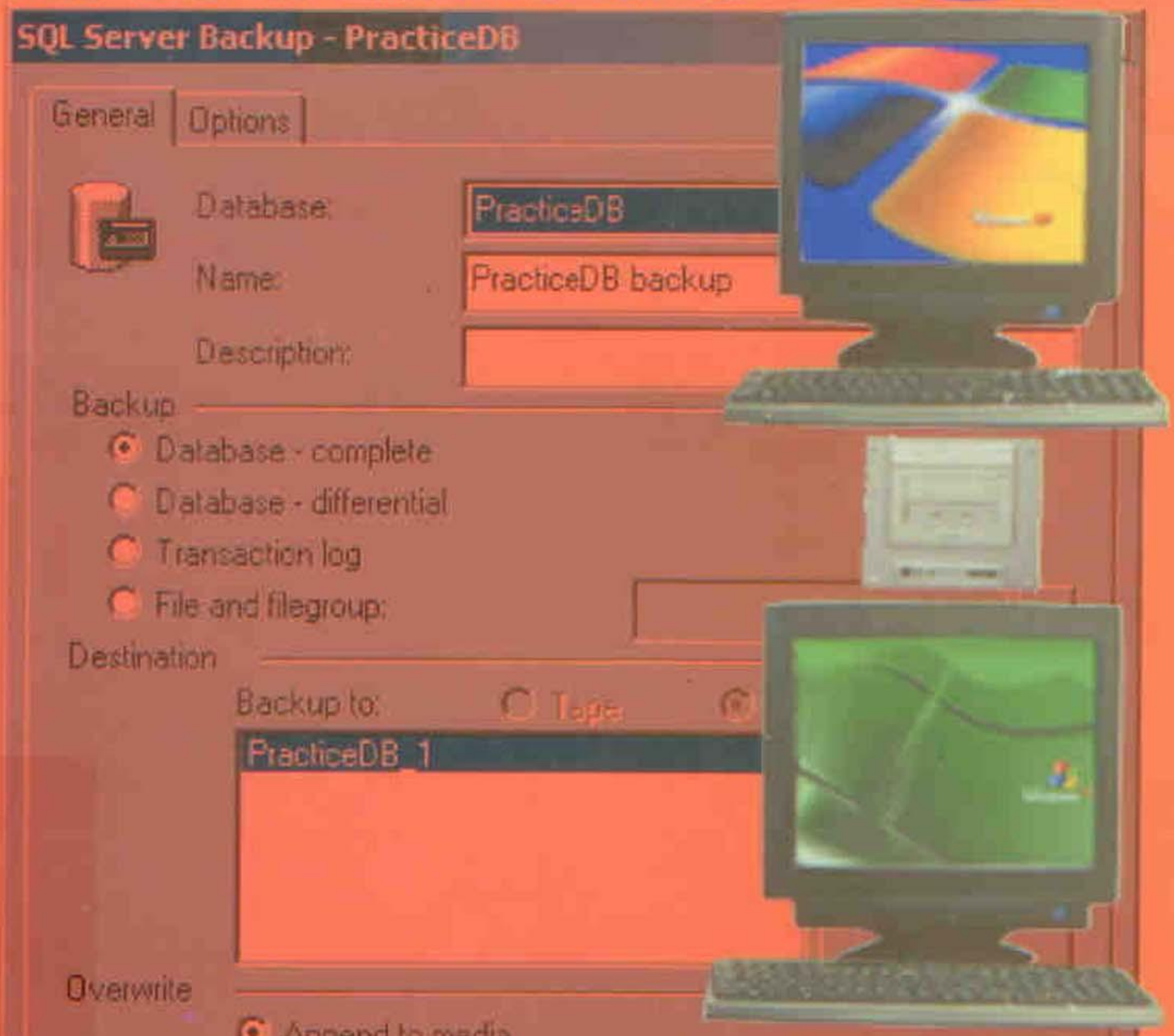
ẠM VĂN ẤT

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)

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 tháp 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 bầy 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 lă 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 HANHnLAM QUEN VOI BAN");
   getch(); /* Tạm dừng máy để xem kết quả */
}
```

Nhận xét: Mỗi cậu 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=%.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:

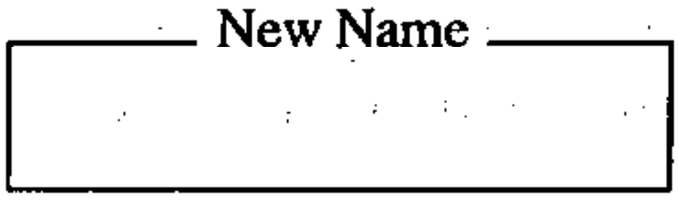
File Edit Run Compile Project Option Debug Break/Watch

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:



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.

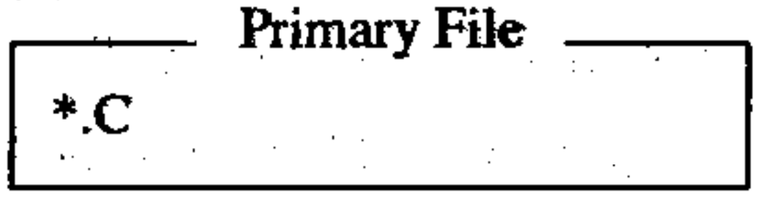
Chú ý: Ta có thể sửa một tệp cũ hoặc tạo một tệp mới bằng cách dùng chức năng File-Load (chức năng load của menu File). Khi hỏi đến tên tệp, ta nạp vào tên tệp cần sửa hay tên tệp cần tạo. Nếu không nói rõ đuôi, trình soạn thảo sẽ hiểu là đuôi C.

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:



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 đến 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 phảy 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.

+ 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 xảy 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 qủả 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 ");
            fprint(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 phấn 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 dãy % 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 đò 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 chường 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=-11;
      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\ny1=%-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

Chương trình 3

```
/*
Chuong trinh minh hoa:
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 đòng 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 mãn 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 lai.

+ 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
                       nhẩy 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 phảy. 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)                    if (biểu thức)
  khối lệnh 1;                      khối lệnh 1;
else
  khối lệnh 2;
(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 nhẩy 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 cậu 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:

```
/* Chương 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 nhẩy 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");
           }
        }
```

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 phảy (;).

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 du 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.

```
/* Chương trình tính tích vô hướng, Bản 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 dẫy n số nguyên chứa trong mảng a (bắt đầu từ a[1]). Hãy xác định xem có tồn tại phần tử âm trong dẫy 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 TẬP CHƯƠNG 5

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

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$$

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

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Bài 5. Lập chương trình để:

- Vào bốn dãy số $a_1,\ldots,a_n$; $b_1,\ldots,b_n$; $c_1,\ldots,c_n$; $d_1,\ldots,d_n$.
- 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

$$C_n^m = \frac{n.(n-1)\ldots(n-m+1)}{m!}$$

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:

$$f(x) = a_1x_1 + \ldots + a_{nn}$$

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 dung 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 thục có thể viết nhưsau:

```
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 môt dẫy 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 thẻ 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 làn 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 trện.

```
#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 trân 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 đùng 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);
       }
}
```

```
..oid 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. Tạ 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 có 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

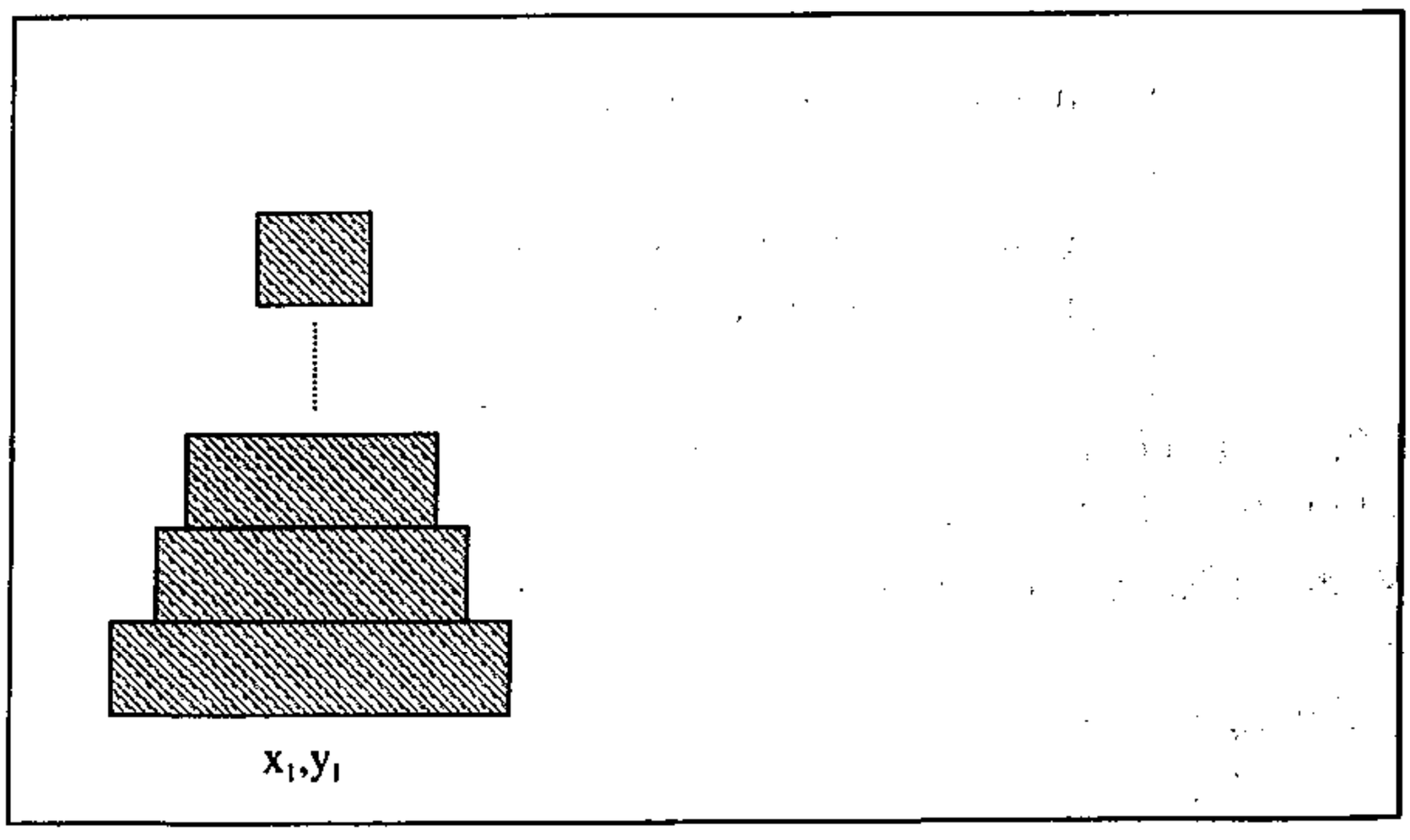
Hàm tìm ước số chung lớn nhất có thể viết như sau:

```
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 đợn 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 nhiêu 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 tuy 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. Tên các 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

$$s = \frac{x^1 + ... + x^n}{n}$$

$$d^2 = \frac{(x_1 - s)\sup 2 + \ldots + (x_n - s)^2}{n}$$

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;
```

§3. TRUY NHẬP ĐẾN CÁC THÀNH PHẦN CỦA CẤU TRÚC

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 giấ 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 tên */
         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 dụ. 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 khộng 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:

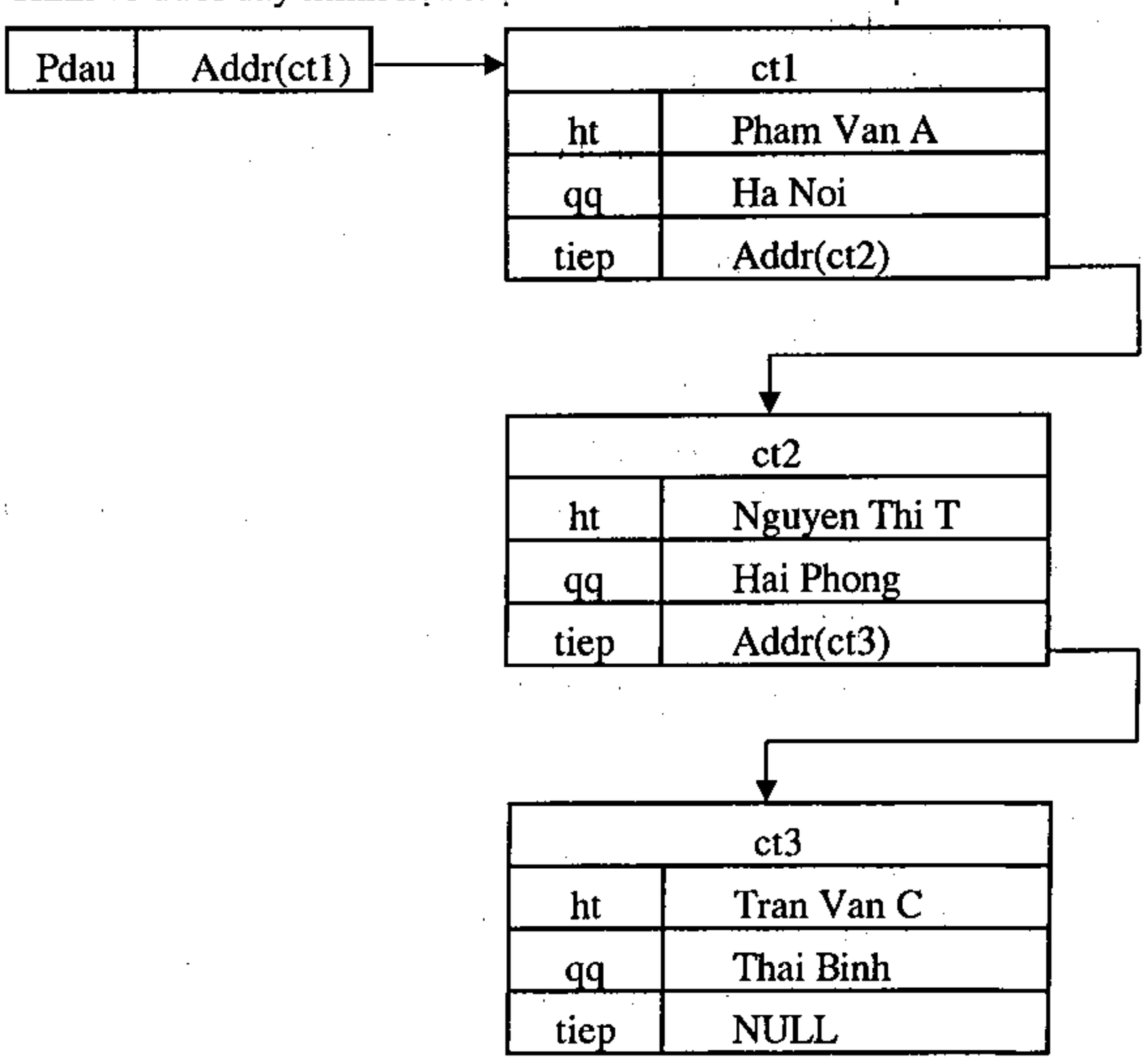
+ Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó (giả sử pdau).

+ 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 chiếu 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 lai:

+ 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 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 dịnh 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;
/* luu 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 cùng 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;
/* Luu mot 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 khối */
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 gấy 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

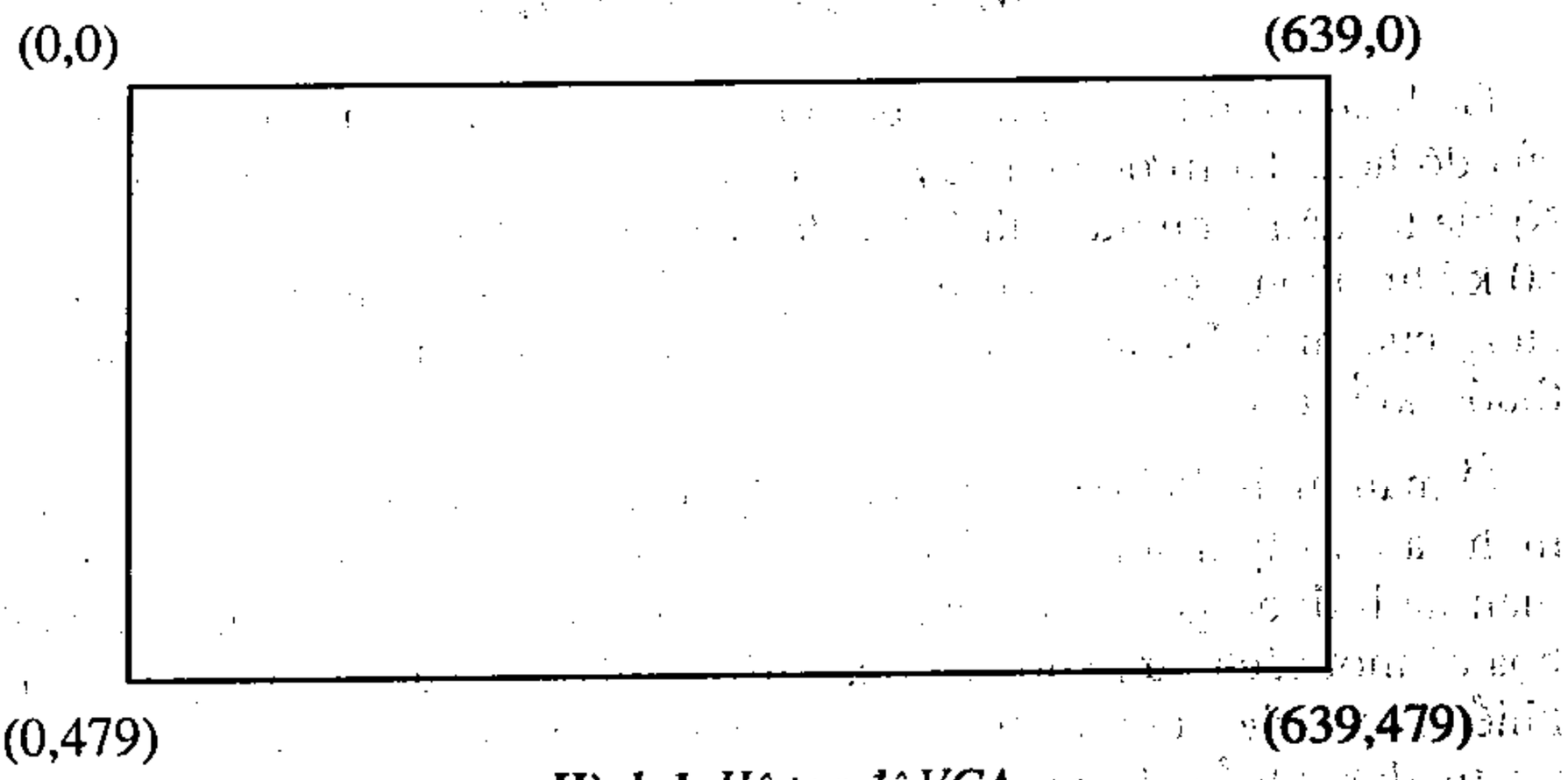
Bảng 1. Các tệp tin điều khiển đồ họa của TURBO C

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 chò 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);

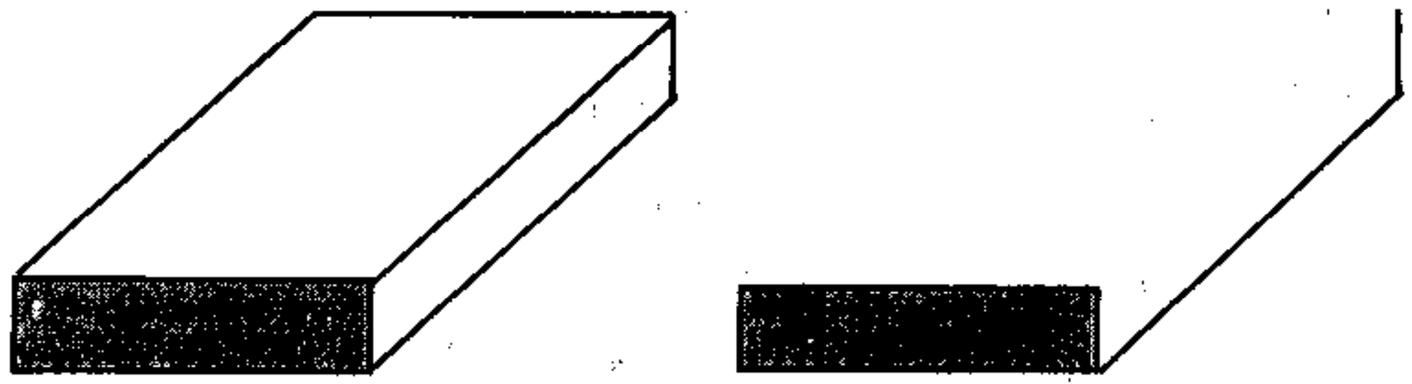
sẽ vẽ và tô mầu một hình chữ nhật. Các giá trị x1, y1, x2 và y2 có ý nghĩa như đã nói trong điểm 1.

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;      /* Nhân 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 */
```

9.1. 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 cua ban: ");
      text_read (&x,&y,name);
      text_writeln (&x,&y,"" );
      text_write(&x,&y,"chao ban ");
      text_write(&x,&y,name);
      getch();
      closegraph();
   }
```

§10. CẤT 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 hoa.

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 dãy các byte. Như vậy để ghi một số thực lên đĩa ta phải dùng dịch vụ ghi 4 byte, để ghi 10 số nguyên ta dùng dịch vụ ghi 20 byte.

+ Mỗi tệp có một số hiệu (còn gọi là danh số hay thẻ, 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 chuộng 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 dẫy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dẫy 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.