TRẦN VIỆT LINH
LÊ ĐĂNG HƯNG
LÊ ĐỨC TRUNG
NGUYỄN THANH THUỶ (Chủ biên)

NHẬP MÔN LẬP TRÌNH NGÔN NGỮ C

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

LÊ ĐĂNG HƯNG, TRẦN VIỆT LINH, LÊ ĐỨC TRUNG
NGUYỄN THANH THUỶ (Chủ biên)

Nhập môn
LẬP TRÌNH
NGÔN NGỮ C

Lý thuyết, Ví dụ, Bài tập mẫu, Bài tập

In lần thứ 3

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2003

Chịu trách nhiệm xuất bản: **PGS. PTS. TÔ ĐĂNG HẢI**
Biên tập: NGỌC KHUÊ- NGUYỄN MẠNH HÙNG
Sửa bản in: NGỌC KHUÊ
Vẽ bìa: HƯƠNG LAN

In 1000 cuốn khổ 16 x 24 cm. Tại Công ty In Hàng không
Giấy phép xuất bản số 113-240 ngày 4/4/ 2003 do Cục Xuất bản cấp.
In xong và nộp lưu chiểu tháng 5/2003.

LỜI GIỚI THIỆU

Khoa Công nghệ Thông tin (CNTT) Đại học Bách khoa Hà Nội là một trong những cơ sở đào tạo trọng điểm về CNTT được nhà nước quan tâm đầu tư xây dựng trong khuôn khổ chương trình CNTT quốc gia.

Để nâng cao chất lượng đào tạo, một trong những yếu tố quyết định là chất lượng của hệ thống chương trình và giáo trình, đặc biệt đối với một ngành phát triển vũ bão như CNTT. Chương trình và giáo trình đào tạo về CNTT đòi hỏi phải được cập nhật và bổ sung thường xuyên.

Cuốn sách " Nhập môn lập trình ngôn ngữ C" là một cố gắng của tập thể cán bộ giảng dạy khoa CNTT, Đại học Bách khoa Hà nội nhằm giới thiệu một ngôn ngữ không phải là mới nhưng bằng những thể nghiệm mới trong phương pháp tiếp cận và chuyển tải kiến thức.

Tài liệu này được dùng làm giáo trình giảng dạy cho các hệ đào tạo kỹ sư, cử nhân và cao đẳng tin học, đồng thời cũng có thể sử dụng để dạy cho các chuyên ngành khoa học và kỹ thuật khác.

Ở nước ta, đây không phải là cuốn sách đầu tiên về ngôn ngữ C. Đã từng có một số tài liệu viết hoặc biên dịch về C, kể cả bản dịch của cuốn sách "The C programming language" nổi tiếng của chính hai tác giả của ngôn ngữ là Brian W. Kernighan và Dennis M. Ritchie viết từ năm 1978. Tuy nhiên chúng tôi muốn nhấn mạnh về tính sư phạm của cuốn sách đang ở trong tay các bạn, với bố cục, các dẫn dắt vấn đề, các ví dụ minh hoạ và bài tập mẫu đã được thử nghiệm trên máy cho phép học viên nhanh chóng nắm bắt được đồng thời cả về kiến thức về ngôn ngữ lẫn kỹ năng lập trình căn bản.

Xin trân trọng giới thiệu cuốn sách với bạn đọc và tin chắc rằng nó sẽ trở thành một cẩm nang có ích cho những ai muốn sử dụng ngôn ngữ C để cài đặt các ứng dụng tin học.

Hà nội, ngày 4 tháng 9 năm 1999

GS. TS Nguyễn Thúc Hải

Trưởng khoa CNTT

Đại học Bách khoa Hà Nội

LỜI NÓI ĐẦU

Cuốn sách "Nhập môn ngôn ngữ lập trình C" trình bày nội dung một môn học cơ sở trong chương trình đào tạo kỹ sư tin học, cử nhân cao đẳng tin học tại khoa Công nghệ Thông tin, trường Đại học Bách khoa Hà Nội. Ở đây, sinh viên được trang bị những kiến thức cơ bản nhất về ngôn ngữ C, các kỹ thuật tổ chức dữ liệu và lập trình căn bản.

Bên cạnh việc trình bày đầy đủ, ngắn gọn, súc tích ngôn ngữ lập trình C còn bổ sung thêm những kinh nghiệm thiết yếu để thực hiện các chương trình C trên các môi trường như Turbo C, Borland C. Vì vậy rất phù hợp cho những người mới học lập trình hoặc những người đã có kiến thức về các ngôn ngữ lập trình khác như PASCAL.

Các đặc điểm của ngôn ngữ lập trình C được trình bày kỹ lưỡng cùng với những ví dụ minh hoạ bám sát chủ đề nhằm giúp bạn đọc khắc sâu được các chủ đề của cuốn sách. Trong các ví dụ chúng tôi còn cố gắng thể hiện các khía cạnh quan trọng của lập trình cấu trúc như: "Tổ chức dữ liệu", "Điều khiển chương trình", "Mô đun hoá chương trình", tuy vậy vấn đề "Làm mịn chương trình" chưa được đề cập ở đây.

Khi viết, chúng tôi đã cố gắng chú ý nhiều đến tính sư phạm của cuốn sách. Các chương mục được sắp xếp đảm bảo tính liên tục và sự thừa kế từ các phần trước. Trong mỗi chương đều có phần tóm tắt nêu lên những nội dung mà người đọc cần lĩnh hội được.

Nội dung cuốn sách gồm 8 chương và 1 phụ lục. Chương 1 tổng quan về ngôn ngữ C, giới thiệu xuất xứ và những nét cơ bản trong một chương trình C. Chương 2 trình bày các khía cạnh cơ sở của lập trình như: kiểu dữ liệu, các toán tử, các cấu trúc điều khiển chương trình. Thao tác với con trỏ là một trong những chủ đề quan trọng của ngôn ngữ C sẽ được trình bày trong các chương 3, 4 và 5. Chương 4 đưa ra một giới thiệu khá đầy đủ về tổ chức và xây dựng các hàm trong ngôn ngữ C. Các khía cạnh liên quan đến kiểu dữ liệu có cấu trúc được trình bày trong chương 5. Chương 6 đề cập tới việc xử lý các tệp và thao tác vào ra. Theo yêu cầu của nhiều sinh viên chúng tôi dành chương 7 để giới thiệu thư viện các hàm đồ hoạ của hãng Borland được cài đặt trong Turbo C và Borland C nhằm giúp bạn đọc có thể xây dựng được các ứng dụng đồ hoạ trên C. Sau cùng, khi đã thành thạo các khái niệm cơ sở trong lập trình với ngôn ngữ C, bạn đọc sẽ được làm quen với một chủ đề thú vị liên quan đến cách sử dụng các chỉ thị tiền xử lý trong chương trình C. Phần này là nội dung của chương 8. Phụ lục duy nhất của cuốn sách tuy không dài nhưng giới thiệu được cách cài đặt các tham số cần thiết trong môi trường Turbo C, Borland C, là môi trường lập trình có tính sư phạm cao và cũng rất mạnh.

Với mục tiêu cung cấp những kiến thức cơ bản nhất, cuốn sách chưa đề cập được các khía cạnh khác như: ghép nối C với các môi trường lập trình khác (Assembler, FoxPro ...), lập trình nâng cao với các kỹ thuật đồ hoạ, lập trình hướng đối tượng...

Tuy đã hết sức cố gắng, nhưng thiếu sót là điều không tránh khỏi. Chúng tôi mong nhận được nhiều ý kiến đóng góp từ đồng nghiệp và bạn đọc. Thư từ góp ý xin gửi về:

Nguyễn Thanh Thuỷ,

Bộ môn các hệ thống Thông tin,

Khoa Công nghệ Thông tin, Đại học Bách khoa Hà Nội

Tel (04).8696124, (04).8692463

thuynt@it-hut.edu.vn

Chúng tôi bày tỏ sự biết ơn tới GS Nguyễn Thúc Hải, Trưởng khoa Công nghệ Thông tin Đại học Bách khoa Hà Nội, người đã cổ vũ tập thể tác giả viết cuốn sách này. Đồng thời các tác giả xin tỏ lòng biết ơn tới GS Nguyễn Văn Ba, GS Đỗ Xuân Lôi, GS Vũ Lục, Ths Đỗ Văn Uy và tập thể cán bộ giảng dạy trong khoa cùng các thế hệ sinh viên đã có nhiều ý kiến đóng góp quý báu trong việc xây dựng đề cương và nội dung các ví dụ.

Tập thể tác giả hi vọng cuốn sách sẽ có ít nhiều đóng góp phần nào trong sự nghiệp đào tạo CNTT ở Việt Nam.

Xin trân trọng cảm ơn.

Hà nội, ngày 11 tháng 9 năm 1999

Các tác giả

Chương 1

TỔNG QUAN VỀ NGÔN NGỮ C

Mục đích chương này

1) Giới thiệu quá trình phát triển của ngôn ngữ C
2) Giới thiệu các đặc điểm của ngôn ngữ C.
3) Cách viết và thực hiện một chương trình C đơn giản.

1. LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN

Ngôn ngữ C do Brian W. Kernighan và Dennis M. Ritchie phát triển vào đầu những năm 70 tại phòng thí nghiệm BELL (Hoa kỳ) với mục đích ban đầu là để phát triển hệ điều hành UNIX. Bối cảnh ra đời xuất phát từ nhu cầu cần phải có một ngôn ngữ lập trình hệ thống thay thế cho hợp ngữ (assembly) vốn nặng nề, độ tin cậy thấp và rất khó chuyển đổi giữa các hệ máy tính khác nhau.

Phần lớn các ý tưởng quan trọng nhất của C xuất phát từ một ngôn ngữ có trước đó với tên gọi BCPL, do Martin Richards nghiên cứu. Ảnh hưởng của BCPL đối với C gián tiếp thông qua ngôn ngữ B, do Ken Thompson viết năm 1970 cho hệ UNIX, chạy trên họ máy tính PDP-7.

Ngoài việc C được dùng để viết hệ điều hành UNIX (hiện nay, trên 90 % chương trình nguồn của các hệ điều hành UNIX được viết bằng C, chưa đầy 10% bằng hợp ngữ), người ta nhanh chóng nhận ra sức mạnh của C trong việc xử lý các vấn đề hiện đại của tin học: xử lý số, văn bản, cơ sở dữ liệu, lập trình hướng đối tượng. Thực tế C đã tổ hợp được các thành tựu tiên tiến của tin học và đã trở thành một chuẩn mặc nhiên.

Liên quan đến sự hình thành và phát triển của ngôn ngữ, có thể kể đến một số sự kiện đáng quan tâm sau:

(i) Năm 1978, cuốn giáo trình dạy lập trình bằng ngôn ngữ C "The C programming langguage" do chính hai tác giả của ngôn ngữ Brian W. Kernighan và Dennis M. Ritchie biên soạn đã được xuất bản và được phổ biến rộng rãi.

(ii) Năm 1983 một tiểu ban của viện tiêu chuẩn quốc gia mỹ (ANSI) được thành lập nhằm đề xuất ra một chuẩn cho ngôn ngữ C.

(iii) Năm 1988 chuẩn ANSI C chính thức được ban hành. Chuẩn này bao gồm các mô tả về ngôn ngữ theo Brian W. Kernighan và Dennis M. Ritchie và

quy định các thư viện chuẩn của ngôn ngữ C, nhờ đó tăng tính khả chuyển của chương trình viết bằng C.

(iv) Trong thế giới máy vi tính có các hệ chương trình dịch C nổi tiếng như:

Turbo C, Borland C của **Borland Inc.**

MSC, VC của **Microsoft Corp.**

Lattice C của **Lattice**

(v) Sự phát triển của ngôn ngữ lập trình trong những năm 80 đã đưa đến phong cách lập trình hướng đối tượng. Một trong những ngôn ngữ rất được ưa dùng là C++, một bổ sung mới các yếu tố hướng đối tượng vào ngôn ngữ C.

The only way to learn a new programming language is by writting programs in it.

Brian W Kernighan.

Tạm dịch là :

Cách duy nhất để học một ngôn ngữ lập trình mới là viết nhiều chương trình bằng ngôn ngữ đó.

2. CÁC TÍNH CHẤT ĐẶC TRƯNG CỦA NGÔN NGỮ

C là ngôn ngữ lập trình vạn năng được dùng để viết các hệ điều hành như UNIX cũng như các chương trình ứng dụng như QLVB (quản lý văn bản), CSDL (cơ sở dữ liệu).

C là một ngôn ngữ có mức độ thích nghi cao. Do các kiểu dữ liệu và cấu trúc điều khiển của C có hầu hết trên các máy tính nên thư viện lúc chạy cần để cài đặt chương trình là khá gọn. Hơn nữa, vì ngôn ngữ phản ánh khả năng của máy tính hiện tại nên chương trình C tỏ ra là đủ hữu hiệu, tức là không nhất thiết phải cần tới hợp ngữ.

Mặc dù vậy, C vẫn độc lập với bất kỳ kiến trúc máy đặc thù nào và với một chút thận trọng vẫn dễ dàng viết được các chương trình "khả chuyển" (portability) tức là những chương trình có thể chạy mà không cần phải thay đổi gì khi có thay đổi về phần cứng.

Nếu như PASCAL là một ngôn ngữ chính để giảng dạy về lập trình thì ngôn ngữ C là một ngôn ngữ được sử dụng rộng rãi trong các lĩnh vực chuyên nghiệp vì đáp ứng được các yêu cầu: hiệu quả cao trong soạn thảo chương trình và dịch ra mã máy; tiếp cận trực tiếp với các thiết bị phần cứng.

Ngôn ngữ C không đưa ra các phép toán xử lý trực tiếp các đối tượng hợp thành (xâu ký tự, danh sách, hoặc bảng) như là đối tượng toàn vẹn. Chẳng hạn, không có các phép toán tương tự như trong PASCAL để thao tác trên bảng hay xâu.

C cũng không xác định bất kỳ một phương tiện cấp phát bộ nhớ nào khác ngoài cấp phát tĩnh cấp phát động theo nguyên tắc xếp chồng dành cho cho các biến cục bộ của hàm. Bản thân C không cung cấp cơ chế vào ra, không có phương pháp truy nhập tệp. Tất cả các cơ chế này đều được thực hiện bằng những lời gọi hàm trong thư viện.

C đưa ra các kết cấu điều khiển cơ bản cần cho các chương trình có cấu trúc như: nhóm tuần tự các câu lệnh, chọn quyết định (**if**); chu trình với phép kiểm tra kết thúc ở đầu (**for**, **while**), hoặc ở cuối (**do ... while**); và việc lựa chọn một trong các trường hợp có thể (**switch**).

C cung cấp con trỏ và khả năng định địa chỉ số học. Các đối của hàm được truyền bằng cách sao chép giá trị của đối và hàm được gọi không thể nào thay đổi được giá trị của đối hiện tại. Khi muốn "gọi theo tham trỏ", bạn phải truyền con trỏ một cách tường minh và hàm có thể thay đổi đối tượng mà con trỏ chỉ tới. Các tên bảng được truyền như vị trí của gốc của bảng cho nên các đối bảng thực tế được gọi theo dẫn trỏ.

C cho phép hàm được gọi đệ quy và các biến cục bộ của hàm sẽ được "tự động" sinh ra hoặc được tạo mới với mỗi lần gọi mới. Các định nghĩa hàm không được lồng nhau nhưng các biến có thể được khai báo theo kiểu cấu trúc khối. Các hàm của chương trình C có thể dịch tách biệt nhau. Các biến có thể là biến trong của hàm hoặc là biến ngoài hàm. Hàm chỉ biết được các biến ngoài trong cùng một tệp gốc, hoặc biến tổng thể **extern**. Các biến tự động có thể đặt trong các thanh ghi để tăng hiệu quả, nhưng việc khai báo thanh ghi chỉ là một hướng dẫn cho chương trình dịch và không liên quan gì đến các thanh ghi đặc biệt của máy cả.

C không phải là một ngôn ngữ có kiểu mạnh mẽ theo nghĩa của PASCAL hoặc Algol 68. Nó tương đối thoải mái trong chuyển đổi dữ liệu nhưng không tự động chuyển các kiểu dữ liệu một cách phóng túng như của PL/I. Các chương trình dịch hiện có đều không đưa ra cơ chế kiểm tra chỉ số mảng, kiểu đối số v.v...

Cuối cùng, giống như mọi ngôn ngữ khác, C vẫn còn có những nhược điểm: một số phép toán có thứ tự thực hiện chưa đúng; một số phần cú pháp có thể làm tốt hơn; hiện có nhiều phiên bản của ngôn ngữ, chỉ khác nhau ở một vài chi tiết.

Dù vậy, C vẫn tỏ ra là một ngôn ngữ cực kỳ hiệu quả và đầy sức diễn cảm đối với nhiều lĩnh vực ứng dụng lập trình.

3. CẤU TRÚC CƠ BẢN CỦA MỘT CHƯƠNG TRÌNH C

Hãy xem các thành phần của chương trình HELLO.C sau đây

```
/*Chương trình hiện lên dòng Hello world trên màn hình*/
#include <stdio.h>
#include  <conio.h>
void main() {
```

```
    printf("hello, world\n");
    getch();
}
```

3.1 Thực hiện chương trình

Các bước để thực hiện chương trình này như sau: trước tiên bạn phải tạo ra chương trình nguồn có tên HELLO.C trên một hệ soạn thảo nào đó; bước tiếp theo là dịch chương trình để tạo ra một tệp chương trình (có phần mở rộng là EXE); sau bước thứ hai nếu không có lỗi biên dịch, bạn đã có thể thực hiện được chương trình từ dấu nhắc của hệ điều hành.

Khi dùng môi trường soạn thảo kết hợp như Turbo C, ban đầu bạn phải khởi động Turbo C bằng câu lệnh (giả sử Turbo C được cài trên thư mục C:\TC):

```
\TC\TC ↵
```

Hệ thống được khởi động và bạn được chuyển vào chế độ soạn thảo chương trình trong Turbo C. Bạn hãy nhập chương trình trên vào máy. Khi nhập xong, ấn phím F2 để ghi chương trình lên đĩa, tất nhiên máy sẽ yêu cầu bạn nhập vào một tên cho chương trình vừa được đánh vào. Sau đó bạn ấn **CTRL-F9** để dịch và chạy chương trình. Nếu không phạm lỗi chính tả hay gõ nhầm thì chương trình sẽ được dịch và chuyển sang bước chạy. Bạn sẽ thấy xuất hiện dòng chữ

```
hello, world
```

trên màn hình. Ấn một phím bất kỳ bạn sẽ trở về màn hình soạn thảo bình thường của Turbo C.

Chú ý:

Xem thêm phụ lục 1 để biết cách thiết lập một số thông số cấu hình cần thiết.

3.2 Định nghĩa Hàm

Bây giờ thêm vài lời giải thích về bản thân chương trình. Một chương trình C, với bất kỳ kích thước nào cũng đều bao gồm một hay nhiều hàm, các hàm này sẽ xác định các thao tác tính toán thực tế cần phải thực hiện. Các hàm của C cũng tương tự như các hàm và thủ tục của chương trình viết bằng PASCAL. Trong chương trình ví dụ trên, main() là một hàm như vậy. Thông thường chúng ta có thể lấy bất kỳ tên nào để đặt cho hàm, nhưng main() là một hàm đặc biệt; chương trình C luôn luôn bắt đầu thực hiện tại điểm đầu của hàm này. Điều này có nghĩa mọi chương trình trong C đều phải có một (và chỉ một) hàm main() ở đâu đó trong chương trình. Hàm main() này thường gọi tới các hàm khác để thực hiện công việc của nó, một số hàm nằm trong chương trình, số khác nằm trong các thư viện của các hàm chuẩn.

Một phương pháp truyền dữ liệu cho các hàm là thông qua danh sách các đối số; ở đây `main()` là hàm không có đối số, được biểu thị bởi cặp dấu "(" và ")". Các dấu ngoặc nhọn "{" và "}" bao quanh các câu lệnh tạo nên hàm; chúng tương tự như `begin` và `end` của PASCAL. Hàm được gọi bởi tên của nó, theo sau là danh sách các đối số được đặt trong dấu ngoặc. Trong trường hợp không có biến vẫn phải viết các dấu ngoặc này. Xét câu lệnh

```
printf ("hello, world\n");
```

là một lời gọi tới hàm có tên là `printf()` với đối số ở đây là một xâu ký tự "`hello, world\n`". `printf()` là một hàm có trong thư viện chuẩn được sử dụng mà không phải viết lại, có chức năng đưa kết quả ra thiết bị đầu ra (thông thường đó là màn hình). Trong trường hợp này hàm sẽ cho hiện lên màn hình dãy ký tự tạo nên đối.

Dãy ký tự bất kỳ nằm trong hai dấu nháy kép "..." được gọi là một xâu ký tự.

Cặp `\n` xuất hiện trong một xâu trên là ký pháp của C để chỉ ký tự xuống dòng, báo hiệu lần sau đưa ra sẽ được thực hiện ở đầu dòng mới. Nếu ta bỏ `\n` đi thì trong lần kết xuất sau đó, các kết quả sẽ được viết tiếp đằng sau xâu kết xuất mà không xuống dòng mới. Hàm `printf()` không bao giờ tự động thêm dấu xuống dòng cho nên ta có thể tạo ra một dòng dài với nhiều lần gọi hàm này. Chương trình ban đầu của chúng ta có thể viết lại như sau:

```
/*Chương trình này cũng hiện lên dòng Hello world trên màn hình*/
#include  <stdio.h>
#include  <conio.h>
void main() {
        printf("hello,");
        printf(" word");
        printf("\n");
        getch();
        }
```

để cho ra cùng một kết quả.

Lưu ý rằng `\n` chỉ là một ký tự đặc biệt. Ngoài ra C còn cho phép dùng `\t` để biểu thị dấu `tab`, `\b` cho `backspace`, `\"` cho dấu nháy kép, và `\\` cho bản thân dấu xổ chéo v.v...

3.3 Khai báo tệp tiêu đề

Khi sử dụng các hàm trong các thư viện chuẩn, chúng ta phải khai báo tệp tiêu đề (`header file`) chứa hàm nguyên mẫu tương ứng của hàm đó, câu lệnh được

bắt đầu bằng `#include` theo sau là tên tệp tiêu đề. Trong ví dụ trên, chúng ta có sử dụng hàm `printf()` là hàm chuẩn được khai báo trong tệp `stdio.h` và hàm `getch()` được khai báo trong `conio.h` do đó trong chương trình có hai dòng sau ở trên đầu

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

3.4 Dòng chú thích

Dòng đầu tiên

/ Chương trình hiện lên dòng Hello world trên màn hình*/*

là lời chú thích, giải thích mục đích của chương trình. Mọi ký tự nằm giữa "/*" và "*/" đều được chương trình dịch bỏ qua; ta được phép dùng chúng để làm cho chương trình dễ hiểu. Lời chú thích có thể xuất hiện ở bất kỳ đâu trong chương trình nguồn, chứa được tất cả các ký tự và có thể trải dài trên nhiều dòng khác nhau trong chương trình.

3.5 Dấu kết thúc câu lệnh

Trong chương trình mỗi câu lệnh được kết thúc bằng dấu chấm phẩy ";". Ví dụ:

```
printf ("Hello world\n");
```

4. MỘT VÀI LỜI VỀ HÀM `printf()`

Câu lệnh

```
printf ("Hello world\n");
```

sẽ đưa ra màn hình dòng chữ nằm trong các dấu nháy kép. Hàm `printf()` thực sự là một hàm kết xuất rất mạnh và linh động được sử dụng rất nhiều trong C. Có thể sử dụng hàm `printf()` để trình bày các đối tượng dữ liệu khác nhau: số, ký tự, xâu ký tự. Để tiện trình bày trong các phần sau, chúng ta khảo sát sơ bộ các tính năng của `printf()`.

4.1 Trình bày số

Hàm `printf()` dùng một dạng thức duy nhất để trình bày hằng và biến số. Xét ví dụ sau:

```
#include <stdio.h>
void main() {
        printf("Day la con so : %d",2);
```

```
}
```

Chương trình này cho cùng kết quả như chương trình sau:

```
#include <stdio.h>
void main() {
    int i=2;
    printf("Day la con so : %d",i);
    }
```

Theo ví dụ này, hàm `printf()` có thể nhận nhiều đối số. Các đối số được phân cách nhau bằng các dấu phẩy. Hàm `printf()` lấy giá trị nằm bên phải dấu phẩy và đưa nó vào xâu ký tự định dạng bên trái ở vị trí có "định dạng" (trong trường hợp này là `%d`).

"Định dạng" cho `printf()` biết nơi sẽ đặt giá trị vào xâu định dạng cũng như dạng thức để trình bày giá trị đó. Kết quả khi chạy hai chương trình ví dụ trên là:

```
Day la con so : 2
```

Trong hai ví dụ này, `%d` báo cho `printf()` trình bày số 2 theo dạng thập phân. Còn có các dấu mô tả khác như `%f`, `%x` v.v... Chúng ta sẽ bàn đến chúng chi tiết hơn trong chương 2.

4.2 Đưa ra ký tự

Một hằng ký tự được định nghĩa bằng một ký hiệu trong bảng mã ASCII nằm giữa hai dấu nháy đơn, ví dụ

'a' 'b'

Dùng hàm `printf()` có thể viết ra các ký tự nhờ dấu mô tả `%c`. Trong lời gọi hàm này có thể sử dụng tên của ký tự (như `a') hoặc mã ASCII của ký tự (đối với `a' là 97). Xét ví dụ sau:

```
#include <stdio.h>
void main(){
    printf("%c %c",'a',97);
    }
```

```
a a
```

4.3 Trình bày xâu ký tự

Một xâu ký tự là một tập các ký tự (có thể chỉ một) nằm giữa hai dấu nháy kép. Ví dụ:

"I am a string"

Dùng hàm printf() có thể kết xuất một xâu nhờ dấu mô tả %s. Xét ví dụ sau đây:

```
#include <stdio.h>
void main(){
    printf("%s thi cach xa mat troi %d trieu miles","Venus",67);
}
```

```
Venus thi cach xa mat troi 67 trieu miles
```

Trong chương trình trên đây, hàm printf() đã thay thế dấu mô tả %s bằng xâu "Venus" và %d bằng con số 67.

5. TÓM TẮT

5.1 Ghi nhớ

Các chú thích bắt đầu bằng "/*" và kết thúc bằng "*/". Người lập trình có thể chèn các chú thích vào bất kỳ chỗ nào có thể đặt dấu phân cách (ví dụ: dấu trắng, tab, enter) trong chương trình. Các chú thích được chương trình dịch bỏ qua. Việc thêm các chú thích làm cho chương trình dễ đọc hơn.

Chỉ thị tiền xử lý #include <stdio.h> báo cho chương trình dịch kết hợp tệp tiêu đề chứa các khai báo của các hàm vào ra chuẩn trong chương trình.

Chương trình C bao gồm một hoặc nhiều hàm, trong số đó nhất định phải có một và chỉ một hàm main(). Mọi chương trình C bắt đầu được thực hiện tại điểm bắt đầu của hàm main().

Hàm printf() có thể sử dụng để in một xâu ký tự được chứa trong cặp dấu nháy kép. Khi in các giá trị số nguyên, đối số đầu của hàm printf() - xâu điều khiển- chứa các mô tả chuyển dạng %d và các ký tự khác có thể in ra được; tham số thứ hai là một biểu thức có giá trị được in ra. Nếu có nhiều giá trị nguyên được đưa ra, khi đó xâu điều khiển chứa nhiều mô tả %d, mỗi mô tả tương ứng với một số nguyên, và các tham số theo sau xâu định dạng (phân cách nhau bằng dấu phẩy) chứa các giá trị nguyên được in ra.

Ngoài mô tả %d cho các giá trị nguyên, hàm printf() còn sử dụng nhiều mô tả định dạng khác tương ứng cho các kiểu dữ liệu khác như %c, %s, %f v.v...

5.2 Các lỗi thường gặp

Thêm ";" sau tiêu đề hàm main.

Quên các chỉ thị #include khi sử dụng các hàm thư viện.

Nhầm lẫn giữa cách chú thích trong C và PASCAL.

Quên mất vai trò của xâu định dạng trong hàm `printf()`.

Chương 2

CÁC YẾU TỐ CƠ BẢN CỦA NGÔN NGỮ C

Mục đích chương này

1) Làm quen với các kiểu dữ liệu cơ sở trong C.
2) Biết cách khai báo biến, hằng.
3) Làm quen với các loại toán tử, đặc biệt là các toán tử mới như: toán tử gán, toán tử điều kiện.
4) Biết cách sử dụng các câu lệnh điều khiển để viết các chương trình.
5) Biết cách sử dụng hai hàm vào ra `printf()` và `scanf()` để xuất và nhập số liệu.

1. TỪ VỰNG

1.1 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 nên các từ. Đến lượt mình, các từ lại được liên kết theo một qui tắc (đó là cú pháp của ngôn ngữ lập trình) nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh diễn đạt một thuật toán nào đó. Ngôn ngữ lập trình C được xây dựng trên bộ ký tự sau:

26 chữ cái hoa: `A B C ...` 26 chữ cái thường: `a b c ...`

10 chữ số: `0 1 2 ... 9`

Các ký hiệu toán học: `+ - * / = < >`

Các dấu ngăn cách: `. ; , : space`

Các dấu ngoặc: `( ) [ ] { }`

Các ký hiệu đặc biệt khác như `_ ? % # $ & ^ \ ! ' "` ~ v.v...

Đây là tập hợp các ký tự hợp lệ được dùng để viết các chương trình trong C. Không được dùng các ký hiệu khác ngoài tập các ký hiệu nói trên.

1.2 Tên (identifier)

Tên bao gồm một dãy các ký tự: chữ, số và dấu gạch nối "_" (underscore) và phải bắt đầu bằng một chữ hoặc dấu gạch nối. Một số ví dụ về tên như là:

Ví dụ 1

Tên hợp lệ

```
name a_1
_function
delta_x
delta_y
```

Tên không hợp lệ

```
1_k          /* vì bắt đầu bằng chữ số*/
```

Độ dài cực đại của tên mặc định là 32, nhưng chúng ta có thể thay đổi lại bằng một giá trị từ 1 đến 32 trong chức năng **Option->Compiler->Source->Identifier** (Môi trường Turbo C). Tất nhiên, ta có thể đặt các tên với độ dài lớn hơn 32, trong trường hợp đó chỉ có 32 ký tự đầu tiên của tên có ý nghĩa.

Khác với PASCAL, ngôn ngữ C phân biệt giữa chữ hoa và chữ thường. Các tên `ABC`, `aBC`, `abC`, `abc`, `AbC` là khác nhau. Theo thói quen người ta thường dùng chữ hoa đặt tên cho các hằng, còn chữ thường được dùng để đặt tên cho các biến, các hàm, các cấu trúc v.v...

Khi đặt tên phải luôn ghi nhớ nguyên tắc tên ít nhiều phản ánh được bản chất của đối tượng được đặt tên. Ví dụ đặt tên cho biến chứa kích thước của mảng là `ArraySize` hơn là `arraysize` hoặc một tên khó hiểu là `as`. Việc đặt tên gợi nhớ rất có ích khi trong khi bảo trì, sửa chữa chương trình.

1.3 Từ khóa

Bảng sau đây giới thiệu một số từ khóa thông dụng do chương trình TURBO C định nghĩa.

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		

Từ khóa của C bao gồm một số từ riêng được C định nghĩa trước với ý nghĩa xác định. Các từ khóa được sử dụng để khai báo các kiểu dữ liệu (**int, float** ...), viết các toán tử (**sizeof**), các câu lệnh (**if, then**). Các từ khóa phải được dùng đúng với ý nghĩa, không được dùng nó vào việc khác hoặc đặt tên mới trùng với tên các từ khóa.

1.4 Các kiểu dữ liệu

Tất cả các biến phải được khai báo trước và kiểu của chúng phải được mô tả ngay khi khai báo. Có bốn kiểu dữ liệu cơ bản trong C là: **char**, **int**, **float** và **double.** Bảng 2.1 mô tả chi tiết các đặc điểm của các kiểu dữ liệu cơ sở đó:

Tên	Ý nghĩa	Phạm vi	Kích thước
char	Ký tự	-128 ÷ 127	1 byte
int	Số nguyên	-32768 ÷ 32767	2 byte
float	Số thực dấu phẩy động, độ chính xác đơn	±3.4E-38 ÷ ±3.4E+38	4 byte
double	Số thực dấu phẩy động, độ chính xác kép	±1.7E-308 ÷ ±1.7E+308	8 byte

Bảng 2.1 Các kiểu dữ liệu cơ sở trong ngôn ngữ C

Một số **float** có độ chính xác là 6 chữ số sau dấu chấm thập phân. Một số **double** có độ chính xác tới 15 chữ số sau dấu chấm thập phân.

Bên cạnh đó chúng ta có thể áp dụng một số các "tiền tố" đi kèm các kiểu dữ liệu cơ bản với mục đích thay đổi phạm vi biểu diễn của biến được khai báo. Có các tiền tố như là: **short, long, signed** (ngầm định đối với **char, int**) và **unsigned**. Bảng 2.2 mô tả cách áp dụng các "tiền tố" với các kiểu dữ liệu cơ sở.

Chú ý

(i) Kiểu **long int** có thể viết gọn thành **long**; kiểu **unsigned int** viết gọn thành **unsigned**.

(ii) Có thể kết hợp nhiều tiền tố với một kiểu dữ liệu cơ sở: chẳng hạn **unsigned long int** (viết gọn thành **unsigned long**).

1.5 Khai báo biến

Biến là yếu tố cơ bản của bất kỳ ngôn ngữ máy tính nào. Biến là vùng bộ nhớ máy tính được dành cho một kiểu dữ liệu nào đó và được đặt tên để tiện tham khảo.

Các biến trong bộ nhớ ở các thời điểm khác nhau có thể cất giữ các giá trị khác nhau. Trước khi sử dụng một biến nào đó, phải khai báo nó. Quy tắc khai báo như sau:

<Kiểu_dữ_liệu> <Tên_biến> ;

Có thể khai báo nhiều biến cùng kiểu trên cùng một hàng, trong trường hợp đó các tên biến được phân cách nhau bằng dấu phẩy. So với PASCAL, cách khai báo biến trong C "thuận tiện hơn". Điều này muốn nói đến việc trong việc khai báo biến C cho phép khai báo đồng thời khởi tạo giá trị cho biến. Xét các ví dụ sau:

Ví dụ 2

```
int var1,var2;/*biến có kiểu nguyên*/
float f;              /*biến nhận giá trị thực*/
char ch;              /*biến ký tự*/
int maxint=32767, zero = 0;
```

Kiểu	Kích thước	Phạm vi
unsigned char	1 byte	0 ÷ 255
char	1 byte	-128 ÷ 127
unsigned int	2 bytes	0 ÷ 65,535
short int	2bytes	-32768 ÷ 32767
int	2 bytes	-32768 ÷ 32767
unsigned long	4 bytes	0 ÷ 4294967,295
long	4 bytes	-2147483648 ÷ 2147483,647
float	4 bytes	±3.4E-38 ÷ ±3.4E+38
double	8 bytes	±1.7E-308 ÷ ±1.7E+308
long double	10 bytes	±3.4E-4932 ÷ ±1.1E+4932

Bảng 2.2 Các tiền tố dùng với các kiểu dữ liệu cơ sở

1.6 Khai báo hằng

1.6.1 Khai báo hằng

Hằng là đại lượng có giá trị không thay đổi khi thực hiện chương trình. Định nghĩa (khai báo) một hằng về thực chất là gắn một tên (tên hằng) với một giá trị cụ thể nào đó. Về sau, khi gặp tên đó chương trình dịch ngầm hiểu đó là giá trị tương ứng. Có hai cách khai báo hằng số trong C:

Dùng toán tử `#define` nhằm định nghĩa hằng tượng trưng.

```
#define <identifier> <string>
```

Chỉ thị này định nghĩa một hằng tượng trưng có tên là `identifier` với giá trị được mô tả bởi xâu `string`. Khi biên dịch chương trình, chương trình dịch

thay thế mọi xuất hiện của <identifier> bằng <string>. Điều cần lưu ý là kiểu giá trị của hằng được xác định dựa theo nội dung của string. Xét các ví dụ khai báo sau:

```
#define MAXLINE 80
#define DHBK "INFORMATIC"
#define NEWLINE '\n'
```

Biến hằng được định nghĩa nhờ từ khoá const với cú pháp như sau:

const <tên_kiểu> <tên_biến_hằng> = <giá_trị>;

Ví dụ 3

```
const   int   MAXLINE   = 100;
const   char  NEWLINE   =  '\n';
const   char  DHBK[18] =  "DAI BACH KHOA";
```

Chú ý rằng các câu lệnh bắt đầu bằng const xác định các biến có giá trị không thay đổi, nghĩa là chương trình dịch sẽ cấp phát vùng nhớ tương ứng với kiểu của hằng và gán cho nó giá trị đã cho nhưng không thể thay đổi giá trị của các biến hằng vì nó sẽ gây ra lỗi biên dịch.

1.6.2 Cách biểu diễn hằng số

Trong ngôn ngữ C, người ta phân biệt các loại hằng sau:

Hằng nguyên (int)

Phạm vi giá trị: -32768 đến 32767.

Biểu diễn trong máy: Có thể biểu diễn một hằng nguyên dưới dạng thập phân, bát phân (cơ số 8) hay thập lục phân (cơ số 16). Ngôn ngữ C quy định để biểu diễn một số dưới dạng bát phân bằng cách thêm 0 ở đầu, dưới dạng thập lục phân bằng cách thêm 0x. Ví dụ:

Giá trị thập phân	Dưới hệ bát phân	Dưới hệ thập lục phân
198	0306	0xC6

Hằng nguyên dài (long int)

Một hằng nguyên dài được viết như hằng nguyên nhưng thêm chữ l hay L ở cuối giá trị để biểu thị đó là hằng có giá trị **long**. Ví dụ: 198l, 1234562L.

Một hằng số nguyên vượt ra ngoài phạm vi giá trị cho phép của hằng nguyên cũng được hiểu là hằng *long*. Ví dụ: 1234567 được tự động hiểu là 1234567L.

Hằng dấu phẩy động

Có hai cách viết giá trị của một hằng dấu phẩy động:

(i) Cách viết thông thường: hằng được viết giống như trong thực tế nhưng thay dấu phẩy thập phân bằng dấu chấm (theo ký pháp viết số kiểu Anh & Mỹ); chẳng hạn: `359.12, -415.16`.

(ii) Cách viết theo ký pháp khoa học: hằng được viết gồm hai phần: phần định trị `mmmm.nnn` và phần mũ; phần định trị là số thực; phần mũ là số nguyên. Chẳng hạn: `12.3E+3` có giá trị bằng `12.3 * 10³ = 12300.0`. Do ta có thể tăng, giảm giá trị của phần mũ và dịch chuyển dấu chấm thập phân trong phần định trị nên số thực được viết dưới dạng này có tên gọi là số thực dấu phẩy động, còn cách viết thông thường có tên gọi số thực dấu phẩy tĩnh.

Hằng ký tự

Trong ngôn ngữ C người ta có thể biểu diễn hằng ký tự:

(i) bằng ký hiệu trong bảng mã ASCII đặt giữa hai dấu nháy đơn (`'`), ví dụ: `'A'`, `'b'`.

(ii) bằng cặp bao gồm ký tự `'\'` và số thứ tự của ký tự trong bảng mã ASCII.

Xét bảng sau:

Ký tự theo cách biểu diễn thứ nhất	Biểu diễn theo cách thứ hai
'A'	'\65' hoặc '\0101' hoặc '\0x41'

Có một số hằng ký tự đặc biệt được viết theo quy ước như sau:

Viết	Ký tự	Diễn giải
`'\'' `	'	Dấu nháy đơn
`'\"'`	"	Dấu nháy kép
`'\\'`	\	Dấu gạch chéo ngược
`'\n'`	\n	Ký tự xuống dòng
`'\0'`	\0	Ký tự NUL
`'\t'`		Ký tự tab

Hằng xâu ký tự

Hằng xâu ký tự được đặt trong hai dấu nháy kép `""`. Hằng xâu được lưu trữ trên một mảng ký tự mà ký tự cuối là ký tự NUL ('\0').

2. BIỂU THỨC

2.1 Biểu thức trong C

Biểu thức là kết quả ghép nối các toán tử và các toán hạng để diễn đạt một công thức nào đó. Các toán hạng có thể là một đại lượng nào đó như: hằng, biến, lời gọi hàm hay một biểu thức con. Số ngôi của một toán tử xác định số toán hạng kết hợp với toán tử đó. Thông thường ta có các toán tử hai ngôi và một ngôi. Mỗi biểu thức có một giá trị. Tuỳ theo giá trị của biểu thức mà ta có các biểu thức nguyên hay biểu thức thực. Chú ý rằng các biểu thức logic được xem là có giá trị nguyên, trong đó giá trị khác 0 tương ứng với mệnh đề đúng, còn giá trị 0 tương ứng với mệnh đề sai. Các dấu ngoặc đơn được sử dụng trong trong các biểu thức trong ngôn ngữ C đúng với ý nghĩa đại số của chúng.

So với PASCAL, ngôn ngữ C đưa ra nhiều quan niệm mới về biểu thức. Đó là các biểu thức gán và biểu thức điều kiện.

Trong chương trình, biểu thức được sử dụng trong:

+ vế phải của lệnh gán,

+ làm tham số thực của các hàm,

+ làm chỉ số,

+ các câu lệnh **if, for, while, do while,**

+ các biểu thức phức tạp hơn.

2.2 Các phép toán số học

Toán tử	Ý nghĩa	Ví dụ
-	Đổi dấu một số thực hoặc nguyên	`-12, -a`
+	Cộng hai số thực hoặc nguyên	`12+13.5, a+b`
-	Trừ hai số thực hoặc nguyên	`15-10, 12.1-3.2, a-b`
*	Nhân hai số thực hoặc nguyên	`a*b, 12*5, 12.3 *2`
/	Chia hai số thực hoặc nguyên	`a/b, 23/3, 23.3/2`
%	Lấy phần dư của phép chia hai số nguyên	`a%b 12 %5`

Các phép toán + và - (hai ngôi) có cùng độ ưu tiên và nhỏ hơn số ưu tiên của hai phép toán * và /. Bốn phép toán này lại có độ ưu tiên thấp hơn so với phép trừ một ngôi.

Phép chia với hai toán hạng nguyên cho kết quả là số nguyên (tương tự như phép `div` trong PASCAL). Chẳng hạn, `7/4` cho kết quả là `1`, còn `7.0/4` hoặc `7/4.0` hoặc `7.0/4.0` tất cả đều có kết quả `1.75`.

2.3 Các phép toán quan hệ

Toán tử	Ý nghĩa	Ví dụ
>	So sánh lớn hơn. Toán hạng là số nguyên hay số thực.	a>b 3>7 có giá trị 0
>=	So sánh lớn hơn hoặc bằng. Toán hạng là số nguyên hoặc số thực	a>=b 7>=7 có giá trị 1
<	So sánh nhỏ hơn. Toán hạng là số nguyên hay số thực	a<b 6<7 có giá trị 1
<=	So sánh nhỏ hơn hoặc bằng. Toán hạng là số nguyên hay số thực	a<=b 3<=7 có giá trị 1
==	So sánh bằng nhau. Toán hạng có thể là số nguyên hay số thực	a==b 7==7 có giá trị 1 6==7 có giá trị 0
!=	So sánh không bằng nhau.	a!=b 7!=7 có giá trị 0

Bốn phép toán đầu có cùng mức độ ưu tiên, hai phép sau có cùng độ ưu tiên nhưng thấp hơn. Các phép toán quan hệ có độ ưu tiên thấp hơn so với các phép toán số học cho nên biểu thức `i<n-1` được hiểu là `i<(n-1)`.

2.4 Các phép toán logic

Trong C sử dụng ba toán tử logic (liên từ logic) giống như trong các ngôn ngữ khác. Điều cần phân biệt chính là ký hiệu của các toán tử và giá trị của biểu thức logic:

Hai phép toán `&&` và `||` có độ ưu tiên thấp hơn so với các phép toán quan hệ. Tất cả các phép toán này lại có độ ưu tiên thấp hơn phép phủ định một ngôi.

Các phép toán quan hệ và logic được sử dụng để thiết lập điều kiện rẽ nhánh trong lệnh if và điều kiện kết thúc chu trình trong các câu lệnh **for**, **while** và **do{...} while**. Giá trị của toán hạng khác 0 sẽ được coi là TRUE và ngược lại giá trị bằng 0 sẽ được coi là FALSE. Giá trị biểu thức quan hệ và logic sẽ bằng 1 nếu đúng và bằng 0 nếu sai. Bảng sau đây mô tả ý nghĩa của ba liên từ logic nói trên:

Liên từ	Ý nghĩa	Ví dụ
&&	Liên kết hai biểu thức logic. Giá trị biểu thức bằng 1 khi cả hai toán hạng có giá trị 1	ch >='0' && ch <='9' 3<7 && 8>6 có giá trị 1
||	Liên kết hai biểu thức quan hệ. Giá trị của biểu thức là 1 khi một trong hai toán hạng bằng 1	n <=0 || n >=100 7<=3 || 8>=3 có giá trị 1

!	Phép phủ định một ngôi.	!a !12 có giá trị 0 !0 có giá trị 1

2.5 Các phép toán thao tác trên bit

Đây là các phép toán thường gặp ở hợp ngữ hơn là ở ngôn ngữ cấp cao, chúng cho phép xử lý đến từng bit của một số nguyên. Các phép toán này không được dùng cho float hay double.

Phép toán	Ý nghĩa
&	VÀ nhị phân
|	HOẶC nhị phân
^	HOẶC có loại trừ
<<	Dịch trái
>>	Dịch phải
~	Bù 1

Ta nhắc lại rằng

```
1 & 1 = 1     1 | 1 = 1     1 ^ 1 = 0     ~ 1  = 0
1 & 0 = 0     1 | 0 = 1     1 ^ 0 = 1     ~ 0 = 1
0 & 1 = 0     0 | 1 = 1     0 ^ 1 = 1     a<<n = a*2^n
0 & 0 = 0     0 | 0 = 0     0 ^ 0 = 0     a>>n = a/2^n
```

Chú ý

Cũng như hợp ngữ, C phân biệt:

(i) các phép dịch chuyển số học, thực hiện trên các giá trị nguyên, bảo toàn bit dấu,

(ii) 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.

Sau đây là một số ví dụ minh họa:

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

```
(-256) >> 2  = - 64
~0xa1b6 = 0x5e49
```

2.6 Biểu thức gán

Biểu thức gán có dạng

<Biến> = <Biểu_thức>

Thực hiện một biểu thức gán thực chất là lấy giá trị của **<Biểu thức>** gán cho **<Biến>.**

Nếu chúng ta thêm dấu ";" vào sau biểu thức gán thì sẽ thu được một câu lệnh gán.

Biểu thức gán có thể được sử dụng trong các phép toán và các câu lệnh như các biểu thức thông thường. Chẳng hạn khi viết

```
a=b=5;
```

thì điều đó có nghĩa rằng gán giá trị của biểu thức b = 5 cho biến a. Kết quả là b = 5 và a = 5. Tương tự câu lệnh

```
x = (a=5)(b=10);
```

sẽ gán 5 cho a, 10 cho b và sau đó gán tiếp 50 cho x.

C cho phép sử dụng một số ký pháp đặc biệt của phép toán gán sau đây:

Dạng viết thông thường	Dạng viết thu gọn	Ý nghĩa
i = i + expr	i += expr	Tự cộng
i = i - expr	i -= expr	Tự trừ
i = i * expr	i *= expr	Tự nhân
i = i / expr	i /= expr	Tự chia
i = i % expr	i %= expr	Tự chia dư
i = i & expr	i&=expr	Tự và
i=i|expr	i |=expr	Tự hoặc
i=i^expr	i^=expr	Tự bù
i=i>>expr	i>>=expr	Tự dịch bit phải
i=i<<expr	i<<=expr	Tự dịch bit trái

2.7 Biểu thức điều kiện

Biểu thức điều kiện có dạng:

```
expr1 ? expr2 : expr3
```

trong đó `expr1`, `expr2`, `expr3` là các biểu thức nào đó. Nếu `expr1` có giá trị khác 0 thì biểu thức điều kiện có giá trị cho bởi `expr2`, trái lại giá trị của `expr3` được dùng làm giá trị của biểu thức điều kiện. Kiểu của biểu thức điều kiện phụ thuộc vào kiểu của hai biểu thức `expr2` và `expr3` và là kiểu bao trùm nhất.

Biểu thức điều kiện cũng được dùng như các biểu thức khác. Ví dụ có thể dùng câu lệnh sau để xác định giá trị lớn nhất giữa hai số a và b:

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

Còn câu lệnh sau đây xác định giá trị lớn nhất trong ba số a,b,c:

```
s=a>b?(a>c?a:c):(b>c?b:c);
```

2.8 Các phép toán tăng, giảm một đơn vị

Giả định có câu lệnh

```
num = num + 1;
```

Ta có thể đưa ra cách viết khác gọn hơn như sau:

```
num ++;
```

Toán tử `++` có tác dụng tăng giá trị của biến num (có kiểu số nguyên hay con trỏ) thêm 1.

Trong C có hai phép toán một ngôi để tăng và giảm các biến. Toán tử tăng `++` sẽ cộng thêm 1 vào toán hạng của nó, toán tử giảm `--` sẽ trừ toán hạng đi 1. Toán hạng ở đây chỉ có thể là một biến (nguyên, thực hoặc con trỏ) mà không thể là một hằng số hay một biểu thức. Các dấu phép toán `++` và `--` có thể đứng trước (tiền tố) hoặc đứng sau (hậu tố) toán hạng, như vậy chúng ta có thể viết:

Toán tử	Dạng tiền tố	Dạng hậu tố
Tăng 1 đơn vị	`++ n`	`n ++`
Giảm 1 đơn vị	`-- n`	`n --`

Sự khác nhau giữa dạng tiền tố và hậu tố nảy sinh khi phép toán nằm trong một biểu thức khác. Điểm khác nhau ở đây là cách sử dụng giá trị của biến toán hạng khi tính toán biểu thức bao. Với dạng tiền tố, ta thay đổi giá trị của biến trước khi sử dụng giá trị biến để tính toán các biểu thức bao quanh; còn với dạng hậu tố,

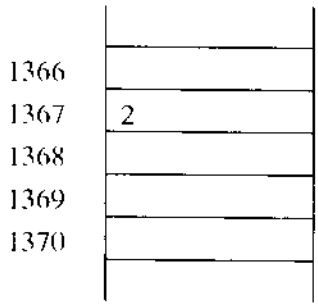
giá trị cũ của biến được sử dụng để tính toán biểu thức bao, rồi sau đó biến mới được thay đổi giá trị. Ta xét các trường hợp sau:

Giả sử trước mỗi phép tính `i=3` và `j=15`

Phép toán	Tương đương	Kết quả
`i = ++j;`	`j = j+1; i = j;`	`i =16` và `j =16`
`i = j++;`	`i = j; j = j+1;`	`i =15` và `j = 16`
`i++ ;`	`i = i + 1;`	`i = 4`
`j = ++ i + 5;`	`i = i + 1; j = i + 5;`	`i = 4` và `j = 9`
`j = i++ +5;`	`j = i + 5; i = i + 1;`	`i = 4` và `i = 8`

2.9 Phép toán lấy địa chỉ biến (&)

Bộ nhớ máy tính bao gồm nhiều bytes được đánh số từ 0 đến giới hạn của bộ nhớ. Các số này được gọi là số thứ tự và cũng là địa chỉ của các bytes bộ nhớ. Mỗi biến được khai báo sẽ chiếm giữ một vùng nào đó trong bộ nhớ và địa chỉ của nó là địa chỉ của byte đầu tiên trong vùng nhớ đó. Hình vẽ 2.1 cho thấy một biến nguyên có giá trị 2 và có địa chỉ là 1367.



Hình 2.1

Để xác định địa chỉ biến người ta sử dụng phép toán một ngôi & đặt trước tên biến. Như vậy, nếu:

```
int i;
```

thì

`&i`

xác định cho ta địa chỉ của vùng nhớ 2 bytes có tên là i. Không thể sử dụng phép toán & này cho các đối tượng khác như hằng số, biểu thức.

2.10 Phép toán chuyển đổi kiểu (bắt buộc)

Ta có thể chuyển đổi một kiểu bất kỳ sang kiểu mong muốn bằng cách dùng toán tử chuyển kiểu bắt buộc theo quy tắc như sau:

(<Kiểu>) <biểu thức>

trong đó **<Kiểu>** có thể là tên của một kiểu dữ liệu nào đó đã định nghĩa trước đó, như là **int, float, char**, ...

Một số **float** khi chuyển sang kiểu **int** sẽ bị chặt cụt phần thập phân. Một số kiểu **long** khi chuyển sang kiểu **int** sẽ bị cắt bỏ đi vài chữ số. Chẳng hạn `n` bằng `2520.40` thì `(int) n` bằng `2520`.

Chú ý

Việc chuyển kiểu ở trên diễn ra do yêu cầu của người lập trình. Trong khi thực hiện chương trình có thể có các chuyển đổi kiểu tự động như trong trường hợp sau: khi biểu thức gồm các toán hạng khác nhau về kiểu, các toán hạng có kiểu thấp hơn (phạm vi biểu diễn hẹp hơn) sẽ được chuyển đổi thành kiểu dữ liệu bao trùm hơn trước khi thực hiện phép toán và kết quả thu được sẽ có kiểu bao trùm nhất. Chuyển đổi kiểu tự động tuân theo sơ đồ sau:

char-->int-->long-->float-->double-->long double

2.11 Thứ tự ưu tiên và trật tự kết hợp của các phép toán

Thứ tự ưu tiên và trật tự kết hợp của các toán tử được sử dụng để xác định cách thức kết hợp các toán hạng với các toán tử khi tính toán biểu thức:

Biểu thức con trong ngoặc được tính toán trước các phép toán khác.

Phép toán một ngôi đứng bên trái toán hạng được kết hợp với toán hạng đi liền nó.

Khi một toán hạng đứng cạnh hai toán tử, ta phân biệt hai trường hợp:

(i) Nếu hai toán tử có độ ưu tiên khác nhau, toán tử nào có độ ưu tiên cao hơn sẽ được kết hợp với toán hạng,

(ii) Trường hợp trái lại, khi các toán tử giống nhau hoặc có độ ưu tiên như nhau thì trật tự kết hợp của các phép toán này sẽ xác định toán tử nào được kết hợp với phép toán. Các toán tử cùng độ ưu tiên sẽ tuân theo trình tự kết hợp hoặc từ trái sang phải hoặc từ phải sang trái.

Bảng 2.3 liệt kê thứ tự ưu tiên và trật tự kết hợp của các phép toán. Trong bảng có một số phép toán vẫn sẽ được giới thiệu tiếp trong các phần có liên quan.

Mức	Các toán tử	Trật tự kết hợp
1	`() [] ->`	-------->
2	`! ~ ++ -- - * & (type) sizeof`	<--------
3	`* /`	-------->
4	`+ -`	-------->
5	`<< >>`	-------->
6	`< <= > >=`	-------->
7	`= = !=`	-------->
8	`&`	-------->
9	`^`	-------->
10	`|`	-------->
11	`&&`	-------->
12	`||`	-------->
13	`?:`	<--------
14	`= += -= v.v...`	<--------

Bảng 2.3 Thứ tự ưu tiên của các phép toán

3. VÀO RA

Một trong những khó khăn khi học ngôn ngữ C là việc thực hiện các thao tác vào ra dữ liệu theo khuôn dạng. Cách thức để diễn tả quá trình nhập xuất dữ liệu trong chương trình như sau:

```
Giá trị cần đưa ra --->Kênh ra-->Khuôn dạng-->Thiết bị ra (Màn hình),
Thiết bị vào (Bàn phím) -->Kênh vào-->Khuôn dạng-->Biến cần nhập giá trị.
```

Các chương trình dịch C cung cấp cho người lập trình hai thư viện các hàm vào ra. Danh sách các hàm này có thể tìm thấy trong hai tệp tiêu đề `stdio.h` và `conio.h`, trong đó `stdio.h` khai báo các hàm vào ra chuẩn (standard input output) không phụ thuộc thiết bị và `conio.h` (console input output) khai báo các hàm vào ra sử dụng với các thiết bị vào ra xác định như là bàn phím, màn hình. Sau đây xin giới thiệu một vài hàm vào ra hay được sử dụng.

3.1 Hàm `printf()`

Cú pháp sử dụng hàm printf() như sau:

```
printf(<format> [, <arguments>, ... ] );
```

trong đó `<format>` là xâu "điều khiển" xác định cách thức hiển thị dữ liệu trên thiết bị ra chuẩn (thường là màn hình). Để thực hiện chức năng này trong xâu

điều khiển có chứa các "định dạng". Danh sách [, <arguments>, ...] nếu có sẽ chứa các giá trị (hằng, biến, biểu thức) được đưa ra. Các giá trị trong danh sách phân cách nhau bởi dấu phẩy ",". Số lượng các giá trị trong [, <arguments>, ...] bằng số "định dạng" có mặt trong xâu điều khiển <format>. Mỗi loại dữ liệu (nguyên, thực, ký tự, xâu ký tự) có thể tương ứng với một số "định dạng" nhất định. Điều này có nghĩa là tương ứng với một kiểu dữ liệu, có thể có nhiều hình thức trình bày khác nhau, nhưng không thể áp dụng một cách "tuỳ ý" các định dạng cho các giá trị bất kỳ. Chẳng hạn, một số nguyên có thể hiển thị dưới dạng ký pháp thập phân hay thập lục phân nhưng không thể hiểu đó là một địa chỉ bộ nhớ; một ký tự có thể hiển thị dưới dạng nguyên thuỷ của nó là một ký hiệu trong bảng mã ASCII hay là số thứ tự của ký tự đó, nhưng không thể hiển thị nó dưới khuôn dạng của một xâu ký tự hoặc một số thực. Việc sử dụng sai định dạng có thể đưa tới các kết quả bất bình thường.

Ở dạng đơn giản nhất, mỗi "định dạng" trong xâu điều khiển <format> là một cặp ký hiệu "%" và một ký hiệu mô tả định dạng. Bảng 2.4 liệt kê một số định dạng tiêu biểu:

Định dạng	Áp dụng cho kiểu dữ liệu	Ghi chú
%d	int	Đối là một số nguyên
%i	int	Đối được coi là hệ 10 có dấu (nếu giá trị âm)
%o	int	Đối được coi là số nguyên hệ 8 không dấu
%u	int	Đối số là số nguyên không dấu
%x	int	Đối số là số nguyên hệ 16 không dấu (không có 0x đứng trước).
%X	int	Số nguyên hệ 16 với các chữ số là A B C D E F
%e, %E	float, double	Đối số được chuyển sang dạng thập phân dấu chấm động.
%f, %lf	float, double	Đối số được chuyển sang dạng thập phân dấu phẩy tĩnh.
%g, %G	float, double	Đối số được chuyển sang dạng thập phân dấu phẩy tĩnh hay động tùy thuộc vào loại nào ngắn hơn
%c	char	Đối số là một ký tự
%s	char *	Đối số là một xâu ký tự.

Bảng 2.4 Một số định dạng tiêu biểu

Chương trình `printf1.c` sau đây minh hoạ cách sử dụng các "định dạng" đã được đề cập. Các chú thích kèm theo mô tả sự "nguy hại" của việc sử dụng sai định dạng.

```
/*printf1.c*/
#include <stdio.h>
void main(void) {
  printf("%d\n",-10);
  printf("%i\n",-10);
  printf("%u\n",-10);/*giá trị -10 được chuyển thành số nguyên không dấu 65526*/
  printf("%x\n",-10);
  printf("%o\n",-10);
  printf("%f\n",-10);/*sử dụng sai định dạng, kết quả sai*/
  printf("%c\n",'A');
  printf("%d\n",'A');
  printf("%f\n",123.4);
  printf("%e\n",123.4);
  printf("%E\n",123.4);
  printf("%s\n","Bo mon Cac he Thong Thong tin");
  printf("%d\n","Bo mon Cac he Thong Thong tin");/*sử dụng sai định dạng, kết
quả sai*/
}
```

```
-10
-10
65526
fff6
177766
-0.000000
A
65
123.400000
1.234000e+02
1.234000E+02
Bo mon Cac he Thong Thong tin
246
```

Để trình bày dữ liệu đẹp hơn, ta bổ sung thêm một số thuộc tính nữa trong các "định dạng" như là độ rộng (số ký tự) cần thiết để trình bày dữ liệu, căn lề dữ liệu bên phải hay bên trái ... Những thuộc tính như vậy thường đặt giữa dấu "%" và ký tự định dạng. Để có được thông tin đầy đủ về các thuộc tính này có thể tham khảo

các tài liệu trợ giúp đi kèm các chương trình dịch C. Bảng 2.5 giới thiệu một số thuộc tính hay sử dụng.

Bên cạnh các "định dạng", trong xâu điều khiển có thể chứa:

(i) Các ký tự thông thường. Các ký tự này được đưa ra cùng với giá trị của các đối số trong phần `[, arguments, ...]`.

(ii) Các ký tự điều khiển (có mã ASCII nhỏ hơn 32). Các ký tự này được sử dụng để gây một số hiệu ứng đặc biệt như xuống dòng ('`\n`'), nhảy trang ('`\f`') ...

Thuộc tính	Sử dụng với định dạng	Ví dụ minh hoạ
Độ rộng tối thiểu	`%d`	`printf("%6d",10);`
	`%f`	`printf("%6f",3.4);`
	`%s`	`printf("%6s","CNTT-DHBK");`
	`%c`	`printf("%6c",'A');`
	...	
Độ rộng dành cho phần thập phân	`%f`	`printf("%6.2f",3.45678);`

Bảng 2.5 Một số thuộc tính định dạng phổ biến

Như đã đề cập, danh sách các giá trị cần đưa ra `[,< arguments>, ...]` chứa các hằng, biến, hay một biểu thức. Giá trị của đối số sẽ được chuyển dạng và in ra theo các "định dạng" tương ứng. Cần nhắc lại là mỗi "định dạng" cần có một giá trị tương ứng. Khi một "định dạng" không tìm thấy đối số tương ứng hoặc khi kiểu dữ liệu của giá trị đưa ra không tương thích với ký tự định dạng thì máy sẽ bị nhầm và có thể đưa ra những kết quả vô nghĩa. Số giá trị đưa ra phải bằng số đặc tả. Ta xét chương trình sau đây:

```
/*printf2.c*/
#include <stdio.h>
void main(void){
printf("So nguyen %4d \nSo thuc %10.3f",10,3.4);
printf("\nKy tu %4c Ma ASCII %4d","A","A");
}
```

```
So nguyen   10
So thuc       3.400
Ky tu    A Ma ASCII   65
```

Cuối cùng có thể sử dụng hàm `printf()` mà không đưa ra giá trị nào khi đó

xâu điều khiển <format> không chứa bất kỳ "định dạng" nào. Khi đó bắt buộc xâu điều khiển phải là một hằng xâu ký tự mà không thể là biến kiểu xâu. Cụ thể:

có thể viết:

```
printf("Xau dieu khien la mot hang xau ky tu\n");
```

mà không viết được:

```
char *str="Noi dung bien xau ky tu";
printf(str);
```

3.2 Hàm scanf()

Cú pháp sử dụng hàm scanf() như sau:

```
scanf(<format> , {<address>, ...});
```

Hàm scanf() có nhiều chức năng chuyển dạng giống như hàm printf(). Hàm này đọc các ký tự từ thiết bị vào chuẩn (thông thường là bàn phím), biến đổi chúng tương ứng theo khuôn dạng được xác định bằng các "định dạng" trong xâu điều khiển <format>, rồi lưu trữ kết quả trong các đối số có địa chỉ được mô tả trong phần {<address>, ...}. Số lượng "định dạng" trong <format> cũng phải bằng số phần tử trong {<address>, ...}. Ví dụ, khi thực hiện các chỉ thị.

```
int n;
...
scanf("%d",&n);
```

Máy sẽ nhập từ bàn phím một số ký tự nạp vào trong một vùng bộ nhớ trong (gọi là vùng đệm bàn phím (stdin)). Quá trình thông dịch của hàm scanf() được thực hiện bằng cách đọc lần lượt các ký tự số có trong stdin cho đến khi gặp một ký tự khác ký tự số, các ký tự số vừa được duyệt sẽ được sử dụng để tạo giá trị của một số nguyên và số nguyên đó sẽ được đặt vào vùng nhớ xác định bởi địa chỉ của biến n. Cần nói thêm là, nếu không tìm thấy các ký tự số trong quá trình duyệt stdin, giá trị của biến n sẽ không bị thay đổi.

Trong trường hợp tổng quát xâu điều khiển <format> có thể chứa thêm những ký tự khác ngoài các "định dạng". Tuy nhiên, theo kinh nghiệm, các ký tự này không làm tăng khả năng của hàm scanf() mà lại rất khó hiểu. Vì vậy nếu chỉ mới học ngôn ngữ C thì nên sử dụng hàm scanf() với xâu điều khiển chỉ chứa các định dạng. Một khi đã có những kỹ năng nhất định ta có thể tìm hiểu những thứ đó nhờ các công cụ trợ giúp đi kèm chương trình dịch. Bảng 2.6 liệt kê các định dạng sử dụng với scanf().

Định dạng	Ý nghĩa
%d	Nhập vào một số nguyên; đối số tương ứng phải là địa chỉ của một biến con trỏ nguyên.
%o	Nhập vào một số nguyên hệ 8; đối số tương ứng phải là địa chỉ của một biến nguyên.
%x	Nhập vào một số nguyên hệ 16; đối số tương ứng phải là địa chỉ của một biến số nguyên.
%c	Nhập vào một ký tự; địa chỉ tương ứng phải là địa chỉ của biến ký tự.
%s	Nhập vào một xâu ký tự; đối số tương ứng phải là một xâu ký tự.
%f	Nhập vào một số thực **float**; đối số tương ứng phải là địa chỉ của một biến thực.
%ld	Nhập vào một số nguyên **long**.
%lf	Nhập vào một số thực **double**

Bảng 2.6 Các định dạng sử dụng trong hàm scanf()

Chương trình scanf1.c sau đây mô tả cách sử dụng scanf() để nhập giá trị cho các biến nguyên, thực, ký tự.

```
/*scanf1.c*/
#include <stdio.h>
void main() {
int n;
long l;
float f;
double d;
char c;
printf("Nhap gia tri cho cac bien\n");
scanf("%d%ld%f%lf%c",&n,&l,&f,&d,&c);
printf("So int %d\n",n);
printf("So long %d\n",l);
printf("So float %f\n",f);
printf("So double %f\n",d);
printf("Ky tu %c\n",c);}
```

```
Nhap gia tri cho cac bien
12
32
222.32
32.3
So int 12
So long 32
So float 222.320007
So double 32.300000
Ky tu
```

Ta có ba nhận xét sau về kết quả thực hiện của chương trình `scanf1.c`. Thứ nhất, giá trị của biến `f` được in ra khác tí chút so với giá trị đã nhập vào. Thứ hai, chương trình không đợi chúng ta nhập vào ký tự tương ứng với `c`. Thứ ba, giá trị của biến ký tự `c` nằm trong vùng ký tự không nhìn thấy được (có mã ASCII nhỏ hơn 32). Nhận xét thứ nhất thực ra không quan trọng lắm vì nó phụ thuộc vào cách biểu diễn số thực trong máy tính. Ta có các nhận xét sau gắn với cách thức đọc vùng đệm bàn phím `sdtin` của hàm `scanf()`.

Quy tắc 1. Khi đọc số, hàm `scanf()` quan niệm rằng tất cả các ký tự số, dấu chấm "." là ký tự hợp lệ còn các dấu phân cách là : dấu trắng, dấu tab, dấu xuống dòng.

Quy tắc 2. Khi đọc ký tự, hàm `scanf()` cho rằng mọi ký tự có trong `stdin` đều hợp lệ. Hàm `scanf()` trong trường hợp này không bỏ qua ký tự xuống dòng trong `stdin`.

Quy tắc 3. Khi đọc xâu ký tự, hàm `scanf()` sử dụng các dấu phân cách như trong đọc số.

Dựa vào quy tắc thứ hai ta có thể giải thích hai nhận xét liên quan đến ký tự `c` trong chương trình `scanf1.c`. Sau khi kết thúc nhập giá trị cho biến `d`, ta ấn phím Enter nên trong vùng đệm `stdin` sẽ chỉ có ký tự xuống dòng. Ký tự này sẽ được gán cho biến `c` tương ứng với mã của ký tự xuống dòng.

Theo kinh nghiệm, để nhập được đúng giá trị cho biến ký tự thì:

(i) phải sử dụng một chỉ thị `scanf()` riêng biệt cho mỗi lần nhập ký tự,

(ii) trước chỉ thị `scanf()`, thực hiện chỉ thị sau:

```
fflush(stdin);
```

Ta xét chương trình `scanf2.c` sau đây:

```
/*scanf2.c*/
#include <stdio.h>
void main() {
```

```
int n;
long l;
float f;
double d;
char c;
clrscr();
printf("Nhap gia tri cho cac bien\n");
scanf("%d%ld%f%lf",&n,&l,&f,&d);
fflush(stdin);
scanf("%c",&c);
printf("So int %d\n",n);
printf("So long %d\n",l);
printf("So float %f\n",f);
printf("So double %f\n",d);
printf("Ky tu %c\n",c);
}
```

```
Nhap gia tri cho cac bien
23
22
22
33
2
So int 23
So long 22
So float 22.000000
So double 33.000000
Ky tu 2
```

3.3 Sử dụng hàm `clrscr()` và `getch()` trong các chương trình

Hai hàm `clrscr()` và `getch()` được khai báo trong tập tin tiêu đề `conio.h`.

Về chức năng, lời gọi hàm

```
clrscr();
```

có tác dụng xoá màn hình giống như thủ tục `clrscr` trong Pascal.

Hàm `getch()` có thể sử dụng với nhiều mục đích khác nhau. Tuy nhiên với chỉ thị:

```
getch();
```

chương trình C làm việc giống như đoạn chương trình Pascal

```
repeat
until keypressed;
```

Chương trình sau đây minh hoạ cách sử dụng `clrscr()` và `getch()`

```
/*scanf3.c*/
/*stdio.h chứa các khai báo của printf() và scanf()*/
#include <stdio.h>
/*conio.h chứa các khai báo của clrscr() và getch()*/
#include <conio.h>
void main() {
int n;
long l;
float f;
double d;
char c;
clrscr();
printf("Nhap gia tri cho cac bien\n");
scanf("%d%ld%f%lf",&n,&l,&f,&d);
fflush(stdin);
scanf("%c",&c);
printf("So int %d\n",n);
printf("So long %d\n",l);
printf("So float %f\n",f);
printf("So double %f\n",d);
printf("Ky tu %c\n",c);
printf("An phim bat ky de thoat");
getch();
}
```

```
Nhap gia tri cho cac bien
23
23
```

```
3443
33
3
So int 23
So long 23
So float 3443.000000
So double 33.000000
Ky tu 3
An phim bat ky de thoat
```

4. CÁC CÂU LỆNH ĐIỀU KHIỂN CHƯƠNG TRÌNH

4.1 Câu lệnh đơn

Một biểu thức lệnh (gán, tăng giá trị) kết thúc bởi dấu ";" là một câu lệnh đơn. Một lời gọi hàm đi sau bằng một dấu ";" cũng là một câu lệnh đơn.

Ví dụ

```
x = 0 ;
i++, printf("\n");
```

bao gồm hai câu lệnh hợp lệ; câu lệnh thứ nhất là một lệnh gán; câu lệnh tiếp theo bao gồm hai thành phần được thực hiện liên tiếp nhau: tăng giá trị cho biến i và gọi hàm thư viện chuẩn printf().

Trong C, dấu ";" dùng để kết thúc một câu lệnh đơn.

4.2 Câu lệnh ghép

Câu lệnh ghép là tập hợp các câu lệnh được bao bởi hai dấu "{" và "}". Về cú pháp, ở đâu có thể đặt một câu lệnh đơn ở đó có thể đặt khối lệnh. Phần thân của một hàm cũng là một câu lệnh ghép. Lưu ý rằng không cần đặt dấu ";" sau một khối lệnh, mặc dù điều đó thì không sai.

Các lệnh thành phần bên trong câu lệnh ghép sẽ được thực hiện một cách tuần tự theo thứ tự xuất hiện. Đây chính là cấu trúc lệnh tuần tự trong các ngôn ngữ lập trình cấp cao. Trong một khối lệnh có thể có các khai báo biến và hằng (khác so với Pascal); ngôn ngữ C chỉ qui định là tất cả các khai báo nếu có phải đặt trước các câu lệnh. Chẳng hạn, không thể khai báo một biến nguyên sau khi đã gọi đến hàm xoá màn hình.

4.3 Câu lệnh if [else]

Câu lệnh **if** [**else**] được dùng để lựa chọn quyết định. Một cách hình thức, cú pháp của câu lệnh như sau:

```
if (expr)
        <lệnh 1>;
[else <lệnh 2>]
```

trong đó [**else**] là phần tùy chọn. Biểu thức `expr` được tính, nếu nó đúng (có giá trị khác 0) thì `<lệnh 1>` được thực hiện. Trái lại, nếu có phần [**else**] thì `<lệnh 2>` sẽ được thực hiện. Trường hợp không có phần else chương trình sẽ bỏ qua `<lệnh 1>` và chuyển điều khiển sang câu lệnh ngay sau lệnh **if**.

Cặp dấu ngoặc đơn "()" bao quanh biểu thức điều kiện là hết sức cần thiết vì nếu không có chương trình sẽ hiểu sai ý nghĩa biểu thức điều kiện dẫn đến biên dịch sai chương trình. Xét ví dụ sau:

```
if a>b
   max = a;
```

Chương trình dịch sẽ hiểu rằng a đóng vai trò biểu thức điều kiện còn `>b` được hiểu là bắt đầu phần thân câu lệnh **if**. Rõ ràng điều này sẽ gây những hiểu lầm tai hại.

Hơn thế nữa, người mới học ngôn ngữ C có thể tình cờ đưa thêm dấu ";" ngay sau biểu thức điều kiện và kết quả là chương trình sẽ cho ta những kết quả không thể kiểm soát được. Vấn đề ở chỗ dấu chấm phẩy đứng ngay sau điều kiện làm cho chương trình dịch hiểu là câu lệnh rỗng trong các ngôn ngữ lập trình cấp cao.

Phần lệnh sau **if** hay **else** có thể là một câu lệnh đơn, lệnh ghép hay một cấu trúc điều khiển chương trình khác. Do vậy cho phép các lệnh **if** [**else**] có thể lồng nhau. Do phần **else** của **if** [**else**] là tùy chọn nên có thể xảy ra nhập nhằng khi một **else** bị bỏ qua trong một dãy các câu lệnh **if** lồng nhau. Vấn đề này được giải quyết theo cách thông thường như sau: **else** được gắn với **if** không có **else** ở gần nhất trước đó. Chẳng hạn, trong:

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

thì **else** sẽ tương ứng với **if** bên trong. Nếu không muốn như vậy thì phải dùng dấu ngoặc nhọn để gắn **else** với **if** tương ứng, chẳng hạn.

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

Cần lưu ý rằng phải có dấu chấm phẩy sau z = a trong câu lệnh:

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

Yêu cầu này có nguyên nhân sâu xa liên quan đến cú pháp của ngôn ngữ C.

Cũng có thể gặp trường hợp dùng câu lệnh **if** đi sau phần **else** để chọn một trong nhiều quyết định khác nhau.

Chú ý

Vì một **if** chỉ kiểm tra giá trị số của biểu thức nên có thể viết gọn hơn nữa, theo cách:

```
        if (expr)
```

thay cho

```
        if (expr != 0)
```

Bài tập mẫu 2.1

Lập chương trình nhập vào ba số thực `a, b, c`. Sau đó tính giá trị lớn nhất giữa ba số.

Cách 1: Sử dụng **if** [**else**] lồng nhau, khó tổng quát cho n số với n bất kỳ.

```
/*if1.c*/
#include <stdio.h>
#include <conio.h>
main(){
  float a, b, c ;
  float max;
  clrscr();
```

```
printf("Nhap vao ba so thuc a b c\n");
printf(" a = "); scanf("%f",&a);
printf(" b = "); scanf("%f",&b);
printf(" c = "); scanf("%f",&c);
if ( a < b )
  if (b < c)
     max=c;
  else max=b;
else if ( a < c )
        max = c;
else max=a;
printf("Gia tri lon nhat cua \n");
printf(" %6.2f % 6.2f %6.2f \n",a,b,c);
printf("la\n %6.2f",max);
getch();
}
```

```
Nhap vao ba so thuc a b c
 a = 12
 b = 45
 c = 6
Gia tri lon nhat cua
  12.00  45.00   6.00
la
  45.00
```

Cách 2: Sử dụng nhiều câu lệnh **if** tuần tự, có thể tổng quát cho n số bất kỳ.

```
/*if2.c*/
#include <stdio.h>
#include <conio.h>
main(){
  float a, b, c ;
  float max;
  clrscr();
  printf("Nhap vao ba so thuc a b c\n");
  printf(" a = "); scanf("%f",&a);
  printf(" b = "); scanf("%f",&b);
```

```
 printf(" c = "); scanf("%f",&c);
 max = a;
 if ( max < b )
   max = b;
 if ( max < c )
   max = c;
 printf("Gia tri lon nhat cua \n");
 printf(" %6.2f % 6.2f %6.2f \n",a,b,c);
 printf("la\n %6.2f",max);
 getch();
 }
```

```
Nhap vao ba so thuc a b c
 a = 12
 b = 45
 c = 6
Gia tri lon nhat cua
  12.00  45.00   6.00
la
  45.00
```

4.4 Câu lệnh while

Cú pháp của while như sau:

```
while ( expr )
        <lệnh>;
```

Về ngữ nghĩa, câu lệnh **while** trong C giống hệt câu lệnh **while do** trong PASCAL. Cách thức xử lý như sau: đầu tiên, biểu thức `expr` được tính. Nếu giá trị của biểu thức khác 0 thì `<lệnh>` (thường gọi là thân vòng **while**) sẽ được thực hiện và biểu thức `expr` được tính lại. Chu trình này tiếp tục cho tới khi biểu thức `expr` có giá trị 0. Tại thời điểm đó việc thực hiện chương trình chuyển sang câu lệnh đứng sau while.

Bài tập mẫu 2.2

Chương trình sau đây dùng vòng lặp while để tính tổng các số từ 1 đến 10.

```
/*while1.c*/
#include <stdio.h>
main(){
```

```
int count = 0;
int  total = 0; /*khai báo và khởi đầu giá trị*/
while (count <= 10)  /*lặp cho đến khi count là 10*/
 {
 total += count;
 printf("count = %d, total = %d\n",count++,total);
 }
}
```

```
count = 0, total = 0
count = 1, total = 1
count = 2, total = 3
count = 3, total = 6
count = 4, total = 10
count = 5, total = 15
count = 6, total = 21
count = 7, total = 28
count = 8, total = 36
count = 9, total = 45
count = 10, total = 55
```

Bài tập mẫu 2.3

Chương trình sau đây sử dụng vòng **while** để nhập ba số thực dùng làm số đo ba cạnh một tam giác.

```
/*tgwhile.c*/
#include<stdio.h>
void main() {
 float a,b,c;
 printf("Nhap vao do dai ba canh tam giac:\n");
 scanf("%f%f%f",&a,&b,&c);
/*nếu ba số a,b, c chưa thoả mãn bất đẳng thức tam giác, thì nhập lại cho tới khi thoả
mãn thì thôi*/
 while(a+b<=c||b+c<=a||c+a<=b){
    printf("Not OK\n");
    scanf("%f%f%f",&a,&b,&c);
    }
 printf("OK");
 }
```

```
Nhap vao do dai ba canh tam giac:
1 2 3
Not OK
1 2 1
Not OK
3 4 5
OK
```

4.5 Câu lệnh do{...}while

Cú pháp:

```
do {
    <lệnh>
}while ( expr );
```

Câu lệnh **while** kiểm tra điều kiện kết thúc chu trình ngay khi bắt đầu chu trình, do đó rất có thể phần thân chu trình không được thực hiện lần nào. Trái lại câu lệnh **do{...}while** kiểm tra điều kiện điều kiện này ở cuối chu trình, sau mỗi lần đi qua thân chu trình. Do vậy thân chu trình bao giờ cũng được thực hiện ít nhất một lần.

Câu lệnh **do{...}while** so với **repeat until** trong PASCAL có một vài điểm khác biệt. Trong **repeat until**, vòng lặp còn được thực hiện chừng nào biểu thức điều kiện đi sau từ khoá **until** còn chưa đúng (nghĩa là sai), ngược lại trong **do{...}while** ta lặp lại công việc khi biểu thức điều kiện sau từ khoá **while** vẫn còn có giá trị khác 0 (vẫn còn đúng). Nói cách khác, điều kiện kết thúc trong vòng **repeat until** và **do{...}while** là ngược nhau. Hãy so sánh đoạn chương trình nhập vào một số nguyên dương bằng ngôn ngữ C và PASCAL:

```
int n;
...
do {
printf("n = ");
scanf("&d",&n);
}while (n<=0);
```

```
n:integer;
...
repeat
write('n= ');
readln(n);
until(n>0);
```

Phần thân câu lệnh **repeat until** có thể là một dãy các lệnh tuy nhiên không cần sử dụng hai từ khoá xác định một câu lệnh khối, trái lại với **do {...} while** điều này là bắt buộc ngay cả khi bên trong **do {...} while** không thực hiện bất cứ lệnh nào. Chương trình sau đây sử dụng vòng lặp **do {...}while** để nhập ba số nguyên dùng làm ba cạnh một tam giác.

Bài tập mẫu 2.4

```
/*tgdowhile.c*/
#include<stdio.h>
void main() {
 float a,b,c;
 printf("\nNhap vao do dai ba canh tam giac:\n");
 do {
    scanf("%f%f%f",&a,&b,&c);
    if (a+b<=c||b+c<=a||c+a<=b)
       printf("Not OK\n");
    }while(a+b<=c||b+c<=a||c+a<=b);
 printf("OK");
}
```

```
Nhap vao do dai ba canh tam giac:
1 2 3
Not OK
1 2 1
Not OK
3 4 5
OK
```

4.6 Câu lệnh for

Cú pháp:

```
for ( [expr1]; [expr2]; [expr3])
        <lệnh>
```

Có thể diễn đạt cú pháp này bằng vòng lặp while như sau:

```
expr1;
while ( expr2) {
        <lệnh>
        expr3;
    }
```

Ba thành phần của for đều là biểu thức. Thông thường `expr1` và `expr3` là các phép gán hoặc lời gọi hàm, còn `expr2` là biểu thức quan hệ hoặc logic.

Theo cách biểu diễn tương đương bằng vòng while, ta thấy rằng câu lệnh thực hiện biểu thức `expr1` được thực hiện đầu tiên và duy nhất một lần, sau đó vòng lặp **for** tính giá trị biểu thức quan hệ `expr2`, nếu cho giá trị khác 0, nhóm lệnh trong phần thân vòng **for** được thực hiện, tiếp đến là thực hiện biểu thức `expr3`. Sau đó biểu thức điều kiện `expr2` được tính lại để kiểm tra sẽ thực hiện một lần lặp mới hay vòng **for** kết thúc. Một cách đơn giản, có thể hiểu ý nghĩa của vòng for như một vòng lặp với số bước xác định, trong đó `expr1` chứa các phép gán khởi đầu giá trị cho các biến điều khiển, `expr3` mô tả sự thay đổi giá trị của các biến điều khiển, còn `expr2` được sử dụng như một phương tiện xác định số bước lặp của vòng for. Xét chương trình in ra 100 số tự nhiên đầu tiên.

Bài tập mẫu 2.5

```
/*for1.c*/
#include <stdio.h>
void main() {
  int i;
  for(i=1; i<=100; i++)
    printf("%4d",i);
  }
```

```
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

Có thể thấy rằng chương trình này được chuyển đổi từ chương trình PASCAL sau:

```
/*for1.pas*/
program for1;
var i:integer;
begin
  for i:=1 to 100 do
    write(i:4);
end.
```

Tuy nhiên vòng **for** trong ngôn ngữ C có thể có nhiều biến điều khiển. Điều đó làm mất đi tính chất "có số vòng lặp xác định" nhưng lại tăng tính mềm dẻo cho vòng **for**. Trong nhiều trường hợp điều này còn làm cho chương trình dễ đọc hơn. Chương trình sau đây sử dụng vòng **for** với hai biến điều khiển để in các số tự

nhiên trong phạm vi 100 thành các cặp sao cho hai số trong một cặp có tổng bằng 101.

Bài tập mẫu 2.6

```
/*for2.c*/
#include<stdio.h>
#include<conio.h>
void main() {
  int i,j;
  clrscr();
  for(i=1,j=100;i<j;i++,j--)
    printf("(%d,%d)",i,j);
  }
```

```
(1,100) (2,99) (3,98) (4,97) (5,96) (6,95) (7,94) (8,93) (9,92) (10,91) (11,90) (12,89) (13,
88) (14,87) (15,86) (16,85) (17,84) (18,83) (19,82) (20,81) (21,80) (22,79) (23,78) (24,77)
(25,76) (26,75) (27,74) (28,73) (29,72) (30,71) (31,70) (32,69) (33,68) (34,67) (35,66) (36
,65) (37,64) (38,63) (39,62) (40,61) (41,60) (42,59) (43,58) (44,57) (45,56) (46,55) (47,54
) (48,53) (49,52) (50,51)
```

Về mặt cú pháp, một trong ba biểu thức thành phần của vòng **for** có thể bỏ qua, nhưng phải có dấu ";". Khi `expr2` không có mặt, thì ta coi biểu thức điều kiện luôn luôn đúng, cho nên

```
for (...; ;... ) {
   ...
   }
```

là một chu trình vô hạn. Có thể thoát ra khỏi vòng lặp vô hạn nhờ các câu lệnh ngắt như là **break** hoặc **return** (sẽ đề cập trong các mục tiếp theo).

Cuối cùng, cũng giống như các câu lệnh có cấu trúc khác, thân của vòng **for** có thể chứa một vòng **for** khác tạo thành các vòng **for** lồng nhau. Xét chương trình sau đây in ra bảng cửu chương.

Bài tập mẫu 2.7

```
/*for3.c*/
#include <stdio.h>
#include <conio.h>
main(){
int cols, rows;
```

```
clrscr();
for (rows = 1;rows <11; rows ++) {
 for (cols =1; cols <11; cols ++)
            printf("%4d",cols * rows);
            printf("\n");
      }
getch();
}
```

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```

4.7 Câu lệnh break và continue

Lệnh **break**

Trong một số trường hợp ta cần phải thoát khỏi một chu trình mà không cần thực hiện kiểm tra điều kiện lặp. Câu lệnh **break** cho phép thoát ra khỏi các vòng lặp **for**, **while**, **do while**. Câu lệnh **break** cho phép chương trình thoát ngay lập tức ra khỏi vòng lặp trong cùng chứa nó. Sau đây là ví dụ dùng lệnh **break** để thoát ra khỏi vòng lặp **do {...} while** vô hạn.

```
/*break.c*/
/*Nhập vào các số nguyên cho đến khi gặp một số dương thì dừng*/
#include <stdio.h>
#include <conio.h>
void main() {
  int n;
  clrscr();
  do {
    printf("\nn=");
    scanf("%d",&n);
```

```
    if(n>0)
      break;
    else printf("Not OK");
   }while(1);
  printf("OK so duong");
  }
```

```
n=-3
Not OK
n=-4
Not OK
n=4
OK so duong
```

Lệnh **continue**

Khi gặp câu lệnh **continue** chương trình không kết thúc nốt vòng lặp chứa nó mà bỏ qua các câu lệnh còn lại trong thân vòng lặp để bắt đầu một lần lặp mới. Chương trình ví dụ sau đây minh hoạ cách sử dụng lệnh **continue** để in ra các số dương trong số 100 lần nhập.

Bài tập mẫu 2.8

```
/*continue.c*/
#include <stdio.h>
#include <conio.h>
void main() {
  int i,n,s=0;
  clrscr();
  for(i=1; i<=10; i++) {
  printf("\n Lan nhap thu : %d",i);
  printf("\n n = ");
  scanf("%d",&n);
  if (n<=0) continue;
  printf(" Da nhap so duong %d",n);
  s+=n;
   }
 printf("\n Tong cac so duong %d",s);
  }
```

```
Lan nhap thu : 1
 n = 3
 Da nhap so duong 3
 Lan nhap thu : 2
 n = -2
 Lan nhap thu : 3
 n = 3
 Da nhap so duong 3
 Lan nhap thu : 4
 n = -4
 Lan nhap thu : 5
 n = -5
 Lan nhap thu : 6
 n = -6
 Lan nhap thu : 7
 n = -8
 Lan nhap thu : 8
 n = 2
 Da nhap so duong 2
 Lan nhap thu : 9
 n = -3

 Tong cac so duong da nhap 8
```

4.8 Câu lệnh switch

Cú pháp của câu lệnh **switch** như sau:

```
switch ( expr ) {
        case <value1>: <lệnh1>; [break;]
        case <value2>: <lệnh2>; [break;]
        ....
        case <valuen>: <lệnhn>; [break;]
        [default  :       [break;]]
}
```

Câu lệnh **switch** cho phép ra quyết định nhiều nhánh hoàn toàn tương tự như câu lệnh **Case of** trong PASCAL. Nó kiểm tra xem một biểu thức `expr` có giá trị ứng với một trong số các giá trị hằng hay không và nhảy đến câu lệnh tương ứng khi so trúng.

Câu lệnh **switch** tính biểu thức số trong ngoặc và so sánh giá trị tính được của biểu thức với các giá trị đứng sau **case**. Mỗi trường hợp đều được đánh nhãn bởi **case**, tiếp theo là một giá trị (số nguyên, ký tự hoặc biểu thức hằng). Nếu so trúng thì sẽ thực hiện nhóm lệnh tương ứng với trường hợp đó. Không cần phải đặt các câu lệnh ứng với một **case** trong khối lệnh. Trường hợp có nhãn **default** các

câu lệnh tương ứng với nhãn này được thực hiện nếu không có trường hợp nào thoả mãn. Chú ý rằng **default** là phần tùy chọn; nếu không có sẽ không thực hiện gì cả. **case** và **default** có thể xuất hiện theo bất kỳ trật tự nào. **case** tương ứng với các trường hợp khác nhau.

Chương trình sau đây đếm số lượng số 1, 50 và 100 được nhập trong số 10 lần nhập số nguyên.

Bài tập mẫu 2.9

```
#include <stdio.h>
#include <conio.h>
void main() {
  int i,n,c1=0,c50=0,c100=0;
  clrscr();
  for(i=1;i<=10;i++) {
      printf(" Lan nhap thu : %d",i);
      printf("\n n = ");
      scanf("%d",&n);
      switch(n) {
       case 1:  c1++;break;
       case 50: c50++;break;
       case 100:c100++;break;
       }
    }
  printf("So lan nhap so 1: %d\n",c1);
  printf("So lan nhap so 1: %d\n",c50);
  printf("So lan nhap so 1: %d\n",c100);
  }
```

```
Lan nhap thu : 1
 n = 2
 Lan nhap thu : 2
 n = 1
 Lan nhap thu : 3
 n = 50
 Lan nhap thu : 4
 n = 100
 Lan nhap thu : 5
 n = 100
 Lan nhap thu : 6
```

```
 n = 100
 Lan nhap thu : 7
 n = 1
 Lan nhap thu : 8
 n = 1
 Lan nhap thu : 9
 n = 50
 Lan nhap thu : 10
 n = 50
So lan nhap so 1: 3
So lan nhap so 1: 3
So lan nhap so 1: 3
```

Trong **switch** câu lệnh **break** tạo ra sự phân cách giữa các nhóm lệnh tương ứng với các trường hợp khác nhau. Điều đó cho phép thoát ngay khỏi **switch** một khi các công việc tương ứng với một trường hợp đã được thực hiện xong. Nếu muốn có nhiều nhãn **case** sử dụng chung một nhóm lệnh nào đó ta nên đặt chúng cạnh nhau, giữa các nhãn **case** này sẽ không đặt lệnh **break**, khi đó nhóm lệnh của các nhãn **case** đi trước sẽ bao gồm cả các lệnh của các **case** đứng sau. Xét chương trình xác định số ngày trong một tháng dựa vào số hiệu chỉ tháng.

Bài tập mẫu 2.10

```
#include<stdio.h>
#include<conio.h>
void main(){
 int thang,songay;
 clrscr();
 printf("Cho vao con so chi thang : ");
 scanf("%d",&thang);
 switch(thang) {
 case 1:
 case 3:
 case 5:
 case 7:
 case 8:
 case 10:
 case 12: printf("So ngay trong thang %d la  31",thang);
            break;
 case 4:
 case 6:
```

```
case 9:
case 11:
  printf("So ngay trong thang %d la 30",thang);
  break;
case 2:
printf("So ngay trong thang %d la 28 hoac 29",thang);
    break;
   }
 }
```

```
Cho vao con so chi thang : 4
So ngay trong thang 4 la 30

Cho vao con so chi thang : 3
So ngay trong thang 3 la 31
Cho vao con so chi thang : 2
So ngay trong thang 2 la 28 hoac 29
```

4.9 Câu lệnh goto

Ngôn ngữ C cũng cung cấp cả câu lệnh goto để nhảy tới các nhãn. Đối với người mới học lập trình không nên sử dụng tuỳ tiện lệnh goto. Thực tế, ta có thể viết các chương trình mà không cần dùng đến câu lệnh goto. Giáo trình này nhằm cung cấp những kiến thức về ngôn ngữ C cho người mới học, do đó sẽ tránh không sử dụng lệnh goto trong các chương trình ví dụ.

5. TÓM TẮT

5.1 Ghi nhớ

Các câu lệnh khai báo có thể đặt bên ngoài các khối lệnh hoặc bên trong các khối lệnh, nhưng phải trước các câu lệnh điều khiển chương trình.

Khai báo biến bắt đầu bằng tên kiểu dữ liệu của biến, tiếp theo là tên biến; nếu có nhiều biến, tên các biến được cách nhau bởi các dấu phẩy. Câu lệnh khai báo biến được kết thúc bằng dấu chấm phẩy.

Định nghĩa hằng bằng toán tử #define hoặc sử dụng từ khoá const.

Câu lệnh lựa chọn if chỉ thực hiện câu lệnh đi kèm khi điều kiện được mô tả trong nó là đúng.

Cấu trúc if else có thể kiểm tra để thực hiện nhiều lựa chọn khác nhau. Nếu có nhiều điều kiện đúng, chỉ có câu lệnh đi sau điều kiện thỏa mãn đầu tiên được thực hiện.

Một câu lệnh ghép có thể đặt ở bất cứ đâu, tại đó một câu lệnh đơn có thể có mặt.

Một câu lệnh rỗng chỉ ra rằng không làm gì. Biểu thị câu lệnh rỗng bằng cách đặt dấu chấm phẩy thay thế cho cả câu lệnh.

Một câu lệnh lặp mô tả rằng một hành động được lặp đi lặp lại chừng nào biểu thức vẫn còn đúng.

Thông thường một vài thao tác bên trong phần thân câu lệnh lặp có xu hướng làm cho điều kiện của câu lệnh lặp trở nên bị sai.

Một giá trị có chứa phần không nguyên được xem là một số thực dấu phẩy động được biểu thị bằng kiểu dữ liệu `float`.

5.2 Các lỗi thường gặp khi lập trình

Quên một hoặc cả hai dấu { và } để bắt đầu và kết thúc một câu lệnh ghép.

Đặt một dấu chấm phẩy sau điều kiện trong câu lệnh điều kiện `if`, điều đó dẫn tới một lỗi logic.

Trong phần thân của câu lệnh while không có các câu lệnh nhằm làm cho điều kiện trong while trở nên sai. Thông thường, cấu trúc lặp như vậy sẽ không bao giờ kết thúc. Chúng ta gọi đây là lỗi "lặp vô hạn".

Viết các từ khóa với chữ hoa, ví dụ While thay vì while.

Chia một số cho 0 (lỗi nặng).

Sử dụng phép toán (toán tử) tăng hoặc giảm số học đối với các hằng hoặc biểu thức, ví dụ `++(i+1)` - lỗi cú pháp.

5.3 Một số thói quen lập trình tốt cần học tập

Tập thói quen viết chương trình có căn chỉnh lề để tăng tính dễ đọc.

Nếu có nhiều mức căn lề, mỗi mức phải được căn lề với cùng số khoảng trống.

Trước khi viết chương trình, nên viết sơ đồ khối mô tả thuật toán.

Một số lập trình viên thích viết trước các dấu { và } bắt đầu và kết thúc của một câu lệnh ghép trước khi viết các câu lệnh thành phần bên trong câu lệnh ghép.

Khi thực hiện phép chia cho một biểu thức mà giá trị của biểu thức có thể bằng 0. Trong trường hợp đó phải kiểm tra trước khi chia để tránh gây lỗi (fatal error).

Các toán tử một ngôi phải được đặt bên cạnh toán hạng của chúng không nên đặt thêm các dấu trắng vào giữa.

Không nên so sánh bằng nhau các giá trị số thực.

6. BÀI TẬP

Bài tập 2.1. In ra màn hình các dấu * theo các hình dạng sau:

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

trong đó hình sao cao nhất ở dòng trên cùng của màn hình, hình * bên trái nhất nằm ở cạnh trái của màn hình.

Bài tập 2.2 Nhập vào bốn số thực. Tìm giá trị lớn nhất, nhỏ nhất trong bốn số đó bằng cách sử dụng biểu thức điều kiện.

Bài tập 2.3 In ra bảng mã ASCII, trong đó mỗi cột bao gồm ký tự và mã của ký tự đó dưới dạng thập phân/thập lục phân/bát phân.

Bài tập 2.4 Nhập vào ba số thực dùng làm số đo ba cạnh một tam giác, sau đó xác định xem tam giác tương ứng có tính chất gì: đều, vuông cân, cân, vuông hay thường.

Bài tập 2.5 Nhập vào một số nguyên dương, kiểm tra xem nó có phải số nguyên tố hay không (Số nguyên tố sẽ không chia hết cho tất cả các số từ 2 đến số tự nhiên đi trước số đó).

Bài tập 2.6 Nhập vào một số nguyên dương n, liệt kê các số nguyên tố nhỏ hơn n.

Bài tập 2.7 Nhập vào một số nguyên dương, kiểm tra xem số đó có phải là số hoàn thiện hay không! (Số hoàn thiện là số có giá trị bằng tổng tất cả các ước số của nó không tính bản thân nó).

Bài tập 2.8 Nhập vào một số nguyên dương n, liệt kê các số hoàn thiện nhỏ hơn n.

Bài tập 2.9 Nhập vào các số thực cho đến khi gặp 0, đưa ra giá trị lớn nhất trong số các số vừa nhập.

Bài tập 2.10 Nhập vào các số nguyên cho đến khi gặp 0, đưa ra giá trị lớn nhất trong số các số chia hết cho 5 vừa nhập được.

Bài tập 2.11 Nhập vào một số nguyên dương, in ra số theo thứ tự ngược lại.

Bài tập 2.12 Nhập vào một số nguyên dương. Đưa ra tổng các chữ số của số đó.

Bài tập 2.13 Nhập vào một số nguyên dương. Cho biết số đó có bao nhiêu chữ số.

Bài tập 2.14 Nhập vào một số nguyên dương n. Nhập liên tiếp không quá n số

tự nhiên. Tính số lần nhập vào số 1, số lần nhập vào số 50, số lần nhập vào số 30. Yêu cầu không được sử dụng mảng.

Bài tập 2.15 Phân tích một số thành thừa số nguyên tố. Và đưa ra kết quả dưới dạng

$(m_1,n_1)(m_2,n_2)...$

trong đó m_i là thừa số nguyên tố, còn n_i là số mũ.

Ví dụ

12 = (2,2)(3,1)

26 = (2,2)(3,2)

Bài tập 2.16 Tổng quát bài tập 2.1 trong đó số dòng có dấu * là n (0<n<25) được nhập từ bàn phím.

Chương 3

CON TRỎ VÀ MẢNG

Mục đích chương này

1) Giới thiệu kiểu dữ liệu mảng.
2) Biết cách sử dụng mảng để lưu, sắp xếp và tìm kiếm danh sách các giá trị của một mảng.
3) Biết cách khai báo một mảng, truy nhập đến các thành phần của mảng.
4) Mảng nhiều chiều.
5) Giới thiệu khái niệm con trỏ.
6) Biết cách sử dụng các con trỏ trong lập trình.
7) Quan hệ tương hỗ giữa giữa con trỏ, mảng và xâu ký tự.
8) Mảng các con trỏ, mảng các xâu.

1. MẢNG

1.1 Khái niệm và định nghĩa

Mảng (array) là một dãy liên tiếp các phần tử trong bộ nhớ, mỗi phần tử chứa một dữ liệu cùng một kiểu. Một mảng số nguyên gồm các phần tử là các biến số nguyên. Một mảng số thực bao gồm các thành phần, mỗi thành phần là một biến thực.

Kích thước của mảng là chính số các phần tử trong mảng. Kích thước này phải được khai báo tường minh trong phần khai báo mảng vì nó xác định vị trí và kích thước của vùng nhớ trong bộ nhớ trong được cấp phát cho mảng.

Mảng có thể có một hay nhiều chiều. Ngôn ngữ C không giới hạn số chiều của mảng.

1.2 Khai báo mảng

1.2.1 Mảng một chiều

Nguyên tắc chung để khai báo một biến mảng như sau.

<type> <array_name>[size];

trong đó:

type là kiểu dữ liệu của các thành phần mảng,

array_name là tên của mảng,

size là số thành phần trong mảng, giá trị này phải là một hằng số.

Ví dụ

```
int   ti [7] ;/*mảng số nguyên 7 thành phần*/
char ch[20]; /*mảng ký tự 20 thành phần*/
```

Kích thước của biến mảng được xác định dựa vào kiểu thành phần và kích thước của biến mảng. Chẳng hạn, mảng `ti` sẽ được cấp phát 7*2=14 bytes trong bộ nhớ. Trong khi đó mảng `ch` bao gồm 20 ký tự, chiếm 20 bytes. Địa chỉ của vùng nhớ cấp pháp cho biến mảng được xác định bởi tên của biến mảng. Nói cách khác, tên biến mảng cho một giá trị mô tả địa chỉ đầu của mảng. Chừng nào mảng còn tồn tại thì tên mảng còn xác định và vùng nhớ cho mảng cũng cố định. Ta gọi những mảng được tạo ra bằng cách khai báo là mảng tĩnh để phân biệt với mảng động được tạo ra và xoá đi nhờ các công cụ cấp phát động.

1.2.2 Truy nhập vào các phần tử của mảng

Mỗi phần tử của mảng được truy nhập thông qua tên biến mảng và chỉ số tương ứng, phần tử đầu tiên có chỉ số là 0. Ví dụ với khai báo mảng `ti` ở trên, ta có:

`ti[0]` là phần tử đầu tiên,

`ti[1]` là phần tử thứ hai,

`ti[2]` là phần tử thứ ba,

`v.v...`

Để làm chỉ số có thể dùng các đại lượng với giá trị là số: hằng, biến, biểu thức. Mỗi phần tử của biến mảng `ti` được xem như một biến nguyên; mỗi phần tử của biến mảng `ch` được xem là một biến ký tự.

Ví dụ

```
ti[1]=5;
ch[10]='A';
```

Một điểm khác biệt quan trọng giữa C và PASCAL là ngôn ngữ C không đưa ra các kiểm tra về sự quá giới hạn chỉ số các thành phần của mảng. Chẳng hạn, với mảng `ti` ở trên, về nguyên tắc chỉ có các thành phần sau `ti[0],...,ti[6]` là hợp lệ. Tuy nhiên chương trình dịch C cũng sẽ không đưa ra bất kỳ cảnh báo nào nếu chúng ta viết `ti[10]` trong một biểu thức nào đó trong chương trình.

1.2.3 Mảng nhiều chiều

Ngôn ngữ C coi rằng một mảng nhiều chiều là mảng một chiều trong đó mỗi phần tử là một mảng với số chiều ít hơn 1. Chẳng hạn, mảng hai chiều các số nguyên thực chất là mảng một chiều có kiểu thành phần là mảng một chiều các số nguyên.

Trong khai báo mảng nhiều chiều ta phải mô tả kích thước của tất cả các chiều; mỗi kích thước như thế được đặt trong một cặp dấu ngoặc vuông "[]". Chẳng hạn, trong khai báo mảng hai chiều sau đây:

```
float tb[10][20];
```

ta có thể xem `tb` như là một mảng một chiều có 10 thành phần và mỗi thành phần là một mảng chứa 20 số thực. Như vậy có thể viết:

`tb[i]` để chỉ một mảng 20 số thực nào đó trong `tb`

hoặc

`tb[i][j]` để xác định một số thực nào đó trong `tb`(không được sử dụng cách viết `tb[i,j]` như trong PASCAL).

1.3 Nhập dữ liệu cho biến mảng

Về nguyên tắc, có hai cách để thực hiện:

(i) Sử dụng các hàm nhập (`scanf()`) để nhập giá trị cho phần tử cần nhập. Chẳng hạn:

```
scanf("%d",&ti[6]);
scanf("%f",&tb[3][4]);
```

Ta gọi đây là cách nhập trực tiếp.

(ii) Sử dụng một biến trung gian có cùng kiểu với các phần tử của mảng. Trước tiên ta nhập giá trị qua biến trung gian, sau đó gán giá trị của biến cho phần tử cần nhập giá trị.

Ví dụ

```
int temp1;
float temp2;
...
scanf("%d",&temp1);
ti[6]=temp1;
scanf("%f",&temp2);
tb[3][4]=temp2;
```

Theo kinh nghiệm, cách thứ hai thường ổn định hơn, không làm treo máy, trong khi đó vì áp dụng phép toán địa chỉ lên các phần tử của mảng cho nên trong một số trường khi kiểu thành phần không phải là nguyên, cách nhập trực tiếp có thể làm treo máy.

Chương trình `array1.c` sau đây mô tả cách sử dụng mảng số nguyên để thực hiện: nhập và in ra một dãy số nguyên. Ở đây ta có thể sử dụng một trong các vòng lặp của ngôn ngữ lập trình C để thực hiện công việc. Trong chương trình ta sử dụng vòng lặp for. Bạn đọc hãy viết lại chương trình bằng các câu lệnh lặp khác.

Bài tập mẫu 3.1

```
/*array1.c*/
/*Chương trình nhập dãy số và in dãy số ra màn hình*/
#include <stdio.h>
#include <conio.h>
void main(){
        int n,i;                    /*n là số phần tử của dãy số*/
        int dayso[100]; /*Dãy số của có tối đa 100 phần tử*/
        clrscr();           /*Xóa màn hình làm việc*/
        /*Nhập số phần tử của dãy số*/
       do {
      printf("\n Nhap vao so phan tu n =");
        scanf("%d",&n);
            }while(n<=0 || n>100);
        /*Nhập vào các thành phần của dãy số*/
        for(i=0;i<n;i++) {
                printf("dayso[%d]=",i);
                scanf("%d",&dayso[i]);
                }
        /*In dãy số vừa nhập ra màn hình*/
        for(i=0;i<n;i++)
                printf("%5d",dayso[i]);
        /*ấn phím bất kỳ để kết thúc chương trình*/
        getch();
        }
```

```
Nhap vao so phan tu n =5
dayso[0]=1
dayso[1]=2
dayso[2]=3
dayso[3]=2
dayso[4]=3
    1    2    3    2    3
```

1.4 Xâu ký tự

1.4.1 Khai báo xâu ký tự

Một biến xâu ký tự được khai báo tương tự như một mảng một chiều chứa các ký tự. Chẳng hạn:

```
char str[10];
```

Sự khác biệt giữa mảng ký tự và xâu ký tự là trong xâu ký tự có ký tự kết thúc xâu (ký tự NUL hay '\0'). Đây là căn cứ của các thao tác xâu như: tính chiều dài, cộng xâu ... Một biến xâu ký tự muốn có chiều dài 9 phải được khai báo như một mảng ký tự có 10 phần tử. Hình 3.1 đưa ra một hình ảnh về nội dung xâu ký tự `str`.

str[0]	'a'
str[1]	'b'
str[2]	'c'
str[3]	'd'
str[4]	'e'
str[5]	'f'
str[6]	'g'
str[7]	'h'
str[8]	'\0'
str[9]	

Hình 3.1 Nội dung một xâu ký tự

Trong hình vẽ trên, các ký tự `str[8]` và `str[9]` không có vai trò gì trong các thao tác trên xâu `str`.

Cần phân biệt giữa ký tự và xâu ký tự chứa một ký tự. Ví dụ 'A' là ký tự A được mã hóa bằng 1 byte, trong khi "A" là một xâu ký tự chứa ký tự A xâu này được mã hóa bằng 2 bytes cho ký tự A và ký tự '\0'.

Trong C không tồn tại các phép toán so sánh, gán nội dung của xâu này cho xâu khác. Để thực hiện các công việc này C cung cấp cho người lập trình một thư

viện các hàm chuẩn, được khai báo trong tệp tiêu đề có tên là `string.h`. Để sử dụng các hàm thao tác xâu, trên đầu chương trình cần phải có dòng khai báo tệp tiêu đề này.

1.4.2 Vào ra với xâu ký tự

Sử dụng `printf()` và `scanf()`

Để nhập, xuất xâu ký tự với `scanf()` và `printf()` ta sử dụng định dạng "`%s`" trong xâu điều khiển, còn trong phần tham số đi sau ta sử dụng tên của biến xâu. Khi nhập xâu ký tự, để đảm bảo việc nhập đúng nên sử dụng một chỉ thị `scanf(...)` riêng cho mỗi lần nhập xâu và trước mỗi chỉ thị đó nên sử dụng chỉ thị sau:

```
fflush(stdin);
```

Chương trình `scanf4.c` sau đây là một phiên bản mới của `scanf3.c` trong đó bổ sung thêm các chỉ thị nhập xâu ký tự.

Bài tập mẫu 3.2

```
/*scanf4.c*/
#include <stdio.h>
#include <conio.h>
void main() {
int n;
long l;
float f;
double d;
char c;
char str[20];
clrscr();
printf("Nhap gia tri cho cac bien\n");
scanf("%d%ld%f%lf",&n,&l,&f,&d);
fflush(stdin);
scanf("%c",&c);
fflush(stdin);
scanf("%s",str);
printf("So int %d\n",n);
printf("So long %d\n",l);
printf("So float %f\n",f);
```

```
printf("So double %f\n",d);
printf("Ky tu %c\n",c);
printf("Xau ky tu %s\n",str);
printf("An phim bat ky de thoat");
getch();
}
```

```
Nhap gia tri cho cac bien
122
33
222
33
2
23233232
So int 122
So long 33
So float 222.000000
So double 33.000000
Ky tu 2
Xau ky tu 23233232
An phim bat ky de thoat
```

Sử dụng `puts()` và `gets()`

Hai hàm này được khai báo trong tệp tiêu đề `stdio.h`. Cú pháp sử dụng như sau:

```
gets(str);
puts(str);
```

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

Khác với `scanf()` hàm `gets()` cho phép nhập cả các ký tự trắng trong nội dung của xâu. Khi lệnh `gets()` đi sau các chỉ thị nhập khác nên sử dụng thêm

```
fflush(stdin);
```

phía trước để đảm bảo việc nhập không bị sai.

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

Chương trình sau đây minh hoạ cách sử dụng `gets()` và `puts()` để nhập và in xâu ký tự.

Bài tập mẫu 3.3

```
/*gets.c*/
#include <stdio.h>
#include <conio.h>
void main() {
        char string[81];
        clrscr();
        printf("\nNhap vao dong moi: ");
                gets(string);
        printf("\nBan da nhap vao xau: ");
                puts(string);
        getch();
        }
```

```
Nhap vao dong moi: aa bb
Ban da nhap vao xau: aa bb
```

1.5 Khai báo và khởi đầu giá trị cho mảng

1.5.1 Mảng một chiều

```
int ti[7]={1,-1,2,3,4,5,6};
```

1.5.2 Mảng nhiều chiều

```
int t2i [3,4] = {{2,3,4,5},
                {3,4,-2,4},
                {3,2,5,6} };
```

1.5.3 Xâu ký tự

Cách 1

```
char str[6]="abc";
```

Cách 2

```
char str[6]={'a','b','c'};
```

Trong danh sách các giá trị có thể không cần mô tả đủ. Khi đó, chỉ những phần tử đầu tiên được gán giá trị, các phần tử còn lại sẽ lấy giá trị ngẫu nhiên. Cũng có thể không cần mô tả kích thước của biến mảng. Khi đó chương trình dịch tự động xác định dựa vào danh sách giá trị mô tả kèm theo. Chẳng hạn:

```
int t[] = {1,2,3};
```

tương ứng khai báo một mảng kích thước 3.

2. CON TRỎ (POINTER)

2.1 Khái niệm và cách khai báo con trỏ

Một con trỏ là một biến/hằng có giá trị (nội dung) là địa chỉ của một đối tượng khác; đối tượng ở đây có thể là một biến hoặc một hàm mà không thể là một hằng.

Việc sử dụng các biến con trỏ sẽ cho phép ta tham chiếu đến một biến thông qua địa chỉ của nó. Khả năng đó cộng với khả năng tính toán linh hoạt trên các con trỏ của C đã khiến con trỏ trở thành một công cụ rất mạnh để thực hiện nhiều thao tác mà thiếu nó thì không thể làm được hoặc làm rất khó khăn.

Khai báo một biến con trỏ theo mẫu sau:

```
type *ptr_name;
```

trong đó :

`type` là kiểu dữ liệu của biến mà con trỏ chứa địa chỉ.

`ptr_name` là tên của biến trỏ.

Nói cách khác `type*` chính là một kiểu dữ liệu con trỏ. Với câu lệnh khai báo trên, biến trỏ chưa chỉ đến địa chỉ nào cả và nó có giá trị bằng NULL.

NULL là một hằng số biểu thị giá trị của con trỏ khi chưa được gán địa chỉ nào.

Giống như khai báo biến, có thể khai báo nhiều biến trỏ liên tiếp nhau.

Ví dụ

```
int *pi, *qi;
float *pf,qf;
char *pc,*qc;
```

Kích thước của biến con trỏ phụ thuộc vào cách biểu diễn địa chỉ bộ nhớ, trong máy PC thường sử dụng giá trị hai thanh ghi 16 bits để biểu thị địa chỉ bộ nhớ, do đó kích thước biến con trỏ là 4 bytes.

Ngôn ngữ C (và cũng như các ngôn ngữ khác) qui định tương ứng với mỗi kiểu dữ liệu sẽ có một kiểu biến con trỏ riêng biệt. Như vậy chúng ta sẽ có con trỏ nguyên int * để chứa địa chỉ các biến nguyên, con trỏ thực float*/double* chứa địa chỉ các biến thực, con trỏ ký tự char * ... Giống như biến nguyên và biến thực, giữa các biến con trỏ khác kiểu nhau cũng có một số điểm khác biệt, chẳng hạn, không thể gán giá trị một con trỏ nguyên cho con trỏ thực mà không thực hiện phép chuyển kiểu.

2.2 Toán tử & và *

Xét các khai báo biến sau:

```
int m,n,p;
float x,y,z;
char ch1,ch2;
```

Toán tử một ngôi & cho ta địa chỉ của một biến, còn biến con trỏ có giá trị là địa chỉ của một biến. Như vậy chỉ thị:

```
pi=&n;
```

là hợp lệ và nó gán địa chỉ của biến nguyên `n` cho con trỏ nguyên `pi`. Khi đó ta nói rằng `pi` "chỉ đến" `n`.

Khi `pi` "chỉ đến" `n`, ta sẽ có hai cách để mô tả giá trị của `n`, hoặc trực tiếp bằng tên của biến như truyền thống hoặc bằng cách áp dụng toán tử một ngôi "*" lên biến trỏ `pi`. Nói cách khác: `*pi` tương đương với `n`. Như vậy:

```
*pi=3;
```

có tác dụng gán giá trị 3 cho biến `n`.

```
*pi = *pi+10;
```

thì thêm `10` vào `n`.

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

```
m = *pi +10;
```

lấy giá trị của biến nguyên `n` được trỏ bởi `pi`, cộng thêm 10 và ghi kết quả trong `m`, còn

```
*pi += 1;
```

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

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

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

2.3 Các phép toán trên con trỏ

2.3.1 Phép gán trên hai con trỏ cùng kiểu

Chỉ có thể gán địa chỉ một biến nguyên cho một biến trỏ nguyên. Cũng vậy không thể gán giá trị của một con trỏ nguyên cho con trỏ thực. Chỉ thị :

```
qi=pi;
```

sẽ cho phép cả pi và qi xác định cùng một biến (biến n). Như vậy *pi, *qi, và n là một.

Các chỉ thị sau đây là không hợp lệ:

```
pf=&m;
pf=pi;
```

Tuy nhiên các chỉ thị sau thì hợp lệ nhờ phép chuyển đổi kiểu được thực hiện trước lệnh gán.

```
pf=(float *) &m;
pf=(float *) pi;
```

Chương trình sau đây minh họa cách dùng toán tử chuyển kiểu bắt buộc để tính byte cao và byte thấp của một số nguyên.

Bài tập mẫu 3.4

```
/*pointer1.c*/
#include <stdio.h>
#include <conio.h>
void main()    {
  int n;
  char *pc;
  clrscr();
  printf("Nhap vao so nguyen : ");
```

```
  scanf("%d",&n);
  pc=(char *)&n; /*pc trỏ đến byte cao của n, pc+1 trỏ đến byte thấp*/
  printf("Byte cao %x ,byte thap %x",*pc ,*pc++);
  getch();
}
```

```
Nhap vao so nguyen : 32762
Byte cao 7f
byte thap fffa
```

2.3.2 Phép cộng con trỏ với số nguyên

Trong phần 2.2 chúng ta biết rằng m,n,p là ba biến nguyên khai báo liên tiếp nhau. Khi đó m, n, p sẽ tương ứng với ba vùng nhớ 2 bytes liên tiếp nhau. Chúng ta cũng biết rằng pi chỉ đến n.

Phép cộng con trỏ pi+1 sẽ cho ta địa chỉ của số nguyên m đi ngay trước n, còn pi+(-1) cho ta địa chỉ của số nguyên p đi liền sau n. Điều này được minh hoạ trong chương trình sau đây:

Bài tập mẫu 3.5

```
/*pointer2.c*/
#include <stdio.h>
#include <conio.h>
void main() {
  int m=2,n=100,p=20;
  int *pi=&n;
  clrscr();
  printf("m = %d n = %d p = %d\n",m,n,p);
  printf("&n = %p &m = %p &p = %p\n",&n,&m,&p);
  printf("pi = %p pi+1 = %p pi-1 = %p\n",pi,pi+1,pi-1);
  printf("*pi = %d ",*pi);
  printf("*(pi+1) = %d ", *(pi+1));
  printf("*(pi-1) = %d ",*(pi-1));
  getch();
}
```

```
m = 2 n = 100 p = 20
```

```
&n = 8FF1:0FFC &m = 8FF1:0FFE &p = 8FF1:0FFA
pi = 8FF1:0FFC pi+1 = 8FF1:0FFE pi-1 = 8FF1:0FFA
*pi = 100 *(pi+1) = 2 *(pi-1) = 20
```

Việc cộng trừ một con trỏ với một số nguyên có ý nghĩa trực quan như sau: nếu ta so sánh con trỏ với địa chỉ các ngôi nhà trên một con đường, mỗi ngôi nhà là một biến có địa chỉ nào đó thì các địa chỉ này được đánh số tăng dần theo một chiều nhất định. Biết địa chỉ của nhà này, ta có thể tính địa chỉ của nhà bên cạnh bằng cách cộng thêm hoặc trừ đi 1 vào địa chỉ của căn nhà đã biết và cứ thế tiếp tục. Điều đó có nghĩa là khi cộng hoặc trừ con trỏ với một số nguyên n, ta sẽ được một địa chỉ mới chỉ đến một biến khác nằm cách biến trước đó n vị trí.

2.3.3 Phép trừ hai con trỏ cùng kiểu

Do phép cộng một con trỏ với một số nguyên cho ta một con trỏ cùng kiểu, nên phép trừ hai con trỏ cùng kiểu được coi là hợp lệ và cho kết quả là một số nguyên biểu thị "khoảng cách" giữa hai biến con trỏ. Chẳng hạn, nếu

```
pi = &n;
qi = &m;
```

thì hiệu

```
qi-pi
```

có giá trị bằng 1.

Tuy vậy không có phép cộng, nhân, chia giữa hai con trỏ (vì chúng không có ý nghĩa thực tế).

2.3.4 Các phép toán quan hệ trên các con trỏ cùng kiểu

Có thể so sánh các con trỏ cùng kiểu, tiêu chuẩn so sánh dựa trên vị trí ô nhớ tương ứng với hai con trỏ. Hằng con trỏ NULL luôn khác với mọi địa chỉ bộ nhớ và có thể đem so sánh với mọi biến trỏ với kiểu khác nhau. Với hai con trỏ khác kiểu nhau, nếu muốn so sánh với nhau phải thực hiện phép chuyển đổi kiểu.

2.4 Con trỏ void *

2.4.1 Kiểu dữ liệu void

Trong ngôn ngữ C, kiểu dữ liệu **void** là kiểu dữ liệu rỗng (empty type). Một biến kiểu **void** không tương ứng với bất kỳ vùng nhớ nào.

2.4.2 Con trỏ kiểu void*

Con trỏ **void*** có thể nhận địa chỉ của bất kỳ vùng nhớ nào dù đó là vùng nhớ tương ứng với một biến nguyên, biến thực ... Các chỉ thị sau đây là hợp lệ:

```
void *px,*py;
int x=1;
float y=0.1;
px=&x;
py=&y;
```

Không thể thực hiện các phép tính số học trên con trỏ **void***. Điều này có thể giải thích được nhờ bản chất của các phép toán.

2.5 Con trỏ đa cấp

Bản thân biến trỏ cũng có địa chỉ, do đó có thể chứa địa chỉ của nó trong một đối tượng khác. Người ta gọi những đối tượng có giá trị là địa chỉ của một biến con trỏ là con trỏ đa cấp. Trong khai báo biến trỏ đa cấp, người ta sử dụng một số dấu "*" trước tên biến trỏ. Số lượng dấu "*" xác định cấp của con trỏ.

Ví dụ

```
int *pi;
int **ppi =&pi;
```

3. LIÊN HỆ GIỮA CON TRỎ VÀ MẢNG

3.1 Con trỏ và mảng một chiều

Xét chỉ thị:

```
int a[10];
```

Khai báo này sẽ xin cấp phát một vùng nhớ 10 số nguyên `a[0],a[1],...` có địa chỉ xác định bởi `a`. Vậy `a` chính là địa chỉ của biến nguyên `a[0]`, do vậy nếu `pi` là một con trỏ nguyên thì phép gán sau đây là hợp lệ.

```
pi =a;
```

Điều thú vị là sau phép gán này chúng ta có thể sử dụng `pi` như một tên mảng, còn tất nhiên có thể thực hiện các phép toán số học trên con trỏ `a`. Quả thực

`pi[0],...,pi[9]`

tương ứng là

`a[0],...,a[9]`

còn

a,a+1,...,a+9

tương ứng là địa chỉ của

a[0],a[1],...,a[9].

Nói cách khác, có một liên hệ mật thiết giữa việc đánh chỉ số trên mảng và các tính toán trên con trỏ.

Tuy vậy vẫn có một số điểm khác nhau giữa tên mảng và biến trỏ cùng kiểu. Một bên là hằng số còn một bên là biến. Không thể thực hiện các phép toán tự tăng giá trị lên tên biến mảng. Cũng không thể thực hiện được phép gán trong đó tên biến mảng đứng một mình bên trái toán tử gán. Chẳng hạn:

```
int b[10];
```

chỉ thị

```
b=a;
```

là sai vì không thể gán giá trị cho một hằng. Rất cần nhấn mạnh điều này vì rất nhiều người vẫn chưa từ bỏ được một số thói quen trong PASCAL:

```
char str[20];
str="abc";
```

3.2 Con trỏ đa cấp và mảng nhiều chiều

Con trỏ một cấp và mảng một chiều có nhiều điểm tương đồng. Mảng hai chiều là mảng một chiều của các mảng một chiều. Con trỏ hai cấp là con trỏ chỉ đến con trỏ một cấp. Vì vậy có thể suy ra rằng giữa con trỏ hai chiều và mảng hai chiều cũng có nhiều điểm tương đồng. Suy luận này cũng có thể mở rộng cho các mảng và con trỏ nhiều mức hơn. Chẳng hạn với các câu lệnh khai báo:

```
int **ppi;
int a2[10][10];
```

ta có thể thực hiện phép gán:

```
ppi=a2;
```

và cũng có thể sử dụng `ppi` như tên một biến mảng.

3.3 Mảng các con trỏ

Khi kiểu phần tử của một biến mảng là kiểu con trỏ thì ta sẽ có một mảng các

con trỏ. Nếu các con trỏ thành phần trong mảng chỉ đến các biến nào đó, thì như vậy, chúng ta có thể truy xuất được các biến này thông qua một mảng mà không cần đến vị trí thực sự của các biến đó có liên tiếp hay không. Điều đó khiến cho chúng ta dường như đã có được một mảng đặc biệt gồm các biến mà vị trí thực sự của chúng trong bộ nhớ là bất kỳ. Và bằng việc đổi chỗ các con trỏ thành phần này chúng ta có thể đổi thứ tự sắp xếp của các biến trong mảng này mà không cần thay đổi thực sự vị trí của chúng. Chương trình sau đây đưa ra một cách sắp xếp các số mà không tác động lên bản thân các phần tử được sắp xếp.

Bài tập mẫu 3.6

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

```
    printf(" %d \n",*ptr_array[i]);
  getch();
  }
```

```
2
3
6
7
10
12
```

Nếu các phần tử của một các con trỏ thành phần lại được gán địa chỉ của các mảng khác thì ta sẽ được một mảng của các mảng. Không giống như các mảng hai chiều, các mảng con của chúng ta có thể nằm ở vị trí bất kỳ. Vì ta chỉ lưu trữ địa chỉ của chúng nên việc sắp xếp lại thứ tự các của các mảng này so với nhau thực chất chỉ là việc sắp xếp lại các địa chỉ của chúng trong mảng các con trỏ của chúng ta mà thôi. Chương trình ví dụ sau đây in ra các hàng của ma trận theo thứ tự tăng dần của tổng giá trị trên từng hàng.

Bài tập mẫu 3.7

```
#include <stdio.h>
#include <conio.h>
void main() {
  int a[10][10];
  int *pa[10];
  int s[10];
  int *ps[10];
  int i,j,m,n;
  int temp;
  clrscr();
  printf("Nhap hai kich thuoc mang ");
  scanf("%d%d",&m,&n);
  for(i=0;i<m;i++){
    pa[i]=a[i];
    ps[i]=s+i;
    }
```

```
for(i=0;i<m;i++)
  for(j=0;j<=n;j++) {
    printf("a[%d,%d]=",i,j);
    scanf("%d",&temp);
    a[i][j]=temp;
    }
for(i=0;i<m;i++) {
  s[i]=0;
  for(j=0;j<n;j++)
    s[i]+=a[i][j];
  }
for(i=0;i<m-1;i++)
  for(j=i+1;j<m;j++)
    if(*ps[i]>*ps[j]) {
      int *t;
      t=ps[i];
      ps[i]=ps[j];
      ps[j]=t;
      t=pa[i];
      pa[i]=pa[j];
      pa[j]=t;
      }
printf("Mang a\n");
for(i=0;i<m;i++) {
  for(j=0;j<n;j++)
    printf(" 4d ",a[i][j]);
  printf("\n");
  }
printf("Cac dong theo pa\n");
for(i=0;i<m;i++) {
  for(j=0;j<n;j++)
    printf("%4d ",pa[i][j]);
  printf("\n");
```

```
  }
 getch();
}
```

```
Nhap hai kich thuoc mang 4 2
a[0,0]=3
a[0,1]=4
a[0,2]=2
a[1,0]=3
a[1,1]=4
a[1,2]=3
a[2,0]=4
a[2,1]=43
a[2,2]=3
a[3,0]=2
a[3,1]=3
a[3,2]=1
Mang a
   3    4
   3    4
   4   43
   2    3
Cac dong theo pa
   2    3
   3    4
   3    4
   4   43
```

4. CẤP PHÁT ĐỘNG

Ý nghĩa của việc cấp phát bộ nhớ động là cho phép chương trình sử dụng vừa đúng khối lượng bộ nhớ mà chương trình cần, và khi không cần dùng tới nữa thì có thể giải phóng để cho các công việc tiếp theo có thể sử dụng được. Như vậy, cùng một vùng bộ nhớ có thể được sử dụng cho các mục đích khác nhau trong thời gian thực hiện của chương trình.

Vùng nhớ heap được sử dụng cho mục đích cấp phát động các khối nhớ có kích thước thay đổi. Có nhiều cấu trúc dữ liệu sử dụng cách cấp phát động, ta có thể liệt kê một vài loại như: cây (tree), các loại danh sách liên kết.

Ngôn ngữ C cung cấp thư viện các hàm cho phép "thuê" và trả một vùng bộ nhớ trước đó đã được "thuê". Sau đây dẫn ra một số hàm tiêu biểu.

4.1 Hàm malloc()

Các tệp tiêu đề liên quan

ALLOC.H

STDLIB.H

Cú pháp thực hiện

```
malloc(size);
```

Hàm `malloc()` dùng để xin cấp phát một vùng bộ nhớ có kích thước `size` bytes từ vùng nhớ heap. Hàm này cho phép một chương trình xin cấp phát một vùng bộ nhớ đúng với kích thước mà chương trình cần.

Giá trị trả về

Trong trường hợp cấp phát thành công, hàm `malloc()` trả về một con trỏ tới khối nhớ mới được cung cấp.

Trong trường hợp có lỗi (không đủ bộ nhớ để cấp phát, `malloc()` trả về giá trị `NULL`.

Nếu tham số `size` bằng 0, `malloc()` trả về con trỏ `NULL`.

Ví dụ

```
int *pi,*qi;
char *pc;
double *pd;
double **ppd;
int i;
/*cấp phát vùng nhớ một số nguyên*/
qi=(int *) malloc(sizeof(int));
```

```
/*cấp phát một mảng các số nguyên-mảng động */
pi=(int *) malloc(12*sizeof(int));
/*cấp phát xâu ký tự*/
pc=(char *) malloc(20);
pd=(double *) malloc(10*sizeof(double));
/*cấp phát ma trận*/
ppd=(double **)malloc(10*sizeof(double *));
for(i=0;i<10;i++)
 ppd[i]=(double*)malloc(10*sizeof(double));
```

Chúng ta phải sử dụng phép chuyển kiểu đối với giá trị trả về của malloc để đảm bảo các yêu cầu của phép gán.

4.2 Hàm calloc ()

Các tập tin tiêu đề liên quan

ALLOC.H

STDLIB.H

Cú pháp sử dụng

`calloc(nitems, size);`

Hàm `calloc()` xin cấp phát một vùng nhớ kích thước (`nitems * size`) bytes và xóa trắng vùng nhớ này. Muốn xin cấp phát vùng nhớ có kích thước lớn hơn 64K, phải sử dụng hàm `farcalloc` (cấp phát xa).

Giá trị trả về

Trong trường hợp thành công, calloc trả về con trỏ tới vùng nhớ mới được cấp phát.

Khi có lỗi cấp phát (không đủ bộ nhớ hoặc một trong hai tham số nitems hoặc size bằng 0) thì hàm trả về NULL.

Ví dụ

```
int *pi;
double *pd;
double **ppd;
int i;
/*cấp phát một mảng các số nguyên-mảng động */
pi=(int *) malloc(12,sizeof(int));
```

```
pd=(double *) malloc(10,sizeof(double));
/*cấp phát ma trận*/
ppd=(double **)malloc(10,sizeof(double *));
for(i=0;i<10;i++)
 ppd[i]=(double*)malloc(10,sizeof(double));
```

4.3 Hàm free()

Các tập tin tiêu đề liên quan

ALLOC.H

STDLIB.H

Cú pháp sử dụng

`free(address);`

Hàm `free()` giải phóng một khối nhớ đã được cấp phát trước đó bằng hàm `calloc()`, `malloc()`, hoặc `realloc()`. Địa chỉ vùng nhớ được giải phóng được truyền làm tham số của hàm `free()`.

Ví dụ

```
free(qi);
free(pi)
free(pc);
free(pd);
for(i=0;i<10;i++)
  free(ppd[i]);
free(ppd);
```

4.4 Ví dụ

Chương trình sau minh hoạ cách sử dụng các hàm cấp phát động để tạo mảng động. Bạn đọc có thể xem thêm các ví dụ trong phần trợ giúp của các chương trình dịch C.

Bài tập mẫu 3.8

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

```
void main() {
  double **ppd;
  int m,n;
  int i,j;
  double temp;
  clrscr();
  printf("Kich thuoc ma tran");
  scanf("%d%d",&m,&n);
  ppd = (double **) calloc(m,sizeof(double *));
  for(i=0;i<m;i++)
    ppd[i]=(double *) calloc(n,sizeof(double));
  for(i=0;i<m;i++)
    for(j=0;j<n;j++) {
      printf("ppd[%d,%d]=",i,j);
      scanf("%lf",&temp);
      ppd[i][j]=temp;
    }
  for(i=0;i<m;i++) {
    for(j=0;j<n;j++)
      printf("%6.2f ",ppd[i][j]);
    printf("\n");
    }
  for(i=0;i<m;i++)
    free(ppd[i]);
  free(ppd);
  getch();
  }
```

```
Kich thuoc ma tran4 2
ppd[0,0]=3
ppd[0,1]=4
ppd[1,0]=3
ppd[1,1]=3
ppd[2,0]=2
```

```
ppd[2,1]=3
ppd[3,0]=4
ppd[3,1]=3
  3.00    4.00
  3.00    3.00
  2.00    3.00
  4.00    3.00
```

5. TÓM TẮT

5.1 Ghi nhớ

Có thể lưu một danh sách các giá trị trong mảng. Một mảng là một nhóm các phần tử có cùng kiểu giá trị được lưu trữ kế tiếp nhau trong bộ nhớ. Truy nhập đến các thành phần mảng bằng tên biến mảng và chỉ số. Chỉ số các phần tử tính từ 0.

Chỉ số của phần tử mảng có thể là giá trị của một biến nguyên, một hằng nguyên hay một biểu thức cho giá trị nguyên.

Một mảng ký tự có thể được sử dụng để lưu trữ xâu ký tự. Xâu ký tự là một mảng ký tự đặc biệt có chứa ký tự '`\0`' ở cuối xâu.

Các thành phần của mảng có thể được khởi tạo theo ba cách. Trong khi khai báo, bằng các câu lệnh gán, bằng các câu lệnh nhập dữ liệu.

Ngôn ngữ C không tự động kiểm tra sự quá giới hạn kích thước của mảng.

Một xâu có thể được nhập bằng `scanf()` với mô tả `%s` hoặc bằng `gets()` và có thể được hiển thị bởi hàm `printf()` với định dạng "`%s`" hoặc `puts()`.

Con trỏ là biến chứa địa chỉ của các biến khác.

Các con trỏ cần phải được khai báo trước khi sử dụng. Tương ứng với các kiểu dữ liệu khác nhau, có các loại biến con trỏ khác nhau.

Có ba giá trị có thể sử dụng để khởi tạo cho một biến con trỏ: `0`, `NULL` hoặc một địa chỉ. Khởi tạo một con trỏ bằng `0` đồng nghĩa với khởi tạo `NULL` cho con trỏ.

Chỉ có một số nguyên có thể gán được cho con trỏ đó là `0`.

Toán tử & trả về địa chỉ của toán hạng của nó. Toán hạng ở đây là một biến nào đó và không phải là biến kiểu thanh ghi (register).

Toán tử * tham chiếu nội dung của một của đối tượng mà toán hạng của * trỏ tới trong bộ nhớ.

Các thao tác số học có thể thực hiện trên contrỏ là `++` , `--` (trên các con trỏ) , `+`, `-` `+=`, `-=` (giữa con trỏ và một số nguyên)

Khi một số nguyên được thêm vào hay trừ vào một con trỏ, con trỏ được tăng hoặc giảm số nguyên đó lần kích thước của đối tượng được trỏ tới.

Các thao tác số học trên con trỏ chỉ được thực hiện trên một phần bộ nhớ liên tục chẳng hạn như một mảng.

Khi thực hiện các thao tác số học con trỏ trên mảng ký tự kết quả thu được giống như đối với các thao tác số học thông thường bởi vì mỗi một ký tự được lưu trữ một byte trong bộ nhớ.

Có thể gán giá trị của một con trỏ cho một con trỏ khác cùng kiểu.

Một con trỏ void không áp dụng được phép toán *.

Các con trỏ có thể được so sánh với nhau theo các toán tử so sánh. Ý nghĩa của các phép so sánh giống như các phép so sánh thông thường.

Một con trỏ có thể được đánh chỉ số như mảng.

Tên mảng là một hằng con trỏ chỉ đến phần tử đầu tiên trong mảng (phần tử thứ tự 0).

Độ lệch giữa hai con trỏ cũng giống như khái niệm chỉ số đối với mảng.

Có thể khai báo một mảng các con trỏ.

5.2 Các lỗi hay gặp khi lập trình

Truy nhập tới một phần tử nằm ngoài giới hạn của mảng.

Truy nhập tới các phần tử của mảng nhiều chiều sai quy cách, chẳng hạn

`a[2,4]` thay vì `a[2][4]`

Nhầm lẫn giữa "phần tử thứ bảy của mảng" và "phần tử mảng có số thứ tự 7".

Quên toán tử * trong khai báo các biến con trỏ. Mỗi biến con trỏ cần phải được khai báo với một * như phần tiền tố.

Tham chiếu tới một con trỏ chưa được khởi tạo giá trị. Điều này có thể gây ra lỗi thực hiện chương trình.

Sử dụng các thao tác số học con trỏ trên một con trỏ không có liên hệ với một mảng cụ thể nào đó.

Dùng toán tử * để lấy nội dung của một con trỏ void.

Sử dụng tên mảng trong các phép toán nhằm thay đổi giá trị của tên đó. Sai vì tên mảng là một hằng con trỏ.

5.3 Các thói quen lập trình tốt cần học tập

Khi duyệt qua các thành phần của một mảng, chỉ số không được bé hơn 0 và phải bé hơn kích thước của mảng.

Thêm ptr ở phần đầu tên của các con trỏ để phân biệt với các biến thông thường khác.

Sử dụng ký pháp mảng khi thao tác trên mảng thay vì ký pháp con trỏ.

6. BÀI TẬP

Bài tập 3.1 Lập chương trình mô phỏng các thao tác trên dãy số:

- Nhập vào một dãy số thực.

- In dãy số thực ra màn hình.

- Sắp xếp dãy số thực tăng dần.

- Nhập vào một số thực và kiểm tra xem số đó đã có mặt trong dãy số chưa, nếu chưa có thì chèn số vào vị trí thích hợp.

- Nhập vào một số nguyên và xoá phần tử có số thứ tự tương ứng.

Bài tập 3.2 Lập chương trình thực hiện các công việc sau:

- Nhập vào một dãy số thực.

- In ra các số dương có mặt trong dãy số.

- Tìm giá trị lớn nhất và nhỏ nhất trong số các giá trị có mặt trong dãy số.

Bài tập 3.3 Lập chương trình mô phỏng phép cộng và nhân vô hướng giữa hai vector.

Bài tập 3.4 Lập chương trình mô phỏng các thao tác trên ma trận vuông:

- Tính định thức của một ma trận vuông.

- Tính ma trận nghịch đảo của ma trận có định thức khác 0.

- Giải hệ phương trình tuyến tính.

Bài tập 3.5 Lập chương trình mô phỏng:

- Phép cộng giữa hai ma trận cùng kích thước

- Xác định ma trận chuyển vị

- Nhân vector với ma trận.

Bài tập 3.6 Viết chương trình kiểm tra ý nghĩa của phép toán & và *.

Bài tập 3.7 Thực hiện các thao tác nhập dữ liệu cho mảng một chiều, hai chiều bằng con trỏ.

Bài tập 3.8 Nhập vào danh sách chứa tên các đội bóng của giải vô địch quốc gia Việt nam. Kết quả thi đấu của các đội trong một mùa bóng được cho trong bảng vuông. Số ở hàng i cột j là điểm mà đội thứ i thu được trong hai trận đấu với đội thứ j (thứ tự tính theo danh sách đội bóng). Nhận xét rằng hai số ở hàng i cột j và hàng j cột i có ít nhiều liên quan. Hãy nhập vào bảng kết quả và in ra tên các đội bóng theo thứ tự tổng điểm giảm dần. Yêu cầu viết bằng hai cách:

(i) không sử dụng mảng con trỏ

(ii) có sử dụng mảng các con trỏ.

Chương 4

HÀM

Mục đích chương này

1) Biết cách xây dựng các chương trình từ các khối nhỏ hơn, gọi là các hàm.
2) Giới thiệu các hàm toán học trong thư viện C chuẩn.
3) Cách tạo ra các hàm mới.
4) Hiểu cơ cấu truyền thông tin giữa các hàm.
5) Hiểu cách dùng và cách viết các hàm đệ qui.

1. GIỚI THIỆU

1.1 Tổng quát về chương trình con

Hầu hết chương trình máy tính giải quyết các vấn đề trong thực tế đều lớn hơn rất nhiều so với các ví dụ được giới thiệu trong cuốn sách này. Kinh nghiệm cho thấy rằng cách tốt nhất để phát triển và bảo trì các chương trình lớn là xây dựng chúng từ những mảnh nhỏ hơn hay các mô đun. Khi đó việc quản lý sẽ dễ hơn đối với chương trình ban đầu.

Các chương trình con cho phép lập trình viên mô đun hoá chương trình. Nói chung, các biến định nghĩa trong chương trình con là các biến cục bộ, chúng chỉ được biết trong bản thân chương trình con tại đó chúng được định nghĩa. Hầu hết các chương trình con đều có các tham số. Các tham số cung cấp thông tin truyền giữa các chương trình con. Các tham số của chương trình con nói chung cũng được xem như các biến cục bộ.

Mô đun hoá chương trình có nhiều ưu điểm. Trước tiên, *"Chia để trị"* giúp cho việc phát triển các ứng dụng dễ dàng hơn. Một ưu điểm nữa là tính *sử dụng lại*, nghĩa là khả năng dùng các chương trình con có sẵn như các khối cơ bản để xây dựng lên các chương trình mới. Ưu điểm thứ ba là tránh được việc lặp đi lặp lại các đoạn mã trong chương trình. Việc gói các đoạn mã vào trong chương trình con cho phép cả đoạn mã đó thực hiện ở nhiều nơi trong chương trình chỉ với một lời gọi hàm đơn giản.

Chương này mô tả các đặc tính của ngôn ngữ lập trình C trong quá trình thiết kế, cài đặt, hoạt động và bảo trì các chương trình lớn.

1.2 Các mô đun chương trình trong C

Các mô đun trong C được gọi là hàm. Các chương trình C nói chung được viết

bằng cách kết hợp các hàm mà lập trình viên tự viết với các hàm chuẩn trong thư viện của C. Chúng ta sẽ bàn đến cả hai loại hàm trên trong chương này. Các hàm chuẩn trong thư viện của C cung cấp rất nhiều các thao tác tính toán, thao tác trên xâu ký tự, thao tác ký tự, vào/ra và nhiều thao tác thông dụng khác. Chúng làm cho công việc của các lập trình viên dễ dàng hơn vì các hàm này đưa ra rất nhiều khả năng cho các lập trình viên. Mặc dù các hàm trong thư viện chuẩn không phải là một phần trong ngôn ngữ lập trình C, theo chuẩn ANSI C chúng hàm này không thay đổi trong các chương trình dịch C khác nhau. Các hàm `printf()` hay `scanf()` ... mà chúng ta dùng trong các chương trước đây chính là các hàm thư viện chuẩn.

Lập trình viên có thể viết các hàm thao tác các công việc xác định dùng tại nhiều nơi trong chương trình. Các hàm loại này đôi khi còn được gọi là *hàm người dùng định nghĩa*. Các câu lệnh cài đặt cụ thể hàm này chỉ phải viết một lần, chúng được ẩn bên trong và có thể dùng lại nhiều lần.

Các hàm được gọi thực hiện bằng các *lời gọi hàm*, các lời gọi hàm xác định tên của hàm và cung cấp các thông tin (hay còn gọi là các tham số) mà hàm được gọi theo đúng trình tự khi khai báo hàm đó.

1.3 Giới thiệu thư viện các hàm toán học

Thư viện các hàm toán học (khai báo trong tệp tiêu đề `math.h`) cho phép lập trình viên thực hiện các thao tác toán học cơ bản. Một vài hàm trong thư viện các hàm toán học của C được tổng kết trong bảng 4.1. Chúng ta sẽ sử dụng thư viện các hàm toán học ở đây để giới thiệu các khái niệm về hàm. Trong các phần sau, các hàm khác sẽ được đề cập tới.

Bình thường các hàm được gọi trong chương trình bằng cách viết tên hàm, có tham số hoặc danh sách tham số cách nhau bởi dấu phẩy "," được đặt trong cặp dấu ngoặc đơn. Ví dụ để tính và in ra căn bậc hai của 900.0 ta viết:

```
printf("%.2f", sqrt(900.0));
```

Khi câu lệnh trên được thực hiện, hàm khai căn bậc 2 `sqrt()` được gọi để tính căn bậc hai của số 900.0 trong dấu ngoặc. Số 900.0 là tham số của hàm `sqrt()`. Tiếp sau câu lệnh sẽ in ra 30.00. Hàm `sqrt()` nhận các tham số kiểu **double** và trả lại giá trị là số kiểu **double**.

Các tham số của hàm có thể là các hằng, biến hay các biểu thức, ví dụ nếu `c1=13.0, d=3.0` thì câu lệnh

```
printf("%.2f", sqrt(c1 + d * 4.0));
```

sẽ tính và in ra căn bậc 2 của `13.0 + 3.0 * 4.0 = 25.0` là `5.00`

Hàm	Mô tả	Ví dụ
sqrt(x)	Căn bậc 2 của x	sqrt(9.00) bằng 3.0
exp(x)	hàm mũ e^x	exp(1.0) bằng 2.718282
log(x)	logarithm tự nhiên (cơ số e) của x	log(2.718282) bằng 1.0
log10(x)	logarithm thập phân (cơ số 10) của x	log10(1.0) bằng 0.0 log10(10.0) bằng 1.0
fabs(x)	giá trị tuyệt đối của x	nếu x≥0 thì fabs(x) bằng x nếu x<0 thì fabs(x) bằng -x
ceil(x)	làm tròn số nguyên nhỏ nhất không lớn hơn x	ceil(9.2) bằng 10.0 ceil(-9.8) bằng -9.0
floor(x)	làm tròn số nguyên lớn nhất không nhỏ hơn x	floor(9.2) bằng 9.0 floor(-9.8) bằng -10.0
pow(x,y)	x mũ y (x^y)	pow(2,7) bằng 128
fmod(x,y)	phần dư của phép chia x cho y	fmod(13.657,2.333) bằng 1.992
sin(x)	sin của x (x theo radian)	sin(0.0) bằng 0.0
cos(x)	cos của x (x theo radian)	cos(0.0) bằng 1.0
tan(x)	tan của x (x theo radian)	tan(0.0) bằng 0.0

Bảng 4.1 Các hàm thường dùng cho thư viện chuẩn các hàm toán học

2. KHAI BÁO VÀ ĐỊNH NGHĨA HÀM

2.1 Một ví dụ

Mỗi chương trình đều có một hàm chính `main()`. Hàm này gọi các hàm khác để thực hiện công việc của chương trình.

Ta sẽ xem xét cách viết một hàm. Chương trình ví dụ `func1.c` dùng hàm `square()` để tính bình phương các số nguyên từ 1 đến 10.

Bài tập mẫu 4.1

```
/*func1.c*/
#include <stdio.h>
int square(int);/* Hàm nguyên mẫu */
void main() {
```

```
  int i;
  for (i = 1; i <= 10; i++)
    printf( "%d    ", square(i));
    printf("\n");
}
/* Định nghĩa hàm square */
int square(int x) {
  int y;
  y = x*x;
  return y;
  }
```

```
1    4    9    16    25    36    49    64    81    100
```

Hàm `square()` được gọi thực hiện trong hàm `main()` trong câu lệnh:

```
      printf( "%d    ", square(x));
```

Nó nhận một bản sao giá trị của `i` trong tham số hình thức `x`, sau đó nó tính `x*x` rồi kết quả được trả lại cho hàm `printf()` trong hàm `main()`, nơi mà hàm `square()` được gọi, sau cùng `printf()` hiển thị kết quả ra màn hình. Quá trình này lặp đi lặp lại 10 lần trong vòng lặp **`for`**.

Định nghĩa hàm `square()` cho thấy hàm này nhận một tham số nguyên `x`. Từ khoá **`int`** đặt trước tên hàm cho thấy hàm `square()` trả lại kết quả là số nguyên. Câu lệnh **`return`** trong hàm `square()` trả kết quả tính toán được lại cho hàm gọi nó.

Dòng

```
      int square(int);
```

chính là dòng *khai báo hàm* (function prototype - hay còn gọi là nguyên mẫu của hàm). Từ khoá **`int`** trong ngoặc thông báo cho trình biên dịch biết hàm `square()` nhận một tham số là số nguyên do hàm gọi nó truyền cho. Từ khoá **`int`** bên trái tên hàm thông báo cho trình biên dịch biết hàm `square()` trả lại kết quả là một số nguyên cho chương trình gọi nó. Chương trình biên dịch sẽ tham khảo dòng khai báo hàm để kiểm tra xem các tham số trong các lần gọi hàm `square()` có đúng như mô tả hay không. Khi tham số truyền cho hàm khác kiểu nguyên hoặc không tự động chuyển sang kiểu nguyên thì chương trình dịch sẽ đưa ra thông báo lỗi. Khai báo hàm sẽ nêu chi tiết hơn trong phần 2.3.

2.2 Tổng quát về định nghĩa hàm

Khuôn dạng của phần cài đặt hàm có dạng

```
[kiểu-giá-trị-trả-về]  tên-hàm ([danh-sách-tham-số])
{
        Các khai báo
        ...
        Các câu lệnh
}
```

Trong đó, *tên-hàm* là bất kỳ tên (identifier) hợp lệ nào, *kiểu-giá-trị-trả-về* là kiểu dữ liệu của kết quả trả lại cho hàm gọi nó. Khác với Pascal, C chỉ không cho phép hàm trả về giá trị là một biến mảng, còn mọi kiểu dữ liệu hợp lệ khác đều có thể sử dụng để mô tả *kiểu giá trị trả về* cho hàm. *Kiểu-giá-trị-trả-về* là **void** thì hàm không trả lại giá trị nào cả. Nếu không xác định *kiểu-giá-trị-trả-về* thì trình biên dịch sẽ ngầm định coi đó là **int**. *danh-sách-tham-số* mô tả kiểu dữ liệu cùng thứ tự của các tham số hàm nhận được khi nó được gọi. Tương ứng với mỗi tham số có một tham số hình thức cùng với kiểu tương ứng. Các tham số hình thức phân cách nhau bằng dấu phẩy ",". Cũng như kiểu giá trị trả về, C không đưa ra hạn chế gì về kiểu giá trị của các tham số. Nó có thể là một kiểu dữ liệu cơ sở hay một kiểu dữ liệu có cấu trúc, có thể khai báo trực tiếp các tham số với kiểu có cấu trúc trong định nghĩa của hàm. Nếu hàm không nhận tham số nào cả, *danh-sách-tham-số* là **void**. Kiểu của từng tham số cần ghi rõ ra, nếu không ghi kiểu, trình biên dịch sẽ ngầm hiểu là kiểu **int**. Hãy xem một số thí dụ định nghĩa hàm trong bảng 4.2:

Các khai báo và các câu lệnh trong cặp dấu "{}" tạo thành *phần thân của hàm*. Phần thân của hàm còn gọi là khối. Một khối đơn giản là một câu lệnh ghép có cả phần khai báo trong đó. Các biến có thể khai báo trong bất kỳ khối nào và các khối có thể lồng nhau. Tuy nhiên, khai báo và cài đặt một hàm lại không được nằm trong một hàm khác.

Có một số cách để trả lại quyền điều khiển lại điểm mà hàm đó được gọi. Nếu hàm không trả lại giá trị thì quyền điều khiển được trả lại khi gặp dấu đóng ngoặc cuối cùng của hàm đó hoặc khi thực hiện câu lệnh

```
return;
```

Nếu hàm trả lại kết quả thì câu lệnh

```
return biểu-thức;
```

sẽ trả lại giá trị tính được của biểu thức cho hàm gọi nó.

Định nghĩa hàm	Giải thích
`float fct1(float t[20]) {...}`	Hàm trả về giá trị là một số nguyên. Tham số truyền cho hàm là một mảng 20 thành phần là các số thực.
`float fct2(float *t) {...}`	Hàm trả về một số thực. Tham số truyền cho hàm là một biến mảng thực hoặc một con trỏ thực.
`float *fct3(float *t) {...}`	Hàm trả về giá trị là một con trỏ thực. Tham số truyền cho hàm là một biến mảng nguyên hoặc một con trỏ thực
`float fct4(int a, int b) {...}`	Hàm trả về giá trị là một số thực. Tham số truyền cho hàm là hai số nguyên.

Bảng 4.2 Một số thí dụ định nghĩa hàm

Chương trình `func2.c` sau đây sử dụng hàm `maximum()` do người sử dụng định nghĩa để trả lại số nguyên lớn nhất trong 3 số nguyên. Ba số nguyên ban đầu được nhập vào từ bàn phím bằng hàm `scanf()`. Sau đó chúng được truyền cho hàm `maximum()` để xác định số lớn nhất. Giá trị này được trả lại vào biến `largest` trong hàm `main()` bằng lệnh **`return`** trong hàm `maximum()`. Giá trị trong biến `largest` này sau đó được in ra trong câu lệnh `printf()`.

Bài tập mẫu 4.2

```
/*func2.c*/
/* Tìm số lớn nhất trong 3 số nguyên*/
#include <stdio.h>
int maximum(int,int,int);/* Hàm nguyên mẫu */
/* Hàm main */
void main() {
 int a, b, c, largest;
 printf( " Vao 3 so nguyen : ");
 scanf("%d %d %d", &a, &b, &c);
 largest = maximum(a,b,c);
 printf(" So lon nhat la : %d\n", largest);
```

```
 return ; /*trong hàm kiểu void thì dùng return ;*/
 }
/* Định nghĩa hàm maximum */
int maximum(int x, int y, int z) {
  int max = x;
  if (y > max)
     max = y;
  if (z > max)
     max = z;
  return max;
  }
```

```
Vao 3 so nguyen : 3 5 3
 So lon nhat la : 5
```

2.3 Hàm nguyên mẫu

Một trong những đặc trưng quan trọng của ANSI C là hàm nguyên mẫu (dạng hàm). Hàm nguyên mẫu thông báo cho chương trình biên dịch biết kiểu dữ liệu hàm trả lại, số lượng, kiểu và thứ tự của các tham số được truyền cho hàm. Chương trình biên dịch dùng hàm nguyên mẫu để kiểm tra các lời gọi hàm. Các phiên bản C trước đây không thực hiện quá trình kiểm tra này nên có thể xảy ra nhiều lỗi khi thực hiện ứng dụng rất khó phát hiện chẳng hạn do truyền sai thứ tự các tham số. Các hàm nguyên mẫu trong ANSI C giúp loại bỏ vấn đề này.

Hàm nguyên mẫu của hàm `maximum()` trong chương trình `func2.c` là:

```
int  maximum( int, int, int);
```

Hàm nguyên mẫu trên thông báo rằng `maximum()` nhận 3 tham số kiểu **int** và trả lại kết quả kiểu **int**. Chú ý rằng hàm nguyên mẫu giống dòng đầu tiên trong phần định nghĩa hàm nhưng có thể không cần các tham số hình thức.

Khi gặp một lời gọi hàm không đúng theo hàm nguyên mẫu, trình biên dịch sẽ thông báo lỗi cú pháp. Lỗi cú pháp cũng xuất hiện khi hàm nguyên mẫu và định nghĩa hàm khác nhau.

Một đặc tính quan trọng của hàm nguyên mẫu là khả năng ép kiểu các tham số, nghĩa là chuyển các tham số truyền vào theo các kiểu thích ứng. Ví dụ câu lệnh:

```
printf("%.3f", sqrt(4));
```

long double
double
float
unsigned long int
long int
unsigned int
int
short
char

Sẽ in ra giá trị 2.000. Hàm nguyên mẫu sẽ yêu cầu trình biên dịchchuyển số nguyên 4 thành số **double** 4.0 trước khi truyền cho hàm **sqrt**.

Nói chung, việc chuyển kiểu áp dụng theo qui tắc chuyển từ những kiểu dữ liệu nhỏ hơn sang những kiểu dữ liệu lớn hơn (thứ tự xem hình bên). Nếu chuyển sang các kiểu thấp hơn có thể dẫn đến sai kết quả.

Nếu không có hàm nguyên mẫu, trình biên dịch sẽ tự tạo ra hàm nguyên mẫu của hàm khi gặp hàm lần đầu tiên thông qua định nghĩa hàm hoặc lời gọi hàm. Khi gặp lời gọi hàm trước tiên, hàm nguyên mẫu tương ứng sẽ có giá trị trả về kiểu nguyên **int** và không có một ràng buộc gì về danh sách các tham số, khi đó trình biên dịch sẽ không phát hiện được các lỗi về truyền sai tham số và sai kiểu trả về của hàm.

2.4 Các tệp tiêu đề (header file)

Mỗi một thư viện chuẩn tương ứng có một tệp tiêu đề chứa các khai báo của tất cả các hàm trong thư viện này cùng với các định nghĩa các kiểu dữ liệu khác nhau, các hằng dùng trong các hàm đó. Bảng 4.3 liệt kê các tệp tiêu đề có thể dùng trong chương trình. Khái niệm macro dùng trong bảng 4.3 sẽ được mô tả chi tiết trong phần *"Tiền xử lý"* (Preprocessor).

Một trong những khó khăn lớn nhất khi mới học ngôn ngữ C là không biết cách sử dụng các hàm thư viện chuẩn trong C. Có rất nhiều cách để vượt qua khó khăn này. Thứ nhất là đọc các tài liệu về ngôn ngữ. Tuy nhiên theo kinh nghiệm, việc sử dụng các công cụ trợ giúp đi kèm các chương trình dịch tỏ ra hiệu quả hơn nhiều. Để thực hiện điều này, có hai cách:

(i) **Nếu biết tên hàm**: trong màn hình soạn thảo của chương trình biên dịch C (Turbo C chẳng hạn) ta đánh vào tên hàm cần tra cứu, màn hình trợ giúp của C hiện lên sẽ cho ta biết các thông tin liên quan đến hàm như là: các tệp tiêu đề cần gộp vào qua chỉ thị tiền xử lý `#include` khi sử dụng hàm, dạng khai báo của hàm; và thậm chí còn có thể có những chương trình ví dụ rất thú vị về hàm cần tra cứu.

(ii) **Tìm hiểu nội dung một tệp tiêu đề**. Cách làm tương tự như trong trường hợp thứ nhất. Trong trường hợp này màn hình trợ giúp hiển thị cho ta danh sách các hàm, hằng, biến khai báo bên trong tệp tiêu đề. Việc tra cứu các hàm, hằng, biến được thực hiện bằng việc di chuyển vệt sáng trên màn hình trợ giúp và ấn Enter để xem.

Lập trình viên có thể tạo các tệp tiêu đề riêng của mình. Các tệp tiêu đề này cũng thường có phần mở rộng `.h` và có thể dùng chỉ thị tiền xử lý `#include` để thêm vào chương trình. Ví dụ tệp tiêu đề `square.h` có thể đưa vào trong chương trình của ta bằng chỉ thị `#include "square.h"` ở đầu tệp chương trình. Chương 8 sẽ giới thiệu thêm về vấn đề này.

Tệp *.h	Nội dung
`<asset.h>`	Các macro và thông tin trợ giúp lập trình debbug.
`<ctype.h>`	Các hàm nguyên mẫu của các hàm kiểm tra một số thuộc tính của các ký tự, hay chuyển ký tự giữa chữ hoa và chữ thường.
`<errno.h>`	Các macro hiển thị các thông tin về lỗi.
`<float.h>`	Các giới hạn của số dấu phẩy động của hệ thống.
`<limits.h>`	Các giới hạn của hệ thống
`<locale.h>`	Các hàm và các thông tin liên quan đến việc thay đổi ngôn ngữ và mã nước của chương trình đang chạy
`<math.h>`	Các hàm nguyên mẫu của các hàm thư viện toán học
`<setjmp.h>`	Các hàm nguyên mẫu của các hàm cho phép bỏ qua lời gọi hàm thông thường và trả quyền điều khiển lại một vị trí khác.
`<signal.h>`	Các hàm nguyên mẫu và các macro điều khiển các điều kiện khác nhau có thể xuất hiện trong quá trình thực hiện chương trình.
`<stdarg.h>`	Định nghĩa các macro làm việc với danh sách các tham số đối với những hàm mà số lượng và kiểu của các tham số là không biết.
`<stddef.h>`	Định nghĩa các kiểu chung dùng để thực hiện tính toán.
`<stdio.h>`	Các hàm nguyên mẫu và các thông tin liên quan của các hàm thư viện vào/ra chuẩn.
`<stdlib.h>`	Các hàm nguyên mẫu và các thông tin liên quan của các hàm chuyển kiểu, cung cấp bộ nhớ, tạo số ngẫu nhiên ...
`<string.h>`	Các hàm nguyên mẫu của các hàm xử lý xâu.
`<time.h>`	Các hàm nguyên mẫu và kiểu dữ liệu cho thao tác thời gian.

Bảng 4.3 Các tệp tiêu đề của các thư viện chuẩn.

3. THAM SỐ TRONG LỜI GỌI HÀM

3.1 Một số khái niệm

Tham số hình thức: Khái niệm này chỉ đến các tham số được khai báo trong phần *danh sách tham số* trong định nghĩa hàm. Ta gọi đây là tham số hình thức vì thực tế chúng chỉ đóng vai trò "người đại diện" cho các tham số thực trong các lời gọi hàm. Mỗi tham số hình thức sẽ tương ứng đại diện cho một tham số thực. Kiểu dữ liệu của tham số hình thức sẽ quyết định kiểu giá trị cho tham số thực tương

ứng. Chẳng hạn, nếu tham số hình thức được khai báo như là con trỏ thì tham số thực tương ứng phải là địa chỉ của một biến nguyên hoặc là tên của biến mảng.

Tham số thực (sự): Khái niệm này để chỉ đến các thông tin được truyền cho hàm trong các lời gọi hàm. Mỗi tham số thực tương ứng với một tham số hình thức.

Liên quan đến việc truyền thông tin cho hàm, ta phân biệt: *truyền theo trị* và *truyền theo tham biến*. Khi các tham số được truyền theo trị, một bản sao giá trị của tham số thực (có thể là giá trị của biến, hằng hay của một biểu thức) được tạo ra và gán cho các tham số hình thức của hàm. Vì vậy mọi sự thay đổi trong hàm trên bản sao sẽ không ảnh hưởng đến giá trị ban đầu của biến (nếu có) nằm trong hàm gọi. Khi một tham số được truyền theo tham biến, hàm gọi sẽ truyền trực tiếp tham số đó (bắt buộc phải là biến) cho hàm được gọi. Trong trường hợp này tham số hình thức và tham số thực là một. Như vậy, nếu bên trong hàm bị gọi có thay đổi đến tham số hình thức thì những thay đổi đó cũng có tác dụng trên các tham số thực tương ứng. Điều này đôi khi dẫn đến hiệu ứng phụ.

Truyền theo trị được dùng bất cứ khi nào hàm bị gọi không cần thiết thay đổi giá trị biến mà hàm gọi truyền vào tham số cho nó. Điều này giúp tránh được hiệu ứng phụ, lỗi mà các chương trình con thay đổi các biến toàn cục của chương trình gọi nó. Các lỗi này rất khó phát hiện. Truyền theo tham biến chỉ nên dùng khi hàm bị gọi thật sự cần thiết thay đổi giá trị của biến truyền vào cho nó.

3.2 Truyền tham số trong C

Trong C, tất cả các tham số được truyền theo trị. Chương trình `swap1.c` sau đây là một minh chứng cho nhận xét này.

Bài tập mẫu 4.3

```
/*swap1.c*/
#include <stdio.h>
#include <conio.h>
void swap(int a, int b);
void main() {
  int x=3,y=4;
  clrscr();
  printf("Truoc khi goi swap\n");
  printf("%4d %4d\n",x,y);
  swap(x,y);
  printf("Sau khi goi swap\n");
  printf("%4d %4d\n",x,y);
```

```
  getch();
  }
```

```
Truoc khi goi swap
   3     4
Sau khi goi swap
   3     4
```

3.3 Tham số hình thức của hàm là con trỏ

Khi tham số hình thức của hàm là con trỏ thì tham số thực tương ứng phải là một địa chỉ, có thể là địa chỉ của một biến hoặc tên của một biến mảng. Khi truyền địa chỉ của biến cho hàm người ta có thể tuỳ thích thay đổi giá trị của biến với địa chỉ được truyền. Còn khi truyền tên của mảng thì sẽ có hai cái lợi so với việc khai báo tham số hình thức như một mảng: thứ nhất tiết kiệm được thời gian sao chép dữ liệu, thứ hai có thể thay đổi giá trị của các phần tử trong biến mảng được truyền vào. Sau đây giới thiệu một số chương trình minh hoạ.

Chương trình `swap2.c` sau đây là một phiên bản sửa đổi từ `swap1.c` trong đó các tham số hình thức trong hàm `swap()` là con trỏ.

Bài tập mẫu 4.4

```
/*swap2.c*/
#include <stdio.h>
#include <conio.h>
void swap(int *a, int *b);
void main() {
  int x=3,y=4;
  clrscr();
  printf("Truoc khi goi swap\n");
  printf("%4d %4d\n",x,y);
  swap(&x,&y);
  printf("Sau khi goi swap\n");
  printf("%4d %4d\n",x,y);
  getch();
  }
void swap(int *a, int *b) {
```

```
  int temp = *a;
  *a = *b; /*thay đổi giá trị của biến có địa chỉ truyền cho hàm*/
  *b = temp;
  }
```

```
Truoc khi goi swap
   3     4
Sau khi goi swap
   4     3
```

Chương trình `func3.c` sau đây thực hiện các thao tác nhập và in và sắp xếp một dãy số trong đó tên mảng được truyền như một tham số cho hàm.

Bài tập mẫu 4.5

```
/*func3.c*/
#include <stdio.h>
#include <conio.h>
void NhapDS(int *d);
void InDS(int *d);
void SapxepDS(int *d);
void main() {
  int a[5];
  clrscr();
  printf("Nhap va day so\n");
  NhapDS(a);
  printf("Day so vua nhap\n");
  InDS(a);
  SapxepDS(a);
  printf("Day so sau khi sap xep\n");
  InDS(a);
  printf("An phim bat ky de ket thuc...");
  getch();
  }
void NhapDS(int *d) {
```

```
  int i;
  int temp;
  for(i =0; i<5;i++) {
    printf("Phan tu thu %d:",i+1);
    scanf("%d",&temp);
    d[i]=temp;
    }
  }
void InDS(int *d) {
  int i;
  for(i=0; i<5; i++)
    printf("%4d ",d[i]);
  printf("\n");
  }
void SapxepDS(int *d) {
  int i,j;
  for(i=0;i<4;i++)
    for(j=i+1;j<5;j++)
      if(d[i]>d[j]) {
        int temp=d[i];
        d[i]=d[j];
        d[j]=temp;
        }
  }
```

```
Nhap va day so
Phan tu thu 1:6
Phan tu thu 2:-3
Phan tu thu 3:3
Phan tu thu 4:2
Phan tu thu 5:10
Day so vua nhap
```

```
   6   -3    3    2   10
Day so sau khi sap xep
  -3    2    3    6   10
An phim bat ky de ket thuc...
```

3.4 Tham số của hàm main()

Chúng ta đề cập đến tham số của hàm `main()` khi viết các chương trình có tham số dòng lệnh. Trong trường hợp này hàm `main()` sẽ được khai báo với hai tham số hình thức:

```
void main(int agrc, char *agrv[]) {...}
```

Tham số `argc` mô tả số lượng các tham số dòng lệnh bao gồm cả tên chương trình. Tham số `argv` là một mảng các xâu ký tự tương ứng với các tham số dòng lệnh. Chương trình sau đây minh hoạ cách viết chương trình sử dụng tham số dòng lệnh. Giá trị của `argc` và nội dung của `argv` được tự động xác định khi thực hiện chương trình.

Bài tập mẫu 4.6

```
#include <stdio.h>
#include <conio.h>
void main(int argc, char *argv[]) {
  int i;
  clrscr();
  for(i=0;i<argc;i++)
 printf("Tham so thu %d la %s",i+1,argv[i]);
  getch();
  }
```

```
Tham so thu 1 la C:\TEMP\COM_LINE.EXE
Tham so thu 2 la abc
Tham so thu 3 la mnp
Tham so thu 4 la xyz
```

Chú ý

Trong môi trường IDE của Borland C hoặc Turbo C ta đưa vào các tham số dòng lệnh cho chương trình nhờ lệnh **Run-->Arguments...**

4. CẤP LƯU TRỮ VÀ PHẠM VI CỦA CÁC ĐỐI TƯỢNG

4.1 Khái niệm

Ta biết rằng mỗi đối tượng (biến hoặc hàm) sử dụng trong một chương trình cần phải có các khai báo (và định nghĩa). Các khai báo (và định nghĩa) này có thể được đặt ở ngoài tất cả các hàm hoặc ở bên trong một hàm nào đó (điều này chỉ đúng đối với các biến). Các khai báo đặt ở các vị trí khác nhau làm cho đối tượng được khai báo có các tính chất khác nhau. Sự khác nhau đó thể hiện ở phạm vi sử dụng (scope), thời gian tồn tại (lifetime) và *cấp lưu trữ* (storage class).

Phạm vi của một đối tượng là phần chương trình mà đối tượng còn được nhìn thấy và có thể sử dụng bởi phần chương trình đó. Phạm vi có thể là trong một khối lệnh, một hàm hoặc một tệp nguồn hoặc toàn thể chương trình.

Một số đối tượng có thời gian tồn tại rất ngắn, một số thì liên tục được tạo ra rồi xoá đi, một số khác tồn tại trong suốt thời gian thực hiện chương trình. Sau khi một đối tượng đã được khai báo (và định nghĩa) máy tính sẽ cung cấp cho biến hay hàm vùng nhớ thường trực hay động trong thời gian thực hiện chương trình. Như vậy, một đối tượng có thể có thời gian tồn tại lâu dài hay tạm thời (so với thời gian thực hiện của chương trình).

Cấp lưu trữ của đối tượng được xác định dựa theo vùng bộ nhớ được cấp phát. Các đối tượng được cấp phát bộ nhớ thường trực thì có cấp lưu trữ cố định (static). Thuộc loại này bao gồm các hàm và các biến khai báo bên ngoài các hàm. Bên cạnh đó cũng có một số biến cục bộ có thuộc tính này, chúng ta sẽ xem xét vấn đề này sau. Các đối tượng (biến hoặc hàm) có cấp lưu trữ cố định tồn tại ngay khi chương trình bắt đầu thực hiện. Đối với biến, vùng nhớ được cung cấp và khởi tạo ngay khi bắt đầu chương trình. Đối với hàm, tên của hàm cũng tồn tại ngay khi chương trình bắt đầu thực hiện. Tuy nhiên, mặc dù biến và tên hàm tồn tại ngay khi bắt đầu chương trình nhưng không có nghĩa là đối tượng này có thể dùng bất cứ đâu trong chương trình.

Các biến có cấp lưu trữ động được tạo ra khi vào khối mà chúng được khai báo, chúng tồn tại khi điều khiển còn đang trong khối đó và chúng sẽ bị xoá khi quyền điều khiển thoát ra khỏi khối.

Phạm vi và thời gian tồn tại và cấp lưu trữ của một đối tượng được xác định bằng cách tổ hợp các từ khoá chỉ định cấp lưu trữ với các khai báo cùng vị trí đặt bên trong hay bên ngoài một hàm hay một khối lệnh.

Ngôn ngữ C cung cấp 4 từ khoá sau: **auto, extern, static** và **register** để chỉ định phạm vi và thời gian tồn tại của các biến. Chúng có thể chia thành hai nhóm theo thời gian tồn tại hay phạm vi. Các từ khoá `auto` và `register` dùng để khai báo các đối tượng có cấp lưu trữ động và phạm vi cục bộ. Các từ khoá **extern** và **static** dùng để khai báo các đối tượng (biến hoặc hàm) có thời gian lưu trữ tĩnh và phạm vi toàn cục.

Từ khoá auto

Từ khoá **auto** được dùng để khai báo các biến cục bộ. Tuy nhiên rất ít khi người ta sử dụng nó vì các biến cục bộ đã mặc nhiên được xem là `auto`.

Từ khoá register

Từ khoá **register** được đặt trước khai báo của các biến tự động nhằm yêu cầu chương trình biên dịch duy trì giá trị biến đó trong thanh ghi của máy tính, do đó nâng cao tốc độ thực hiện (vì truy nhập, thao tác dữ liệu trực tiếp trên thanh ghi máy tính là nhanh nhất). Ví dụ, sau đề nghị trình biên dịch đặt biến `counter` vào một trong các thanh ghi của máy và khởi tạo nó giá trị ban đầu bằng 1:

```
register int counter = 1;
```

Chương trình biên dịch có thể bỏ qua từ khoá **register** khi không còn đủ thanh ghi nữa và nói chung chúng chỉ áp dụng cho các biến kiểu **int** và **char**. Ngày nay, các khai báo **register** thường cũng rất ít được dùng vì bộ tối ưu của trình biên dịch thường có khả năng nhận biết được các biến thường xuyên được sử dụng và đặt chúng vào các thanh ghi nếu thấy cần thiết mà không cần thiết có các từ khoá **register** trong chương trình nguồn.

Từ khoá static

Biến tĩnh là biến có thời gian tồn tại cùng với chương trình. Biến tĩnh được tạo ra khi chương trình khởi động chương trình. Chỉ khi nào chương trình kết thúc các biến tĩnh mới được giải phóng khỏi bộ nhớ.

Theo định nghĩa, các biến khai báo bên ngoài các hàm sẽ là biến tĩnh vì chúng tồn tại độc lập với các hàm, có thể sử dụng trong mọi hàm miễn là hàm đó không khai báo các biến cục bộ cùng tên.

Bên cạnh các biến ngoài có thể khai báo các biến cục bộ tĩnh bằng cách đặt từ khoá **static** trước khai báo biến. Cú pháp như sau:

```
static <khai báo biến>
```

Ví dụ

```
{
...
static int x;/*x là biến cục bộ tĩnh*/
}
```

Từ khoá **static** đi với biến cục bộ thông báo cho trình biên dịch biết cần cung cấp một vùng lưu trữ thường xuyên cho biến cục bộ này. Khác với biến toàn cục, các biến tĩnh cục bộ không được nhận biết ở bên ngoài hàm khai báo chúng nhưng

chúng vẫn duy trì được giá trị của chúng giữa mỗi lần gọi hàm, đây là điểm khác với biến cục bộ thông thường. Các biến loại này có ứng dụng trong việc viết những hàm cần phải bảo toàn một số giá trị giữa mỗi lần gọi.

Chương trình `func4.c` sau đây minh hoạ cách sử dụng biến cục bộ `static`:

Bài tập mẫu 4.7

```
/*func4.c*/
#include <stdio.h>
#include <conio.h>
void fct() ;
void main() {
  int i;
  clrscr();
  for(i=1;i<=10;i++)
    fct();
  getch();
  }
void fct() {
  static int count = 1;
  printf("Loi goi ham thu %d\n",count++);
  }
```

```
Loi goi ham thu 1
Loi goi ham thu 2
Loi goi ham thu 3
Loi goi ham thu 4
Loi goi ham thu 5
Loi goi ham thu 6
Loi goi ham thu 7
Loi goi ham thu 8
Loi goi ham thu 9
Loi goi ham thu 10
```

Tất cả các biến có thời gian lưu trữ tĩnh, ngầm định đều được khởi tạo giá trị 0 (con trỏ thì được khởi tạo giá trị NULL).

Trong chương trình func4.c nếu ta khai báo một biến toàn cục để lưu giá trị của count thì bất kỳ chương trình nào muốn sử dụng hàm fct() cũng phải xem xét việc khai báo biến đó, phải đảm bảo không có sự nhầm lẫn nào với các biến khác được khai báo và rất khó có thể đưa hàm này vào thư viện.

Chú ý rằng tất cả các hàm đều là đối tượng ngoài vì ngôn ngữ C không cho phép một hàm được định nghĩa bên trong hàm khác.

Từ khoá extern

Các biến ngoài chỉ có thể sử dụng được trong các hàm định nghĩa trong tệp nguồn khai báo chúng. Khi phát triển các ứng dụng lớn, hiển nhiên là các chương trình nguồn phải được lưu trữ trên nhiều tệp khác nhau. Bởi vì chương trình có thể do nhiều người cùng lập, đồng thời việc phân nhỏ chương trình nguồn còn làm tăng tốc độ biên dịch. Vậy làm cách nào các hàm ở một tệp này có thể truy nhập tới các biến được khai báo hoặc gọi tới các hàm được định nghĩa ở một tệp khác? Từ khoá **extern** cho phép sử dụng các biến hàm trên phạm vi nhiều tệp. Nguyên tắc như sau:

Chẳng hạn tệp A có chứa khai báo các biến toàn cục sau :

```
int    x, y;
char   ch;
```

Tệp B muốn sử dụng các biến toàn cục trên ở tệp A thì cần khai báo :

```
extern int    x, y;
extern char   ch;
```

Từ khoá **extern** thông báo với trình biên dịch rằng tên và kiểu của biến đặt sau đã được khai báo ở đâu đó rồi. Nói cách khác, **extern** thông báo với trình biên dịch không cần cấp phát thêm vùng nhớ cho các biến đó một lần nữa. Khi liên kết, trình liên kết sẽ tự giải quyết vấn đề tham khảo tới biến công cộng được phân phối ở ngoài tệp B.

4.2 Bốn kiểu phạm vi trong chương trình C

Phạm vi của một đối tượng (biến hay hàm) là phần của chương trình mà đối tượng đó có thể được tham chiếu tới. Ví dụ khi ta khai báo một biến cục bộ trong một khối, nó chỉ có thể được tham chiếu trong khối đó hoặc các khối lồng trong khối đó thôi. Có bốn loại phạm vi khác nhau : phạm vi trong hàm, phạm vi trong tệp, phạm vi trong khối và phạm vi trong hàm nguyên mẫu.

Các nhãn của chương trình (danh biểu theo sau là dấu hai chấm như **start:**) chỉ

có phạm vi trong hàm. Các nhãn được dùng trong lệnh **switch** và lệnh nhảy **goto** có thể được dùng bất cứ đâu trong thân hàm nhưng không thể bị tham chiếu bởi các câu lệnh ở ngoài thân hàm đó.

Một đối tượng khai báo ở ngoài các hàm thì có phạm vi trong tệp đó. Các biến toàn cục, định nghĩa hàm và các hàm nguyên mẫu được đặt ở ngoài tất cả các hàm đều có phạm vi tệp.

Các danh biểu khai báo trong khối có phạm vi trong khối đó. Phạm vi của khối kết thúc khi gặp kí tự đóng ngoặc kết thúc khối "}" của khối đó. Các biến cục bộ được khai báo đầu hàm có phạm vi khối như các tham số của hàm đó. Bất cứ một khối nào cũng có thể khai báo các biến trong đó. Khi các khối lồng nhau, các biến ở trong một khối chỉ có phạm vi là khối đó và tất cả các khối lồng trong nó. Các biến tĩnh cục bộ cũng có phạm vi khối, mặc dù chúng tồn tại từ khi chương bắt đầu thực hiện. Có thể nói thuộc tính thời gian tồn tại độc lập với thuộc tính phạm vi.

Cuối cùng, các danh biểu dùng trong danh sách các tham số của hàm nguyên mẫu chỉ có phạm vi trong hàm nguyên mẫu. Các hàm nguyên mẫu không yêu cầu tên trong danh sách các tham số mà chỉ yêu cầu kiểu, nếu có tên tham số, trình biên dịch sẽ bỏ qua. Vì vậy các danh biểu dùng trong hàm nguyên mẫu có thể dùng lại ở những nơi khác.

4.3 Tệp project

Khi viết các chương trình lớn người ta thường để các hàm trên nhiều tệp nguồn khác nhau. Tệp project là hình thức để liên kết các tệp nguồn đó với nhau trước khi biên dịch. Như vậy có thể hiểu rằng nội dung của một tệp project là danh sách tên các tệp dùng để tạo nên một chương trình ứng dụng. Thông thường các tệp này là chương trình nguồn, tuy nhiên có thể sử dụng các tệp đối tượng (*.OBJ) sinh ra do việc biên dịch các chương trình nguồn *.C hoặc các tệp viết trên ngôn ngữ Assembly. Mặc dù vậy vẫn chỉ có duy nhất một tệp nguồn chứa hàm `main()`, các tệp thành phần khác chứa các hàm thư viện tự định nghĩa để sử dụng trong chương trình. Thứ tự các tệp thành phần trong tệp project không quan trọng. Sau đây chúng ta nói về các thao tác trên một tệp project.

Mở một tệp project

Tệp project thường có phần mở rộng là "`prj`". Để mở một tệp project , thực hiện lệnh **Project-->Open project ...** sau đó lựa chọn một tệp project đã có hoặc đăng ký tên cho một tệp project mới. Tại mỗi thời điểm chỉ có thể mở không quá một tệp project. Như vậy nếu đang làm việc với một tệp project việc mở một tệp project khác sẽ đóng tệp project trước đó.

Cửa sổ project

Một khi đã đăng ký tên tệp project mới hay mở một tệp project đã có, trên màn hình xuất hiện thêm một cửa sổ tệp project. Trong cửa sổ này hiện lên tên các tệp thành phần.

Thêm một tệp vào tệp project

Để thêm một tệp thành phần, chọn cửa sổ project, sau đó thực hiện lệnh **Project-->Add item...** hay nhấn phím INS (Insert) và thực hiện theo các chỉ dẫn trên màn hình.

Bỏ một tệp khỏi tệp project

Để loại bỏ một tệp thành phần khỏi tệp project,; trước hết chọn cửa sổ project, sau đó lựa chọn tệp cần xoá, cuối cùng thực hiện lệnh **Project-->Delete** item hoặc nhấn phím Del(Delete).

Biên dịch và thực hiện một tệp project

Khi một tệp project được mở thì mọi thao tác biên dịch (F9, Alt-F9) và thực hiện (Ctrl - F9) chỉ liên quan đến các tệp thành phần trong tệp project. Mọi tệp chương trình khác đều không được xem xét. Vì vậy nếu muốn biên dịch và thực hiện một chương trình nguồn đơn lẻ khác thì phải đóng tệp project đang mở.

Đóng tệp project

Thực hiện lệnh **Project-->Close project.**

5. CON TRỎ HÀM

5.1 Khái niệm

Mặc dù một hàm không phải là một biến nhưng nó vẫn chiếm vị trí trong bộ nhớ và ta có thể gán vị trí của nó cho một loại biến con trỏ. Con trỏ này trỏ đến điểm xâm nhập vào hàm. Ta gọi đây là con trỏ hàm. Con trỏ hàm có thể sử dụng thay cho tên hàm và việc sử dụng con trỏ cho phép các hàm cũng được truyền như là tham số cho các hàm khác.

Để hiểu được các con trỏ hàm làm việc như thế nào, ta cần hiểu một chút về cách biên dịch và gọi một hàm. Khi biên dịch hàm, trình biên dịch chuyển chương trình nguồn sang dạng mã máy và thiết lập một điểm xâm nhập vào hàm (chính là vị trí của chỉ thị mã máy đầu tiên của hàm). Khi có lời gọi thực hiện hàm, máy tính sẽ thực hiện một chỉ thị call chuyển điều khiển đến điểm xâm nhập này. Trong trường hợp gọi hàm bằng tên hàm thì điểm xâm nhập này là trị tức thời (gần như là một hằng và không chứa trong biến nào cả), cách gọi hàm này gọi là cách gọi hàm trực tiếp. Trái lại, khi gọi hàm gián tiếp thông qua một biến trỏ thì biến trỏ đó phải trỏ tới chỉ thị mã máy đầu tiên của hàm đó. Cách gọi hàm thông qua biến trỏ hàm gọi là cách gọi hàm gián tiếp.

5.2 Khai báo biến con trỏ hàm

```
[kiểu_giá_trị] (*tên_biến_trỏ_hàm) ([danh_sách_tham_số]);
```

Ý nghĩa của các thành phần trong khai báo con trỏ hàm giống như đối với khai báo hàm.

Một con trỏ hàm chỉ có thể nhận giá trị là tên của các hàm có cùng kiểu giá trị trả về và kiểu giá trị của các tham số. Chẳng hạn con trỏ hàm:

```
int f_ptr1(int);
```

chỉ có thể lấy giá trị của các hàm một tham số kiểu nguyên **int** và có giá trị trả về cũng là kiểu nguyên **int**.

trong khi con trỏ hàm:

```
void f_ptr2(int,float);
```

thì có thể nhận giá trị là tên các hàm có kiểu giá trị trả về là **void** và có hai tham số, trong đó tham số thứ nhất có kiểu nguyên **int** còn tham số thứ hai kiểu thực **float**.

5.3 Tham số hình thức của hàm là con trỏ hàm

Như phần 2.2 đã đề cập, C không hạn chế gì về kiểu giá trị của tham số hình thức. Vì vậy, có thể khai báo tham số hình thức của hàm là một con trỏ hàm. Khi đó trong lời gọi hàm chúng ta có thể sử dụng tên một hàm khác làm tham số thực. Chương trình `func5.c` sau đây thực hiện việc sắp xếp dãy số theo nhiều tiêu chuẩn khác nhau:

(i) sắp xếp tăng dần theo giá trị.

(ii) sắp xếp tăng dần theo trị tuyệt đối.

mà chỉ dùng một hàm sắp xếp.

Bài tập mẫu 4.8

```
/*func5.c*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
void NhapDS(int *d);
void InDS(int *);
void Sapxep(int *, int (*cmp)(int,int));
int tanggt(int,int);
int tangtd(int,int);
void main() {
```

```
  int a[5];
  clrscr();
  printf("Nhap vao day so nguyen\n");
  NhapDS(a);
  printf("Day so vua nhap\n");
  InDS(a);
  Sapxep(a,tanggt);
  printf("Day so sau khi sap xep tang\n");
  InDS(a);
  Sapxep(a,tangtd);
  printf("Day so sau khi sap xep tang theo gt tuyet doi\n");
  InDS(a);
  getch();
  }
void NhapDS(int *d) {
  int i;
  int temp;
  for(i=0;i<5;i++) {
    printf("Phan tu thu %d:",i+1);
    scanf("%d",&temp);
    d[i]=temp;
    }
  }
void InDS(int *d) {
  int i;
  for(i=0;i<5;i++)
    printf("%4d ",d[i]);
  printf("\n");
  }
void Sapxep(int *d,int (*cmp)(int,int)) {
  int i,j;
  for(i=0;i<4;i++)
```

```
    for(j=i+1;j<5;j++)
      if(cmp(d[i],d[j])) {
        int temp=d[i];
        d[i]=d[j];
        d[j]=temp;
      }
  }
int tanggt(int a,int b) {
  return a>b?1:0;
}
int tangtd(int a,int b) {
  return abs(a)>abs(b)?1:0;
}
```

```
Nhap vao day so nguyen
Phan tu thu 1:5
Phan tu thu 2:3
Phan tu thu 3:-10
Phan tu thu 4:-100
Phan tu thu 5:6
Day so vua nhap
   5    3  -10 -100    6
Day so sau khi sap xep tang
-100  -10    3    5    6
Day so sau khi sap xep tang theo gt tuyet doi
   3    5    6  -10 -100
```

Chú ý

(i) Các hàm có thể trở thành đối tượng được gọi trong hàm khác nên khai báo trước.

(ii) Khi gọi một hàm bằng con trỏ hàm `f` cần phải có dấu ngoặc, hàm được gọi theo kiểu `f`(danh sách các tham số nếu có).

(iii) Không có sự khác biệt nào giữa việc gọi bằng tên của hàm và gọi bằng con trỏ hàm.

6. ĐỆ QUI

Các chương trình đã xét đều có chung cấu trúc phân cấp giữa hai hàm gọi và hàm bị gọi. Tuy nhiên trong một số bài toán, việc dùng hàm gọi ngay chính nó rất hữu dụng. Có thể định nghĩa *hàm đệ qui* là hàm sẽ gọi đến chính nó trực tiếp hay gián tiếp thông qua các hàm khác. Vấn đề đệ qui là một vấn đề rất phức tạp, vì vậy trong phần này chỉ giới thiệu những khía cạnh cùng với những ví dụ đơn giản nhất của vấn đề đệ qui. Trước tiên ta xem xét khái niệm đệ qui, sau đó kiểm tra trên một vài chương trình có chứa các hàm đệ qui. Cách tiến hành giải một bài toán đệ qui nhìn chung có những điểm chung sau:

Hàm đệ qui thực ra chỉ giải bài toán trong trường hợp đơn giản nhất hay còn gọi là *trường hợp cơ sở*. Khi hàm đệ qui được gọi trong trường hợp cơ sở, hàm chỉ cần trả lại kết quả. Nếu hàm được gọi trong các trường hợp phức tạp hơn, hàm đệ qui sẽ chia công việc cần giải quyết thành hai phần. Một phần hàm biết cách giải quyết như thế nào, còn phần kia vẫn không biết cách giải quyết như thế nào tuy nhiên để được gọi là có khả năng đệ qui, phần sau phải giống với bài toán ban đầu nhưng đơn giản hơn hay nhỏ hơn bài toán ban đầu. Bởi vì bài toán mới giống với bài toán ban đầu nên hàm sẽ thực hiện gọi chính nó để giải quyết công việc đơn giản hơn này - đây chính là lời gọi đệ qui hay còn gọi là một bước đệ qui. Để đảm bảo việc đệ qui có kết thúc, mỗi một lần gọi đệ qui thì bài toán phải đảm bảo đơn giản hơn và các bước đệ qui này còn thực hiện tiếp cho đến khi nào bài toán đơn giản dần, đơn giản tới mức trở thành *trường hợp cơ sở*. Ta nhận thấy hàm đệ qui xử lý *trường hợp cơ sở* để trả lại kết quả tính được cho các hàm mức phức tạp hơn, rồi đến lượt các hàm này lại tính trả lại kết quả cho các hàm phức tạp hơn nữa ... cứ như vậy cho đến lời gọi hàm ban đầu. Trước khi đưa ra các nhận xét tiếp, ta xem xét ví dụ tính n giai thừa (**n!**) sau:

Theo định nghĩa n! bằng tích của tất cả các số tự nhiên từ 1 đến n:

```
n! = n * (n-1) ... 2 * 1
```

Ví dụ `5! = 5 * 4 * 3 * 2 * 1 = 120`, `0!` được định nghĩa bằng 1. Để tính n! có thể dùng vòng lặp như sau :

```
factorial = 1;
for (i = n; i >= 1; i--)
factorial *= i;
```

Tuy nhiên cũng có thể định nghĩa đệ qui hàm giai thừa như sau :

nếu n>0 $\quad$ n! = n * (n-1)!

nếu n=0 $\quad$ n! = 0! = 1

Khi đó để tính 3! quá trình tính phải lần ngược trở về tính 0!, sau đó lấy kết quả để tính 1!, lại lấy kết quả để tính 2! và cuối cùng mới tính được 3! :

3! = 3*(2!) = 3*(2*(1!)) = 3*(2*(1*(0!)))

3! = 3*(2*(1*1))) = 3*(2*1) = 3*2 = 6

Bài tập mẫu 4.9

```
/*Tính giai thừa theo phương pháp đệ qui*/
#include <stdio.h>
long factorial( long ); /*Hàm nguyên mẫu*/
void main() {
int i;
for ( i=0; i<=10; i=i+2)
printf("%2d! = %ld\n", i, factorial(i));
}
/*Định nghĩa đệ qui hàm factorial*/
long factorial( long number){
if ( number = 0 )
        return 1;
else
        return (number*factorial(number-1));
}
```

```
0! = 1
2! = 2
4! = 24
6! = 720
8! = 40320
10! = 3628800
```

Ví dụ thứ hai dùng đệ qui là tính dãy số Fibonaci. Dãy số Fibonaci gồm những số:

```
0, 1, 1, 2, 3, 5, 8, 13, 21 ...
```

bắt đầu từ hai số 0 và 1, tiếp sau đó các số Fibonaci sau bằng tổng của 2 số Fibonaci trước nó.

Dãy Fibonaci gặp rất nhiều trong thực tế, đặc biệt khi mô tả các mô hình xoắn ốc. Tỉ số giữa 2 số Fibonaci liên tiếp khoảng 1,618... Số này gặp rất nhiều trong thực tế và được gọi là *tỉ số vàng*. Các kiến trúc sư thường thiết kế kích thước cửa sổ, phòng, nhà ở có tỉ lệ giữa chiều dài và chiều rộng bằng tỉ số vàng. Các bưu thiếp cũng thường có tỉ lệ giữa chiều dài và chiều rộng bằng tỉ số vàng.

Dãy Fibonaci có thể định nghĩa đệ qui như sau:

fibonaci(0) = 0; fibonaci(1) = 1;

fibonaci(n) = fibonaci(n-1) + fibonaci(n-2) ∀n>1

Chương trình trong bài tập mẫu 4.10 tính đệ qui số Fibonaci thứ i bằng cách dùng hàm `fibonaci()`. Chú ý rằng dãy số Fibonaci tăng rất nhanh, vì vậy chúng ta chọn kiểu long cho tham số và giá trị trả về của hàm `fibonaci()`.

Bài tập mẫu 4.10

```
/* Tính dãy số Fibonaci, theo phương pháp đệ qui */
#include <stdio.h>
long fibonaci( long );        /* Hàm nguyên mẫu */
void main() {
long result, number;
printf("Hay nhap vao mot so nguyen : ");
scanf("%ld", &number);
result = fibonaci(number);
printf("Fibonaci thu %ld la : %ld\n", number, result);
}
/* Định nghĩa đệ qui hàm fibonaci */
long fibonaci( long n){
if ( n = 0 || n = 1 )
        return n;
else
        return fibonaci(n-1) + fibonaci(n-2);
}
```

```
Hay nhap vao mot so nguyen : 0
Fibonaci thu 0 la : 0

Hay nhap vao mot so nguyen : 1
Fibonaci thu 1 la : 1
```

```
Hay nhap vao mot so nguyen : 2
Fibonaci thu 2 la : 1

Hay nhap vao mot so nguyen : 3
Fibonaci thu 3 la : 2

Hay nhap vao mot so nguyen : 4
Fibonaci thu 4 la : 3

Hay nhap vao mot so nguyen : 5
Fibonaci thu 5 la : 5

Hay nhap vao mot so nguyen : 6
Fibonaci thu 8 la : 8

Hay nhap vao mot so nguyen : 10
Fibonaci thu 10 la : 55

Hay nhap vao mot so nguyen : 20
Fibonaci thu 20 la : 6765

Hay nhap vao mot so nguyen : 35
Fibonaci thu 35 la : 9227465
```

Mỗi lần hàm `fibonaci()` được gọi, nó kiểm tra ngay lập tức xem liệu đây có phải là trường hợp cơ sở, n bằng 0 hay 1 hay không. Nếu đúng n được trả lại, bằng không, tức là khi n>1, nó sẽ tạo ra hai lời gọi đệ qui đơn giản hơn lời gọi đến hàm fibonaci ban đầu.

So sánh đệ qui và lặp

Ta đã xét cách tính giai thừa của một số, có thể tính theo phương pháp lặp (phi đệ qui) hoặc phương pháp đệ qui. Trong phần này ta sẽ so sánh hai phương pháp này và tìm hiểu tại sao trong thực tế thường chọn phương pháp lặp hơn.

Lập trình đệ qui sử dụng cấu trúc lựa chọn, còn lặp sử dụng cấu trúc lặp. Cả hai phương pháp đều liên quan đến quá trình lặp. Tuy nhiên, phương pháp lặp sử

dụng vòng lặp tường minh, còn phương pháp đệ qui có được quá trình lặp bằng cách sử dụng liên tục các lời gọi hàm. Cả hai phương pháp đều phải kiểm tra khi nào thì kết thúc. Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai, còn lời gọi đệ qui kết thúc khi gặp trường hợp cơ sở. Quá trình lặp thực hiện các tác động lên điều kiện lặp nhờ các lệnh trong thân câu lệnh lặp cho đến khi điều kiện lặp sai, còn lời gọi đệ qui làm cho các lời gọi hàm đơn giản dần cho đến khi đạt tới trường hợp cơ sở. Cả hai phương pháp đều có thể dẫn đến trường hợp vô hạn không dừng, quá trình lặp sẽ không thoát ra được khi điều kiện lặp không bao giờ sai, còn lời gọi đệ qui không thoát ra được khi các bước đệ qui không làm cho bài toán đơn giản hơn để cuối cùng đạt tới trường hợp cơ sở.

Tuy nhiên phương pháp đệ qui không hiệu quả do nó liên tục thực hiện các lời gọi hàm, do vậy tốn thời gian xử lý và không gian nhớ. Mỗi lần gọi hàm, lại cần thêm một bản sao của hàm, tốn thêm bộ nhớ (lưu các biến của hàm, địa chỉ trở về của hàm ...). Vì vậy phương pháp lặp có lẽ được ưa chuộng hơn do khả năng tiết kiệm. Tuy nhiên trong nhiều trường hợp người ta vẫn dùng phương pháp đệ qui vì nó cho phép diễn đạt được tương đối trực quan và dễ hiểu quá trình giải quyết bài toán.

Tuy vậy, đệ qui là một vấn đề lớn và phức tạp, những ai quan tâm đến vấn đề này nên tìm nhưng cuốn sách nói về đệ qui. Phần này chỉ nêu các vấn đề đơn giản của đệ qui, giúp người lập trình viết các hàm đệ qui dễ dàng.

Để kết thúc chương này, chúng tôi xin có một lời khuyên rằng các kỹ thuật trong *"công nghệ phần mềm"* là quan trọng nhưng *hiệu năng của chương trình* cũng quan trọng và thật không may hai mục tiêu đó lại thường cọc cạch, không đi đôi với nhau. Tuân theo các kỹ thuật trong công nghệ phần mềm sẽ làm cho nhiệm vụ phát triển các phần mềm lớn và phức tạp dễ dàng hơn, còn đạt hiệu năng cao là chìa khoá cho việc hệ thống được chấp nhận trong hiện tại và cả tương lai.

Công nghệ phần mềm
Module hoá chương trình thành các phần nhỏ, gọn hơn theo mô hình phân cấp nhằm đạt được các tiêu chuẩn về sản phẩm phần mềm tốt.
Hiệu năng
So sánh một chương trình phân quá nhỏ thành các hàm với chương trình chỉ có một khối, tuần tự từ trên xuống không có một hàm nào cả, ta thấy chương trình có nhiều lời gọi hàm sẽ tốn nhiều thời gian thực hiện hơn nhưng chương trình nguyên khối lại rất khó khăn trong việc lập trình, kiểm tra, gỡ rối, bảo trì và phát triển.

Vì vậy, khi mô đun hoá chương trình ta phải cân đối một cách hài hoà giữa hai mục tiêu là *Công nghệ phần mềm* và *Hiệu năng* của chương trình đạt được.

7. TÓM TẮT

7.1 Các lỗi thường gặp khi lập trình

Quên không gộp tệp tiêu đề của thư viện các hàm đã dùng vào đầu tệp.

Khai báo các tham số của hàm có cùng kiểu như float x, y sẽ làm cho trình biên dịch hiểu nhầm thành y có kiểu là int, bởi vì int là kiểu ngầm định (đúng phải là float x, float y).

Đặt dấu chấm phẩy sau dấu đóng ngoặc danh sách các tham số của phần định nghĩa hàm sẽ gây ra lỗi cú pháp.

Định nghĩa một hàm nằm trong một hàm khác.

Chuyển các dữ liệu cấp cao hơn (lưu tốn nhiều ô nhớ hơn) có thể dẫn đến thay đổi giá trị của dữ liệu.

Quên không viết hàm nguyên mẫu sẽ gây ra lỗi cú pháp khi kiểu trả về của hàm không phải là int và lời gọi hàm được đặt trước định nghĩa hàm, ngoài ra không có hàm nguyên mẫu có thể gây ra các lỗi nguy hiểm khi chạy.

Khi có các danh biểu cùng tên ở trong và ngoài một khối, lập trình viên muốn tham chiếu đến cái ở ngoài thì kết quả danh biểu ở trong khối lại được tham chiếu.

Đối với hàm đệ qui, quên xử lý trường hợp cơ sở hay các bước đệ qui không hội tụ dần về trường hợp cơ sở.

7.2 Các thói quen tốt khi lập trình

Cần hiểu rõ các hàm trong thư viện ANSI chuẩn.

Đặt các dòng trắng, các lời chú thích ở những nơi thích hợp làm cho chương trình dễ đọc hơn.

Mặc dù có thể bỏ qua kiểu trả về của hàm có kiểu trả về là int nhưng vẫn nên viết tường minh ra, tuy nhiên riêng đối với hàm main thì không cần thiết.

Ghi rõ các kiểu của tham số, kể cả trong trường hợp là kiểu ngầm định, kiểu int.

Nên dùng các danh biểu khác tên nhau, tránh nhầm lẫn không đáng có.

Chọn tên hàm, tên các tham số gợi nhớ, làm cho chương trình dễ đọc hơn, tránh phải viết nhiều chú thích không cần thiết.

Viết các hàm nguyên mẫu để sử dụng khả năng kiểm tra kiểu chặt chẽ của ANSI C.

Các biến chỉ dùng trong hàm nên đặt là biến cục bộ thì tốt hơn là đặt biến toàn cục.

7.3 Nâng cao mục tiêu "đảm bảo tiêu chuẩn phần mềm" tốt

Tránh viết lại các chức năng đã có, nếu có thể sử dụng các hàm trong thư viện chuẩn của ANSI C thay cho giải pháp viết mới.

Mỗi hàm chỉ thực hiện một công việc xác định, tên của hàm cần diễn đạt công việc này. Khi một hàm thực hiện nhiều công việc, chia nhỏ nó thành các hàm nhỏ hơn, các hàm không nên dài quá 1 trang.

Chương trình nên viết thành nhiều hàm nhỏ. Điều đó làm cho chương trình dễ viết, kiểm tra, gỡ rối, bảo trì và thay đổi, ngoài ra nó nâng cao khả năng dùng lại của các hàm con.

Hàm nguyên mẫu, định nghĩa hàm và các lời gọi hàm cần phải có cùng số lượng, kiểu và trình tự các tham số, phải cùng kiểu trả về.

Các hàm nguyên mẫu cần đặt trước định nghĩa và các lời gọi hàm.

8. BÀI TẬP

Bài tập 4.1 Viết lại các chương trình trong các chương 2, 3 dưới dạng chương trình con.

Bài tập 4.2 Cải tiến các chương trình `func3.c` và `func5.c` sao cho dãy số thao tác có chiều dài tuỳ ý do người dùng nhập từ bàn phím.

Bài tập 4.3 Viết dưới dạng hàm chương trình giải phương trình bậc nhất. Hàm thực hiện việc giải phương trình bậc nhất có ba tham số hình thức trong đó hai tham số đầu là các hệ số của phương trình, tham số thứ ba sử dụng để chứa nghiệm. Hàm trả về một số nguyên cho biết phương trình có nghiệm, vô nghiệm hay có vô số nghiệm.

Bài tập 4.4 Tương tự cho phương trình bậc hai.

Bài tập 4.5 Viết một chương trình bao gồm hai tệp chương trình nguồn. Tệp thứ nhất `sohoc.c` chứa bốn hàm

`int nt(int n)`	trả về 1 nếu n là số nguyên tố, trái lại trả về 0.
`int ht(int n)`	trả về 1 nếu n là số hoàn thiện, trái lại trả về 0.
`int bcnn(int a, int b)`	trả về giá trị là bội số chung nhỏ nhất của hai số nguyên a,b.
`int uscln(int a, int b)`	trả về ước số chung lớn nhất của hai tham số.

Tệp nguồn thứ hai `main.c` chứa hàm `main()` và ba hàm

`CacSoHoanthien(int n)`	Liệt kê các số hoàn thiện nhỏ hơn n.
`CacSoNguyenTo(int n)`	Liệt kê các số nguyên tố nhỏ hơn n.
`bcnn_dayso(int *)`	Trả về bộ số chung nhỏ nhất của một dãy số.

Bài tập 4.6 Viết dưới dạng đệ quy hàm daonguoc(int n) in ra các chữ số của n theo thứ tự đảo ngược.

Bài tập 4.7 Tương tự cho hàm tính số chữ số của một số nguyên.

Chương 5

CẤU TRÚC

Mục đích chương này

1) Biết cách khai báo các kiểu dữ liệu phức tạp từ các kiểu dữ liệu cơ sở: cấu trúc, hợp.
2) Biết cách khai báo các cấu trúc thành phần bits để lưu trữ dữ liệu gọn hơn.
3) Cách biểu diễn các kiểu danh sách liên kết nhờ cấu trúc tự trỏ.
4) Các thao tác trên danh sách liên kết.

1. CẤU TRÚC

1.1 Khái niệm và định nghĩa cấu trúc

Trong thực tế lập trình, chúng ta có thể sẽ cần đến những kiểu dữ liệu phức tạp hơn, được tạo thành từ những kiểu dữ liệu đơn giản mà đã biết. Những kiểu dữ liệu này cho ta một khả năng kết hợp một nhóm các biến để thể hiện một đối tượng chung. Chẳng hạn, để lưu giữ những thông tin liên quan đến một đối tượng nhân viên, ta có thể cần đến một biến nào đó có khả năng lưu trữ được cả tên, địa chỉ, ngày sinh lẫn mã số nhân viên, mức lương v.v... để có thể xử lý biến này như một phần tử thống nhất, thể hiện thông tin của một nhân viên cụ thể. Ngôn ngữ C cho phép chúng ta tự xây dựng những kiểu dữ liệu phức hợp như vậy và sử dụng những kiểu dữ liệu này để khai báo cho các biến sử dụng sau đó. Chúng ta gọi những kiểu dữ liệu như vậy là các cấu trúc.

Cấu trúc là một kiểu dữ liệu bao gồm nhiều thành phần có thể thuộc nhiều kiểu dữ liệu khác nhau. Các thành phần được truy nhập thông qua tên. Khái niệm cấu trúc trong C có nhiều nét tương tự như khái niệm về bản ghi (record) trong PASCAL hay trong FOXPRO.

1.2 Khai báo cấu trúc

Khai báo kiểu dữ liệu cấu trúc

```
struct <tên_cấu_trúc> {
        <khai_báo_các_thành_phần>
        };
```

trong đó **struct** là từ khóa đứng trước một khai báo cấu trúc, *<tên_cấu_trúc>* là một tên hợp lệ được dùng làm tên cấu trúc. *<khai báo các thành phần>* tựa như khai báo biến, bao gồm kiểu dữ liệu và một danh sách các khai báo tên của các thành phần tạo nên cấu trúc này. Thông thường người ta sử dụng thuật ngữ trường để chỉ các thành phần của cấu trúc. Nhưng cũng có một số tài liệu gọi chúng là các biến trong cấu trúc. Chúng ta xét các ví dụ sau:

Ví dụ

```
struct hoc_sinh {
        char ho_ten[20];
        float diem_thi;
        };
struct diem {
        float x,y;
        };
```

Khai báo thứ nhất xác định một kiểu cấu trúc có tên là `hoc_sinh` gồm hai thành phần: (i) Họ tên học sinh (`ho_ten`), (ii) Điểm thi (`diem_thi`). Khai báo thứ hai mô tả các toạ độ của một điểm trên mặt phẳng toạ độ thực. Dù rằng hai thành phần của cấu trúc giống hệt nhau và có thể sử dụng mảng để thay thế nhưng việc sử dụng cấu trúc trong trường hợp này làm cho thông tin được mô tả tường minh hơn.

Khai báo biến cấu trúc

struct *<tên_cấu_trúc> <tên_biến_cấu_trúc>*;

Ví dụ

```
struct hoc_sinh hs,dshs[100];
struct p,q,dsdiem[20];
```

Khai báo con trỏ cấu trúc

struct *<tên_cấu_trúc> *<tên_biến_cấu_trúc>*;

Ví dụ

```
struct diem *ptr_p;
```

Giống như mọi con trỏ khác, sau khai báo con trỏ cấu trúc `ptr_p` có giá trị NULL. Con trỏ này có thể nhận giá trị là địa chỉ của các biến cấu trúc kiểu `diem`. Chẳng hạn:

```
ptr_p = &p;
```

Khai báo đồng thời cấu trúc và biến cấu trúc

struct [*<tên_cấu_trúc >*]
{
<khai_báo_các_thành_phần>
}*<danh_sách_biến_cấu_trúc>*;

Trong cú pháp trên ta thấy rằng phần *<tên_cấu_trúc>* là tuỳ chọn. Tuy vậy, nếu không có *<tên_cấu_trúc>* thì cấu trúc khai báo tương ứng không được sử dụng về sau.

Ví dụ

```
struct dagiac {
  int n;
  struct diem dsdinh[20];
  } dg1,dg2;
```

Giống như PASCAL, ngôn ngữ C cho phép khai báo trực tiếp kiểu của các thành phần là biến cấu trúc bên trong một cấu trúc lớn hơn. Chẳng hạn nếu không định nghĩa riêng cấu trúc `diem`, ta có thể viết lại khai báo của cấu trúc `dagiac` như sau:

```
struct dagiac {
  int n;
  struct {
       float x,y;
       }dsdinh[20];
  }dg1,dg2;
```

1.3 Đặt tên kiểu dữ liệu bằng typedef

1.3.1 Từ khoá typedef

Ngôn ngữ C cho phép thêm tên mới cho một kiểu dữ liệu bằng câu lệnh như sau:

typedef *<tên_kiểu_đã_có>* *<tên_mới >;*

trong đó *<tên_kiểu_đã_có>* là kiểu dữ liệu mà ta muốn thêm tên mới. *<tên_mới>* là tên mới mà ta muốn đặt.

Xét câu lệnh sau

```
typedef unisgned char byte
```

Sau câu lệnh này `byte` được xem như là kiểu dữ liệu tương đương với **unsigned char** và có thể được sử dụng trong các khai báo biến như các kiểu dữ liệu khác. Xét chương trình sau:

Bài tập mẫu 5.1

```
/*struct1.c*/
#include <stdio.h>
typedef unsigned char byte
void main() {
        byte ch=12, ch1;
        int i;
        ch1=ch;
        for (i=0;i<5;i++)
                ch>=1;
printf("byte %X sau khi dich phai nam lan la %X",ch1,ch);
        getch();
        }
```

```
byte C sau khi dich phai nam lan la 0
```

typedef thường được sử dụng để định nghĩa các kiểu dữ liệu phức hợp thành một tên duy nhất để dễ dàng viết hơn trong khi viết.

Ví dụ

```
typedef  int * PTR_INT;
```

định nghĩa một kiểu dữ liệu con trỏ nguyên. Sau câu lệnh này để khai báo một biến con trỏ nguyên chúng ta chỉ cần viết:

```
PTR_INT  ptr_int;
```

1.3.2 Sử dụng typedef với cấu trúc

Đặc biệt đối với các kiểu dữ liệu cấu trúc, **typedef** cho phép đơn giản hóa cách viết khai báo. Xét ví dụ sau:

```
struct hoc_sinh{
            char ho_ten[20];
            float diem_thi;
            };
typedef struct hoc_sinh t_hoc_sinh;
typedef struct hoc_sinh *ptr_hoc_sinh;
```

với câu lệnh này tên mới của cấu trúc **struct** `hoc_sinh` sẽ là `t_hoc_sinh`. Đồng thời câu lệnh này còn định nghĩa một kiểu con trỏ cấu trúc có tên là `ptr_hoc_sinh`. Khi đó câu lệnh khai báo biến cấu trúc viết gọn lại như sau:

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

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

```
ptr_hoc_sinh ptrhs;
```

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

```
struct hoc_sinh *ptrhs.
```

Có thể sử dụng **typedef** ngay khi khai báo cấu trúc như trong ví dụ sau:

```
typedef struct {
            char ho_ten[20];
            float diem_thi;
            }t_hoc_sinh;
```

Trong khai báo trên cần chú ý hai điểm: thứ nhất, tên cấu trúc bây giờ có thể không cần, thứ hai các tên đi sau dấu "}" không phải là tên biến mà là tên khác

nhau của kiểu dữ liệu cấu trúc đang định nghĩa.

1.4 Thao tác trên biến cấu trúc

1.4.1 Truy nhập tới thành phần trong cấu trúc

Mỗi biến cấu trúc là một thể hiện của cấu trúc, nghĩa là nó mô tả cụ thể giá trị của các thành phần được khai báo bên trong cấu trúc. Chẳng hạn, biến cấu trúc p và q sẽ có các thành phần x và y khác nhau trong cấu trúc `diem`. Và lẽ dĩ nhiên có thể gán cho hai thành phần x và y của cấu biến cấu trúc p các giá trị khác nhau. Như vậy, để xác định một thành phần bên trong biến cấu trúc người ta cần đến cả tên thành phần và biến tương ứng. Nguyên tắc chung như sau:

```
<tên_biến_cấu_trúc>.<tên_thành_phần>
```

Dấu chấm "." ở đây được gọi là toán tử truy nhập thành phần cấu trúc. Nếu bản thân thành phần của biến cấu trúc lại là một biến cấu trúc thì vẫn áp dụng được toán tử này để xác định các thành phần "sâu hơn" trong các cấu trúc thành phần. Sau đây là ví dụ:

```
p.x = 5;
p.y = 10;
q.5 = 15;
dgl.dsdinh[2].x=100;
dgl.n = 10;
```

1.4.2 Truy nhập tới thành phần cấu trúc từ con trỏ cấu trúc

Với biến cấu trúc p để xác định các thành phần của nó ta sử dụng toán tử ".". Giả sử có con trỏ `ptr_p` chỉ đến p khi đó có hai cách để xác định p là bản thân tên biến p và áp dụng toán tử * lên `ptr_p` nghĩa là `*ptr_p`, như vậy để xác định thành phần x trong p có thể sử dụng ký pháp sau:

```
(*ptr_p).x
```

Tuy nhiên để giảm bớt sự rườm rà trong cách viết ngôn ngữ C đưa ra toán tử mới là "->", được kết hợp từ hai ký tự là dấu trừ "-" và dấu lớn hơn ">", sử dụng với các con trỏ cấu trúc. Theo cách này thành phần x của biến cấu trúc p do `ptr_p` trỏ tới xác định như sau:

```
ptr_p->x
```

1.4.3 Nhập dữ liệu cho biến cấu trúc

Ngôn ngữ C không đưa ra các phương tiện để nhập dữ liệu cho biến cấu trúc. Như vậy, người lập trình phải tự mình thực hiện các thao tác nhập dữ liệu cho các thành phần kiểu cơ sở trong biến cấu trúc. Theo kinh nghiệm, nếu áp dụng trực tiếp hàm `scanf()` lên trên các thành phần cấu trúc thì nhiều trường hợp sẽ xảy ra hiện tượng treo máy. Biện pháp được đề nghị sử dụng là sử dụng biến trung gian. Trước hết chúng ta hãy nhập các số liệu qua biến trung gian, sau đó mới đem gán các giá trị nhập được cho các thành phần cần nhập. Điều này tương tự như việc nhập dữ liệu cho các phần tử trong biến mảng. Đoạn chương trình sau đây trình bày cách nhập dữ liệu cho biến cấu trúc `p`.

```
...
float temp;
...
printf("Toa do x: ");
scanf("%f",&temp);
p.x = temp;
printf("Toa do y:");
scanf("%f",&temp);
p.y = temp;
...
```

Trong chương trình `struct2.c` sau đây có khai báo một mảng các cấu trúc `hoc_sinh` và thực hiện một số thao tác quen thuộc trên danh sách sinh viên:

(i) vào số liệu

(ii) in ra danh sách sinh viên phải thi lại.

Bài tập mẫu 5.2

```
/*struct2.c*/
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct {
        char ho_ten[20];
        float diem;
     }hoc_sinh;
```

```
void main(){
        int n; /*Số lượng học sinh của lớp đã được nhập*/
        int k; /*Số lượng học sinh thi lại*/
        int i;
        hoc_sinh dshs[100];
        char name[20];
        float d;
        /*Nhập thông tin các học sinh*/
        clrscr();
        n=0; /*Ban đầu chưa có học sinh nào*/
 do {
 printf("Nhap vao hoc sinh thu %d\n",n+1);
 printf(" Ho ten : ");
 fflush(stdin) ;/*đặt trước lệnh nhập xâu ký tự*/
 gets(name);
 if(strcmp(name,"")!=0) {
   strcpy(dshs[n].ho_ten,name);
   printf(" Diem : ");
   scanf("%f",&d);
   dshs[n++].diem=d;
  }
 } while (strcmp(name,"")!=0);
 if (!n) {
  /*Không nhập được học sinh nào !*/
   printf("Chua hhap hoc sinh nao\n");
   return ;
  }
 else {
  k=0;
 printf(" Danh sach sinh vien thi lai\n");
       for (i=0;i<n;i++)
```

```
        if(dshs[i].diem <5)
        printf("%d.%s %6.3f ",++k,dshs[i].ho_ten,dshs[i].diem);
        if (!k)
        printf("Khong co sinh vien thi lai\n");
        }
        getch();
        }
```

```
Nhap vao hoc sinh thu 1
 Ho ten : Le Dang Hung
 Diem : 9
Nhap vao hoc sinh thu 2
 Ho ten : Pham Huyen Linh
 Diem : 9
Nhap vao hoc sinh thu 3
 Ho ten : Phan Thanh Hong
 Diem : 8
Nhap vao hoc sinh thu 4
 Ho ten : Le Hai Ha
 Diem : 8
Nhap vao hoc sinh thu 5
 Ho ten : Tran Quynh Anh
 Diem : 6
Nhap vao hoc sinh thu 6
 Ho ten : Dao Hong Anh
 Diem : 4
Nhap vao hoc sinh thu 7
 Ho ten : Le Huong Lan
 Diem : 8
Nhap vao hoc sinh thu 8
 Ho ten : Dam Luu Ly
 Diem : 3
```

```
Nhap vao hoc sinh thu 9
 Ho ten : Bui Bich Phuong
 Diem : 2
Nhap vao hoc sinh thu 10
 Ho ten : Nguyen Minh Thuy
 Diem : 7
Nhap vao hoc sinh thu 11
 Ho ten :
 Danh sach sinh vien thi lai
1.Dao Hong Anh  4.000
2.Dam Luu Ly  3.000
3.Bui Bich Phuong  2.000
```

1.4.4 Phép gán giữa các biến cấu trúc

Có thể thực hiện phép gán nội dung một biến cấu trúc cho một biến cấu trúc khác cùng kiểu. Chẳng hạn nếu `p` và `q` cùng có kiểu `diem`, ta có thể thực hiện câu lệnh gán sau đây:

```
q = p;
```

Câu lệnh gán cấu trúc thực hiện gán giá trị các trường của cấu trúc bên vế phải cho các trường tương ứng trong biến cấu trúc bên vế trái. Chương trình `struct3.c` sau đây cụ thể hoá lời giải thích vừa rồi.

Bài tập mẫu 5.3

```
/*struct3.c*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
void main(){
  struct  {
    char ht[20];
    int  x,y;
    }a,b;
```

```
  clrscr();
  a.x=10;a.y=10;
  strcpy(a.ht,"hhsdhs");
  b=a;
  printf("a: {%s %d %d}\n",a.ht,a.x,a.y);
  printf("b: {%s %d %d}\n",b.ht,b.x,b.y);
  getch();
}
```

```
a: {hhsdhs 10 10}
b: {hhsdhs 10 10}
```

1.5 Truyền biến cấu trúc cho hàm

1.5.1 Truyền biến cấu trúc bằng tham trị

Việc khai báo tham số hình thức của hàm là một biến cấu trúc ảnh hưởng ít nhiều đến tốc độ thực hiện chương trình. Lý do là vì trong trường hợp này nội dung của biến cấu trúc dùng làm tham số thực sẽ được sao chép sang vùng nhớ dành cho tham số hình thức. Chúng ta biết rằng kích thước của các biến cấu trúc thường rất lớn. Vì vậy, chương trình sẽ chậm đi do phải tốn nhiều thời gian cho việc sao chép dữ liệu trong mỗi lời gọi hàm liên quan đến cấu trúc. Và hơn nữa nội dung của biến cấu trúc dùng làm tham số thực lại không thể thay đổi được.

Nhằm mục đích nâng cao tốc độ thực hiện chương trình và đôi khi cần thực hiện các thay đổi dữ liệu trên các cấu trúc thao tác, thay vì truyền nội dung cấu trúc người ta chỉ truyền cho hàm địa chỉ của cấu trúc.

1.5.2 Truyền biến cấu trúc bằng tham biến

Việc truyền địa chỉ biến cấu trúc cho hàm có hai lợi ích: thứ nhất, dù kích thước biến lớn nhường nào thì địa chỉ của biến vẫn chỉ là 4 bytes; thứ hai, ta có thể sử dụng toán tử "->" để thực hiện các truy xuất dữ liệu lên biến cấu trúc liên quan.

Chương trình `struct4.c` sau đây khai báo cấu trúc mô tả một tam giác dựa trên toạ độ ba đỉnh và thực hiện một số tính toán trên tam giác.

Bài tập mẫu 5.4

```
/*struct4.c*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```
typedef struct {
  float x,y;
  }diem;
typedef struct {
  diem A,B,C;
  }tamgiac;
/*cấu trúc tamgiac không quá lớn cho nên chỉ sử dụng con trỏ trong tham số hình thức khi cần thay đổi giá trị các thành
phần cấu trúc*/
void NhapToado(tamgiac *);
float canh(diem,diem);
float canhbp(diem ,diem );
int LoaiTG(tamgiac);
float dientich(tamgiac);
void main() {
 tamgiac tg;
 clrscr();
 printf("Nhap toa do cac dinh tam giac\n");
 NhapToado(&tg);
 printf("Canh BC: %f\n",canh(tg.B,tg.C));
 printf("Canh CA: %f\n",canh(tg.C,tg.A));
 printf("Canh AB: %f\n",canh(tg.A,tg.B));
 switch(LoaiTG(tg)) {
  case 1:
     printf("Tam giac deu\n");break;
  case 2:
     printf("Tam giac vuong can\n");break;
  case 3:
      printf("Tam giac can\n");break;
  case 4:
      printf("Tam giac vuong\n");break;
  case 5:
      printf("Tam giac thuong\n");break;
```

```
    }
  printf("Dien tich %f\n",dientich(tg));
  getch();
  }
/*nhập vào tọa độ các đỉnh một tam giác*/
void NhapToado(tamgiac *t) {
  float temp;
  printf("Dinh A\n");
  printf("x = ");
  scanf("%f",&temp);
  t->A.x=temp;
  printf("y = ");
  scanf("%f",&temp);
  t->A.y=temp;
  printf("Dinh B\n");
  printf("x = ");
  scanf("%f",&temp);
  t->B.x=temp;
  printf("y = ");
  scanf("%f",&temp);
  t->B.y=temp;
  printf("Dinh C\n");
  printf("x = ");
  scanf("%f",&temp);
  t->C.x=temp;
  printf("y = ");
  scanf("%f",&temp);
  t->C.y=temp;
  }
/*tính khoảng cách giữa hai điểm*/
float canh(diem p,diem q) {
  return(sqrt(pow(p.x-q.x,2.0)+
```

```
                pow(p.y-q.y,2.0)));
  }
/*khoảng cách giữa hai điểm được bình phương lên. Ta sẽ sử dụng thủ tục này trong việc xác định kiểu tam giác. Biện
pháp này tốt hơn việc sử dụng hàm canh(...) bạn đọc tự kiểm tra điều này*/
float canhbp(diem p,diem q) {
  return(pow(p.x-q.x,2.0)+
                pow(p.y-q.y,2.0));
  }
float dientich(tamgiac t) {
  float a,b,c,p;
  a=canh(t.B,t.C);
  b=canh(t.C,t.A);
  c=canh(t.A,t.B);
  p=(a+b+c)/2;
  return(sqrt(p*(p-a)*(p-b)*(p-c)));
  }
int LoaiTG(tamgiac t) {
  float a2,b2,c2;
  a2=canhbp(t.B,t.C);
  b2=canhbp(t.C,t.A);
  c2=canhbp(t.A,t.B);
  if(a2==b2||b2==c2||c2==a2)
    if(a2==b2&&b2==c2)
      return 1;
      else
       if (a2==b2+c2||b2==c2+a2||c2==a2+b2)
           return 2;
         else return 3;
  else if (a2==b2+c2||b2==c2+a2||c2==a2+b2)
        return 4;
       else return 5;
  }
```

```
Nhap toa do cac dinh tam giac
Dinh A
x = 1
y = 1
Dinh B
x = 10
y = 10
Dinh C
x = 1
y = 10
Canh BC: 9.000000
Canh CA: 9.000000
Canh AB: 12.727922
Tam giac vuong can
Dien tich 40.500000
```

2. CÁC CẤU TRÚC TỰ TRỎ

2.1 Nhắc lại khái niệm

Cấu trúc có ít nhất một thành phần là con trỏ chỉ đến bản thân cấu trúc được gọi là cấu trúc tự trỏ. Dưới đây đưa ra một số ví dụ về cấu trúc tự trỏ:

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

Khai báo này định nghĩa một cấu trúc tự trỏ có thể dùng để quản lý danh sách các học sinh. Việc sử dụng cấu trúc tự trỏ để quản lý danh sách tỏ ra linh hoạt hơn so với cách quản lý danh sách truyền thống sử dụng mảng tĩnh. Với cấu trúc tự trỏ, chúng ta có thể kéo dài danh sách ra vô hạn miễn là bộ nhớ máy tính có thể cấp phát được, đồng thời sẽ rất tiết kiệm bộ nhớ khi chỉ có một số lượng hạn chế đối tượng cần quản lý. Tất cả điều này có được nhờ các phương tiện cấp phát bộ nhớ.

Hạn chế của cách quản lý này là việc quản lý danh sách trở nên phức tạp hơn so với mảng. Việc duyệt danh sách phải tuân theo một số kiểu nhất định, phụ thuộc vào các con trỏ liên kết trong mỗi thành phần. Chẳng hạn, nếu mỗi phần tử chỉ có một mối nối khi đó ta chỉ có thể duyệt danh sách theo một chiều nhất định. Khi số

mối nối tăng lên, việc duyệt danh sách sẽ linh hoạt hơn nhưng phải trả giá cho việc cập nhật danh sách cũng như chi phí bộ nhớ để duy trì các liên kết. Sau đây là một ví dụ về cấu trúc tự trỏ để tạo các danh sách liên kết hai chiều hoặc cây nhị phân.

```
struct h_sinh{
        char ho_ten[20];
        float diem;
        struct h_sinh *truoc, *sau;
        };
```

Trong câu lệnh khai báo này, nhờ hai thành phần con trỏ trước và sau tương ứng chỉ địa chỉ cấu trúc chứa thông tin về các học sinh đứng liền trước và liền sau trong danh sách, chúng ta có thể dễ dàng duyệt danh sách theo cả hai chiều "tiến" hoặc "lui".

Trong các phần tiếp theo chúng ta sẽ bàn tới việc cài đặt các thao tác trên các danh sách móc nối quen thuộc như: ngăn xếp (stack), hàng đợi (queue), cây nhị phân (binary tree).

2.2 Ngăn xếp (stack)

Ngăn xếp là kiểu danh sách đặc biệt với các thao tác (chèn, xóa, thay đổi nội dung các phần tử) được thực hiện chỉ ở một đầu của danh sách.

Cấu trúc tự trỏ quản lý ngăn xếp chỉ có một thành phần tự trỏ (thành phần con trỏ chỉ đến bản thân cấu trúc). Cú pháp khai báo như sau:

```
typedef struct stack{
        <khai báo phần dữ liệu chứa trong mỗi phần tử>
        struct stack *prev;
        }t_stack;
```

Ngăn xếp được quản lý thông qua một biến trỏ xác định địa chỉ của phần tử cuối cùng đưa vào danh sách. Sau mỗi thao tác lấy ra một phần tử (POP) hay đưa thêm một phần tử (PUSH) ta cần phải cập nhật lại con trỏ đầu. Để làm ví dụ minh hoạ xét chương trình xác định dạng biểu diễn nhị phân của một số nguyên dương. Cấu trúc tự trỏ trong trường hợp này được sử dụng để mô tả danh sách các chữ số nhị phân của số. Định nghĩa của cấu trúc như sau:

```
typedef struct stack_binary{
   unsigned char d;
   struct stack_binary *prev;
}binary;
```

trong đó thành phần d chứa chữ số nhị phân, prev xác định địa chỉ của biến cấu trúc chứa chỉ số nhị phân hàng thấp hơn.

Các chỉ số nhị phân của số được chứa trong một danh sách xác định bởi con trỏ head.

```
binary *head;
```

Ban đầu head có giá trị NULL, sau đó cùng với quá trình lặp của thuật toán Ơcơlit danh sách head sẽ được kép dài thêm nhờ hàm.

```
binary *push(binary *,unsigned);
```

trong đó tham số thứ nhất là giá trị của head hiện tại, giá trị trả về của push() sẽ là địa chỉ của cấu trúc mới được thêm vào với nội dung là tham số thứ hai của push().

Để in ra số nhị phân tương ứng chúng ta lần lượt lấy ra các phần tử trong danh sách cấu trúc xác định bởi head, đồng thời giải phóng vùng nhớ động cấp phát cho chúng. Việc giải phóng bộ nhớ được thực hiện bởi hàm:

```
binary *pop(binary*);
```

Hàm pop() nhận một tham số là giá trị của head hiện tại, giải phóng vùng nhớ cấp phát cho head và trả về địa chỉ của phần tử vừa mới trở thành đỉnh mới của ngăn xếp.

Bài tập mẫu 5.5

```
/*stack.c*/
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
typedef struct stack_binary {
  unsigned char d;
  struct stack_binary *prev;
  } binary;
binary * push(binary*,unsigned char);
binary * pop(binary *);
void main() {
  unsigned int n;
```

```
  binary *head=NULL;
  clrscr();
  do {
  printf("\nNhap vao mot so nguyen khac 0: ");
  scanf("%d",&n);
  if (n==0) break;
  do {
    head=push(head,(unsigned char)(n%2));
    n/=2;
  }while(n!=0);
  printf("Dang bieu dien nhi phan\n");
  do {
   printf("%1d",head->d);
   head=pop(head);
  }while(head!=NULL);
  } while(1);
  }
binary *push(binary *head,unsigned char c){
  binary *p;
  p=(binary *)malloc(sizeof(binary));
  p->d=c;
  p->prev=head;
  return p;
  }
binary *pop(binary *head) {
  binary *p = head->prev;
  free(head);
  return p;
  }
```

```
Nhap vao mot so nguyen khac 0: 23
Dang bieu dien nhi phan
10111
```

```
Nhap vao mot so nguyen khac 0: 44
Dang bieu dien nhi phan
101100
Nhap vao mot so nguyen khac 0: 2334
Dang bieu dien nhi phan
100100011110
Nhap vao mot so nguyen khac 0: 433
Dang bieu dien nhi phan
110110001
Nhap vao mot so nguyen khac 0: 0
```

2.3 Hàng đợi (queue)

Hàng đợi là một kiểu danh sách trong đó việc thêm các phần tử mới diễn ra ở cuối của danh sách còn việc xoá các phần tử diễn ra ở đầu của danh sách. Với cách tổ chức như vậy, phần tử được lấy ra nếu có thời gian tồn tại trong danh sách lâu nhất so với các phần tử khác. Vì vậy, cấu trúc này còn có tên FIFO (First In First Out-Vào trước ra trước).

Cấu trúc tự trỏ mô tả hàng đợi có thể được khai báo giống như cấu trúc dùng cho ngăn xếp. Sự khác nhau giữa ngăn xếp và hàng đợi được thể hiện khi cài đặt cụ thể của các thao tác thêm, xoá, sửa.

```
struct queue{
        <Phần thông tin cần quản lý>
        <Phần liên kết>
        }t_queue;
```

Cấu trúc hàng đợi thường được dùng làm bộ đệm lưu giữ các công việc cần giải quyết theo trình tự, ví dụ, giải quyết danh sách xếp hàng của khách hàng. Yêu cầu là khách hàng nào xếp hàng trước được ưu tiên giải quyết trước. Một ví dụ khác rất quen thuộc với những người sử dụng máy nhắn tin để ghi lại các thông báo gửi cho mình. Thông báo đến trước sẽ được hiện lên đầu tiên v.v...

Chương trình `queue.c` sau đây mô tả cách sử dụng cấu trúc tự trỏ để mô phỏng việc xếp hàng mua vé tàu tại ga Hà nội. Việc xếp hàng được thực hiện theo nguyên tắc ai đến trước sẽ được mua vé trước.

Thông tin về khách hàng là số chứng minh thư và họ tên. Ta có cấu trúc thông tin như sau:

```
typedefstruct khachhang{
```

```
        long  cmt;
        char ho_ten[20];
        struct khachhang *truoc;
        }t_khachhang;
```

Việc quản lý danh sách các khách hàng đang mua vé được thực hiện thông qua hai biến trỏ dautien và cuoicung, chỉ đến hai biến cấu trúc mô tả người đầu tiên và người cuối cùng tương ứng:

```
t_khachhang *dautien,*cuoicung;
```

Có hai tác động lên danh sách khách hàng:

(i) Thêm một khách hàng vào danh sách. Thao tác này sẽ cập nhật lại cuoicung. Công việc tương ứng được thực hiện trong hàm:

```
t_khachhang *them_khach(t_khachhang*,long,char *);
```

Hàm them_khach(...) thực chất bổ sung thêm một cấu trúc tiếp theo sau cấu trúc xác định bởi cuoicung đồng thời cập nhật lại giá trị của cuoicung.

(ii) Gọi một khách hàng vào mua vé. Công việc này đồng nghĩa với việc loại khách hàng đầu tiên trong danh sách. Như vậy nó sẽ tác động đến dautien. Các công việc tương ứng sẽ được cài đặt trong hàm

```
t_khachhang *muave(t_khachhang *);
```

Hàm này sẽ xoá khỏi danh sách cấu trúc đang được trỏ bởi dautien và cho biến trỏ này chỉ đến phần tử tiếp theo.

Bài tập mẫu 5.6

```
/*queue.c*/
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <string.h>
typedef struct queue_khachhang {
  long cmt;
  char ho_ten[20];
  struct queue_khachhang *next;
```

```
  } t_khachhang;
t_khachhang * them_khach(t_khachhang*,long,char str[20]);
t_khachhang * muave(t_khachhang *);

void main() {
  unsigned int n;
  long temp_cmt;
  char ht[20];
  t_khachhang *dautien=NULL,*cuoicung=NULL;
  clrscr();
  n=0;
  do {
  printf("\nNhap them mot khach hang: \n");
  printf("So chung minh thu: ");
  scanf("%ld",&temp_cmt);
  if (temp_cmt==0) break;
  fflush(stdin);
  printf("Ten khach hang: ");
  gets(ht);
  if (dautien==NULL)
    dautien=cuoicung=them_khach(cuoicung,temp_cmt,ht);
  else cuoicung=them_khach(cuoicung,temp_cmt,ht);
  }while(1);
  /*Gọi khách hàng vào mua vé*/
  printf("Danh sach khach hang mua ve\n");
  n=0;
  do {
   printf("%2d",++n);
   printf("%8ld %20s\n",dautien->cmt,dautien->ho_ten);
   dautien=muave(dautien);
  }while(dautien!=NULL);
  getch();
```

```
  }
t_khachhang *them_khach(t_khachhang *cuoi, long c, char t[20]){
  t_khachhang *p;
  p=(t_khachhang *)
       malloc(sizeof(t_khachhang));
  p->cmt=c;
  strcpy(p->ho_ten,t);
  p->next=NULL;
  cuoi->next=p;
  return p;
  }
t_khachhang *muave(t_khachhang *dautien) {
  t_khachhang *d=dautien->next;
  free(dautien);
  return d;
  }
```

```
Nhap them mot khach hang:
So chung minh thu: 123456
Ten khach hang: To Huong Lan

Nhap them mot khach hang:
So chung minh thu: 234567
Ten khach hang: Pham Thuy Hang

Nhap them mot khach hang:
So chung minh thu: 345678
Ten khach hang: Ha Kieu Anh

Nhap them mot khach hang:
So chung minh thu: 0
Danh sach khach hang mua ve
```

```
1  123456          To Huong Lan
2  234567        Pham Thuy Hang
3  345678           Ha Kieu Anh
```

2.4 Cây nhị phân (Binary tree)

Một cách phi hình thức, cây nhị phân được định nghĩa như sau:

(i) Hoặc cây rỗng không chứa thành phần nào.

(ii) Hoặc cây được tạo thành bằng một gốc và hai cây con tương ứng là cây con trái và cây con phải.

Mỗi nút trong cây tương ứng với một đối tượng được quản lý bao gồm:

(i) Thông tin về bản thân đối tượng.

(ii) Thông tin xác định các nút bên trái và bên phải nếu có của nút đang xem xét. Thông thường các thông tin này là địa chỉ của các đối tượng. Như vậy để quản lý cây nhị phân ta phải sử dụng cấu trúc tự trỏ có hai thành phần liên kết:

```
typedef struct binary_tree{
        <phần mô tả thông tin về đối tượng>
        struct binary_tree *left,*right;
        }t_binary_tree;
```

Có ba giải thuật duyệt cây nhị phân là:

(i) Duyệt theo thứ tự trước: duyệt gốc trước, rồi duyệt cây con trái, cuối cùng duyệt cây con phải.

(ii) Duyệt theo thứ tự giữa: duyệt cây con trái trước, sau đó duyệt gốc; cây con phải được duyệt cuối cùng.

(iii) Duyệt theo thứ tự sau: đầu tiên duyệt cây con trái, cây con phải được duyệt tiếp sau; nút gốc được duyệt cuối cùng.

Cả ba giải thuật trên là đệ qui vì rằng các cây con trái, cây con phải vẫn có thể là một cây tuy nhiên với kích thước nhỏ hơn.

Ta xem xét cách cài đặt các thao tác này trên một lớp đặc biết các cây nhị phân tìm kiếm.

Trong cây nhị phân tìm kiếm, mỗi nút có một trường dữ liệu đặc biệt được gọi là khóa đặc trưng của nút. Giá trị của khóa phải có kiểu dữ liệu sao cho có định nghĩa một quan hệ "thứ tự" giữa các giá trị. Thí dụ, kiểu dữ liệu có thể định nghĩa một quan hệ "thứ tự " là tập các số nguyên với quan hệ ">" hoặc tập các xâu ký tự

cũng với quan hệ ">". Trong cây nhị phân tìm kiếm thông thường ta qui định khóa của nút gốc bé hơn khóa của nút con phải và lớn hơn khóa của nút con trái.

Các thao tác trên cây nhị phân tìm kiếm bao gồm:

(i) *Thêm một nút mới vào cây nhị phân.* Nút mới này phải có khoá khác với tất cả các nút đã có trong cây. Việc thêm nút đồng nghĩa với việc xác định vị trí trong cây của nút mới.

(ii) *Loại bỏ một nút trong cây.* Thao tác này bắt đầu bao gồm xác định nút cần xoá, rồi sau đó tuỳ theo đặc điểm của nút đó để thực hiện thao tác tiếp theo. Ta phân biệt ba trường hợp:

(ii.1) Nếu nút cần xoá là nút lá, mọi chuyện dễ dàng.

(ii.2) Trong trường hợp nút cần xoá chỉ có một nút con, việc xoá cũng không có vấn đề gì lớn.

(ii.3) Vấn đề chỉ thực sự phức tạp khi ta phải xoá một nút có đầy đủ con trái và con phải. Trong trường hợp này vai trò của nút cần được chuyển sang cho nút con trái có khoá lớn nhất.

Chương trình `tree.c` sau đây cài đặt các thao tác trên cây tìm kiếm có khoá là các số nguyên dương.

Cấu trúc để biểu diễn một nút của cây nhị phân tìm kiếm như sau:

```
typedef struct tree{
            unsigned int value;
            struct tree *left,*right;
      }t_tree;
```

Để quản lý cây nhị phân ta cần đến một biến trỏ xác định địa chỉ của nút gốc trong cây.

```
t_tree root;
```

Bài tập mẫu 5.7

```
/*tree.c*/
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
/*cấu trúc mô tả cây nhị phân tìm kiếm*/
typedef struct tree {
```

```
  unsigned int n;
  struct tree *left,*right;
  }t_tree;
/*Hàm tạo nút gốc*/
t_tree *Tao_cay(unsigned int );
/*Giải phóng các vùng nhớ cấp phát cho cây */
t_tree *Xoa_cay(t_tree *);
/*Thêm nút mới vào cây*/
void Them_nut(t_tree *,unsigned int);
/*Loại khỏi cây một nút*/
t_tree *Xoa_nut(t_tree*,unsigned int);
/*Duyệt cây theo thứ tự giữa*/
void infix(t_tree *);
/*Xác định nút cha hoặc có tiềm năng làm nút cha của một nút có khoá là một giá trị nào
đó*/
t_tree *Nutcha(t_tree*,unsigned int,int *);
/*Tìm nút với khoá nào đó*/
t_tree *Nut(t_tree *,unsigned int);
/*Xác định nút con lớn nhất trong nhánh bên trái*/
t_tree *Con_trai_lon_nhat(t_tree *);
void main() {
  unsigned int n;
  t_tree *root=NULL;
  clrscr();
  /*tạo cây*/
  do {
    printf("Nhap vao mot so nguyen: ");
    scanf("%d",&n);
    if (n==0) break;
    /*Nếu là nút đầu tiên trong cây thì phải tạo nó*/
    if (root==NULL)
      root = Tao_cay(n);
    /*Trái lại ta thêm một nút mới vào cây*/
    else Them_nut(root,n);
```

```
    } while(1);
  printf("Day so vua nhap duoc sap xep tang dan\n");
  infix(root);
  do {
  printf("Nhap vao mot so nguyen can loai bo: ");
  scanf("%d",&n);
  if (n==0) break;
  root=Xoa_nut(root,n);
  printf("Day so sau khi xoa\n");
  infix(root);
  }while(1);
  root=Xoa_cay(root);
  }

t_tree *Tao_cay(unsigned int n) {
  t_tree *p;
  p = (t_tree *)malloc(sizeof(t_tree));
  p->left=NULL;
  p->right=NULL;
  p->n=n;
  return p;
  }

void Them_nut(t_tree *r,unsigned int n) {
/*Việc thêm nút chỉ diễn ra khi nút có khoá tương ứng chưa có trong cây*/
  t_tree *p,*q;
  int k;
  q=(t_tree *)malloc(sizeof(t_tree));
  q->left=NULL;
  q->right=NULL;
  q->n=n;
  p = Nutcha(r,n,&k);
  switch(k) {
    case 1: if (p->left!=NULL)
```

```
            printf("Nut da co trong cay\n");
          else p->left=q;
          break;
    case 2: if(p->right!=NULL)
            printf("Nut da co trong cay");
          else p->right=q;
          break;
    }
  }
t_tree *Xoa_nut(t_tree *r,unsigned int n) {
/*Để xoá một nút người ta phân biệt ba trường hợp*/
  t_tree *p,*q,*s,*t;
  int k;
  p = Nut(r,n);
  if (p!=NULL) {
/*Nếu nút cần xoá là nút lá, chỉ việc xoá nút đó và cập nhật các liên kết trong nút cha của
nó */
    if (p->left==NULL && p->right==NULL) {
      if (p==r) {
        free(p);
        return NULL;
        }
      q=Nutcha(r,n,&k);
      switch (k) {
        case 1: q->left=NULL;free(p);break;
        case 2: q->right=NULL;free(p);break;
        }
      return r;
      }
/*  Nếu nút chỉ có một nút con (bên trái hoặc bên phải) thay thế nó bởi con của nó trong
cây. */
     if (p->left==NULL) {
       if (p==r) {
         q=p->right;
```

```
        free(p);
        return q;
        }
      q=Nutcha(r,n,&k);
      switch(k) {
        case 1:q->left=p->right;
                free(p);
                break;
        case 2:q->right=p->right;
                free(p);
                break;
        }
      return r;
      }
    if (p->right==NULL) {
      if (p==r) {
        q=p->left;
        free(p);
        return q;
        }
      q=Nutcha(r,n,&k);
      switch(k) {
        case 1:q->left=p->left;
                free(p);
                break;
        case 2:q->right=p->left;
                free(p);
                break;
        }
      return r;
      }
/*Trường hợp khó nhất xảy ra khi nút cần xoá có hai nút con.
 Chúng ta phải tìm cho được nút con trong nhánh bên trái
```

```
có giá trị lớn nhất và thay thế nút bị xoá bởi nút này  trong cây. */
    s=Con_trai_lon_nhat(p);
    t=Nutcha(p,s->n,&k);
    /*Cập nhật lại các liên kết của bố s*/
    if(t!=p) {
      t->right=s->left;
      s->left=p->left;
      s->right=p->right;
      }
     else {
      s->right=p->right;
      }
    /*s bây giờ đóng vai trò của p */
    if ((q=Nutcha(r,n,&k))!=NULL) {
      switch(k) {
       case 1:q->left=s;break;
       case 2:q->right=s;break;
       }
      }
     if (p==r) {
      free(p);
      return s;
      }
      else {
       free(p);
       return r;
       }
    }
   else {
     printf("Khong tim thay nut tuong ung\n");
     return r;
     }
 }
```

```
void infix(t_tree *r){
  if (r==NULL) return;
  infix(r->left);
  printf("%4u ",r->n);
  infix(r->right);
  }
t_tree *Nutcha(t_tree *r,unsigned int n, int *k){
  if (r->n==n) return NULL;
  if (r->n>n)
    if (r->left->n==n || r->left==NULL) {
      *k = 1;
      return r;
      }
    else return Nutcha(r->left,n,k);
  else
    if (r->right->n==n || r->right==NULL) {
      *k=2;
      return r;
      }
    else return Nutcha(r->right,n,k);
  }
t_tree *Nut(t_tree *r,unsigned int n){
  if (r==NULL)
    return NULL;
  if(r->n==n)
    return r;
  if (r->n>n)
    return Nut(r->left,n);
  else return Nut(r->right,n);
  }
t_tree *Con_trai_lon_nhat(t_tree *r){
  t_tree *p;
  p=r->left;
  while(p->right!=NULL) p=p->right;
```

```
  return p;
  }
t_tree *Xoa_cay(t_tree *r) {
   if (r->left!=NULL)
     Xoa_cay(r->left);
   if(r->right!=NULL)
     Xoa_cay(r->right);
   free(r);
   return NULL;
  }
```

```
Nhap vao mot so nguyen: 10
Nhap vao mot so nguyen: 4
Nhap vao mot so nguyen: 16
Nhap vao mot so nguyen: 1
Nhap vao mot so nguyen: 8
Nhap vao mot so nguyen: 3
Nhap vao mot so nguyen: 2
Nhap vao mot so nguyen: 5
Nhap vao mot so nguyen: 9
Nhap vao mot so nguyen: 12
Nhap vao mot so nguyen: 19
Nhap vao mot so nguyen: 14
Nhap vao mot so nguyen: 17
Nhap vao mot so nguyen: 0
Day so vua nhap duoc sap xep tang dan
   1    2    3    4    5    8    9   10   12   14   16   17
19 Nhap vao mot so
 nguyen can loai bo: 1
Day so sau khi xoa
   2    3    4    5    8    9   10   12   14   16   17   19
Nhap vao mot so nguy
en can loai bo: 4
Day so sau khi xoa
```

```
  2    3    5    8    9   10   12   14   16   17   19 Nhap
vao mot so nguyen ca
n loai bo: 0
```

3. KIỂU HỢP (UNION)

3.1 Đặt vấn đề

Một biến kiểu union cũng bao gồm nhiều thành phần giống như một biến cấu trúc, nhưng khác biến cấu trúc ở chỗ: các trường trong biến cấu trúc được cấp phát các vùng nhớ khác nhau, còn các trường của biến union được cấp phát chung một vùng nhớ. Như vậy union cho phép ta sử dụng một vùng nhớ cho nhiều biến khác nhau. Độ dài của vùng nhớ đủ để chứa được thành phần dài nhất trong union. Trong phần tiếp sau chúng ta bàn tới cách khai báo và sử dụng **union**.

3.2 Khai báo kiểu dữ liệu union

Khai báo một union tương tự như khai báo cấu trúc, thay cho từ khoá **struct** ta dùng từ khoá **union**. Việc định nghĩa một biến kiểu union, mảng các union, con trỏ union cũng như cách thức truy nhập đến các thành phần của biến union được thực hiện hoàn toàn tương tự như đối với các cấu trúc. Một cấu trúc có thể có thành phần union và ngược lại các thành phần của union lại có thể là cấu trúc. Ta xét ví dụ khai báo sau:

```
typedef union{
        unsigned int n;
        unsigned char ch[2];
        }val;
val a,b,x[10];
```

Câu lệnh trong ví dụ trên định nghĩa kiểu **union** val gồm hai thành phần là `n` và mảng ký tự `ch`. Độ dài của `val` bằng độ dài của trường `n` và bằng 2. Tiếp đó khai báo các biến `union a, b` và mảng các union `x`.

Phép gán

```
a.n=0x1b1a;
```

một số hệ 16 là `0xa1b2` cho thành phần `ival` của `a`. Do ival chỉ chiếm hai byte đầu của union, nên sau câu lệnh trên ta có:

`a.ch[0]=0x1a`

và

```
a.ch[1]=0x1b
```

như vậy ta đã dùng khai báo union để tách ra byte cao và thấp của một số nguyên. Hãy xem chương trình ví dụ sau:

Bài tập mẫu 5.8

```
/*union.c*/
#include<stdio.h>
#include<conio.h>
typedef union {
  unsigned int n;
  char ch[2];
  }u;
void main() {
 u a;
 clrscr();
 printf("a.n = 0x1b1a;\n");
 a.n = 0x1b1a;
 printf("Kich thuoc union %d\n",sizeof(a));
 printf("a.ch[0]=%x\n",a.ch[0]);
 printf("a.ch[1]=%x\n",a.ch[1]);
 getch();
 }
```

```
a.n = 0x1b1a;
Kich thuoc union 2
a.ch[0]=1a
a.ch[1]=1b
```

4. GHI NHỚ

4.1 Các lỗi hay gặp trong khi lập trình

Quên dấu chấm phẩy kết thúc câu lệnh khai báo một cấu trúc.

Gán một biến cấu trúc cho một biến khác kiểu.

So sánh hai cấu trúc.

Chèn khoảng trắng giữa - và > trong phép toán truy nhập thành phần của con trỏ cấu trúc.

Truy nhập đến các thành phần của một biến cấu trúc bằng tên của chúng mà quên mất tên biến.

Quên mô tả chỉ số khi truy nhập đến các thành phần của một thành phần mảng các cấu trúc.

4.2 Các thói quen lập trình tốt cần học tập

Khi định nghĩa một cấu trúc nên khai báo tên cho cấu trúc để có thể sử dụng lại về sau.

Đặt tên phải gợi nhớ, tức là tên phải mô tả được nội dung bên trong của cấu trúc.

Không chèn thêm dấu trắng xung quanh các toán tử -> hoặc .

5. BÀI TẬP

Bài tập 5.1 Thực hiện các công việc sau:

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 và hóa (điểm thập phân).

2. Đọc số liệu từ một phiếu điểm cụ thể và lưu trữ vào các thành phần của cấu trúc nói trên (lưu ý không nên sử dụng toán tử & truy nhập trực tiếp vào các thành phần của cấu trúc).

Bài tập 5.2 Xây dựng một mảng cấu trúc, mỗi thành phần của nó có kiểu như trong bài 5.1.

Nhập số liệu cho hai 20 phiếu điểm và lưu trữ vào mảng cấu trúc nói trên.

Nhập vào điểm chuẩn thi đại học và in ra danh sách các thí sinh trượt (có tổng ba môn bé hơn điểm chuẩn). Thông tin về mỗi thí sinh được in ra trên một dòng, bao gồm:

Họ tên

Quê quán

Số báo danh

Điểm thi các môn.

Bài tập 5.3 Tổng quát hóa bài tập trên để quản lý các đa giác. Xây dựng một cấu trúc dữ liệu để mô tả đa giác.

Tính diện tích của đa giác.

Bài tập 5.4 Trong chương trình ở bài tập mẫu 5.2, nếu chúng ta sử dụng cấu trúc:

```
struct hoc_sinh{
   char ho_ten[20];
   float diem;
   struct *tiep, *truoc;
   };
```

trong đó thành phần con trỏ `truoc` chỉ đến địa chỉ của đối tượng chứa các thông tin của học sinh đứng trước, cần phải sửa đổi chương trình như thế nào cho phù hợp.

Bài tập 5.5 Mô phỏng cách quản lý hàng đợi bằng danh sách móc nối hai chiều.

Bài tập 5.6 Mô phỏng cách quản lý một dãy số nguyên đơn điệu tăng bằng danh sách móc nối. Các thao tác cần thực hiện bao gồm:

Khởi tạo dãy số.

Thêm một số nguyên mới vào trong dãy số.

Xoá một số nguyên khỏi danh sách.

Bài tập 5.7 Viết lại các hàm đệ quy trong chương trình `tree.c`.

Chương 6

TỆP VÀ THAO TÁC VÀO/RA

Mục đích chương này

1) Biết các thao tác xử lý tệp như tạo tệp, đọc, ghi tệp ..., các thủ tục vào ra tệp mức thấp cũng như các thủ tục vào/ra theo dòng.
2) Hiểu xử lý tệp truy nhập tuần tự.
3) Hiểu xử lý tệp truy nhập ngẫu nhiên.

1. GIỚI THIỆU

Việc lưu dữ liệu ở bộ nhớ trong (các biến, mảng ...) chỉ là tạm thời, tất cả các dữ liệu như vậy sẽ bị mất khi chương trình kết thúc. Các tệp dữ liệu được dùng để lưu dữ liệu và chương trình để dùng tiếp cho nhiều lần sau trên các thiết bị lưu trữ như đĩa cứng, đĩa mềm, băng từ ... Trong chương này chúng ta sẽ tìm hiểu các thao tác xử lý tệp, cả với tệp truy nhập tuần tự cũng như tệp truy nhập ngẫu nhiên.

2. CÁC KHÁI NIỆM VỀ TỆP

Các khái niệm về hệ thống tệp và các cơ chế dùng để truy nhập tệp là các khái niệm chính trong vấn đề xử lý vào ra.

Hệ thống các tệp mô tả cách tổ chức lưu trữ thông tin trên thiết bị lưu trữ. Trong hệ điều hành MS-DOS trên mỗi đĩa cứng hay đĩa mềm người dùng hệ thống phân cấp các tệp, nó giống như một cấu trúc hình cây trong đó có gốc là thư mục gốc tương ứng với toàn bộ ổ đĩa cứng hay ổ đĩa mềm. Tiếp dưới là các tệp hoặc các thư mục con, mỗi một thư mục có thể có các tệp và các thư mục khác nằm dưới nó.

2.1 Đường dẫn để truy nhập các tệp

Các tệp trong MS-DOS được xác định bởi đường dẫn của nó với 4 thành phần: Tên ổ đĩa, tên các thư mục, tên tệp và phần mở rộng. Tên ổ đĩa gồm một chữ cái chỉ tên ổ đĩa theo sau là dấu hai chấm, xác định xem tệp nằm trên ổ đĩa nào. Tên thư mục bắt đầu bằng thư mục gốc (thể hiện bằng dấu "\"), tiếp theo là các thư mục con của nó được ngăn cách với tên thư mục trước đó bằng dấu "\", tên thư mục cuối cùng có dấu "\" và tiếp theo là tên tệp (tối đa 8 kí tự) và phần mở rộng (tối đa 3 kí tự).

2.2 Thẻ tệp (File handle)

Đường dẫn của tệp là một cách để xác định tệp đó. Tuy nhiên, còn có một

cách khác để mô tả tệp. Khi tạo hay mở tệp, hệ điều hành trả lại một số nguyên gọi là thẻ tệp. Hệ điều hành dùng thẻ tệp để truy nhập đến một cấu trúc lưu trữ các thông tin cần thiết cho các thao tác với tệp.

2.3 Các thông tin về tệp

Trong quá trình thao tác với một tệp đã mở, cần có một số thông tin quan trọng như các cờ thuộc tính tệp, kích thước và trạng thái tệp, chế độ truyền.

- *Các thuộc tính của tệp đã mở* : Bình thường, có thể vừa ghi vừa đọc tệp. Tuy nhiên trong một số trường khi tệp chứa các thông tin quan trọng, không cho phép người khác ghi đè thông tin lên, khi đó tệp sẽ mang thuộc tính chỉ đọc. Trong mỗi hệ điều hành đều cung cấp các quyền nhất định khi thao tác với tệp. Khi mở tệp, có thể cho quyền đọc hay ghi. Các hằng `S_IREAD` và `S_IWRITE` tương ứng với các quyền đọc và ghi. Nếu cho phép cả thao tác ghi và đọc thì sử dụng `S_IREAD | S_IWRITE`. Tuy nhiên trong MSDOS, tất cả các tệp đều ngầm định là đọc được, vì vậy đặt quyền `S_IWRITE` tương đương với `S_IREAD | S_IWRITE`. Hàm `access()` cho phép kiểm tra các quyền này.
- *Kích thước và trạng thái tệp* : Kích thước hay chiều dài của tệp là số byte của tệp. Hàm `long filelength(int handle)` trả lại kích thước của tệp. Kích thước của tệp đã mở có thể thay đổi bằng hàm `int chsize( int handle, long size)`. Khi tệp chưa mở, kích thước tệp có thể tìm thấy từ trạng thái của tệp bằng hàm `int stat( char* path, struct stat statbuf)`. Tương tự, hàm *int* `fstat( int handle, struct stat statbuf)` trả lại trạng thái của tệp đang mở.
- *Chế độ truyền* : Một tệp mở ra có thể theo hai chế độ, kiểu nhị phân và kiểu văn bản. Trong chế độ nhị phân, các byte được xử lý đúng như giá trị trên thiết bị ngoại vi, không phải thực hiện một sự chuyển đổi nào. Trong chế độ văn bản, các kí tự được tổ chức thành từng dòng, mỗi dòng được kết thúc bằng kí tự xuống dòng (`'\n'`). Khi đó có thể xảy ra sự chuyển đổi của một số ký tự theo yêu cầu của môi trường. Ví dụ, cặp kí tự xuống dòng (`'\r'`) và kí tự về đầu dòng (`'\n'`) được chuyển thành một kí tự (`'\n'`). Kí tự Control-Z là kí tự báo hiệu kết thúc tệp, mọi kí tự sau kí tự này đều không được đọc.

2.4 Các kiểu vào ra với tệp

Trong ngôn ngữ lập trình C không có các thao tác vào ra. Các hàm xử lý vào ra là của các thư viện trình biên dịch C. Các trình biên dịch C tuân theo chuẩn ANSI có hai nhóm hàm vào ra với tệp : các hàm vào ra theo dòng qua vùng đệm (Buffered Stream I/O) và các hàm vào ra tệp mức thấp, không qua vùng đệm (Unbuffered Low-Level I/O). Các thủ tục vào ra qua vùng đệm coi tệp như dòng

các byte, có sử dụng bộ đệm, một vùng nhớ trung gian do bản thân chương trình cung cấp (thực ra trình biên dịch tự động thực hiện giúp) cho dòng các byte đọc từ tệp lên hoặc ghi xuống tệp. Các thủ tục vào ra mức thấp thực hiện các thao tác này một cách trực tiếp, không sử dụng vùng đệm chương trình dịch (tất nhiên, các vùng đệm của hệ điều hành vẫn được sử dụng).

3. KÊNH NHẬP XUẤT VÀ CÁC HÀM THAO TÁC VÀO RA TỆP QUA VÙNG ĐỆM

3.1 Khái niệm

Khái niệm kênh xuất nhập (stream) là khái niệm quan trọng trong hệ thống vào ra qua vùng đệm (gọi tắt là vào ra mức II). Hệ thống vào ra mức II của C cung cấp cho các lập trình viên một phương tiện trao đổi thông tin độc lập với các thiết bị truy xuất, nghĩa là C cung cấp các kênh trao đổi thông tin giữa chương trình và các thiết bị được sử dụng. Hệ thống vào ra mức II được thiết kế để làm việc với khá nhiều thiết bị khác nhau. Dù rằng mỗi thiết bị có cấu tạo và cách thức hoạt động khác nhau, nhưng từ mỗi thiết bị vật lý này, hệ thống vào ra mức II tạo ra một thiết bị logic tương ứng gọi là kênh xuất nhập

Tên hàm	Chức năng
`fopen()`	Mở một kênh xuất nhập
`fclose()`	Đóng một kênh xuất nhập
`putc()`	Ghi một kí tự lên một kênh xuất nhập
`getc()`	Đọc một kí tự vào từ một kênh xuất nhập
`fseek()`	Tìm đến một byte được chỉ định trong kênh xuất nhập
`fprintf()`	Giống như `printf()` nhưng in ra một kênh xuất nhập
`fscanf()`	Giống như `scanf()` nhưng nhập vào từ một kênh xuất nhập
`fflush()`	Ghi ra tệp nội dung vùng đệm tương ứng với kênh xuất nhập
`feof()`	Trả về giá trị khác 0 nếu vị trí hiện thời trong kênh xuất nhập là cuối tệp
`ferror()`	Trả về giá trị khác 0 nếu có lỗi xuất hiện trong thao tác trên kênh xuất nhập
`rewind()`	Khởi tạo lại vị trí con trỏ về đầu tệp tương ứng với một kênh xuất nhập

Bảng 6.1 Các hàm vào ra mức II phổ biến nhất

Tất cả các kênh xuất nhập này được xử lý giống nhau. Bởi vì các kênh xuất nhập được thiết kế độc lập với phần cứng nên các hàm dùng để ghi dữ liệu lên tệp trên đĩa cũng có thể dùng để ghi lên màn hình.

Hệ thống vào ra dữ liệu mức II có những hàm phổ dụng được liệt kê trong bảng 6.1. Để sử dụng các hàm này, bạn cần gộp tệp tiêu đề `stdio.h` vào trong chương trình.

3.2 Con trỏ tệp (File Pointer)

Một con trỏ tệp là một con trỏ kiểu **FILE**, chỉ đến vùng nhớ chứa các thông tin liên quan đến một kênh xuất nhập như tên tệp tương ứng, tình trạng và vị trí đang làm việc.

Mode	Ý nghĩa
"r"	Mở một tệp chỉ cho thao tác đọc, fopen sẽ trả lại lỗi nếu tệp không tồn tại
"w"	Mở một tệp mới để ghi, nếu tệp đã tồn tại, nội dung của nó sẽ bị xoá hết.
"a"	Mở một tệp để ghi thêm dữ liệu vào cuối tệp, tệp mới sẽ được tạo nếu tệp đó chưa tồn tại
"r+"	Mở một tệp cho cả thao tác đọc và ghi, fopen sẽ trả lại lỗi nếu tệp không tồn tại
"w+"	Mở một tệp mới để đọc và ghi, nếu tệp đã tồn tại, nội dung của nó sẽ bị xoá hết.
"a+"	Mở một tệp để ghi thêm dữ liệu vào cuối tệp, tệp mới sẽ được tạo nếu tệp đó chưa tồn tại
"b"	Mở tệp theo kiểu nhị phân
"t"	Mở tệp theo kiểu văn bản

Bảng 6.2 Các chế độ truy nhập khi mở các tệp cho thao tác vào ra qua vùng đệm

3.3 Một số hàm tiêu biểu

- *Hàm* `fopen()`

Hàm `fopen()` thực hiện mở một kênh xuất nhập và kết nối một tệp với kênh xuất nhập đó. Hàm nguyên mẫu của nó như sau :

```
FILE*fopen(const char *filename, const char *mode);
```

với `filename` là một xâu ký tự bao gồm tên tệp hợp lệ với hệ điều hành và đường dẫn thư mục tới nó, `mode` là mô tả trạng thái làm việc của tệp yêu cầu sau khi mở. Bảng 6.2 mô tả các giá trị hợp lệ của `mode`.

Hàm trả lại `NULL` nếu thao tác mở thất bại, ngược lại con trỏ tệp sẽ được trả lại.

Ví dụ

Đoạn chương trình mở tệp `readme.txt` để đọc theo kiểu văn bản, có thể viết như sau :

```
if (( fp = fopen("readme.txt", "rt") == NULL){
      puts("Không mở được tệp readme.txt");
      exit(1);            /* Đóng các tệp và trở về OS */
      }
```

Cần lưu ý là khi mở một tệp để thực hiện các thao tác đọc ghi, cần phải thực hiện ghi các thông tin trong vùng đệm ra đĩa giữa các lần đọc ghi. Chẳng hạn đang thực hiện các thao tác ghi, muốn chuyển sang thao tác đọc thì cần phải ghi nội dung vùng đệm lên đĩa trước khi thực hiện thao tác đọc. Công việc đó, được thực hiện bằng cách gọi hàm `fflush(fp)` trong đó `fp` là con trỏ tệp chỉ tới tệp đang được truy xuất. Tuy nhiên, nếu trong các giai đoạn chuyển tiếp này, có lời gọi tới các hàm `fseek()` hay `rewind()` thì có thể không cần gọi hàm `fflush()`.

- *Hàm* `putc()`

Hàm `putc()` ghi một kí tự lên một kênh xuất nhập mà đã mở trước đó, được khai báo như sau :

```
int putc( int ch, FILE *fp);
```

ở đây `fp` là con trỏ tệp trả về trong thao tác mở tệp trước đây và `ch` là kí tự cần ghi ra. Con trỏ `fp` báo cho `putc()` biết phải ghi lên tệp nào trên đĩa. Ký tự `ch` được truyền vào dưới dạng **int**, nhưng máy tính chỉ sử dụng byte đầu (tức là byte thực sự chứa ký tự) để ghi lên kênh xuất nhập.

Nếu thao tác thành công, `putc()` trả lại kí tự vừa ghi được, nếu gặp lỗi `putc()` sẽ trả lại EOF.

- *Hàm* `getc()`

Hàm `getc()` ngược lại với hàm `putc()` đọc một kí tự tiếp theo từ kênh xuất nhập đã mở trước đó. Con trỏ vị trí đang thao tác trên tệp tự động tăng thêm 1, trỏ đến kí tự tiếp theo. Hàm này được khai báo như sau :

```
int getc(FILE *fp);
```

ở đây `fp` là con trỏ tệp trả về trong thao tác mở tệp trước đây, `fp` báo cho `getc()` biết phải đọc dữ liệu từ tệp nào trên đĩa. Nếu thao tác thành công, hàm trả lại kí tự đọc được, sau khi đã chuyển nó thành số **int**, nếu gặp lỗi chẳng hạn như đã đọc hết tệp rồi, `getc()` trả lại EOF.

Chẳng hạn đoạn chương trình sau sẽ thực hiện đọc đến cuối tệp văn bản :

```
ch = getc();
while (ch!=EOF){
      ch = getc( fp);
      ...
}
```

Như đã biết, hệ thống vào ra cấp II cũng có khả năng thao tác với tệp nhị phân. Khi mở một tệp theo kiểu nhị phân, có khả năng `getc()` sẽ đọc được cả giá trị EOF trong tệp. Khi đó đoạn chương trình trên sẽ không đọc hết đến cuối tệp như mong muốn. Để giải quyết vấn đề này, ANSI C cung cấp hàm `feof()`. Nó sẽ xác định đâu là dấu hiệu thực sự kết thúc tệp tin. Đoạn chương trình nhỏ dưới đây sẽ đọc một tệp nhị phân cho đến khi máy gặp kí tự kết thúc tệp thật sự :

```
while (!feof(fp)) ch = getc(fp);
```

Cách này có thể áp dụng cho cả tệp văn bản lẫn tệp nhị phân.

- *Hàm* `fclose()`

Hàm `fclose()` thực hiện đóng một kênh xuất nhập mà được mở bởi hàm mở trước đó. Khai báo như sau :

```
int fclose( FILE *fp);
```

`fclose()` đóng kênh xuất nhập trỏ bởi `fp`, tất cả bộ đệm của kênh xuất nhập này được dọn dẹp trước khi đóng, các bộ đệm của kênh xuất nhập được hệ thống cung cấp trong thao tác mở trước đây được giải phóng sau khi thực thi thao tác đóng, các bộ đệm xin trước đây bằng setbuf hay setvbuf cũng được tự động giải phóng

Hàm `fclose()` trả lại 0 khi thành công, nếu gặp lỗi nó sẽ trả lại giá trị EOF

Luôn nhớ rằng hệ điều hành hạn chế số tệp mở ra đồng thời, vì thế nên đóng tệp lại khi không dùng đến nó nữa. Đặc biệt cần lưu ý phải đóng hết tất cả các kênh xuất nhập khi kết thúc chương trình.

Các hàm `fopend()`, `getc()`, `putc()`, `fclose()` tạo thành một bộ tối thiểu các thao tác xử lý tệp. Chương trình `file1.c` thực hiện chép các ký tự từ một tệp này sang một tệp khác trên đĩa.

Bài tập mẫu 6.1

```
/*file1.c*/
#include <stdio.h>
void main() {
      FILE *in, *out;
```

```
    char ch;
    if ((fp = fopend( "SOURCE.DAT", "rb")) == NULL){
        printf("Khong mo duoc tep nguon.\n");
        exit(1);
        }
    if ((fp = fopend( "DEST.DAT", "wb")) == NULL){
        printf("Khong mo duoc tep dich.\n");
        exit(1);
        }
    while (!feof(in)) putc(getc(in), out);
    fclose(in);
    fclose(out);
    }
```

- *Hàm* `ferror()`

Hàm `ferror()` xác định xem thao tác truy xuất một kênh xuất nhập trước đó có lỗi hay không. Khai báo của hàm này có dạng :

```
int ferror( FILE *fp);
```

Hàm này trả về một số khác 0, nếu có lỗi trong thao tác vào ra trước đó liên quan đến kênh xuất nhập trỏ bởi fp, nếu thao tác vào ra trước đó thành công thì `ferror()` sẽ trả lại 0. Chú ý rằng cờ thông báo lỗi của kênh xuất nhập vẫn còn dựng cho đến khi `clearerr()` hay `rewind()` được gọi hoặc khi đóng tệp.

- *Hàm* `rewind()`

Thực hiện khởi tạo lại con trỏ vị trí tệp tin về đầu tệp. Dạng khai báo tổng quát của nó là :

```
void rewind( FILE *fp);
```

- *Hàm* `getw()` *và* `putw()`

Các hàm `getw()` và `putw()` giống như các hàm `getc()` và `putc()` nhưng thực hiện đọc/ghi các số nguyên 2 byte, thay vì đọc ghi một kí tự (1 byte).

Các khai báo của chúng như sau :

```
int getw(FILE *stream);
int putw(int w,FILE * stream);
```

- *Hàm* `fgets()` *và* `fputs()`

Hệ thống các hàm vào ra mức II còn có các hàm thực hiện đọc ghi các chuỗi trên một kênh xuất nhập được chỉ định `fgets()` và `fputs()`. Dạng khai báo của chúng như sau :

```
char *fputs(char *str,FILE *stream);
char *fgets(char *str,int num,FILE *stream);
```

Hàm `fputs()` làm việc hoàn toàn giống như `puts()`, chỉ khác là `fputs()` ghi một chuỗi lên một kênh xuất nhập được chỉ định. Hàm `fgets()` đọc một chuỗi từ kênh xuất nhập được chỉ định cho tới khi nó đọc được hoặc một kí tự xuống dòng hay đã đọc được `num-1` kí tự. Khi `fgets()` kết thúc, hàm sẽ tự động bỏ thêm 1 kí tự rỗng vào cuối chuỗi mà không bỏ đi kí tự xuống dòng.

- *Các hàm* `fread()` *và* `fwrite()`

Hệ thống các hàm vào ra mức II cung cấp hai hàm `fread()` và `fwrite()` cho phép đọc và ghi các khối dữ kiện. Dạng khai báo của chúng:

```
int fread(void *buffer,int so_byte,int so_muc,FILE *fp);
int fwrite(void *buffer, int so_byte, int so_muc,FILE *fp);
```

Trong trường hợp của `fread()`, buffer là con trỏ trỏ đến vùng nhớ mà nó sẽ nhận dữ liệu đọc từ tệp. Đối với `fwrite()`, buffer là con trỏ trỏ đến vùng nhớ mà từ đó thông tin sẽ được ghi lên tệp. Trong cả hai trường hợp `so_byte` mô tả số byte đọc hay ghi của mỗi mục (thường gọi là bản ghi) còn `so_muc` xác định xem có bao nhiêu mục hay bản ghi. Cuối cùng `fp` là con trỏ trỏ đến kênh xuất nhập đã mở trước đó. Xét chương trình ví dụ sau:

Bài tập mẫu 6.2

```
/*file2.c*/
#include <stdio.h>
void main(){
FILE *fp;
float  f = 12.34;
if ((fp = fopend( "TEST.DAT", "wb")) == NULL){
      printf("Khong mo duoc tep.\n");
      exit(1);
      }
fwrite( &f, sizeof(float), 1, fp);
fclose(fp);
```

```
}
```

Chỉ cần tệp được mở với chế độ truy xuất kiểu nhị phân thì các hàm trên có thể thao tác với kiểu dữ liệu bất kỳ. Chương trình `file2.c` trên đây ghi một số kiểu **float** lên một tệp trên đĩa.

Từ ví dụ trên, ta có thể nhận thấy rằng buffer có thể là biến đơn. Tuy nhiên, `fread()` và `fwrite()` được ứng dụng nhiều trong các thao tác đọc ghi các bảng hay dãy các cấu trúc. Chương trình `file3.c` ghi nội dung của bảng a chứa 100 số **float** lên tệp ngoài chỉ bằng một lời gọi duy nhất đến `fwrite()`.

Bài tập mẫu 6.3

```
/*file3.c*/
#include <stdio>
void main(){
FILE *fp;
float  a[100];
if ((fp = fopend( "TEST.DAT", "wb")) == NULL){
      printf("Khong mo duoc tep.\n");
      exit(1);
      }
fwrite( &f, sizeof(a), 1, fp);
fclose(fp);
}
```

3.4 Truy xuất vào ra ngẫu nhiên (random access I/O) và fseek()

Trong hệ thống các hàm vào ra mức II, các thao tác truy xuất ngẫu nhiên có thể thực hiện được nhờ hàm `fseek()`, nó xác lập con trỏ chỉ vị trí của tệp (file-position locator) chỉ đến một vị trí mới theo yêu cầu. Dạng khai báo của `fseek()` như sau :

```
int fseek(FILE *fp, long int so_byte, int xuat_phat);
```

trong đó, `fp` là một con trỏ tệp mà nó trả về bằng phép mở tệp trước đó, `so_byte` kiểu **long** là khoảng cách tính theo byte kể từ vị trí xuất phát `xuat_phat` đến vị trí mới, còn `xuat_phat` là một trong các hằng trong bảng 6.3.

Cần nhớ rằng so_byte phải là kiểu `long` để thao tác được trên các tệp có kích thước lớn hơn 64 kbytes.

Không nên dùng `fseek()` trên các tệp văn bản vì sự chuyển đổi kí tự khi nhập hoặc xuất sẽ làm cho kết quả của việc định vị trong tệp bị sai. Chỉ nên sử dụng `fseek()` với những tệp truy xuất ở chế độ nhị phân.

Xuất phát từ	Tên hằng	Giá trị
Đầu tệp	SEEK_SET	0
Vị trí hiện thời	SEEK_CUR	1
Cuối tệp	SEEK_END	2

Bảng 6.3 Các hằng dùng trong fseek

Chương trình `file4.c` sẽ đọc byte thứ 234 trong tệp DATA.DAT:

Bài tập mẫu 6.4

```
/*file4.c*/
#include <stdio.h>
void main() {
FILE *fp;
float  f = 12.34;
if ((fp = fopend( "DATA.DAT", "rb")) == NULL) {
      printf("Khong mo duoc tep.\n");
      exit(1);
      }
fseek( fp, 234L, SEEK_SET);
printf( "Ki tu thu 234 la : %c", getc(fp));
fclose(fp);
}
```

Chú ý rằng trong lời gọi hàm `fseek()` ở trên có thêm hậu tố L vào sau hằng 234 để bắt buộc trình biên dịch xử lý hằng này như một số **long int**. Cũng có thể sử dụng chuyển kiểu để thực hiện điều này như sau :

```
pos = 234;
fseek( fp, (long)pos, SEEK_SET);
```

Nếu `fseek()` trả về giá trị 0 thì có nghĩa là thao tác dịch chuyển thành công, ngược lại khi `fseek()` trả lại một giá trị khác 0 có nghĩa là thao tác thất bại.

3.5 Các kênh xuất nhập stdin, stdout, stdaux, stdprn, stderr

Khi một chương trình C được thực hiện, ba kênh xuất nhập sau đều được tự động mở ra. Các kênh xuất nhập đó là của các thiết bị nhập chuẩn - `stdin`, thiết bị xuất chuẩn - `stdout` và thiết bị báo lỗi chuẩn - `stderr`. Thông thường các kênh xuất nhập này chỉ tới bàn phím và màn hình, nhưng hệ điều hành có thể đổi hướng chúng chúng đến các thiết bị kênh xuất nhập (thiết bị logic) khác.

Vì các kênh xuất nhập này là những con trỏ tệp, do đó hệ thống vào ra mức II có thể sử dụng chúng để thực hiện các thao tác vào ra trên trên bàn phím và màn hình. Ví dụ hàm `putchar()` sau sẽ ghi một kí tự ra thiết bị ra chuẩn:

```
void putchar( char c){
        putc( c, stdout);
    }
```

Nói chung, `stdin` dùng để đọc vào từ bàn phím, `stdout` dùng để ghi ra màn hình và `stderr` cũng dùng để ghi ra màn hình.

Cần nhớ rằng các con trỏ tệp này được tự động tạo ra khi chương trình bắt đầu và tự động đóng lại khi chương trình kết thúc, không nên cố tình đóng hay mở chúng.

Ngoài ba thiết bị chuẩn trên, ANSI C còn cung cấp thêm hai kênh xuất nhập của hai thiết bị khác đó là :

- `stdaux` : là thiết bị phụ trợ chuẩn, thường là COM1, dùng để kết nối với các thiết bị vào ra tuần tự
- `stdprn` : là thiết bị in chuẩn (máy in), thường là PRN hay LPT1

♦ *Các hàm* `fprintf()` *và* `fscanf()`

Ngoài các hàm vào ra cơ bản được trình bày ở trên, hệ thống vào ra mức II còn có hai hàm khác là `fprintf()` và `fscanf()`. Các hàm này xử lý giống `printf()` và `scanf()`, chỉ có một điểm khác là chúng thao tác với các kênh xuất nhập. Dạng khai báo tổng quát của là :

```
int fprintf( FILE *fp, xau_dieu_khien, danh_sach_tham_so);
int fscanf( FILE *fp, xau_dieu_khien, danh_sach_tham_so);
```

Trong đó `fp` là con trỏ tệp trả về bởi thao tác mở kênh xuất nhập trước đó. Như đã nói ở trên, hàm `fprintf()` kết xuất dữ liệu ra theo khuôn dạng qui định của `xau_dieu_khien` lên một kênh xuất nhập (chứ không phải ra thiết bị ra chuẩn như của `printf()`), còn `fscanf()` sẽ nhập dữ liệu theo khuôn dạng qui định của chuỗi điều khiển từ một kênh xuất nhập (chứ không phải từ thiết bị vào chuẩn như của `scanf()`).

Hai hàm này trả lại một trị số nguyên, mô tả số byte đã ghi hoặc đọc được. Khi gặp lỗi, cả hai hàm trả lại EOF

4. CÁC HÀM THAO TÁC VÀO RA TỆP MỨC THẤP

4.1 Tổng quan về các hàm vào ra mức thấp

Do tính lịch sử, ở giai đoạn mới hình thành và phát triển, ban đầu ngôn ngữ C được phát triển trong môi trường hệ điều hành UNIX. Ngày nay, do những ưu điểm của mình, ngôn ngữ C trở nên thông dụng, phát triển mạnh mẽ trên các hệ điều hành khác như DOS, OS/2 ... Nhưng để duy trì tính kế thừa và khả năng tương thích của các chương trình viết bằng C dưới hệ điều hành UNIX, hệ thống các thao tác vào ra với tệp mức thấp vẫn được đưa ra. Các thao tác này thấp hơn với các thao tác mà ta đã làm quen trong phần trước. Chúng gần gũi với các thao tác truy xuất của máy hơn, mà thực ra các hàm vào ra tệp mức II chủ yếu cũng dựa trên hệ thống các thao tác vào ra vào ra mức thấp này nên nó còn được gọi là hệ thống vào ra mức I hay hệ thống vào ra giống UNIX hay hệ thống vào ra không được cung cấp vùng đệm. Thực ra, để thực hiện các thao tác vào ra, hệ điều hành sử dụng các vùng đệm riêng. Ở đây ta hiểu là các hàm này không sử dụng các vùng đệm của trình biên dịch C cung cấp cho riêng các thao tác vào ra này. Bảng 6.4 dưới đây trình bày một số hàm vào ra mức I .

Tên hàm	Chức năng
`read()`	Đọc dữ liệu từ đĩa vào vùng đệm
`write()`	Ghi dữ liệu từ vùng đệm ra đĩa
`open()`	Mở một tệp trên đĩa
`close()`	Đóng một tệp trên đĩa
`lseek()`	Chuyển vị trí thao tác trên tệp
`unlink()`	Xoá một tệp khỏi thư mục

Bảng 6.4 Một số hàm truy xuất thông dụng mức thấp

Không giống với `getc()` và `putc()` là những hàm đọc kí tự từ kênh xuất nhập dữ kiện hay ghi kí tự ra một kênh xuất nhập, `read()` và `write()` đọc hoặc ghi thông tin lên vùng nhớ mỗi lần được gọi (tức là mỗi lần truy xuất cả khối thông tin chứ không phải từng byte kí tự(cách xử lý này tương tự như cách xử lý của `fread()` và `fwrite()`).

Để có thể sử dụng các hàm vào ra mức thấp này, cần phải cung cấp các thông tin cần thiết về các hàm này cho chương trình bằng cách đặt chỉ thị

```
#include  <io.h>
```

vào đầu chương trình.

4.2 Một số hàm vào ra mức thấp thông dụng

- *Hàm* `open()`

Không giống như hệ thống vào ra tệp mức II, hệ thống vào ra mức I nay không sử dụng kênh xuất nhập và con trỏ tệp, mà nó sử dụng thẻ tệp.

Cũng như trong hệ thống vào ra mức II, một tệp phải được mở ra trước khi thực hiện các thao tác trên tệp đó. Một tệp được mở bằng hàm `open()`, nếu thao tác mở thành công thì hàm này trả về một số nguyên là thẻ tệp tương ứng với tệp vừa mở. Dạng khai báo tổng quát của `open()` là :

```
int open(char*ten_tep,int mode [,unsigned access]);
```

Trong đó :

`ten_tep` là tên hợp lệ của một tệp

`mode` là một trong các hằng sau :

Hằng kí hiệu	Giá trị	Hiệu quả
O_RONLY	0	Chỉ đọc (read-only)
O_WRONLY	1	Chỉ ghi (write-only)
O_RDWR	2	Đọc/ghi (Read/Write)

Bảng 6.5 Các giá trị cơ bản của mode

Ba chế độ trên là ba chế độ cơ bản, các hằng kí hiệu tương ứng với chúng được định nghĩa trong tệp "`fcntl.h`". Do đó, nếu chương trình muốn sử dụng các hằng kí hiệu này, dòng chỉ thị

```
#include  <fcntl.h>
```

cần phải được đặt ở đầu chương trình.

Ngoài ra, ANSI C còn cho phép một số mục lựa chọn khác của `mode`, các lựa chọn này có thể kết hợp với nhau nhờ toán tử OR trên bit. Xem bảng 6.6.

Nếu không có mô tả cụ thể tệp mở theo chế độ văn bản hay nhị phân thì tệp được mở ra theo giá trị được lưu trong biến công cộng `_fmode` (giá trị mặc nhiên của biến này là chế độ văn bản).

Việc sử dụng tham số `access` trong hàm `open()` có liên quan tới hệ điều hành, một phiên bản khác của hàm `open()` liên quan đến hệ điều hành DOS có tên là `_open`, dạng khai báo tổng quát của hàm này là :

```
int _open( char *ten_tep, int mode);
```

Như vậy trong hàm này bỏ qua tham số `access`.

Hàm `open()` khi gọi thành công sẽ trả về một trị nguyên dương, đó chính là thẻ tệp. Sau này, các thao tác với tệp được thực hiện thông qua thẻ tệp trên.

Khi hàm `open()` gặp lỗi, nó sẽ trả lại giá trị -1 và biến toàn cục `errno` bằng một trong các giá trị sau :

`ENOENT` : Không có tệp như vậy

`EMFILE` : Quá nhiều tệp được mở

`EACCES` : Không có quyền truy nhập

`EINVACC` : Không đúng mã truy nhập

Các hằng trên định nghĩa trong tệp tiêu đề `errno.h`

Hằng kí hiệu	Hiệu quả
O_ANPEND	Tất cả các tác vụ ghi đều thực hiện ở cuối tệp
O_CREATE	Nếu tệp chưa tồn tại trên đĩa thì nó sẽ được tạo ra. Nếu sử dụng mode này, cần phải cung cấp thêm tham số thứ ba access mô tả trạng thái truy xuất của tệp đối với hệ điều hành từ các hằng kí hiệu sau : • S_IWRITE : để cho phép ghi • S_IREAD : để cho phép đọc • S_IWRITE | S_IREAD : cho phép cả hai Các hằng kí hiệu này được định nghĩa trong tệp sys\stat.h
O_TRUNC	Xoá nội dung của một tệp nếu nó đang tồn tại, thuộc tính của nó được giữ không đổi
O_TEXT	Mở tệp ở dạng văn bản
O_BINARY	Mở tệp ở dạng nhị phân

Bảng 6.6 Các giá trị khác của mode

◆ *Hàm* `close()`

Thực hiện đóng một tệp tương ứng với một thẻ tệp được mở ra trước đó. Khai báo của nó có dạng :

```
int close( int handle);
```

Khi gọi `close()`, nó sẽ giải phóng thẻ tệp, thẻ tệp này có thể sử dụng cho tệp khác. Ngoài ra `close()` còn thực hiện ghi các thông tin còn chứa trong vùng đệm lên đĩa. Nếu đóng một tệp không thành công thường dẫn tới hậu quả mất số liệu.

Hàm `close()` trả lại giá trị 0 khi nó thực hiện thành công và -1 khi nó gặp lỗi, chẳng hạn bạn đã lấy đĩa mềm ra khỏi ổ.

- *Hàm* `creat()`

Thực hiện việc tạo một tệp mới cho thao tác ghi về sau, nếu tệp đã tồn tại, nó sẽ ghi đè lên tệp đó. Dạng khai báo của nó :

```
int creat(char *ten_tep, int access);
```

Ý nghĩa của các tham số giống như của hàm `open()`.

Khi tệp đã tồn tại, hàm `creat()` đặt độ dài tệp bằng 0, các thuộc tính khác của tệp không thay đổi. Còn khi tệp đó lại có thuộc tính chỉ đọc, hàm `creat()` sẽ không thực hiện được và tệp không thay đổi. Một tệp tạo bằng `creat()` luôn mở theo chế độ nhị phân hay văn bản tuỳ vào biến toàn cục `_fmode`.

Giá trị trả lại của hàm `creat()` giống như giá trị trả lại của hàm `open()`.

Tương tự như hàm `open()`, việc sử dụng tham số `access` có liên quan đến hệ điều hành UNIX, một phiên bản khác của C gắn liền với môi trường DOS là hàm `_creat()`. Dạng khai báo của hàm `_creat()` như sau:

```
int _creat( char* ten_tep, int attrib);
```

Trong đó `attrib` là thuộc tính tệp mới được tạo ra, có thể sử dụng các hằng sau để xác định thuộc tính của tệp :

`FA_RDONLY`	thuộc tính chỉ đọc
`FA_HIDDEN`	thuộc tính ẩn
`FA_SYSTEM`	thuộc tính hệ thống
`FA_ARCH`	thuộc tính lưu trữ

- *Hàm* `write()`

Hàm `write()` thực hiện ghi dữ liệu ra tệp. Dạng khai báo tổng quát của nó là :

```
int write( int handle, void *buf, unsigned len);
```

Trong đó `handle` là thẻ tệp nhận được từ các thao tác `open()`, `creat()` . . . trước đây, `len` là số byte hàm muốn ghi và `buf` là con trỏ trỏ đến vùng đệm mà hàm đó ghi.

Khi thành công, hàm trả về số byte được ghi thực sự còn nếu có lỗi hàm trả lại giá trị -1. Nếu số byte thực sự được ghi nhỏ hơn số byte yêu cầu ghi thì có nghĩa là đã xảy ra lỗi (chẳng hạn đĩa đầy).

Khi thao tác với tệp văn bản, nếu hàm `write()` nhận được một kí tự LF, nó sẽ ghi ra cặp kí tự LF/CR.

- *Hàm* `read()`

Ngược với hàm `write()`, hàm `read()` thực hiện đọc dữ liệu từ một tệp đã mở vào bộ nhớ. Dạng khai báo của hàm `read()` như sau:

```
int read( int handle, void *buf, unsigned len);
```

Các tham số như trong `write()`: `handle` là thẻ tệp, `buf` là địa chỉ vùng nhớ đọc dữ liệu vào và `len` là số byte mà `read()` thử đọc vào bộ đệm `buf`.

Khi thành công, hàm `read()` trả lại số byte thực sự đặt vào `buf`. Khi truy nhập "đến cuối tệp - End Of File", read trả lại 0. Còn khi có lỗi, `read()` trả lại -1 và gán mã lỗi tương ứng cho biến toàn cục `errno`.

Khi mở tệp theo kiểu văn bản, `read()` khi đọc sẽ bỏ qua các kí tự trở về đầu dòng CR và nó sẽ thông báo hết tệp khi gặp kí tự Ctrl-Z đầu tiên. Chú ý rằng số byte mà `read()` đọc thành công được trả lại không đếm các kí tự CR và Ctrl-Z.

- *Hàm* `lseek()`

Hàm `lseek()` cũng dùng để thực hiện truy xuất ngẫu nhiên các tệp tương tự như hàm `fseek()` như thao tác trực tiếp với tệp không qua các kênh xuất nhập. Khai báo của `lseek()` như sau :

```
long lseek(int handle, int so_byte, int xuat_phat);
```

Ý nghĩa các tham số hoàn toàn giống như hàm `fseek()`.

Ví dụ sau thực hiện tạo một tệp mới, ghi lên tệp một xâu, sau đó thực hiện đọc lại xâu đó.

Bài tập mẫu 6.5

```
/*file5.c*/
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
int main(void) {
int handle;
char msg[] = "This is a test";
char ch;
```

```
/* Tạo tệp */
handle = open("TEST.$$$", O_CREAT | O_RDWR, S_IREAD | S_IWRITE);
/* Ghi dữ liệu lên tệp */
write(handle, msg, strlen(msg));
/* Chuyển đến đầu tệp */
lseek(handle, 0L, SEEK_SET);
/* Đọc các kí tự từ tệp cho đến khi gặp EOF */
do {
read(handle, &ch, 1);
printf("%c", ch);
}while (!eof(handle));
close(handle);
return 0;
}
```

5. CHỌN PHƯƠNG ÁN SỬ DỤNG

Đối với những chương trình mới, nên sử dụng hệ thống vào ra mức II . Tuy nhiên, các chương trình sử dụng hệ thống vào ra cấp I vẫn có thể dùng trong nhiều năm nữa.

Trong hệ thống xuất nhập cấp II, khi làm việc với các tệp văn bản, bạn nên sử dụng hàm `getc()` và `putc()` ở chế độ văn bản. Tuy nhiên khi thao tác với những tệp nhị phân hoặc các tệp có kiểu dữ liệu phức tạp, nên sử dụng các tệp nhị phân với các hàm `fread()` và `fwrite()`.

Đặc biệt, cần lưu ý rằng không bao giờ được trộn lẫn các hàm thuộc hai hệ thống vào ra khác nhau vào trong một chương trình, bởi vì phương pháp tiếp cận và xử lý các tệp của mỗi hệ thống khác nhau và chúng có thể gây rối loạn lẫn nhau.

6. TÓM TẮT

6.1 Các lỗi thường gặp khi lập trình

Quên không gộp tệp tiêu đề của thư viện các hàm đã dùng.

Quên mở tệp trước khi thực hiện các thao tác xử lý.

Dùng con trỏ tệp qui chiếu đến một tệp không hợp lệ.

Mở một tệp chưa tồn tại cho thao tác đọc.

Mở một tệp để đọc, ghi mà quên không gán các quyền truy nhập tương ứng cho nó (các quyền này tuỳ vào từng hệ điều hành).

Mở một tệp để ghi mà không còn đủ vùng trống trên đĩa.

Mở một tệp đã tồn tại cho cả thao tác ghi và đọc trong khi mục đích sử dụng chỉ cần đọc, khi đó nội dung của tệp có thể bị thay đổi do không để ý mà chẳng có một lưu ý gì cho người dùng.

6.2 Các thói quen tốt khi lập trình

Đảm bảo rằng con trỏ tệp trong các lời gọi hàm là hợp lệ.

Đóng tệp ngay mỗi khi chương trình không cần đến nó nữa

Phải đóng hết tất cả các tệp trước khi thoát ra khỏi chương trình

Mở tệp theo chế độ chỉ cho phép đọc nếu nội dung tệp cần được giữ nguyên, điều này tránh các thay đổi nội dung tệp ngoài mong muốn.

7. BÀI TẬP

Bài tập 6.1 Mở một tệp văn bản, đếm xem tệp tin đó có bao nhiêu dòng, bao nhiêu ký tự và bao nhiêu từ.

Bài tập 6.2 Đọc các số nguyên từ bàn phím rồi lưu chúng vào một tệp văn bản tên là "NGUYEN.DAT", mỗi số nguyên trên một dòng. Đọc lại tệp đó và lưu các số vào một mảng số nguyên. Sắp xếp tăng dần dãy số nguyên rồi ghi chúng vào cuối tệp "NGUYEN.DAT".

Bài tập 6.3 Có một tệp văn bản chứa các số. Dòng đầu tiên là một số nguyên, các dòng tiếp theo có hai số thực trên mỗi dòng. Hai số thực phân cách nhau bằng một dấu phẩy và một số dấu cách. Hãy đọc các số từ tệp và lưu một cấu trúc mô tả toạ độ một đa giác. Số nguyên mô tả số đỉnh của đa giác, mỗi cặp số thực tương ứng là toạ độ một đỉnh của đa giác.

Bài tập 6.4 Làm lại các bài tập 6.1 và 6.2 với các tệp nhị phân.

Bài tập 6.5 Cải tiến các chương trình trong các bài tập mẫu 5.5, 5.6, 5.7 trong chương 5 bằng cách thêm vào khả năng lưu lại các danh sách (ngăn xếp, hàng đợi, cây nhị phân) trong các tệp mở ở chế độ nhị phân. Khôi phục lại các danh sách ban đầu bằng cách đọc lại nội dung các tệp vừa ghi.

Chương 7

ĐỒ HOẠ

Mục đích chương này

1) Khái niệm chung về màn hình đồ họa
2) Biết cách khởi động chế độ đồ họa
3) Làm quen với các hàm đồ họa của Borland
4) Cách sử dụng các hàm đã nêu để viết các chương trình đồ họa.

1. GIỚI THIỆU

Giao diện đồ hoạ đang ngày càng trở nên phổ biến trong các ứng dụng trên máy vi tính. Các phần mềm trên DOS trước đây dùng giao diện văn bản nay cũng chuyển dần sang giao diện đồ hoạ, thậm chí hệ điều hành DOS giờ đây cũng có DOS-Shell, một hình thức giao diện thân thiện với người dùng. Các hệ điều hành ngày nay như Windows, OS/2 trên các máy PC, hay System trên các máy Mac là ví vụ điển hình cho loại giao diện trên.

Tuy nhiên, có một thực tế là danh sách các loại bo mạch và màn hình ngày càng nhiều, vì vậy việc viết một chương trình đồ hoạ đảm bảo về mặt tốc độ nhưng vẫn đảm bảo tính tương thích cao bằng hợp ngữ là không thể được nữa.

Các hãng viết trình biên dịch C có đưa ra các thư viện đồ hoạ thực hiện các thao tác đồ hoạ cơ bản, giúp cho việc phát triển những ứng dụng đồ hoạ trở nên dễ dàng hơn, đảm bảo tính tương thích cao, cho phép ứng dụng chạy trên nhiều cấu hình khác nhau về phần cứng.

Có thể nhận thấy thư viện các hàm đồ hoạ của các trình biên dịch khác nhau không giống nhau. Ở đây, chúng tôi xin giới thiệu thư viện các hàm đồ hoạ của hãng Borland.

Tuy nhiên, trước hết ta sẽ tìm hiểu các vấn đề cơ bản của đồ hoạ trên các máy tính cá nhân.

2. ĐỒ HOẠ TRÊN MÁY TÍNH CÁ NHÂN

Trong các máy IBM PC (và tương thích), thiết bị ra màn hình gồm có hai thành phần: bộ điều khiển màn hình (display adapter - hay còn gọi là bo mạch hiển

thị) và màn hiển thị (display monitor). Adapter là một bo mạch cắm trong một khe bên trong máy tính và monitor để kết xuất các kí tự hay các hình vẽ. Màn hình được chia thành các điểm sáng nhỏ (mỗi điểm sáng này được gọi là một điểm ảnh - pixel - viết tắt của picture element). Số lượng các điểm sáng này trên màn hình xác định độ phân giải của nó. Chẳng hạn màn hình có độ phân giải 640 * 480 có nghĩa là màn hình được chia nhỏ ra thành ma trận các điểm, chiều ngang có 640 điểm, chiều dọc có 480 điểm. Các hình ảnh hiển thị trên màn hình được là do bộ điều khiển màn hình quản lý màn hình điều khiển các điểm sáng trên monitor (bật hay tắt).

Máy tính có thể hoạt động theo hai chế độ hiển thị khác nhau: *chế độ văn bản* và *chế độ đồ hoạ*. Hai chế độ này khác nhau ở cách quản lý các điểm ảnh trên.

Trong chế độ văn bản, các kí tự được lưu thành các mẫu (các ô hình chữ nhật) khi muốn hiển thị một kí tự trên màn hình, chương trình chỉ cần gửi kí tự yêu cầu hiển thị đến cho bo mạch màn hình, bo mạch màn hình sẽ hiển thị kí tự trên theo mẫu đăng kí trước. Các điểm ảnh trong hình hộp khung của kí tự sẽ được bật hay tắt đồng thời. Làm việc trong chế độ văn bản, các điểm ảnh được quản lý theo từng nhóm, các hình hộp có kích thước như nhau, chẳng hạn 8 * 8, 8 * 16, 9 * 16 ... tuỳ vào độ phân giải và chế độ hoạt động. Nói chung, trong chế độ văn bản, màn hình có khả năng hiển thị 25 dòng, mỗi dòng 80 kí tự.

Trong chế độ đồ hoạ, ta có thể thao tác từng điểm ảnh một, có thể tạo những hình hộp vẽ các kí tự trong đó như trong chế độ văn bản, hay có thể vẽ các điểm riêng biệt, các đường thẳng, đường cong, các ảnh tuỳ thích. Tuy nhiên, khi đó do phải tự quản lý từng điểm ảnh nên lượng thông tin cần quản lý rất lớn. Hãy thử so sánh, trong chế độ văn bản 25 dòng 80 cột, tổng cộng có 25 * 80 = 2000 kí tự, trong khi đó với độ phân giải 640 * 480 tức là chiều ngang có 640 điểm ảnh, chiều dọc có 480 điểm ảnh thì tổng cộng có tới 640 * 480 = 307200 điểm. Đó là lý do tại sao xử lý các thao tác đồ hoạ lâu và tốn bộ nhớ hơn các thao tác trong chế độ văn bản.

2.1 Các bộ điều khiển màn hình và màn hình của các máy IBM PC và tương thích

Có 6 kiểu màn hình thông dụng là: MDA (Monochrome Display Adapter), CGA (Color Graphics Adapter), EGA (Enhanced Graphics Adapter), HGC (Hercules Graphics Card), MCGA (Multicolor Graphics Adapter) và VGA (Video Graphics Adapter).

Ban đầu, MDA được dùng trong các máy IBM PC. Nó có khả năng hiển thị 25 dòng, mỗi dòng có 80 kí tự trong chế độ văn bản với 2 màu. Nhưng nó không có khả năng hiển thị đồ hoạ. HGC hiển thị văn bản giống như MDA, tuy nhiên nó có khả năng hoạt động ở chế độ đồ hoạ với độ phân giải 720 * 348.

CGA có khả năng hiển thị cả đồ hoạ lẫn văn bản và có khả năng hiển thị màu trên các monitor màu. Trong chế độ văn bản, CGA có thể hiển thị 16 màu chữ khác nhau (8 màu với 2 mức sáng khác nhau) và 8 màu nền khác nhau. Trong chế độ đồ hoạ, nó có khả năng hiển thị 4 mầu với độ phân giải 320 * 200. Nếu chọn chỉ hiển thị 2 màu, độ phân giải của nó tăng lên 640 * 200. Mối quan hệ giữa độ phân giải và số màu sẽ được giải thích sau.

EGA, do IBM đưa ra năm 1984, nâng cao độ phân giải và số lượng mầu hơn nữa. Để đảm bảo tính tương tích, EGA có khả năng mô phỏng theo CGA. Độ phân giải cao nhất trong chế độ đồ hoạ của EGA là 640 * 350 với 16 màu khác nhau.

VGA được IBM đưa ra cùng với MCGA. VGA nâng cao các đặc tính của EGA. Trong video mode (mode 12h), VGA có độ phân giải 640 * 480 với 16 màu khác nhau được chọn từ bảng 256 màu. VGA cũng cho phép hoạt động ở chế độ 256 màu khác nhau với độ phân giải 320 * 200 (mode 13h). VGA có thể hoạt động ở tất cả các chế độ của CGA và EGA. MCGA tương tự như CGA nhưng có chế độ đồ hoạ 2 màu với độ phân giải 640 * 480.

Màn hình VGA chuẩn thuộc loại màn hình tương tự, khác với các màn hình số như MDA, CGA, EGA. Đúng như tên gọi, các màn hình số dùng một số xác định các đường tín hiệu on/off để hiển thị màu. Ví dụ màn hình EGA sử dụng 6 đường tín hiệu là các đường red, green, blue và các đường sáng hơn của 3 màu này. Vậy tổng cộng, nó có khả năng hiển thị tối đa 2^6 = 64 màu khác nhau. Tuy nhiên tại mỗi thời điểm nó chỉ có khả năng hiển thị 16 mầu khác nhau (trong số 64 màu trên), vì chỉ có 4 bit cho mỗi điểm. Trong các màn hình tương tự, các đường tín hiệu red, green, blue có thể thay đổi chứ không phải chỉ có tắt và bật như trong màn hình số. Điều này cho phép màn hình tương tự hiển thị rất nhiều màu khác nhau (có thể tạm hiểu là vô số màu). Tuy nhiên khả năng sinh màu trên các bộ điều khiển màn hình lại hạn chế, ví dụ MCGA dùng 6 bit để biểu diễn mầu. Có một bộ DAC (Digital-to-Analog Converter) sẽ chuyển các tín hiệu số này sang thành tín hiệu tương tự trước khi chuyển sang cho monitor. Vì vậy thực ra, số lượng màu khác nhau có thể hiển thị được các định bởi miền giá trị được biểu diễn số trong bộ điều khiển màn hình.

2.2 Các bộ điều khiển màn hình có độ phân giải cao hơn

IBM 8514/A là bo mạch đồ hoạ có phần cứng đồng xử lý các tín hiệu. 8514/A chuẩn cho phép độ phân giải tới 1024 * 1024, các phiên bản sau này có độ phân giải lên tới 2560 * 2048. Giữa độ phân giải của 8514/A và VGA chuẩn là độ phân giải của bộ điều khiển Super VGA, độ phân giải 800 * 600 hay 1024* 768 với 16 màu. Do sự phát triển mạnh mẽ của màn hình và bộ điều khiển màn hình, nên ngày nay các phần mềm thương mại đạt được chất lượng tốt thường sử dụng các màn hình và bộ điều khiển từ loại VGA trở lên.

2.3 Lập trình trên bo mạch màn hình

Giống như hầu hết các thiết bị ngoại vi của máy IBM PC và các máy tương thích, hầu hết các bo mạch màn hình đều có thể lập trình qua các thanh ghi bằng cách truy nhập trực tiếp các cổng vào ra. Tất cả các bộ điều khiển màn hình của PC đều sử dụng một nguyên lý chung: mỗi một điểm ảnh đều tương ứng với một số bit trong bộ nhớ nằm trên bo mạch màn hình (Video RAM). Bộ điều khiển màn hình đọc các giá trị từ Video RAM để hiển thị các điểm ảnh lên màn hình. Điều này cho phép bạn hiển thị các văn bản cũng như các hình vẽ trên màn hình bằng cách thao tác trực tiếp trên Video RAM.

Trong chế độ hiển thị văn bản, các mẫu hình chữ nhật các điểm được dùng để hiển thị các kí tự. Mỗi kí tự cần dùng 2 byte để hiển thị: byte thấp dùng để lưu mã 8 bit ASCII của kí tự, byte cao lưu các thuộc tính hiển thị của kí tự như màu chữ, màu nền, thuộc tính nhấp nháy. Mẫu các kí tự cần thiết cho việc hiển thị kí tự được lưu trong các bảng riêng lưu trong bộ nhớ (đó là các bảng font - các font ngầm định sẽ được lưu trong ROM và được đọc vào bộ nhớ khi khởi động)

Trong chế độ đồ hoạ, số các bit cần thiết để biểu diễn một điểm ảnh tuỳ thuộc vào số màu được hiển thị. Đối với màn hình đen trắng, mỗi điểm ảnh hoặc tắt hoặc bật, vì vậy chỉ cần 1 bit để biểu diễn 1 điểm ảnh. Đối với trường hợp 16 màu, ta cần 4 bit để lưu giữ thông tin về mỗi điểm. Nói chung nếu dùng n bit để lưu giữ thông tin cho mỗi điểm thì có thể có tối đa 2^n mầu khác nhau. Ta biết rằng bộ nhớ trên bo mạch màn hình là hạn chế, thường đối với VGA là 256K. Do đó khi tăng độ phân giải, số màu có thể biểu diễn đồng thời sẽ giảm đi và ngược lại.

Tổ chức vật lý của Video RAM không giống nhau giữa các loại bộ điều khiển màn hình. Ở đây không phải là một cuốn sách viết sâu về vấn đề này, nên không đi sâu cụ thể vào cấu trúc các loại bộ điều khiển màn hình. Tuy nhiên có thể thấy rằng việc lập trình trực tiếp trên bo mạch màn hình là một việc khó khăn, chưa kể đến vấn đề tương thích nhằm đảm bảo một chương trình có thể hoạt động tốt trên loại bo mạch màn hình này vẫn có thể hoạt động trên bo mạch màn hình khác.

2.4 Các chương trình con BIOS Video

Các chương trình con BIOS nằm trong ROM của các máy PC. Đó là những chương trình con chuẩn phục vụ cho việc lập trình với các bộ điều khiển màn hình. Dùng ngắt 10h, ta có thể thao tác với một loạt các hàm thao tác với màn hình, cho phép đặt chế độ video, đọc và ghi từng điểm... BIOS chỉ cung cấp các thao tác cơ bản nhất đối với màn hình, nhưng các thao tác này tương đối chậm. Để vẽ nhanh hơn, đặc biệt là trong chế độ đồ hoạ, ta cần phải thực hiện các thao tác trực tiếp với bộ nhớ màn hình. Rõ ràng dùng BIOS thì nâng cao được tính tương thích với nhiều

loại bộ điều khiển màn hình khác nhau nhưng tốc độ sẽ chậm hơn rất nhiều. Còn thao tác trực tiếp với bộ nhớ thì tốc độ nhanh hơn, nhưng tính tương thích lại giảm.

2.5 Giải pháp

Trong khung cảnh như vậy, các hãng viết trình biên dịch C đưa ra thư viện các hàm phong phú hơn, chạy nhanh hơn BIOS mà vẫn đảm bảo tính tương thích. Hơn nữa, các hàm này sử dụng dễ dàng hơn các chương trình con cung cấp bởi ngắt 10h của BIOS. Để sử dụng các hàm này, không cần phải hiểu về cấu trúc cụ thể của từng bộ điều khiển màn hình.

Cụ thể trong giải pháp của Borland, ta chỉ cần hiểu mô hình được Borland sử dụng. Sau đây, chúng ta sẽ làm quen dần dần với các khái niệm hệ toạ độ, mầu sắc, bảng mầu, các kiểu đường, các hình cơ bản ...

3. CÁC CHẾ ĐỘ ĐỒ HOẠ, HỆ TOẠ ĐỘ

3.1 Màn hình và hệ toạ độ

Trong chế độ đồ hoạ, màn hình được coi như một ma trận các điểm ảnh riêng lẻ. Tuỳ theo chế độ đồ hoạ và bộ điều khiển màn hình, chiều rộng và chiều dài của ma trận này (được đo bằng số điểm) có thể là: 320 * 200, 640 * 200, 640 * 400, 640 * 480, 720 * 348, 800*600, 1024*768, 1024*1024 ...

3.2 Các tệp chứa trình điều khiển màn hình của Turbo C

Turbo C có rất nhiều trình điều khiển màn hình cho các bộ điều khiển màn hình khác nhau như CGA, EGA. Các tệp này có phần mở rộng là `.BGI`

3.3 Tệp tiêu đề `graphics.h`

Tệp `graphics.h` chứa các nguyên mẫu của các hàm đồ họa. Khi sử dụng các hàm đồ họa phải đặt chỉ thị sau đây lên đầu chương trình.

```
#include  <graphics.h>
```

Tệp `graphics.h` cũng còn chứa các hằng được các hàm đồ họa dùng đến.

3.4 Khởi động hệ đồ hoạ

Để khởi tạo đồ họa, bạn dùng hàm `initgraph()` như sau:

```
void far initgraph( int far *driver,int far *mode,char far *path);
```

trong đó:

`driver` là địa chỉ của số hiệu trình điều khiển đồ họa. Trong Turbo C, có thể chọn các trình điều khiển đồ họa trong bảng 7.1.

`mode` là địa chỉ của số hiệu kiểu điều khiển đồ họa. Bảng 7.2 liệt kê các kiểu đồ họa tương ứng với các điều khiển khác nhau.

`path:` mô tả đường dẫn tới nơi chứa tệp có chương trình điều khiển đồ họa. Nhớ là phải dùng hai dấu "\\" để phân tách các thư mục.

Giá trị	Tên hằng	Ghi chú
0	DETECT	Hệ thống tự phát hiện kiểu đồ họa cao nhất có thể được.
1	CGA	Sử dụng trình điều khiển của màn hình CGA
2	MCGA	như CGA
3	EGA	256 K bộ nhớ trên board EGA
4	EGA64	64 K bộ nhớ trên board EGA
5	EGAMONO	
6	IBM8514	
7	HERCMONO	Hercules
8	ATT400	
9	VGA	
10	PC3270	

Bảng 7.1 Các giá trị khác nhau của driver

Kiểu điều khiển	Kiểu đồ họa	Giá trị	Độ phân giải	Bảng màu	Số trang
CGA	CGAC0	0	320x200	C0	1
	CGAC1	1	320x200	C1	1
	CGAC2	2	320x200	C2	1
	CGAC3	3	320x200	C3	1
	CGAHI	4	640x200	2color	1
MCGA	MCGAC0	0	320x200	C0	1
	MCGAC1	1	320x200	C1	1
	MCGAC2	2	320x200	C2	1
	MCGAC3	3	320x200	C3	1
	MCGAMED	4	640x200	2color	1
	MCGAHI	5	640x480	2color	1
EGA	EGALO	0	640x200	16color	4
	EGAHI	1	640x350	16color	2
EGA64	EGA64LO	0	640x200	16color	1
	EGA64HI	1	640x350	4color	1
EGA-MONO	EGAMONOHI	3	640x350	2color	1*
	EGAMONOHI	3	640x350	2color	2**
HERC	HERCMONOHI	0	720x348	2color	2
ATT400	ATT400C0	0	320x200	C0	1
	ATT400C1	1	320x200	C1	1
	ATT400C2	2	320x200	C2	1
	ATT400C3	3	320x200	C3	1
	ATT400MED	4	640x200	2color	1
	ATT400HI	5	640x400	2color	1
VGA	VGALO	0	640x200	16color	2
	VGAMED	1	640x350	16color	2
	VGAHI	2	640x480	16color	1
PC3270	PC3270HI	0	720x350	2color	1
IBM8514	IBM8514HI	1	1024x760	256color	
	IBM8514LO	0	640x480	256color	

Bảng 7.2 Các giá trị khác nhau của mode

Chương trình sau sẽ khởi tạo đồ họa rồi vẽ một đường thẳng và một vòng tròn.

Bài tập mẫu 7.1

```
/*cirline.c*/
/*vẽ một đường thẳng và một vòng tròn*/
#include <stdio.h>
#include <graphics.h>
void main()    {
        int driver, mode;
        int x1 = 0, y1 = 0;
        int x2 = 199, y2 = 199;/*Đường thẳng*/
        int xC = 100, yC = 100; /*Tâm*/
        int radius = 90;          /*Bán kính*/
        driver = CGA;
        mode   = CGAC0;
        /*Khởi tạo đồ họa*/
        initgraph(&driver,&mode,"c:\\tc\\bgi");
        line(x1,y1,x2,y2);
        circle(xC,yC,radius);
        getche();
        /*Kết thúc việc sử dụng các hàm đồ họa*/
        closegraph();
        }
```

3.5 Lỗi đồ hoạ

Để biết lỗi đồ họa, hãy dùng hàm `graphresult()`:

```
int far graphresult(void);
```

Hàm này trả về mã lỗi xảy ra khi ta khởi tạo chế độ đồ họa.

Sau khi biết được mã lỗi, muốn biết các thông báo lỗi tương ứng hãy chuyển mã lỗi sang hàm `grapherrormsg()`.

```
char far * grapherrormsg(int codeerror);
```

Hàm này trả về xâu ký tự mô tả lỗi đồ họa vừa xảy ra. Bảng 7.3 mô tả các mã lỗi do hàm `graphresult()` trả về và các thông báo tương ứng do hàm

`grapherrormsg()` đưa ra.

Mã lỗi	Tên hằng	Các thông báo tương ứng
0	grOk	No error
-1	grNoInitGraph	(BGI) graphics not installed (use initgraph)
-2	grNotDetected	Graphics hardware not detected
-3	grFileNotFound	Device driver file not found
-4	grInvalidDriver	Invalid device driver file
-5	grNoLoadMem	Not enough memory toload driver
-6	grNoScanMem	Out of memoryi n scanfill
-7	grNoFloodMem	Out of memory in floodfill
-8	grFontNotFound	Font file not found
-9	grNoFontMem	Not enough memory to load font
-10	grInvalidMode	invalid graphics mode for selected driver
-11	grError	Graphics error
-12	grIOerror	GraphicsI/Oerror
-13	grInvalidFont	Invalidfontfile
-14	grInvalidFontNum	Invalidfontnumber
-15	grInvalidDeviceNum	Invaliddevicenumber
-18	grInvalidVersion	Invalidversionnumber

Bảng 7.3 Mã lỗi của hàm graphresult()

Chương trình sau đây minh họa cách sử dụng các hàm `graphresult()` và `grapherrormsg()`.

Bài tập mẫu 7.2

```
/*error.c*/
/*vẽ một đường thẳng và một vòng tròn*/
#include <stdio.h>
#include <graphics.h>
void main()    {
        int driver, mode, error;
        int x1 = 0, y1 = 0;
        int x2 = 199, y2 = 199;/*Đường thẳng*/
```

```
int xC = 100, yC = 100; /*Tâm*/
int radius = 90;        /*Bán kính*/
driver = CGA;
mode   = CGAC0;
/*Khởi tạo đồ họa*/
initgraph(&driver,&mode,"c:\\tc\\bgi");
error = graphresult();
if (error)  /*có lỗi*/ {
printf("Ma loi = %d",error);
/*đưa ra thông báo lỗi tương ứng*/
puts(grapherrormsg(error));
}
else   {
line(x1,y1,x2,y2);
circle(xC,yC,radius);
getche();
/*Kết thúc việc sử dụng các hàm đồ họa*/
closegraph();
}
}
```

3.6 Khởi tạo đồ hoạ tự động

Gán hằng `DETECT` cho đối số `driver` của hàm `initgraph()` sẽ khởi tạo tự động kiểu đồ họa ở chế độ "mịn" nhất cho phép. Trong trường hợp này để xác định độ phân giải của màn hình, sử dụng các hàm `getmaxx()` và `getmaxy()`.

```
int far getmaxx();
```

cho biết số điểm ảnh trên bề rộng của màn hình đồ họa hiện tại (tọa độ x lớn nhất)

```
int far getmaxy();
```

cho biết số điểm ảnh trên bề cao của màn hình đồ họa hiện tại (tọa độ y lớn nhất).

Chương trình sau đây minh họa cách dùng hằng DETECT.

Bài tập mẫu 7.3

```
/*border.c*/
/*vẽ viền quanh màn hình*/
/*làm việc trong tất cả các kiểu đồ họa*/
#include <graphics.h>
void main()    {
        int driver = DETECT;
        int mode;
        int error;
        int maxx, maxy;
        int left, top, right, bot;
        initgraph(&driver,&mode,"c:\\tc\\bgi");
        error = graphresult();
        if (error)  /*có lỗi*/{
        printf("Ma loi = %d",error);
        /*đưa ra thông báo lỗi tương ứng*/
        puts(grapherrormsg(error));
        }
        else   {
        maxx = getmaxx();
        maxy = getmaxy();
        left = top = 0;
        right = maxx;
        bot   = maxy;
        /*vẽ đường viền quanh màn hình*/
        rectangle(left,top,right,bot);
        getche();
        closegraph();
        }
        }
```

Có thể khởi tạo tự động kiểu đồ họa bằng hàm `detectgraph()`. Hàm này trả về `driver` và `mode` tốt nhất tương ứng với màn hình bạn đang dùng.

Bài tập mẫu 7.4

```
/*detect1.c*/
/*tìm driver và mode đang dùng*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
/* tên các loại card đồ họa */
char *dname[] = { "requests detection",
                  "a CGA",
                  "an MCGA",
                  "an EGA",
                  "a 64K EGA",
                  "a monochrome EGA",
                  "an IBM 8514",
                  "a Hercules monochrome",
                     "an AT&T 6300 PC",
                  "a VGA",
                  "an IBM 3270 PC"
                };
int main(void)
{
int gdriver, gmode, errorcode;
   /*Phát hiện card đồ họa của máy */
   detectgraph(&gdriver, &gmode);
   /* Đọc kết quả của lời gọi detect */
   errorcode = graphresult();
   if (errorcode != grOk)  /* có lỗi*/   {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1); /*kết thúc có lỗi*/
```

```
    }
    /* Hiển thị các thông tin đã được phát hiện */
    clrscr();
  printf("You have %s video display card.\n", dname[gdriver]);
  printf("Press any key to halt:");
  getch();
}
```

Chú ý

Trong thư viện các hàm đồ họa, chỉ có hàm detectgraph() có khả năng được gọi trước hàm initgraph(). Các hàm khác phải được gọi sau khi đã khởi động đồ họa.

4. ĐẶT MẪU VÀ MẪU TÔ CHO CÁC ĐƯỜNG VÀ CÁC HÌNH

4.1 Mẫu tô cho đường

Hàm `setlinestyle()` để thiết lập kiểu (style), mẫu (pattern) và bề dày (thickness) của đường. Nguyên mẫu của hàm như sau

```
void far setlinestyle (style,pattern,thick);
```

Trong đó

Tham số	Giải thích
`int style`	Kiểu của đường: nét liền, chấm, gạch.
`int pattern`	Mẫu tô khi style=4
`int thick;`	Độ dày của đường

Giá trị của `style` có thể là:

Giá trị	Tên hằng	Ghi chú
0	SOLID_LINE	Đường nét đậm
1	DOTTED_LINE	Đường nét chấm
2	CENTER_LINE	Đường chấm gạch
3	DASHED_LINE	Đường nét đứt
4	USERBIT_LINE	Kiểu do người dùng định nghĩa

Chú ý

Nếu sử dụng USERBIT_LINE cho đối số `style` thì đối số `pattern` sẽ ấn định một giá trị 16 bits cho điểm nào bật điểm nào tắt. Ví dụ mẫu `0xFF00` sẽ bật nửa đầu của đoạn và tắt nửa sau.

Tham số `thick` có thể nhận một trong hai giá trị sau:

Giá trị	Tên hằng	Ý nghĩa
1	NORM_WIDTH	Dày 1 điểm ảnh
3	THICK_WIDTH	Dày 3 điểm ảnh

Nếu muốn biết kiểu (style), mẫu tô (pattern) và độ dày (thick) đang được dùng để vẽ các đường hãy sử dụng hàm:

```
void far getlinesettings(struct linesettingstype far *lineinfo);
```

`getlinesettings()` chứa các thông tin này trong biến cấu trúc

`struct linesettingstype *lineinfo`

trong đó cấu trúc `linesettingstype` được định nghĩa như sau:

```
struct linesettingstype {
  int       linestyle;
  unsigned  upattern;
  int       thickness;
};
```

4.2 Đặt màu nền

Sử dụng hàm `setbkcolor()`:

```
void far setbkcolor(int color);
```

4.3 Đặt màu tô

Để xác định màu cho đường, hãy dùng hàm `setcolor()`:

```
void far setcolor(int color);
```

Số màu và tên màu sử dụng tùy thuộc vào kiểu đồ họa. Sau đây là màu trong từng palette của kiểu CGA:

Palette	Màu 0	Màu 1	Màu 2	Màu 3
CGAC0	BLACK	LIGHTGREEN	LIGHTRED	YELLOW
CGAC1	BLACK	LIGHT CYAN	LIGHTMAGENTA	WHITE
CGAC2	BLACK	GREEN	RED	BROWN
CGAC3	BLACK	CYAN	MAGENTA	LIGHTGRAY

Với các màn hình EGA và VGA, ta có thể chọn nhiều màu hơn. Các màu này đã được định nghĩa trong tệp tiêu đề `graphics.h`. Sau đây là 16 hằng cho EGA

Tên hằng	Giá trị
EGA_BLACK	0
EGA_DARKGRAY	56
EGA_BLUE	1
EGA_LIGHTBLUE	57
EGA_GREEN	2
EGA_LIGHTGREEN	58
EGA_CYAN	3
EGA_LIGHTCYAN	59
EGA_RED	4
EGA_LIGHTRED	60
EGA_MAGENTA	5
EGA_LIGHTMAGENTA	61
EGA_LIGHTGRAY	7
GA_YELLOW	62
EGA_BROWN	20
EGA_WHITE	63

Chương trình sau đây minh họa cách dùng các hàm đã nêu.

Bài tập mẫu 7.5

```
/*lincolor.c*/
/*Vẽ đường tròn và đường thẳng bằng hai màu khác nhau*/
#include <graphics.h>
#define IGNORED 0
#define CGAC0_LIGHTGREEN 1
```

```
#define CGAC0_LIGHTRED 2
void main() {
        int driver, mode;
        int x1 = 0, y1 = 0;
        int x2 = 199, y2 = 199;
        int xC = 100, yC = 100;
        int radius = 90;
        driver = CGA;
         mode = CGAC0; /*palette 0*/
         initgraph(&driver,&mode,"c:\\tc\\bgi");
         setlinestyle(DASHED_LINE,IGNORED,THICK_WIDTH);
         setcolor(CGAC0_LIGHTGREEN);
         line(x1,y1,x2,y2);
         setcolor(CGAC0_LIGHTRED);
         circle(xC,yC,radius);
         getche();
        closegraph();
       }
```

Để xác định màu tô đường hiện tại sử dụng hàm `getcolor()`:

```
int far getcolor();
```

Hàm này trả về màu tô hiện tại (được thiết lập bởi `setcolor()`)

4.4 Đặt dải màu

Chương trình dịch cho phép thay đổi diải màu ngầm định nhờ hàm

```
void setpalette(colornum,  color);
```

Trong đó:

Các tham số	Giải thích
`int colornum`	Số hiệu màu bị thay thế
`int color`	Màu dùng để thay thế

Người ta định nghĩa dải màu ngầm định được định nghĩa như sau:

```
#define MAXCOLORS 15
struct palettetype {
        unsigned char size;
        unsigned char color;
};
```

Ở đây, `size` là số lượng màu trong palette, `colors` là mảng màu chứa các giá trị của các với chỉ số chạy từ 0 tới `size-1`. Bảng sau đây liệt kê giá trị các màu quen thuộc:

Tên hằng	Giá trị
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Nhờ hàm `setpalette()` ta có thể gán cho màu thứ `colornum` giá trị màu mới là `color`. Câu lệnh

```
setpalette(0,LIGHTCYAN);
```

sẽ đặt lại màu đầu tiên trong dải màu thành xanh lơ nhạt. Các màu khác không bị ảnh hưởng bởi câu lệnh này.

Để nhận được thông tin về dải màu hiện tại, sử dụng hàm:

```
void getpalette(struct palettetype *palette).
```

4.5 Đặt mẫu tô cho các hình

Để thiết lập màu và mẫu tô cho các hình sử dụng hàm `void far setfillstyle(pattern, color)`, trong đó

Tham số	Giải thích
`int pattern`	Mẫu tô
`int color`	Màu

Biến `pattern` có thể nhận một trong các hằng đã được định nghĩa trong tệp `graphics.h` như sau:

Tên hằng	Giá trị	Ý nghĩa
EMPTY_FILL	0	màu nền
SOLID_FILL	1	màu đặc
LINE_FILL	2	đường ngang ----
LTSLASH_FILL	3	gạch chéo mảnh ///
SLASH_FILL	4	gạch chéo dày ///
BKSLASH_FILL	5	chéo ngược dày \\\
LTBKSLASH_FILL	6	chéo ngược mảnh \\\
HATCH_FILL	7	sọc dừa thưa
XHATCH_FILL	8	sọc dừa dày
INTERLEAVE_FILL	9	đường xen kẽ
WIDE_DOT_FILL	10	chấm thưa
CLOSE_DOT_FILL	11	chấm dày
USER_FILL	12	mẫu tự tạo

Biến `color` được chọn theo theo danh sách đã liệt kê trong phần `setcolor()`. Nếu sử dụng giá trị không hợp lệ cho `pattern` và `color` thì hàm `graphresult()` sẽ trả về mã lỗi là -11.

Mẫu tô và mầu tô sẽ được sử dụng trong các hàm:

`pieslice(),fillpoly(),bar(),bar3d(),floodfill();`

Xét chương trình ví dụ sau đây:

Bài tập mẫu 7.6

```
/*floodoo.c*/
/*Tô màu vòng tròn bằng floodfill()*/
#include <graphics.h>
 #define xC 300
 #define yC 175
 #define RADIUS 150
void main() {
        int gdriver = EGA, gmode = EGAHI;
        initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
        setcolor(RED);/**/
        circle(xC,yC,RADIUS);
        /* select fill color */
        setfillstyle(SOLID_FILL, BLUE);
        floodfill(xC,yC,RED);/**/
        getch();
        closegraph();
 }
```

Hàm `floodfill()` được khai báo như sau

```
void far floodfill(x, y, border);
```

Trong đó

Tham số	Giải thích
`int x`	Toạ độ điểm nằm trong hình cần tô.
`int y`	
`int border`	Màu giới hạn của biên hình cần tô.

Hàm `floodfill()` tô một vùng chứa điểm `(x,y)` và được giới hạn bởi một đường biên có màu là `border`. Vùng được điền bằng màu và mẫu tô hình hiện tại.

Điểm có tọa độ `(x,y)` được gọi là "điểm nhìn", được dùng để xác định miền được tô. Nếu điểm này nằm trong đường biên `border` vùng giới hạn bên trong được tô. Trái lại miền nằm ngoài đường biên được tô.

Để xác định màu tô, mẫu tô các hình, sử dụng hàm

```
void far getfillsettings (struct fillsettingstype far *fillinfo);
```

Hàm `getfillsettings()` lưu các thông tin về màu tô và mẫu tô hiện thời vào trong biến cấu trúc được trỏ bởi `fillinfo`. Cấu trúc `fillsettingstype` được định nghĩa như sau:

```
struct fillsettingstype {
  int  pattern; /* mẫu tô hiện tại */
  int  color;   /*màu tô hiện tại */
  };
```

5. CÁC HÀM VẼ VÀ TÔ

5.1 Vẽ điểm ảnh

Hàm `putpixel()` dùng để vẽ từng điểm ảnh. Hàm này có dạng:

```
void far putpixel(x, y, color);
```

Trong đó:

Tham số	Giải thích
`int x`	Toạ độ điểm cần tô màu
`int y`	
`int color`	Màu tô cho điểm

Sau đây là chương trình dùng hàm `putpixel()` để vẽ đồ thị hàm sin.

Bài tập mẫu 7.7

```
/*plotsin.c*/
/*Vẽ đồ thị hàm sin bằng putpixel()*/
#include <graphics.h>
#include <math.h>
/*Thư viện các hàm tính toán*/
#define CGAC0_YELLOW 3
void main()   {
        int driver, mode;
        double angle, sinofA;
        int x, y;
```

```
        driver = CGA;
        mode = CGAC0;
        /*khởi tạo đồ họa*/
        initgraph(&driver,&mode,"c:\\tc\\bgi");
        line(0,100,200,100);
        for (x = 0; x <200; x++){
                angle = ((double)x / 200)*(2 * M_PI);
                sinofA = sin(angle);
                y = 100 - 100 * sinofA;
                putpixel(x,y,CGAC0_YELLOW);
                }
         getche();
         closegraph();
        }
```

5.2 Lấy màu điểm ảnh

Hàm

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu màu của điểm ảnh có tọa độ (x,y). Nếu điểm ảnh này chưa được tô màu bởi các hàm vẽ hoặc `putpixel()`, nghĩa là có cùng màu với màu nền thì hàm trả về giá trị bằng 0. Vì vậy có thể dùng hàm `getpixel()` theo cách dưới đây để xác định các nét vẽ trên màn hình đồ họa và vẽ ra giấy.

```
if (getpixel(x, y) !=0)        {
                /*điểm (x,y) được vẽ*/
                }
```

5.3 Vẽ đoạn thẳng

- Hàm

```
void line(x1, y1, x2, y2);
```

Trong đó

Tham số	Giải thích
`int x1, y1`	Toạ độ điểm đầu
`int x2, y2`	Toạ độ điểm cuối

vẽ một đoạn thẳng nối hai điểm (x1,y1) và (x2, y2). Đoạn thẳng này được tô bằng màu tô và mẫu tô hiện tại (được thiết lập bằng hàm setlinestyle() hoặc ngầm định).

Cũng như màn hình văn bản, trên màn hình đồ họa có một con trỏ màn hình.

- Hàm

```
void linerel (int dx, int dy);
```

vẽ một đoạn thẳng từ vị trí hiện tại của con trỏ màn hình (ký hiệu là (x,y)) đến điểm có tọa độ (x+dx, y+ dy). Con trỏ màn hình được di chuyển đến vị trí mới (x+dx,y+dy).

- Hàm

```
void lineto(int x, int y);
```

vẽ một đoạn thẳng từ vị trí hiện tại của con trỏ màn hình đến điểm (x, y). Sau hàm này, con trỏ màn hình có tọa độ mới là (x, y).

- Hàm

```
void  moveto(int x, int y);
```

sẽ di chuyển con trỏ màn hình tới vị trí mới (x, y).

Chương trình dưới đây tạo 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).

Bài tập mẫu 7.8

```
#include <graphics.h>
void main()    {
        int driver = DETECT, mode;
        initgraph(&driver, &mode, "c:\\tc\\bgi");
        setbkcolor(GREEN);
        setcolor(YELLOW);
        moveto(230,100);
        line(20,20,620,20);
        linerel(-300,80);
        lineto(620,180);
        lineto(620,20);
        getche();
```

```
        closegraph();
        }
```

5.4 Vẽ và tô màu hình chữ nhật

Để vẽ một hình chữ nhật, dùng hàm:

```
void rectangle(left, top, right, bottom);
```

Trong đó

Tham số	Giải thích
`int left, top`	Toạ độ đỉnh trên bên trái
`int right, bottom`	Toạ độ đỉnh dưới bên phải

Hàm `rectangle()` sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Tọa độ đỉnh trên bên trái của hình chữ nhật là (`left, top`) và đỉnh dưới bên phải là (`right, bottom`). Màu vẽ được xác định bằng `setcolor()`. Mẫu vẽ xác định bằng `setlinestyle()`.

Để tô một hình chữ nhật sử dụng hàm:

```
void far bar(left, top, right, bottom);
```

Trong đó

Tham số	Giải thích
`int left, top`	Tọa độ đỉnh trên bên trái
`int right, bottom`	Tọa độ đỉnh dưới bên phải

Hàm `bar()` tô hình chữ nhật có các cạnh song song với các cạnh màn hình theo mẫu và màu tô hiện tại (được thiết lập bằng `setfillstyle()`). Các thiết lập của `setcolor()` và `setlinestyle()` không ảnh hưởng đến `bar()`.

Xét chương trình ví dụ sau:

Bài tập mẫu 7.9

```
/*barrect.c*/
/*Vẽ và tô hình chữ nhật với các mẫu tô khác nhau*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void) {
    /* khởi tạo đồ họa tự động*/
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;
   initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
   /*kiểm tra lỗi khởi tạo */
    errorcode = graphresult();
    if (errorcode != grOk)  /* cã lçi*/   {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* kết thúc chương trình*/
    }
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    /*dùng cho hàm rectangle(), không ảnh hưởng đến bar() */
    setlinestyle(SOLID_LINE,1,3);
    /* sử dụng các mẫu tô khác nhau*/
    for (i=SOLID_FILL; i<USER_FILL; i++)   {
       /* đặt lại mẫu tô */
        setfillstyle(i, getmaxcolor());
       /*vẽ và tô màu hình chữ nhật*/
       rectangle(midx-50, midy-50, midx+50,  midy+50);
       bar(midx-50, midy-50, midx+50,  midy+50);
       getch();
    }
   closegraph();
   }
```

5.5 Vẽ và tô màu đa giác

Muốn vẽ một đường gấp khúc đi qua n điểm: (x_1,y_1), (x_2,y_2), . . . , (x_n,y_n) trước hết chúng ta phải đưa các tọa độ vào một mảng `points` nào đó có kiểu

nguyên. Nói một cách chính xác hơn, cần gán x_1 cho `points[0]`, y_1 cho `points[1]`, x_2 cho `points[2]`, y_2 cho `points[3]` v.v. . Sau đó viết lời gọi hàm :

```
drawpoly(n, points);
```

Khi điểm cuối (x_n, y_n) trùng với điểm đầu (x_1, y_1) thì chúng ta nhận được một đường gấp khúc khép kín.

Giả sử `points` là mảng đã nói trong phần trên, khi đó lời gọi hàm

```
fillpoly(n, points);
```

sẽ có tác dụng vẽ và tô màu một đa giác có đỉnh là các điểm

(x_1, y_1), (x_2, y_2) ... (x_n, y_n)

Bài tập mẫu 7.10

```
/*drawpoly.c*/
/*vẽ một đa giác nhờ hàm drawpoly()*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
   /* tự động khởi tạo đồ họa*/
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* mảng chứa tọa độ đa giác*/
   int poly[10];

   /* khởi tạo đồ họa*/
   initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

   /* kiểm tra lỗi khởi tạo đồ họa*/
   errorcode = graphresult();
   if (errorcode != grOk)/* một lỗi xảy ra */   {
```

```
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
  /* có lỗi: kết thúc chương trình*/
      exit(1);
  }

  maxx = getmaxx();
  maxy = getmaxy();

  poly[0] = 20;          /* tọa độ thứ nhất */
  poly[1] = maxy / 2;

  poly[2] = maxx - 20; /*thứ hai */
  poly[3] = 20;

  poly[4] = maxx - 50; /* thứ ba */
  poly[5] = maxy - 20;

  poly[6] = maxx / 2;  /*thứ năm */
  poly[7] = maxy / 2;
/* hàm draw không tự động đóng kín đa giác. Tự chúng ta phải làm điều đó*/
  poly[8] = poly[0];
  poly[9] = poly[1];

  /* vẽ đa giác*/
  drawpoly(5, poly);
  getch();
  closegraph();
}
```

Bài tập mẫu 7.11

```
/*fillpoly.c*/
/*vẽ và tô màu đa giác bằng các mẫu tô khác nhau*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void){
   /* khởi tạo đồ họa tự động */
   int gdriver = DETECT, gmode, errorcode;
   int i, maxx, maxy;

   /* mảng chứa đa giác */
   int poly[8];

   /* khởi động đồ họa*/
   initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

   /* kiểm tra lỗi khởi tạo */
   errorcode = graphresult();
   if (errorcode != grOk) /* một lỗi xảy ra */   {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
    /* thoát chương trình*/
   exit(1);
   }
   maxx = getmaxx();
   maxy = getmaxy();
   /* tọa độ đỉnh thứ nhất*/
   poly[0] = 20;
   poly[1] = maxy / 2;
```

```
   /* đỉnh thứ hai */
   poly[2] = maxx - 20;
   poly[3] = 20;
   /* đỉnh thứ ba */
   poly[4] = maxx - 50;
   poly[5] = maxy - 20;
   /*tọa độ đỉnh thứ tư*/
   poly[6] = maxx / 2;
   poly[7] = maxy / 2;
    /*khác với drawpoly(), fillpoly() tự động khép kín đa giác*/

   /*lặp qua các mẫu tô khác nhau */
   for (i=EMPTY_FILL; i<USER_FILL; i++)  {
      /* đặt mẫu tô mới */
      setfillstyle(i, getmaxcolor());

      /* vẽ  và tô đa giác */
      fillpoly(4, poly);
      getch();
   }
       closegraph();
}
```

5.6 Vẽ đường tròn và hình tròn

- Hàm

```
void far arc(x,y, stangle, ndangle,radius);
```

vẽ một cung tròn bằng màu vẽ hiện thời (được xác định bằng `setcolor()`).

- Hàm

```
void far circle(x, y, radius);
```

vẽ một đường tròn bằng màu vẽ hiện thời (được xác định bằng `setcolor()`).

- Hàm

```
void far pieslice(x, y, stangle,endangle,radius);
```

vẽ một hình quạt tròn bằng màu vẽ hiện thời, sau đó tô nó bằng mẫu tô và màu tô hiện dùng (do `setfillstyle()` qui định).

Ý nghĩa các tham số được mô tả trong bảng sau đây:

Các tham số	Giải thích
`int x,y`	Tâm của cung tròn, đường tròn, quạt tròn
`int stangle`	Góc bắt đầu (tính theo độ) của cung tròn, quạt tròn
`int endangle`	Góc kết thúc
`int radius`	Bán kính của cung tròn, đường tròn, quạt tròn.

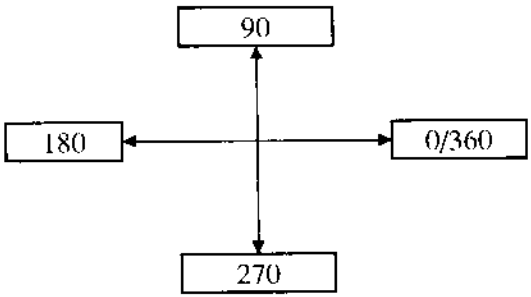
Cung tròn và quạt tròn được tính từ góc bắt đầu đến góc kết thúc.

Nếu `stangle=0` và `endangle = 360` thì lời gọi tới `arc()` vẽ một vòng tròn đầy đủ.

Tham số `style` (trong `setlinestyle()`) không làm ảnh hưởng tới các cung, quạt và hình tròn. Chỉ có tham số `thick` có ảnh hưởng đến độ dày của đường.

Nếu bạn sử dụng màn hình CGA ở chế độ phân giải cao các hàm này có thể không cho kết quả theo như mô tả.

Cách tính góc đối với `arc()`, `circle()`, và `pieslice()` theo chiều ngược chiều kim đồng hồ.



Bài tập mẫu 7.11

```
/*arc.c*/
/*Vẽ một cung tròn */
#include <graphics.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void){
    /* khởi tạo đồ họa tự động*/
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135;
   int radius = 100;
   initgraph(&gdriver, &gmode, " c:\\tc\\bgi ");
   /* kiểm tra lỗi khởi tạo*/
   errorcode = graphresult();
   if (errorcode != grOk) /*có lỗi */  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);     /* thoát chương trình*/
   }
   /*tính toán các thông số cho arc()*/
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());
   /* vẽ cung*/
   arc(midx, midy, stangle, endangle, radius);
   getch();
   closegraph();
 }
```

Bài tập mẫu 7.12

```
/*circle.c*/
/*Vẽ một đường tròn*/
#include <graphics.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>
int main(void){
   /* khởi tạo đồ họa tự động*/
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int radius = 100;
   initgraph(&gdriver, &gmode, " c:\\tc\\bgi ");
   /* kiểm tra lỗi*/
   errorcode = graphresult();
   if (errorcode != grOk)  /* có lỗi*/    {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
   getch();
   exit(1);
   }
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* vẽ một đường tròn*/
   circle(midx, midy, radius);
   getch();
   closegraph();
 }
```

Bài tập mẫu 7.13

```
/*pieslice.c*/
/*Vẽ và tô một quạt tròn*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(void){
   /*khởi tạo đồ họa ở chế độ tốt nhất */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135, radius = 100;
   initgraph(&gdriver, &gmode, " c:\\tc\\bgi ");
   /* kiểm tra lỗi*/
   errorcode = graphresult();
   if (errorcode != grOk)  /* có lỗi*/   {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
   }
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* đặt mẫu tô, màu tô*/
   setfillstyle(EMPTY_FILL, getmaxcolor());
   pieslice(midx, midy, stangle, endangle, radius);
   getch();
   closegraph();
   return 0;
}
```

5.7 Vẽ hình ellipse

Các hàm sau đây được dùng để vẽ và tô màu cho một hình ellipse hay một cung ellipse:

- Hàm

```
void far ellipse(x,y,stangle,endangle,xradius, yradius);
```

vẽ một cung ellipse bằng màu vẽ hiện thời (được xác định bằng `setcolor()`).

- Hàm

```
void far fillellipse(x,y,xradius, yradius);
```

vẽ một hình ellipse, sau đó tô ellipse này bằng mẫu tô và màu tô hiện tại (do `setfillstyle()` qui định).

- Hàm

```
void far sector(x,y,stangle, endangle, xradius, yradius);
```

vẽ và tô một hình quạt ellipse theo màu vẽ, màu tô và mẫu tô hiện thời (được xác định bởi `setlinestyle()` và `setfillstyle()`).

Cung ellipse và quạt ellipse được tính từ góc bắt đầu đến góc kết thúc.

Nếu `stangle=0` và `endangle=360` thì lời gọi tới `ellipse()` sẽ vẽ một đường ellipse đầy đủ.

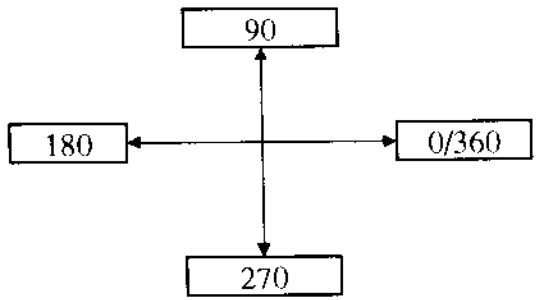
Ý nghĩa các tham số được mô tả trong bảng sau đây:

Các tham số	Giải thích
`int x,y`	tọa độ tâm của ellipse
`int xradius`	bán kính ngang
`int yradius`	bán kính dọc
`int stangle`	góc bắt đầu
`int endangle`	góc kết thúc

Tham số `style` (strong `setlinestyle()`) không làm ảnh hưởng tới các cung, quạt và hình ellipse. Chỉ có tham số thick có ảnh hưởng đến độ dày của đường.

Nếu bạn sử dụng màn hình CGA ở chế độ phân giải cao các hàm này có thể không cho kết quả theo như mô tả.

Cách tính góc đối với `ellipse()` và `sector()` theo chiều ngược chiều kim đồng hồ.



Xét một số chương trình ví dụ sau:

Bài tập mẫu 7.14

```
/*ellipse.c*/
/*Vẽ một ellipse*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
  /* khởi tạo đồ họa ở chế độ tốt nhất*/
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 0, endangle = 360;
    int xradius = 100, yradius = 50;
    initgraph(&gdriver, &gmode, " c:\\tc\\bgi ");
  /*kiểm tra lỗi*/
    errorcode = graphresult();
    if (errorcode != grOk)/*có lỗi*/{
     printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);
    }
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());
    /* vẽ một ellipse*/
    ellipse(midx, midy, stangle, endangle, xradius, yradius);
    getch();
    closegraph();
  }
```

Bài tập mẫu 7.15

```
/*pllipse.c*/
/*Vẽ và tô một hình ellipse*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy, i;
  int xradius = 100, yradius = 50;

  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  errorcode = graphresult();
  if (errorcode != grOk) /*có lỗi*/{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  /* sử dụng các mẫu tô khác nhau*/
  for (i = EMPTY_FILL; i < USER_FILL; i++) {
    /* đặt mẫu tô mới */
    setfillstyle(i, getmaxcolor());
    /*vẽ và tô ellipse bằng mẫu tô hiện tại*/
   fillellipse(midx, midy, xradius, yradius);
    getch();
  }
 closegraph();
 }
```

Bài tập mẫu 7.16

```
/*sector.c*/
/*Vẽ và tô một ellipse bằng các mẫu tô khác nhau*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;
   int stangle = 45, endangle = 135;
   int xrad = 100, yrad = 50;
   initgraph(&gdriver, &gmode, " c:\\tc\\bgi ");
   errorcode = graphresult();
   if (errorcode != grOk)  /* có lỗi */   {
     printf("Graphics error: %s\n",grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);
     }
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   /* dùng các mẫu tô khác nhau*/
   for (i=EMPTY_FILL; i<USER_FILL; i++) {
         /* set the fill style */
         setfillstyle(i, getmaxcolor());
         /*vẽ một quạt ellipse*/
         sector(midx, midy, stangle, endangle, xrad, yrad);
         getch();
   }
   closegraph();
 }
```

5.8 Vẽ hình hộp

- Hàm

```
void far bar3d(left, top,right, bottom,depth, topflag);
```

vẽ và tô một hình hộp chữ nhật. Các đường bao quanh được vẽ bằng mẫu vẽ và màu xác lập bởi `setlinestyle()` và `setcolor()`. Hình hộp được tô bằng màu và mẫu tô xác lập bởi `setfillstyle()`.

Ý nghĩa của các tham số cho trong bảng sau:

Các tham số	Giải thích
`int depth`	chiều xâu của hình hộp
`int topflag`	xác định hình hộp có nắp hay là không (mặt trên)
`int left, top`	góc trên bên trái của hình chữ nhật cơ sở (mặt ngoài)
`int right, bottom`	góc dưới bên phải của hình chữ nhật cơ sở (mặt ngoài)

Nếu `topflag` khác 0 hình hộp được vẽ có nắp. Trái lại hình hộp được vẽ không có nắp.

Sau đây là chương trình minh họa việc dùng `bar3d()` để tạo một đồ thị gồm nhiều khối ba chiều.

Bài tập mẫu 7.17

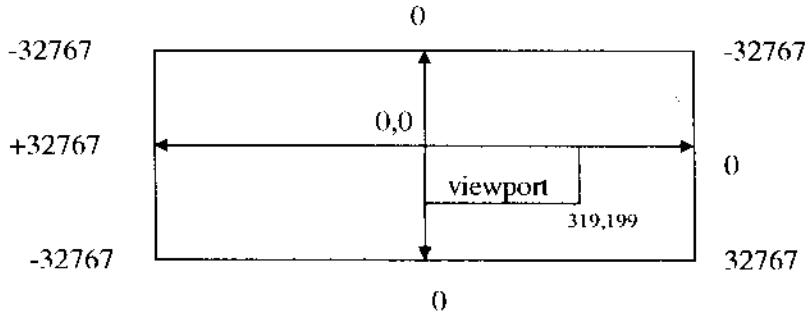
```
/*bargraph.c*/
/*Vẽ biểu đồ bằng bar3d()*/
#include <graphics.h>
#define N 10            /*số trị của đồ thị*/
#define BWIDTH 10       /*bề rộng mỗi khối*/
#define STEP 12                 /*khoảng cách các khối*/
#define DI (BWIDTH + STEP)
#define  SHFT 15                        /*giữa viền và khối 1*/
#define WIDTH ((N + 1)*DI)      /*bề rộng của đồ thị*/
#define LEFT 5                          /*mặt trái của đồ thị*/
#define DEPTH 3                         /*chiều xâu của khối*/
#define TOPFLAG 1                       /*có nắp*/
```

```
#define BOT 170                          /*đáy của đồ thị*/
#define TOP 5                            /*đỉnh của đồ thị*/
#define PPD (float)(BOT-TOP)/100 /*số điểm ảnh cho 1 đơn vị*/
void main()    {
        int driver, mode, error, j;
        /*dữ liệu ảnh cần thể hiện*/
        int data[N] = {41,47,54,62,63,59,75,83,89,96};
        driver = CGA;
        mode   = CGAC0;
        initgraph(&driver, &mode, "c:\\tc\\bgi");
        /*vẽ viền*/
        rectangle(LEFT, TOP, LEFT + WIDTH, BOT);
       /*vẽ khối*/
        for (j = 0; j < N; j++){
                setfillstyle(SOLID_FILL,1+j %3);/*ba màu*/
                bar3d(LEFT+SHFT+j*DI,BOT-data[j]*PPD,
                        LEFT+SHFT + j * DI + BWIDTH, BOT,
                DEPTH,TOPFLAG);
                }
         getche();
         closegraph();
        }
```

6. KHUNG NHÌN VÀ TỌA ĐỘ

Mỗi điểm ảnh trên màn hình đồ họa có một tọa độ (x,y) duy nhất. Trục x nằm ngang, trục y thẳng đứng. Điểm ảnh nằm ở góc trái trên của màn hình có tọa độ là (0,0). Các tọa độ x lớn dần khi đi sang phải. Các tọa độ y lớn dần khi đi xuống dưới. Qui ước này hơi ngược với qui ước trong toán học.

Tất cả các tọa độ trong đồ họa Turbo C là số nguyên, có thể âm hoặc dương. Như hình chữ nhật bên trong hình 7.1, cho thấy phần khả biến của màn hình đồ họa chỉ là một vùng giới hạn của toàn bộ bề mặt hay mặt phẳng logic mà chương trình có thể sử dụng để vẽ. Vùng này được gọi là khung nhìn (viewport). Các nét vẽ ra ngoài khung nhìn đều không thấy. Chỉ có các điểm nằm trong biên của khung nhìn mới được trình bày.



Hình 7.1 Khung nhìn định nghĩa vùng khả biến của bề mặt hay mặt phẳng theo tọa độ logic

Chương trình biên dịch Turbo C luôn lưu giữ toạ độ con trỏ màn hình hiện tại Current point (CP). CP là điểm mà thao tác đồ hoạ tiếp theo diễn ra. CP có thể ở điểm đầu của dòng, hay là vị trí xuất hiện văn bản. Để tìm vị trí của CP, hãy dùng 2 hàm `getx()`, `gety()`:

```
int far getx(void);
int far gety(void);
```

Sau khi khởi tạo đồ họa CP có vị trí là (0, 0). Để tìm các tọa độ lớn nhất hãy dùng các hàm:

```
int far getmaxx (void);
int far getmaxy(void);
```

Bốn hàm `getx()`, `gety()`, `getmaxx()`, `getmaxy()` cho phép kiểm tra các điểm có nằm trong khung nhìn hay không.

```
if (0 < getx()  && getx() < getmaxx() &&
    0 < gety()  && gety() < getmaxy())
        {/*CP nằm trong khung nhìn*/}
else
        {/*CP nằm ngoài khung nhìn*/}
```

Tổng số các trị tọa độ ngang là `getmaxx()+1`, còn số các trị tọa độ đứng là `getmaxy()+1`. Bằng cách dùng các hàm này để thay thế cho các giới hạn cố định, bạn có thể triển khai chương trình chạy tốt trên bất kỳ độ phân giải nào.

6.1 Thiết lập khung nhìn

Có hai lý do để thiết lập lại khung nhìn:

Thứ nhất, bạn có thể giới hạn việc vẽ các hình trong một phần của màn hình nhằm bảo vệ các đối tượng đồ họa trong các miền khác.

Thứ hai, điều đó cho phép dịch chuyển gốc (góc trái trên của khung nhìn) để cho nhiều công việc được thực hiện dễ dàng hơn.

Để thiết lập lại khung nhìn sử dụng hàm:

```
void far setviewport(left, top, right,bottom,clip);
```

Trong đó

```
int left, top, right, bottom,clip;
```

Hàm `setviewport()` xác lập một khung nhìn để vẽ các đối tượng đồ họa mới. Tọa độ của khung nhìn mới trên màn hình được cho bởi

Tham số	Giải thích
`int left, top`	Góc trái trên
`int right, bottom`	Góc phải dưới

Sau khi thiết lập lại khung nhìn, con trỏ màn hình đồ họa CP được chuyển đến tọa độ (0, 0) trong cửa sổ mới

Tham số `clip` xác định liệu mọi nét vẽ ra ngoài khung nhìn có bị cắt hay không. Nếu `clip` khác không, mọi nét vẽ vượt ra ngoài khung nhìn sẽ bị cắt.

Để xác định thông tin về khung nhìn hiện thời sử dụng hàm

```
void far getviewsettings (struct
                viewporttype far *viewport);
```

Hàm `getviewsettings()` lưu các thông tin về khung nhìn hiện thời trong một biến cấu trúc có kiểu được định nghĩa như sau:

```
struct viewporttype {
  int  left, top; /*tọa độ góc trên bên trái*/
  int  right,bottom; /*tọa độ góc dưới bên phải*/
  int  clip;
  };
```

6.2 Xóa khung nhìn

- Hàm

```
void far clearviewport(void);
```

xóa nội dung khung nhìn và chuyển con trỏ màn hình đồ họa về gốc của khung nhìn.

- Hàm

```
void far cleardevice(void);
```

xóa toàn bộ màn hình đồ họa và chuyển con trỏ màn hình đồ họa về toạ độ (0,0) của màn hình.

Để đưa khung nhìn thành toàn màn hình, hãy gọi hàm `graphdefaults()` được khai báo như sau:

```
void far graphdefaults(void);
```

tiếp theo là hàm `clearviewport()` nếu bạn muốn xóa màn hình. Cũng có thể thực hiện điều đó nhờ câu lệnh sau:

```
setviewport(0,0,getmaxx(),getmaxy(),1);
```

Xét chương trình ví dụ sau.

Bài tập mẫu 7.18

```
/*viewport.c*/
/*vẽ đường tròn và đường thẳng trong viewport*/
#include <graphics.h>
void main()    {
        int driver, mode;
        int left = 0, top = 0;
        int right = 200, bot = 150;/*khung nhìn*/
        int x1 = 0, y1 = 0;          /*đường thẳng*/
        int x2 = 199, y2 = 199;
        int xC = 100, yC = 100; /*tâm vòng tròn*/
        int clip =1;
        int radius = 90;    /*bán kính*/
        driver = CGA;
        mode   =CGAC0;
```

```
initgraph(&driver, &mode,"c:\\tc\\bgi");
setviewport(left,top,right,bot,clip);
rectangle(left,top,right,bot);/*vẽ khung của viewport*/
line(x1,y1,x2,y2);
circle(xC,yC,radius);
getche();
/*thiết lập lại viewport cả màn hình*/
graphdefaults();
clearviewport();
line(x1,y1,x2,y2);
circle(xC,yC,radius);
closegraph();
}
```

7. ASPECT RATIO

7.1 Định nghĩa

Aspect ratio được định nghĩa như sau:

$$\text{Aspectratio} = \frac{\text{Bề rộng màn hình}}{\text{Bề cao màn hình}}$$

Giá trị của aspect ratio cho biết tỉ số chiều cao và về rộng của điểm ảnh. Dựa vào aspect ratio chúng ta có thể thực hiện biến đổi các thông số của hình để thu được các hình tròn trịa và không bị méo.

Giá trị lý tưởng của Aspect ratio theo chuẩn NTSC là 4:3. Như vậy một màn hình lý tưởng có bội số của 4 điểm ảnh theo chiều rộng chia cho bội số của 3 các điểm ảnh theo chiều cao.

Các màn hình EGA 640*350 gần đạt được tỷ lệ này. Màn hình VGA 640*480 hoặc 800*600 đạt được tỷ lệ lý tưởng này.

Turbo C cung cấp hàm các hàm sau đây để làm việc với aspectratio:

```
void far getaspectratio(int far*xasp,int far*yasp);
void far setaspectratio(int xasp, int yasp);
```

Hàm `getaspectratio()` xác định giá trị của aspect ratio hiện tại và lưu vào trong hai biến nguyên `*xasp`, `*yasp`.

Hàm `setaspectratio()` thay đổi aspect ratio ngầm định của hệ thống đồ họa. Bảng sau trình bày ý nghĩa của các tham số:

Tham số	Hàm	Giải thích
xasp	getaspectratio()	Trỏ tới biến chứa aspect_ratio của tọa độ x.
	setaspectratio()	Giá trị mới cho thành phần aspect_ratio của tọa độ x
yasp	getaspectratio()	Trỏ tới biến chứa aspect_ratio của tọa độ y
	setaspectratio()	Giá trị mới của thành phần aspect_ratio của tọa độ y

Sau đây là một chương trình minh họa cách sử dụng hai hàm `getaspectratio()` và `setaspectratio()` để vẽ một vòng tròn với các hình dạng khác nhau.

Bài tập mẫu 7.19

```
/*aspect.c*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void){
   /*tự khởi động đồ họa*/
   int gdriver = DETECT, gmode, errorcode;
   int xasp, yasp, midx, midy;
   initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
   /* kiểm tra lỗi */ .
   errorcode = graphresult();
   if (errorcode != grOk)  /* có lỗi xảy ra*/    {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
   getch();
   exit(1); /* kết thúc chương trình */
   }
```

```
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  setcolor(getmaxcolor());
  /* lấy các thông số aspect_ratio hiện tại*/
  getaspectratio(&xasp, &yasp);
  /* vẽ một đường tròn thông thường */
  circle(midx, midy, 100);
  getch();
  /* xóa màn hình*/
  cleardevice();
  /* thay đổi aspect của chiều rộng, đường tròn bị dẹt */
  setaspectratio(xasp/2, yasp);
  circle(midx, midy, 100);
  getch();
  /* thay đổi aspect của chiều cao, đường tròn bị bóp lại*/
  cleardevice();
  setaspectratio(xasp, yasp/2);
  circle(midx, midy, 100);
  getch();
  closegraph();
}
```

8. VĂN BẢN TRONG ĐỒ HỌA

Turbo C có nhiều hàm để vẽ ký tự trong chế độ đồ họa. Các hàm này cho phép vẽ ký tự trên các ảnh đồ họa.

8.1 Các fonts

Font là một bộ ký tự theo một khuôn dạng trình bày chữ nào đó. Turbo C cung cấp nhiều fonts đồ họa rất thuận tiện cho việc xử lý văn bản. Các font này được chứa trong các tệp có phần mở rộng là .CHR. Có hai loại font đồ họa: bitmap và vector, font mặc định là bitmap tương tự như ảnh các ký tự thông thường trong ROM của máy PC. Tất cả các font khác đều là font vector, được cấu tạo gồm các đoạn giống như đa giác mà bạn đã vẽ.

Sử dụng font vector tỏ ra tiện lợi khi tăng kích thước ký tự. Do mẫu được tạo

lập bằng các đoạn, ký tự vẫn giữ được hình dạng tương đối khi phóng lên. Trái lại, với các font dạng bitmap khi phóng to các ký tự sẽ xuất hiện các rìa dọc theo đường chéo. Mỗi bit trở thành một điểm ảnh trong kích thước mặc định của font - được phát triển theo mọi hướng làm cho đường thẳng trở nên mập hơn và cao hơn. Tuy nhiên font bitmap cũng có một điểm lợi là nó thể hiện rất nhanh. Vì font vector được kết hợp từ các đoạn thẳng riêng biệt, do vậy nó cần nhiều thời gian để hiển thị hơn so với font bitmap. Turbo C có cài sẵn cả hai loại font. Hàm `settextstyle()` sau đây cho phép chọn font, chỉ định hướng(dọc hay ngang) và kích thước của font. Nguyên mẫu của hàm như sau:

```
void far settextstyle(font, direction, charsize);
```

Ý nghĩa của các tham số như sau

Tham số	Giải thích
`int font`	font cần dùng
`int direction`	Hướng
`int charsize`	kích thước

Có năm tệp chứa fonts tương ứng với năm hằng cho tham số `font` như sau:

Trị	Hằng	Tệp
0	DEFAULT_FONT	Cài sẵn
1	TRIPLEX_FONT	TRIP.CHR
2	SMALL_FONT	LITT.CHR
3	SANSSERIF_FONT	SANS.CHR
4	GOTHIC_FONT	GOTH.CHR

Tham số `direction` có thể nhận một trong hai giá trị:

Trị	Tên hằng	Giải thích
0	HORIZ_DIR	Từ trái qua phải
1	VERT_DIR	Từ dưới đáy lên

Khi đối số `charsize` là 1, ký tự có kích thước nhỏ nhất. Giá trị 2 sẽ tăng đôi kích thước các ký tự; giá trị 3 sẽ tăng kích thước lên 3 lần v.v .

8.2 Hiện văn bản trên màn hình đồ họa

Turbo C cung cấp hai hàm để viết ra màn hình đồ họa các xâu ký tự:

```
void far outtext(char far *textstring);

void far outtextxy(int x,int y,char far *textstring);
```

Cả hai hàm này hiện lên màn hình đồ họa các xâu ký tự theo các thiết lập hiện thời do hai hàm `settextstyle()` và `settextjustify()` quy định.

Hàm `outtext()` hiển thị một xâu ký tự trong khung nhìn tại vị trí hiện thời của con trỏ màn hình đồ họa. Trong khi `outtextxy()` hiển thị xâu ký tự tại điểm có tọa độ (x, y) trong khung nhìn.

Để đảm bảo tính tương thích khi sử dụng các fonts khác nhau ta sử dụng hai hàm `textwidth()` và `textheight()` để xác định kích thước của xâu ký tự:

```
int far textheight(char far *textstring);
int far textwidth(char far *textstring);
```

Hàm `textheight()` trả về số điểm theo chiều cao cần cho việc hiển thị xâu ký tự.

Hàm `textwidth()` trả về số điểm theo chiều dài dùng để vẽ xâu ký tự.

8.3 Điều chỉnh lề của văn bản được đưa ra

- Hàm

```
void far settextjustify(int horiz, int vert);
```
Với

Tham số	Giải thích
`int horize`	căn lề theo chiều ngang
`int vert`	căn lề theo chiều đứng

Xác lập các thông số để căn lề cho văn bản được đưa ra trên màn hình đồ họa.

Sau lời gọi tới `settextjustify()` Các văn bản viết ra màn hình đồ họa được căn lề xung quanh con trỏ màn hình đồ họa theo các chiều ngang và dọc theo các tham số đi cùng hàm `settextjustify()`. Bảng sau đây mô tả giá trị mà các tham số `horize` và `vert` có thể nhận được:

Tham số	Tên hằng	Giá trị	Giải thích
`horize`	LEFT_TEXT	0	Văn bản đưa ra được căn lề về bên trái vị trí lấy làm mốc (trong `outtext()` đó là CP, đối với `outtextxy()` đó là điểm có tọa độ được mô tả kèm theo lời gọi hàm). Đó là thiết lập ngầm định.

	CENTER_TEXT	1	Căn lề giữa. Điểm lấy làm mốc nằm giữa văn bản được đưa ra (theo chiều dọc).
	RIGHT_TEXT	2	Căn lề phải
vert	BOTTOM_TEXT	0	Điểm lấy làm mốc nằm dưới văn bản được đưa ra.
	CENTER_TEXT	1	Điểm lấy làm mốc nằm giữa văn bản được đưa ra (theo chiều đứng).
	TOP_TEXT	2	Điểm lấy làm mốc nằm trên văn bản được đưa ra.

8.4 Thay đổi kích thước và tỷ lệ của các ký tự

- Hàm

```
void far setusercharsize(multx,divx,multy, divy);
```

Với

Tham số	Giải thích
int multx	Giãn chiều rộng của ký tự
int divx	Co chiều cao của ký tự
int multy	Giãn chiều cao của ký tự
int divy	Co chiều cao của ký tự

Để thay đổi kích thước các ký tự. Thí dụ nếu multx=3 và divx=2 thì ký tự sẽ rộng bằng 3/2 lần ký tự thông thường.

Bài tập mẫu 7.20

```
/*stroke.c*/
/*kiểm tra các font*/
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#define FONT_SIZE 4
void main(){
        int driver, mode;
        driver = DETECT;
```

```
        initgraph(&driver, &mode, "c:\\tc\\bgi");
        settextstyle(GOTHIC_FONT,HORIZ_DIR,FONT_SIZE);
        outtext("Gothic");
        settextstyle(TRIPLEX_FONT,HORIZ_DIR,FONT_SIZE);
        outtext("Triplex");
        moveto(0,60);
        settextstyle(SMALL_FONT,HORIZ_DIR,FONT_SIZE);
        outtext("Small");
        settextstyle(SANS_SERIF_FONT,HORIZ_DIR,FONT_SIZE);
        outtext("Sansserif");
        moveto(0,120);
        settextstyle(DEFAULT_FONT,HORIZ_DIR,FONT_SIZE);
        outtext("Default");
        moveto(250,0);
        settextstyle(TRIPLEX_FONT,VERT_DIR,FONT_SIZE);
        outtext("Triplex");
        getche();
        closegraph();
        }
```

Bài tập mẫu 7.21

```
/*justify.c*/
/*kiểm tra cách căn lề các xâu ký tự*/
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#define CL  150
#define LEAD   40
#define FONT_SIZE 3
void main()    {
        int driver = DETECT, mode, j;
        initgraph(&driver,&mode, "c:\\tc\\bgi");
        settextstyle(TRIPLEX_FONT,HORIZ_DIR,FONT_SIZE);
```

```
        line(CL,0,CL,200);
        for(j =0; j < LEAD * 5; j+=LEAD)
                line(0,j,300,j);
        moveto(CL,0);
        outtext("Default");
        moveto(CL,LEAD);
        settextjustify(LEFT_TEXT,TOP_TEXT);
        outtext("Left-Top");
        moveto(CL,LEAD * 2);
        settextjustify(RIGHT_TEXT,TOP_TEXT);
        outtext("Right-Top");
        moveto(CL,LEAD * 3);
        settextjustify(CENTER_TEXT,TOP_TEXT);
        outtext("Center-Top");
        moveto(CL,LEAD * 4);
        settextjustify(CENTER_TEXT,BOTTOM_TEXT);
        outtext("Center-Bottom");
        getche();
        dosegraph();
        }
```

Bài tập mẫu 7.22

```
/*barega.c*/
/*tạo đồ thị khung trên màn hình EGA*/
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#define N                       12 /* số mức dữ kiện*/
#define BWIDTH                  20 /*bề rộng khối*/
#define SEP                     24 /*khoảng cách giữa các khối*/
#define SHFT            30 /*giữa viền và khối 1*/
#define LEFT            5
#define DEPTH           6
```

```
#define TOPFLAG              1
#define BOT          300
#define TOP          5 /*khối ba chiều có nắp*/
#define TICKS        10
#define TWIDTH               10
#define MAXDATA              100 /*vị trí tiêu đề*/
#define XTITLE               40
#define YTITLE               40
#define FONT                 SANS_SERIF_FONT
#define DI                   (BWIDTH + SEP)
#define WIDTH        ((N+1) *DI) /TICKS
#define PBT                  ((float)(BOT-TOP))
#define PPD                  ((float) (BOT-TOP)/MAXDATA)
void main() {
        int driver = EGA, mode = EGAHI, j;
        char string[40];
        int data[N] = {41,47,54,62,63,59,75,83,89,96,55,2};
        char months[12][4] = {"Jan","Feb","Mar","Apr","May",
                              "Jun","Jul","Sep","Oct","Nov","Dec"};
        initgraph(&driver,&mode,"c:\\tc\\bgi");
        /*vẽ viền*/
        rectangle(LEFT, TOP, LEFT + WIDTH, BOT);
        /*vẽ tiêu đề*/
        setusercharsize(4,3,4,3);
        settextstyle(FONT,HORIZ_DIR,0);
        moveto(XTITLE,YTITLE);
        outtext("1996 SALES");
        setusercharsize(2,3,2,2);
        settextstyle(FONT,HORIZ_DIR,0);
        for (j = 0; j < TICKS; j++) {
             /*vạch trái*/
                line(LEFT, BOT- j * PBT,
                        LEFT + WIDTH , BOT - j * PBT);
                /*vạch phải*/
```

```
            line(LEFT + WIDTH - TWIDTH, BOT- j * PBT,
                        LEFT + WIDTH , BOT - j * PBT);
            moveto(LEFT + WIDTH + SEP, BOT - j * PBT -
                    PBT /3);
            itoa(j * (MAXDATA/TICKS),string,10); /*so*/
            outtext(string);
            }
      settextstyle(FONT,VERT_DIR,0);
      for (j =0; j < N; j ++) {
            setfillstyle(SOLID_FILL,j);
            /*vẽ khối và khung*/
            bar3d(LEFT+SHFT+j*DI,BOT-data[j]*PPD,
                LEFT + SHFT + j * BWIDTH, BOT,
                DEPTH,
                    TOPFLAG);
            moveto(LEFT+SHP+j*DI+BWIDTH,BOT+5);
            outtext(months[j]);
            }
       getche();
       closegraph();
      }
```

9. CẮT DÁN HÌNH VÀ TẠO ẢNH CHUYỂN ĐỘNG

Để cất giữ hình ảnh vào bộ nhớ, sử dụng hàm:

```
void far getimage(left, top, right, bottom, addrBuff);
```

Trong đó

Tham số	Giải thích
`int left, top`	tọa độ góc trên của vùng
`int right, bottom`	tọa độ góc dưới
`void far *addrBuff`	địa chỉ bộ nhớ lưu ảnh

Về yêu cầu, hàm này cần biết tọa độ của vùng màn hình và địa chỉ của vùng nhớ chứa ảnh. Để biết kích thước của hình chúng ta sử dụng hàm `imagesize()` như sau:

```
unsigned far imagesize(left, top, right, bottom);
```

Với

Tham số	Giải thích
`int left, top`	góc trên bên trái
`int right, bottom`	góc dưới bên phải

Giá trị trả về của hàm này được truyền đến hàm `malloc()` để xin một vùng nhớ chứa ảnh.

Để khôi phục lại hình ảnh đã được lưu bằng `getimage()` trên màn hình (tại tọa độ bất kỳ), sử dụng hàm `putimage()` như sau:

```
void far putimage(left, top, addrBuff,putop);
```

Với

Tham số	Giải thích
`int left, top`	Vị trí bắt đầu của ảnh
`void far *addrBuff`	Vùng nhớ chứa ảnh
`int putop`	Cách đưa ảnh lên màn hình

Trong tệp `graphics.h` đã định nghĩa các phương thức đưa ảnh ra màn hình:

Giá trị	Tên hằng	Giải thích
0	COPY_PUT	Thay hình cũ bằng hình mới
1	XOR_PUT	XOR hình cũ và hình mới (xem lại phép xor bit để hiểu rõ điểm này)
2	OR_PUT	OR hình cũ và hình mới
3	AND_PUT	AND hình cũ và hình mới
4	NOT_PUT	Thay hình cũ bằng âm bản của hình mới.

Nói chung, hình cũ bị xóa bằng XOR_PUT, còn hình mới được vẽ bằng COPY_PUT. Sau đây là chương trình vẽ quả bóng dội trên màn hình.

Bài tập mẫu 7.23

```
/*image.c*/
/*vẽ quả bóng dội*/
#include <graphics.h>
#include <alloc.h>
```

```
#include <conio.h>
#include <stdlib.h>
#define LEFT   0
#define TOP    0
#define RIGHT  319
#define BOTTOM        199
#define RADIUS        8
#define CGAC0_LIGHTGREEN 1
void main() {
        int driver = CGA, mode = CGAC0;
        int x, y, dx, dy, oldx,oldy;
        void *ballbuff;
        unsigned size;
        initgraph(&driver, &mode, "c:\\tc\\bgi");
        /*vẽ khung*/
        rectangle(LEFT,TOP, RIGHT, BOTTOM);
        x = y = RADIUS + 10; /*qua bong*/
        setcolor(CGAC0_LIGHTGREEN);
        setfillstyle(SOLID_FILL, CGAC0_LIGHTGREEN);
        circle(x, y, RADIUS);
        floodfill(x, y, CGAC0_LIGHTGREEN);
        /*lấy kích thước ảnh*/
        size = imagesize(x - RADIUS, y - RADIUS,
                         x + RADIUS, y + RADIUS);
        ballbuff = malloc(size);/*cấp phát bộ nhớ*/
        getimage(x - RADIUS, y -RADIUS,
                         x + RADIUS, y + RADIUS, ballbuff);
        dx = 2; /*tốc độ bóng*/
        dy = 1;
        while (!kbhit()) /*chấm dứt khi gõ phím*/          {
                /*khôi phục hình*/
                putimage(x - RADIUS, y - RADIUS,
```

```
                        ballbuff, COPY_PUT);
        oldx = x; /*lưu vị trí cũ*/
        oldy = y;
        x +=dx;    /*di chuyển bóng*/
        y +=dy;
/*đổi hướng khi đụng mép*/
if (x <= LEFT + RADIUS + 2 || x >= RIGHT-RADIUS -2)
        dx = -dx;
if (y <= TOP + RADIUS + 2 || y >= BOTTOM-RADIUS -1)
                dy = -dy;
/*xóa ảnh cũ*/
putimage(oldx-RADIUS, oldy - RADIUS, ballbuff,XOR_PUT);
}
free(ballbuff);
getche();
closegraph();
}
```

Chương 8

BỘ TIỀN XỬ LÝ

Mục đích chương này

1) Cách dùng chỉ thị `#include` để phát triển các ứng dụng lớn.
2) Cách dùng chỉ thị `#define` để tạo các macro
3) Hiểu cách biên dịch có điều kiện
4) Hiện các thông báo lỗi trong quá trình biên dịch có điều kiện
5) Kiểm tra xem giá trị của một biểu thức có đúng như mong muốn không

1. GIỚI THIỆU

Chương này giới thiệu bộ tiền xử lý của C. Các chỉ thị tiền xử lý xuất hiện trước khi chương trình được biên dịch. Một vài thao tác có thể thực hiện trong giai đoạn này như : gộp thêm các tệp khác vào tệp đang biên dịch, định nghĩa các *hằng, macro, biên dịch* có điều kiện mã chương trình, *thực hiện có điều kiện các chỉ thị tiền dịch*. Tất cả các chỉ thị tiền dịch đều bắt đầu bằng kí tự # và chỉ có các kí tự dấu cách được phép đứng trước các chỉ thị tiền xử lý trong dòng đó.

2. CHỈ THỊ TIỀN DỊCH `#include`

Chỉ thị tiền dịch `#include` được xử dụng rất rộng rãi, chỉ thị này thực hiện chép tệp xác định vào vị trí đặt nó. Có hai dạng của chỉ thị `#include`

```
#include      <ten_tep>
#include      "ten_tep"
```

Hai dạng này chỉ khác nhau ở quá trình tìm vị trí của tệp được gộp vào. Trong dạng sau, bộ tiền xử lý sẽ tìm tệp trong thư mục cùng với thư mục tệp đang được dịch, phương pháp này thường áp dụng cho các tệp header do người dùng định nghĩa. Dạng đầu tiên được dùng khi gộp các tệp tiêu đề thư viện chuẩn, việc tìm kiếm được thực hiện trong các thư mục được đặt trước, thường là các thư mục chứa các tệp tiêu đề thư viện chuẩn.

Để mô tả đường dẫn thư mục các tệp tiêu đề thư viện chuẩn, Sử dụng lệnh Options->Directory trong môi trường Borland C hoặc Turbo C.

Thường thì `#include` được dùng với các tệp tiêu đề, nhưng đôi khi nó cũn hữu ích khi dùng để gộp các tệp chương trình nguồn.

3. CHỈ THỊ TIỀN XỬ LÝ #DEFINE

3.1 Định nghĩa các hằng

Chỉ thị `#define` được dùng để định nghĩa các hằng, biểu diễn các hằng bằng các xâu dễ đọc hơn. `#define` được dùng dưới dạng :

```
#define    danh_biểu xâu_kí_tự_thay
```

Khi đó, tất cả *danh-biểu* xuất hiện trong chương trình sẽ được thay tự động thành *xâu-kí-tự-thay* trước khi chương trình được dịch.

Ví dụ

```
#define    PI     3.14159
```

sẽ thay tất cả các hằng `PI` thành `3.14159`. Các hằng này khi cần thay đổi giá trị chỉ cần thay đổi một lần tại dòng định nghĩa một lần chứ không phải thay đổi tất cả các lần xuất hiện trong chương trình. Chú ý rằng tất cả các ký tự ở bên phải danh biểu tên hằng đều được thay cho tên hằng, ví dụ dòng

```
#define PI = 3.14159
```

sẽ thông báo cho bộ tiền xử lý thay tất cả các lần xuất hiện `PI` bằng xâu các kí tự "= 3.14159", khi đó sẽ gây ra lỗi cú pháp hay logic khó phát hiện. Việc định nghĩa lại hằng một lần nữa cũng gây ra lỗi cú pháp.

3.2 Định nghĩa các macro

Tương tự như định nghĩa hằng, tên một macro sẽ được thay trong chương trình bằng một đoạn văn bản trước khi chương trình được biên dịch. Macro có thể được định nghĩa có hoặc không có tham số. Một macro không có tham số được xử lý hoàn toàn tương tự như định nghĩa hằng. Trong các macro có tham số, các tham số sẽ được thay bằng một đoạn văn bản tương ứng, hãy thử xem ví dụ sau :

```
#define CIRCLE_AREA(r)  PI * (r) * (r)
```

Mỗi khi `CIRCLE_AREA(x)` xuất hiện trong chương trình, giá trị của `x` sẽ được thay bằng đoạn văn bản truyền vào cho `x`. Ví dụ như :

```
area = CIRCLE_AREA(4)
```

sẽ được chuyển thành:

```
area = 3.14159 * (4) * (4)
```

trước khi chương trình được biên dịch.

Chú ý rằng trong định nghĩa macro, các dấu đóng mở ngoặc tham số là cần thiết, ví dụ

```
area = CIRCLE_AREA(c + 2)
```

sẽ được chuyển thành:

```
area = 3.14159 * (c + 2) * (c + 2)
```

Khi không sử dụng dấu ngoặc, tức là dòng định nghĩa macro như sau:

```
#define CIRCLE_AREA(r)  PI * r * r
```

gặp trường hợp

```
area = CIRCLE_AREA(c + 2)
```

sẽ được chuyển thành:

```
area = 3.14159 * c + 2 * c + 2
```

Kết quả này sai hoàn toàn.

Macro CIRCLE_AREA trên cũng có thể định nghĩa như một hàm :

```
double CircleArea( double r){
return 3.14159 * r * r;
        }
```

Khi đó, công việc được thực hiện hoàn toàn tương tự. Tuy nhiên, chi phí cho lời gọi hàm sẽ nhiều hơn. Ưu điểm của macro so với hàm là nó chèn trực tiếp mã chương trình vào, tránh các chi phí khi gọi hàm mà chương trình vẫn giữ được tính dễ đọc, bởi vì việc tính toán cuả CIRCLE_AREA cũng được thực hiện riêng rẽ. Tuy nhiên điều bất lợi là các tham số được tính đi tính lại nhiều lần.

Phần văn bản thay thế của cả macro và hằng là tất cả ký tự bên phải tên danh biểu cho đến hết dòng. Nếu đoạn văn bản thay thế đó quá dài cần xuống dòng khác, hãy thêm kí tự "\" vào cuối dòng. Kí tự này thông báo rằng đoạn văn thay thế còn tiếp tục ở dòng tiếp theo.

3.3 Loại bỏ macro, hằng ký hiệu

Khi không muốn dùng một hằng hay một macro, ta dùng chỉ thị

```
#undefine danh_biểu
```

Phạm vi của hằng hay macro sẽ bắt đầu từ vị trí định nghĩa nó và kết thúc ở chỉ thị #undefine nó hoặc ở cuối tệp.

4. BIÊN DỊCH CÓ ĐIỀU KIỆN

Biên dịch có điều kiện cho phép lập trình viên điều khiển quá trình thực hiện của các chỉ thị tiền xử lý và quá trình biên dịch chương trình.

Cấu trúc biên dịch có điều kiện rất giống với cấu trúc rẽ nhánh **if**. Xem xét ví dụ sau :

```
#if  !define (NULL)
        #define  NULL  0
#endif
```

Các chỉ thị trên thực hiện kiểm tra xem NULL đã được định nghĩa chưa. Biểu thức `define(NULL)` trả lại giá trị 1 nếu NULL đã được định nghĩa, 0 trong trường hợp ngược lại. Nghĩa là trong trường hợp cụ thể trên, nếu `NULL` chưa được định nghĩa thì sẽ thực hiện định nghĩa lại giá trị `NULL` bằng 0. Tất cả các chỉ thị `#if` đều kết thúc bằng `#endif`. Chỉ thị `#elif` tương tự như `else` trong cấu trúc `if`. Các chỉ thị `#ifdef` và `#ifndef` là dạng viết tắt của `#if  define( name)` và `#if  !define( name)`

Trong quá trình xây dựng chương trình, nếu muốn không biên dịch một phần chương trình mà không sử dụng được `/*` và `*/` thì có thể sử dụng cấu trúc các chỉ thị tiền xử lý sau:

```
#if  0
        Phần chương trình không muốn biên dịch
#endif
```

Để cho phần chương trình đó lại tiếp tục được biên dịch chỉ cần đơn giản thay 0 bằng 1.

Biên dịch có điều kiện thường dùng để trợ giúp quá trình gỡ rối. Hầu hết các trình biên dịch C đều cung cấp khả năng gỡ rối. Tuy nhiên, gỡ rối thường khó hiểu và khó sử dụng, nên chúng ít được dùng đối với những người mới lập trình.

Ví dụ

```
#ifdef DEBUG
printf("Bien x = %d\n", x);
#endif
```

Trong đoạn chương trình trên, thường khi đang phát triển ứng dụng chương trình được dịch ở chế độ gỡ rối, khi đó trình biên dịch tự động định nghĩa DEBUG. Tức là lúc đó câu lệnh printf bên trong cũng được biên dịch, chương trình chạy đến vị trí đó sẽ in ra giá trị x cho lập trình viên. Tuy nhiên, khi dịch chương trình thành sản phẩm cuối cùng, không ở trong chế độ dịch gỡ rối, các dòng mã lệnh giúp cho

quá trình gỡ rối như dòng lệnh printf("Biến x = %đ\n", x) ở trên sẽ không được biên dịch nữa.

Một số hằng được trình biên dịch định nghĩa trước giúp cho quá trình gỡ rối được liệt kê trong bảng 8.1

Tên hằng	Chú thích
__LINE__	Số nguyên biểu diễn số dòng trong tệp chương trình nguồn hiện thời
__FILE__	Xâu kí tự chỉ tên của chương trình nguồn hiện thời
__DATE__	Ngày tháng tệp chương trình nguồn được biên dịch. Xâu kí tự theo khuôn dạng "tháng ngày năm" theo tiếng Anh, ví dụ "Jan 20 1996"
__TIME__	Giờ tệp chương trình nguồn được biên dịch. Xâu kí tự theo khuôn dạng "giờ : phút : giây"

Bảng 8.1 Các hằng được định nghĩa trước

Chẳng hạn, một chương trình đang chạy gặp một lỗi nào đó sẽ hiện đúng vị trí xẩy ra lỗi đó : ở tệp nào, ở dòng nào.

Ví dụ

```
#ifdef DEBUG
    if (có_lỗi_xẩy_ra){
        printf("Có lỗi xẩy ra tại tệp %s ở dòng %d",
            __FILE__, __LINE__);
    }
#endif
```

Khi lập trình nên chịu khó viết thêm các chỉ thị tiền dịch và các câu lệnh trợ giúp quá trình gỡ rối như trên, đặc biệt trong quá trình viết các thư viện dùng chung, các ứng dụng lớn có nhiều người tham gia rất hữu ích.

Có thể nhận thấy, chương trình nguồn của các thư viện phần mềm được phát triển bởi các hãng lớn (như MFC của Microsoft hay OWL của Borland) đều sử dụng rất nhiều các chỉ thị tiền dịch giúp cho quá trình gỡ rối ứng dụng sau này.

5. MỘT SỐ CHỈ THỊ TIỀN DỊCH KHÁC

5.1 Chỉ thị #error

Cú pháp

```
#error <message>
```

gây ra một lỗi khi biên dịch, dừng ngay quá trình biên dịch và in kèm theo thông báo `<message>`.

Ví dụ

```
#ifndef (MODEL)
        #error Bulding model not defined
#endif
```

5.2 Chỉ thị `#line`

Cú pháp

```
#line <hằng> [<Danh biểu thay cho tên tệp>]
```

thông báo cho trình biên dịch biết dòng tiếp nay sau sẽ được đánh số lại bằng giá trị của *hằng*, và *danh biểu* sẽ được dùng thay cho tên tệp hiện thời.

6. KHUÔN DẠNG TỆP TIÊU ĐỀ

6.1 Đặt vấn đề

Các chỉ thị biên dịch có điều kiện sử dụng để tạo các tệp tiêu đề. Như đã biết, trong một tệp tiêu đề người ta khai báo các hàm, biến, hằng được sử dụng trong nhiều tệp chương trình nguồn khác nhau. Các khai báo trong tệp tiêu đề sẽ được gộp vào trong các tệp chương trình nguồn sử dụng chúng khi biên dịch. Trong một chương trình lớn ta thường chia các chương trình nguồn thành nhiều tệp nhỏ khác nhau. Khi đó thường có các tệp tiêu đề chứa khai báo các biến, hằng, hàm được nhiều tệp sử dụng. Để tránh sự lặp lại của các khai báo trong tệp tiêu đề ta thường đặt chỉ thị tiền dịch có điều kiện #ifndef tại đầu mỗi tệp.

6.2 Ví dụ minh hoạ

Chương trình ví dụ sau đây mô tả cách viết một chương trình có sử dụng tệp tiêu đề tự định nghĩa. Chương trình bao gồm một tệp tiêu đề tg.h chứa khai báo cấu trúc cùng với các hàm liên quan. Tệp tiêu đề này được sử dụng trong hai tệp chương trình nguồn:

(i) `tgmain.c` chứa hàm `main()` và các lời gọi đến các hàm thư viện trong `tg.h`.

(ii) `tglib.c` chứa định nghĩa của các hàm thư viện trên cấu trúc khai báo trong tệp tiêu đề.

Đây đồng thời cũng là một ví dụ tốt cho việc xây dựng một tệp project.

Tệp tg.h

```
/*tg.h*/
#ifndef tamgiac_h
#define tamgiac_h
struct diem {
  double x,y;
  };
struct tamgiac {
  struct diem A,B,C;
  };
void nhapDiem(struct diem *p);
void nhapTamgiac(struct tamgiac *tg);
int loaiTamgiac(struct tamgiac tg);
double canhbp(struct diem A, struct diem B);
double canh(struct diem A, struct diem B);
#endif
```

Tệp tgmain.c

```
/*tgmain.c*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "tg.h"
void main() {
  struct tamgiac tg;
  clrscr();
  nhapTamgiac(&tg);
  switch(loaiTamgiac(tg)) {
    case 1: printf("Tam giac deu\n");break;
    case 2: printf("Tam giac vuong can\n");break;
    case 3: printf("Tam giac can\n");break;
    case 4: printf("Tam giac vuong\n");break;
    case 5: printf("Tam giac thuong\n");break;
```

```
    }
  getch();
}
```

Tệp tglib.c

```
/*tglib.c*/
#include <stdio.h>
#include <math.h>
#include "tg.h"
/*định nghĩa macro*/
#define sqr(x) (pow(x,2.0))
double canhbp(struct diem A, struct diem B) {
  return sqr(A.x-B.x)+sqr(A.y-B.y);
  }
int loaiTamgiac(struct tamgiac tg) {
  double a=canhbp(tg.B,tg.C);
  double b=canhbp(tg.C,tg.A);
  double c=canhbp(tg.A,tg.B);
  if (a==b||b==c||c==a)
    if (a==b&&b==c)
      return 1;
    else if (a==b+c||b==c+a||c==a+b)
            return 2;
        else return 3;
  else if (a==b+c||b==c+a||c==a+b)
          return 4;
       else return 5;
  }
void nhapDiem(struct diem *d) {
  double x;
  printf("Toa do x = ");
  scanf("%lf",&x);
  d->x=x;
```

```
  printf("Toa do y = ");
  scanf("%lf",&x);
  d->y=x;
  }
void nhapTamgiac(struct tamgiac *p) {
  struct diem d;
  printf("Nhap dinh A\n");
  nhapDiem(&d);
  p->A=d;
  printf("Nhap dinh B\n");
  nhapDiem(&d);
  p->B=d;
  printf("Nhap dinh C\n");
  nhapDiem(&d);
  p->C=d;
  }
```

7. TÓM TẮT

Những lỗi thường gặp khi lập trình

Quên không đóng ngoặc các tham số trong macro

Chèn các câu lệnh biên dịch có điều kiện nhằm mục đích gỡ rối như các câu lệnh `printf()` vào những nơi khi thêm 1 câu lệnh đơn có thể gây ra lỗi chẳng hạn như ví dụ sau :

```
if (x & 4)
    #ifdef DEBUG
            printf("Giá trị x = %d", x);
#endif
    x = 10;
```

Phụ lục

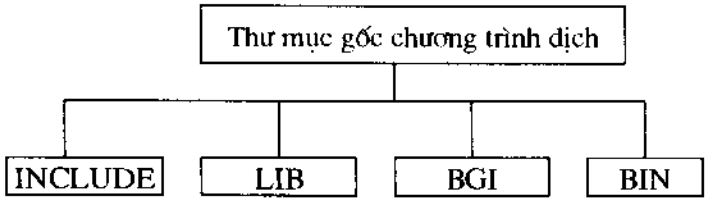
LÀM VIỆC VỚI TURBO C, TURBO C++, BORLAND C (FOR DOS)

Phụ lục này giới thiệu với bạn mới học ngôn ngữ C một số vấn đề liên quan đến kỹ năng thực hành viết các chương trình ngôn ngữ C trên các môi trường Turbo C, Turbo C++ và Borland C. Chúng tôi giới thiệu các môi trường này vì hai lý do:

Thứ nhất, phần lớn bạn đọc đều đã từng làm việc với môi trường Turbo PASCAL nên dễ dàng tiếp cận được với một môi trường tương tự như thế.

Thứ hai, cách tổ chức giao diện các sản phẩm của chương trình của Borland vốn nổi tiếng về tính sư phạm và dễ hiểu.

1. CÁC THƯ MỤC QUAN TRỌNG



Trên đĩa cứng (chẳng hạn ổ đĩa C:), chương trình dịch Turbo C, Turbo C++ đặt trong thư mục `\TC`, chương trình dịch Borland C đặt trong thư mục `\BORLANDC`. Bên trong thư mục gốc của chương trình biên dịch có hai thư mục con hết sức quan trọng là `INCLUDE` và `LIB`. Thư mục `INCLUDE` chứa tất cả các tệp tiêu đề thư viện chuẩn do chương trình dịch cung cấp như là `stdio.h`, `conio.h`, `math.h`, `string.h` v.v. Phần cài đặt của các hàm, biến, hằng tương ứng đặt trong các tập tin nằm trong thư mục `LIB`. Đối với Turbo C++ và Borland C, bên trong thư mục gốc chương trình dịch còn có thư mục con `BIN` chứa các chương trình thực hiện ví dụ `TC.EXE` hoặc `BC.EXE`(với Turbo C, các tệp thực hiện nằm ngay trên thư mục gốc chương trình dịch). Các tệp liên quan đến đồ hoạ nằm trong thư mục con `BGI` của thư mục gốc chương trình dịch.

Khi biên dịch và thực hiện các chương trình nguồn trên ngôn ngữ C, chương trình dịch sinh ra hai loại tệp là *.OBJ và *.EXE. Do vậy người lập trình cần chuẩn

bị một thư mục nào đó để chứa các tệp đó nhằm thuận tiện cho các thao tác xoá các tệp đó đi khi không thấy cần thiết. Theo mặc định ta thường sử dụng thư mục C:\TEMP để lưu các tệp kể trên.

2. KHỞI ĐỘNG MÔI TRƯỜNG IDE CỦA CHƯƠNG TRÌNH BIÊN DỊCH

Thực hiện chương trình TC.EXE đối với Turbo C, Turbo C++ và thực hiện BC.EXE đối với Borland C.

Các thao tác soạn thảo chương trình nguồn, cách sử dụng các lệnh trên menu giống hệt với Turbo PASCAL. Ở đây xin nhắc một số thao tác quan trọng.

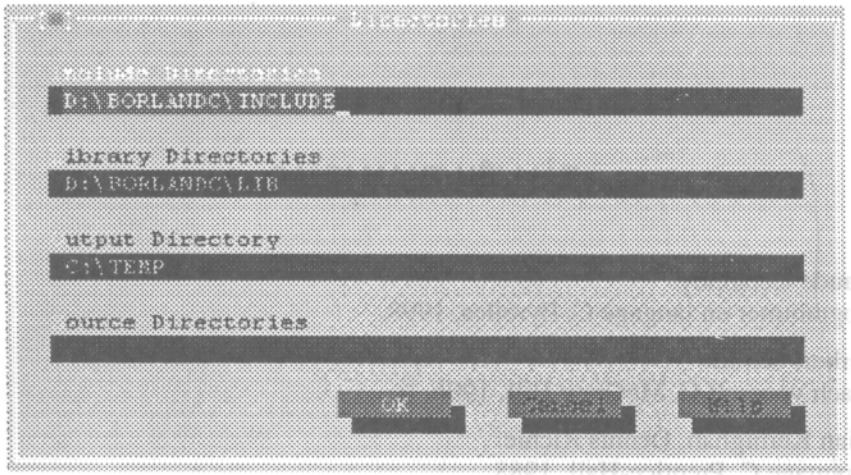
Thao tác	Phím nóng hoặc lệnh menu
Mở một tệp chương trình nguồn	F3 hoặc File->Open
Ghi chương trình nguồn lên đĩa	F2 hoặc File->Save
Ghi với tên mới	File->Save as
Đóng một cửa sổ soạn thảo	Alt+F3
Biên dịch	Alt+F9
Tạo tệp tin EXE	F9
Biên dịch và thực hiện	Ctrl+F9
Chạy lần vết	F7, F8, F4, Ctrl+F2, Ctrl+F8
Thoát khỏi IDE	Alt+X, File->Quit
Xem trợ giúp	F1, Ctrl+F1

3. CÀI ĐẶT CẤU HÌNH

3.1 Đường dẫn thư mục

Sử dụng lệnh

Options->Directories...



Trong hộp **Include directories**, ta nhập vào đường dẫn thư mục INCLUDE. Nếu có thêm các thư mục khác chứa các tệp tiêu đề tự định nghĩa thì nhập đường dẫn tương ứng và sử dụng dấu chấm phẩy để ngăn cách các đường dẫn. Tương tự cho hộp **Library directories**. Hộp **Output Directory** chứa đường dẫn mô tả thư mục chứa các tệp *.OBJ và *.EXE sinh ra khi biên dịch và thực hiện chương trình nguồn. Không cần thiết phải quan tâm đến **Source Directories**.

3.2 Mô hình bộ nhớ

Mô hình bộ nhớ mô tả cấu trúc của tệp chương trình thực hiện được sinh ra. Có 6 mô hình bộ nhớ khác nhau là: **tiny, small, medium, compact, large, huge**. Chi tiết về từng mô hình có thể xem trong phần trợ giúp đi kèm.

Để xác lập mô hình bộ nhớ, sử dụng lệnh :

Options->Compiler->Code generation ...

3.3 Ghi lại các thiết lập cấu hình

Lệnh

Options->Save

TÀI LIỆU THAM KHẢO

1. Claude Delannoy
   Programmer en langage C, Eyrolles, 1998.
2. Herbert Schildt
   Teach yourself C, McGraw-Hill, 1996.
3. Brian Kernighan, Dennis Richie
   Language C, Prentice-Hall, 1988.
4. Bryon Gottfried
   Programming with C, McGraw-Hill, 1996.
5. Al Stevens
   Extending Turbo C Professional, 1989.
6. Allen Holub
   Compiler design in C, Prentice-Hall, 1990.
7. Nguyễn Thanh Thủy, Nguyễn Quang Huy
   Bài tập lập trình ngôn ngữ C, Nhà xuất bản Khoa học và Kỹ thuật, 1999.
8. Lê Đăng Hưng, Tạ Tuấn Anh, Nguyễn Hữu Đức, Nguyễn Thanh Thủy (Chủ biên)
   với C++, Nhà xuất bản Khoa học và Kỹ thuật, 1999.
9. Ngô Trung Việt et al.
   Ngôn ngữ lập trình C, Viện Tin học, Dự án VIE/88/035, 1990.
10. Phạm Văn Ất
    Kỹ thuật lập trình C, Nhà xuất bản Khoa học và Kỹ thuật, 1999.
11. Phạm Văn Ất
    Nhập môn C++ và lập trình hướng đối tượng, Nhà xuất bản Khoa học và Kỹ thuật, 1999.

MỤC LỤC

Chương 3. Mảng và Con trỏ

Chương 4. Hàm

Chương 5. Cấu trúc

Chương 6. Tệp và thao tác vào ra

Chương 7. Đồ hoạ

Chương 8. Bộ tiền xử lý

Phụ lục

203056

Giá: 30.000đ