vùng nhớ đã cấp phát

void free(void *ptr);

Công dụng: Giải phóng vùng nhớ (đã được cấp phát bởi calloc hoặc malloc) do ptr trỏ tới

4. coreleft Cho biết bộ nhớ khả dụng trong vùng cấp phát động

+ Đối với mô hình tiny, small và medium

unsigned coreleft(void);

+ Đối với mô hình compact, large và huge

unsigned long coreleft(void);

5. realloc Cấp phát lại bộ nhớ

void *realloc(void *ptr, unsigned size);

Đối:

+ ptr là con trỏ trỏ đến một vùng nhớ đã được cấp phát từ trước

+ size là số byte cần cấp phát lại

Công dụng: 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 và ta có thể tiếp tục sử dụng chúng.

Khi thành công hàm trả về địa chỉ vùng nhớ mới, trái lại hàm trả về NULL.

Ví dụ: Đầu tiên cấp phát vùng nhớ để chứa 10 số thực, sau đó tăng kích thước lên để chứa thêm 5 số nữa. Chương trình dưới đây đáp ứng yêu cầu này.

```
#include <stdio.h>
#include <alloc.h>
main()
  {
     float *p; int i;
     p = (float*)malloc(10*sizeof(float));
     if(!p)
       {
         puts("\nKhong du bo nho lan 1");  exit(1);
       }
     for(i=0;i<10;++i) p[i]= 10.0 + i*0.5;
           /* tang kich thuoc */
     p = (float*)realloc(p,15*sizeof(float));
     if(!p)
       {
         puts("\nKhong du bo nho lan 2");
         exit(1);
       }
     for(i=10;i<15;++i) p[i]= 10.0 + i*0.5;
     for(i=0;i<15;++i)
           printf("\nSo thu %d= %0.2f",i+1,p[i]);
  }
```

§9. CÁC HÀM KIỂM SOÁT THƯ MỤC

Các hàm này khai báo trong dir.h

1. chdir đổi thư mục chủ

```
int chdir(char *s);
```

trong đó s chứa tên thư mục cần đổi (có cả ổ đĩa đường dẫn).

Nếu thành công hàm trả về 0, nếu có lỗi hàm trả về -1.

2. getcwd lấy tên thư mục chủ

```
char *getcwd(char *s, int n);
```

Ở đây:

+ n là độ dài tối đa của tên đường dẫn và thư mục (n cần lớn hơn độ dài của tên đường dẫn và thư mục)

+ s là vùng nhớ dùng để chứa tên thư mục chủ.

Hàm trả về s.

3. mkdir Tạo thư mục mới

```
int mkdir(char *s);
```

Hàm tạo một thư mục mới có tên cho trong vùng s (có thể gồm cả đường dẫn).

Hàm cho 0 nếu thành công, cho -1 nếu có lỗi

4. rmdir Xoá thư mục

```
int rmdir(char *s);
```

Hàm xoá thư mục có tên cho trong vùng s (có thể gồm cả đường dẫn).

Hàm cho 0 nếu thành công, cho -1 nếu có lỗi

5. findfirst Tìm tệp trên thư mục

```
int findfirst(char *path, struct ffblk *fb, int attrib);
```

Đối:

+ path là tên đường dẫn, tên thư mục và tên tệp cần tìm, nó có dạng như trong câu lệnh DIR của DOS ví dụ

```
path = "D:\TC\*.C"
path = "A:chg?.exe"
```

+ fb trỏ tới cấu trúc ffblk được định nghĩa trong dir.h như sau:

```
struct ffblk
    {
        char ff_reserved[21];
        char ff_attrib;
        unsigned ff_ftime;
        unsigned ff_fdate;
        long ff_fsize;
        char ff_fname[13];
    };
```

Thông tin về tệp tìm được sẽ lưu vào các thành phần của cấu trúc do con trỏ fb trỏ tới.

+ attrib là thuộc tính của tệp cần tìm, có thể là hợp của các giá trị sau:

FA_RDONLY - tệp chỉ đọc

FA_HIDDEN - tệp ẩn (trong lệnh DIR)

FA_SYSTEM - tệp hệ thống

FA_LABEL - nhãn

FA_DIREC - thư mục

FA_ARCH - tệp lưu trữ (các tệp thông thường thuộc loại này)

Công dụng: Tìm trong phạm vi quy định bởi đối path các tệp có thuộc tính attrib.

+ Nếu tìm thấy một tệp đầu như vậy thì:

- Thông tin về tệp đầu tiên tìm thấy sẽ đặt vào cấu trúc do fb trỏ tới. Thông tin này sẽ dùng cho hàm findnext để tiếp tục tìm các tệp như vậy.
- Hàm trả về giá trị 0

+ Nếu không tìm thấy: Hàm trả về -1

6. findnext tiếp tục tìm tệp trên thư mục

```
int findnext(struct ffblk *fb);
```

Công dụng: Tiếp tuc tìm tệp theo các chỉ dẫn cho trong fb.

+ Nếu tìm thấy một tệp như vậy thì:

- Thông tin về tệp đó sẽ đặt vào fb. Thông tin này sẽ dùng cho hàm findnext sau đó để tiếp tục tìm các tệp như vậy.
- Hàm trả về giá trị 0

+ Nếu không tìm thấy: Hàm trả về -1.

Ví dụ: Chương trình sau dùng các hàm findfirst, findnext và kỹ thuật đệ quy để hiện lên màn hình danh sách tất cả các tệp bên trong một thư mục nào đó. Cần báo cho chương trình biết tên thư mục hay ổ đĩa cần xét.

```
/*
    dung findnext findnext
    de tim cac tep va thu muc
*/
#include "dir.h"
#include "dos.h"
#include "stdio.h"
#include "conio.h"
char bs[]="\\*.*";
```

```
#define attr\
    (FA_RDONLY|FA_HIDDEN|FA_SYSTEM|FA_LABEL|FA_DIR
EC|FA_ARCH)
int sf,nn;
void scandir(char *d)
  {
    char dd[30],ddd[30];
    int first=1;
    struct ffblk f; int s;
    sprintf(dd,"%s%s",d,bs);
    printf("\n\n%s",dd);
    while(1)
        {
        if(first)
            {
            s=findfirst( dd, &f, attr);
            first=0;
            }
        else
          s=findnext(&f);
        if( s!=0 )
            return;
        if(f.ff_name[0]=='.')
            continue;
        if(f.ff_attrib==FA_DIREC)
            {
            sprintf(ddd,"%s\\%s",d,f.ff_name);
            scandir(ddd);
            }
        else
            {
            ++sf;
            printf("\nTen %s ",f.ff_name);
            if (sf%20==0)
                {
```

```
            printf("\nBam enter xem tiep ");
            getch();
          }
        }
      }
    }
main()
  {
  char d[30];
    clrscr();
    printf("\n Cho biet thu muc can xet (Vi du c: d:\\tc) ");
    gets(d);
    scandir(d);
    getch();
    clrscr();
  }
```

§10. CÁC HÀM KIỂM SOÁT QUÁ TRÌNH

Các hàm sau khai báo trong process.h

1. abort Kết thúc chương trình không bình thường

void abort(void);

Công dụng: Làm cho chương trình kết thúc tức thời một cách không bình thường. Hàm này không đẩy dữ kiện 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 thật cần thiết.

2. exit Kết thúc chương trình một cách bình thường

void exit(int status);

Công dụng: Làm cho chương trình kết thúc tức thời một cách bình thường. Hàm sẽ đẩy dữ kiện còn sót lại trong vùng ký ức đệm lên tệp và đóng các tệp. Hàm chuyển trị status cho hệ điều hành. Theo quy ước nếu trả về trị 0 thì chương trình kết thúc bình thường. Ta có thể sử dụng một trị khác 0 để chỉ thị một lỗi.

3. system Thực hiện một lệnh DOS

int system(char *l_dos);

Công dụng: Thực hiện câu lệnh DOS cho trong chuỗi l_dos. Khi thành công hàm trả về 0, khi có lỗi trả về -1.

Ví dụ: Chương trình dưới đây minh hoạ cách dùng hàm exit và các lệnh DOS: cls dir del

```
#include <stdio.h>
#include <process.h>
#include <ctype.h>
main()
   {
     system("cls");
     system("dir *.bak");
     puts("\nCo xoa? - C/K");
     if(toupper(getch())=='C')
       {
          system("del *.bak");
          system("dir *.bak");
       }
     else
       exit(0);
     getch();
   }
```

PHỤ LỤC 3

BẢNG MÃ ASCII VÀ MÃ QUYẾT

1. Bảng mã ASCII

Bộ ký tự ASCII gồm 256 ký được phân bố như sau:

+ 32 ký tự đầu tiên là các ký tự điều khiển không in được như ký tự Enter (mã 13), ký tự ESC (mã 27).

+ Các mã ASCII 32-47, 58-64, 91-96 và 123-127 là các ký tự đặc biệt như dấu chấm, 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 hoa từ A đến Z

+ Các mã ASCII 97-122 là các chữ cái thường từ a đến z

Lưu ý: Chữ thường có mã ASCII lớn hơn 32 so với chữ hoa tương ứng.

Ví dụ mã ASCII của a là 97 còn mã ASCII của A là 65.

+ Các mã ASCII 128-255 là các ký tự đồ hoạ.

Bảng sau cho mã ASCII của 128 ký tự đầu tiên. Để nhận được các ký tự đồ hoạ (có mã từ 128 đến 255) có thể dùng chương trình sau:

```
   /* In các ký tự đồ hoạ lên màn hình */
#include <stdio.h>
#include <conio.h>
main()
  {
    int i;
    clrscr();
    for(i=128; i<=255; ++i)
      printf("%6d%2c",i,i);
  }
```

MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
0	NUL	26	SUB	52	4
1	SOH	27	ESC	53	5
2	STX	28	FS	54	6
3	ETX	29	GS	55	7
4	EOT	30	RS	56	8
5	ENQ	31	US	57	9
6	ACK	32	Space	58	:
7	BEL	33	!	59	;
8	BS	34	"	60	<
9	HT	35	#	61	=
10	LF	36	$	62	>
11	VT	37	%	63	?
12	FF	38	&	64	@
13	CR	39	'	65	A
14	SO	40	(	66	B
15	SI	41	)	67	C
16	DLE	42	*	68	D
17	DC1	43	+	69	E
18	DC2	44	,	70	F
19	DC3	45	-	71	G
20	DC4	46	.	72	H
21	NAK	47	/	73	I
22	SYN	48	0	74	J
23	ETB	49	1	75	K
24	CAN	50	2	76	L
25	EM	51	3	77	M

MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
78	N	95	_	112	P
79	O	96	**	113	Q
80	P	97	a	114	R
81	Q	98	b	115	S
82	R	99	c	116	T
83	S	100	d	117	U
84	T	101	e	118	V
85	U	102	f	119	W
86	V	103	g	`20	X
87	W	104	h	121	Y
88	X	105	i	122	Z
89	Y	106	j	123	{
90	Z	107	k	124	|
91	[	108	l	125	}
92	\	109	m	126	~
93	]	110	n	127	DEL
94	^	111	o		

2. Bảng mã scan từ bàn phím

Mỗi phím trên bàn phím của IBM PC đều được gán một con số, gọi là mã scan, từ 1 đến 83. IBM PC AT dùng một nhóm mã khác, từ 1 đến 108 các mã này bắt đầu bằng các phím số, các phím chữ, rồi đến các phím chức năng và cuối cùng là các phím cho con trỏ, khi một phím được nhấn thì bộ xử lý của bàn phím gửi cho CPU mà scan tương ứng, khi nó được nhả thì mã scann cộng thêm 80 hex sẽ được gửi tiếp cho CPU.

Hex	Thập phân	Phím của PC	Phím của PC-AT
1	1	ESC	Tilde
2-B	2-11	1-90	1-90
C	12	trừ, gạch dưới	trừ, gạch dưới
D	13	=, +	=, +

Hex	Thập phân	Phím của PC	Phím của PC-AT
E	14	Backspace	\thanh đứng
F	15	Tab	Backspace
10	16	Q	Tab
11	17	W	Q
12	18	E	W
13	19	R	E
14	20	T	R
15	21	Y	T
16	22	U	Y
17	23	I	U
18	24	O	I
19	25	P	O
1A	26	[	P
1B	27	]	[
1C	28	Enter	]
1D	29	Ctrl	
1E	30	A	Ctrl
1F	31	S	A
20	32	D	S
21	33	F	D
22	34	G	F
23	35	H	G
24	36	J	H
25	37	K	J
26	38	L	K
27	39	Chấm phẩy,:	L
28	40	Nháy	Chấm phẩy,:
29	41	Tidle	Nháy
2A	42	Shift	trái
2B	43	\, thanh đứng	Enter
2C	44	Z	Shift trái
2D	45	X	
2E	46	C	Z

Hex	Thập phân	Phím của PC	Phím của PC-AT
2F	47	V	X
30	48	B	C
31	49	N	V
32	50	M	B
33	51	Phẩy	N
34	52	Chấm	M
35	53	/, ?	Phẩy
36	54	Shift phải	Chấm
37	55	*, PrtScr	/, ?
38	56	Alt	
39	57	Space bar	Shift phải
3A	58	Caps Lock	Alt
3B	59	F1	
3C	60	F2	
3D	61	F3	Space bar
3E	62	F4	
3F	63	F5	
40	64	F6	Caps Lock
41	65	F7	F2
42	66	F8	F4
43	67	F9	F6
44	68	F10	F8
45	69	Num Lock	F10
46	70	Scroll Lock, Break	F1
47	71	Home	F3
48	72	mũi tên lên	F5
49	73	PgUp	F7
4A	74	Dấu trừ bàn tính	F9
4B	75	Mũi tên trái	
4C	76	5	của bàn tính
4D	77	Mũi tên phải	
4F	79	End	

Hex	Thập phân	Phím của PC	Phím của PC-AT
50	80	Mũi tên xuống	
51	81	PgDn	
52	82	Ins	
53	83	Del	
5A	90		ESC
5B	91		Home
5C	92		Mũi tên trái
5D	93		End
5F	95		Num Lock
60	96		Mũi tên lên
61	97		5 của bàn tính
62	98		Mũi tên xuống
63	99		Ins
64	100		Scroll Lock
65	101		PgUp
66	102		Mũi tên phải
67	103		PgDn
68	104		Del
69	105		Sys
6A	106		*, PrtScr
6B	107		-
6C	108		+

PHỤ LỤC 4

CÀI ĐẶT TURBO C VÀO ĐĨA CỨNG

1. Bộ đĩa mềm chứa chứa các file của Turbo c

Có thể dùng 2 đĩa 1.2 để chứa các file của Turbo C. Nên dùng phương án DISKCOPY để có được các phiên bản dự trữ.

Đĩa 1 gồm 15 file (trong đó 14 file đuôi com và exe). Các file quan trọng gồm:

INSTALL.EXE dùng để cài đặt Turbo C
TC.EXE là tập tin quan trọng nhất chứa toàn bộ môi trường kết hợp.
TCC.EXE trình biên dịch đòng lệnh
MAKE.EXE chương trình quản lý tập tin
GREP.COM tìm chuỗi trong nhóm tập tin
TOUCH.COM cập nhật ngày tháng và giờ của tập tin
CPP.EXE trình tiện ích tiền xử lý
TCINST.EXE cấu hình Turbo C
TLIB.EXE thư viện các trình quản lý tập tin
UNPACK.COM bung tập tin
OBJXREF.COM trình tiện ích tham khảo tập tin đối tượng
CINSTXFR.EXE chép cài đặt version 1.5 sang version 2.0
BGIOBJ.EXE đổi trình điều khiển đồ hoạ sang tập tin đối tượng
THEPL.COM trình tiện ích giúp đỡ
README.COM trình tiện ích để đọc các tập tin README

Đĩa 2 gồm các tệp tin tiêu đề (đuôi h), tập tin thư viên (đuôi lib) và tập tin đối tượng (đuôi obj).

2. Cách cài đặt Turbo C vào đĩa cứng

Đặt đĩa 1 vào ổ A và chuyển sang A sau đó lần lượt thực hiện các thao tác sau:

1/ Khởi động chương trình install bằng cách bấm

INSTALL ↵ (ký hiệu ↵ là Enter).

2/ Tại đáy màn hình xuất hiện thông báo

ENTER - Continue ESC - Cancel

Hãy bấm Enter

3/ Xuất hiện thông báo màn hình

Enter the SOURCE drive to use: A
ENTER - Select ESC - Cancel

Hãy bấm Enter

4/ Xuất hiện thông báo màn hình

Install Turbo C on a hard Drive

F1 - Help ENTER - Select ESC - Exit

Hãy bấm Enter

5/ Trên màn hình hiện lên các dòng thông báo và chỉ dẫn:

Install Turbo C on a hard Drive

Turbo C Directory:	C:\TC
Header Files Subdirectory:	C:\TC\INCLUDE
Library Subdirectory:	C:\TC\LIB
BGI Subdirectory:	C:\TC Examples
Subdirectory:	C:\TC

Unpack Examples Yes

Memory Models ...

Start Installation

Thông báo trên nói rằng máy sẵn sàng cài đặt Turbo C tại thư mục TC trên ổ C.

Ta có thể chọn thư mục và ổ đĩa cứng để cài đặt Turbo C bằng cách di chuyển con trỏ đến dòng Turbo C Directory rồi bấm Enter. Trên màn hình hiện hộp sáng với nội dung

C:\TC

ta chọn lại ổ đĩa và thư mục sẽ cài Turbo C vào đó. Chẳng hạn ta muốn cài Turbo C vào thư mục TURBOC trên ổ D thì ta đưa vào hộp sáng nội dung sau và bấm Enter

D:\TURBOC

Khi đó trên các dòng thông báo đều có sự thay đổi: cụm từ C:\TC được thay bằng D:\TURBOC

Sau khi đã chọn ổ đĩa và thư mục ta bấm F9 (hoặc đưa con trỏ đến dòng Start Installation rồi bấm Enter). Việc cài đặt bắt đầu, máy sẽ tạo thư mục TURBOC trên ổ D và sao các file từ đĩa mềm vào đó.

6/ Khi đã sao hết các tệp của đĩa 1 máy dừng và đưa ra thông báo

Please Insert the

LIBRARY

disk into drive A:

Press any key to continue

Đưa đĩa vào ổ A và bấm Enter, việc cài đặt tiếp tục, hai thư mục con được tạo là:

+ Thư mục INCLUDE để chứa các tệp đuôi h

+ Thư mục LIB để chứa các tệp đuôi LIB và OBJ

7/ Khi đã hoàn tất việc cài đặt máy thông báo

Turbo C is now installed on your system

....

Press any key to continue

Hãy bấm Enter, việc cài đặt kết thúc, máy ra khỏi chương trình install trở về DOS tại ổ A.

Bây giờ ta có thể thực hiện các chương trình C tại thư mục đã cài đặt (ví dụ D:\TURBOC).

PHỤ LỤC 5

HƯỚNG DẪN SỬ DỤNG MÔI TRƯỜNG KẾT HỢP TURBO C

Ở Chương 1 đã giới thiệu vài nét về việc áp dụng TURBO C để soạn thảo, dịch và thực hiện chương trình. Dưới đây sẽ trình bầy các khả năng của TURBO C một cách đầy đủ hơn.

§1. VÀO MÔI TRƯỜNG C VÀ RA KHỎI MÔI TRƯỜNG C

1. Khi đang ở hệ thống (DOS) muốn vào TURBO C ta bấm TC và ENTER.

2. Để ra khỏi TURBO C trở về DOS ta có thể:

- hoặc bấm Alt-X (bấm đồng thời hai phím Alt và X),
- hoặc về menu File và bấm phím Q,
- hoặc về menu File, đưa con trỏ đến Quit Alt - X và bấm phím ENTER.

3. Khi đang ở menu File, nếu bấm phím O (hoặc đưa con trỏ đến OS Shell và bấm ENTER) thì máy sẽ tạm thời rời khỏi TURBO C và trở về DOS. Để quay trở lại TURBO C ta cần bấm exit.

§2. TRONG MÔI TRƯỜNG TURBO C

1. Từ menu chính. Để chọn một menu nào đó, ta đưa con trỏ tới menu này và bấm phím ENTER.

2. Từ trong từng menu

- Bấm chữ đầu của một tên chức năng (hoặc đưa con trỏ tới một chức năng và bấm ENTER) để thực hiện chức năng này.
- Bấm phím Esc để ra khỏi menu 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.

3. Từ bất kỳ đâu trong TURBO C

- Bấm F1 để xem lời hướng dẫn về tình hình hiện tại.
- Bấm 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 về cửa sổ Edit, bấm Alt-F để về menu File, bấm Alt-C để đến menu Compile,...

4. Tổng kết các cách bấm phím và tác dụng của chúng.

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-F vào menu File.
Alt-O	vào menu option. Alt-P vào menu Proect.
Alt-R	vào menu Run, cho chạy thử chương trình.
Alt-X	Ra khỏi TURBO C và trở về DOS.

§3. XÂY DỰNG TỆP MỚI.

1. Cửa sổ soạn thảo văn bản Edit.

Khi chọn menu Edit ta sẽ nhận được cửa sổ Edit trên màn hình. Bên trong cửa sổ có dòng thông báo về tình hình thực hiện soạn thảo bao gồm các thông tin: tên tệp đang soạn thảo, vị trí con chạy trong tệp, mốt soạn thảo đang sử dụng.

Dòng thông báo có dạng:

Line m Col n Insert Indent Tab D:Filename.ext

+ Line m	- Con chạy đang ở dòng m
+ Col n	- Con chạy đang ở cột n
+ Insert	- Đang ở mốt chèn. Để thay đổi giữa mốt chèn và mốt viết đè ta dùng phím Ins hoặc Ctrl-V.
+ Indent	- Tự động sắp thẳng lề trái với dòng trước. Bỏ hoặc vào chế độ này bằng Ctrl-OI.

+ Tab - Đang nhận dấu Tab. Dùng hay không dùng Tab bằng cách bấm các phím Ctrl-OT.

+ D:File name.ext - tên tệp đang soạn thảo

Trình soạn thảo văn bản sử dụng cấu trúc lệnh tương tự như của Notepad trong Sidekick hoặc trình soạn thảo của TUBRO PASCAL, hoặc như của Wordstar. Chiều dài tối đa của một dòng là 248 ký tự. Cửa sổ Edit có chiều rộng 77 ký tự. Nếu dòng văn bản đưa vào vượt quá chiều rộng của cửa sổ thì cửa sổ sẽ tự cuộn theo chiều ngang. Để về menu chính trong khi soạn thảo ta bấm phím F10.

Màn hình soạn thảo vẫn được hệ thống cất giữ để sau đó có thể quay trở lại làm việc.

2. 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, phải, trái), phím Page up (trang trên) và phím Page down (trang dưới).
- Xoá một dòng bằng Ctrl-Y
- Xoá một từ bằng Ctrl-T
- Đánh dấu đầu khối bằng Ctrl-K-B và đánh dấu cuối khối bằng Ctrl-K-K
- Chuyển khối bằng Ctrl-K-V
- Sao khối bằng Ctrl-K-C - Xoá khối bằng Ctrl-K-Y

Chú ý: Trong phụ lục 6 sẽ trình bầy hệ soạn thảo C một cách đầy đủ hơn.

3. 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 điểm 2.

4. Ghi tên tệp đang soạn thảo lên đĩa

Cách 1. Khi ta bấm phím F2 (từ bất kỳ chỗ nào trong TURBO C) hoặc khi ta chọn chức năng SAVE của menu File thì tệp đang soạn thảo được ghi ra đĩa. Tên của tệp chính là tên được chỉ ra trên cửa sổ Edit.

Cách 2. Khi chọn chức năng WRITE TO của menu File máy sẽ mở một hộp sáng trên màn hình và đợi. Ta cần đưa vào hộp sáng tên tệp và đường dẫn, sau đó bấm phím ENTER. Chẳng hạn như:

A:\NGAN-HANG\NGOAI-TE.M

Khi đó tên tệp đang soạn thảo sẽ được ghi vào đĩa. Tên tệp sẽ là NGOAI-TE.M, tệp nằm trong thư mục NHAN-HANG ở ổ đĩa A.

Chú ý 1: Khi ta không đưa vào dấu chấm và phần mở rộng của tên tệp thì máy sẽ xem tệp có đuôi C.

Chú ý 2: Khi ta không đưa vào đường dẫn và thư mục thì tệp sẽ được ghi vào thư mục chủ.

§4. HIỆU CHỈNH SỬA CHỮA TỆP

Để hiệu chỉnh một tệp đã có sẵn ta dùng chức năng Load của menu File để đọc nó vào bộ nhớ. Sau đó ta thực hiện các bổ sung và sửa chữa cần thiết (sử dụng các lệnh nói trong điểm 2 của §3). Cuối cùng ta ghi tệp nhận được ra đĩa (sử dụng các cách nói trong điểm 4 của §3).

Chú ý: Khi đọc một tệp chưa tồn tại thì cửa sổ Edit vẫn mở, tên tệp được chỉ ra trên màn hình. Nội dung của tệp là rỗng. Bây giờ ta có thể soạn thảo một tệp mới.

§5. CÁC MENU TRONG MÔI TRƯỜNG KẾT HỢP CỦA TURBO C

5.1. Menu File. Menu File có các chức năng sau.

1. **Load (F3).** Nạp một tệp vào để làm việc.
2. **Pick (Alt-F3).** Lấy một tệp trong danh sách 8 tệp trước đây đã nạp vào cửa sổ Edit.
3. **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. Ta có thể thay đổi tên khi ghi tệp vào đĩa (xem §3, điểm 4).
4. **Save (F2).** Ghi tệp đang soạn thảo vào đĩa.
5. **Write to.** Ghi tệp đang soạn thảo lên đĩa theo tên mới hoặc đè lên tệp đã có trên đĩa.
6. **Directory.** Hiện thư mục hoặc tập hợp các tệp cần quan tâm.
7. **Change dir.** Hiện nội dung thư mục hiện tại và cho phép đổi ổ đĩa hoặc thư mục.
8. **OS Shell.** Tạm thời rời bỏ môi trường kết hợp TURBO C và trở về DOS. Để quay trở lại TURBO C ta bấm exit.
9. **Quit (Alt-X).** Ra hẳn TURBO C và trở về DOS.

5.2. Menu Edit.

Menu Edit có tác dụng khởi động chương trình soạn thảo văn bản có sẵn của TURBO C. Trong chương trình soạn thảo này ta có trể trở về menu chính hoặc trở về các menu khác (xem §2). Màn hình soạn thảo vẫn được cất giữ để sau đó có thể trở lại tiếp tục làm việc.

5.3. Menu Run. Menu Run có các chức năng:

1. **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 sáng Primary C file (menu Compile) được xét. Nếu hộp sáng này rỗng thì chương trình đang soạn thảo được xét.
2. **Program reset (Ctrl-F2).** Lập lại chế độ thực hiện toàn bộ chương trình.
3. **Go to cusor (F4).** Thực hiện chương trình cho đến dòng lệnh chứa con trỏ.
4. **Trace into (F7).** Thực hiện từng lệnh một.
5. **Step over (F8).** Thực hiện từng câu lệnh và xem lời gọi hàm là một câu lênh, như vậy sẽ nhẩy qua một hàm.
6. **User screen (Alt-F5).** Trở lại màn hình sử dụng để xem kết quả do chương trình tạo ra. Bấm Enter sẽ trở về môi trường C

5.4. Menu Compile.

Ta có thể dùng các chức năng của menu Compile để dịch chương trình gốc sang tệp .OBJ (Compile to OBJ), để tạo ra tệp .EXE (Make EXE File), để liên kết chương trình đã dịch thành .EXE (Linh EXE File), để dịch lại toàn bộ (Build all) hoặc chọn tệp C cần dịch (Primary C File).

1. **Compile to OBJ.** Dịch một tệp C thành tệp OBJ. Tệp được xác định theo trật tự sau:
   - Trong project Name (nếu có)
   - Trong Primary C File (nếu có).
   - Trong cửa sổ soạn thảo.
2. **Make EXE file.** Dịch và liên kết để tạo thành tệp EXE. Các tệp nguồn được xác định như trong Compile to OBJ.
3. **Link EXE.** Liên kết các tệp OBJ và LIB để tạo thành tệp chương trình thực hiện đuôi EXE.
4. **Build all.** Dịch lại và liên kết để tạo thành tệp EXE. Các tệp nguồn được xác định như trong Compile to OBJ (không cần biết chúng có bị thay đổi hay không).
5. **Primary C file.** Dùng chức năng này để xác định tệp C cần dịch thành OBJ hoặc EXE.

Khi TURBO C đang tiến hành dịch chương trình, một cửa sổ chính giữa màn hình được mở để thông báo cho bạn biết về tình hình dịch.

Nếu trong quá trình dịch có lỗi, TURBO C sẽ thông báo lỗi vào cửa sổ Message và hiện sáng dòng lỗi đầu tiên.

5.5. Menu Project. Menu này có các chức năng sau.

1. **Project Name.** Chức năng này cho phép chọn tệp Project chứa thông tin về các tệp cần dịch hoặc liên kết.
2. **Break make on.** Chức năng này cho phép xác định liệu việc dịch có phải dừng lại khi gặp warning hay error hay không.
3. **Auto dependencies.** Không xét vì ít dùng.
4. **Clear project.** Chức năng này dùng để xoá tệp Project và đặt lại cửa sổ Message.
5. **Remove massage.** Xoá cửa sổ Message.

Chú ý: Trong phụ lục 7 sẽ trình bầy cách dùng menu Project để dịch một chương trình viết trên nhiều tệp.

5.6. Menu option. menu này gồm các chức năng:

1. **Compiler.** Dùng để chọn mô hình bộ nhớ (chức năng con Model) và xác định độ dài cực đại của tên trong chương trình (chức năng con Source).
2. **Linker.** Chọn phương thức liên kết.
3. **Enviroment.** Chọn môi trường.
4. **Directory.** Đây là chức năng hay dùng để thiết lập các thư mục liên quan tới quá trình dịch.
5. **Argument.** Đưa vào các tham số dòng lệnh cho chương trình chạy trong môi trường C.
6. **Save option.** Ghi lại các thay đổi trong menu option.
7. **Retrieve option.** Khôi phục lại thông tin option từ một tệp nào đó.

Các menu **Debug** và **Break/watch** sẽ khảo sát trong phụ lục 9.

PHỤ LỤC 6

HỆ SOẠN THẢO CỦA TURBO C

Trình soạn thảo Turbo C dùng để soạn thảo những chương trình nguồn viết bằng ngôn ngữ Turbo C, trong đó bao gồm nhiều chức năng của một hệ soạn thảo giúp cho công việc soạn thảo trở nên thuận tiện, dễ dàng. Phần này sẽ nói đến cách sử dụng chương trình soạn thảo của Turbo C.

§1. KHỞI ĐỘNG CHƯƠNG TRÌNH SOẠN THẢO

Vì chương trình soạn thảo của Turbo C là một thành phần của môi trường kết hợp Turbo C. Do vậy để khởi động chương trình soạn thảo của Turbo C, ta chỉ việc khởi động môi trường kết hợp của Turbo C, bằng cách thực hiện file chương trình TC.EXE được đặt trên thư mục nào đó trên đĩa cứng hoặc đĩa mềm.

Ví dụ:

```
C:\>CD TC ↵
C:\TC>TC ↵
```

Sau khi khởi động xong, màn hình của Turbo C có dạng như sau:

```
 File  Edit  Run  Compile  Project  Options  Debug  Break/watch
 ─────────────────────────── Edit ──────────────────────────────
   Line  Col  Insert  Indent  Tab  Fill  Unindent  C:NONAME.C




 ────────────────────────── Message ────────────────────────────
 F1-Help  F5-Zoom  F6-Switch  F7-Trace  F8-Step  F9-Make  F10-Menu
```

Khi đó, nếu phiên làm việc trước đây đang làm việc với file nào đó, thì nội dung của file đó sẽ được tự đưa ra. Còn nếu không màn hình sẽ có dạng như trên với tên file là NONAME.C và con trỏ sẽ ở đầu của dòng đầu tiên trong cửa sổ soạn thảo Edit, hoặc cũng có thể không nhìn thấy con trỏ mà thấy hộp đen trên menu File của Menu nằm ngang, khi đó ta chỉ việc bấm phím Esc để đưa con trỏ về cửa sổ soạn thảo Edit.

§2. CÁC PHÍM CHỨC NĂNG THÔNG DỤNG KHI SOẠN THẢO CHƯƠNG TRÌNH

+ ← ↑ ↓ →	: Chuyển con trỏ sang trái, lên trên, xuống dưới, sang phải một vị trí
+ Ctrl + →	: Chuyển con trỏ sang phải một từ
+ Ctrl + ←	: Chuyển con trỏ sang trái một từ
+ Home, End	: Chuyển con trỏ về đầu, về cuối dòng
+ Ctrl Home	: Chuyển con trỏ về đầu trang màn hình
+ Ctrl End	: Chuyển con trỏ về cuối trang màn hình
+ Page Up	: Chuyển con trỏ lên trang màn hình trước
+ Page Down	: Chuyển con trỏ xuống trang màn hình sau
+ Ctrl Page Up	: Chuyển con trỏ về đầu chương trình
+ Ctrl Page Down	: Chuyển con trỏ về cuối chương trình
+ ← (BackSpace)	: Xoá ký tự ngay trước con trỏ
+ Delete	: Xoá ký tự tại vị trí con trỏ
+ Ctrl + T	: Xoá từ đang chứa con trỏ
+ Ctrl + Q + Y	: Xoá từ vị trí con trỏ đến cuối dòng
+ Ctrl + Y	: Xoá dòng chứa con trỏ
+ Ctrl + N	: Chèn thêm dòng trống vào trước dòng đang chứa con trỏ
+ Caps Lock	: Chuyển đổi giữa hai chế độ chữ hoa và chữ thường
+ Insert	: Chuyển đổi giữa hai chế độ chèn và đè.

Ở chế độ đè, ký tự đánh vào sẽ đè vào ký tự tại vị trí con trỏ. Còn ở chế độ chèn, ký tự đánh vào sẽ được chèn vào tại vị trí con trỏ (tức là đồng thời đẩy các ký tự từ vị trí con trỏ tới cuối dòng sang phải một vị trí)

+ ← (Enter) : Khi đang ở chế độ đè, có tác dụng chuyển con trỏ xuống dòng tiếp theo. Còn nếu đang ở chế độ chèn sẽ chuyển các ký tự từ vị trí con trỏ đến hết dòng xuống dòng mới tiếp theo.

§3. QUẢN LÝ CÁC FILE TRÊN ĐĨA

3.1. Ghi nội dung file đang soạn thảo vào đĩa

Bấm phím F2. Khi đó nếu chưa đặt tên file (đang dùng tên là NONAME.C) thì máy sẽ mời đưa vào tên file tại vị trí con trỏ rồi bấm .

Chú ý: Nếu không gõ phần đuôi vào tên file thì Turbo C sẽ tự gán vào đuôi .C

3.2. Gọi file mới ra soạn thảo

- Bấm phím F3. Khi đó có thể máy sẽ hỏi có muốn ghi file đang soạn thảo vào đĩa hay không: Bấm Y - nếu có, bấm N - nếu không.

Sau đó mới chọn tên file cần soạn thảo tại vị trí con trỏ. Tại đây ta có hai cách chọn như sau:

Cách 1: Gõ trực tiếp tên file cần soạn thảo vào tại vị trí con trỏ (có thể bao gồm cả ổ đĩa và đường dẫn nếu muốn chỉ định ở nơi khác), rồi bấm ↵. Khi đó nếu file đã tồn tại ở nơi được chỉ ra, thì nội dung cũ của file sẽ được đưa ra màn hình để soạn thảo tiếp. Ngược lại, nó được coi như file mới và mời ta bắt đầu soạn thảo từ vị trí con trỏ. Như vậy cách này có thể dùng để tạo file mới.

Cách 2: Gõ vào nhóm ký tự *.C hoặc nếu đã có thì bấm ngay phím ↵ để chọn những file có đuôi .C đã có trên đĩa. Lúc đó trên màn hình sẽ hiện ra một bảng chứa danh sách các tên file có đuôi .C cùng với các tên thư mục con nếu có của thư mục được chọn. Muốn chọn file hoặc thư mục nào ta đưa hộp sáng tới đó rồi bấm ↵. Nếu muốn chuyển về thư mục mẹ, thì chuyển hộp sáng về kí hiệu "..\" rồi bấm ↵ . Như vậy cách này thường được dùng để gọi những file chương trình đã có trên đĩa mà ta không nhớ chính xác tên hoặc nơi chứa nó.

Chú ý: - Hai chức năng trên có thể chọn từ menu File bằng cách bấm Alt-F, sau đó chọn chức năng "Save F2" để ghi nội dung file đang soạn thảo lên đĩa, hoặc chọn "Load F3" để gọi file mới ra soạn thảo.

- Ngoài ra menu File còn chứa những chức năng thông dụng sau:
  + Pick Alt-F3: để chọn một trong 8 file đang soạn thảo gần đây nhất ra soạn thảo tiếp.
  + Write to: ghi nội dung file đang soạn thảo lên đĩa với một tên khác.
  + New: soạn thảo tập tin mới với tên là NONAME.C
  + Directory: xem các tập tin trong thư mục chủ
  + Change dir: đổi thư mục chủ

§4. SỬ DỤNG CÁC CHỨC NĂNG VỀ KHỐI DÒNG

Trong khi soạn thảo nội dung file, ta thường gặp những đoạn văn bản được xuất hiện nhiều lần trong nội dung, thì với những đoạn đó ta có thể đánh dấu nó dưới dạng một khối dòng, để sau đó có thể copy khối đó tới vị trí cần xuất hiện mà không cần phải gõ lại toàn bộ đoạn đó, hoặc cũng có thể áp dụng khối dòng để thực hiện các mục đích thường gặp khác sẽ được trình bày dưới đây.

4.1. Cách đánh dấu khối dòng

- Đưa con trỏ tới dòng đầu tiên của khối, bấm Ctrl + K + B, sau đó đưa con trỏ tới dòng cuối cùng của khối, bấm Ctrl + K + K Khi đó các dòng trong phạm vi đánh dấu sẽ được đổi mầu nền để tiện theo dõi.

4.2. Các thao tác với khối dòng

+ Copy khối dòng: Đưa con trỏ tới nơi cần Copy rồi bấm Ctrl + K + C

+ Di chuyển khối dòng: Đưa con trỏ tới nơi cần di chuyển rồi bấm Ctrl + K + V

+ Tắt/hiện đánh dấu khối dòng: Bấm Ctrl + K + H

+ Xoá khối dòng: Bấm Ctrl + K + Y

+ In khối dòng ra máy in: Bấm Ctrl + K + P. Nếu không có khối dòng nào được chọn thì toàn bộ chương trình sẽ được in ra máy in.

+ Ghi khối dòng ra đĩa: Bấm Ctrl + K + W. Sau đó vào tiếp tên file tại vị trí con trỏ để ghi khối dòng vào đó rồi bấm .

Chức năng này được áp dụng khi ta muốn Copy khối dòng sang một chương trình khác nhờ áp dụng tiếp chức năng dưới đây.

+ Chèn nội dung file trên đĩa vào tại vị trí con trỏ: Bấm Ctrl + K + R, tiếp theo vào tên file cần chèn tại vị trí con trỏ rồi bấm . Khi đó nội dung file chèn vào sẽ được đánh dấu dưới dạng một khối dòng.

§5. TÌM KIẾM VĂN BẢN TRONG NỘI DUNG FILE ĐANG SOẠN THẢO

Chức năng này được dùng để di chuyển nhanh con trỏ tới những vị trí xuất hiện của nhóm từ nào đó mà ta cần quan tâm.

Để thực hiện bấm Ctrl + Q + F. Tiếp theo:

+ Vào nhóm từ cần tìm tại vị trí con trỏ rồi bấm ↵ ví dụ: Do ↵

+ Vào các tuỳ chọn trong quá trình tìm tại vị trí con trỏ rồi bấm . Các tuỳ chọn khi tìm gồm có:

B: Tìm ngược từ vị trí con trỏ về đầu chương trình

G: Tìm xuôi từ vị trí con trỏ về cuối chương trình

L: Chỉ tìm trên khối được đánh dấu

U: Tìm không phân biệt chữ hoa, chữ thường

W: Tìm từ nguyên vẹn

Chú ý:

- Các tuỳ chọn có thể gõ vào là chữ hoa hoặc chữ thường và có thể kết hợp nhiều tuỳ chọn với nhau.
- Nhóm từ cần tìm không vượt quá 30 kí tự.

Ví dụ: Khi vào các tuỳ chọn: GuW ↵ tức là sẽ tìm từ vị trí con trỏ về cuối chương trình, trong quá trình tìm sẽ chuyển con trỏ đến cả các từ "do", "DO", "dO", nhưng sẽ không chuyển con trỏ tới các từ dạng "dong", "Don" ... vì có tuỳ chọn W.

Khi tìm con trỏ sẽ chuyển đến vị trí đầu tiên thoả mãn trọng phạm vi được chọn.

Để chuyển con trỏ tới vị trí thoả mãn tiếp theo bấm Ctrl + L.

§6. TÌM KIẾM VÀ THAY THẾ VĂN BẢN

- Bấm Ctrl + Q + A. Sau đó:
  + Vào nhóm từ cần tìm tại vị trí con trỏ rồi bấm ↵, ví dụ Bien1 ↵
  + Vào nhóm từ cần thay thế tại vị trí con trỏ rồi bấm ↵, ví dụ Bien2 ↵
  + Tiếp theo vào những tuỳ chọn khi tìm và thay thế. Các tuỳ chọn ở đây bao gồm tất cả các tuỳ chọn của chức năng tìm.

Chú ý: Tuỳ chọn G sẽ có ý nghĩa khác: Tìm từ đầu đến cuối chương trình.

Thêm hai tuỳ chọn là:

N - nếu có sẽ thay thế ngay khi tìm thấy. Còn nếu không, mỗi khi tìm thấy máy sẽ hỏi có muốn thay thế hay không.

s - Là một số nguyên dương cụ thể để chỉ số lầnđược tìm và thay thế, nếu không có tuỳ chọn này thì chỉ tiến hành tìm và thay thế một lần (tuỳ chọn này không có tác dụng đối với G, nó chỉ có tác dụng đối với B).

Ví dụ: Nếu vào các tuỳ chọn GUW10 ↵ thì sẽ tìm từ đầu đến cuối chương trình, trong quá trình tìm sẽ không phân biệt chữ hoa, chữ thường, tìm một từ nguyên vẹn. Khi tìm thấy máy sẽ hỏi có muốn thay thế hay không (Replace (Y/N)):

+ Bấm Y nếu có,
+ Bấm N nếu không và con trỏ sẽ chuyển xuống từ tiếp theo cũng thoả mãn điều kiện tìm kiếm,
+ Bấm ESC nếu muốn kết thúc quá trình tìm và thay thế.

(Chú ý tham số 10 ở đây không có tác dụng gì).

§7. Ý NGHĨA CỦA DÒNG TRẠNG THÁI

Dòng trạng thái chính là dòng:

Line Col Insert Indent Tab Fill Unindent

nằm trong cửa sổ soạn thảo (Edit) để thông báo trạng thái soạn thảo tại thời điểm hiện tại.

Ý nghĩa của các thành phần trên dòng như sau:

+ Line: Cho biết vị trí dòng của con trỏ trên màn hình.

+ Col: Cho biết vị trí cột của con trỏ trên màn hình.

+ Insert: Cho biết đang ở chế độ chèn, nếu không có thì đang ở chế độ đè. Để chuyển đổi giữa hai chế độ này ta bấm phím Insert như đã biết.

+ Indent: Thông báo đang ở chế độ tự động căn lề trái khi chuyển xuống dòng mới bằng phím. Tức là khi đó con trỏ sẽ tự căn thẳng lề với dòng ngay trên. Còn nếu không có Indent (chế độ không tự động căn lề), thì con trỏ luôn chuyển về cột đầu tiên của dòng. Để chuyển đổi giữa hai chế độ này, ta gõ tổ hợp phím: Ctrl + O + I.

+ Tab: Khi xuất hiện, thì mỗi lần gõ phím Tab, con trỏ sẽ rời sang phải 8 ký tự. Muốn hiển thị, hoặc tắt chữ Tab, ta gõ tổ hợp phím Ctrl+ O + T.

+ Fill: Khi được hiển thị nghĩa là đang ở chức năng tạo bảng. Nếu muốn tắt chức năng tự động căn bảng ta gõ tổ hợp phím: Ctrl + O + F.

+ Unindent: Thông báo đang ở chế độ tự căn lề trái khi dùng phím xoá Backspace. Tức là khi đó nếu đưa con trỏ tới chữ cái đầu của dòng nào đó, rồi bấm phím Backspace, thì dòng đó sẽ được căn thẳng mép trái của dòng phía trên gần nhất có mép trái thụt vào ít hơn. Để tắt mở chức năng này ta gõ: Ctrl + O + U.

PHỤ LỤC 7

DÙNG MENU PROJECT DỊCH CHƯƠNG TRÌNH VIẾT TRÊN NHIỀU TỆP

Menu Project cho phép tổ chức một chương trình lớn thành nhiều mô đun độc lập, mỗi mô đun chứa trên các tệp khác nhau và có thể ở dạng nguồn (đuôi C) hay dạng đích (đuôi OBJ). Project có khả năng:

+ Dịch các mô đun chương trình một cách độc lập (nếu chúng ở dạng nguồn)

+ Liên kết các mô đun vừa dịch và các mô đun đã ở dạng đích (được dịch từ trước) để tạo thành một tệp chương trình thực hiện đuôi EXE.

Như vậy bằng cách dùng project ta chỉ phải dịch lại nhưng mô đun nào có thay đổi. Bây giờ ta minh hoạ cách dùng project trên 2 ví dụ.

Ví dụ 1. Giả sử chương trình của ta khá lớn và phải viết trên nhiều tệp. Hơn nữa có những tệp lại bao hàm các tệp khác (bằng #include). Xét một thí dụ cụ thể là cần phải dịch chương trình trên các tệp:

- myprog.C có lệnh bao hàm tệp myprog.h
- myprog1.C có lệnh bao hàm tệp myprog.h và defs.h
- myprog2.C có lệnh bao hàm tệp myprog.h và ifdefs.h

và phải tạo ra tệp thực hiện với tên myprog.exe.

Muốn vậy đầu tiên sử dụng menu Project để chọn tên project. Trong trường hợp này đó là tệp:

myprog.prj

Sau đó ta sử dụng trình soạn thảo văn bản của TURBO.C để soạn thảo nội dung cho tệp myprog.prj như sau:

myprog.c (myprog.h)

myprog1.c (myprog.h defs.h)

myprog2.c (myprog.h ifdefs.h)

Một khi ta đã thiết lập được tệp myprog.prj như vậy rồi thì bất kỳ lúc nào ta cũng có thể dịch hay dịch lại chương trình nêu trên bằng cách dùng phím chức năng F9.

Ví dụ 2. Giả sử trong tệp chương trình nguồn TINH.C có sử dụng các hàm trong các tệp thư viện (do ta xây dựng):

MA_TRAN.OBJ

HAM_SO.OBJ

Khi đó để dịch tệp TINH.C ta làm như sau:

+ Tạo tệp TINH.PRJ với nội dụng:

TINH.C

MA_TRAN.OBJ

HAM_SO.OBJ

+ Dùng chức năng Project name của menu Project để chọn tệp project là tệp TINH.PRJ

+ Bấm phím F9

Kết quả ta sẽ nhận được tệp TINH.EXE.

Chú ý 1: Các tệp MA_TRAN.OBJ, HAM_SO.OBJ phải được dịch theo cùng mô hình bộ nhớ (xem Phụ luc 10) như khi dịch tệp TINH.C. Nếu mỗi mô đun được dịch theo một mô hình khác nhau thì không thể liên kết được.

Chú ý 2: Các menu Project và Compile tiến hành dịch trong môi trường kết hợp của Turbo C vì vậy chỉ cho phép dịch các chương trình vừa phải. Đối với các chương trình lớn cần dịch bằng trình TCC từ môi trường DOS (xem Phụ lục 8).

PHỤ LỤC 8

DỊCH CHƯƠNG TRÌNH THEO CHẾ ĐỘ DÒNG LỆNH TCC

Trình biên dịch TCC thực hiện ở mức DOS cho phép dịch các chương trình lớn gồm nhiều mô đun trên nhiều tệp ở dạng nguồn (đuôi C) và dạng đích (đuôi OBJ). TCC sẽ tiến hành:

+ Dịch một cách độc lập các mô đun nguồn

+ Liên kết các mô đun vừa dịch, các mô đun dạng đích (đã dịch từ trước) và các mô đun thư viện (đuôi LIB) để tạo thành tệp chương trình thực hiện đuôi EXE.

1. Dòng lệnh TCC ở mức DOS có dạng sau:

tcc [tuỳ chọn,...] tên tệp [tên tệp,...]

2. Chú ý khi viết:

+ Giữa tcc các tuỳ chọn và tên tệp cách nhau ít nhất một khoảng trống.

+ Các tuỳ chọn đều bắt đầu bằng dấu gạch ngang ví dụ -ml

3. Quy ước về tên tệp:

+ ten_tep hiểu là cần dịch tệp ten_tep.c

+ ten_tep.c hiểu là cần dịch tệp ten_tep.c

+ ten_tep.xyz hiểu là cần dịch tệp ten_tep.xyz

+ ten_tep.obj được gép vào tệp chính khi liên kết

+ ten_tep.lib được gép vào tệp chính như tệp thư viện

+ ten_tep.asm gọi MASM để dịch thành OBJ và liên kết

4. Tên tệp chương trình thực hiện

Thông thường trình biên dịch đặt tên cho tệp thực hiện là tên của tệp gốc hoặc tệp đích đầu tiên trên dòng lệnh.

Nếu muốn dùng tên khác đặt cho tệp thực hiện cần sử dụng tuỳ chọn

-etên_tệp_thực_hiện

đặt trước mọi tên tệp khác.

5. Vài ví dụ

Ví dụ 1. Dòng lệnh:

tcc -Id:\tc\include -Ld:\tc\lib -ekq tt1.c tv.obj tt2

có ý nghĩa sau:

+ Thư mục chứa các tệp tiêu đề (đuôi h) là d:\tc\include

+ Thư mục chứa các tệp thư viện (đuôi lib) là d:\tc\lib

+ Tên chương trình thực hiện là kq.exe

+ Tệp tt1.c cần được dịch

+ Tệp tv.obj cần được ghép vào khi liên kết

+ Tệp tt2.c cần được dịch

Ví dụ 2. Dòng lệnh:

tcc -Ic:\tc\include -Lc:\tc\lib -ml -C -K s1 s2.c tv.lib

có ý nghĩa sau:

+ Thư mục chứa các tệp tiêu đề (đuôi h) là c:\tc\include

+ Thư mục chứa các tệp thư viện (đuôi lib) là c:\tc\lib

+ Mô hình bộ nhớ large (tuỳ chọn -ml)

+ Cho phép các lời chú thích lồng nhau (tuỳ chọn -C)

+ Khai báo char chuyển thành unsigned char (tuỳ chọn -K)

+ Tệp thực hiện là s1.exe

+ Các tệp nguồn s1.c và s2.c cần được dịch

+ Tệp thư viên tv.lib được ghép vào khi liên kết

6. Các tuỳ chọn thường dùng

Tuỳ chọn mô hình bộ nhớ

- mc mô hình compact
- mh mô hình huge
- ml mô hình large
- mm mô hình medium
- ms mô hình small
- mt mô hình tiny

Tuỳ chọn xử lý chương trình gốc

- A Tạo ra chương trình tương thích với ANSI, mọi từ khoá mở rộng đều bị bỏ qua và có thể dùng như tên gọi. Đó là các từ khoá sau:

  near far huge cdecl asm pascal interrupt

  _es _ds _cs _ss và các biến giả thanh ghi như _AX _BX

  ...

- C Cho phép các lời chú thích được lồng nhau.

- K Xử lý mọi khai báo char là unsigned char (ngầm định char là signed char)

Tuỳ chọn môi trường

-Itm Tìm các tệp tiêu đề trong thư mục tm (gồm cả ổ đĩa và đường dẫn)

-Ltm Tìm các tệp thư viện trong thư mục tm (gồm cả ổ đĩa và đường dẫn)

Tuỳ chọn kết quả

- eten_tep Tệp thực hiện có tên là ten_tep.exe
- oten_tep Dịch tệp ten_tep.c thành tệp đích ten_tep.obj

PHỤ LỤC 9

SỬA LỖI CÚ PHÁP VÀ GỠ RỐI CHƯƠNG TRÌNH

Các chức năng sửa lỗi cú pháp và gỡ rối chương trình thuộc phần Turbo C Debugger. Trong đó có những khả năng rất mạnh và thuận tiện giúp ta hiệu chỉnh những lỗi cú pháp, theo dõi quá trình thực hiện từng bước của chương trình để tìm ra nguyên nhân chương trình bị quẩn, hoặc chạy sai kết quả ... . Phần này sẽ trình bầy các khả năng đó của Turbo C Debugger.

Turbo C Debugger được cài sẵn vào trong môi trường kết hợp của Turbo C và trở thành một thành phần của môi trường kết hợp. Do vậy khi môi trường kết hợp được khởi động Turbo C Debugger cũng được tự khởi động và chờ sẵn sàng làm việc (menu Debug).

§1. SỬA LỖI CÚ PHÁP

Thông thường khi vừa soạn thảo xong một chương trình, ta vẫn còn mắc một số lỗi cú pháp nào đó. Do vậy nếu thực hiện biên dịch hoặc chạy thử chương trình thì trên màn hình sẽ hiện ra những thông báo về những sai sót đó.

Ví dụ: Trên màn hình soạn thảo đã vào xong nội dung của file THU1.C có nội dung như sau:

```
/*thu1.c*/    /* Vi du ve chuong trinh sai cu phap */
        main()
          {
            int a=8
            int b=5;
            tich=a*b;
            clrscr();
            printf(" Tich a * b= %d",tich);
          }
```

Khi chạy hoặc biên dịch chương trình, cửa sổ Compile sẽ có thông báo lỗi:

"Error: Press any key"

Khi bấm phím bất kỳ, trên màn hình sẽ hiện ra cửa sổ Message, cửa sổ này thông thường luôn xuất hiện ở cuối màn hình ngay dưới cửa sổ Edit. Song cũng có thể nó chiếm toàn bộ màn hình (không còn chỗ cho cửa sổ Edit), khi đó để lại hiện hai cửa sổ ta bấm phím F5.

Trong cửa sổ Message có chứa danh sách những lỗi cú pháp mà chương trình mắc phải. Dựa trên danh sách này ta tiến hành việc nối kết giữa thông báo lỗi và lỗi trong cửa sổ Edit thông qua môi trường kết hợp.

1.1. Nối kết giữa thông báo lỗi và lỗi.

Việc kết nối này nhằm mục đích phát hiện các nguyên nhân sai và chuyển nhanh con trỏ tới nơi đó để sửa. Còn đối với những lỗi đơn giản, dễ phát hiện, và dễ chuyển đến, thì có thể bấm ngay ↵, hoặc Alt+E để chuyển vào cửa sổ Edit và tiến hành sửa.

Quá trình kết nối được thực hiện ngay khi xuất hiện cửa sổ Message. Thông báo lỗi đầu tiên được làm sáng và dòng chứa lỗi ứng với thông báo đó cũng được làm sáng. Ngoài ra trên dòng chứa lỗi còn xuất hiện một điểm đánh dấu (có hình khối) cho biết nơi chương trình biên dịch phát hiện ra lỗi đầu tiên.

Hình ảnh màn hình trước khi kết nối có dạng:

```
File  Edit  Run  Compile  Project  Options  Debug  Break/watch
 ------------------------- Edit ------------------------------
 Line 5 Col 7 Insert Indent Tab Fill Unindent C:NONAME.C
   /*thu1.c*/
   /* Vi du ve chuong trinh sai cu phap */
   main()
    {
    int a=8
    int b=5;
    tich=a*b;
    clrscr();
    printf(" Tich a * b= %d",tich);
    }
 ------------------------- Message ---------------------------
 Compiling C:\TC\THU1.C
 Error C:\TC\THU1.C 5: Declaration syntax error in function main
 Error C:\TC\THU1.C 6: Symbol 'tich' in in function main
 Error C:\TC\THU1.C 6: Symbol 'b' in function main

F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu
```

Thông báo lỗi thứ nhất trong cửa sổ Message là "Declaration syntax error in function main". Dòng này được làm sáng trong cửa sổ. Nếu máy đang dùng có màn hình mầu, thì dòng "int b=5; " trong chương trình cũng được làm sáng và điểm đánh dấu là một khối mầu đỏ xuất hiện ngay sau chữ "int" trên dòng đó.

Lỗi xẩy ra do ta quên đưa dấu chấm phẩy khi kết thúc khai báo:

int a=8

Chương trình biên dịch không biết đã xẩy ra lỗi cho đến khi nó thấy chữ "int" trên dòng kế tiếp, là nơi có điểm đánh dấu. Do vậy cách đánh dấu nơi có lỗi của chương trình biên dịch thường không trùng với điểm thực sự có lỗi mà thường ở những vị trí lân cận để từ đó gợi ý ta chuyển đến nơi thực sự có lỗi và tiến hành sửa.

Có thể dùng các phím mũi tên để duyệt qua các thông báo lỗi trong cửa sổ Message. Khi vệt sáng đến thông báo khác, dòng tương ứng của chương trình có lỗi đó trong cửa sổ Edit sẽ được làm sáng.

Biến "tich" chưa được khai báo mà đã được sử dụng trong lệnh gán:

tich=a*b;

nên phát sinh thông báo lỗi thứ hai: "Symbol 'tich' is in function main". Vì biến "tich" chưa được khai báo mà chương trình biên dịch không hiểu biến "b" trong dòng lệnh trên mặc dù biến "b" đã được khai báo, nên phát sinh tiếp lỗi thứ 3:

"Symbol 'b' in function main ".

Từ đây ta thấy từ một lỗi gặp phải có thể đánh lừa chương trình dịch là có nhiều lỗi khác.

1.2. Sửa lỗi

Để sửa lỗi, dùng các phím mũi tên đưa hộp sáng tới dòng thông báo lỗi trong cửa sổ Message rồi bấm . Khi đó con trỏ sẽ lại xuất hiên trên cửa sổ Edit tại vị trí có đánh dấu, lúc này ta đưa con trỏ tới nơi cần sửa và tiến hành sửa. Trong quá trình sửa có thể bấm phím F6 để mở rộng toàn bộ cửa sổ Edit (cửa sổ Message khi đó sẽ không xuất hiện).

Để quay về cửa sổ Message ta bấm phím F6. Sau đó lại chọn thông báo lỗi khác, bấm để chuyển sang cửa sổ soạn thảo và sửa.

Chú ý: Vì chương trình biên dịch có sinh ra những thông báo lỗi không chính xác từ một lỗi trước đó. Nên sau khi sửa xong một số lỗi nào đó, ta nên biên dịch lại để bỏ đi những dòng thông báo lỗi thừa trong cửa sổ Message.

Khi biên dịch và chạy chương trình, có thể gặp các trường hợp chương trình bị quẩn hoặc chạy sai kết quả mà không thể phát hiện nguyên nhân bằng cách xem tổng thể toàn bộ chương trình. Khi đó ta có thể dùng các chức năng thực hiện chương trình từng bước của chương trình gỡ rối và có thể kết hợp với việc sử dụng những tiện ích hữu hiệu khác được trình bày dưới đây.

§2. LẦN TỪNG BƯỚC (MỖI BƯỚC MỘT DÒNG)

2.1. Mỗi lần một bước

Ở mỗi bước thực hiện ta phải bấm phím F7. Với lần bấm F7 đầu tiên, dòng đầu tiên trong chương trình (dòng main())sẽ được làm sáng, dòng được làm sáng này được gọi là dòng chuẩn bị thực hiện, và nó sẽ được thực hiện ở lần bấm phím F7 tiếp theo. Mỗi lần bấm phím F7 dòng đang được làm sáng sẽ được thực hiện, sau đó trở về mầu nền bình thường, và tuỳ theo nội dung của dòng đó mà một dòng lệnh tiếp theo nào đó sẽ được làm sáng để chuẩn bị thực hiện ở bước tiếp theo.

Như vậy nhờ chạy từng bước, chúng ta dễ dàng nắm được các lỗi logic trong chương trình.

Chú ý: Ta cũng có thể dùng phím F8 thay cho F7 đối với những dòng không có lời gọi hàm được khai báo trong chương trình. Sự khác nhau giữa F7 và F8 chỉ xẩy ra khi trong dòng được làm sáng có lời gọi hàm được khai báo trong chương trình (xem điểm 5.1 mục §5).

2.2. Tái lập lại quá trình gỡ rối

Bấm Ctrl + F2 hoặc vào menu Run chọn mục Program reset. Khi đó bộ nhớ dùng cho việc gỡ rối sẽ được giải toả, không có dòng nào được làm sáng và kết thúc quá trình gỡ rối, để chuẩn bị cho lần gỡ rối tiếp theo.

§3. DÙNG CỬA SỔ WATCH

Lần từng bước thường được dùng kèm với việc sử dụng cửa sổ Watch để theo dõi giá trị của biến trong mỗi bước thực hiện để dễ tìm ra nguyên nhân chương trình thực hiện sai.

Để làm điều đó ta phải nhập vào các biến cần theo dõi, bằng cách chọn mục "Add watch" của menu Break/watch, hoặc có thể bấm Ctrl+F7, sau đó nhập tên biến vào tại vị trí con trỏ trong cửa sổ nhỏ "Add Watch" và bấm ↵.

Chú ý: Mỗi lần chọn chức năng này chỉ được phép nhập vào nhiều nhất một tên biến. Do vậy nếu muốn theo dõi nhiều biến thì phải chọn nhiều lần chức năng này.

Trong khi soạn thảo, nếu chưa nhìn thấy cửa sổ Watch (ở bên dưới màn hình), thì ta bấm phím F5. Khi đó trên màn hình sẽ đồng thời hiện ra cả cửa sổ soạn thảo và cửa sổ Watch, để chuyển đổi giữa hai cửa sổ ta bấm phím F6. Mỗi biến trong cửa sổ Watch được hiện trên một dòng. Khi cửa sổ Watch được chọn sẽ có một dòng được làm sáng để chỉ rằng biến đó đang được chọn, để sau đó có thể thực hiện những thao tác khác được trình bày dưới đây với biến đó. Khi cửa sổ nào đang được chọn nếu bấm phím F5, thì cửa sổ đó sẽ được mở rộng ra toàn bộ màn hình, do vậy cửa sổ còn lại sẽ bị khuất. Để hiện lại cả hai cửa sổ ta lại bấm phím F5.

Khi đã có cửa sổ Watch, và nếu chạy chương trình từng bước, thì giá trị của các biến trong cửa sổ Watch sẽ được hiện ra và thay đổi theo kết quả của từng bước thực hiện.

Các thao tác sau đây được chọn từ menu Break/watch.

Nếu muốn sửa lại tên một biến nào đó trong cửa sổ Watch, hay thay biến đó bằng biến khác, trước hết ta chọn cửa sổ Watch bằng cách bấm F6 (nếu hiện thời đang ở cửa sổ Edit), sau đó chuyển hộp sáng về dòng chứa tên biến đó, chọn chức năng "Edit watch", hoặc bấm Enter, rồi tiến hành sửa lại tên và bấm Enter.

Còn nếu muốn xoá một biến nào đó trong cửa sổ Watch, ta cũng đưa hộp sáng tới dòng chứa tên biến đó, rồi chọn chức năng "Delete watch". Nếu muốn xoá toàn bộ các biến, ta chọn chức năng "Remove all watch", mà không cần quan tâm con trỏ ở đâu.

§4. SỬ DỤNG ĐIỂM GÃY

Như trên ta đã biết, nếu dùng chức năng chạy từng bước bằng cách bấm phím F7, thì tại mỗi bước chương trình chỉ thực hiện một dòng lệnh sau đó tạm thời dừng lại. Trong khi có nhiều trường hợp ta chỉ muốn kiểm tra việc thực hiện chương trình tại một số dòng nào đó. Khi đó nếu vẫn dùng chức năng chạy từng bước bằng cách bấm phím F7 thì sẽ mất rất nhiều thời gian (đặc biệt là với những chương trình dài) vì phải đi qua từng dòng một. Để khắc phục điều đó, chương trình gỡ rối cho phép đặt trước những điểm gãy tại những dòng cần quan tâm, sau đó mỗi lần bấm Ctrl + F9 chương trình sẽ thực hiện và dừng lại ở từng điểm gãy theo thứ tự từ trên xuống dưới của các điểm gãy đã được đặt.

4.1. Cách đặt điểm gãy

Đưa con trỏ tới dòng cần đặt, bấm Ctrl-F8, hoặc Alt-B để vào menu Break/watch sau đó chọn mục Toggle Breakpoint. Khi một dòng được đặt điểm gãy, thì dòng đó sẽ được đổi thành màu đỏ, khác với màu của dòng được làm sáng để chuẩn bị thực hiện khi dùng chức năng chạy từng bước F7, hoặc F8.

Muốn xoá một điểm gãy, ta đưa con trỏ tới dòng cần xoá, sau đó cũng thực hiện như trên.

Trong chương trình có thể đặt nhiều điểm gãy. Khi có nhiều điểm gãy, chương trình thực hiện đến điểm gãy nào thì điểm gãy đó sẽ được làm sáng giống như dòng được làm sáng khi bấm F7, hoặc F8.

4.2. Dùng một điểm gãy

Khi chỉ cần dùng một điểm gãy, thì ta có thể không cần đặt theo cách trên, mà đưa ngay con trỏ tới dòng cần quan tâm, bấm phím F4, hoặc chọn chức năng "Go to Cursor" trong menu Run.

§5. GỠ RỐI HÀM

Khi có nhiều hàm được định nghĩa trong chương trình và hàm này lại gọi hàm khác, thì quá trình gỡ rối sẽ trở nên rất phức tạp. Song chương trình gỡ rối đã cung cấp một số chức năng để giảm bớt sự phức tạp đó.

5.1. Lần từng dòng so với bước qua.

Trong khi thực hiện từng bước chương trình bằng cách dùng F7, hoặc F8. Khi dòng được làm sáng (dòng đang chuẩn bị thực hiện) có chứa lời gọi hàm: Nếu muốn kiểm tra thực hiện từng dòng trong định nghĩa hàm, thì ta phải bấm phím F7. Còn nếu muốn thực hiện luôn cả hàm đó (bước qua hàm), thì ta bấm phím F8.

5.2. Xem mục Call stack

Khi lần theo vết qua nhiều cấp của hàm, ta có thể quên mất nơi xuất phát. Để nhớ lại ta bấm Ctrl-F3, hoặc chọn mục "Call stack" trong menu Debug . Khi đó màn hình sẽ hiện ra cửa sổ nhỏ "Call stack" chứa danh sách toàn bộ các lời gọi hàm hướng đến dòng đang chuẩn bị thực hiện.

Chú ý:

- Khi ta thực hiện tiếp quá trình chạy từng bước chương trình, thì cửa sổ "Call stack" sẽ bị biến mất. Do vậy ta không thể quan sát được sự thêm bớt các hàm mới vào danh sách "Call stack" trong khi lần từng bước chương trình.

- Để chức năng Call stack làm việc được, thì mục Standard Stack Frame phải đặt là ON. Mục này đặt trong Code Generation của mục Compiler trong menu Option. Giá trị ON luôn là mặc định, nên nếu không có sự sửa lại trước đó, thì ta không cần phải làm gì cả.

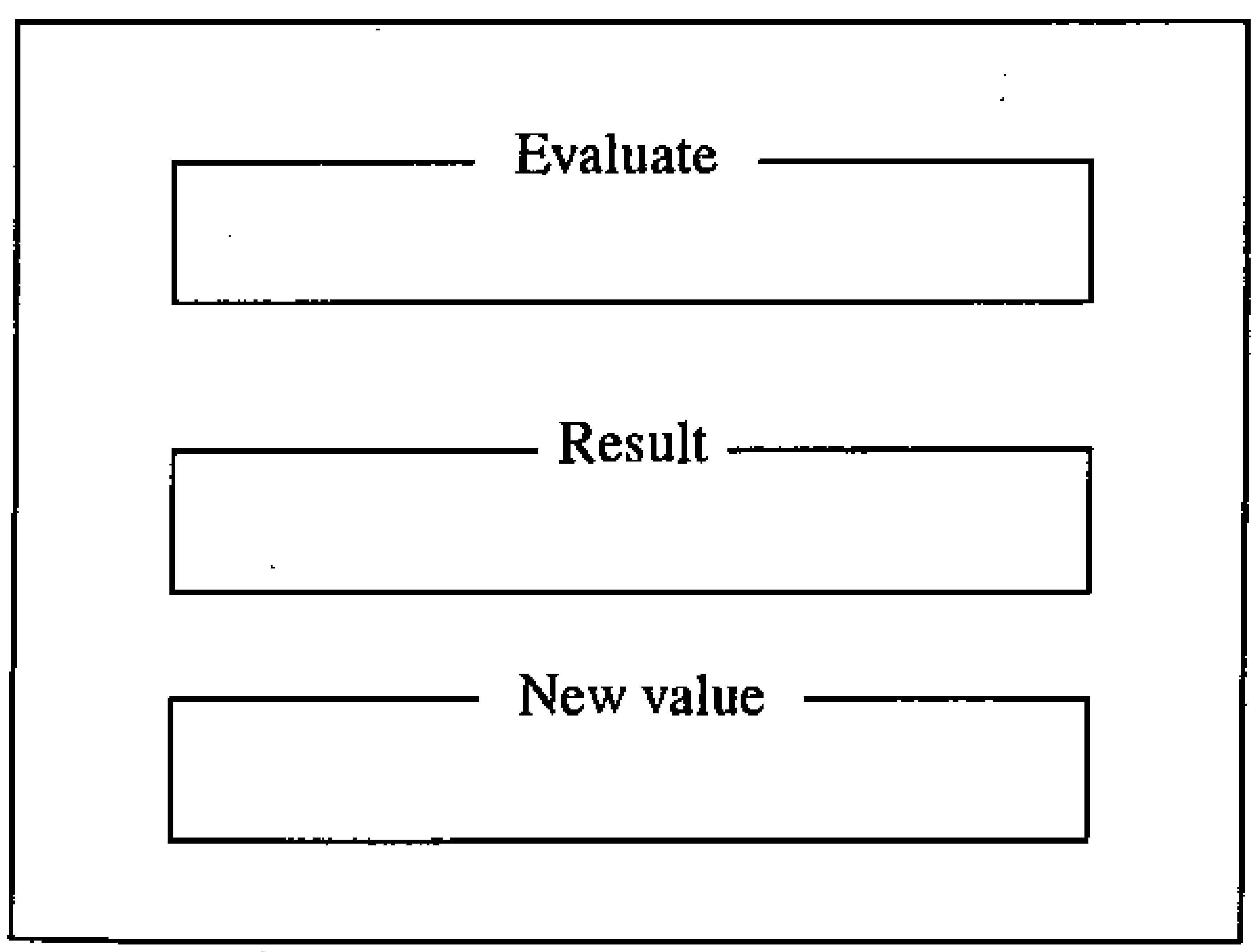
5.3. Tìm hàm

Trong các chương trình dài có định nghĩa hàm, để khỏi mất thời gian khi muốn tìm đến hàm nào đó trong chương trình. Ta chọn chức năng Find Function trong menu Debug, sau đó gõ vào tên hàm tại cửa sổ "Find Function" rồi bấm (không cần gõ các dấu mở đóng ngoặc và tham số của hàm). Con trỏ khi đó sẽ chuyển ngay về đầu nơi định nghĩa hàm.

§6. DÙNG CỬA SỔ EVALUATION

6.1. Thay đổi giá trị một biến

Trong quá trình gỡ rối ta có thể làm thay đổi giá trị của các biến cần quan tâm. Để làm điều đó, chọn mục Evaluation trong menu Debug, hoặc bấm Ctrl-F4. Khi đó trên màn hình sẽ hiện ra cửa sổ sau:



Vào tên biến tại ô Evaluate (chỉ được nhập một tên biến), rồi bấm ↵. Khi đó trong ô Result sẽ hiện ra giá trị hiện tại của biến đó, tiếp theo đưa con trỏ tới ô New value để vào giá trị mới cho biến, rồi bấm ↵.

Để ra khỏi cửa sổ trên, bấm phím Esc, hoặc cứ thực hiện tiếp các chức năng gỡ rối, thì cửa sổ trên sẽ tự tắt.

6.2. Tính biểu thức

Ta cũng có thể nhập biểu thức vào ô Evaluate rồi bấm phím ↵ thì giá trị của biểu thức sẽ được hiện ra trong ô Result. Nhưng khi đó không được vào giá trị mới cho biểu thức tại ô New value, vì chương trình dịch sẽ không hiểu được sự thay đổi này.

Có thể nhập vào tất cả các loại biểu thức trong ô Evaluate. Trong biểu thức có thể chứa một hay nhiều biến và nhiều loại toán tử. Song sẽ không xử lý được nếu trong biểu thức có các hàm như sin(), sqrt(), hay bất kỳ thứ gì được tạo bằng #define hay typedef.

PHỤ LỤC 10

CÁC MÔ HÌNH BỘ NHỚ

§1. BẢN ĐỒ BỘ NHỚ CỦA CHƯƠNG TRÌNH

Bộ nhớ chương trình gồm 3 vùng chính là:

+ vùng mã (gồm các câu lệnh và các hằng) do thanh ghi đoạn CS trỏ tới.

+ vùng dữ liệu (gồm các biến tĩnh và biến toàn bộ) do thanh ghi đoạn DS trỏ tới.

+ vùng ngăn xếp (chứa các biến cục bộ) do thanh ghi đoạn SS trỏ tới.

Ngoài ra có thanh ghi đoạn ES (Extra Segment) được dùng để truy nhập tới dữ liệu bên ngoài 3 vùng nói trên (ví dụ như vùng nhớ cấp phát động).

Với các chương trình lớn mỗi vùng nằm trong các đoạn (64 KB) khác nhau và các con trỏ CS, DS, SS trỏ tới các đoạn khác nhau. Với các chương trình nhỏ, có thể đặt cả 3 vùng mã, dữ liệu và ngăn xếp trong một đoạn, khi đó cả 3 thanh ghi CS, DS, SS đều trỏ tới cùng một đoạn.

Trong C có 6 mô hình bộ nhớ, mỗi mô hình có một giới hạn về độ lớn cho mỗi vùng: mã, dữ liệu và ngăn xếp. Nó cũng xác định cách tham chiếu đến mỗi vùng.

Như vậy việc chọn mô hình bộ nhớ phụ thuộc vào kích thước chương trình. Ví dụ đối với chương trình rất nhỏ có thể dùng mô hình tiny, chương trình rất lớn thì nên dùng mô hình huge.

§2. CON TRỎ GẦN VÀ XA

Vì kiến trúc phân đoạn của nó nên 8086 có hai kiểu địa chỉ bộ nhớ: gần (near) và xa (far).

Thanh ghi CS luôn luôn trỏ tới đoạn mã hiện thời. Một thủ tục gần phải nằm trong đoạn mã hiện thời. Chỉ thị gọi thủ tục gần chỉ đòi hỏi một địa chỉ 16 bit như là đối số của nó. Ví dụ nếu thanh ghi CS chứa 2000 thì mã lệnh sau sẽ gọi thủ tục ở 2000:0100

CALL NEAR 0100H; Gọi thủ tục ở CS:0100

Một thủ xa có thể được đặt ở bất kỳ đâu trong bộ nhớ. Để gọi một thủ tục xa cần phải chỉ rõ cả địa chỉ đoạn và độ lệch. Ví dụ để gọi một thủ tục tại 6000:0200 ta dùng mã lệnh

CALL FAR 6000H:0200H

Chỉ thị này đổi thanh ghi CS tới 6000 và thực hiện thủ tục. Khi ra khỏi thủ tục thì nội dung của CS cần được khôi phục.

Trong C một con trỏ gần (near) tới hàm chỉ cần 2 byte để chứa địa chỉ độ lệch của hàm (mà nó trỏ tới). Con trỏ xa (far) cần 4 byte để chứa địa chỉ đoạn và độ lệch của hàm mà nó trỏ tới. Như vậy có hai cách tham chiếu tới vùng mã: tham chiếu gần dùng con trỏ gần và tham chiếu xa dùng con trỏ xa.

Tương tự có 2 cách tham chiếu đến vùng dữ liệu, chỉ có khác là vai trò của CS được thay bằng DS. Trong C một con trỏ dữ liệu gần có kích cỡ 2 byte, một con trỏ xa có kích cỡ 4 byte. Ví dụ khi khai báo:

```
char near *p_gan;
char far *p_xa;
```

thì

```
sizeof(p_gan) = 2
sizeof(p_xa) = 4
```

Khi không chỉ rõ near hay far thì tính chất của con trỏ (gần hay xa) được quy định bởi mô hình bộ nhớ được chọn.

Tóm lại mô hình bộ nhớ quy định 2 điểm quan trọng là:

+ Độ lớn của các vùng mã, dữ liệu, ngăn xếp.

+ Kiểu con trỏ (gần hay xa) mặc định ứng với mỗi vùng. Ngoài ra nó còn quy định độ lớn của vùng nhớ phân chia động.

§3. CÁC MÔ HÌNH BỘ NHỚ C CHO 8086

3.1. Đặc trưng của các mô hình được tóm lược trong bảng sau:

	Tiny	Small	Medium	Compact	Large	Huge
Vùng mã	64K	64K	1MB	64K	1MB	1MB
Vùng dữ liệu	64K	64K	64K	64K	64K	1MB
Vùng ngăn xếp	64K	64K	64K	64K	64K	64K
Phân chia động	64K	64K	64K	1MB	1MB	1MB
Bộ nhớ phân chia động ngầm định	Gần	Gần	Gần	Xa	Xa	Xa
Bộ nhớ phân chia động xa	Không	Có	Có	Ngầm định	Ngầm định	Ngầm định
Con trỏ mã	Gần	Gần	Xa	Gần	Xa	Xa
Con trỏ dữ liệu	Gần	Gần	Gần	Xa	Xa	Xa

Chú ý 1. Các giới hạn khác:

Mô hình Giới hạn đặc biệt

Tiny	Mã + Dữ liệu + Ngăn xếp + Phân chia động được kết hợp lại phải nhỏ hơn 64K
Small	Dữ liệu + Ngăn xếp + Phân chia động được kết hợp lại phải nhỏ hơn 64K

Chú ý 2.

+ Trong mô hình small và medium, C chia bộ nhớ phân chia động thành 2 phần: gần và xa. Các con trỏ dữ liêu ngầm định là gần. Như vậy một lệnh gọi malloc sẽ trả lại một con trỏ gần. Lệnh gọi phiên bản xa của hàm này (farmalloc) sẽ trả lại một con trỏ xa.

+ Trong các mô hình khác chỉ có một kiểu phân chia động.

3.2. Các chi tiết mô hình bộ nhớ

+ **Tiny.** Tất cả dữ liệu và mã được gán trong một đoạn. Các thanh ghi đoạn CS, DS và SS đều trỏ tới cùng một đoạn. Đây là mô hình nhỏ nhất có thể. Các con trỏ gần được sử dụng cho mọi thứ. Bằng cách dùng lựa chọn dòng lệnh

  tcc -lt -mt tên_tệp_chương_trình_nguồn

  có thể tạo nên một tệp chương trình COM.

  Đây là mô hình lý tưởng cho các chương trình tiện ích thường trú.

+ **Small.** Dữ liệu và mã ở trong các đoạn riêng biệt. Ngăn xếp và bộ nhớ cấp phát động được gộp với dữ liệu. Mô hình này cho 64K với mã và 64 K với dữ liệu các kiểu (dữ liệu, ngăn xếp và cấp phát động).

  Các con trỏ gần được sử dụng thông suốt, tuy nhiên nếu cần có thể dùng bộ nhớ phân chia động xa. Khi đó cần khai báo rõ con trỏ far.

+ **Medium.** Mô hình này được thiết kế cho chương trình với nhiều mã và ít dữ liệu. Có một giới hạn 64K của mã cho mỗi tệp, tuy nhiên các tệp có thể liên kết để tạo thành một chương trình có vùng mã lơn hơn 64K.

  Dữ liệu, ngăn xếp và bộ nhớ cấp phát động được kết hợp với nhau phải nhỏ hơn 64K.

+ **Compact.** Kích thước mã phải nhỏ hơn 64K. Nhưng dữ liệu, ngăn xếp và bộ nhớ cấp phát động đều có các đoạn riêng của chúng.

Trong mô hình này con trỏ gần được dùng cho mã vàcon trỏ xa cho dữ liệu.

+ **Large**. Mô hình này tương tự như mô hình compact trừ một điều là kích thước mã có thể vượt quá 64K và con trỏ mã là con trỏ xa. Hầu hết các chương trình lớn đòi hỏi mô hình này.

+ **Huge**. Đây là mô hình bộ nhớ lớn nhất có thể. Các con trỏ xa được sử dụng cho mã và dữ liệu. Tuy nhiên vẫn bị hạn chế 64K cho ngăn xếp.

Đây là mô hình hữu ích cho các chương trình sử dụng một lượng lớn cả mã và dữ liệu.

3.3. Trộn lẫn 2 kiểu con trỏ gần và xa trong chương trình

Có thể dùng cả hai loại con trỏ gần và xa trong một chương trình bằng cách thêm vào từ khoá near và far. Khi đó ta đã bỏ qua kiểu con trỏ ngầm định được quy định bởi mô hình bộ nhớ. Nói chung nên hạn chế dùng trộn lẫn 2 kiểu con trỏ trong một chương trình vì dễ gây lỗi.

§4. KIỂM TRA KÍCH THƯỚC CHƯƠNG TRÌNH

Bây giờ chúng ta đã thấy cách sắp đặt bộ nhớ của một chương trình. Một câu hỏi đặt ra là: Điều chỉnh bộ nhớ chương trình như thế nào? Kích thước của vùng mã và vùng dữ liệu được quyết định bởi chương trình dịch.

Kích thước ngăn xếp ngầm định là 4K và kích thước vùng cấp phát động ngầm định càng lớn càng tốt. C cung cấp hai biến _stklen và _heaplen (khai báo trong dos.h) dùng để xác định độ lớn của ngăn xếp và bộ nhớ cấp phát động.

Ví dụ để có kích cỡ tối thiểu cho ngăn xếp và vùng cấp phát động ta có thể dùng các câu lệnh:

```
#include <dos.h>
unsigned _stklen = 1024;
unsigned _heaplen = 2;
```

Để có độ lớn tối đa cho hai vùng trên ta dùng các câu lênh:

```
#include <dos.h>
unsigned _stklen = 0xFFFF; /* 64K */
unsigned _heaplen = 0; /* Càng lớn càng tốt */
```

PHỤ LỤC 11

TÓM TẮT CÁC HÀM CỦA TURBO C THEO THỨ TỰ ABC

1.	_chmod	<io.h> chg10, §11
2.	_close	<io.h> chg10, §11
3.	_creat	<io.h> chg10, §11
4.	_open	<io.h> chg10, §11
5.	abort	<process.h> phụ lục 2, §10
6.	abs	<stdlib.h> phụ lục 2, §6
7.	acos	<math.h> phụ lục 2, §6
8.	arc	<graphics.h> chg9, §5
9.	asin	<math.h> phụ lục 2, §6
10.	atan	<math.h> phụ lục 2, §6
11.	atan2	<math.h> phụ lục 2, §6
12.	atof	<ctype.h> phụ lục 2, §2
13.	atoi	<ctype.h> phụ lục 2, §2
14.	atol	<ctype.h> phụ lục 2, §2
15.	bar	<graphics.h> chg9, §5
16.	bar3d	<graphics.h> chg9, §5
17.	cabs	<math.h> phụ lục 2, §6
18.	calloc	<alloc.h> phụ lục 2, §8
19.	ceil	<math.h> phụ lục 2, §6
20.	chdir	<dir.h> phụ lục 2, §9
21.	chmod	<io.h> chg10, §11
22.	circle	<graphics.h> chg9, §5
23.	cleardevive	<graphics.h> chg9, §7
24.	clearviewport	<graphics.h> chg9, §7
25.	close	<io.h> chg10, §11

26.	clreol	<conio.h> chg8, §5
27.	clrscr	<conio.h> chg4, §7 và chg8, §5
28.	coreleft	<alloc.h> phụ lục 2, §8
29.	cos	<math.h> phụ lục 2, §6
30.	cosh	<math.h> phụ lục 2, §6
31.	cprintf	<conio.h> chg8, §4
32.	creat	<io.h> chg10, §10
33.	cscanf	<conio.h> chg8, §4
34.	delay	<dos.h> chg16, §1
35.	delline	<conio.h> chg8, §5
36.	disable	<dos.h> chg15, §2
37.	drawpoly	<graphics.h> chg9, §5
38.	ecvt	<ctype.h> phụ lục 2, §2
39.	ellipse	<graphics.h> chg9, §5
40.	enable	<dos.h> chg15, §2
41.	exit	<process.h> phụ lục 2, §10
42.	exp	<math.h> phụ lục 2, §6
43.	fabs	<math.h> phụ lục 2, §6
44.	fclose	<stdio.h> chg10, §3
45.	fcloseall	<stdio.h> chg10, §3
46.	fcvt	<ctype.h> phụ lục 2, §2
47.	feof	<stdio.h> chg10, §3
48.	ferror	<stdio.h> chg10, §3
49.	fflush	<stdio.h> chg4, §5 và chg10, §3
50.	fflushall	<stdio.h> chg10, §3
51.	fgetc	<stdio.h> chg10, §4
52.	fgets	<stdio.h> chg10, §5
53.	fillpopy	<graphics.h> chg9, §5
54.	findfirst	<dir.h> phụ lục 2, §9
55.	findnext	<dir.h> phụ lục 2, §9

116. isspace	<ctype.h> phụ lục 2, §3
117. isupper	<ctype.h> phụ lục 2, §3
118. itoa	<ctype.h> phụ lục 2, §2
119. kbhit	<conio.h> chg4, §7
120. keep	<dos.h> chg15,§4
121. labs	<stdlib.h> phụ lục 2, §6
122. line	<graphics.h> chg9, §5
123. linerel	<graphics.h> chg9, §5
124. lineto	<graphics.h> chg9, §5
125. log	<math.h> phụ lục 2, §6
126. log10	<math.h> phụ lục 2, §6
127. lseek	<io.h> chg10, §12
128. ltoa	<ctype.h> phụ lục 2, §2
129. malloc	<alloc.h> chg9, §10 và phụ lục 2, §8
130. memccpy	<memory.h> hoặc <string.h> phụ lục 2, §5
131. memchr	<memory.h> hoặc <string.h> phụ lục 2, §5
132. memcmp	<memory.h> hoặc <string.h> phụ lục 2, §5
133. memcpy	<memory.h> hoặc <string.h> phụ lục 2, §5
134. memicmp	<memory.h> hoặc <string.h> phụ lục 2, §5
135. memset	<memory.h> hoặc <string.h> phụ lục 2, §5
136. MK_FP	<dos.h> chg13,§1 và chg 14,§2
137. mkdir	<dir.h> phụ lục 2, §9
138. movedata	<mem.h> chg 14,§1
139. movedata	<memory.h> hoặc <string.h> phụ lục 2, §5
140. moveto	<graphics.h> chg9, §5
141. nosound	<dos.h> chg16,§1
142. open	<io.h> chg10, §11
143. outtext	<graphics.h> chg9, §9
144. outtextxy	<graphics.h> chg9, §9
145. peek	<dos.h> chg14,§1

146. peekb <dos.h> chg14,§1

147. perror <stdio.h> chg10, §3

148. pieslice <graphics.h> chg9, §5

149. poke <dos.h> chg14,§1

150. pokeb <dos.h> chg14,§1

151. pow <math.h> phụ lục 2, §6

152. printf <stdio.h> chg4, §1

153. putc <stdio.h> chg10, §4

154. putch <conio.h> chg4, §7

155. putchar <stdio.h> chg4, §6

156. putimage <graphics.h> chg9, §10

157. putpixel <graphics.h> chg9, §8

158. puts <stdio.h> chg4, §6

159. putw <stdio.h> chg10, §7

160. rand <stdlib.h> phụ lục 2, §6

161. random <stdlib.h> phụ lục 2, §6

162. randomize <stdlib.h> và <time.h> phụ lục 2, §6

163. read <io.h> chg10, §12

164. realloc <alloc.h> phụ lục 2, §8

165. rectangle <graphics.h> chg9, §5

166. remove <stdio.h> chg10, §3

167. rewind <stdio.h> chg10, §8

168. rmdir <dir.h> phụ lục 2, §9

169. scanf <stdio.h> chg4, §2

170. segread <dos.h> chg13,§3

171. setbkcolor <graphics.h> chg9, §4

172. setcolor <graphics.h> chg9, §4

173. setdate <time.h> phụ lục 2, §7

174. setfillstyle <graphics.h> chg9, §4

175. setlinestyle <graphics.h> chg9, §6

176. setpalette	<graphics.h> chg9, §4
177. settextjustify	<graphics.h> chg9, §9
178. settextstyle	<graphics.h> chg9, §9
179. settime	<time.h> phụ lục 2, §7
180. setvect	<dos.h> chg15,§2
181. setviewport	<graphics.h> chg9, §7
182. setwritemode	<graphics.h> chg9, §6
183. sin	<math.h> phụ lục 2, §6
184. sinh	<math.h> phụ lục 2, §6
185. sleep	<dos.h> chg16,§1
186. sound	<dos.h> chg16,§1
187. sprintf	<stdio.h> phụ lục 2, §2
188. sqrt	<math.h> phụ lục 2, §6
189. srand	<stdlib.h> phụ lục 2, §6
190. strcat	<string.h> phụ lục 2, §4
191. strchr	<string.h> phụ lục 2, §4
192. strcmp	<string.h> phụ lục 2, §4
193. strcmpi	<string.h> phụ lục 2, §4
194. strcpy	<string.h> phụ lục 2, §4
195. strcspn	<string.h> phụ lục 2, §4
196. strdup	<string.h> phụ lục 2, §4
197. stricmp	<string.h> phụ lục 2, §4
198. strlen	<string.h> phụ lục 2, §4
199. strlwr	<string.h> phụ lục 2, §4
200. strncat	<string.h> phụ lục 2, §4
201. strncmp	<string.h> phụ lục 2, §4
202. strncpy	<string.h> phụ lục 2, §4
203. strnicmp	<string.h> phụ lục 2, §4
204. strnset	<string.h> phụ lục 2, §4
205. strpbrk	<string.h> phụ lục 2, §4

PHỤ LỤC 12

HÀM VỚI ĐỐI SỐ BẤT ĐỊNH TRONG C

Trong các giáo trình C thường chỉ hướng dẫn cách xây dựng hàm với các đối cố định. Mỗi đối cần có một tham số (cùng kiểu với nó) trong lời gọi hàm. Tuy nhiên một vài hàm chuẩn của C lại không như vậy, mà linh hoạt hơn, chẳng khi dùng hàm printf hay scanf thì số tham số mà ta cung cấp cho hàm là không cố định cả về số lượng lẫn kiểu cách. Ví dụ trong câu lệnh:

```
printf("\n Tổng = %d " , 3+4+5) ;
```

có 2 tham số, nhưng trong câu lệnh:

```
printf("\n Hà Nội" ) ;
```

chỉ có một tham số.

Như vậy cần phân biệt các khái niệm sau:

- Đối số cố định được khai báo trong dòng đầu của hàm, nó có tên và kiểu
- Tham số ứng với đối số cố định gọi là tham số cố định
- Đối bất định được khai báo bởi ba dấu chấm: bất định cả về số lượng và kiểu
- Tham số bất định (ứng với đối bất định) là một danh sách giá trị với số lượng và kiểu tuỳ ý (không xác định)

Trong phụ lục này sẽ trình bầy cách xây dựng các hàm với đối số bất định. Công cụ chủ yếu được dùng là con trỏ và danh sách.

1. Biến con trỏ

Biến con trỏ (hay con trỏ) dùng để chứa địa chỉ của biến, mảng, hàm, ... Có nhiều kiểu địa chỉ, vì vậy cũng có nhiều kiểu con trỏ. Biến con trỏ được khai báo theo mẫu:

```
Kiểu   *Tên_biến_con_trỏ ;
```

Ví dụ:

```
float  px ;  // px là con trỏ thực
```

Các phép toán quan trọng trên con trỏ gồm:

+ Gán địa chỉ một vùng nhớ cho con trỏ (dùng toán tử gán, phép lấy địa chỉ, các hàm cấp phát bộ nhớ)

+ Truy nhập vào vùng nhớ thông qua con trỏ, dùng phép toán:

*Tên_con_trỏ

(Để ý ở đây có 2 vùng nhớ: vùng nhớ của biến con trỏ và vùng nhớ mà địa chỉ đầu của nó chứa trong biến con trỏ)

+ Cộng địa chỉ để con trỏ chứa địa chỉ của phần tử tiếp theo, dùng phép toán:

++ Tên_con_trỏ hoặc Tên_con_trỏ ++

Chú ý: các phép toán trên chỉ có thể thực hiện đối với con trỏ có kiểu.

2. Danh sách không cùng kiểu

Dùng con trỏ có kiểu chỉ quản lý được một danh sách giá trị cùng kiểu, ví dụ dãy số thực, dãy số nguyên, dãy các cấu trúc,....

Khi cần quản lý một danh sách các giá trị không cùng kiểu ta phải dùng con trỏ không kiểu (void) khai báo như sau:

void * Tên_con_trỏ ;

Con trỏ void có thể chứa các địa chỉ có kiểu bất kỳ, và dùng để trỏ đến vùng nhớ chứa danh sách cần quản lý. Một chú ý quan trọng là mỗi khi gửi vào hay lấy ra một giá trị từ vùng nhớ, thì tuỳ theo kiểu giá trị mà ta phải dùng phép chuyển kiểu thích hợp đối với con trỏ. Ví dụ sau minh hoạ cách lập một danh sách gồm một số nguyên, một số thực và một chuỗi ký tự. Chúng ta cần một bộ nhớ để chứa số nguyên, số thực và địa chỉ chuỗi và dùng các con trỏ void để quản lý vùng nhớ này.

```
void *list , *p ;  // Con trỏ list trỏ tới đầu danh sách
                   // p dùng để duyệt qua các phần tử của danh sách
list=malloc(sizeof(int) + sizeof(float)+ sizeof(char*) );
 p=list;
 *((int*)p) = 12;        // Đưa số nguyên 12 vào danh sách
 ((int*)p)++ ;           // Chuyển sang phần tử tiếp theo
 *((float*)p) = 3.14;    // Đưa số thực 3.14 vào danh sách
 ((float*)p)++ ;         // Chuyển sang phần tử tiếp theo
 *((char**)p) = "HA NOI"; // Đưa địa chỉ chuỗi "HA NOI"
                          // vào danh sách
                          // Nhận các phần tử trong danh sách
 p=list;                 // Về đầu danh sách
```

```
int a = *((int*)p);        // Nhận phần tử thứ nhất
((int*)p)++ ;              // Chuyển sang phần tử tiếp theo
float  x= *((float*)p);  // Nhận phần tử thứ hai
((float*)p)++ ;            // Chuyển sang phần tử tiếp theo
char *str  =  *((char**)p) ; // Nhận phần tử thứ ba
```

3. Hàm với đối số bất định

+ Đối bất định bao giờ cũng đặt sau cùng và được khai báo bằng dấu ba chấm. Ví dụ ví dụ hàm

```
void f(int n, char *s, ...) ;
```

có 2 đối cố định là n, s và đối bất định.

+ Để nhận được các tham số bất định trong lời gọi hàm ta cần lưu ý các điểm sau:

- Các tham số bất định chứa trong một danh sách. Để nhận được địa chỉ đầu danh sách ta dùng một con trỏ void và phép gán sau:

```
void  *list ;
list = ...  ;
```

- Dùng một tham số cố định kiểu chuỗi để quy định số lượng và kiểu của mỗi tham số trong danh sách, ví dụ:

"3i"   hiểu là : tham số bất định gồm 3 giá trị int

"3f"   hiểu là : tham số bất định gồm 3 giá trị float

"fiss" hiểu là có 4 tham số bất định có kiểu lần lượt là float, int, char*, char*

Một khi đã biết được địa chỉ đầu danh sách, biết được số lượng và kiểu của mỗi tham số , thì dễ dàng nhận được giá trị các tham số để sử dụng trong thân hàm.

Ví dụ sau đây minh hoạ cách xây dựng các hàm với tham số bất định. Hàm dùng để in các giá trị kiểu int, float và char. Hàm có một tham số cố định để cho biết có bao nhiêu giá trị và kiểu các giá trị cần in. Kiểu quy định như sau: i là int, f là float, s là char*. Tham số có 2 cách viết: lặp (gồm một hằng số nguyên và một chữ cái định kiểu) và liệt kê (một dãy các chữ cái định kiểu). Ví dụ:

"4s"   có nghĩa in 4 chuỗi

"siif" có nghĩa in một chuỗi, 2 giá trị nguyên và một giá trị thực:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <stdarg.h>
void InDanhSachGiaTri(char *st,...)
  {
    void *list ;
    int gt_int ;
    float gt_float;
    char *gt_str;
    int n,i ;
    char kieu;
    int lap;
    list = ... ;  // list tro toi vung nho chua danh sach dia chi cac
                  // tham so
    lap = isdigit(st[0])  ;
    if (lap)
          n=st[0] - '0' ;
    else
          n=strlen(st);
    printf("\n n= %d lap = %d",n,lap);
    getch();
    for(i=0;i<n;++i)
      {
        if(lap)
          kieu=st[1];
        else
          kieu = st[i];
        printf("\nKieu= %c",kieu);
        getch();
        switch(kieu)
```

```
{
    case 'i' :
    gt_int =  *((int*)list);
    if(!lap)
      ((int*)list)++ ;
      printf("\nGia tri %d = %d",i,gt_int);
      break;
   case 'f' :
    gt_float = (float) (*((double*)list));
   if(!lap)
    ((double*)list)++  ;
    printf("\nGia tri %d = %0.2f",i,gt_float);
    break;
   case 's' :
    gt_str = *((char**)list) ;
   if(!lap)
    ((char**)list)++  ;
    printf("\nGia tri %d = %s",i,gt_str);
 }
 }
}
void main()
{
    float x=3.14;
    int a=123;
    char *tp="HAI PHONG";
    InDanhSachGiaTri("4i",a);
    InDanhSachGiaTri("4s","HA NOI");
    InDanhSachGiaTri("ifsssffii", a, x, tp, tp,"QUY NHON",
    x, 6.28, a, 246);
    InDanhSachGiaTri("4f",6.28);
    getch();
}
```

4. Hàm không đối và hàm với đối bất định

Nhiều người nghĩ hàm khai báo như sau

```
void f();
```

là hàm không đối trong C. Trong C++ thì hiểu như thế là đúng, còn trong C thì đó là hàm có đối bất định (hàm không đối trong C khai báo như sau: f(void)). Do không có đối cố định nào cho biết về số lượng và kiểu của các tham số bất định, nên giải pháp ở đây là dùng các biến toàn bộ. Rõ ràng giải pháp này không không thuận tiện cho người dùng vì phải khai báo đúng tên biến toàn bộ và phải khởi gán giá trị cho nó trước khi gọi hàm. Ví dụ trình bầy một hàm chỉ có đối bất định dùng để tính max và min của các giá trị thực. Các tham số bất định được đưa vào theo trình tự sau: Địa chỉ chứa max, địa chỉ chứa min, các giá trị nguyên cần tính max, min. Chương trình dùng biến toàn bộ N để cho biết số giá trị nguyên cần tính max, min.

```
int N;
void maxmin()
  {
    void *lt = ... ;
    float *max, *min , tg;
    int i;
    max = *((float**)lt)++;
    min = *((float**)lt)++;
    *max = *min = (float) *((double*)lt)++;
    for(i=1;i<N;++i)
      {
        tg= (float) *((double*)lt)++;
        if(tg > *max) *max = tg;
        if(tg < *min) *min = tg;
      }
  }
```

PHỤ LỤC 13

MỘT SỐ CHƯƠNG TRÌNH HỮU ÍCH

Trong phụ lục này trình bầy một số hàm thường cần dùng đến khi viết chương trình, đó là các hàm tạo lập menu và vẽ đồ thị hàm số. Các chương trình trên có dùng tới phím ENTER và các phím mũi tên, vì vậy đầu tiên chúng ta sẽ nói về kỹ thuật bắt phím.

§1. KỸ THUẬT BẮT PHÍM

Khi bấm một phím hay một tổ hợp phím ta nhận được một hoặc hai mã. Ví dụ khi bấm phím A ta nhận được mã 65, phim F1 được hai mã là 0 và 59 (viết là 0/59), tổ hợp phím Ctrl - Page Down cho mã 0/118. Để ý khi có hai mã thì mã thứ nhất bao giờ cũng là 0. Để bắt được một hay một tổ hợp phím bất kỳ ta phải dùng hai biến nguyên, hai lần thực hiện hàm getch theo mẫu sau :

```
int ch1, ch2;
ch1 = getch();
   If (ch1 == 0)
      ch2 = getch();
```

Chương trình sau minh họa cách dùng phím End (mã 0/79). Chương trình sẽ liên tục réo chuông cho tới khi bấm phím End thì mới kết thúc.

```
#include "stdio.h"
#include "conio.h"
 void main()
  {
   int ch1, ch2;
   clrscr();
   tt: gotoxy(1,1);
   putchar(7) ;
  if(! kbhit())
     goto tt;
  ch1 = getch();
  If (ch1 == 0)
     ch2 = getch();
  if ( ! (ch1 ==0 && ch2 == 79) )
 goto tt;
}
```

Dưới đây là 3 bảng mã của các phím và các tổ hợp phím.

Bảng 1 nói về các phím đặc biệt như Esc, Home, End, Page Up, ...

Bảng 2 gồm 4 cột trong đó cột đầu ghi các phím F1, ... , F10, Cột thứ hai ghi mã của các phím tương ứng, Cột thứ ba là mã của các tổ hợp: Ctrl+F1, ..., Ctrl+F10, Cột thứ tư là mã của Alt+F1,..., Alt+F10.

Bảng 3 gồm ba cột, cột đầu ghi các chữ cái từ A đến Z, cột hai là mã của 26 tổ hợp: Ctrl+A, ... , Ctrl+Z, còn cột thứ ba là mã của Alt+A, ... , Alt+Z.

Bảng 1

Phím	Mã
Esc	27
Tab	9
Shift+Tab	0/15
Enter	13
BackSpace	8
Insert	0/82
Home	0/71
Ctrl+Home	0/119
End	0/79
Ctrl + End	0/117
Page Up	0/73
Ctrl + Page Up	0/132
Page Down	0/81
Ctrl + Page Down	0/118
Delete	0/83
↑	0/72
↓	0/80
←	0/75
→	0/77
Ctrl + ←	0/115
Ctrl + →	0/116

Bảng 2

Phím	Mã phím	Mã Ctrl+phím	Mã Alt+phím
F1	0/59	0/94	0/104
F2	0/60	0/95	0/105

F3	0/61	0/96	0/106
F4	0/62	0/97	0/107
F5	0/63	0/98	0/108
F6	0/64	0/99	0/109
F7	0/65	0/100	0/110
F8	0/66	0/101	0/111
F9	0/67	0/102	0/112
F10	0/68	0/103	0/113

Bảng 3

Chữ cái	Mã CTRL+Chữ cái	Mã ALT+Chữ cái
A	1	0/30
C	2	0/48
B	3	0/46
D	4	0/32
E	5	0/18
F	6	0/33
G	7	0/34
H	8	0/35
I	9	0/23
J	10	0/36
K	11	0/37
L	12	0/38
M	13	0/50
N	14	0/49
O	15	0/24
P	16	0/25
Q	17	0/16
R	18	0/19
S	19	0/31
T	20	0/20
U	21	0/22
V	22	0/47
W	23	0/17
X	24	0/45
Y	25	0/21
Z	26	0/44

§2. LẬP MENU MỘT MỨC TRONG MỐT VĂN BẢN

Hàm menu1 dùng để tạo menu một mức được khai báo như sau:

```
void menu1(int n, int hd, int cd, int dai, char nd[][40],
    int mnc, int mcc, int mnr, int mcr, int *ch)
```

Ý nghĩa các đối:

n là số chức năng

hd, cd, dai là hàng đầu, cột đầu và độ dài của khung menu

nd là một mảng 2 chiều chứa nội dung (tên) các chức năng

mnc, mcc là mầu nền và mầu chữ chung cho các chức năng

mnr, mcr là mầu nền và mầu chữ riêng cho các chức năng được chọn.

ch trỏ tới một biến nguyên chứa số hiệu chưc năng được chọn (từ 1 đến n)

Trong hàm menu1 có gọi tới hàm hienkt. Chương trình dưới đây minh họa cách dùng hàm menu1 để xây dựng menu một mức gồm 3 chức năng : TAM GIAC, HINH TRON và KET THUC. Để đi đến một chức năng nào đó ta dùng các phím mũi tên lên hoặc mũi tên xuống. Để chọn chức năng ta dùng phím ENTER (sau khi đưa hộp sáng đến chức năng này).

```
/* Menu một mức trong mode văn bản */
        #include "stdio.h"
        #include "conio.h"
        void hienkt (char *dkt, int h, int c, int dai, int mn, int mc)
        {
           textbackground(mn);
           textcolor(mc);
           window(c, h, c+dai - 1, h) ;
           clrscr();
              /* Cắt bớt ký tự nếu cần */
           if( strlen(dkt) >= dai)
             dkt[dai] = 0;
             puts(dkt);
             gotoxy(1,1);
        }
void menu1(int n, int hd, int cd, int dai, char nd[][40],
        int mnc, int mcc, int mnr, int mcr, int *ch)
        {
           int i, chonm, mn, mc;
           int ch1, ch2, chon;
```

```
chon = *ch;
 /* Chỉnh lý biến chon và kiểm tra n */
if (n <= 0 || n >= 20)  return ;
if (chon < 0 || chon > n)  chon=1;
    /* Trình bầy */
chonm = chon;
for (i=1 ; i <= n; ++i)
 hienkt(nd[i], hd+i-1, cd, dai, mnc, mcc);
 hienkt(nd[chonm], hd+chonm-1, cd, dai, mnr, mcr);
   /* Bắt phím, chọn chức năng */
ch1 = 0;
while (ch1 != 13)
 {
  ch1 = getch();
  if (ch1 == 0)
    ch2 = getch();
  if (ch1 == 0)
   if (ch2 == 72)
    {
      chonm = chon-1;
      if (chonm <= 0)
        chonm = n;
    }
  else
   if (ch2 == 80)
     {
      chonm = chon+1;
      if (chonm > n)  chonm =1;
     }
 if (chonm != chon)
  {
   hienkt(nd[chon], hd+chon-1, cd, dai, mnc, mcc);
   hienkt(nd[chonm],hd+chonm-1, cd, dai, mnr, mcr);
   chon= chonm;
  }
 }
*ch = chon;
}
```

```
/* Hàm main */
void main()
  {
    char nd[4][40]
    int chon;
    strcpy(nd[1], "TAM GIAC");
    strcpy(nd[2], "HINH TRON");
    strcpy(nd[3], "KET THUC");
    textmode(C80);
    clrscr();
    chon= 1;
    while (chon != 3)
     {
       textbackground(BLUE);
       clrscr();
       menu1(3,8,35,10,nd,CYAN,YELLOW,BROWN,WHITE,&chon);
               if (chon == 1)
                 {
                   window(1,1,80,25);
                   textbackground(BLACK);
                   clrscr();
                   puts("Tam giac");
                   getch();
                 }
               if( chon == 2)
                 {
                   window(1,1,80,25);
                   textbackground(BLACK);
                   clrscr();
                   puts("Hinh tron");
                   getch();
                 }
          }
    }
```

§3. MENU HAI MỨC TRONG MỐT BẢN

Hàm menu2 dùng để tạo menu 2 mức, được khai báo như sau:

void menu2(char *td, int n, int *socn, char nd[][10][40],

int mmh, int mkhung, int mnc, int mcc, int mnr, int mcr,

int *sh_menu, int *sh_cn)

Ở đây:

td là tiêu đề của hệ menu.

n là số menu con, n cần không quá 10.

socn là mảng kiểu int chứa số chức năng của các menu con.

Mỗi menu con không được có quá 20 chức năng.

nd là mảng 2 chiều kiểu char[40] chứa nội dung sau: Mỗi cột chứa thông tin của một menu con, trong đó phần tử đầu chứa tên menu con, các phần tử khác chứa tên các chức năng của menu con.

mmh là mầu màn hình (phần bên ngoài khung chứa hệ menu).

mkhung là mầu khung của toàn hệ thống menu.

mnc, mcc là mầu nền và mầu chữ của các menu con và các chức năng.

mnr, mcr là mầu nền và mầu chữ của các menu con và chức năng được chọn.

sh_menu trỏ tới biến chứa số hiệu menu con được chọn.

sh_cn trỏ tới biến chứa số hiệu chức năng được chọn.

Trong hàm menu2 có sử dụng hàm hienkt. Chương trình dưới đây minh họa cách dùng hàm menu2 để xây dựng một hệ menu hai mức. Hệ gồm 2 menu con: KET THUC và TINH TOAN. Menu con TINH TOAN có 2 chức năng là D.T. TAM GIAC và D.T. HINH TRON. Để đi đến một menu con ta dùng các phím mũi tên trái, phải. Để đi đến một chức năng ta dùng các phim mũi tên lên, xuống. Để chọn menu con hay chức năng ta bấm phím ENTER .

```
/* Menu hai mức trong mode văn bản */
        #include "stdio.h"
        #include "conio.h"
        #include "string.h"
        void hienkt (char *dkt, int h, int c, int dai, int mn, int mc)
          {
            textbackground(mn);
            textcolor(mc);
            window(c, h, c+dai - 1, h);
            clrscr();
              /* Cắt bớt ký tự nếu cần */
```

```
    if( strlen(dkt) >= dai)
        dkt[dai] = 0;
      puts(dkt);
      gotoxy(1,1);
}
void menu2(char *td, int n, int *socn, char nd[][10][40],
                int mmh, int mkhung, int mnc, int mcc, int mnr, int mcr,
                int *sh_menu, int *sh_cn)
    {
        int vtmenu[10], ddaimenu[10];
        int vtcn[10], ddaicn[10];
        int i, j, ngang, doc, shmenum, shcnm;
        int maxdai, hd, cd, htd, ctd, xx;
        int ch1, ch2;
        char tdd[40];
        int shmenu, shcn;
        shmenu = *sh_menu;
        shcn = *sh_cn;
        textbackground(mmh);
        clrscr();
        /* Chỉnh lý shmenu và shcn */
        if (shmenu < 1 || shmenu > n)
            shmenu=1;
            j = shmenu;
        if (shcn < 1)
            shcn=1;
        if (shcn > socn[j])
            shcn=socn[j];
            shmenum= shmenu;
            shcnm= shcn;
      /* Xác định ddaimenu của các menu và bề ngang */
      ngang = 0;
      for (j= 1; j<=n; ++j)
        {
         ddaimenu[j]= strlen(nd[1][j]) + 1;
         ngang = ngang + ddaimenu[j];
        }
```

```
/* Xác định chiều dọc */
        doc= 0;
        for (j= 1; j<= n; ++j)
         if (doc < socn[j])
            doc= socn[j];
/* Xác định ddaicn của các menu con và max của chúng */
        maxdai= 0;
        for (j= 1; j<= n; ++j)
        {
         ddaicn[j] = 0 ;
         for (i = 2; i<= socn[j]+1; ++i)
          if (ddaicn[j] < strlen(nd[i][j])+1)
           ddaicn[j] = strlen(nd[i][j])+1;
          if (maxdai < ddaicn[j])
            maxdai= ddaicn[j];
        }
        /* Kiểm tra ngang, maxdai và doc */
        if (ngang>80 || maxdai>ngang || doc>20 || n>10)
          return ;
            /* Xác định hd, cd, htd, in tiêu đề */
        cd = (80-ngang) / 2 + 1;
        xx= strlen(td)+1;
        ctd= (80-xx) / 2 ;
        htd= (25-doc-3) / 2 + 1;
        hd = htd+2;
        strcpy(tdd,td);
        hienkt(tdd,htd,ctd,xx,mnc,mcc);
            /* Xác định vtmenu của các menu con*/
        for (j = 1 ; j<= n; ++j)
         if  (j == 1)
            vtmenu[j] = cd;
         else vtmenu[j] = vtmenu[j-1] + ddaimenu[j-1];
            /* Xác định vtcn của các menu con*/
        for (j = 1; j<= n; ++j)
         {
           vtcn[j]= vtmenu[j];
           xx= ddaicn[j] - ((cd+ngang-1)- vtmenu[j]);
```

```
    if (xx > 0)
     vtcn[j]= vtcn[j]-xx+1;
  }
    /* Trình bày các menu con trên một hàng */
textbackground(mkhung);
window(cd,hd+1,cd+ngang-1,hd+doc); clrscr();
for (j = 1; j<= n; ++j)
 hienkt(nd[1][j], hd, vtmenu[j], ddaimenu[j], mnc, mcc);
 j= shmenu;
 hienkt(nd[1][j], hd, vtmenu[j], ddaimenu[j], mnr, mcr);
   /* Trinh bày các chức năng của menu con được chon */
if (socn[j] > 0)
   {
    for (i=1; i<= socn[j]; ++i)
     hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnc,mcc);
    i= shcn;
    hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnr,mcr);
   }
    /*Bắt phím chọn menu con và chức năng*/
ch1 = 0 ;
while (ch1 != 13)
{
ch1= getch();
if (ch1 == 0)  ch2 = getch() ;
if (ch1 == 0)
 {
  if (ch2 == 75)  /* <-- */
   {
    shmenum = shmenu - 1 ;
    if (shmenum <= 0)
      shmenum = n ;
   }
  else if(ch2 == 77)  /* --> */
   {
    shmenum = shmenu + 1 ;
    if(shmenum < n)
      shmenum = 1;
   }
```

```
else if(ch2==72 && socn[shmenu]>0) /* trên */
 {
  shcnm = shcn - 1 ;
  if(shcnm <= 0)
    shcnm = socn[shmenu];
 }
else
  if(ch2== 80 && socn[shmenu]>0) /* Dưới */
   {
     shcnm = shcn + 1;
     if(shcnm > socn[shmenu])
      shcnm = 1;
   }
}
          /* Trình bày lại */
if (shmenum != shmenu || shcnm != shcn)
 {
  if (shmenum != shmenu)
   {
   /* Trình bày lại */
      textbackground(mkhung);
      window(cd,hd+1,cd+ngang-1,hd+doc);
      clrscr();
   for(j = 1; j<= n; ++j)
      hienkt(nd[1][j],hd,vtmenu[j],ddaimenu[j],mnc,mcc);
   j=shmenum;
   hienkt(nd[1][j],hd,vtmenu[j],ddaimenu[j],mnr,mcr);
   if (socn[j] > 0)
    {
      /* Chỉnh lý shcnm */
      if (shcnm <= 0 || shcnm > socn[j])
         shcnm= 1;
      for (i=1; i<= socn[j]; ++i)
       hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnc,mcc);
       i=shcnm;
       hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnr,mcr);
    }
   }
```

```
        else
          {
            /* Thay đổi mầu của chức năng */
            j = shmenu;
            i = shcn;
            hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnc,mcc);
            i= shcnm;
            hienkt(nd[i+1][j],hd+i,vtcn[j],ddaicn[j],mnr,mcr);
          }
          shmenu= shmenum;
          shcn= shcnm;
        }
      }          /* while */
      *sh_menu = shmenu;
      *sh_cn = shcn;
    }        /* Kết thúc hàm menu2 */
             /* Hàm main */
void main()
  {
    char nd[20][10][40];
    int socn[3];
    int shcn, shmenu;
    strcpy(nd[1][1],"KET THUC");
    socn[1] = 0;
    strcpy(nd[1][2], "TINH TOAN");
    socn[2]= 2;
    strcpy(nd[2][2], "D.T. TAM GIAC");
    strcpy(nd[3][2], "D.T HINH TRON");
    shcn= 1;
    shmenu= 1;
   ttt:
   menu2 ("THU MENU", 2, socn, nd, BLACK, BLUE, MAGENTA,
          WHITE, BROWN, YELLOW, &shmenu, &shcn);
   if (shmenu == 1)
      exit(0);
```

```
        clrscr();
        if (shcn == 1)
          {
            window(1,1,80,25);
            textbackground(BLACK);
            clrscr();
            puts("DIEN TICH TAM GIAC");
            getch();
          }
        else
          {
            window(1,1,80,25);
            textbackground(BLACK);
            clrscr();
            puts("DIEN TICH HINH TRON");
            getch();
          }
        goto ttt;
    }
```

§4. LẬP MENU MỘT MỨC TRONG MỐT ĐỒ HỌA

Trong mục này chúng ta xây dựng hai hàm: hienktdh và menudh. Hàm hienktdh cho phép in ký tự lên màn hình đồ họa theo kiểu văn bản thông thường (25 dòng, 80 cột). Hàm được xây dựng cho màn hình VGA ở độ phân giải cao (480 x 640).

Hàm hienktdh có dạng:

void hienktdh(char *dkt, int h, int c, int d, int mn, int mc) ;

Giải thích:

Cho hiện lên màn hình đồ họa dãy ký tự (được dkt trỏ tới) tại hàng h từ cột c đến cột (c+d-1), ở đây độ dài của dãy là d. Giá trị của hàng từ 1 đến 25, của cột từ 1 đến 80, mn là mầu nền (mầu khung), mc là mầu chữ.

Hàm menudh có dạng:

void menudh(int n, int hd, int cd, int dai, char nd[][40],
            int mnc, int mcc, int mnr, int mcr, int *cn_chon)

Ở đây:

n là số chức năng (n cần không quá 20)

hd, cd, dai là hàng đầu, cột đầu và độ dài của khung menu

nd là mảng 2 chiều chứa nội dung (tên) các chức năng

mnc, mcc là mầu nền và mầu chữ chung cho các chức năng

mnr, mcr là mầu nền và mầu chữ riêng cho chức năng được chọn

cn_chon trỏ tới biến chúa số hiệu của chức năng được chọn tính từ 1

Trong hàm menudh có dùng hàm hienktdh. Chương trình dưới đây minh họa cách dùng hàm menudh để xây dựng menu một mức gồm 3 chức năng: TAM GIAC, HINH TRON và KET THUC. Để đi đến một chức năng nào đó ta dùng các phím mũi tên lên hoặc mũi tên xuống. Để chọn chức năng ta dùng phím ENTER (sau khi đưa hộp sáng đến chức năng này).

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void hienktdh(char *dkt, int h, int c, int d, int mn, int mc)
  {
    int x, y;
    setfillstyle(1,mn);
    setcolor(mc);
    x= 8*(c-1);
    y= 19*h-1;
    bar(x, y-14, x+8*d, y);
    settextjustify(0, 1);
    outtextxy(x, y-7, dkt);
  } /* Kết thúc hàm hienktdh */
void menudh(int n, int hd, int cd, int dai, char nd[][40],
            int mnc, int mcc, int mnr, int mcr, int *cn_chon)
  {
    int i, chonm, mn, mc, chon;
    int ch1, ch2;
    chon = *cn_chon;
       /* Trình bầy */
    chonm= chon;
    for (i=1; i<= n; ++i)
     hienktdh(nd[i], hd+i-1, cd, dai, mnc, mcc);
     hienktdh(nd[chonm], hd+chonm-1, cd, dai, mnr, mcr);
```

```
      /* Bắt phím, chọn chức năng */
    ch1= 0;
    while( ch1 != 13)
      {
        ch1 = getch();
        if (ch1 == 0)
           ch2 = getch();
        if (ch1 == 0)
           {
             if (ch2 == 72)
              {
               chonm = chon-1;
               if (chonm < 1)
                 chonm = n;
              }
             if (ch2 == 80)
              {
               chonm= chon+1;
               if (chonm > n)
                 chonm = 1;
              }
           }
      if (chonm != chon)
       {
        hienktdh(nd[chon],hd+chon-1,cd,dai,mnc,mcc);
        hienktdh(nd[chonm],hd+chonm-1, cd, dai, mnr,mcr);
        chon = chonm;
       }
    }      /*Kết thúc  while */
   *cn_chon = chon;
}           /*Kết thúc hàm menudh */
  /* Hàm main */
void main()
  {
    char nd[4][40];
    int cao, rong;
```

```
int chon, mh, mode;
mh = DETECT;
initgraph(&mh,&mode,"");
setbkcolor(1);
settextstyle(0,0,1);
strcpy(nd[1], "TAM GIAC") ;
strcpy(nd[2], "HINH TRON") ;
strcpy(nd[3], "KET THUC") ;
chon = 1;
while (chon != 3)
   {
     initgraph(&mh,&mode,"");
     setbkcolor(1);
     hienktdh("THU MENU", 6, 36, 9, RED, WHITE + BLINK);
     menudh(3,8,36,9,nd,CYAN,YELLOW,BROWN,WHITE,&chon);
   if (chon == 1)
      {
        closegraph();
        textmode(C80);
        puts("TAM GIAC");
        getch();
      }
if (chon == 2)
   {
     closegraph();
     textmode(C80);
     puts("HINH TRON");
     getch();
   }
}
closegraph();
}
```

§5. VẼ ĐỒ THỊ HÀM SỐ

Hàm dothi được thiết kế như sau:

```
void dothi(double (*f)(double), float a, float b, int x1,int y1,
    int x2, int y2, int mn, int md)
```

Trong đó:

f là con trỏ tới hàm kiểu double, có một đối double

[a, b] là miền xác định của hàm

[x1, y1, x2, y2] là miền chữ nhật dùng để vẽ đồ thị của hàm

mn là mầu nền

md là mầu đồ thị

Hàm sẽ vẽ đồ thị hàm y = f(x) với a <= x <= b trên cửa sổ [x1,y1,x2,y2] của màn hình đồ hoạ.

Hai chương trình dưới đây sẽ minh hoạ cách dùng hàm dothi để vẽ đồ thị của hàm chuẩn C++ và vẽ đồ thị hàm tự xây dựng.

Chương trình 1: Vẽ đồ thị hàm sin

```
#include "graphics.h"
#include "conio.h"
#include "math.h"
void dothi(double (*f)(double), float a, float b, int x1,int y1,
        int x2, int y2, int mn, int md)
  {
   double fx, k, h, r, c, d;
   int x, y, n, m, i;
   int x0, y0;  /* ứng với x=0 và y=0 */
              /* Xác định d=max f(x), c=minf(x) */
  c = f(a);
  d=f(a);
  r= a;
  h= (b-a)/1000;
  while (r<= b)
    {
     fx = f(r);
     if (fx < c)
       c = fx;
     if (fx > d)  d = fx;
     r = r+h;
    }
```

```
setcolor(md); setbkcolor(mn);
n= x2-x1;
h=(b-a)/n;
m= y2-y1;
k= (d-c)/m;
x0 = x1 + (int)(-a/h+0.5);
y0 = y2 - (int)(-c/k+0.5);
    //Vẽ trục hoành
line(x1-20, y0, x2+20, y0);
line(x2+10, y0-10, x2+20, y0);
line(x2+10, y0+10, x2+20, y0);
    //Vẽ trục tung
line(x0, y1-20, x0, y2+20);
line(x0-10, y1-10, x0, y1-20);
line(x0+10, y1-10, x0, y1-20);
for (i= 0; i<= n; ++i)
 {
   x= x1+i;
   fx= f(a+i*h);
   y= (int) ((fx-c)/k + 0.5) + y1;
   y= y2-y+y1;
   if (i == 0)  moveto(x,y);
   else  lineto(x,y);
 }
}  /*Kết thúc hàm dothi*/
   /* Hàm main */
void main()
 {
  int mh=0, mode=0;
  initgraph(&mh,&mode,"");
  dothi(sin, -2*M_PI, 4*M_PI, 100, 250, 500, 350, MAGENTA, WHITE);
  getch();
  closegraph();
 }
```

Chương trình 2: Vẽ đồ thị hàm f(x) = (x-1)*(x-3)

```
#include "graphics.h"
#include "conio.h"
#include "math.h"
double  px2(double x)
      {
        return (x-1)*(x-3);
      }
void dothi(double (*f)(double), float a, float b, int x1,int y1,
            int x2, int y2, int mn, int md)
   {
     double fx, k, h, r, c, d;
     int x, y, n, m, i;
     int x0, y0; /* ứng với x=0 và y=0 */
                 /* Xác định d=max f(x), c=minf(x) */
    c = f(a);
    d=f(a);
    r= a;
    h= (b-a)/1000;
    while (r<= b)
      {
        fx = f(r);
        if (fx < c) c = fx;
        if (fx > d)
             d = fx;
        r = r+h;
      }
     setcolor(md);
     setbkcolor(mn);
     n= x2-x1;
     h=(b-a)/n;
     m= y2-y1;
     k= (d-c)/m;
     x0 = x1 + (int)(-a/h+0.5);
     y0 = y2 - (int)(-c/k+0.5);
```

```
  //Vẽ trục hoành
 line(x1-20, y0, x2+20, y0);
 line(x2+10, y0-10, x2+20, y0);
 line(x2+10, y0+10, x2+20, y0);
  //Vẽ trục tung
 line(x0, y1-20, x0, y2+20);
 line(x0-10, y1-10, x0, y1-20);
 line(x0+10, y1-10, x0, y1-20);
 for (i= 0; i<= n; ++i)
   {
     x= x1+i;
     fx= f(a+i*h);
     y= (int) ((fx-c)/k + 0.5) + y1;
     y= y2-y+y1;
     if (i == 0)
      moveto(x,y);
     else
      lineto(x,y);
   }
 } /*Kết thúc hàm dothi*/
 /* Hàm main */
void main()
  {
    int mh=0, mode=0;
    initgraph(&mh,&mode,"");
    dothi(px2, -1, 4, 50, 50, 500, 400, MAGENTA, WHITE);
    getch();
    closegraph();
  }
```

MỤC LỤC

Chịu trách nhiệm xuất bản:

Lê Tử Giang

Biên tập:

Trần Quang

Sửa bản in:

Đặng Quốc Đoàn

Trình bầy bìa:

Hương Lan

In 1000 cuốn khổ 16x24 cm tại Công ty Cổ phần In 15 -Bộ CN
Giấy phép xuất bản số 230/XB- QLXB cấp ngày 03/3/2005
In xong và nộp lưu chiểu tháng 2 năm 2006

CÙNG MỘT TÁC GIẢ:

Đã xuất bản

1. NGÔN NGỮ C - LÝ THUYẾT VÀ THỰC HÀNH
   Với 80 chương trình mẫu trong KHKT và quản lý kinh tế
2. KỸ THUẬT LẬP TRÌNH TURBO PASCAL 5-7
   Giáo trình cơ sở và nâng cao
   với kỹ thuật lập trình hướng đối tượng
3. KỸ THUẬT LẬP TRÌNH C
   Cơ sở và nâng cao
4. HƯỚNG DẪN SỬ DỤNG MICROSOFT ACCESS 97
   Có minh hoạ trên nhiều bài toán thực tế
5. KỸ THUẬT LẬP TRÌNH ACCESS TRÊN WINDOWS
6. C^{++} & LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
7. KỸ THUẬT LẬP TRÌNH WINDOWS
   dùng ngôn ngữ C và các hàm API

Đón đọc

8. LẬP TRÌNH INTERNET
   sử dụng ASP

TỔNG PHÁT HÀNH
NHÀ SÁCH MINH CHÂU
SỐ NHÀ 30 - NGÕ 22 TẠ QUANG BỬU - HÀ NỘI
ĐIỆN THOẠI: 04.8692785 * FAX: 04.8683995

GS. PHẠM VĂN ẤT

KỸ THUẬT LẬP TRÌNH C
CƠ SỞ VÀ NÂNG CAO