W. BUCHANAN

LẬP TRÌNH C TRONG KỸ THUẬT ĐIỆN TỬ

NGƯỜI DỊCH: NGÔ DIÊN TẬP
PHẠM HUY QUỲNH

NHÀ XUẤT BẢN
KHOA HỌC VÀ KỸ THUẬT

W. Buchanan

LẬP TRÌNH C
TRONG
KỸ THUẬT ĐIỆN TỬ

Người dịch: **Ngô Diên Tập**
Phạm Huy Quỳnh

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 1999

Nguyên bản tiếng Anh:

W. Buchanan

C for Electronic Engineering

60 - 6T7 - 3
---------- 41 - 72 - 99
KHKT

Lời nói đầu

Sách viết về lập trình bằng ngôn ngữ C đã có nhiều nhưng lập trình C cho các chuyên ngành thì rất hiếm. Khi biết cách lập trình bằng ngôn ngữ C, ta lại cần hiểu sâu về các lĩnh vực ứng dụng mới có thể viết ra những chương trình ứng dụng có giá trị. Khi vượt qua được hai điều kiện này thì hoặc không có nhiều thời gian, hoặc không muốn ngồi viết lại những kinh nghiệm nên ít ai đã dành thời gian viết về các sách lập trình cho các chuyên ngành. Cuốn sách của W. Buchanan mà chúng tôi giới thiệu ở đây là một trường hợp đặc biệt.

Cuốn sách này được viết nhằm giúp cho sinh viên, kỹ sư các ngành kỹ thuật điện tử, kỹ thuật điện, kỹ thuật máy tính nhanh chóng làm quen với kỹ thuật lập trình bằng ngôn ngữ C theo cách trình bày ngắn gọn về ngôn ngữ rồi dẫn dắt người đọc đến với những bài toán cụ thể về kỹ thuật điện, điện tử.

Trong sách có rất nhiều chương trình dùng làm thí dụ minh hoạ, từ đơn giản đến phức tạp, đôi khi lặp lại ở một vài chủ đề nhưng khó dần để người đọc dần dần thành thạo. Điểm cần lưu ý là các chương trình minh hoạ này được giới thiệu trước hết nhằm mục đích đọc hiểu, nên để tránh sự hiểu lầm, đặc biệt với một số bạn đọc còn khó khăn với tiếng Anh, những lời chú thích và một vài thông báo trong chương trình cũng như ở phần kết quả chạy thử ở ***một số chương đầu*** *chúng tôi đã dịch* ***sang tiếng Việt có dấu;*** *và ta đều biết trình dịch không thể hiểu được các chữ có dấu này nên khi gõ chương trình vào máy tính để chạy thử ta phải bỏ đi các dấu hoặc chuyển sang tiếng Anh.*

Hy vọng nội dung cuốn sách sẽ giúp ích rất nhiều cho các sinh viên, kỹ sư chuyên cũng như không chuyên về kỹ thuật lập trình.

Về phần công việc:

- *Phạm Huy Quỳnh: dịch chương 2; 3 và soát lỗi các chương trình.*
- *Ngô Diên Tập: dịch các chương 1; 4 đến 12 và hoàn thiện toàn bộ bản thảo.*

Mặc dù đã dành thời gian thích đáng nhưng không tránh khỏi một số lỗi, đặc biệt là các lỗi về dấu ngắt câu trong phần chương trình vẫn có thể còn sót lại trong cuốn sách. Rất mong bạn đọc gần xa chỉ dẫn cho. Thư từ góp ý xin gửi về Nhà xuất bản Khoa học và Kỹ thuật, 70 Phố Trần Hưng Đạo, Hà nội.

Những người dịch

MỤC LỤC

Chương 1
MỞ ĐẦU

Các kỹ sư trong lĩnh vực điện, điện tử và phần mềm cần có tính linh hoạt rất cao khi tiếp cận với kỹ thuật phát triển hệ thống. Họ phải có hiểu biết về tất cả các mức trừu tượng hóa của hệ thống, dù đó là phần cứng, phần mềm hay phần sụn (firmware). Hệ thống có thể bao hàm một phạm vi rộng, từ bộ điều khiển hệ thống sưởi tập trung 4 bit tới hệ thống điều khiển lớn dùng trong công nghiệp. Trong quá trình nghiên cứu phát triển một hệ thống bất kỳ, người kỹ sư phải hiểu các đặc tính của hệ thống từ các yêu cầu giao diện, các yêu cầu phân chia thời gian (timing), các đặc trưng điện của hệ thống v. v... Các phần mềm chạy trên hệ thống phải có tính linh hoạt trong cấu trúc bởi vì người viết chương trình có thể yêu cầu kiểm tra, sửa đổi các địa chỉ bộ nhớ cũng như nội dung trong ô nhớ hoặc có thể thay đổi một phần của hệ thống để thực hiện một giải thuật nào đấy. Để đạt được mục đích này ngôn ngữ lập trình C tỏ ra là xuất sắc bởi vì nó cho phép đạt đến một mức cao của sự trừu tượng hóa (cụ thể là các đặc tính của giải thuật) và cho phép thực hiện các phép toán bậc thấp (như các phép toán trên các digit nhị phân). Nó có một phạm vi ứng dụng rộng rãi, từ giao dịch và kinh doanh tới công nghiệp và nghiên cứu. Đây là một thuận lợi nổi bật bởi vì nhiều ngôn ngữ lập trình có các phương tiện vốn làm cho các ngôn ngữ này trở thành có ích chỉ trong một môi trường đặc biệt. Chẳng hạn, ngôn ngữ COBOL được ứng dụng rộng rãi trong kinh doanh và thương

mại, trong khi ngôn ngữ FORTRAN lại được sử dụng nhiều trong công nghệ và nghiên cứu khoa học.

1.1 PHẦN CỨNG, PHẦN MỀM VÀ PHẦN SỤN

Một hệ thống bao gồm phần cứng, phần mềm và phần sụn, tất cả được ghép nối với nhau. Phần cứng là những phần của hệ thống mà ta có thể tiếp xúc đến, thí dụ các linh kiện, ốc vít và đai ốc, vỏ máy, các dây dẫn điện v. v... Phần mềm là các chương trình chạy trên phần cứng lập trình được và sự thay đổi hoạt động của chúng phụ thuộc vào các đầu vào của hệ thống. Các đầu vào này có thể được lựa chọn từ một bàn phím phần cứng ghép nối hoặc từ một thiết bị ngoại vi. Chương trình không thể tồn tại mà không có một vài dạng của phần cứng lập trình được như một bộ vi xử lý hoặc bộ điều khiển. Phần sụn là một thiết bị phần cứng được lập trình bằng cách sử dụng phần mềm. Thiết bị có thể xem như phần sụn tiêu biểu là các bộ EEPROM (bộ nhớ chỉ để đọc nhưng có thể lập trình hoặc xóa được bằng điện), và các thiết bị ghép nối, được lập trình bằng cách sử dụng các thanh ghi. Trong hầu hết các ứng dụng. phần cứng chuyên dụng hoạt động nhanh hơn phần cứng hoạt động kết hợp với phần mềm, mặc dù các hệ thống chạy phần mềm chương trình có khả năng dễ dàng sửa đổi và đòi hỏi thời gian cho nghiên cứu phát triển ít hơn.

1.2 LỊCH SỬ NGÔN NGỮ C

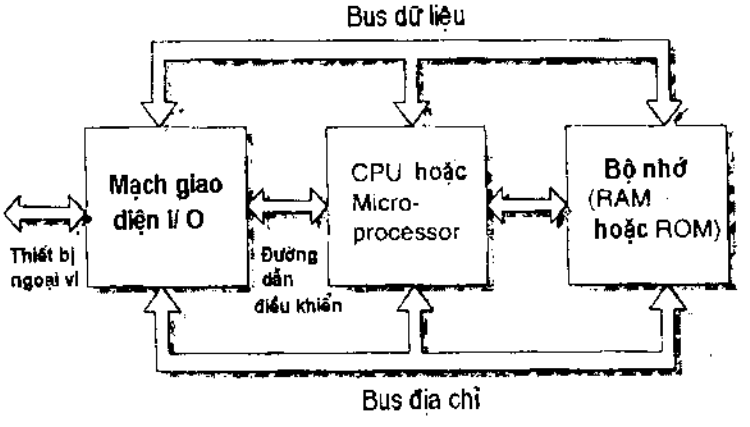
Dennis Ritchie phát triển ngôn ngữ lần đầu tiên tại phòng thí nghiệm Bell của công ty AT & T (Hoa Kỳ). Mục tiêu ông ta đặt ra khi đó là sử dụng một ngôn ngữ bậc cao để xây dựng hệ điều hành UNIX theo phương châm dễ dàng trong việc bảo trì, hiệu quả và dễ di chuyển (portable). Để đạt được mục đích này ông ta đã thiết kế một ngôn ngữ có nhiều ảnh hưởng, quan trọng nhất là ngôn ngữ BCPL. Ảnh hưởng của ngôn ngữ này lên C được tiếp tục một cách gián tiếp qua ngôn ngữ B.

Thoạt đầu, ngôn ngữ C được ứng dụng chủ yếu là trong lập trình hệ thống. Các chương trình dịch, các hệ điều hành và các tiện ích đều được viết bằng C. Ngôn ngữ C tỏ ra là ưu việt hơn trong các lĩnh vực này và bây giờ đã được ứng dụng trong nhiều lĩnh vực khác, trên nhiều loại hệ

thống máy tính khác nhau. Sau khi được chấp nhận, không giống như nhiều ngôn ngữ khác, ngôn ngữ C đã có xu hướng hình thành những nhóm tách biệt. Tới năm 1983, ngôn ngữ này đã phát triển đến giai đoạn cần có biện pháp để tiêu chuẩn hóa. Để đạt được mục đích này Ủy ban X3J11 của ANSI (Viện Tiêu chuẩn Quốc gia của Hoa Kỳ) được thành lập để chuẩn hóa ngôn ngữ này và đã xây dựng lên tiêu chuẩn ANSI-C. Hầu hết các sản phẩm C hiện nay đều tuân theo tiêu chuẩn này mà không theo những quy định lúc ban đầu.

1.3 SƠ LƯỢC VỀ CẤU TRÚC MÁY TÍNH

Các phần chính của một hệ thống máy tính cơ bản là một đơn vị xử lý trung tâm (hoặc thường gọi là bộ vi xử lý), bộ nhớ, và các mạch vào/ ra (I/ 0). Các phần tử đó được nối với nhau qua ba đường bus chính: bus địa chỉ, bus điều khiển và bus dữ liệu. Hình 1.1 mô tả một hệ thống cơ bản. Các thiết bị ngoại vi như một bàn phím, màn hình, các ổ đĩa, v. v..., có thể nối trực tiếp với các bus: dữ liệu, địa chỉ và điều khiển, hoặc ghép nối thông qua các mạch vào/ ra.



Hình 1.1: **Sơ đồ khối của một hệ thống máy tính đơn giản.**

Thông thường bộ nhớ bao gồm: bộ nhớ truy nhập ngẫu nhiên (RAM: Random Acesses Memory) và bộ nhớ chỉ để đọc (ROM: Read Only Memory). Bộ nhớ ROM cất giữ thông tin nhị phân để sử dụng lâu dài, trong khi bộ nhớ truy nhập ngẫu nhiên không phải là bộ nhớ các dữ liệu để sử dụng lâu dài và dữ liệu sẽ mất đi khi ngắt nguồn nuôi. Bộ nhớ

RAM được sử dụng để chạy các chương trình ứng dụng và để cất giữ thông tin một cách tạm thời.

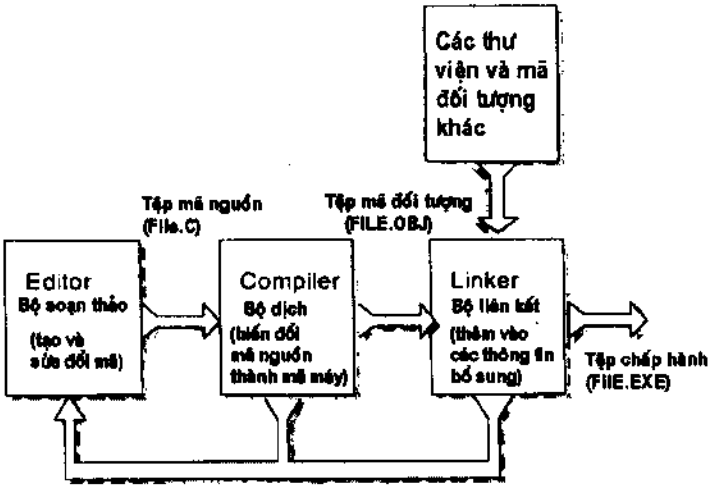
Bộ vi xử lý là bộ điều khiển chính của máy tính. Nó đem về từ bộ nhớ các lệnh (hoặc chỉ thị) nhị phân (hay thường gọi là mã máy), rồi giải mã đó thành một dãy các tác động đơn giản và thực hiện các tác động theo các bước kế tiếp nhau. Các bước này được đồng bộ hóa nhờ một đồng hồ hệ thống. Để truy nhập lên một vị trí trong bộ nhớ, hay thường gọi là ô nhớ, bộ vi xử lý đặt địa chỉ ô nhớ lên bus địa chỉ. Nội dung của địa chỉ này được đặt lên bus dữ liệu và bộ vi xử lý đọc dữ liệu từ bus dữ liệu. Để cất giữ dữ liệu vào trong bộ nhớ, bộ vi xử lý đặt dữ liệu lên bus dữ liệu. Sau đó, địa chỉ của ô nhớ trong bộ nhớ được đặt lên bus địa chỉ và dữ liệu lại được đọc từ bus dữ liệu vào địa chỉ bộ nhớ (ô nhớ) cần có.

1.4 BIÊN DỊCH, LIÊN KẾT VÀ VIẾT MỘT CHƯƠNG TRÌNH CHẤP HÀNH

Bộ vi xử lý chỉ hiểu được thông tin dưới dạng nhị phân và thao tác trên một dãy các lệnh nhị phân hay thường gọi là mã máy. Khi viết các chương trình lớn trực tiếp dưới dạng mã máy sẽ gặp rất nhiều khó khăn, nên các *ngôn ngữ bậc cao* đã được phát triển để sử dụng thay cho mã máy. Một *ngôn ngữ bậc thấp* rất giống với mã máy và thường kéo theo việc sử dụng các macro từ khóa để thay thế các chỉ thị mã máy. Ngôn ngữ bậc cao có cú pháp gần như viết tiếng Anh và như vậy làm cho chương trình trở nên dễ dàng đọc và dễ sửa đổi. Trong đa số các chương trình, hoạt động thực tại của phần cứng như thế nào, người viết chương trình không thể nhìn thấy được. Khi đó, một *chương trình dịch* sẽ chuyển đổi ngôn ngữ bậc cao thành ra mã máy. Các ngôn ngữ bậc cao có thể kể ra là: C, BASIC, COBOL, FORTRAN và PASCAL. Một thí dụ về ngôn ngữ bậc thấp là hợp ngữ (ngôn ngữ Assembly) dùng cho 80386.

Hình 1.2 giới thiệu một dãy các công đoạn cần thực hiện để tạo ra một chương trình mã máy từ một chương trình mã nguồn trong ngôn ngữ C (các tên tệp tin sử dụng trong thí dụ này liên quan đến một hệ

thống dựa trên máy tính cá nhân PC). *Bộ soạn thảo* được dùng để viết và sửa đổi một tệp mã nguồn C; sau đó một chương trình dịch sẽ chuyển đổi mã nguồn này sang một dạng mà bộ vi xử lý có thể hiểu được, cụ thể là mã máy. Tệp tin do chương trình dịch tạo ra được đặt tên là tệp mã đối tượng. Tệp tin này không thể thực hiện được bởi vì nó không có đầy đủ các thông tin cần có để chạy chương trình. Giai đoạn cuối cùng của quá trình là liên kết; quá trình này đòi hỏi phải đưa thêm mã máy vào chương trình để nó có thể sử dụng các thiết bị như bàn phím, màn hình, v. v... Một *bộ liên kết* sẽ kết nối tệp mã đối tượng với các tệp mã đối tượng khác và với các thư viện để tạo ra một chương trình chấp hành được (executable). Các thư viện này chứa các môđun mã đối tượng khác đã được biên dịch sang mã nguồn.

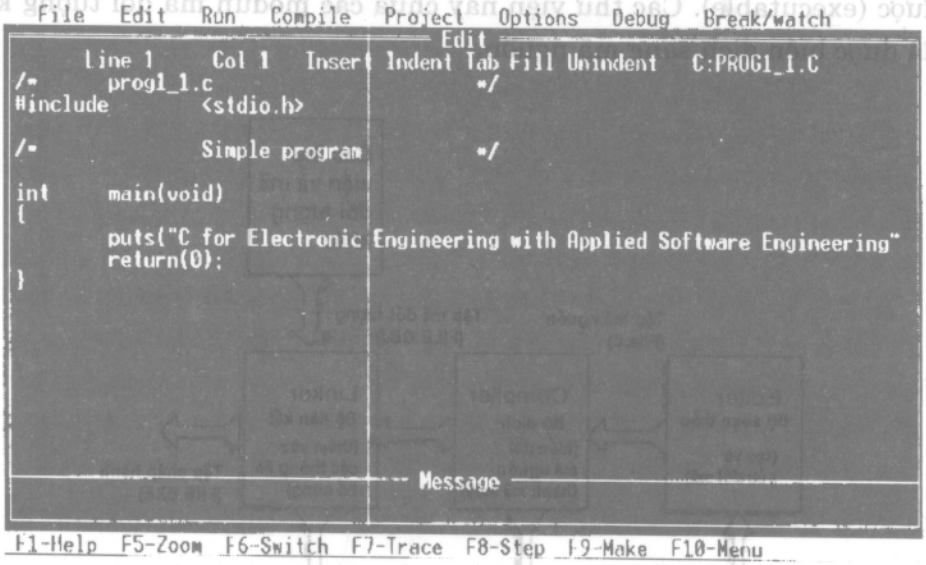


Hình 1.2: **Các quá trình soạn thảo, biên dịch và liên kết.**

Nếu quá trình dịch hoặc liên kết phát sinh các lỗi hoặc những lời cảnh báo thì mã nguồn phải được sửa đổi để loại trừ các lỗi. Quá trình biên dịch/ liên kết phải được tiến hành một lần nữa. Các thông báo trong quá trình biên dịch/ liên kết không làm dừng chương trình dịch hoặc bộ kết nối để tránh tạo ra một đầu ra, nhưng các lỗi thì sẽ làm ngừng. Vì vậy tất cả các lỗi trong giai đoạn biên dịch hoặc liên kết đều

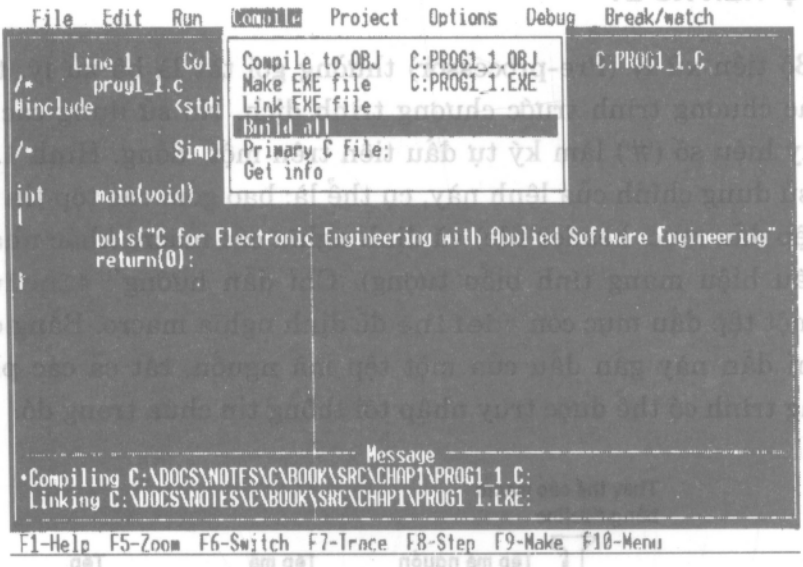
bắt buộc phải được loại bỏ, trong khi đối với các lời cảnh báo thì yêu cầu đặt ra là chỉ nên loại bỏ.

Turbo C phiên bản 2.0 là một gói phần mềm phát triển đã được tích hợp để thích hợp với các hệ thống trên cơ sở máy tính cá nhân. Nó bao gồm: bộ soạn thảo, chương trình dịch, bộ liên kết và trình gỡ rối (được sử dụng để chạy thử chương trình). Bộ soạn thảo được dùng để viết và sửa đổi tệp mã nguồn và được khởi động bằng cách cho chạy tệp chấp hành TC.EXE. Hình 1.3 mô tả màn ảnh chính với một tệp mã nguồn FILE1.C.



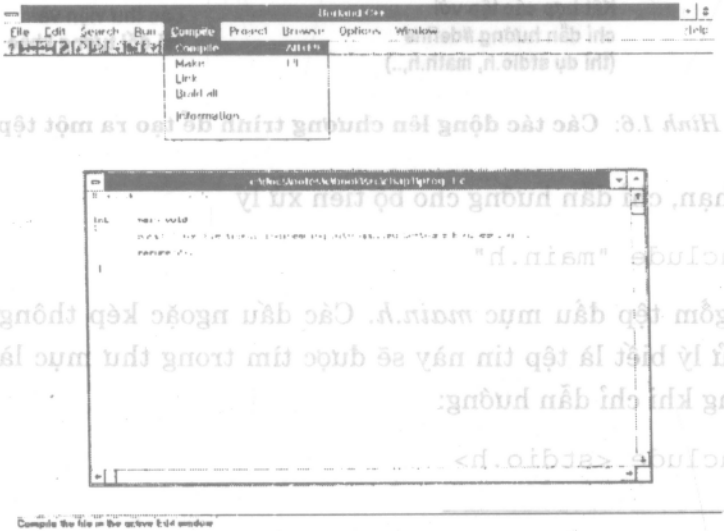
Hình 1.3: **Màn hình của Turbo C phiên bản 2.0.**

Hình 1.4 mô tả các tùy chọn thực đơn biên dịch trong gói phần mềm này. Một tệp mã nguồn được biên dịch bằng cách lựa chọn Compile to OBJ. Nếu như không có lỗi thì một tệp mã đối tượng sẽ được tạo ra (trong trường hợp này là FILE1.OBJ). Tệp này được liên kết bằng cách sử dụng tệp Link EXE file (khi tạo ra tệp FILE1.EXE). Một quá trình biên dịch và liên kết cũng có thể được khởi tạo bằng cách sử dụng tùy chọn Make EXE file.



Hình 1.4: **Tuỳ chọn thực đơn dịch của Turbo C phiên bản 2.0.**

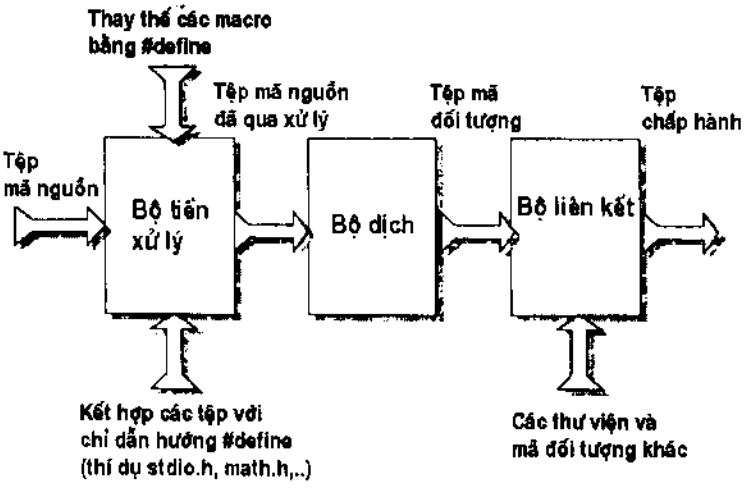
Hình 1.5 giới thiệu màn hình của gói phần mềm được phát triển tích hợp để chạy trên môi trường Windows của hãng Microsoft. Gói phần mềm này là Borland C++ chạy với phiên bản Windows 3.0.



Hình 1.5: **Borland C++ cho phiên bản Windows 3.0.**

1.5 BỘ TIỀN XỬ LÝ

Bộ tiền xử lý (Pre-processor) thường gọi tắt là bộ xử lý, tác động lên các chương trình trước chương trình dịch. Nó sử dụng các lệnh có một ký hiệu số ('#') làm ký tự đầu tiên trên một dòng. Hình 1.6 mô tả cách sử dụng chính của lệnh này, cụ thể là: bao gồm các tệp tin đặc biệt (các tệp đầu mục: header file) và định nghĩa các macro khác nhau (hoặc các dấu hiệu mang tính biểu tượng). Chỉ dẫn hướng*) `#Include` bao gồm một tệp đầu mục còn `#define` để định nghĩa macro. Bằng cách đặt các chỉ dẫn này gần đầu của một tệp mã nguồn, tất cả các phần của chương trình có thể được truy nhập tới thông tin chứa trong đó.



Hình 1.6: **Các tác động lên chương trình để tạo ra một tệp tin.**

Chẳng hạn, chỉ dẫn hướng cho bộ tiền xử lý

```
#include "main.h"
```

sẽ bao gồm tệp đầu mục *main.h*. Các dấu ngoặc kép thông báo cho bộ (tiền) xử lý biết là tệp tin này sẽ được tìm trong thư mục làm việc hiện tại, trong khi chỉ dẫn hướng:

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

*) Tiếng Anh: Directive, có tài liệu dùng chữ Pseudo instruction. Thuật ngữ chỉ dẫn hướng trong tiếng Việt có tài liệu gọi là chỉ thị hoặc dẫn hướng. Các lệnh tác động lên bộ xử lý còn chỉ dẫn hướng là để hướng dẫn trình dịch, (ND).

sẽ bao gồm tệp *stdio.h* được tìm thấy trong thư mục bao gồm mặc định. Thông thường thì thư mục này được tự động đặt bằng hệ thống. Chẳng hạn, Turbo C phiên bản 2.0 cất giữ các tệp đầu mục, theo mặc định, trong thư mục `\TC\INCLUDE` còn Borland C sử dụng thư mục `\BORLANDC\INCLUDE`. Điển hình là các tệp đầu mục trên hệ thống với hệ điều hành UNIX được cất giữ trong thư mục `/usr/include`.

Cuối cùng, cặp dấu ngoặc kép (" ") thông báo cho bộ xử lý để tìm kiếm tệp đầu mục đã chỉ định trong thư mục hiện tại (hoặc thư mục chỉ định trong tên đường dẫn). Các dấu so sánh (<>) hay còn gọi là ký tự "hình lon" thông báo cho bộ xử lý để tìm kiếm trong thư mục `include` mặc định. Không nên để bất kỳ một tệp khác ở xa tệp đầu mục. Các tệp này có phần mở rộng là ".h" (mặc dù điều này là không bắt buộc). Các tệp đầu mục chuẩn đã sử dụng trong mối liên kết với các hàm chứa trong các thư viện. Các tệp này không chứa mã chương trình, nhưng có các thông tin liên quan với các hàm. Một tập hợp cho trước của các hàm, chẳng hạn *maths* (toán học) hoặc I/ 0, đều có một tệp đầu mục liên kết với nó. Bảng 1.1 liệt kê các tệp tin đầu mục tiêu biểu và chức năng của chúng.

Bảng 1.1: **Các tệp đầu mục tiêu biểu.**

Tệp đầu mục	Chú giải
ctype.h	sự phân loại và sự chuyển đổi
math.h	các hàm math (toán học)
stddef.h	định nghĩa vài kiểu dữ liệu chung và vĩ mô
stdio.h	các đoạn chương trình vào/ ra (I/ 0), chẳng hạn nhập vào từ bàn phím, xuất ra màn hình và xử lý tệp tin -file handling (stdio là các chữ viết tắt từ standard input/ output)
stdlib.h	các đoạn chương trình pha trộn
string.h	hàm quản lý xâu
time.h	các hàm thời gian

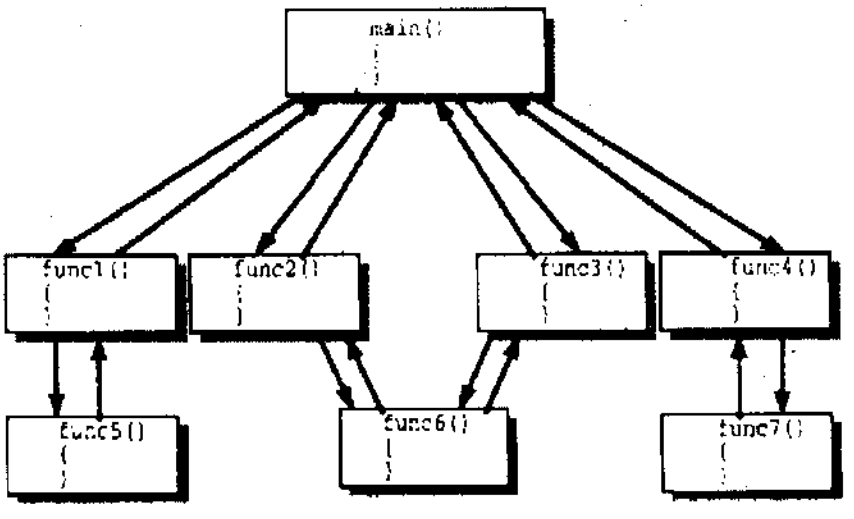
Một macro thay thế mỗi lần xuất hiện một dấu hiệu (thẻ) nào đó bằng một dấu hiệu (thẻ) được chỉ định rõ khác. Các thí dụ sau đây chỉ ra sự thay thế khi sử dụng chỉ dẫn hướng `#define`.

```
#define    PI                3.14
#define    BEGIN (bắt đầu)   {
#define    END (kết thúc)    }
#define    _sqr(x)           ((X)*(X))
#define    SPEED_OF_LIGHT    3e8
```

Bình thường thì khi viết chương trình, các định nghĩa của các hằng số, chẳng hạn như số π, được viết bằng chữ in (chữ hoa).

1.6 CẤU TRÚC

Thông thường thì chương trình được chia thành những nhiệm vụ nhỏ được gọi là các *hàm*. Có những đoạn mã được phân định một cách rõ ràng để thực hiện những tác động riêng biệt. Hàm chính `main()` là đoạn chương trình (routine) cơ bản dùng để điều khiển tiến trình của chương trình và những hàm con (subfunction). Hình 1.7 mô tả một hàm chính khi gọi các hàm khác.



Hình 1.7: **Cấu trúc theo kiểu môđun.**

Bảng 1.2: **Tên các hàm.**

Tên hàm	VALID	Ghi chú
`calc_impedance_RC()`	✓	hàm đã được đặt tên thuận tiện cho việc giải thích xem hàm làm gì
`3_point_rms()`	✗	bắt đầu với một ký tự không hợp lệ
`get average value()`	✗	không gian đã được sử dụng trong tên gọi.
`show_memory`	✗	không có dấu ngoặc đơn ở cuối của tên hàm.
`$temp1`	✗	bắt đầu với một ký tự không hợp lệ
`calc1()`	✓	tên hợp lệ, nhưng khó xác định xem hàm này làm cái gì
`calculateimpedanceofRC()`	✓	khó đọc tên của hàm này; nó tỏ ra là thuận tiện để rút từng từ và chèn vào các dấu gạch dưới để phân định chúng
`calc_boolean_eq()`	✓	tốt hơn thí dụ vừa qua, làm cho người dùng dễ dàng đọc hàm này khi tính phương trình Boole.
`CalcImpedanceRC()`	✓	một kiểu chung sử dụng các chữ hoa để biểu thị chỗ bắt đầu của một từ mới
`do()`		Từ khoá C

Các tên hàm có thể phân biệt đến 31 ký tự (các tên với nhiều hơn con số này, sẽ phụ thuộc vào sự hoàn thiện của chương trình dịch). Tất cả các tên hàm được kế tiếp bởi một tập hợp mở của các dấu ngoặc và không nên là một từ khóa dự trữ của C. Ký tự đầu tiên của tên phải là một chữ cái ('a' - 'z'; 'A' - 'Z') tiếp theo là các chữ cái, các con số, các dấu gạch dưới ('_') hoặc ký hiệu đô la ('$'). Các ký tự khác như xoá trái (blankspace) hoặc dấu trống `tab` (tabspace) là không hợp lệ. Khi viết chương trình, các dấu gạch dưới thường được sử dụng để ngắt đoạn tên hàm với mục đích làm cho dễ đọc. Bảng 1.2 giới thiệu một số tên hàm hợp lệ và không hợp lệ.

Chương trình 1.1 là một chương trình dùng làm thí dụ, trong đó sử dụng hàm `puts()` để hiển thị dòng văn bản "C for Electronic Engineering" ("Lập trình C trong kỹ thuật điện tử"). Hàm `puts()` là một hàm chuẩn đã sử dụng để xuất văn bản ra màn hình; tệp đầu mục gắn bó với hàm này là *stdio.h*. Tệp đầu mục này được sử dụng với chỉ dẫn hướng `#include`.

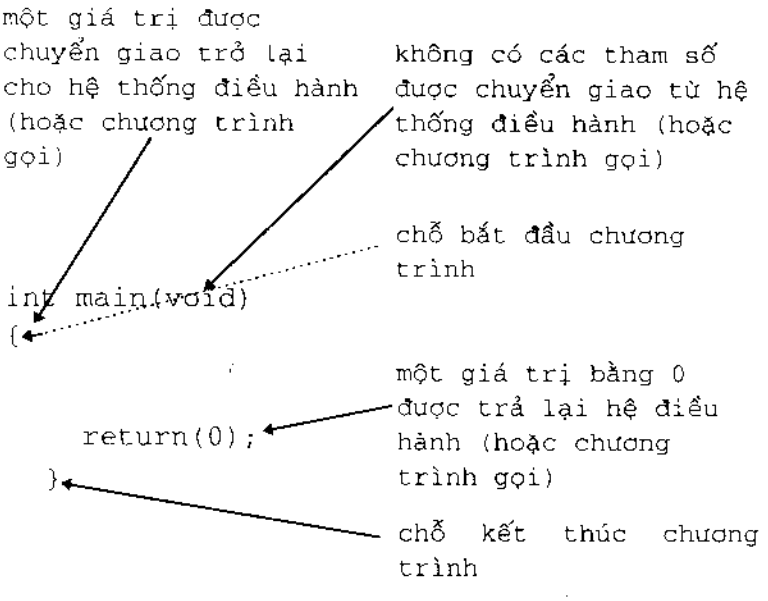
Dấu kết thúc lệnh (;) được sử dụng để kết thúc một dòng mã (hoặc lệnh) và dấu ngoặc nhọn để chỉ chỗ bắt đầu ({) và chỗ kết thúc (}) của một khối mã. Lời chú giải được chèn trong chương trình giữa một ký hiệu nhận dạng bắt đầu (/*) và một ký hiệu nhận dạng kết thúc (*/) lời chú giải.

Tất cả các chương trình C đều có một hàm chính `main()`, định nghĩa điểm nhập vào chương trình và bằng các hàm được gọi, điều khiển toàn bộ tiến trình của chương trình. Nó có thể được đặt ở bất kỳ chỗ nào trong chương trình mã nguồn, nhưng thường được đặt ở gần đầu của tệp tin mà nó được định vị (làm cho nó dễ dàng tìm hơn). Từ khóa `Int` đặt trước `main()` xác định rằng chương trình trả lại một giá trị cho hệ điều hành (hoặc chương trình được gọi). Trong trường hợp này, giá trị trả lại bằng 0 (`return(0)`). Bình thường thì một giá trị trả lại khác 0 được sử dụng khi chương trình bị thoát ra do có lỗi, giá trị hiện tại của giá trị này cho ta một chỉ dẫn để biết vì sao chương trình đã thoát ra. Từ `void` bên trong dấu ngoặc đơn của phần `main()` xác định rằng không có trao đổi dữ liệu giữa chương trình và hệ điều hành, thí dụ không có các giá trị được chuyển giao vào chương trình.

Chương trình 1.1

```
/* prog1_1.c                                          */
#include <stdio.h>

int  main (void)
{
  puts ("C for Electronic Engineering");
        /* C trong kỹ thuật điện tử */
  return(0);
}
```

Hình 1.8 chỉ ra rằng chỗ bắt đầu và kết thúc của chương trình được định nghĩa bên trong hàm `main()`.



Hình 1.8: **Các điểm bắt đầu và kết thúc của một chương trình.**

1.7 SỐ VÀ CÁCH BIỂU DIỄN SỐ

1.7.1 CÁC SỐ ÂM

Một cách viết, được gọi là lấy bù theo 2, biểu diễn các số âm (hoặc các giá trị số nguyên). Trong cách biểu diễn này các digit nhị phân là số '1' trong cột bit có giá trị cao nhất nếu như số đó là số âm, nếu không thì bằng '0'. Để biến đổi một số sang cách viết bù theo 2 phần độ lớn của số âm được biểu diễn dưới dạng nhị phân. Tiếp theo, tất cả các bit được lấy đảo và cộng thêm '1'. Thí dụ dưới đây minh họa cách biểu diễn 16 bit theo cách viết lấy bù theo 2 của số thập phân - 65.

+ 65	00000000 01000001
Lấy bù (đảo)	11111111 10111110
Thêm 1	11111111 10111111

Như vậy, - 65 là 11111111 10111111 trong cách biểu diễn bù theo 2, 16 bit. Bảng 1.3 cho thấy là: với 16 bit, phạm vi các giá trị có thể được

biểu diễn theo cách lấy bù theo 2 bao gồm các giá trị từ -32.767 đến 32.768 (nghĩa là có 65.536 giá trị).

Bảng 1.3: **Cách viết lấy bù theo 2, 16 bit.**

Thập phân	Lấy bù theo 2
-32.768	10000000 00000000
-32,767	10000000 00000001
: : : :	: : : :
- 2	11111111 11111110
- 1	11111111 11111111
0	00000000 00000000
1	00000000 00000001
2	00000000 00000010
: : : :	: : : :
32,766	01111111 11111110
32,767	11111111 11111111

1.7.2 SỐ THẬP LỤC PHÂN VÀ BÁT PHÂN

Bảng 1.4: **Chuyển đổi thập phân, nhị phân, bát phân và thập lục phân.**

Thập phân	Nhị phân	Bát phân	Thập lục phân
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Không có khả năng biểu diễn các mẫu bit trong ngôn ngữ C khi sử dụng khuôn mẫu nhị phân. Có một giải pháp là phải biểu diễn các digit nhị phân trong cách biểu diễn thập lục phân (hệ cơ số 16) hoặc bát phân (hệ cơ số 8). Kỹ thuật này cũng trợ giúp để giảm số lượng các ký hiệu được sử dụng để biểu diễn giá trị nhị phân cỡ một phần tư đối với hệ thập lục phân và một phần ba đối với hệ bát phân. Bảng 1.4 chỉ ra quan hệ chuyển đổi cơ bản giữa các số theo các hệ thập phân, nhị phân, bát phân và thập lục phân.

Trong cuốn sách này, một số nhị phân được chỉ định rõ bằng chữ b đứng kế tiếp các con số, một số bát phân bằng chữ o còn một số trong hệ thập lục phân bằng chữ h. Để biểu diễn một digit nhị phân như một số hệ thập lục phân thì giá trị nhị phân được phân thành các nhóm bốn bit (bắt đầu từ bit có giá trị nhỏ nhất). Một giá trị tương đương hệ sơ số 16 thay thế từng nhóm nhị phân. Chẳng hạn, để biểu diễn số nhị phân 01110101 11000000b các bit được chia ra thành từng nhóm bốn digit để chuyển đổi theo cách sau đây:

Nhị phân	0111	0101	1100 . 0000
Thập lục phân	7	5	C 0

Như vậy, 75C0h biểu diễn số nhị phân 01110101 11000000b. Để chuyển đổi từ hệ thập phân sang hệ thập lục phân, giá trị thập phân được chia liên tiếp cho 16 và ghi lại các số dư. Số dư đầu tiên chính là digit có giá trị nhỏ nhất và số dư cuối cùng chính là digit có giá trị lớn nhất, hay nói khác đi là các số dư được viết lần lượt từ phải qua trái. Chẳng hạn, số thập lục phân tương đương với số phập phân 1103 có thể tìm ra như sau:

```
16 | 1103
     68    rF <<< LSD ( chữ số có nghĩa nhỏ nhất)
     4     r4
     0     r4 <<<MSD ( chữ số có nghĩa nhất)
```

Như vậy số thập phân 1103 tương đương tới 044Fh.

Để chuyển đổi một giá trị nhị phân sang bát phân các bit được chia ra thành các nhóm; mỗi nhóm ba digit, sau đó mỗi nhóm được biểu diễn bằng giá trị bát phân tương đương với nó. Chẳng hạn:

Nhị phân	0	111	010	111	000	000
Bát phân	0	7	2	7	0	0

Trong ngôn ngữ C, các hằng số tính theo hệ thập lục phân được đặt trước bằng số 0 và ký tự 'x', nghĩa là (0x), trong khi một số bát phân được đặt trước chỉ bởi số 0. Dưới đây là một thí dụ về các khuôn mẫu số khác nhau:

Số	Cơ số
0xf2c	Thập lục phân
0432	Bát phân
321	Thập phân

Các số tương đối lớn hoặc quá nhỏ có thể được biểu diễn dưới dạng hàm số mũ của e. Bảng 1.5 đưa một vài thí dụ cho khuôn mẫu này.

Bảng 1.5: **Sự chuyển đổi khuôn mẫu số mũ.**

Số (hoặc hằng số vật lý)	Khuôn mẫu số mũ
0,0000000001	1e-9
1234320	1,23432 e 6
1000000000000	1 e 12
0,023	2,3 e-2
0,943230	9,4323 e-1
Điện tích của điện tử	1,602 e-19
Khối lượng của điện tử	9,109 e-31
Hằng số điện môi của không gian tự do	8,854 e-12
Tốc độ ánh sáng	2,998 e 8

1.8 CÁC HẰNG KÝ TỰ

Các ký tự được cất giữ bằng cách sử dụng hoặc là mã ASCII hoặc là mã EBCDIC. Tên mã ASCII có nguồn gốc từ các chữ cái đầu của tên gọi *American Standard Code for Information Interchange* (Mã tiêu chuẩn của Hoa Kỳ dùng cho việc trao đổi thông tin) còn mã truyền tin EBCDIC là từ *Extended Binary Coded Decimal Interchange Code* (Mã trao đổi thập phân mở rộng được mã hoá nhị phân). Bảng 1.6 đưa ra một danh sách đầy đủ của các ký tự ASCII.

Bảng 1.6: **Bộ ký tự mã ASCII.**

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	NL	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Các ký tự ASCII từ 0 đến 32 tính theo hệ thập phân là các ký tự không được in ra (non-printing) và được sử dụng để tạo khuôn mẫu lối ra hoặc để điều khiển phần cứng. Chương trình 1.2 hiển thị ký tự ASCII khi nhập vào một giá trị thập phân. Hàm `printf()` được sử dụng để hiển thị ký tự ASCII và `scanf()` được sử dụng để cho giá trị thập phân. Các hàm này sẽ được bàn luận chi tiết hơn trong chương 2.

Chương trình 1.2

```
/* prog1_2.c                                        */
/* Program to display an ASCII charracter           */
/* for an entered decimal value                     */
/* Chương trình để hiển thị một ký tự ASCII         */
/* khi một giá trị thập phân được nhập vào          */
```

```
int  main(void)
{
int  value;
  printf("Enter a decimal value >>");
        /* Nhập vào một giá trị thập phân */
  scanf("%d",&value);
  printf("Equivalent ASCII character is %c\n",value);
  return (0);
}
```

Kết quả chạy thử 1.1 giới thiệu một thí dụ làm mẫu. Trong trường hợp này giá trị thập phân được nhập vào là 65, dẫn đến một giá trị tương đương dưới dạng mã ASCII là 'A'.

Chạy thử 1.1

```
Enter a decimal value(0-255)>>>65
Equivalent ASCII character is 'A'
(Ký tự ASCII tương đương là 'A')
```

Khi muốn cất giữ các ký tự như mã ASCII ta lưu trữ bằng các digit nhị phân gắn (associate) với ký tự. Thí dụ, mã ASCII cho ký tự 'A' là 65 trong hệ thập phân (0x41); như vậy ô nhớ nhị phân dùng cho ký tự này là 0100 0001. Một xâu các ký tự "R1" được cất giữ như mẫu bit minh hoạ trên hình 1.9; ký tự NULL được sử dụng để báo hiệu sự kết thúc một xâu.

01010010	01100101	01110011	00100000	00110001	00000000
'R'	'e'	's'	SPACE	'1'	NULL (Kết thúc xâu)

Hình 1.9: **Ô nhớ mã ASCII dùng cho xâu "Res 1".**

Một vài thí dụ về mã ASCII được giới thiệu trong bảng 1.7.

Bảng 1.7: **Các thí dụ về ký tự ASCII.**

Thập phân	Hex	Nhị phân	Đặc tính
32	0x20	0010 0000	Dấu trống
65	0x41	0100 0001	'A'
66	0x42	0100 0010	'B'
90	0x5A	0101 1010	'Z'
97	0x61	0110 0001	'a'
122	0x7A	0111 1010	'z'
7	0x07	0000 0111	Tiếng chuông (BELL)
8	0x08	0000 1000	Xoá cột bên trái

Cặp dấu phẩy trên bao một ký tự đơn, thí dụ `'a'`, trong khi cặp dấu ngoặc kép bao một xâu ký tự, thí dụ "C for Electronic Engineering" ("Lập trình C trong kỹ thuật điện tử"). Một số hằng ký tự, không in ra, được định nghĩa bằng một dấu sổ ngược (\). Các hằng ký tự này được cho trong bảng 1.8.

Bảng 1.8: **Các định nghĩa ký tự được sử dụng trong C.**

Chức năng	Macro ASCII	Ký tự
Tiếng chuông	BELL	`\a`
Dòng mới	NL	`\n`
Dấu trống TAB	HT	`\t`
Xoá trái (backspace)	BS	`\b`
Sang trang mới	FF	`\f`
Dấu sổ ngược	\	`\\`
Ngoặc kép	"	`\"`
Dấu phẩy trên	'	`\'`

1.9 KIỂU DỮ LIỆU

Các biến trong một chương trình có thể được lưu trữ hoặc dưới dạng các số hoặc dưới dạng các ký tự. Thí dụ, điện trở của một đoạn dây đồng

được lưu trữ như một con số (một giá trị thực) còn tên của một linh kiện (thí dụ "R1") được cất giữ như các ký tự. Bảng 1.9 giới thiệu bốn kiểu dữ liệu cơ bản xác định khuôn mẫu của các biến.

Bảng 1.9: **Các kiểu dữ liệu cơ bản.**

Kiểu	Cách sử dụng
`char`	Ký tự đơn `'a'`, `'1'`, vân vân.
`int`	Số nguyên có dấu
`float`	Dấu phảy động đơn chính xác
`double`	Dấu phảy động kép chính xác

Có ba loại mở rộng cơ bản dùng cho bốn kiểu, đó là:

```
short
long
unsigned implementation
```

Bảng 1.10: **Một số giới hạn tiêu biểu cho các kiểu dữ liệu.**

Kiểu	Bộ nhớ (byte)	Phạm vi
`char`	1	- 128 đến 127
`unsigned char`	1	0 đến 255
`int`	2	-32.768 đến 32.767
	hoặc	hoặc
	4	-2.147.483.648 đến 2.147.483.647
`unsigned int`	2	0 đến 65535
	hoặc	hoặc
	4	0 đến 4.294.967.295
`short int`	2	-32.768 đến 32.767
`long int`	4	-2.147.483.648 đến 2.147.483. 647
`float`	4 (điển hình)	$\pm$ 3.4x10^{-38} đến $\pm$ 3.4x10^{38}
`double`	8 (điển hình)	$\pm$ 1.7x10^{-311} đến $\pm$ 1.7x10^{311}
`long double`	10 (điển hình)	$\pm$ 3,4x10^{-4932} đến $\pm$ 1,1x10^{4932}

Số nguyên là một giá trị bất kỳ không chứa dấu phẩy thập phân; vùng giá trị của số nguyên phụ thuộc vào số byte được sử dụng để lưu trữ số đó. Một giá trị dấu phẩy động là một số bất kỳ và có thể chứa một dấu phẩy thập phân; giá trị này luôn được mô tả trong khuôn mẫu có dấu. Lại một lần nữa, vùng giá trị phụ thuộc vào số byte được sử dụng.

Bình thường các số nguyên chiếm 2 hoặc 4 byte trong bộ nhớ, tuỳ thuộc vào việc thực hiện (implementation) chương trình dịch. Các số này nằm trong giới hạn từ -32.768 đến 32.767 (đối với `int` 2 byte) và từ -2.147.483.648 đến 2.147.483.647 (đối với `int` 4 byte), tương ứng. Bảng 1.10 đưa cho một số giới hạn tiêu biểu cho các kiểu dữ liệu.

1.10 KHAI BÁO CÁC BIẾN

Một chương trình sử dụng các biến để lưu trữ dữ liệu. Trước khi chương trình có thể sử dụng một biến thì tên biến và kiểu dữ liệu của biến phải được khai báo. Một dấu phẩy chia nhóm các biến có cùng kiểu dữ liệu. Chẳng hạn, nếu như một chương trình yêu cầu các biến số nguyên `num-steps` và `bit_mask`, các biến dấu phảy động `resisror1` và `resistor2`, và hai biến ký tự `char1` và `char2`, thì phải khai báo như sau:

```
int         num_steps,bit_mask
float       resistor1, resistor2;
char        char1, char2;
```

Chương trình 1.2 là một chương trình đơn giản xác định điện trở tương đương của hai điện trở 1000 Ω và 500 Ω đấu song song với nhau. Chương trình có chứa ba khai báo dấu phẩy động dành cho các biến `resistor1`, `resistor2` và `eq_resistance`.

Chương trình 1.3

```
/* prog1_3.c                                              */
/* Chương trình xác định điện trở tương đương của */
/* hai điện trở 1000 Ôm và 500 Ôm mắc song song */

#include <stdio.h>,
int  main(void)
{
```

```
float resistor1, resistor2,equ_resistance;

  resistor1=1000.0;
  resistor2=500.0;
  equ_resistance=1,01(1.01resistor1+1.0/resistor2);
  printf("Equivalent resistance is %f\n",equ_resistance);
  return (0);
}
```

Cũng có thể phải gán một giá trị ban đầu cho một biến ở một điểm trong chương trình, mà tại đó nó được khai báo; công việc này được gọi là sự khởi tạo biến hay thiết lập trạng thái ban đầu cho biến. Chương trình 1.4 cho ta một thí dụ với các biến khai báo là `resistor1`, và `resistor2` được khởi tạo bằng `1000,0` và `500,0`, tương ứng.

Chương trình 1.4

```
/* prog1_4.c                                          */
/* Chương trình xác định điện trở tương đương */
/* của hai điện trở 1000 và 500 Ôm mắc song song */
#include <stdio.h>

int main(void)
{
float resistor1=1000.0, resistor2=500.0,equ_resistance;

  equ_resistance=1.0/(1.0/resistor1+1.0/resistor2);
  printf("Equivalent resistance is %f \n",equ_resistance);
  return(0);
}
```

1.11 CÁC TOÁN TỬ TRONG NGÔN NGỮ C

Ngôn ngữ C có một tập hợp rất phong phú của các toán tử; nhưng có thể chia thành bốn kiểu chính:

- ❒ Số học;
- ❒ Lôgic;
- ❒ Xử lý tới bit;
- ❒ So sánh.

1.11.1 TOÁN TỬ SỐ HỌC

Các toán tử số học thực hiện các phép toán trên các giá trị số. Các phép toán số học cơ bản là: cộng (+), trừ (-), nhân (*), chia (/) và chia theo môđun (%). Phép chia theo môđun đưa ra số dư của một phép chia số nguyên. Sau đây là cú pháp cơ bản của hai toán hạng (operand) với một toán tử số học.

```
toán hạng   toán tử   toán hạng
```

Toán tử gán giá trị (-) được sử dụng khi một biến 'chiếm một giá trị' của một phép toán. Các toán tử shorthand khác được sử dụng với nó, bao gồm cộng bằng (+=), trừ bằng (-=), nhân bằng (*=), chia bằng (/=) và môđun bằng (%=). Các thí dụ sau đây minh họa cho cách sử dụng các phép toán này.

Lệnh	Tương đương
`x+=3.0;`	`x=x+3.0;`
`voltage/=sqrt(2);`	`voltage=voltage/sqrt(2);`
`bit_mask *=2;`	`bit_mask=bit_mask*2;`
`screen_val%=22+1;`	`screen_val=screen_val%22+1;`

Trong nhiều ung dụng ta lại cần phải thêm hoặc bớt đi một lượng bằng 1. Nhằm mục đích này ngôn ngữ C đã có hai toán tử đặc biệt; ++ để tăng và -- để giảm đi một lượng.

Lệnh	Tương đương
`no_values--;`	`no_values=no_values-1;`
`i--;`	`i=i-1;`
`screen_ptr++;`	`screen_ptr=screen_ptr+1;`

Các toán tử này có thể hoặc đặt trước hoặc đặt sau biến. Nếu như đặt trước biến, thì việc tăng/ giảm trước một lượng, trong khi đặt sau

biến thì tăng/ giảm sau một lượng. Các thí dụ sau đây minh hoạ cho cách sử dụng của các toán tử này.

Khi đoạn mã dùng làm thí dụ sau đây được chấp hành thì các giá trị của i, j, k, y và z sẽ tương ứng là 10, 12, 13, 10 và 10. Lệnh z=--i sẽ giảm i và gán giá trị này vào z (tăng trước), trong khi y=i++ sẽ gán giá trị của i vào y và sau đó các giảm i (giảm sau).

```
i = 10; j=11; k=12;
y=i++;  /* gán i vào y sau đó tăng i       */
z=--i;  /* giảm i sau đó gán vào z */
j++;    /* tăng j  */
++k;    /* tăng k */
```

Bảng 1.11 tóm tắt các toán tử số học.

Bảng 1.11: **Các toán tử số học.**

Toán tử	Phép toán	Thí dụ
-	Phép trừ hoặc dấu trừ	5-4→1
+	Phép cộng	4 + 2 → 6
*	Phép nhân	4*3→12
/	Phép chia	4/2→2
%	Lấy môđun	13%3→1
+=	Cộng bằng	% += 2 là tương đương với x=x+2
-=	Trừ bằng	x-=2 tương đương với x=x-2
/=	Chia bằng	x/=y tương đương với x=x/y
*=	Nhân bằng	x *= 32 tương đương với x=x*32
=	Gán giá trị	x = 1
++	Tăng (increment)	Count++ tương đương với Count=Count+1
--	Giảm (decrement)	Sec-- tương đương với Sec=Sec-1

1.11.2 TOÁN TỬ SO SÁNH

Các toán tử so sánh hay còn gọi là toán tử quan hệ (relationship) xác định liệu kết quả của một phép so sánh là đúng (TRUE) hay sai (FALSE). Các toán tử này là:

- lớn hơn (>),
- lớn hơn hoặc bằng với (>=),
- nhỏ hơn (<),
- nhỏ hơn hoặc bằng với (<=),
- bằng với (= =) và
- không bằng với (! =).

Bảng 1.12 liệt kê các toán tử so sánh.

Bảng 1.12: **Các toán tử so sánh.**

Toán tử	Chức năng	Thí dụ	Điều kiện đúng
>	lớn hơn	(b > a)	khi b lớn hơn a
>=	lớn hơn hoặc bằng	(a >= 4)	khi a lớn hơn hoặc bằng 4
<	nhỏ hơn	(c < f)	khi c nhỏ hơn f
<=	nhỏ hơn hoặc bằng	(x <= 4)	khi x nhỏ hơn hoặc bằng 4
= =	bằng	(x = = 2)	khi x bằng 2
! =	không bằng với	(y ! = x)	khi y không bằng x

1.11.3 TOÁN TỬ LÔGIC (ĐÚNG hoặc SAI)

Phép toán lôgic là một phép toán trong đó một quyết định được lựa chọn còn tuỳ thuộc toán đã được thực hiện là TRUE (đúng) hay FALSE (sai). Nếu cần, một vài phép toán so sánh có thể lập thành nhóm cùng nhau để có được tính năng cần thiết.

Bảng 1.13: **Các toán tử lôgic.**

Toán tử	Hàm	Thí dụ	Điều kiện đúng
&&	AND	((x==1) && (y<2))	khi x bằng 1 và y nhỏ hơn 2
||	OR	((!=b) 11 (> 0))	khi a khác b hoặc a lớn hơn 0
!	NOT	(i (a>0))	khi a không lớn hơn 0

Ngôn ngữ C giả thiết rằng một giá trị số bằng 0 là SAI (FALSE) còn các giá trị bất kỳ khác là ĐÚNG (TRUE). Bảng 1.13 liệt kê các toán tử lôgic.

Phép toán lôgic AND (VÀ) chỉ dẫn đến kết quả TRUE khi tất cả các toán hạng đều là TRUE. Bảng 1.14 dẫn ra kết quả toán tử VÀ (&&) đối với phép toán `Operand1 && Operand2`.

Bảng 1.14: **Bảng giá trị chân lý lôgic AND.**

Toán hạng1	Toán hạng2	Toán hạng3
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Phép toán lôgic HOẶC (OR) dẫn đến kết quả TRUE nếu bất kỳ một toán hạng là TRUE. Bảng 1.15 dẫn ra kết quả lôgic của toán tử OR (||) đối với lệnh `operand1 || operand2`.

Bảng 1.15: **Bảng giá trị chân lý lôgic OR.**

Toán hạng1	Toán hạng2	Toán hạng3
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Bảng 1.16 đưa ra kết quả lôgic của toán tử NOT (!) đối với lệnh `!operand`.

Bảng 1.16: **Bảng giá trị chân lý lôgic NOT.**

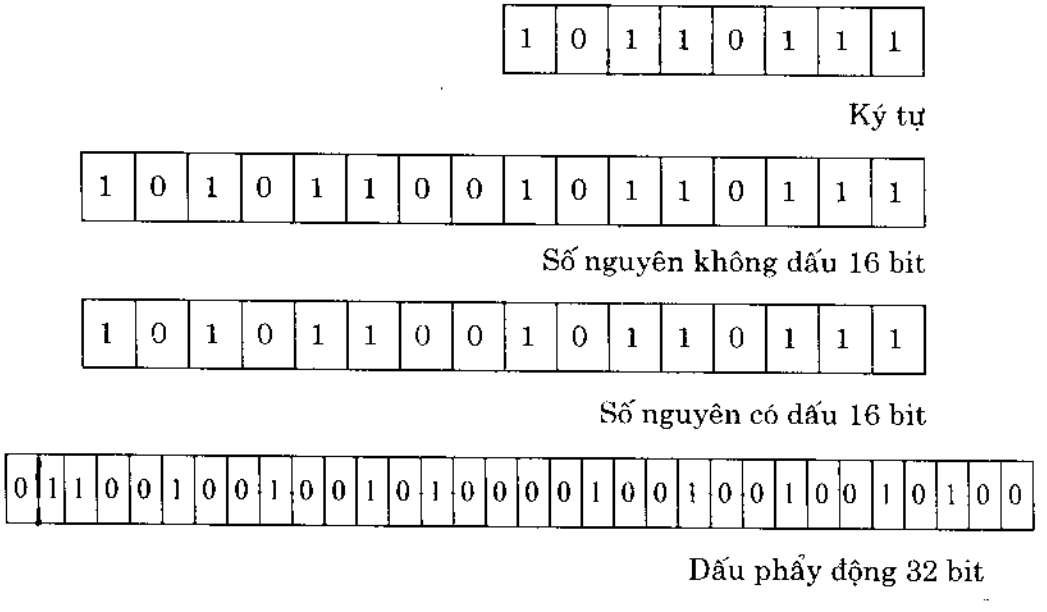
Toán hạng	Kết quả
FALSE	TRUE
TRUE	FALSE

Chẳng hạn, nếu a có giá trị bằng 1 và b cũng bằng 1 thì các lệnh so sánh sau đây sẽ được áp dụng:

Lệnh	Kết quả
`(a==1) && (b==1)`	TRUE
`(a>1) && (b==1)`	FALSE
`(a==10) || (b==1)`	TRUE
`!(a==12)`	TRUE

1.11.4 TOÁN TỬ XỬ LÝ TỚI BIT

Các toán tử xử lý tới bit (*bitwise*) tương tự với các toán tử lôgic nhưng không được nhầm lẫn chúng với nhau bởi vì phép toán của chúng khác nhau. Các toán tử xử lý tới bit tác động trực tiếp lên các bit riêng lẻ của một (hoặc các) toán hạng, trong khi các toán tử lôgic xác định liệu một điều kiện là ĐÚNG hay SAI.



Hình 1.10: **Các khuôn mẫu thí dụ đối với các kiểu dữ liệu khác nhau.**

Các giá trị số được cất giữ như mẫu bit trong một khuôn mẫu số nguyên không dấu khác, số nguyên có dấu (lấy bù theo 2) hoặc cách viết (ký hiệu) dấu phẩy động (một số mũ và phần định trị). Thông thường các ký tự được cất giữ như các ký tự mã ASCII. Hình 1.10 mô tả thí dụ của các khuôn mẫu khác nhau.

Các phép toán xử lý tới bit cơ bản là: VÀ (&), HOẶC (1), lấy bù theo 1, hoặc phép xử lý đảo tới bit (-), XOR(^), phép dịch trái (<<) và phép dịch phải (>>). Bảng 1.17 đưa các kết quả của phép toán xử lý tới bit VÀ trên hai bit: `bit1` và `bit2`.

Bảng 1.17: **Bảng giá trị chân lý của phép toán xử lý tới bit AND.**

Bit1	Bit2	Kết quả
0	0	0
0	1	0
1	0	0
1	1	1

Bảng 1.18 chỉ ra bảng giá trị chân lý dùng cho phép toán xử lý tới bit OR trên hai bit: `bit1` và `bit2`.

Bảng 1.18: **Bảng giá trị chân lý của phép toán xử lý tới bit OR.**

Bit1	Bit2	Kết quả
0	0	0
0	1	1
1	0	1
1	1	1

Bảng 1.19 đưa bảng giá trị chân lý đối với lý phép toán xử lý tới bit EX-OR trên hai bit: `bit1` và `bit2`.

Bảng 1.19: **Bảng giá trị chân lý của phép toán hoặc-loại trừ (EX-OR).**

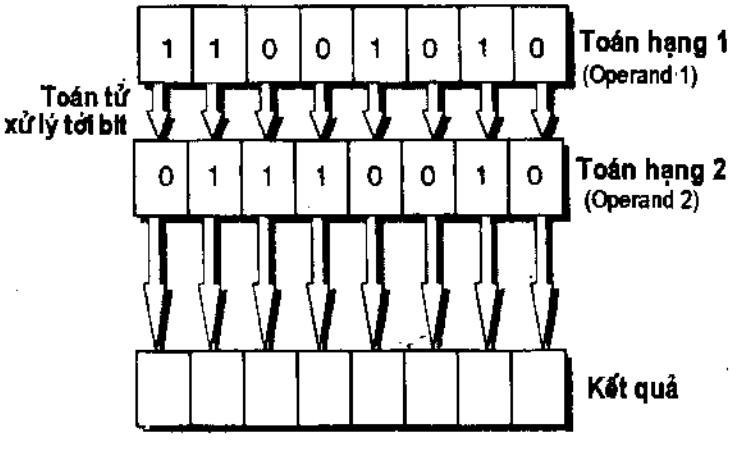
Bit1	Bit2	Kết quả
0	0	0
0	1	1
1	0	1
1	1	0

Bảng 1.20 giới thiệu một bảng chân lý dùng cho toán tử xử lý tới bit NOT trên một bit đơn.

Bảng 1.20: **Bảng giá trị chân lý của phép toán xử lý tới bit NOT.**

Bit	Kết quả
0	1
1	0

Các toán tử xử lý tới bit tác động lên từng bit riêng lẻ của các toán hạng, như được chỉ ra trên hình 1.11.



Hình 1.11: **Phép toán xử lý tới bit trên hai toán hạng.**

Chẳng hạn, nếu như hai số nguyên thập phân 58 và 41 (khi giả thiết rằng các giá trị nhị phân không dấu tám bit) được thao tác khi sử

dụng các toán tử xử lý tới bit: VÀ, HOẶC và EX-OR, thì kết quả sau đây sẽ được áp dụng.

	VÀ	HOẶC	EX-OR
58	00111010	00111010	0111010
41	00101001	00101001	00101001
Kết quả	10101000	00111011	00010011

Các kết quả của các phép toán xử lý tới bit này như sau:

```
58 & 41 = 40        (thí dụ  00101000)
58 | 41 = 59        (thí dụ  00111011)
58 ^ 41 = 19        (thí dụ  00010011)
```

Toán tử lấy bù theo 1 tác động lên một toán hạng đơn. Chẳng hạn, nếu như một toán hạng có giá trị bằng 17 (00010001) thì phần bù theo 1 của giá trị này, trong hệ nhị phân, sẽ là 11101110.

Để thực hiện phép dịch chuyển bit SHIFT, các toán tử << và >> đã sử dụng. Các toán tử này dịch chuyển các bit trong toán hạng đi một số bước cho trước được định nghĩa bằng một giá trị cho ở bên phải của phép toán. Toán tử dịch sang trái (<<) chuyển chỗ các bit của toán hạng sang trái và các số 0 sẽ lấp đầy kết quả ở bên phải. Toán tử dịch sang phải (>>) chuyển chỗ các bit của toán hạng sang phải và các số 0 sẽ lấp đầy kết quả ở bên trái nếu số nguyên là dương; còn nếu không phải là dương thì các số 1 sẽ lấp đầy kết quả. Khuôn mẫu chuẩn là:

```
operand >> no_of_bit_shift_positions
operand << no_of_bit_shift_positions
(toán hạng >> số vị trí dịch chuyển bit
toán hạng << số vị trí dịch chuyển bit)
```

Chẳng hạn, nếu như y = 59 (00111011), thì y >> (00000111) còn với y<<2 đến 236 thì là (11101100),

Bảng 1.21 đưa kết quả tóm lược của các toán tử xử lý tới bit cơ bản.

Bảng 1.21: **Các toán tử xử lý tới bit.**

Thí dụ	Hàm	Toán tử
&	VÀ	`c = A & B`
|	HOẶC	`f = Z | Y`
^	HOẶC LOẠI TRỪ	`h = 5 ^ f`
~	lấy bù theo 1	`x = ~y`
>>	dịch phải	`x = y >> 1`
<<	dịch trái	`x = y << 2`

Các thí dụ sau sử dụng dạng viết ngắn gọn của các toán tử xử lý tới bit:

i <<= 2	tương đương với i = i<<2	*dịch các bit của i đi 2 vị trí về bên trái.*
time |= 32	tương đương với time = time | 32	*OR các bit của time với 32 hệ thập phân*
bitval ^= 22	tương đương với bitval ^= bitval^22	*bitval được lấy EX-OR với 22*

1.12 QUYỀN ƯU TIÊN

Có một số quy tắc được áp dụng cho các toán tử, đó là:

- Hai toán tử, không kể phép gán giá trị, sẽ không bao giờ được đặt kề nhau. Chẳng hạn x*% 3 là không hợp lệ;
- Việc chia nhóm được thực hiện bằng các dấu ngoặc; bất cứ cái gì bên trong dấu ngoặc đều sẽ được đánh giá trước. Các dấu ngoặc lồng vào nhau cũng có thể được sử dụng để đặt các quyền ưu tiên;
- Các toán tử cũng có mức ưu tiên khác nhau. Các toán tử với mức ưu tiên cao hơn được đánh giá trước; nếu hai toán tử có cùng mức ưu tiên, thì toán tử ở bên trái được đánh giá trước.

Các mức ưu tiên cho các toán tử như sau:

QUYỀN ƯU TIÊN CAO NHẤT

( ) [ ]	hàng đầu (primary)
! - ++ -- -	một toán hạng (unary)
* / %	nhân (multiply)
+ -	cộng (additive)
<< >>	dịch chuyển (shift)
< > <= >=	so sánh (relation)
== !=	bằng (equality)
&	xử lý tới bit
^	
|	
&&	lôgic
||	
= += -=	gán giá trị

QUYỀN ƯU TIÊN THẤP NHẤT

Toán tử gán giá trị có mức ưu tiên thấp nhất. Thí dụ sau đây chỉ ra cho thấy các toán tử được ưu tiên như thế nào trong một lệnh (=> chỉ ra các bước khi xác định kết quả):

```
23 + 5 % 3 / 2 << 1        =>
23 + 2 / 2 << 1            =>
23 + 1 << 1                =>
23 + 2                     => 25
```

1.13 CHUYỂN ĐỔI KIỂU DỮ LIỆU

Khi các kiểu dữ liệu khác nhau hỗn hợp trong một phép toán với hai toán hạng, quy tắc sau đây xác định kiểu dữ liệu của kết quả:

1. Kiểu ký tự bất kỳ (chẳng hạn như `char`, `unsigned char`, `signed char` và `short int`) biến đổi sang số nguyên (`int`).

2. Cách khác, nếu một trong hai toán hạng là `long double`, thì toán hạng kia sẽ biến đổi sang `long double`.

3. Cách khác, nếu một trong hai toán hạng là `double`, thì toán hạng kia sẽ biến đổi sang `double`.

4. Cách khác, nếu một trong hai toán hạng là float, thì toán hạng kia sẽ biến đổi sang float.

5. Cách khác, nếu một trong hai toán hạng là `unsignet long`, thì toán hạng kia sẽ biến đổi sang `unsignet long`.

6. Cách khác, nếu một trong hai toán hạng là `long int`, thì toán hạng kia sẽ biến đổi sang `long int`.

7. Cách khác, nếu như hoặc toán hạng là `unsigned int` (số nguyên không dấu) thì toán hạng kia biến đổi thành `unsigned int`.

8. Cách khác, cả hai toán hạng đều là kiểu `int`.

Một kiểu dữ liệu của biến có thể bị thay đổi một cách tạm thời khi sử dụng một kỹ thuật có tên là: casting (đúc) hoặc coercion (sự bắt buộc). Từ bổ nghĩa `cast` được đặt trước toán hạng và kiểu dữ liệu được định nghĩa trong dấu ngoặc. Các từ bổ nghĩa tiêu biểu là `(float)`, `(int)`, `(char)` và `(double)`. Trong chương trình 1.5, hai số nguyên b và c được chia cho nhau và kết quả được gán vào a. Bởi vì b và c đều là số nguyên, nên quy tắc 8 sẽ được áp dụng. Như vậy, kết quả sẽ là 1, bởi vì một phép chia số nguyên được thực hiện.

Chương trình 1.5

```
/* progl_5.c
#include <stdio.h>
{
float a;
int   b, c;
  b=6; c=11;
  a = c / b;
  printf("a = %f",a);
```

```
    return(0);
}
```

Chương trình 1.6 thực hiện một phép chia dấu phẩy động biến c đã được tính lại (recast) hoặc ép buộc (coerced) thành một float. Như vậy quy tắc 4 được áp dụng.

Chương trình 1.6

```
/* prog1_6.c                                          */
#include <stdio.h>

int main(void)
{
floata;
int b,c;

  b=6; c=11;
  a = (float)c 1b;
  printf("a = %f",a);
  return(0);
}
```

1.14 TỪ KHÓA

ANSI-C có không nhiều từ khóa được dự trữ (chỉ có 32); các từ này không thể được sử dụng làm các ký hiệu nhận dạng (identifier) chương trình và phải viết bằng chữ thường. Các chương trình lớn có thể được xây dựng từ các khối xây dựng theo cách đơn giản này. Dưới đây là danh sách các từ khóa.

auto	do	for	return	switch
break	double	goto	short	typedef
case	else	if	signed	union
char	enum	int	sizeof	unsigned
const	extem	long	static	void
continue	float	register	struct	volatile
default				while

Các hàm là các đoạn (section) mã thực hiện một tác động đã được chỉ rõ. Các hàm này nhận được một vài đầu vào và tạo ra đầu ra theo cách "đọc chính tả" bởi cách hoạt động của chúng. Chúng có thể là các hàm chuẩn, được chèn vào trong các thư viện hoặc được người lập trình viết ra. ANSI-C định nghĩa một số hàm chuẩn mà cung cấp các đầu vào/ ra cơ bản tới/ từ bàn phím và màn hình, các hàm toán học, quản lý (handling) ký tự v. v... Chúng lập thành nhóm với nhau trong thư viện các tệp tin và không có sẵn (intrinsic) trong ngôn ngữ. Các thư viện này liên kết vào trong một chương trình để tạo ra một chương trình chấp hành (excutable).

1.15 MỘT SỐ THUẬT NGỮ

Sau đây sẽ giải nghĩa một vài thuật ngữ sử dụng trong chương này và các chương sau.

Argument — *Đối số*. Đối số là các giá trị hiện thời chuyển giao cho một hàm. Các dấu phẩy được sử dụng để ngăn cách các đối số*) trong một lời gọi hàm *(tên_hàm (arg1, arg2,..., argn))*.

Array — *Mảng*. Một mảng được sử dụng để cất giữ một tập hợp các biến có cùng một kiểu dữ liệu dưới một tên chung.

Block — *Khối*, xem thêm lệnh hỗn hợp.

Comment — *Lời chú giải*. Lời chú giải là các đoạn văn bản được bổ sung vào một chương trình để giúp cho việc giải thích hoạt động của chương trình. Trình dịch bỏ qua tất cả các lời chú giải.

Compiler — *Chương trình dịch*. Một trình dịch biến đổi một chương trình C sang một dạng khác, mà máy tính có thể hiểu được (thí dụ mã máy).

Compound statement — *Lệnh hỗn hợp*. Một lệnh hỗn hợp là một số các lệnh được gửi kèm theo bởi dấu ngoặc nhọn ({ }).

*) Trong tiếng Việt nhiều tài liệu vẫn gọi chung là tham số, giống như parameter. (ND)

Constant *Hằng*. Hằng biểu diễn một số hoặc một ký tự cố định, vì vậy còn chia ra: hằng số và hằng ký tự.

Declaration *Sự khai báo*. Khai báo là định nghĩa kiểu và tên của một biến.

Function prototype *Hàm đặt làm mẫu*. Một hàm đặt làm mẫu xác định cú pháp của một hàm liên quan với nó phải được sử dụng như thế nào.

Definitions *Định nghĩa*. Một định nghĩa giới thiệu một hoặc nhiều tên ký hiệu nhận dạng (identifier) vào trong một chương trình.

Expression statement *Lệnh biểu thức*. Một biểu thức được kế tiếp bằng một dấu chấm phẩy được mô tả như các một lệnh biểu thức.

Expression *Biểu thức*. Một biểu thức là một dãy các toán tử, các toán hạng và các dấu ngắt câu (punctuator), chỉ rõ quá trình tính toán.

Function *Hàm*. Một hàm thực hiện nhiệm vụ đã được định nghĩa. Hàm có một giao diện xác định trong đó các đối số chuyển giao vào các biến, được gọi là tham số.

Identifier *Ký hiệu nhận dạng*. Các ký hiệu nhận dạng là tên đặt cho các thứ như hàm, biến, v. v...

Initialization *Sự khởi tạo, sự thiết lập trạng thái ban đầu*, được dùng để chỉ sự thiết lập trạng thái ban đầu hoặc đặt giá trị ban đầu cho một biến, một mảng, v. v...

Iteration *Sự lặp*. Các lệnh lặp cho phép tạo thành vòng của một tập hợp các lệnh (`while()`, `do...while()` và `for()`).

Keyword *Từ khóa*. Từ khoá là các từ được dự trữ cho các mục đích riêng (special).

Linker *Bộ liên kết*. Một bộ liên kết bổ sung thêm thông tin vào một chương trình để làm cho nó trở thành một chương trình chấp hành.

Macro *Macro*. Một macro thay thế một ký hiệu nhận dạng thẻ cho một dãy các thẻ.

Operand *Toán hạng, ôpêran*. Các toán hạng được tác động lên bởi các toán tử.

Operator *Toán tử*. Các toán tử thực hiện một tác động xác định chẳng hạn như các phép toán: số học, xử lý tới bit, lôgic và so sánh.

Parameter *Tham số, thông số*. Các tham số là các biến được định nghĩa trong một đầu mục hàm để giữ (hold) các giá trị đã được chuyển giao

Pointer *Con trỏ*. Một con trỏ cất giữ một địa chỉ bộ nhớ.

Punctuators *Dấu ngắt câu*. Dấu ngắt câu còn được gọi là các dấu ngăn cách, bao gồm: dấu ngoặc đơn (( )), dấu ngoặc vuông ([ ]), dấu ngoặc nhọn ({ }), dấu hai chấm (:), dấu phẩy (,) và dấu chấm phẩy (;). Dấu ngoặc đơn có thể nhóm lại các biểu thức, chỉ thị sự gọi hàm và ngăn cách các biểu thức có điều kiện. Các dấu ngoặc vuông xác định các xâu. Dấu ngoặc nhọn chỉ ra chỗ bắt đầu và kết thúc của một số các lệnh được chia thành nhóm như một lệnh hỗn hợp. Dấu hai chấm chỉ báo lệnh đã được gắn nhãn còn một dấu phẩy ngăn cách các sự khai báo biến.

Selection *Sự lựa chọn, chọn lọc*. Lệnh lựa chọn thực hiện việc chọn lựa từ các tiến trình luân phiên nhau của các tác động bằng cách kiểm tra các giá trị nào đó (`if()...else` và rẽ nhánh `switch()`.

Statement *Lệnh*. Các lệnh dùng để điều khiển tiến trình của một chương trình.

Structure *Cấu trúc*. Một cấu trúc là tập hợp của các thành viên (hoặc các phần tử) được người dùng định nghĩa.

Variable *Biến*. Các biến cất giữ các giá trị và các ký tự mà chương trình sử dụng

1.16 THỰC HÀNH

Q1.1 Chỗ bắt đầu và kết thúc của một chương trình C được định nghĩa như thế nào ?

Q1.2 Bộ xử lý dịch chỉ dẫn hướng *include* sau đây như thế nào ?

```
#include <stdio.h>
#include "main.h"
```

Q1.3 Xác định mã ASCII bát phân, thập phân và thập lục phân cho các phím sau đây:

- ❒ Esc (Thoát)
- ❒ DEL (Xóa)
- ❒ '~' (dấu ngã)
- ❒ '_' (đường gạch chân)
- ❒ NULL (Ký tự trắng)

Q1.4 Ngôn ngữ C sử dụng các ký tự đặc biệt nào cho các trường hợp sau đây:

- ❒ một dòng mới
- ❒ một dấu trống tab
- ❒ một dấu phảy trên ('), còn gọi là dấu chân ngỗng.
- ❒ dấu ngoặc kép (")

Q1.5 Chương trình 1.7 cho phép trả lời trong câu hỏi này đã được kiểm tra bằng cách thay thế giá trị 0x12 bằng giá trị cần có.

Chương trình 1.7

```
/* prog1_7.c                                          */
#include <stdio.h>
int main (void)
{
int x=0x12;
printf("Decimal %d, Hex %x, Octal %o \n",x,x,x);
return (0);
}
```

Điền vào các chỗ trống trong bảng sau bằng cách cho các giá trị thập phân, thập lục phân, bát phân tương đương

Thập phân	Thập lục phân	Bát phân
15		
201		
14,655		
	0x12	
	0xA1	
	0x1f0	
		013
		027
		0206

Q1.6 Chương trình 1.8 hoặc 1.9 có thể được sử dụng để kiểm tra các kết quả trong câu hỏi này. Hãy thay thế các phép toán được viết bằng kiểu chữ in đậm bằng các lệnh cần có:

Chương trình 1.8

```
/* prog1_8.c                                          */
#include <stdio.h>
int main(void)
{
  printf(" The answer is %d ",5*3);
  return(0);
}
```

Chương trình 1.9

```
/* prog1_9.c                                  */
#include <stdio.h>
int    main(void)
{
     x=13;
     y=8;
     printf("The answer is %d",x%y);
     return (0);
}
```

(a) Xác định các kết quả của các phép toán sau đây:

1) `21 % 4 % 2 * 3 + 2`
2) `25 + 5 % 2 * 4 - 1`
3) `3 * 3 * 7 % 2`
4) `(7 + 4) % 4 * 2`
5) `25 % 3`

(b) Giả sử rằng x = 14 và y = 8, hãy xác định kết quả của các phép toán sau đây:

1) `x % y + 2`
2) `x++`
3) `--x`
4) `x & y`
5) `x | y`
6) `4*(x + y * 2)`
7) `x << 3`
8) `y >> 1`

(c) Xác định các kết quả của các phép toán sau đây:

1) `0x32' + 011`
2) `0x31 & 044`
3) `0x4B | 013`
4) `011 * 0x31`
5) `-32`
6) `0XAA - 055`
7) `0XFF + 0x10`
8) `! (OXCF)`
9) `~(032)`

Q1.7 Giả sử rằng x=1 và y=2, hãy xác định xem liệu các phép toán sau đây dẫn đến kết quả là đúng (TRUE) hay sai (FALSE). Câu trả lời có thể được kiểm tra bằng cách sử dụng bảng mẫu của chương trình 1.10.

(a) `((x==i) && (y!=2))`

(b) `((x!=1) || (y==2))`

(c) `(! (x==2))`

(d) `(! ((x==1) && (y==2)))`

(e) `((x>o) && (y<2))`

(f) `(X<=1)`

(g) `((Y>1) || (x==1))`

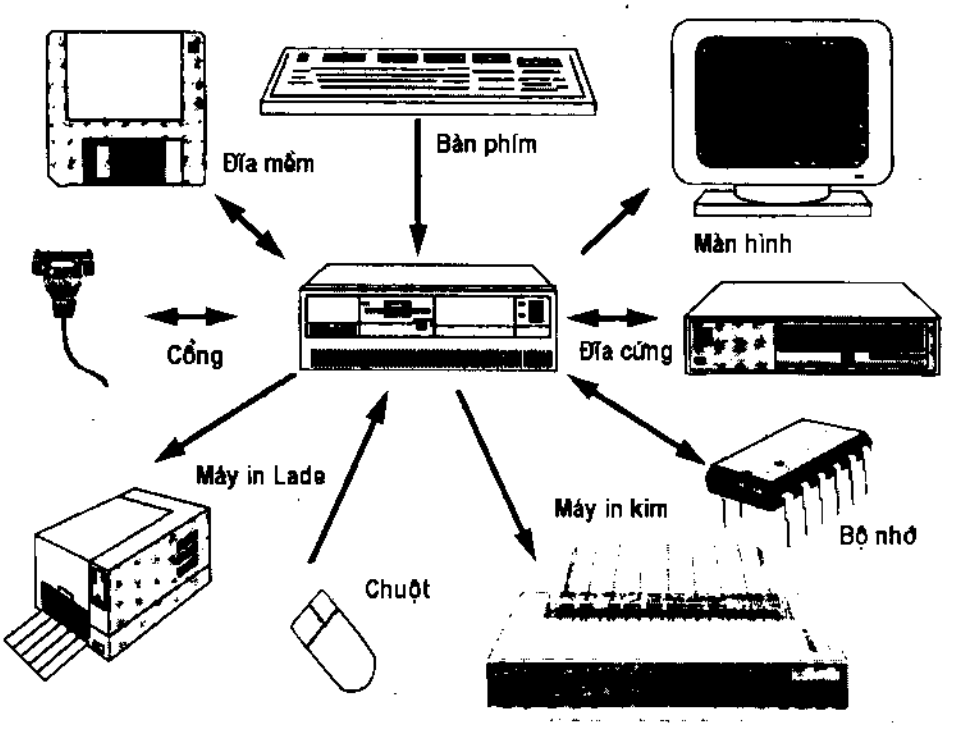
Chương trình 1.10

```
/* progl_10.c                                          */
#include <stdio.h>
int main(void)
{
int x=1,y=2;
if (x==1)      puts("TRUE");
else           puts("FALSE");
return(0);
}
```

Chương 2

NHẬP VÀO VÀ XUẤT RA

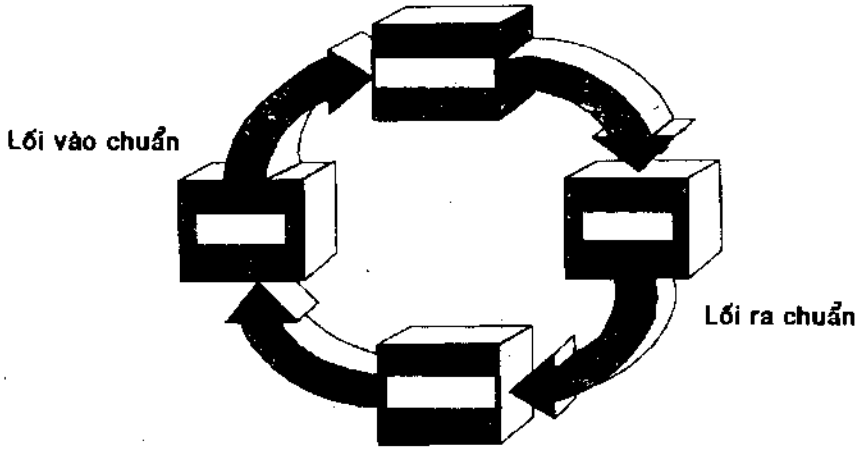
Mỗi chương trình có một vài dạng xuất ra và thông thường một đầu vào. Theo cách nhìn tổng thể thì các thiết bị như bàn phím, một tệp tin, các cổng vào ra như cổng nối tiếp hoặc cổng song song, chuột, v. v..., đều có thể cung cấp dữ liệu nhập vào chương trình. Dữ liệu xuất ra có thể được gửi tới những thiết bị như màn hình, máy in, đĩa cứng... Hình 2.1 minh hoạ một vài thiết bị nhập vào và xuất ra thường gặp.



Hình 2.1: **Các thiết bị vào/ ra.**

Đáng chú ý là các kỹ sư điện tử cũng trao đổi thông tin với các bộ biến đổi A/ D hoặc D/ A, các điốt phát quang (LED), cũng như với các bộ điều khiển Ethenet, các bộ lập trình cho vi mạch v. v...

Thông thường thì bàn phím một thiết bị vào là mặc định đối với chương trình còn màn hình là thiết bị đầu ra mặc định. Việc định hướng lại đối với đầu vào hoặc đầu ra là hoàn toàn có thể. Chẳng hạn, một tệp văn bản có thể tác động giống như một đầu vào của một chương trình và máy in như một đầu ra. Hình 2.2 mô tả một chu trình nhập vào/ xuất ra, trong đó người sử dụng nhập dữ liệu vào qua bàn phím; chương trình sẽ xử lý các dữ liệu và xuất các kết quả ra màn hình. Người sử dụng lúc đó có thể nhập dữ liệu mới và như vậy chu trình lại tiếp tục.



Hình 2.2: **Chu trình nhập vào/ xuất ra theo chương trình.**

Với ngôn ngữ C các chức năng vào/ ra chuẩn là không có sẵn ở bên trong (intrinsic), mà được lưu trữ trong các thư viện được liên kết trong chương trình. Chỉ dẫn hướng `#Include` bao gồm các tệp đầu mục gắn liền với chúng. Các hàm vào/ ra sử dụng tệp đầu mục *stdio.h*, tệp này thường được xếp trong thư mục mặc định `include`. Để cho phép tất cả các phần mã nguồn truy nhập tới các hàm đã định nghĩa trong tệp đầu mục chỉ dẫn hướng dành cho bộ xử lý đã được sắp xếp ở gần đỉnh của tệp mà trong đó nó được sử dụng. Chương trình dịch lúc đó sẽ kích hoạt chức năng kiểm tra lỗi mỗi khi có hàm vào/ ra chuẩn được sử dụng.

Chương trình 2.1 giới thiệu cho thấy một chương trình chứa tệp *stdio.h* như thế nào.

Chương trình 2.1

```
/* prog2_1.c                                    */
#include <stdio.h>
int main(void)
{
   printf("Nạp vào một giá trị của điện trở");
   return(0);
}
```

2.1 CÁC LỆNH CHUẨN (printf (), put () và putchar ()) DÙNG ĐỂ XUẤT RA

Có ba hàm xuất ra cơ bản, đó là:

`printf` (*"format", arg1, arg2... argn*) xuất ra một xâu văn bản đã định dạng tới lối ra dưới dạng đã được xác định bởi *"format"*- khuôn mẫu khi sử dụng những tham số *arg1...argn*

`puts` (*" xâu "*) xuất ra một xâu văn bản tới lối ra chuẩn và gắn nó với một dòng mới.

`putchar`(*ch*) xuất ra một ký tự đơn (*ch*) tới lối ra chuẩn.

Hàm `printf()` gửi một xâu đã định dạng tới đầu ra chuẩn (màn hình). Xâu này có thể hiển thị các biến đã được định dạng và những ký tự điều khiển đặc biệt, như những dòng mới (`'\n'`), xoá ngược - backspace (`'\b'`) và dấu trống tab (`'\t'`); các ký tự này được liệt kê trong bảng 2.1.

Hàm `put()` viết một xâu văn bản vào đầu ra chuẩn và các biến không được định dạng có thể được sử dụng. Ở chỗ kết thúc văn bản một dòng mới được tự động nối vào.

Chương trình 2.2 sử dụng hàm `put()` để xuất ra văn bản *`Enter a value of resistance`* (nhập vào một giá trị điện trở). Một dòng mới được tự động tạo ra khi kết thúc văn bản này.

Bảng 2.1: **Các ký tự điều khiển đặc biệt (hoặc chuỗi thoát).**

Ký tự	Chức năng
\"	Dấu nháy kép
\'	Dấu nháy đơn
\\	Dấu sổ ngược
\nnn	Ký tự ASCII trong mã bát phân, thí dụ. \ 041 cho '!'
\0xnn	Ký tự ASCII trong mã thập lục phân, thí dụ: \0x41 cho 'A'
\a	Tín hiệu âm thanh (tiếng chuông)
\b	Xoá bên trái
\f	Nạp giấy
\n	Dòng mới (nạp dòng)
\r	Về đầu dòng (Carriage Return)
\t	Dấu trống tab

Chương trình 2.2

```
/* prog2_2.c                                    */
#include <stdio.h>
int main(void)
{
printf ("Enter a value of resistance");
return 0);
}
```

Chương trình 2.3

```
/* prog2_3.c                                    */
#include <stdio.h>
int main(void)
{
printf("Enter a value of resistance \n");
return (0);
```

}

Chương trình 2.3 là thí dụ cho thấy lệnh `printf()` đã được sử dụng như thế nào với một ký tự dòng mới. Lệnh này sẽ viết "nhập vào một giá trị điện trở" *(Enter a value of resistance)* lên màn hình. Sau đó một dòng mới được chiếm chỗ nhờ ký tự `'\n'`. Các ký tự điều khiển đặc biệt sử dụng một dấu sổ ngược để báo cho chương trình thoát ra khỏi cách mà thông thường được dịch. Vì lý do này, tổ hợp dấu sổ ngược và một ký tự đặc biệt được gọi là một chuỗi thoát (Escape).

Các tham số chuyển giao vào trong `printf()` được gọi là các đối số (argument) hoặc tham số; các tham số này được phân ra bởi những dấu phẩy. Chương trình 2.3 có lệnh `printf()` với chỉ một đối số, thí dụ một xâu văn bản. Xâu này được so sánh với xâu thông báo và luôn luôn là tham số đầu tiên của `printf()`. Nó có thể bao gồm các ký tự điều khiển đặc biệt và/ hoặc ký tự điều khiển chuyển đổi tham số.

2.1.1 CÁC KÝ TỰ ĐIỀU KHIỂN ĐẶC BIỆT

Con trỏ trở về đầu dòng (`'\r'`) được sử dụng để đưa vết nháy trên màn hình trở về đầu dòng màn hình (trên nhiều màn hình đó là chỗ tận cùng bên trái của màn hình). Ký tự điều khiển nạp giấy - `form feed` (`'\f'`) dùng để nạp dòng cho các máy in và dấu trống tab (`'\t'`) đẩy vị trí ký tự hiện tại về phía trước một khoảng trống tab.

Một vài ký tự ASCII là không được in (non-printing) hoặc không có những ký tự điều khiển đặc biệt gán cho chúng. Tuỳ chọn `\nnn` cho phép truy nhập tới các ký tự này. Chẳng hạn, `'\177'`(00 111 111) định nghĩa ký tự DEL, `'\007'`(00 000 111) định nghĩa tiếng chuông (Bell) và `'\033'`(00 011 011) là ESC. Chương trình 2.4 có một vài ký tự điều khiển đặc biệt (chẳng hạn một dòng mới và một dấu trống tab) và các số bát phân tương đương (chuông Bell và `'H', 'E', 'L', 'L', 'O'`).

Chương trình 2.4

```
/* prog2_4.c                                          */
#include <stdio.h>
int main(void)
{
```

```
printf("\007");          /* âm của tiếng chuông */
printf("\ HELLO TO YOU \n");
printf("\110\105\114\114\117\041");/* print HELLO ! */
return(0);
}
```

Kết quả chạy thử 2.1 là một thí dụ được đưa ra dưới đây làm mẫu. Khi chương trình làm việc ký tự Bell tạo ra một tiếng chuông ở loa (nếu như hệ thống có). Ký tự này không in ra được và không hiển thị trên màn hình. Dấu trống tab được chèn trước từ "Chào bạn" và một dòng mới xuất hiện trước từ "Xin chào !". Tiếng chuông được tạo ra bởi: `\041` (00 100 001).

Chạy thử 2.1

```
HELLO TO YOU
HELLO !
```

Bảng 2.2 liệt kê các ký tự ASCII xuất ra và mã bát phân tương đương `\nnn` của chúng.

Bảng 2.2: **Mã nhị phân và bát phân của các ký tự ASCII.**

Các ký tự ASCII	Bát phân	Digit nhị phân
BELL	007	0 000 111
'H'	110	1 001 000
'E'	105	1 000 101
'L'	114	1 001 100
'O'	117	1 001 111
'!'	041	0 100 001

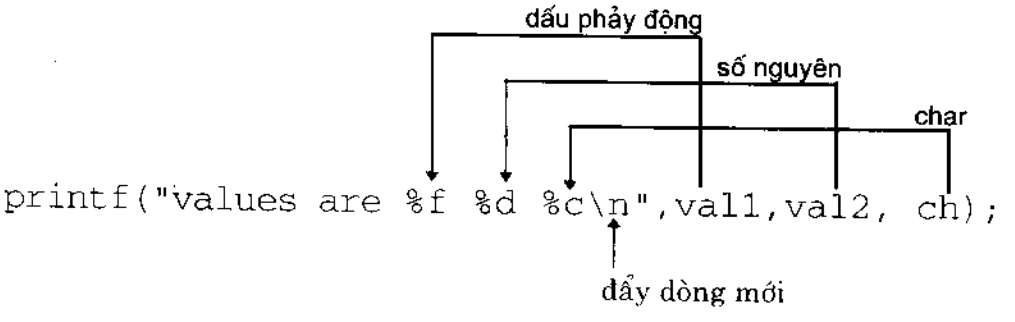
2.1.2 CÁC KÝ TỰ ĐIỀU KHIỂN CHUYỂN ĐỔI

Các ký tự điều khiển chuyển đổi mô tả khuôn mẫu xâu thông báo sử dụng những tham số khác như thế nào. Nếu như `printf()` có nhiều hơn một tham số thì khuôn mẫu đầu ra được xác định bằng cách sử

dụng ký tự phần trăm (%) tiếp theo là ký tự mô tả khuôn mẫu. Số nguyên có dấu sử dụng ký tự điều khiển chuyển đổi %d, còn số nguyên không dấu sử dụng ký tự điều khiển chuyển đổi %u. Dấu phẩy động sử dụng ký tự điều khiển chuyển đổi %f, còn cách viết khoa học*) sử dụng %e. Bảng 2.3 liệt kê các ký tự chính dùng để điều khiển chuyển đổi.

Bảng 2.3: **Các ký tự điều khiển sự chuyển đổi.**

Toán tử	Khuôn mẫu	Toán tử	Khuôn mẫu
%c	ký tự đơn	%s	xâu ký tự
%d	số nguyên thập phân có dấu	%o	số nguyên bát phân không dấu
%e	dấu phẩy động khoa học	%%	ký tự % để in
%f	dấu phẩy động	%x	số nguyên thập lục phân không dấu
%u	số nguyên thập phân không dấu	%g	dấu phẩy động khác hoặc cách viết khoa học



Hình 2.3: **Một thí dụ về lệnh** printf().

Hình 2.3 cho một thí dụ về lệnh printf() với bốn tham số. Tham số đầu tiên là xâu thông báo kế tiếp theo các thông số đã được in trong xâu thông báo đó. Trong trường hợp các tham số là val1, val2 và

*) Trong các công trình khoa học thường có những số rất lớn được biểu diễn dưới dạng hàm mũ, thí dụ: tốc độ ánh sáng bằng 299800 km/gy hay thường viết là $2,988.10^8$ m/s. Để tránh viết dưới dạng hàm mũ vì phải dùng cách viết nhô lên trên dòng người ta thường dùng một quy ước để chuyển số mũ xuống cùng dòng bằng cách viết là 2,998 e8. Cách viết này trong tiếng Anh gọi là *Scientific natation*, tạm gọi là cách viết khoa học. (ND)

ch; val1 được định dạng trong xâu thông báo như một dấu phẩy động (%f), val2 là một số nguyên (%d) còn ch là ký tự (%c). Cuối cùng, một ký tự dòng mới('\n') được sử dụng để đẩy một dòng mới ra đầu ra.

Chương trình 2.5 là một thí dụ cho thấy xâu thông báo được cấu trúc như thế nào khi sử dụng các ký tự điều khiển ký tự. Các số nguyên val1, val2 và val1+val2 dùng ký tự điều khiển %d và dấu phảy động val3 dùng ký tự %f.

Chương trình 2.5

```
/* prog2_5.c                                          */
#include <stdio.h>

int     main (void)
{
int     var1,var2;
float   var3;
   var1 = 5; var2 = 6;
   var3= 15.0;

   printf("The sum of %d and %d is equal to %d\n",
           var1, var2, var1+var2);
   printf("The value of var3 is %f and var1 is
                  %d\n",var3,var1);
   return(0);
 }
```

Kết quả chạy thử 2.2 giới thiệu đầu ra từ chương trình 2.5.

Chạy thử 2.2

```
The sum of 5 and 6 is equal to 11
The value of var3 is 15.000000 and var1 is 5
```

Một giá trị bằng số xuất ra đặc tính đã cho bằng cách sử dụng một dấu hiệu, dấu hiệu này chỉ định rõ số các ký tự được sử dụng để hiển thị

giá trị và số các vị trí sau dấu phẩy thập phân. Khuôn mẫu chung của dấu phẩy động là:

```
%m.nX
```

ở đây: ❒ m là độ rộng giá trị (số các digit trong đó có tính cả dấu phẩy thập phân),

❒ n là số các digit kế tiếp theo dấu phẩy thập phân và

❒ X kiểu khuôn mẫu (f dùng cho dấu phẩy động).

Khuôn mẫu chung của một xâu hoặc số nguyên là :

```
%mX
```

ở đây X là loại kiểu khuôn mẫu (c dùng cho ký tự, s dùng cho chuỗi còn d dùng cho số nguyên) và m là độ rộng của đầu ra. Bảng 2.4 chỉ ra một số thí dụ.

Bảng 2.4: **Thí dụ về dấu hiệu điều khiển chuyển đổi khác nhau.**

Định dạng	Hàm
%.3f	Định dạng dấu phẩy động với ba vị trí thập phân và độ rộng mặc định
%8.3 f	Định dạng dấu phẩy động với 8 vị trí dự trữ và 3 vị trí sau dấu thập phân, thí dụ 32,453
%10d	Định dạng số nguyên cho 10 khoảng dự trữ thí dụ 23
%3o	Định dạng số nguyên bát phân cho 3 ký tự thập lục phân
%10.6 e	Định dạng dạng số mũ với 6 vị trí thập phân

Chương trình 2.6 là thí dụ các định dạng khác nhau cho đầu ra.

Chương trình 2.6

```
/* prog2_6.c                                          */
#include <stdio.h>

int  main(void)
{
```

```
float  X;
int    i;
    x=31.43523;
    i = 143;
    printf("Value of x is <%f> <%8.2f> <%.3f>\n",x,x,x);
    printf("Value of i is <%d> <%10d>\n",i,i);
    return(0);
}
```

Kết quả chạy thử 2.3 chỉ ra một thí dụ chạy mẫu chương trình này. Trong trường hợp này, số mặc định của các vị trí thập phân dùng cho dấu phẩy động là 6.

Chạy thử 2.3

```
Value of x is <31.435230 > <       31.44 > <31.435>
Value of i is  <143 >    <              143 >
```

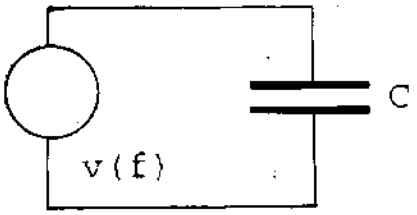
2.1.3 MỘT SỐ THÍ DỤ

Điện kháng của tụ điện

Điện kháng của một tụ điện phụ thuộc tần số của điện áp đặt vào. Ở tần số thấp điện kháng cực kỳ lớn và ở tần số cao điện kháng lại nhỏ. Điện kháng (Xc) của một tụ điện quan hệ với điện dung C (Farad), ở tần số của tín hiệu đặt vào f (Hertz) theo công thức:

$$X_C = \frac{1}{2\pi . fC} \quad [\Omega]$$

Hình 2.4 mô tả sơ đồ của mạch điện.



Hình 2.4: **Tụ điện được đấu vào nguồn điện áp hình sin.**

Chương trình 2.7 xác định dung kháng tụ điện ở tần số sử dụng. Trong trường hợp này, điện dung bằng 1 μF (1e-6) và tần số bằng 10 kHz (10 e3). Chỉ dẫn hướng #define đã được sử dụng để xác định hằng số π.

Chương trình 2.7

```
/* prog2_7.c                                          */
/* Chương trình để tính dung kháng                    */
/* theo tần số và điện dung                           */

#include <stdio.h >
#define PI 3.14159
int     main (void)
{
float   freq, cap, X_c;

freq=10.0e3;              /* 1kHz          */
cap=1.0e-6;              /* 1 microF      */
X_c=1/(2.0*PI*freq*cap);

/* print freq with 3 decimal places in exponent format */
/* print cap with 2 decimal places in exponent format  */
/* print x_c as floating point format with 3 decimal   */
printf ("frequency %8.2e Hz, capacitance %8.3e Farad\n")
               freq, cap);
printf("Capacitive reactance is %8.3f ohms\n", X_c);
return (0);
}
```

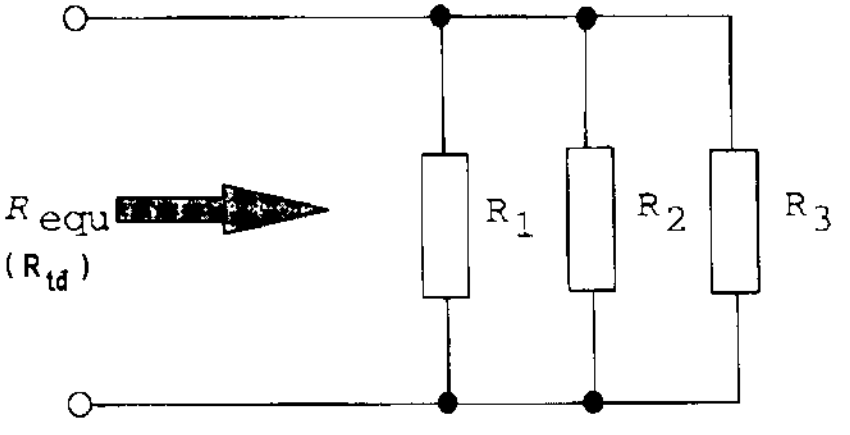
Kết quả chạy thử 2.4 giới thiệu một thí dụ đầu ra của chương trình. Khuôn mẫu số mũ (`%8.2e`) được sử dụng để hiển thị tần số, khuôn mẫu này có độ rộng bằng 8 và 2 vị trí sau dấu phẩy thập phân; điện dung cũng được hiển thị trong khuôn mẫu số mũ với 3 vị trí thập phân. Biến dung kháng (`X_c`) dùng khuôn mẫu dấu phẩy động với 8 ký tự dự trữ dùng để trả lời và 3 vị trí thập phân (`%8.3f`). Số của các ký tự dự trữ cho giá trị là một số tối thiểu và nếu nhiều hơn được yêu cầu thì chương trình sẽ tự động dùng chúng.

Chạy thử 2.4

```
Tần số 1.00 e + 04 Hz, điện dung 1.000 e - 06 Farad
Dung kháng là 15.916 ôm
```

Các điện trở mắc song song

Chương trình 2.8 dùng để xác định điện trở tương đương của ba điện trở mắc song song. Hình 2.5 mô tả một sơ đồ của tập hợp này. Các điện trở là R_1, R_2, và R_3, và điện trở lối vào tương đương R_{td}



Hình 2.5: **Ba điện trở mắc song song.**

Công thức để xác định điện trở tương đương là:

$$R_{td} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}} \quad [\Omega]$$

Chương trình 2.8 xác định điện trở tương đương của ba điện trở: 1 kΩ (1e3), 500 Ω và 250 Ω.

Chương trình 2.8

```
/* prog2_8.c                                          */
/* Chương trình để xác định điện trở tương đương     */
/* của ba điện trở mắc song song                      */
#include <stdio.h>

int       main(void)
{
float     R1,R2,R3,R_equ;
     R1=1.0e3; /*  1 kohms      */
     R2=500.0; /*  500 ohms     */
     R3=250.0; /*  250 ohms      */

     puts("Chương trình xác định điện trở tương đương của");
     puts("ba điện trở mắc song song");

     R_equ=1/(1/R1+1/R2+1/R3);

     printf("R1=%8.3f, R2=%8.3f, R3=%8.3f\n",R1,R2,R3);
     printf("Equiv. resistance is %8.3f omhs\n", R_equ);
     return(0);
}
```

Kết quả thử chạy 2.5 cho thấy điện trở tương đương của mạch này là 142,857 Ω.

Chạy thử 2.5

```
Chương trình để xác định điện trở tương đương của ba
điện trở được mắc song song R1=1000.000, R2= 500.000,
R3= 250.000
Điện trở tương đương là 142.857 ohms
```

Tần số cộng hưởng của mạch LC nối tiếp

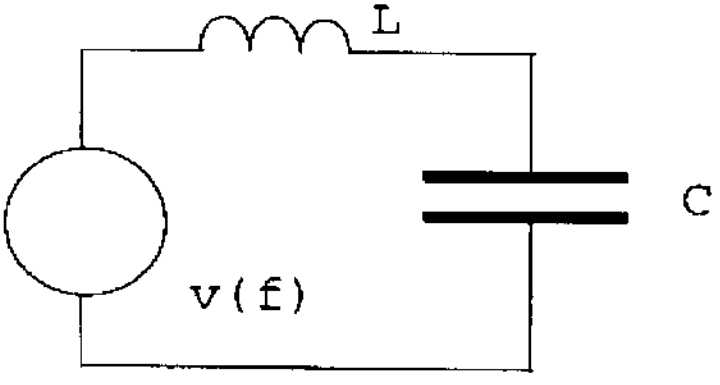
Khi cuộn cảm và tụ điện mắc nối tiếp không bị tổn hao, trở kháng vào là :

$$Z = j.\left[2\pi.fL - \frac{1}{2\pi.fC}\right] \quad [\Omega]$$

Ở tần số thấp trở kháng vào cao vì có tụ điện và ở tần số cao nó cũng cao vì có cuộn cảm. Ở tần số riêng, điện kháng của cuộn cảm và tụ điện bằng nhau và như vậy sẽ triệt tiêu nhau. Tần số này xuất hiện khi mạch cộng hưởng, kết quả là trở kháng vào bằng không. Tần số cộng hưởng cho bởi công thức :

$$f_{ch} = \frac{1}{2\pi\sqrt{LC}} \quad [\text{Hz}]$$

Hình 2.6 mô tả sơ đồ mạch này.



Hình 2.6: **Mạch LC nối tiếp.**

Chương trình 2.9 xác định tần số cộng hưởng của mạch LC nối tiếp với điện cảm bằng 1 mH và điện dung bằng 1 μF. Các giá trị L và C ở thời điểm này đã được khai báo trong chương trình.

Chương trình sử dụng hàm căn bậc hai (`sqrt()`) vì vậy tệp đầu mục *math.h* được đặt sau `#include`. Việc sử dụng cách viết #include này giúp chương trình dịch kiểm tra khuôn mẫu các giá trị gửi cho hàm

như nó kiểm tra cú pháp chung của lời gọi hàm. Nó cũng thông báo cho chương trình dịch biết là giá trị được trả lại là dấu phẩy động (điều này sẽ được thảo luận chi tiết hơn trong chương sau).

Chương trình 2.9

```
/* prog2_9.c                                          */
/* Chương trình xác định tần số cộng hưởng của        */
/* mạch LC nối tiếp                                   */

#include <stdio.h>
#include <math.h> /* requid for sqrt(0)               */

#define PI 3.14159
int      main (void)
{
float     L=1e-3,C=1e-6,f_res,

   /* L is 1 mH, C is 1 uF    */

   puts(*Chương trình để tính tần số cộng hưởng của mạch nối tiếp")
   puts("Mạch LC");

   f_res=1/(2*PI*sqrt(L*C));

   printf("C=%.3e F, L=%.3e H\n" C, L);
   printf("tần số cộng hưởng là %.3f Hz\n",f_res);
   return (0);
  }
```

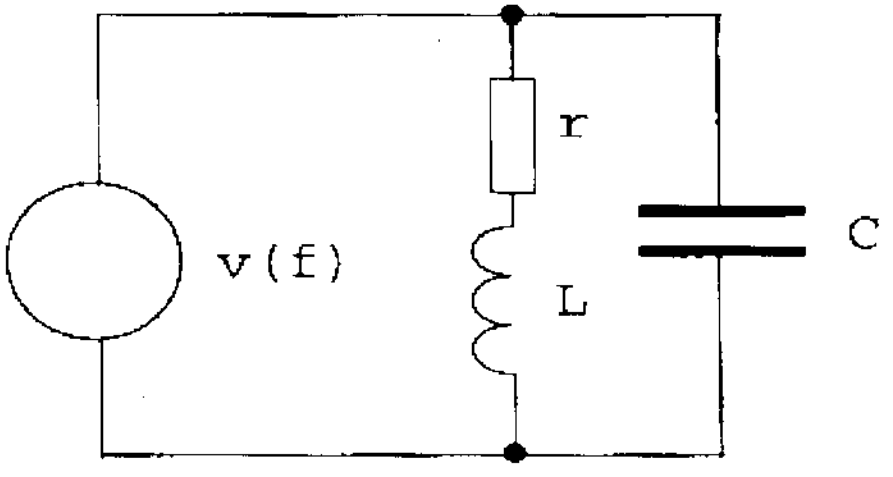
Kết quả chạy thử 2.6 giới thiệu một thí dụ làm mẫu.

Chạy thử 2.6

```
Chương trình tính tần số cộng hưởng của
các mạch LC nối tiếp
C=1.000e-06 F, L=1.000e-03 H
Tần số cộng hưởng là 5032.958
```

Tần số cộng hưởng của mạch LC song song

Trong mạch LC song song hiệu ứng xảy ra trái ngược với trường hợp mạch LC nối tiếp. Ở tần số thấp thì trở kháng thấp do điện cảm bị tụ điện làm cho ngắn mạch. Ở tần số cao, tụ điện lại làm ngắn mạch cuộn cảm và cũng dẫn đến trở kháng thấp. Ở tần số riêng mạch cộng hưởng dẫn đến trở kháng vào rất cao. Hiện tượng này làm cho dòng trong mỗi nhánh của mạch gần như bằng nhau và ngược chiều; như vậy có dòng vào rất nhỏ. Hiện tượng này được gọi là cộng hưởng song song. Một cuộn cảm không tổn hao có điện trở bằng 0 và không có dòng vào (nếu như tụ điện không tổn hao). Đương nhiên mạch cộng hưởng song song hoàn hảo không có trong thực tế vì thường có một giá trị điện trở trong cuộn cảm và tụ điện cũng có đôi chút tổn hao. Hình 2.7 giới thiệu sơ đồ mạch LC song song.



Hình 2.7: **Mạch LC mắc song song.**

Tần số cộng hưởng xác định bằng phương trình:

$$f_{ch} = \frac{1}{2\pi}\sqrt{\frac{1}{LC} - \frac{r^2}{4L^2}} \quad [\text{Hz}]$$

Chương trình 2.10 xác định tần số cộng hưởng của một mạch LC mắc song song.

Chương trình 2.10

```
/* prog2_10.c                                          */
/* Chương trình để xác định tần số cộng hưởng của      */
/* mạch LC song song                                   */
#include <stdio.h>
#include <math.h>       /* required for sqrt(0) */

#define PI 3.14159

int main(void)
{
float L=1.0e-3,C=1.0e-6,r=1.0,f_res;
 /* L is 1mH, C is 1uF, điện trở của cuộn cảm là 1 ôm*/

   puts("Chương trình tính toán tần số cộng hưởng");
   puts("của một mạch LC mắc song song");


   f_res=1/(2*PI)*sqrc(1/(L*C)-(r*r)/(4*L*L));
   printf("r=%.3e ohm,C=%,3e F,L=%,3e H\n",r,C,L);
   printf("tần số cộng hưởng là %.3f Hz\n",f_res):
   return (0);
 }
```

Kết quả chạy thử 2.7 nhận được với r = 1 Ω, C = 1 μF và L = 1 mH.

Chạy thử 2.7

```
Chương trình để tính tần số cộng hưởng của một
mạch LC mắc song song
r=1.000e+00 ohm, C=1.000 e-06 F, L=1.000e-03 H
Tần số cộng hưởng là 5.032;328 Hz
```

2.2 CÁC LỆNH NHẬP VÀO CHUẨN: (scanf (), gets () và getchar ())

Thông thường bàn phím là đầu vào chuẩn để nhập vào chương trình. Cũng như các hàm đầu ra, các hàm đầu vào không phải là một

phần của ngôn ngữ chuẩn và được đặt trong một thư viện chuẩn của C. Các định nghĩa (hoặc nguyên mẫu) của các hàm này nằm trong tệp đầu mục *stdio.h*. Bằng cách chứa các tệp đầu mục này việc kiểm tra lỗi đã được kích hoạt trong quá trình dịch. Thêm vào đó, chương trình dịch kiểm tra kiểu dữ liệu các tham số đưa vào trong các hàm. Đây là cách để có lỗi thời gian chạy (run-time) nhỏ nhất.

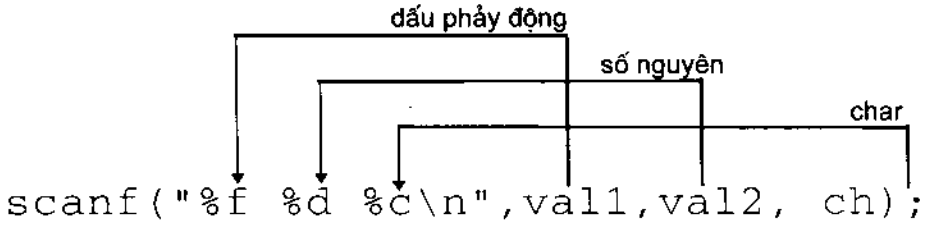
Có ba hàm đầu vào chính, đó là:

`scanf` (*"khuôn dạng " , & argl, & arg2..&argn)*	Đọc giá trị của hàm từ bàn phím với định dạng xác định bởi *format* và đọc chúng cho các đối số.
`Gets` (*string*)	Đọc xâu văn bản từ bàn phím vào *string* (cho đến một dòng mới)
ch= `getchar ()`	Đọc ký tự đơn từ bàn phím vào *ch*

Nếu như một biến ở dạng số hoặc ký tự được dùng với hàm `scanf()` thì một ký hiệu và `(&)` được đặt trước mỗi tham số trong danh sách đối số (điều này sẽ được thảo luận trong chương 6 và 7). Tiền tố này làm cho địa chỉ bộ nhớ của biến phải được sử dụng như một tham số chứ không phải là một giá trị. Việc này cho phép `scanf()` thay đổi giá trị của biến (điều này cũng sẽ giải thích kỹ trong chương 6). Bây giờ, ta sẽ giả thiết rằng một ký hiệu đặt trước tất cả các kiểu dữ liệu số đơn giản và ký tự khi sử dụng `scanf()`. Khuôn mẫu chung của hàm `scanf()` là:

```
scanf(format,&arg1,&arg2...)
```

Tham số đầu tiên *format* là một xâu xác định khuôn mẫu của tất cả các giá trị được nhập vào. Thí dụ, `"%f %d"` chỉ rõ rằng *arg1* đã được nhập vào như một dấu phẩy động và *arg2* như một số nguyên. Xâu này chỉ chứa các ký tự điều khiển chuyển đổi như `%d %f %c %s` v.v... được ngăn cách bởi các dấu trống. Hình 2.8 mô tả một thí dụ về hàm `scanf` khi đọc một dấu phẩy động, một số nguyên và một ký tự vào trong các biến `val1, val2,` và `ch`.



Hình 2.8: **Một thí dụ của lệnh** `scanf()`.

Chương trình 2.11 là thí dụ cho thấy các biến được nhập vào như thế nào khi sử dụng hàm `scanf()`.

Chương trình 2.11

```
/* prog2_11.c                                          */

#include <stdio>
int     main (voild)
{
float val1, val2, vald3, val4;
int   t,j,k;

 printf("\nType in two float values followed by <return> >>")
 scanf("%f, %f",&val1,&val2);
 printf("Type in two interger values followed by <return> >>")
 scanf("%d, %d",&i,&j);
 printf("Type in an interger and two floats followed by <return> >>")
 scanf("%d %f %f',&k,&val3,&val4);

 prinf("Floats enterred were %6.2f %6.2f %6.2f %6.2f\n",
                    val1,val2,val3,val4);
 printf("Integers intered were %6d %6d %6d\..,i,j,k),
 return(0);
}
```

Kết quả chạy thử 2.8 giới thiệu một thí dụ làm mẫu.

Chạy thử 2.8

```
Type in two float values followed by <Return> >> 1.23 23.8
Type in two interger values followed by <Return> >> 10 122
Type in an interger and two floats followed by <Return>
   >> 12 12.32 0.234
Floats entered were  1.23 23.80 12.32 023
Intergers entered were   10   122   12
```

Hàm `Gets(str)` đọc một số ký tự vào trong một biến (trong trường hợp này là `str`); những ký tự này được đọc cho đến chừng nào phím ENTER được nhấn. Hàm `Getchar()` đọc một ký tự đơn từ đầu vào. Ký tự này được quay trả lại qua đầu mục hàm và không qua danh sách tham số.

Xâu là một mảng ký tự được thiết lập khi sử dụng khai báo `charstrname` [*SIZE*], ở đó *SIZE* là số cực đại của các ký tự trong mảng và `strname` là tên xâu (xâu và mảng sẽ bàn luận chi tiết hơn trong chương sau). Chương trình 2.12 là thí dụ về các xâu nhập vào khi sử dụng hàm `gets()` và hàm `scanf()`.

Chương trình 2.12

```
/* prog2_12.c                                    */
#include <stdio.h>

int  main(void)
{
char  str1(100),str2(100),str3(100),text(100);
                /* storage of text strings */
   printf ("Enter a line of text >>");
   gets (text);

  printf("Entered line is %s\n", text);

  printf("Entered a line of text >>);
  scanf ("&s %s %s", str1, str2, str3);
```

```
    printf("Entered line if %s %s %s\n", str1, str2, str3);
    return (0)
}
```

Kết quả chạy thử 2.9 là một thí dụ tiêu biểu về việc chạy chương trình này. Hàm `scanf()` đọc mỗi từ được phân cách bởi dấu trống. Các từ và khoảng trống có thể được dịch như là một xâu và các từ được đưa vào trong từng tham số của xâu. Trong trường hợp này chỉ có ba từ sẽ được đọc từ bàn phím, nhưng ngược lại hàm `gets()` có thể đọc được tất cả dòng của văn bản cho đến khi phím ENTER được nhấn. Bởi vậy hàm `scanf()` không thể đọc các khoảng trống trong một xâu văn bản.

Chạy thử 2.9

```
Enter a line of text >>Some text input
Enter line is Some text input
Enter a line of text >>Some text input
Entered line is Some  text input
```

2.2.1 MỘT SỐ THÍ DỤ

Dung kháng

Chương trình 2.13 là kết quả hoàn thiện từ chương trình 2.7. Người dùng nhập các giá trị của f và C bằng cách sử dụng `scanf()`. Nó cho phép nhập vào các giá trị linh kiện được qua bàn phím.

Chương trình 2.13

```
/* prog2_13.c                                    */
/* Chương trình để tính dung kháng               */
#include <stdio.h>
#define PI 3.14159

int     main(void)
{
fload  freq,cap,x_c;
```

```
    puts("Enter frequency and capacitance");
    scanf("%f %f",&freq,&cap);
    x_c=1.0/(2.0*freq*cap);
    printf("Capacitive Reactance is %6.3f ohms\n", x_c);
    return(0);
}
```

Kết quả chạy thử 2.10 là một thí dụ làm mẫu.

Chạy thử 2.10

```
Enter frequence and capacitance
10 e3    1e-6
Capacitiance Reactance is 15.916 ohms
```

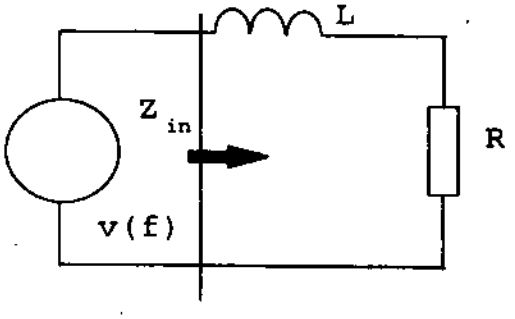
Trở kháng của mạch RL nối tiếp

Độ lớn*) của trở kháng trong mạch RL nối tiếp (môđun | Z |) xác định bằng phương trình:

$$|Z| = \sqrt{R^2 + X_L^2} \quad [\Omega]$$

và góc của trở kháng (đối số (Z)) xác định bằng:

$$\langle Z \rangle = \tan^{-1} \frac{X_L}{R}$$



Hình 2.9: **Mạch RL nối tiếp.**

*) Các thuật ngữ môđun và argument đối với trở kháng trong cuốn sách này tác giả dùng: *magnitude* (độ lớn) và *angle* (góc) khi dịch chúng tôi vẫn giữ nguyên để bảo đảm tính khách quan (ND).

Hình 2.9 mô tả sơ đồ mạch RL nối tiếp.

Chương trình 2.14 xác định độ lớn và góc của trở kháng khi sử dụng các giá trị của điện trở (R), điện cảm (L) và tần số (freq) được nhập vào. Nghịch đảo của hàm tang ($\tan^{-1}$) được định nghĩa trong *math.h* và có tên atan(). Hàm này trả trở lại hàm ngược của tang tính theo radian để chuyển chúng sang độ phải có hệ số π /180.

Chương trình 2.14

```
/* prog2_14.c                                          */
/* Chương trình để xác định trở kháng (của)           */
 mạch RL nối tiếp                                      */

#include <stdio.h>
#include  <math.h> /*yêu cầu cho sqrt()và atan()*/

#define PI 3.14159

int maint(void)
{
float  R,L,freq,Xl,Zin_mag,Zin_angle;

  printf("Enter R, L and frequence >>");
  scanf("%f %f %f",&R,&L,&freq);

  Xl= 2 * PI * freq * L;
  Zin_mag= sqrt (R*R+Xl*Xl);
  Zin_angle= atan(Xl/R)*180.0/PI;
   /* atan is arc tan and returs radians */
   /*180/PI converts to degrees                      */

  printf("Zin mag %6.2f ohms, angle %6.2f degrees\n"
                      Zin_mag,Zin_angle);
  return (0);
}
```

Kết quả chạy thử 2.11 là một thí dụ cho đầu ra khi nhập các giá trị R = 100 Ω, L = 100 mH và f = 1 kHz. Trở kháng có độ lớn 118,10 Ω và góc 32,14 °

Chạy thử 2.11

```
Enter R, L and frequence >>100 10e-3 1000
Zin mag 118.10 ohm,angle 32.14 degrees
```

Các điện trở mắc song song

Chương trình 2.15 là kết quả cải tiến từ chương trình 2.8, trong đó các giá trị nhập vào qua bàn phím thường được sử dụng ít hơn, so với các giá trị cố định. Ba giá trị điện trở được nhập vào bằng cách sử dụng `scanf()` với tham số khuôn mẫu là `"%f %f %f"`.

Chương trình 2.15

```
/* prog2_15.c                                                    */
/* Chương trình để xác định điện trở tương đương                 */
#include <stdio.h>

int main(void)
{
 float R1,R2,R3,R_equ;

   puts(" Chương trình để xác định điện trở tương đương");
   puts("của ba điện trở nối song song");

   scanf("%f %f %f",&R1,&R2,&R3);

   R_equ=1.0/(1.0/R1+1/R2+1/R3);
   printf("R1=%8.3f, R2=%8.3f và R3=%8.3f ohms\n", R1,R2,R3);

   printf(" Điện trở tương đương là %8.3f ohms\n",R_equ);
   return (0);
 }
```

Kết quả chạy thử 2.12 nhận được với các giá trị 250; 500 và 1000Ω

Chạy thử 2.12

```
Chương trình để xác định điện trở tương đương
của ba điện trở mắc song song
Nhập ba giá trị của điện trở >>
1000 500 250 11
RI=1000.000, R2. 500.000, R3= 250.000 ohms
Điện trở tương đương là 142.857 ohms
```

Tần số cộng hưởng của một mạch LC nối tiếp

Chương trình 2.16 được cải tiến từ chương trình 2.9.

Chương trình 2.16

```
/* Chương trình 2_16.c                                    */
/* Chương trình để xác định tần số cộng hưởng     */
/* mạch LC nối tiếp                                       */
#include <stdio.h>
#include <math.h>          /* required for sqrt () */

#define PI 3.14159

int       main(voild)
{
float    L,C,f_res;

     puts("Chương trình xác định tần số cộng hưởng của");
     puts("mạch LC nối tiếp");

     puts("Nhập vào giá trị điện dung và điện cảm");
     scanf("%f %f",&C,&L");

     f_res=1.0/(2.0*PI*sqrt(L*C));

     printf("C=%.3e F, L=%.3e\n",C,L);
     print("Tần số cộng hưởng là %.3f Hz\n",f_res);
```

```
    return(0)
}
```

Kết quả chạy thử 2.13 chạy với các giá trị đầu vào C = 1 μF và L=1mH.

Chạy thử 2.13

```
Chương trình cho tần số cộng hưởng của
mạch LC
Nhập vào giá trị điện dung và điện cảm
1e-6  1e-3
C=1.000e-06 F, L=1.000e-03
Tần số cộng hưởng là 5032.958 Hz
```

Tần số cộng hưởng của mạch LC mắc song song

Chương trình 2.17 xác định tần số cộng hưởng của một mạch LC song song tương ứng với các giá trị của L, C và r được nhập vào.

Chương trình 2.17

```
/* prog2_17.c                                          */
/* Chương trình xác định tần số cộng hưởng mạch        */
/* LC song song                                        */

#include <stdio.h>
#include <math.h>

#define PI  3.14159

int  main(void)
{
float L,C,r,f_res;
puts("Chương trình xác định tần số cộng hưởng của");
puts("mạch LC song song");

puts("nhập vào C, L, và r>>>");

scanf("%f %f %f", &C,&L,&r);
```

```
    f_res=1.0/(2.0*PI)*sqrt(1/L*C)-(r*r)/(4*L*L));

    printf("r=%3.e ohm, C=%.3e F, L=%.3e\n",r,C,L);
    printf("tần số cộng hưởng là %.3f Hz\n",f_res);
    return(0)
}
```

Kết quả chạy thử 2.14 nhận được với các giá trị dùng để kiểm tra C = 1 μF, L = 1 mH và điện trở trong bằng 4 Ω

Chạy thử 2.14

```
Chương trình để xác định tần số cộng hưởng của mạch LC
song song
ENTER C, L và r>>>
1e-6 F, 1e-3  4
r=4.00e+00 ohms, C=1.00e-06F, L=1.00e-03 H
Tần số cộng hưởng là 4992.531 Hz
```

Đáng chú ý là, đơn vị của điện dung được cho bằng μF (hoặc đôi khi bằng nF) và của điện cảm là mH; do đó sự cải tiến của chương trình là để nhắc người dùng nhập vào các giá trị bằng μF và mH. Chương trình 2.18 sử dụng chỉ dẫn hướng #define để giới thiệu các macro MILLI và MICRO.

Chương trình 2.18

```
/*prog2_18.c                                          */
/*Chương trình để xác định tần số cộng hưởng        */
/* của mạch LC song song                              */

#include <stdio.h>
#include <math.h>
/* Một vài định nghĩa để giúp đỡ chương trình tài liệu */

#define MIlll 1e-3
#define MICRO 1e-6
#define KILO  1e3
```

```
#define PI      3.14159

int  main(void)
{
float  L,C,r,f_res;

   puts("Chương trình xác định tần số cộng hưởng")
   puts("của mạch LC song song")
   puts("Nhập vào C"(μF), L(mH) và r(ohms)")

   scanf("%f %f %f",&C,&L,&r);

   C=C*MICRO;
   L=L*MILLI;

   f_res=1.0/(2.0*PI)*sqrt(1/(L*C)-(r*r)/(4*L*L));

   printf("r=%.2f ohm, C=%.2f μF, L=%.2fmH\n", r,C/MICRO,L/MILLI);
   printf(Tần số cộng hưởng là %.3f kHz\n", f_res/KILO);
   return(0);
  }
```

Kết quả chạy thử 2.15 là một thí dụ làm mẫu. Chú ý rằng các giá trị linh kiện nhập vào đơn giản hơn trong chạy thử 2.14. Giá trị tần số được tính bằng kHz.

Chạy thử 2.15

```
Chương trình để xác định tần số cộng hưởng
của một mạch LC song song
Nhập C (bằng uF), L (bằng mH) và r (bằng ohms)
1 10 4
r=4.00 ohms, C = 1.00 uF, L=10.00 mH
Tần số cộng hưởng là 1.591 (kHz)
```

Các phép toán bit

Chương trình 2.19 liên quan đến kỹ thuật điện tử số và minh họa khả năng của ngôn ngữ C với các toán tử bit mức thấp. Nó sử dụng các toán tử xử lý đến bit AND (&),OR (|), EX-OR (^) và NOT (~) để tạo ra các hàm Boole AND, OR, EX-OR, NAND và NOR. Các hàm NAND và NOR được tạo ra bằng cách lấy nghịch đảo các phép toán AND và OR.

Chương trình 2.19

```
/* prog2_19.c                                          */
/* Chương trình có bitwise AND,OR,NAND, NOR và EX-OR   */
/* hai giá trị hệ thập lục phân                        */

#include <stdio.h>

int main(void)
{
int value1, value2;

        /* & - bitwise toán tử AND                     */
        /* | - bitwise toán tử OR                      */
        /* ^ - bitwise toán tử EX-OR                   */
        /* ~ - bitwise toán tử NOT                     */

  printf("nhập vào hai giá trị hex>>>")
  scanf("%x %x",&value1,&value2);

  printf("Các giá trị ANDed là %x\n", value1 & value2)
  printf("Các giá trị ORed là %x\n", value1 | value2)
  printf("Các giá trị Ex-ORed là %x\n", value1 ^ value2)
  printf("Các giá trị NANDed là %x\n",( value1 & value2)
  printf("Các  gía trị NORed là %x\n",(value1 |,value2))
  return(0);
}
```

Kết quả chạy thử 2.16 giới thiệu một thí dụ làm mẫu.

Chạy thử 2.16

```
Nhập vào hai giá trị hex >>> E215 C431
Các giá trị ANDed là c011
Các giá trị ORed là e635
Các giá trị Ex-ORed là 2624
Các giá trị NANDed là 3fee
Các giá trị NORed là 19ca
```

Các mẫu bit được sử dụng trong chạy thử là 1110 0010 0001 0101 (E215h) và 1100 0100 0011 0001 (C431h). Để kiểm tra chương trình, tương đương thập lục phân của các giá trị này được xử lý bởi các toán tử Boole và các kết quả kiểm tra so sánh với các kết quả chạy thử.

Phép toán AND cho ra như sau:

Thập lục phân	Nhị phân
E215	1110 0010 0001 0101
C431	1100 0100 0011 0001
C011	1110 0000 0001 0001

Phép toán OR cho ra như sau:

Thập lục phân	Nhị phân
E215	1110 0010 0001 0101
C431	1100 0100 0011 0001
E635	1110 0110 0011 0101

Phép toán EX-OR cho ra như sau:

Thập lục phân	Nhị phân
E215	1110 0010 0001 0101
C431	1100 0100 0011 0001
2624	0010 0110 0010 0100

Nghịch đảo của AND (NAND) là 0011 1111 1110 1110 (3FEEh), Nghịch đảo của OR (NOR) là 0001 1001 1100 1010 (19CAh).

Các kết quả này đồng nhất với kết quả trong chạy thử 2.16. Như vậy quá trình kiểm tra đã thực hiện thành công.

Chuyển đổi giữa các hệ thập phân, thập lục phân và bát phân

Các chương trình 2.20, 2.21 và 2.22 chứng minh cho thấy các giá trị nhập vào và xuất ra như thế nào khi sử dụng các cơ số khác nhau. Các số nguyên có thể được nhập/ xuất như một số thập phân (%d), thập lục phân (%%) hoặc bát phân (%o). Cách sử dụng hệ thập lục phân và bát phân tỏ ra là quan trọng trong các phép toán biến đổi bit. Chương trình 2.20 hiển thị lên màn hình một số thập phân được nhập vào và các giá trị tương đương trong cách biểu diễn của chúng với số thập lục phân và bát phân đã trình bày.

Chương trình 2.20

```
/* prog2_20.c                                              */
/* Chương trình hiển thị số nguyên thập phân thành thập */
/* lục phân và bát phân                                    */

#include <stdio.h>

int main(void)
{
int value;

  puts (" nhập giá trị số nguyên(-32768 đến 32767)
  scanf("%d",&value);

  printf("Decimal = %d, Octal = %0, Hex - %x\n"
                        value, value, value);
  return(0)
}
```

Chạy thử 2.17 là một thí dụ tiêu biểu

Chạy thử 2.17

```
Nhập vào giá trị thập phân(-32768 đến 32767)>>
36
Thập phân= 36, bát phân = 44, thập lục phân =
```

Chương trình 2.21 sử dụng số thập lục phân ở đầu vào và hiển thị số đó theo hệ thập lục phân và bát phân.

Chương trình 2.21

```
/* prog2_21.c                                          */
/* Chương trình hiển thị giá trị thập lục phân */
/* dưới dạng giá trị thập phân và bát phân     */

#include <stdio.h>

int main(void)
{
int value;

    puts ("Nhập giá trị thập lục phân")
    scanf("%x",&value);

    printf("decimal= %d, octal=%0,
       hexadecimal=%x\n", value, value, value;
    return(0);
   }
```

Kết quả chạy thử 2.18 là một thí dụ làm mẫu với giá trị thập lục phân được nhập vào bằng FAh.

Chạy thử 2.18

```
Nhập vào giá trị thập lục phân
fa
Thập phân = 250, Bát phân = 372, Thập lục phân = fa
```

Chương trình 2.7 hiển thị giá trị thập phân mã ASCII và các ký tự ASCII tương đương.

Chương trình 2.22

```
/* prog2_22.c                                    */
/* Program to ASCII code to character            */

#include <stdio.h>

int  main(void)
{
int  value;
    puts("Enter ASCII code (greater than 33)>");
    scanf("%d",&value);

   printf("code= %d, character =%c\n",value,value,value);
   return(0);
}
```

Kết quả chạy thử 2.19 là một thí dụ làm mẫu giá trị thập phân ASCII được nhập vào bằng 45; giá trị này in ký tự ASCII ' - '.

Chạy thử 2.19

```
Enter ASCII code (greater than 33)>
45
Code = 45, ký tự = -
```

CHÚ Ý: Có thể dẫn đến rắc rối nếu hàm `getchar()` được sử dụng sau `scanf()`. Điều này xảy ra là do những ký tự dòng mới (carriage return) được lưu giữ trong bộ đệm bàn phím. Vấn đề này có thể được giải quyết bằng cách sử dụng lệnh `fflush(stdin)` trước mỗi lệnh nhập vào bằng bàn phím. Các thí dụ sử dụng `getchar()` sẽ được giới thiệu trong chương tiếp theo.

2.3 THỰC HÀNH

Q2.1 (a) Hãy viết một chương trình hiển thị văn bản sau đây bằng cách sử dụng hai lệnh `printf()`. Tham khảo thêm chương trình 2.3.

Chạy thử 2.20

```
Giá trị của điện trở trong mạch
nhỏ hơn trở kháng của tụ điện
```

(b) Hãy sửa đổi chương trình (a) sao cho chỉ dùng một lệnh `printf()`.

Q2.2 Hãy tìm và sửa ba loại lỗi trong các chương trình 2.23, 2.24 và 2.25.

(a)

Chương trình 2.23

```
/* prog2_23.c                                          */
/* Chương trình xác định điện trở tương đương          */
/* của hai điện trở mắc song song                      */
#include <stdio.h>
int main(void)
{
float R1,R2,Requ;

  puts ("Enter R1 và R2");
  scanf("%d %d", &R1, &R2);

  Requ=1.0(1.0/R1+1/R2);

  printf("Parallel resistance is %f",Requ);
    return (0),
  }
```

(b)

Chương trình 2.24

```
/* prog2_24.c                                          */
/* Chương trình để xác định dòng điện đi qua           */
```

```
/* một điện trở khi sử dụng định luật Ohm      */

#include <stdio.h>

int main(void)
{
float current, voltage, resistance;


printf("Enter voltage and resistance">>");
scanf(" %f %f", voltage, resistance;
current= voltage/ resistance;
printf("Current is" current," amps\n");
return (0);
}
```

(c)

Chương trình 2.25

```
/* prog2_25.c                                          */
/* Program to bitweise OR two hexadecimal values  */

#include <stdio.h>

int main (void)
int val1.val2.val3;
{
  printf("Enter two hex values");
  scanf("%d %d",&val1,&val2);

  val3= val1 & val2;
  printf("%x OR %x = %x\n", val1, val2, val3);
  return(0);
}
```

Kết quả chạy thử 2.21 giới thiệu đầu ra đúng từ chương trình 2.25.

Chạy thử 2.21

```
Nhập vào giá trị thập lục phân>> 12 6a
```

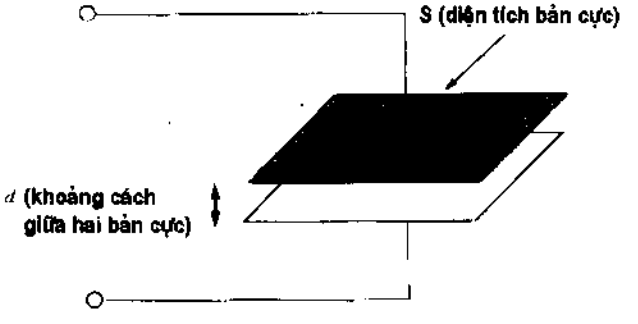
```
12 OR 6a = 7a
```

Q2.3 Điền vào bảng 2.5 bằng cách sử dụng một chương trình chuyển đổi. Tham khảo thêm các chương trình 2.20 và 2.21.

Bảng 2.5: **Các giá trị tương đương thập phân, thập lục phân và bát phân.**

Thập phân	Thập lục phân	Bát phân
145		
-54		
2222		
	0x1fff	
	0x81a0	
	0xbbb	
		393
		076

Q2.4 (a) Tụ điện tấm phẳng song song trong hình 2.10 gồm hai bản cực song song cách ly bởi chất điện môi là không khí. Hãy viết một chương trình xác định điện dung của cơ cấu này. Chương trình sẽ nhắc nhập vào diện tích của các bản cực và khoảng cách giữa chúng. Kết quả chạy thử 2.22 là một thí dụ làm mẫu.



Hình 2.10: **Tụ điện tấm phẳng song song**

$$C = \frac{\varepsilon_o . A}{d} \quad [F]$$

ở đây hằng số điện môi của không gian tự do $\varepsilon_0 = 8{,}854.10^{-12}$ F.m^{-1}

Chạy thử 2.22

```
Nhập vào giá trị khoảng cách và diện tích hai bản cực
>>> 10e-3 100e-4
Điện dung bằng 8.8e-12 F
```

(b) Hãy sửa đổi chương trình (a) để người dùng có thể nhập vào một giá trị của hằng số điện môi (ε_r). Điện dung của một tụ điện với các bản cực phẳng cách ly bằng một chất điện môi có hằng số điện môi ε_r được cho bởi phương trình sau:

$$C = \frac{\varepsilon_o . \varepsilon_r . A}{d} \quad [F]$$

Bảng 2.6 giới thiệu hằng số điện môi của một số vật liệu thường gặp.

Bảng 2.6: **Hằng số điện môi của một số vật liệu cách điện.**

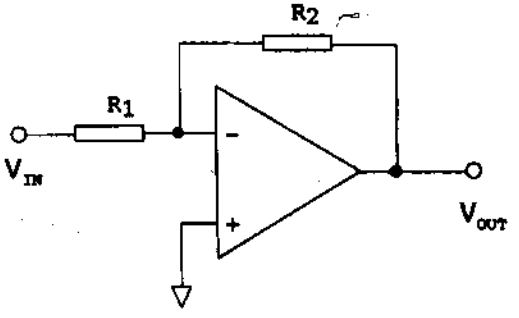
Vật liệu	Hằng số điện môi
Thuỷ tinh	7 - 8
Mica	5,5 - 8
Giấy (khô)	4,5 - 4,7
Polyester	2,8 - 3,2
Polystyrene	2,5

(c) Hãy sửa đổi chương trình (b) sao cho diện tích bản cực được nhập vào tính bằng mm^2 và khoảng cách bằng mm.

(d) Hãy sửa đổi chương trình (c) sao cho điện dung được hiển thị bằng μF.

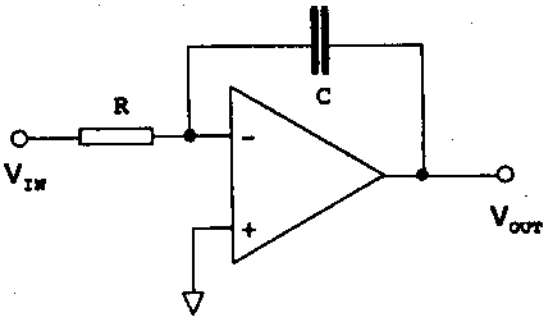
Q2.5 Hình 2.11 mô tả một bộ khuếch đại đảo sử dụng khuếch đại thuật toán. Hệ số khuếch đại tỷ lệ với R1 và R2. Hãy viết một chương

trình xác định hệ số khuếch đại khi các giá trị điện trở được nhập vào.



Hình 2.11: **Bộ khuếch đại đảo.**

Q2.6 Hình 2.12 mô tả một bộ lọc tích cực dùng khuếch đại thuật toán. Mạch này có hệ số khuếch đại lớn ở tần số thấp và hệ số khuếch đại nhỏ ở tần số cao. Như vậy mạch này hoạt động giống như một



bộ lọc dải thông tần số thấp.

Hình 2.12: **Bộ lọc tích cực dùng RC.**

Hệ số khuếch đại trong mạch này được tính theo công thức:

$$|\text{Hệ số khuếch đại}| = \frac{1}{2\pi . fRC}$$

(a) Hãy viết chương trình trong đó người dùng nhập vào giá trị tần số, điện trở (chẳng hạn 1MΩ) và điện dung (chẳng hạn 1μF), và chương trình xác định hệ số khuếch đại. Tham khảo thêm chương trình 2.14.

(b) Hãy sửa đổi chương trình (a) để đầu ra được hiển thị bằng đêxiben (dB). Hàm `log10()` xác định lôgarit cơ số 10. Hàm này có đặt trong tệp đầu mục *math.h*. Hệ số khuếch đại có thể được tính bằng cách sử dụng côr ; thức sau:

Hệ số khuếch đại (dB) = $20.\log_{10}$(HSKĐ điện áp) [dB]

Q2.7 Hãy sửa đổi chương trình 2.16 sao cho chương trình hiển thị điện kháng cuộn cảm tại thời điểm cộng hưởng ($X_L=2\pi f_{ch}L$). Chú ý rằng điện kháng của tụ điện sẽ bằng giá trị này tại thời điểm cộng hưởng.

Q2.8 Hãy sửa đổi chương trình 2.17 sao cho nó hiển thị điện kháng của tụ điện tại thời điểm cộng hưởng ($X = 1/(2\pi.f_{ch}C)$). Đồng thời, xác định độ lớn trở kháng của cuộn cảm ($Z = \sqrt{r^2 + X_L^2}$).

Q2.9 Chương trình 2.26 có một lỗi. Chương trình xác định năng lượng bị tiêu tán trên điện trở do điện áp xoay chiều đặt vào với một điện áp đỉnh được nhập vào. Chạy thử 2.23 là một thí dụ tiêu biểu của chương trình 2.26 và chạy thử 2.24 chạy từ một chương trình làm việc đúng. Chú ý rằng ở hệ thống hoặc/ và chương trình dịch khác nhau có thể chạy khác với chạy thử 2.24 (chương trình không thể đoán trước).

Chương trình 2.26

```
/* prog2_26.c                                          */
/* Chương trình xác định công suất bị tiêu tán trên */
/* điện trở khi có điện áp xoay chiều đặt vào    */

#include <stdio.h>

int main(void)
{
float  Vmax, Vrms,R,power;


  printf("Enter peak voltage and resistance > >");
  scanf("%f %f", Vmax,&R);
  Vrms=Vmax/sqrt(2);
```

```
power=(Vrms*Vrms)/R;
printf("power is %6.2f\n watts",power);
   return (0);
}
```

Chạy thử 2.23

```
Enter peak voltage and resistance >> 10 2
Power is 50,00 Watt
```

Chạy thử 2.24

```
Enter peak voltage and resistance >> 10 2
Power is 25,00 Watt
```

Q2.10 Xác định đầu ra từ chương trình 2.27.

Chương trình 2.27

```
/* prog2_27.c                                    */
#include <stdio.h>
int  main(void)
{
int       y;
/*let this be a lesson to use parenthese */
  y = -3+6 * 3-2;printf("%d \n",y);
  y = 3+4 % 5-6; printf("%d \n",y);
  y = -4 *3 % -6 / 5;       printf("%d \n",y);
  y = (7 + 6) % 5 / 2;     printf("%d \n",Y);
  return(0);
}
```

Q2.11 Xác định đầu ra từ chương trình 2.28.

Chương trình 2.28

```
/* prog2_28.c                                    */
#include <stdio.h>
int main(void)
```

```
{
int x=4, y, z;

x *= 3 + 2;          printf("%d\n",x);
x *= y = z = 4;      printf("%d \n", x);
X++;                 printf("x = %d \n".x);
return(0);
 }
```

Q2.12 Hãy viết chương trình trong đó người dùng nhập vào hai giá trị bát phân. Chương trình sẽ hiển thị kết quả xử lý tới bit AND, OR, NAND, EX-OR và NOR các giá trị bát phân. Tham khảo chương trình 2.19.

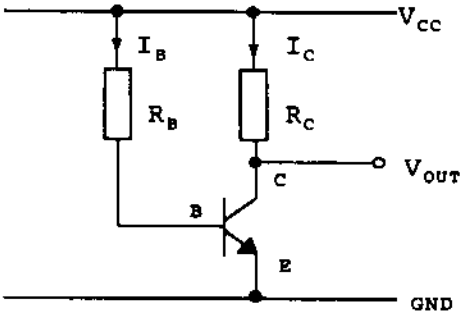
Q2.13 Hãy viết chương trình xác định độ lớn và góc của trở kháng của mạch RC nối tiếp. Tham khảo chương trình 2.14.

Q2.14 Hãy viết chương trình xác định biên độ trở kháng của mạch LC nối tiếp theo giá trị tần số (f), điện cảm (L) và điện dung (C) được nhập vào. Công thức sau có thể dùng để tính độ lớn:

$$|Z_{in}| = \left[2\pi.fL - \frac{1}{2\pi.fC}\right] \quad [\Omega]$$

Hàm `abs()`, có trong *math.h*, xác định giá trị biên độ (hay còn gọi là giá trị tuyệt đối). Tham khảo thêm chương trình 2.14.

Q2.15 Hãy sửa đổi Q2.14 để các giá trị nhập vào được tính theo mH và μF. Tham khảo thêm chương trình 2.18.



Hình 2.13: **Mạch tranzito lưỡng cực định thiên đơn giản.**

Q2.16 Hình 2.13 mô tả mạch tranzito lưỡng cực thiên áp chỉ đơn giản bằng một điện trở cực gốc.

Đáng chú ý là, để có tín hiệu xoay chiều ở lối ra biên độ cực đại, điện áp cực góp được đặt ở giá trị bằng một nửa điện áp nguồn, trong trường hợp này là $V_{CC}/2$. Như vậy, đối với một giá trị dòng cực góp đã được chỉ định (I_C), điện trở cực góp (R_C) có thể xác định được (xem phương trình 1). Dòng cực gốc được xác định bằng cách chia dòng góp cho hệ số khuếch đại dòng một chiều (xem phương trình 2). Nếu tranzito đang ở trạng thái dẫn ON thì lớp tiếp giáp gốc-phát sẽ có điện áp của điốt silic ở trạng thái dẫn (V_{BE}(ON)). Sử dụng giá trị gần đúng của điện áp này (-0.65 V) điện trở cực gốc có thể được xác định (phương trình 3). Các phương trình này là:

$$R_C = \frac{V_{CC}/2}{I_C} \qquad [\Omega] \qquad (1)$$

$$I = \frac{I_C}{h_{FE}} \qquad [A] \qquad (2)$$

$$R = \frac{V_{CC} - V_{BE}}{I_B} \qquad [\Omega] \qquad (3)$$

Hãy viết chương trình xác định R_B, cho phép nhập Ic và Vcc. Giả sử V_{BE}(ON) bằng 0,65 V và h_{FE} bằng 100. Giá trị được nạp của Vcc nằm giữa 5 và 30 V, và I_C giữa 0.1 và 10 mA. Kết quả chạy thử 2.25 giới thiệu một đầu ra tiêu biểu.

Chạy thử 2.25

```
Nhập vào Vcc (5 -> 15 V)>> 15
Nhập vào Ic (mA) >> 1
Điện trở cực góp là 15000 Ohm
Điện trở cực gốc là 1435000 Ohm
```

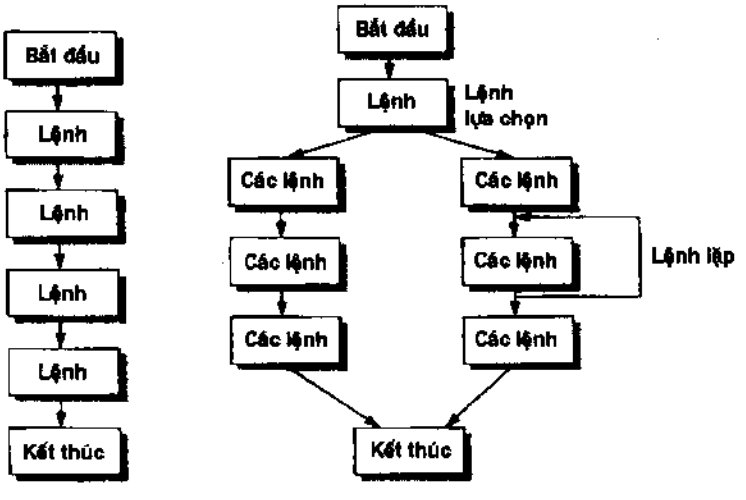
Q2.17 Hãy sửa đổi chương trình trong Q2.16 sao cho nó hiển thị điện trở cực góp ra kΩ và điện trở cực gốc ra MΩ. Chạy thử 2.26 giới thiệu một thí dụ làm mẫu.

Chạy thử 2.26

```
Nhập vào Vcc (5 -> 15 V)>> 15
Nhập vào Ic (mA) >> 1
Điện trở cực góp là 15 kOhm
Điện trở cực gốc là 1.43 MOhm
```

Chương 3
CÁC LỆNH LỰA CHỌN

Trong một chương trình tuần tự, một dãy các lệnh được thực hiện kế tiếp nhau. Đường dẫn chương trình không bao giờ thay đổi. Tuy nhiên, trong phần lớn các chương trình, việc đi đến quyết định còn phụ thuộc vào các giá trị được nhập vào hoặc các giá trị đã được xử lý. Quá trình này được thực hiện bằng các lệnh điều khiển. Mọi quyết định đều lặp đi lặp lại. Các lệnh quyết định, hoặc lựa chọn để cho phép luân phiên tác động là tuỳ thuộc vào kết quả của phép toán. Hai lệnh được dùng để đi đến quyết định là: `if...else` và `switch`. Quá trình lặp cho phép tạo thành một vòng kín từ tập hợp các lệnh. Có ba dạng lặp đi lặp lại: `while`, `do` và `for`. Hình 3.1 chỉ ra tiến trình (lưu đồ) của một chương trình tuần tự và một chương trình không tuần tự với sự lựa chọn và các lệnh lặp đi lặp.



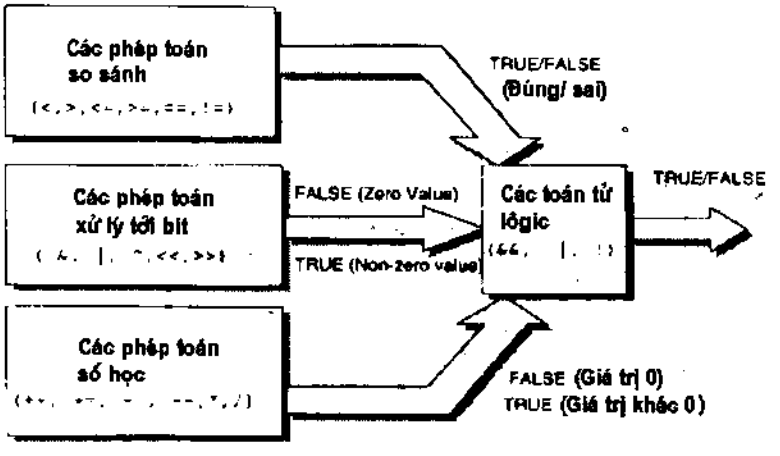
Hình 3.1: **Chương trình tuần tự và không tuần tự.**

Lệnh điều khiển tác động lên các biểu thức có điều kiện và chúng có thể tác động đến các lệnh đơn giản hoặc các lệnh phức tạp. Dấu ngoặc nhọn*) ({ }) xác định chỗ bắt đầu và kết thúc của khối.

3.1 LỆNH if ... else

Sự quyết định được tiến hành bằng lệnh `if` (*nếu*) để xác định xem về mặt lôgic thì một biểu thức điều kiện là TRUE (*đúng*) hay FALSE (*sai*). Nếu là TRUE, chương trình sẽ chấp hành (execute) một khối mã; nếu là FALSE thì sẽ chọn lựa khả năng chấp hành khác (nếu có). Từ khóa `else` (*không thì*) xác định khối FALSE. Dấu ngoặc nhọn được sử dụng để xác định chỗ bắt đầu và kết thúc của khối. Điều đáng chú ý là không có kiểu dữ liệu số Boole trong ngôn ngữ C.

Các toán tử so sánh (>; <; >=; <=; ==; !=) dẫn đến kết quả là TRUE hoặc FALSE từ phép tính tương ứng. Các lệnh lôgic (&&, ||, !) có thể được chia nhóm để có được chức năng mong muốn. Nếu như phép toán không phải là so sánh, như xử lý tới bit hoặc phép toán số học, thì một giá trị khác 0 là TRUE và bằng 0 là FALSE. Hình 3.2 cho thấy các toán tử so sánh, xử lý tới bit và số học nhóm lại cùng với các toán tử lôgic như thế nào.



Hình 3.2: **Cấu trúc tổng thể của lệnh if...**

*) Còn gọi là dấu ngoặc ôm, ở đây dùng thuật ngữ ngoặc nhọn để phân biệt với dấu ngoặc vuông ([ ]) (ND).

Sau đây là thí dụ về cú pháp của lệnh `if`. Nếu như khối lệnh chỉ có một lệnh, thì dấu ({ }) có thể được bỏ đi.

```
if (expression)
{
statement block
}
```

Sau đây là một mẫu các câu lệnh có bổ sung thêm `else`.

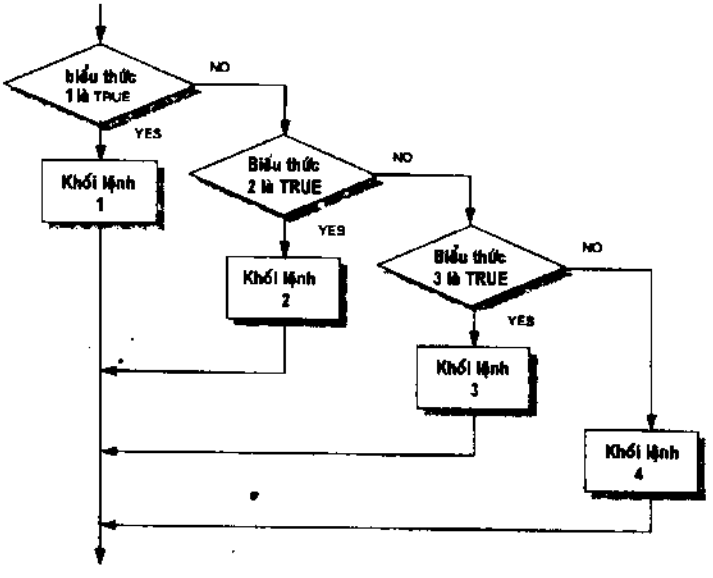
```
if (expression)
{
 statement block1
},
else
{
 statement block2
}
```

Có thể kết hợp các lệnh `if...else` để tạo các chức năng mong muốn. Trong thí dụ tiếp theo, lệnh *statement block1* được thực hiện nếu như biểu thức *expression1* là TRUE. Nếu là FALSE thì chương trình sẽ kiểm tra biểu thức tiếp theo. Nếu biểu thức này là TRUE thì chương trình thực hiện lệnh *statement block2*, nếu không nó sẽ kiểm tra biểu thức tiếp theo v. v... Nếu như tất cả các biểu thức đều là FALSE thì chương trình sẽ thực hiện khối lệnh `else` cuối cùng, trong trường hợp này là lệnh *statement block4*:

```
if (expression1)
{
statement block1
}
else if (expression2)
{
statement block2
}
else if (expression3)
{
```

```
    statement block3
}
    else
    {
    statement block4
}
```

Hình 3.3 minh hoạ lệnh này dưới dạng sơ đồ.



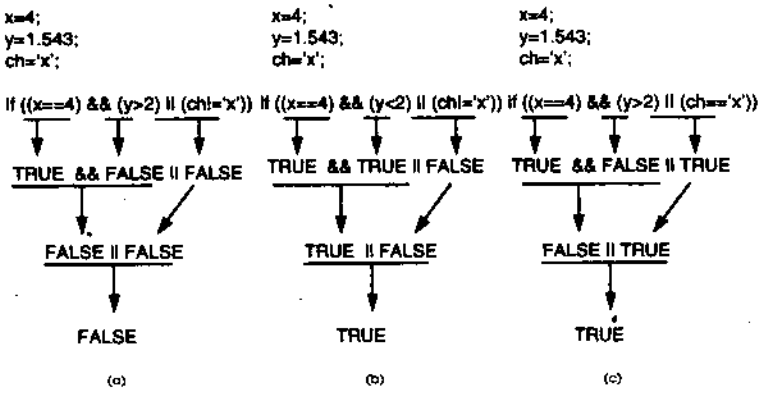
Hình 3.3: **Cấu trúc của lệnh if hỗn hợp.**

Hình 3.4 giới thiệu ba thí dụ về lệnh `if`. Trong mỗi thí dụ, các phép tính so sánh (==, !=, >) được giải trước vì chúng có mức độ ưu tiên cao hơn so với các phép tính lôgic (&&, | |). Sau phép tính so sánh chương trình giải phép tính lôgic VÀ (AND, &&) vì nó có mức độ ưu tiên cao hơn phép tính lôgic HOẶC (OR, | |). Sau các phép tính lôgic VÀ (AND), HOẶC (OR), thì từ biểu thức sẽ nhận được kết quả là TRUE hoặc FALSE. Có thể tham khảo các thông tin trước đây đã đề cập trong chương I.

Trên hình 3.4(a) phần thứ nhất của biểu thức (x==4) dẫn đến giá trị TRUE khi x bằng 4. Kết quả của phần thứ hai (y>2) là FALSE khi y không lớn hơn 2. Phép toán so sánh cuối cùng cụ thể là ch!= 'x'; có kết quả là FALSE khi giá trị của ch bằng 'x'. Sau khi các phép toán so sánh này đã được xác định, phép toán lôgic AND (&&) tác động lên kết quả của hai phép toán so sánh đầu tiên ((x==4) && (y>2)). Phép toán này dẫn đến kết quả là FALSE khi có ít nhất một trong các phép tính so sánh là FALSE. Kết quả phần cuối cùng của lệnh (ch! = 'x') cũng là FALSE, như vậy kết quả của toàn bộ biểu thức này là FALSE.

Trong hình 3.4(b) toán tử so sánh đối với y đã được thay đổi sao cho phép tính xác định xem liệu giá trị của y là nhỏ hơn 2 (y<2). Điều kiện này dẫn đến kết quả là TRUE, khi kết quả của phép tính lôgic AND là TRUE. Ngược lại điều kiện này, làm cho toàn bộ biểu thức là TRUE, khi phép toán lôgic OR là TRUE nếu như bất kỳ một trong các toán hạng là TRUE.

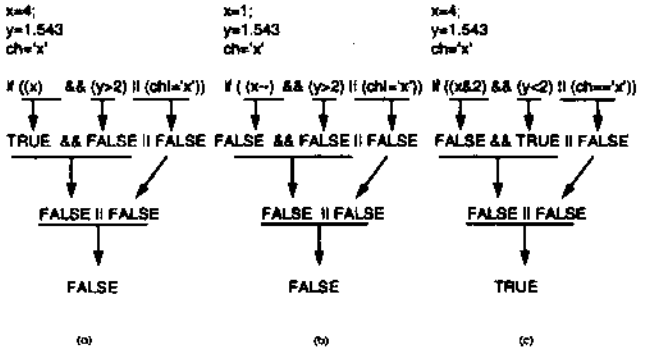
Trong hình 3.4(c) phép toán (ch=='x') dẫn đến kết quả là TRUE thì ngược lại sẽ làm cho toàn bộ biểu thức là TRUE.



Hình 3.4: **Các thí dụ của lệnh if.**

Hình 3.5 minh họa các toán tử số học và xử lý tới bit có thể được dùng với lệnh `if` như thế nào. Trong hình 3.5(a) toán tử lôgic AND coi như giá trị của x là TRUE khi giá trị của nó khác 0 (zêrô). Trong hình 3.5(b), kết quả của phép tính x sẽ bằng 0; như vậy toán tử lôgíc cũng coi như kết quả này là FALSE. Tương tự, trong hình 3.5(c) kết quả của

phép toán xử lý tới bit x & 2 (00000100 & 00000010) dẫn đến một giá trị 0; như vậy toán tử lôgic xử lý giá trị này như là FALSE.



Hình 3.5: **Các thí dụ của lệnh if.**

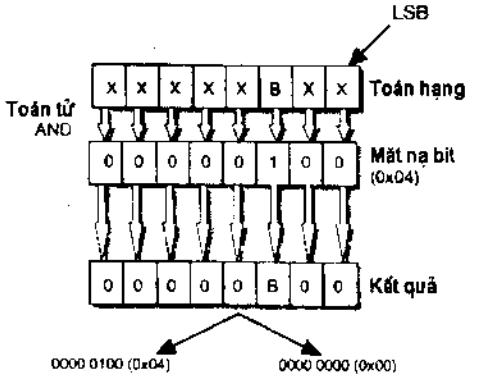
3.1.1 MỘT SỐ THÍ DỤ

Biến đổi thập phân sang nhị phân

Chương trình trong mục này biến đổi một giá trị số nguyên thập phân sang dạng nhị phân 8 bit không dấu. Một kỹ thuật được gọi là *che bit* được sử dụng để nhận biết các bit riêng lẻ bằng cách đặt các bit khác, khác với bit được quan tâm, bằng 0. Kỹ thuật này sử dụng toán tử xử lý tới bit (&), dẫn đến kết quả bằng 0 đối với một bit nếu như một trong những toán hạng bit bằng 0; nói khác đi, nếu như một trong các toán hạng bit bằng 1 nó sẽ dẫn đến giá trị của bit toán hạng khác. Hình 3.6 là một thí dụ về việc che bit nhỏ thứ ba (b_2). Trong trường hợp này, mặt nạ bit được sử dụng là 0x04; mặt nạ này sau đó được liên kết AND theo từng bit. Chỉ có hai kết quả khả dĩ từ phép tính này: là 0 (nếu b_2 bằng 0) hoặc 4 (nếu b_2 bằng 1). Dấu x có nghĩa là không cần quan tâm đến trạng thái, trong đó một bit có thể nhận giá trị nhị phân bất kỳ (nghĩa là 0 hoặc 1).

Chương trình 3.1 sử dụng lệnh `if` với các giá trị che bit 0x80 (1000 0000), 0x40 (0100 0000) v. v..., để xác định xem liệu có bit nào trong 8 bit thấp hơn của số nguyên thập phân được đặt. Lệnh `printf()` hiển thị từng bit trên một dòng khi ở đó không có ký tự dòng mới ('\ n') ở cuối mỗi xâu định dạng cho hàm `printf()`. Bit có giá trị cao nhất được hiển

thị trước tiên khi xuất hiện ở phía bên trái màn hình. Chú ý rằng đây không phải là phương pháp hiệu quả nhất để giải quyết vấn đề này; một vòng lặp sẽ làm cho chương trình ngắn gọn hơn nhiều (xem chương trình 4.2).



Hình 3.6: **Phép toán che bit.**

Chương trình 3.1

```
/* prog3_1.c                                          */
/* Chương trình để biến đổi một số thập phân         */
/* thành một số nhị phân không dấu 8 bit             */

int     main(void)
{
int   i; /* only the bottom eight bits are used. An    */
         /* unsigned char data type could also be used */

printf("Enter decimal value (0-255)>>>  *);
scanf("%d",&i);
printf ("Binary equivalent is ") ;
if  (i & 0x80)  printf("1"); else printf("0");
if  (i & 0x40)  printf("1");  else printf("0");
if  (i & 0x20)  printf("1");  else printf("0");
if  (i & 0x10)  printf("1");  else printf("0");
if  (i & 0x08)  printf("1");  else printf("0");
```

```
    if  (i & 0x04)  printf("1");  else printf("0");
    if  (i & 0x02)  printf("1");  else printf("0");
    if  (i & 0x01)  printf("1");  else printf("0");
    puts("");
    return(0);
}
```

Chạy thử 3.1 và 3.2 chỉ ra hai thí dụ làm mẫu với các giá trị nhập vào tương ứng bằng 33 và 255. Cả hai lần chạy thử đều tạo ra các giá trị nhị phân thích hợp.

Chạy thử 3.1

```
Nhập vào giá trị thập phân(0-255)>>> 33
Tương đương nhị phân là 00100001
```

Chạy thử 3.2

```
Nhập vào giá trị thập phân (0-255)>>> 255
Tương đương nhị phân là 11111111
```

Một vài điểm sửa đổi đã được bổ sung vào chương trình 3.2, chẳng hạn một cái bẫy để phát hiện các giá trị nhập vào không hợp lệ, cụ thể là nhỏ hơn 0 và lớn hơn 255. Chương trình sẽ thoát ra mà không in giá trị nhị phân nếu như một giá trị không hợp lệ được nhập vào. Một điểm bổ sung khác dùng chỉ dẫn hướng `#define` để thay thế những giá trị bit che bằng các thẻ bài, thí dụ bit che có giá trị cao nhất (0x80) được thay thế bằng ký hiệu BIT7. Kỹ thuật này cải thiện cách đọc chương trình.

Chương trình 3.2

```
/* prog3_2.c                                          */
/* Chương trình để hiển thị một số thập phân */
/* không dấu 8 bit thành định dạng nhị phân  */
/* A few bit mask defines        */
```

```
#define  BIT0 0x01
#define  BIT1 0x02
#define  BIT2 0x04
#define  BIT3 0x08
#define  BIT4 0x10
#define  SITS 0x20
#define  BIT6 0x40
#define  BIT7 0x80
int main(void)
{
int i;

    printf ("Enter decimal value (0-255)>>> ");
    scanf ("%d",&i);

    if ( (i<0) || (i>255))
    puts("Invalid entered value"):
    else
   {
   printf("Binary equivalent is ")
   if (i & BIT7) printf("1"); else printf("0");
   if (i & BIT6) printf("1"); else printf("0");
   if (i & BIT5) printf("1"); else printf("0");
   if (i & BIT4) printf("1"); else printf("0");
   if (i & BIT3) printf("1"); else printf("0");
   if (i & BIT2) printf("1"); else printf("0");
   if (i & BIT1) printf("1"); else printf("0");
   if (i & BIT0) printf("1\n"); else printf("0\n");
  }
  return(0);
}
```

Chạy thử 3.3 và 3.4 giới thiệu các lối ra làm mẫu từ chương trình này. Phép thử đầu tiên với một giá trị phù hợp với giới hạn (36) và phép thử thứ hai với giá trị nhập vào không hợp lệ (430).

Chạy thử 3.3

```
Nhập vào giá trị thập phân(0-255)>>> 36
Tương đương nhị phân là 00100100
```

Chạy thử 3.4

```
Nhập vào giá trị thập phân (0-255)>>> 430
Giá trị không thích hợp
```

Các phép toán xử lý tới bit (bitwise)

Chương trình 3.3 là kết quả bổ sung của chương trình 2.19. Một thực đơn tùy chọn đơn giản đã được bổ sung để người dùng có thể lựa chọn hàm Boole cần có, thí dụ OR, AND, EX-OR, NOR hoặc NAND.

Chương trình 3.3

```
/* prog3_3.c                                          */
/* Chương trình sử dụng một thực đơn đơn giản         */
/* để lựa chọn một phép toán phân theo bit            */
/* trên hai giá trị được nạp vào                      */
#include <stdio.h>
#define   OR    1
#define   AND   2
#define   EX-OR 3
#define NOR     4
#define   NAND  5
int main(void)
{
int value1,value2,option;

    puts("Program to determine bitwise operation on two values");
    printf ("Enter two hex values >>>");
    scanf("%x %x",&value1,&value2);

    puts("Enter function required");
```

```
    printf("1-OR \n2-AND \n3-EX-OR \n");
    printf("4-NOR \n5-NAND \nenter >> ");
    scanf("%d",&option);

  if (option== OR)
   printf ("Ans is %x (hex)\n",value1 | value2);
                  else if (option==AND)
   printf ("Ans is %x (hex)\n",value1 & value2);
                  else if (Option==EX-OR)
   printf ("Ans is %x (hex)\n",value1 ^ value2);
                  else if (option==NOR)
   printf ("Ans is %x (hex)\n",~(value1 | value2));
                  else if (option==NAND)
   printf ("Ans is %x (hex)\n",~(value1 & value2));
                  else puts ("INVALID OPTION");

  return(0);
 }
```

Chạy thử 3.5 chỉ ra số liệu nhập vào không hợp lệ, nghĩa là một số bất kỳ khác với 1; 2; 3; 4 hoặc 5.

Chạy thử 3.5

```
Program to determine bitwise operation on two values
Enter two hex values >>> 532e 3210
Enter function required
1 - OR
2 - AND
3 - EX - OR
4 - NOR
5 - NAND
Nhập vào Option>>7
INVALID OPTION
```

Chạy thử 3.6 là kết quả chạy mẫu với phép tính AND.

Chạy thử 3.6

```
Program to determine bitwise operation on two values
Enter two hex values >>> 532e 3210
Enter function required
1 - OR
2 - AND
3 - EX - OR
4 - NOR
5 - NAND
ENTER >> 2
Answer is 1200 (hex)
```

Các điện trở mắc nối tiếp và song song

Chương trình 3.4 xác định điện trở tương đương của hai điện trở mắc nối tiếp hoặc song song. Chương trình sử dụng `scanf()` để cho hai giá trị điện trở và `getchar()` lựa chọn loại mạch. Vấn đề có thể xuất hiện khi sử dụng `getchar()` sau `scanf()` vì những ký tự dòng mới được lưu giữ trong bộ đệm bàn phím. Lệnh `fflush(stdin)` được chèn vào chương trình để xoá bộ đệm trước khi `getchar()` được gọi (`stdin` có nghĩa là thiết bị nhập vào chuẩn - standard input, cụ thể là bàn phím).

Chương trình cũng sử dụng `tolower()` để chuyển đổi ký tự lựa chọn mạch điện được nhập vào thành ra chữ thường (hàm này đã được đặt làm mẫu - protyped - trong tệp *ctype.h*).

Chương trình 3.4

```
/* prog3_4.c                                          */
/* Chương trình xác định điện trở tương đương của     */
/* hai điện trở đấu hoặc nối tiếp hoặc song song      */
#include <stdio.h>
#include <ctype.h> /* required for tolower() function */
 int   main(void)
 {
```

```
float R1,R2,R_equ;
char  ch;

printf("Enter two resistance values >>");
scanf("%f %f",&R1,&R2);
fflush(stdin); /* flush keyboard buffer      */
printf("Do you require (s)eries or (p)arallel>>");
ch=getchar();

if (tolower(ch)=='s')/*convert character to lowercase */
{
  R_equ=R1+R2;
  printf("Equivalent series resistance is %8.2f ohms",R_equ);
}
else if (tolower(ch)=='p')
  R_equ=(R1.R2)/(R1+R2);
  printf ("Equivalent parallel resistance is.2fhms",R_equ);
}
else puts("Invalid entry");
return(0);
}
```

Các phương trình bậc hai

Một số bài tập điện dẫn đến việc giải phương trình bậc hai. Dạng chuẩn của phương trình bậc hai là:

$$ax^2 + bx + c = 0$$

Nghiệm x trong phương trình này được viết dưới dạng:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Nghiệm này có thể dẫn đến ba dạng kết quả:

1. Nếu như $b^2 = 4ac$ thì sẽ có nghiệm thực (x = -b/ 2a)

2. Nếu như $b^2 > 4ac$, sẽ có hai nghiệm thực:

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \quad \text{và} \quad x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

3. Nếu không (nghĩa là $b^2 < 4ac$) các nghiệm sẽ là phức:

$$x_1 = \frac{-b}{2a} + j\frac{\sqrt{4ac-b^2}}{2a} \quad \text{và} \quad x_1 = \frac{-b}{2a} - j\frac{\sqrt{4ac-b^2}}{.2a}$$

Chương trình 3.5 xác định nghiệm của một phương trình bậc hai. Trong chương trình này lệnh `if...else` được sử dụng để xác định nếu nghiệm là thực, phức hoặc đơn. Giá trị nhập vào hàm căn bậc hai (`sqrt()`), có trong *math.h*, sẽ được kiểm tra để xác định xem liệu nó có giá trị âm. Nếu như nó âm, thì có thể làm cho chương trình dừng lại vì không thể tính được căn bậc hai của một số âm. Chương trình cũng có thể bị dừng lại nếu như a bằng 0 bởi vì không thể thực hiện phép chia cho số 0 (cái bẫy để phát hiện lỗi này được tháo gỡ ở một bài tập thực hành).

Chương trình 3.5

```
/* prog3_5.c                                                    */
/* Chương trình tìm nghiệm của một phương trình bậc hai */
#include <stdio.h>
#include <math.h>
int     main(void)
{
float a,b,c,real1,real2,img;

   puts("Program to determine roots of a quadratic equation");
   printf("Enter a,b and c >>>");
   scanf("%f %f %f",&al&b,&c);
   printf("Equation is %.2fx*x+%.2fx+%.2f\n",a,b,c);
   if ((b*b)-(4*a*c))
   ( /*  singular root      :    */
   real1=(b*b)/(2*a);
   printf("Root is %.2f\n",real1);
   else if ((b*b)>(4*a*c))
```

```
    {     /*    real roots              */
    real1=(-b+sqrt( (b*b)-4*a*c )) /(2*a);
    real2=(-b-sqrt( (b*b)-4*a*c )) /(2*a);
    printf("Roots are %.2f, %.2f\n",real1,real2);
    } else
    {    /*    complex roots           */
    real1=(b*b)/(2*a);
    imag=sqrt(4*a*c-b*b)/(2*a);
    printf("Roots are %.2f +/- j%.2f\n",real1,imag);
    }
    return (0) ;
}
```

Ba kết quả chạy thử: 3.7, 3.8 và 3.9 được tiến hành khi kiểm tra với ba loại nghiệm. Trong kết quả chạy thử 3.7 các nghiệm của phương trình là thực.

Chạy thử 3.7

```
Chương trình tìm nghiệm của một phương trình bậc hai
ENTER a, b và c  >>> 1 1 -2
Phương trình là 1.00 x*x + 1.00x - 2.00
Các nghiệm là 1.00 và -2.00
```

Kết quả chạy thử 3.8 là phức, nghĩa là có dạng x + *j*y.

Chạy thử 3.8

```
Chương trình tìm nghiệm của một phương trình bậc hai
ENTER a, b và c  >>> 2 2 4
Phương trình là 2.00 x*x + 2.00x - 4.00
Các nghiệm là 1.00 +/- j1.32
```

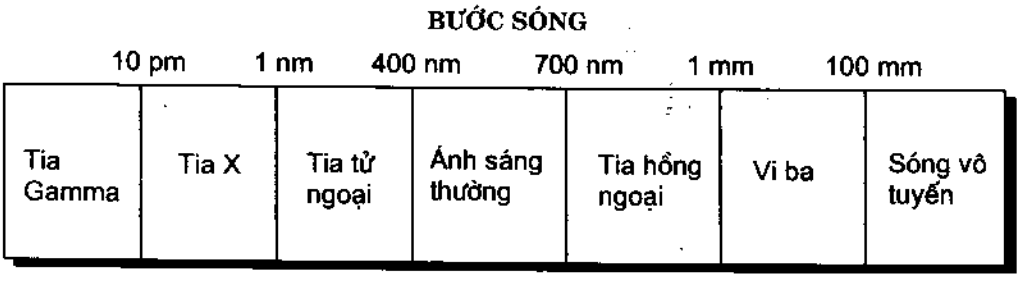
Trong kết quả chạy thử 3.9 ta nhận được nghiệm đơn.

Chạy thử 3.9

```
Chương trình tìm nghiệm của một phương trình bậc hai
ENTER a, b và c  >>> 1 2 1
Phương trình là 1.00 x*x + 2.00x - 1.00
Các nghiệm là 2.00
```

Sóng điện từ (EM)

Chương trình 3.6 sử dụng lệnh `if` để phân loại sóng điện từ theo bước sóng. Hình 3.7 minh họa phổ sóng điện từ khi dãn rộng bước sóng theo tỷ lệ khác nhau. Việc phân loại sóng điện từ được tiến hành hoặc theo tần số hoặc theo bước sóng (thông thường sóng vô tuyến và vi ba được đặc trưng bằng tần số của chúng, trong khi còn có một cách khác là dùng bước sóng). Thí dụ như sóng điện từ với bước sóng 10 cm xếp vào loại sóng vô tuyến, bước sóng 500 nm là ánh sáng nhìn thấy và bước sóng 50 cm nằm trong vùng vi ba.



Hình 3.7: **Phổ sóng điện từ.**

Chương trình 3.6

```
/* prog3_6.c                                          */
/* Chương trình xác định loại sóng điện từ            */
/* ứng với một giá trị bước sóng cho trước.           */
#include <stdio.h>
int main (void)
{
float lamda;
```

```
printf("Enter bước sóng >>>")
scanf ("%f",&lambda);

printf ("Electromagnetic wave");
if     (lambda<1e-11)      puts ("Gamma Ray !!!")
else if  (lambda<1e-9)     puts ("X-ray")
else if  (lambda<400e-9)   puts ("Ultraviolet")
else if  (lambda<700e-9)   puts ("LIGHT")
else if  (lambda<1e-3)     puts ("infrared")
else if  (lambda<1e-1)     puts ("Microwave")
else                       puts ("Radio wave")

return (0) ;
}
```

Kết quả chạy thử 3.10 là một thí dụ làm mẫu

Chạy thử 3.10

```
Nhập vào giá trị bước sóng >>> 1e-10
Sóng điện từ trong trường hợp này là tia X
```

Các sóng điện từ cũng có thể được đặc trưng bằng tần số của chúng. Chương trình 3.7 cho phép người dùng nhập vào tần số của sóng, và chương trình lúc đó xác định bước sóng bằng cách sử dụng công thức:

$$\lambda = \frac{c}{f}$$

ở đây c là tốc độ của ánh sáng còn f là tần số của sóng điện từ.

Chương trình 3.7

```
/* prog3_7.c                                                    */
/* Chương trình để xác định dạng sóng để nhập vào tần số  */
 #include <stdio.h>
 #define  SPEED_OF_LIGHT 3e8

 int   main (void)
 (
```

```
float     lambda, freq;
  printf ("Enter frequency>>>");
  scanf ("%f",&freq");

  lambda=SPEED_OF_LIGHT/freq;

  printf ("Wavelength is %. 2e m. EM wave is ", lambda);
  if        (lambda<1e-11)   puts ("Gamma Rays!!!")
  else if   (lambda<1e-9)     puts ("X-ray")
  else if   (lambda<400e-9)   puts ("Ultraviolet")
  else if   (lambda<700e-9)   puts ("LIGHT")
  else if    (lambda<1e-3)    puts ("infrared")
  else if   (lambda<0.3e-1)   puts ("Microwave")
  else                        puts ("Radio waves")

  return (0) ;
}
```

Kết quả chạy thử 3.11 là một thí dụ làm mẫu.

Chạy thử 3.11

```
Nhập vào giá trị tần số >>>  10e9
Bước sóng là 3,0e-02 m. Sóng điện từ là sóng vi ba
```

3.2 LỆNH switch

Lệnh `switch` (*rẽ nhánh*) được sử dụng khi có nhiều quyết định phải thực hiện. Bình thường lệnh này được sử dụng để thay thế cho lệnh `if` khi có nhiều cách thực hiện chương trình. Cú pháp của lệnh `switch` như sau:

```
switch (expression)
{
  case const1:  statement(s) : break;
  case const2:  statement(s) ; break.
  :     :
  default:      statement(s) ; break;
 }
```

Lệnh `switch` kiểm tra biểu thức (*expression*) gắn vào mỗi hằng số trong chuỗi (hằng số phải là số nguyên hoặc kiểu dữ liệu ký tự). Khi có sự phù hợp (các) lệnh kết hợp với hằng số đã được thực hiện. Việc chấp hành diễn ra với tất cả các lệnh khác cho đến khi lệnh `break` xuất hiện hoặc đến sự kết thúc của `switch`, bất kể là sự việc nào xảy ra trước.

Nếu như không có hằng số phù hợp với biểu thức `switch` thì một tập hợp các lệnh kết hợp với điều kiện mặc định (`default`) sẽ được thực hiện.

3.2.1 MỘT SỐ THÍ DỤ

Mã màu dùng cho điện trở

Thông thường các giá trị điện trở có thể đọc được thông qua hệ thống các vòng màu hay còn gọi là mã màu, như minh hoạ trong bảng 3.1.

Bảng 3.1: **Hệ thống mã màu điện trở.**

Digit	Màu	Hệ số nhân	Digit	Màu	Hệ số nhân
	Bạc nhũ	0,01	4	Vàng	10 k
	Vàng nhũ	0,1	5	Xanh	100 k
0	Đen	1	6	Lam	1000 k = 1 M
1	Nâu	10	7	Tím	10 M
2	Đỏ	100	8	Xám	
3	Da cam	1000 = 1 k	9	Trắng	

Chương trình 3.8 sử dụng một lệnh `switch` để xác định màu của các vòng màu trên thân điện trở đối với một giá trị điện trở được nhập vào. Biến sử dụng `colour` (màu) đã được khai báo như một `unsigned int` (số nguyên không dấu) bởi vì giá trị nhập vào luôn luôn dương. Muốn thế, `scanf()` có ký hiệu `%u` để chỉ rõ khuôn mẫu.

Chương trình 3.8

```
/* prog3_8.c                                          *
/* Program to determine colour code for a single */
```

```
/* resistor band digit                                    */
#include <stdio.h>

int main(void)
{
unsigned int colour;
  /* char or unsigned char could also be used     */
 printf("Enter value of colour band (0-9)>>")
  scanf("%u",&colour)

  printf("Resistor colour band is ");

 switch(colour)
 {
   case 0 : printf ("BLACK");  break;
   case 1 : printf ("BROWN");  break;
   case 2 : printf ("RED");      break;
   case 3 : printf ("ORANGE");   break;
   case 4 : printf ("YELLOW");   break;
   case 5 : printf ("GREEN");  break;
   case 6 : printf ("BLUE");     break;
   case 7 : printf ("VIOLET");   break;
   case 8 : printf ("GREY");     break;
   case 9 : printf ("WHITE");  break;
}
return(0) ;
{
```

Kết quả chạy thử 3.12 giới thiệu một thí dụ làm mẫu.

Chạy thử 3.12

```
Nhập vào số các vòng màu (0-9)>> 3
Vòng màu là da cam (ORANGE)
```

Chương trình 3.9 sử dụng chỉ dẫn hướng `#define` để định nghĩa các vòng màu trên điện trở. Có thể xảy ra xung đột với định nghĩa này nếu những đầu mục tập tin khác chứa những định nghĩa này. Trong trường hợp đó phải thay đổi các định nghĩa bằng cách đặt tên khác đi, chẳng hạn: RES_BLACK, RES_BROWN, v. v...

Sự mặc định: đã được bổ sung để bẫy các giá trị nhập vào không hợp lệ (như nhỏ hơn 0 hoặc lớn hơn 9).

Chương trình 3.9

```
/* Prog 3_9.c                                    */
/* Program to determine colour code              */
/* for resistor or band digit                    */
  #include <stdio.h>
  #define    BLACK    0
  #define    BROWN    1
  #define    RED      2
  #define    ORANGE   3
  #define    YELLOW   4
  #define    GREEN    5
  #define    BLUE     6
  #define    VIOLET   7
  #define    GREY     8
  #define    WHITE    9

 int main(void)
{
 unsigned int colour;

      printf("Enter value of colour band (0-9)>>");
      scanf ("%u",&colour);
      printf ("Resistor colour band is ");

      switch  (colour)
      {
     case BLACK:     printf ("BLACK");  break;
```

```
    case BROWN:     printf ("BROWN");  break;
    case RED:       printf ("RED");       break;
    case ORANGE:    printf ("ORANGE");    break;
    case YELLOW:    printf ("YELLOW");    break;
    case GREEN:.    printf ("GREEN");  break;
    case BLUE:      printf ("BLUE");      break;
    case VIOLET:    printf ("VIOLET");    break;
    case GREY:      printf ("GREY");      break;
    case WHITE:     printf ("WHITE");  break;
    default:          printf ("NO COLOUR"); break;
    }
return(0);
{
```

Một kiểu khuôn mẫu mới để định nghiã một dãy giá trị số nguyên là sử dụng kiểu dữ liệu enum. Chương trình 3.10 có cùng tác dụng như chương trình trước; nó định nghĩa BLACK (màu đen) là 0, BRAUN (màu nâu) là 1, v. v... Khai báo enum thiết lập một dãy các giá trị số nguyên. Nếu như dãy này bắt đầu từ 0, thì không cần phải thiết lập trạng thái ban đầu của giá trị, bằng không thì tham số đầu tiên phải được tạo khởi. Một mẫu khai báo enum cho ra dưới đây, khai báo này khởi tạo dãy từ -2 (silver) và dãy sẽ là bạc nhũ (silver)=-2, vàng nhũ (gold) =-1, đen (black)= 0, v. v... tới màu trắng (white) = 9 và biến được khai báo là colour.

```
enum (SILVER=-2, GOLD, BLACK, BROWN, RED, ORANGE,
YELLOW, GREEN, BLUE, VIOLET, GREY, WHITE) colour;
```

Một ưu điểm của việc sử dụng kiểu dữ liệu enum là biến chỉ có thể lấy các tên của các sự khai báo, thí dụ colour = 2 là sai, trong khi colour = đỏ là đúng.

Biện pháp này hoàn thiện khả năng kiểm tra lỗi của chương trình bằng cách hạn chế sử dụng khả năng của biến.

Chương trình 3.10

```
/* prog3_10.c                                    */
#include <stdio.h>
```

```
int main(void)
{
enum     (BLACK,BROWN,RED,ORANGE,YELLOW,
          GREEN,BLUE,VIOLET,GREY,WHITE) colours;

 printf("Enter value >>");
 scanf("%d",&colour);

 printf("Resistor colour band is") ·

 switch (colours)
 {
  case BLACK:     printf("BLACK");      break;
  case BROWN:     printf("BROWN");      break;
  case RED:       printf("RED");        break;
  case ORANGE:    printf("ORANGE");     break.
  Case YELLOW:    printf("YELLOW");     break:
  case GREEN:     printf("GREEN");      break;
  case BLUE:      printf("BLUE");       break;
  case VIOLET:    printf("VIOLET");     break;
  case GREY:      printf("GRAY");       break;
  case WHITE:     printf("WHITE");      break;
  default:        printf("NO COLOUR");     break;
      }
      return(0);
     }
```

Chức năng của vi mạch TTL

Một họ các vi mạch tổ hợp (IC) có tên là họ 74 có thể được dùng để bổ sung các hàm lôgic tiêu chuẩn như AND, OR, NOR, NAND, v. v... Chẳng hạn, 7400, 7402 và 7408 có chứa các cổng NAND, NOR, và AND tương ứng.

Trong chương trình 3.11 người dùng nhập vào kiểu của hàm và chương trình xác định kiểu của vi mạch TTL mà nó có thể bổ sung. Nó

sử dụng lệnh switch để chọn lựa từ một thực đơn cho AND, OR, NAND, NOR và NOT 2 lối vào (bộ đệm đảo). Như trước đây, kiểu dữ liệu enum được sử dụng để định nghĩa các hàm lôgic.

Chương trình 3.11

```
/* prog3_11.c                                    */
/* Chương trình để xác định TTL cho hàm          */
#include <stdio.h>

int     main(void)
{
enum    (AND=1,OR,NAND,NOR,NOT) gate_type;

   puts("Enter logic gate required")
   puts("1  - 2-input  AND gate");
   puts("12 - 2-input  OR gate");
   puts("3  - 2-input  NOR gate");
   puts("4  - 2-input  NOR gate");
   puts("5  - 2-input  NOT gate");
   puts("6 - exit program");
   scanf("%d",&gate_type) ;

  printf("TTL gate(s) available is (are)")
  switch(gate_type)
  {
    case AND:  puts("7408");                          break;
    case OR:   puts("7432");                          break;
    case NAND:  puts("7400,7401,7403,7437,7438");     break;
    case NOR:  puts("7402,7428,7433");                break;
    case NOT:  puts("7404,7405,7505,7416");           break;
    default:   puts("Invalid option");
    }
    return(0);
    }
```

Các thí dụ tiêu biểu được cho trong kết quả chạy thử 3.13 và 3.14.

Chạy thử 3.13

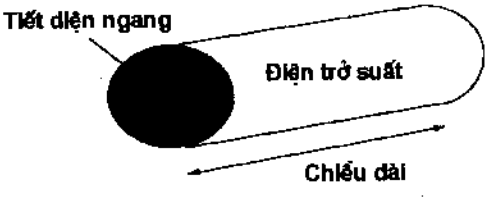
```
Nhập vào cổng lôgic cần có
1 - 2: lối vào của cửa AND
2 - 2: lối vào của cửa OR
3 - 2: lối vào của cửa NAND
4 - 2: lối vào của cửa NOR
5 - 2: lối vào của cửa NOT
1
Các cửa TTL tương ứng là 7408
```

Chạy thử 3.14

```
Nhập vào cổng lôgic cần có
1 - 2: lối vào của cửa AND
2 - 2: lối vào của cửa OR
3 - 2: lối vào của cửa NAND
4 - 2: lối vào của cửa NOR
5 - 2: lối vào của cửa NOT
3
Các cửa TTL tương ứng là 7400, 7401, 7403, 74037, 7438.
```

Điện trở của một dây dẫn

Điện trở của một dây dẫn hình trụ là một hàm của điện trở suất, tiết diện ngang và độ dài của nó. Những tham số này được minh họa trong hình 3.8.



Hình 3.8: **Dây dẫn hình trụ.**

Điện trở được tính theo công thức:

$$R = \frac{\rho.l}{S} \quad [\Omega]$$

ở đây:

ρ điện trở suất của dây dẫn (Ω.m)
l độ dài của dây dẫn (m)
S tiết diện ngang của dây dẫn (m^2)

Chương trình 3.12 xác định điện trở của một dây dẫn hình trụ làm từ bạc, mangan, nhôm hoặc đồng. Điện trở suất của các chất này có thể được định nghĩa khi sử dụng macro `#define`.

Người dùng nhập vào kiểu dây dẫn dưới dạng một ký tự ('c', 'a', 's', hoặc 'm') theo khuôn mẫu chữ hoa hoặc chữ thường đều được bởi vì hàm `tolower()` biến đổi ký tự nhập vào thành chữ thường (hàm này đã được đặt làm mẫu trong *ctype.h*). Khi nhập vào một ký tự không hợp lệ điều kiện mặc định của lệnh `case` sẽ tác động và thông báo: `Invalid option` sẽ xuất hiện trên màn hình. Chương trình gọi hàm `exit();` tham số chuyển giao vào hàm này là sự kết thúc quá trình. Một giá trị 0 mô tả sự kết thúc bình thường; bất kỳ giá trị khác báo hiệu sự kết thúc chương trình dị thường.

Lệnh `printf()` hiển thị điện trở trong cách viết khoa học `(%e)` như những giá trị tiêu biểu nhiều hoặc ít hơn 1 Ω (như mΩ hoặc Ω).

Chương trình 3.12

```
/* prog3_12.c                                              */
/* Chương trình xác định điện trở dây dẫn hình trụ        */
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

/* Define resistivities */
#define  RHO-copper   17e-9
#define  RHO-AL       25.4e-9
#define  RHO-SILVER     16e-9
```

```
#define  RHO-MANGANESE  1400e-9
#define  PI             3.14

int main(void)
{
float    radius,length.area,rho,resistance;
char     ch;

    puts("Type of conductor >>");
    puts("(c)opper");
    puts("(a)luminim") ;
    puts("(s)ilver");
    puts("(m)anganese");
    /* get conductor type                    */
    ch=getchar();

    printf("Enter,radius and length of conductor >>") ;
    scanf("%f %f",&radius,&length);
    /* area of conductor                     */
    area=PI-(radius*radius);
    /* convert to lawercase and determine resistivity  */
    switch (tolower(ch))
    {
    case 'c': rho=RHO-COPPER;       break;
    case 'a': rho=RHO-AL;           break;
    case 's': rho=RHO-SILVER;       break;
    case 'm': rho=RHO-MANGANESE;    break;
    default: puts("invalid option"); exit(0); break;
  }
  resistance=rho*length/area;
  printf("Resistance of conductor is %.3e ohm", resistance)break;
  return(0)
    }
```

Kết quả chạy thử 3.15 sử dụng một dây dẫn bằng nhôm với bán kính 1 mm và chiều dài 1000 m. Điện trở tìm được là 8,08 Ω.

Chạy thử 3.15

```
Loại vật liệu dẫn điện >>
(c)opper: đồng
(a)luminum: nhôm
(s)ilver: bạc
(m)anganese: mangan
a
Nhập vào bán kính và chiều dài của dây dẫn >> 1e-3 1000
Điện trở của dây dẫn là 8.09e+000 Ohm
```

Có thể có vài tùy chọn case trong lệnh switch. Thí dụ, nếu như hàm tolower()không được sử dụng trong chương trình 3.12, thì tùy chọn case có thể được sửa đổi để bao gồm tuỳ chọn trên và dưới, như chỉ ra trong các dòng mã dưới đây:

```
switch (ch)
{
   case 'c': rho=RHO-COPPER;      break;
   case 'a': rho=RHO-AL;          break;
   case 's': rho=RHO-SILVER;      break;
   case 'm': rho=RHO-MANGANESE;     break;
   default: puts("invalid option");exit(0);
}
```

3.3 THỰC HÀNH

Q3.1 Xác định các lỗi trong chương trình 3.13 và 3.14.

Chương trình 3.13

```
/* prog3_13.c                                    */
#include <stdio.h>
int  main(void)
{
int , i;
puts("Enter value of i");
scanf("%d",i);
```

```
if (i - 5) puts("i is equal to five");
return(0);
}
```

Chương trình 3.14

```
/* prog3_14.c                                    */
/* Simple calculator                             */
#include <stdio.h>
int     main(void)
{
int     a=5,b=3;
char     ch;
puts("Enter operator (+,-,* or /);
ch = getchar();
switch (ch)
case '+':  c=a+b;
case '-':  c=a-b;
case '.':  c=a*b;
case '/':  c=a/b;
print("%d  %c %d = %d",a,ch,b,c);
return(0);
}
```

Q3.2 Xác định đầu ra từ chương trình 3.15 nếu như người dùng nhập vào một giá trị bằng 2. Sửa đổi chương trình để nó hoạt động đúng.

Chương trình 3.15

```
/* prog3_15.c                                    */
#include <stdio.h>

int:  main(void)
{
int   a;
puts("Enter a number");
scanf("%d",&a);
```

```
switch (a)
{
 case 1:  puts("1 entered");
 case 2:  puts("2 entered");
 case 3:  puts("3 entered");
 case 4:  puts("4 entered");
default:  puts("Not 1,2,3 or 4")
}
return(0);
}
```

Q3.3 Hãy sửa đổi chương trình 3.2 sao cho nó hiển thị những giá trị trong phạm vi 0 đến 65.535 dưới dạng nhị phân không dấu 16 bit. Kiểu dữ liệu `unsigned int` có thể được sử dụng để khai báo một số nguyên 16 bit với bit không dấu.

Kết quả chạy mẫu giới thiệu trong chạy thử 3.16.

Chạy thử 3.16

```
Enter unsigned integer (0 to 65535)>> 1
Binary value is 0000000000000001
Enter unsigned integer (0 to 65535)>> 5
Binary value is 0000000000000101
Enter unsigned integer (0 to 65535)>> 43
Binary value is 0000000000101011
Enter unsigned integer (0 to 65535)>> 245
Binary value is 0000000011110101
Enter unsigned integer (0 to 65535)>> 5321
Binary value is 0001010011001001
Enter unsigned integer (0 to 65535)>> 32865
Binary value is 1000000001100001
Enter unsigned integer (0 to 65535)>> 65534
Binary value is 1111111111111110
Enter unsigned integer (0 to 65535)>> 0
Binary value is 0000000000000000
```

Q3.4 Hãy sửa đổi chương trình Q3.3 để nó hiển thị những giá trị trong khoảng từ - 32,768 đến 32,767 dưới dạng một số nhị phân có dấu 16 bit. Sử dụng kiểu dữ liệu `int` cho giá trị được nhập vào.

Q3.5 Hãy sửa đổi chương trình 3.4 sao cho nó không thể phát sinh lỗi chia cho số không, nghĩa là khi a bằng 0. Chú ý là nếu như a bằng 0 thì nghiệm sẽ bằng `-c/b`.

Q3.6 Hãy sửa đổi chương trình 3.7 sao cho người dùng có thể nhập vào thông số đặc trưng của sóng điện từ dưới dạng giá trị tần số hoặc bước sóng. Kết quả chạy mẫu được minh hoạ trong chạy thử 3.17.

Chạy thử 3.17

```
Do you wish to enter
(f)requency or
(w)avelegth >>>
f
Enter frequency>>> 10eg
Wavelength is 3.0e-02 Electromagnetic wave is Microwave
```

Q3.7 Hãy sửa đổi chương trình Q3.6 sao cho nó sử dụng chỉ dẫn hướng `#define` để xác định các giới hạn bước sóng, chẳng hạn:

```
#define GAMMA_RAY_LIMIT 1e-11
```

Q3.8 Hãy viết một chương trình trong đó người dùng nhập vào một số nguyên và được lưu ý liệu có phải nó được hiển thị như một số bát phân, một số thập lục phân hoặc một ký tự ASCII. Kết quả chạy mẫu được minh hoạ trong chạy thử 3.18.

Chạy thử 3.18

```
Enter decimal : 15
Do you wish (H)ex, (O)ctal : h
Value in hexadecimal is F (Giá trị thập nhị phân là F)
```

Q3.9 Hãy viết lại chương trình trong Q3.8 sao cho người dùng có thể nhập vào một giá trị hoặc dưới dạng một số thập phân, một số bát phân hoặc số thập lục phân hoặc một ký tự.

Chạy thử 3.19

```
Type of entered value (H)e.,(O)ct.,(D)ecimal or (C)har:
Enter decimal : 15
Do you wish to convert to (H)e., (O)ctal, (C)har : h
Value in hexadecimal is F (Giá trị thập nhị phân là F)
```

Q3.10 Thông thường điện dung được xác định qua một giá trị và một đơn vị được đo như pF, nF, μF, mF hoặc F. Hãy viết một chương trình trong đó giá trị điện dung và đơn vị được nhập vào và chương trình hiển thị giá trị số hiện tại ra Fara. Đơn vị điện dung sẽ nhập vào dưới dạng một ký tự. Kết quả chạy làm thí dụ được minh hoạ trong chạy thử 3.20. Tham khảo thêm chương trình 3.12.

Chạy thử 3.20

```
Nhập vào giá trị: 1
Nhập vào đơn vị: p, n, u hoặc m: u
Giá trị điện dung bằng 0.000001 Fara
```

Q3.11 Hãy sửa đổi chương trình Q3.10 để người dùng nhập vào giá trị điện dung hiện tại và chương trình hiển thị giá trị này với đơn vị thích hợp nhất. Một thí dụ minh hoạ được dẫn ra trong chạy thử 3.21. Tham khảo thêm chương trình 3.7.

Chạy thử 3.21

```
ENTER capacitance: 5e-6 (Nhập vào giá trị điện dung)
Điện dung bằng 5 μF
```

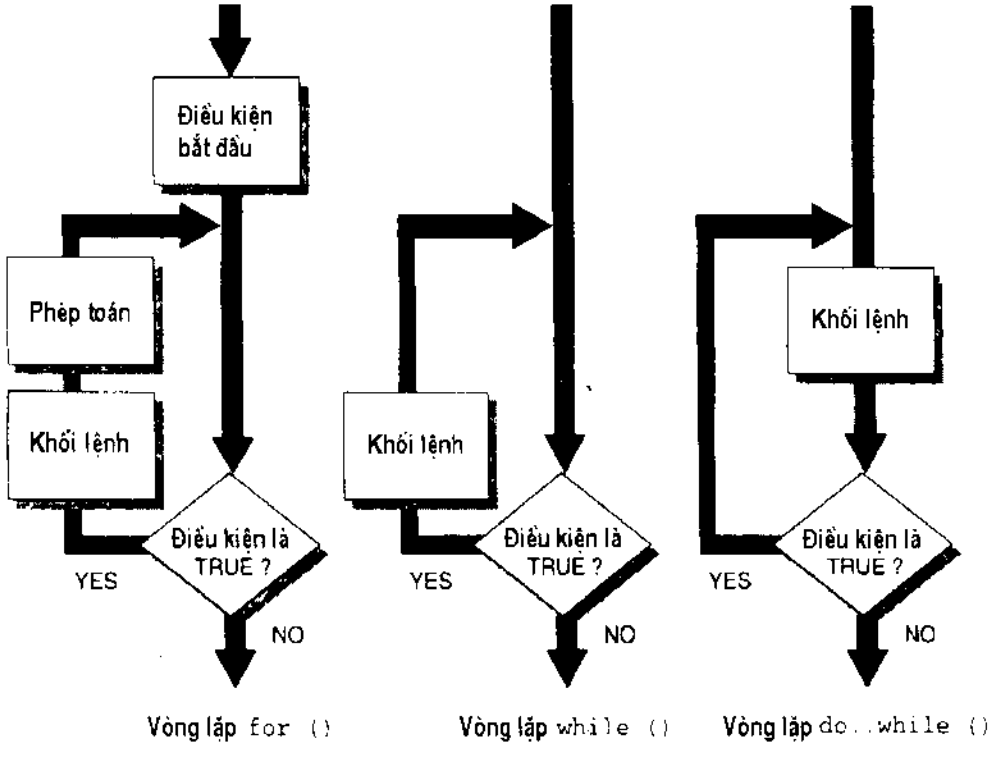
Q3.12 Hãy lặp lại Q3.10 đối với giá trị của điện trở. Các đơn vị được nhập vào hoặc là mΩ ('m'), Ω ('1'), kΩ ('k') hoặc là MΩ ('M'). Kết quả chạy thử 3.22 chỉ ra một kết quả làm thí dụ.

Chạy thử 3.22

```
Enter value: 3,21  ; Nhập vào giá trị
Enter unit m,1,k, or M : k
Resistance value is 3210 Ohms ; Giá trị điện trở
```

Chương 4
LỆNH LẶP

Một quá trình được tái diễn lại, hoặc lặp đi lặp lại, cho phép xếp một tập hợp các lệnh thành một vòng kín. Có ba dạng lặp đi lặp lại là các vòng: `while`, `do` và `for`. Hình 4.1 chỉ ra tiến trình chung của các vòng này.



Hình 4.1: **Tiến trình của các vòng lặp.**

Tất cả đều đòi hỏi một điều kiện kiểm tra để quyết định xem liệu vòng lặp có tiếp tục nữa hay thôi. Nếu điều kiện kiểm tra này là đúng (TRUE) thì vòng lặp sẽ tiếp tục, nếu không thì vòng lặp sẽ dừng. Vòng lặp `for` có một điều kiện bắt đầu và một phép toán ở bên trong vòng lặp. Các vòng `do...while()` và `while()` gần giống nhau, nhưng chúng kiểm tra điều kiện lặp lại vòng ở những điểm khác nhau.

4.1 VÒNG LẶP for...

Nhiều nhiệm vụ trong một chương trình có thể được lặp lại, chẳng hạn nhắc nhập dữ liệu, đếm giá trị, v. v... Vòng lặp `for` cho phép chấp hành một khối mã dành cho một chức năng điều khiển đã đặt trước. Sau đây là một khuôn dạng được đưa ra làm thí dụ; nếu như chỉ có một lệnh trong khối thì dấu ngoặc nhọn ({}) có thể được bỏ đi.

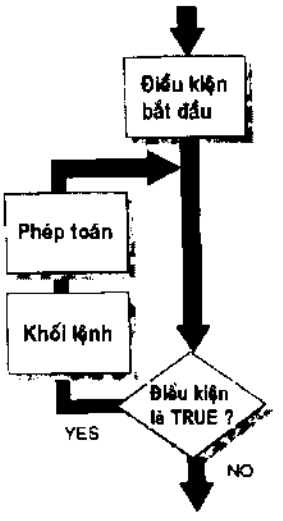
```
for (starting condition; test condition; operation)
{
statement block
}
```

ở đây:

`starting condition`	- giá trị bắt đầu của vòng lặp;
`test condition`	- nếu điều kiện kiểm tra này là đúng (TRUE) thì vòng lặp sẽ được tiếp tục chấp hành;
`operation`	- phép toán được sắp đặt ở cuối của vòng lặp.

Hình 4.2 chỉ ra tiến trình (lưu đồ) của lệnh này.

```
for  (starting condition; test condition; operation)
{
statement block
}
```



Hình 4.2: **Biểu diễn lệnh for bằng lưu đồ.**

Một số thí dụ:

(a) `for (i=1;i<10;i++)`

i sẽ bắt đầu ở một giá trị bằng 1, và cứ mỗi lần chạy xong một vòng nó lại được gia tăng thêm 1(`i++`). Vòng lặp sẽ dừng lại khi nó đạt giá trị bằng 10, nghĩa là giá trị cuối cùng mà i có được ở bên trong vòng sẽ là 9.

(b) `for (index=100 ; index < 500 ; index += 50)`

chỉ số (index) sẽ bắt đầu ở 100; mỗi lần chạy xong một vòng thì 50 lại được cộng thêm vào (`index += 50` là tương đương với `index = index+50`). Bên trong vòng, chỉ số (index) sẽ bằng 100, 150, 200, 250....400, 450.

(c) `for (; ;)`

diễn tả một vòng lặp vô tận.

(d) `for (i=2;i<=128;i *= 2)`

bắt đầu với i bằng 2; mỗi lần chạy qua một vòng i lại được tăng thêm 2. Qúa trình này tiếp tục chừng nào i còn nhỏ hơn hoặc bằng

128. Các giá trị của i ở bên trong vòng sẽ là 2; 4; 8; 16; 32; 64 và 128.

(e) `for (k=-20.2; h>32; z--,j++)`

bắt đầu với k bằng -20,2; vòng lặp sẽ tiếp tục chừng nào h còn lớn hơn 32. Hết một vòng z bị giảm đi 1 còn j được tăng thêm 1. Dấu phẩy cho phép có nhiều hơn một biểu thức trong vùng thứ nhất và vùng thứ ba của `for()`.

4.1.1 MỘT SỐ THÍ DỤ

Các ký tự mã ASCII

Chương trình 4.1 hiển thị các ký tự mã ASCII đối với các giá trị thập phân bắt đầu và kết thúc được nhập vào.

Chương trình 4.1

```
/* prog4_1.c
/* Chương trình để in các ký tự mã ASCII  */
#include <stdio.h>
int   main(void)
{
int   i,start,end;

  printf("Enter start and end for ASCII characters>>");
  scanf("%d %d",&start,&end);

  puts("INTEGER   HEX    ASCII");

  for    (i=start;i<=end;i++)
    printf("%5d %5x %5c\n",i,i,i);
  return(0);
 }
```

Kết quả chạy thử 4.1 hiển thị ký tự mã ASCII từ các số thập phân 40 ('(') đến 50 ('z').

Chạy thử 4.1

```
INTEGER   HEX     ASCII
40        28      (
41        29      )
42        2a      *
43        2b      +
44        2c      ,
45        2d      -
46        2e      .
47        2f      /
48        30      0
49        31      1
50        32      2
```

Chuyển đổi số nguyên thành số nhị phân

Chương trình 4.2 là một phiên bản đã được hoàn thiện của chương trình 3.1. Nó hiển thị một giá trị thập phân được nhập vào dưới dạng nhị phân không dấu 8 bit. Trước hết, chương trình che bit có giá trị cao nhất (0x80 hay 10000000b) và xác định nếu giá trị là TRUE (nghiã là bit được đặt) hoặc FALSE (nghĩa là bit được đặt bằng 0). Nếu như bit được đặt thì số "1" sẽ được hiển thị; nếu không thì số "0" được hiển thị. Sau đó mặt nạ bit đổi chỗ về phía dưới một vị trí bit bằng cách sử dụng toán tử xử lý tới bit SHIFT phải (>>); phép toán che một bit lại được thực hiện một lần nữa. Quá trình này sẽ tiếp tục cho đến khi vòng lặp đạt tới bit có giá trị thấp nhất (cụ thể là 0x01 hay 0000 0001b). Sau đó, vòng lặp sẽ kết thúc và chương trình ngừng lại.

Vòng lặp `for()` sử dụng phép toán bit >>= 1 để di chuyển mặt nạ bit đi một vị trí sang bên trái. Đây là một tương đương `shorthand` của `bit = bit >> 1`.

Kết quả chạy thử 4.2 giới thiệu một thí dụ chạy mẫu.

Chương trình 4.2

```
/* prog4_2.c                                          */
/* Program to determine binary equivalent of an       */
```

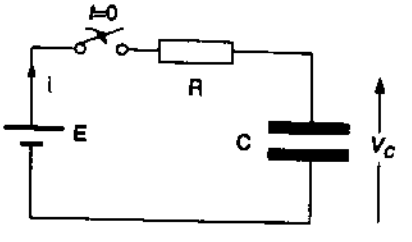
```
/* 8-bit integer                                          */
int     main (void)
{
int     val, bit;
  printf("Enter value >>");
  scanf("%d",&val);

  printf("Binary value is ");
  for (bit=0x80;bit>0;bit>>=1)
  {
  if (bit & val)     printf("1");
  else               printf("0");
  }
  return(0);
}
```

Chạy thử 4.2

```
Enter value >> 21
Binary value is
```

Quá trình quá độ của mạch RC

Hình 4.3 minh họa một mạch RC với một bước nhảy điện áp đặt vào ở thời điểm t = 0. Khi một bước nhảy điện áp với biên độ V vôn được đặt vào mạch này sẽ gây ra một dòng điện có dạng hàm mũ.



Hình 4.3: **Mạch RC với điện áp nhảy bậc đặt ở lối vào tại t=0.**

Biểu thức dưới đây cho phép xác định dòng điện tức thời trong mạch:

$$i = \frac{E}{R} e^{-\frac{t}{RC}}$$

và điện áp sụt trên điện trở sẽ bằng:

$$V_R = E.e^{-\frac{t}{RC}}$$

Chương trình 4.3 xác định điện áp sụt trên điện trở ở những khoảng thời gian cho trước. Người dùng nhập vào thời gian kết thúc và số các khoảng thời gian cần có; chương trình xác định điện áp ở mỗi bước thời gian. Nó sử dụng hàm số mũ (exp()) đã được đặt làm mẫu (prototype) trong tệp *math.h*.

Chương trình 4.3

```
/* prog4_3.c                                          */
/* Program to determine transient response     */
/* of an RC circuit                                   */
#include <math.h>
                    /* required for exp()       */
#include <stdio.h>
int      main(void)
{
float    R,C,tend,t,E,Vr;
int      tsteps;

  puts("Program to determine voltage across");
  puts("Resistor in an RC circuit");

  printf("Enter R,,C >> ");
  scanf("%f %f",&R,&C);

  printf("Enter number of time steps and end time>>");
  scanf("%d %f", &tsteps, &tend);
             /* enter integer and float*/

  printf("Enter voltage step applied>>");
  scanf("%f",&E);
```

```
    puts("    TIME      VOLTAGE");

    for     (t=0;t<tend;t+=tend/tsteps)
    {
    Vr=E*exp(-t/(R*C));
    printf("%8.4f %8.2f\n",t,Vr);
    }
    return(0);
}
```

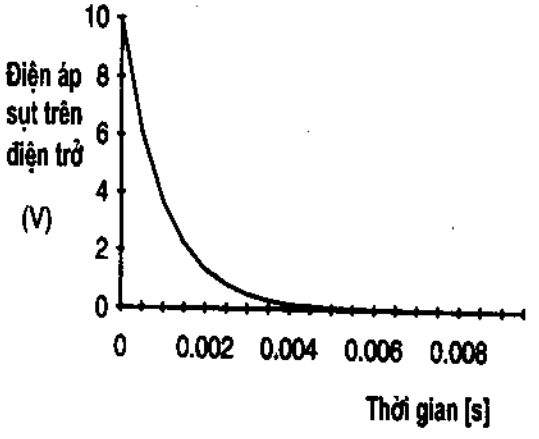
Kết quả chạy thử 4.3 cho thấy là điện áp sụt trên điện trở bắt đầu từ một cực đại tại thời điểm t = 0. Nguyên do là điện áp trên tụ điện thoạt đầu bằng 0. Khi tụ được nạp điện, điện áp trên tụ sẽ tăng lên cho đến khi nó gần như bằng với điện áp đặt vào. Dòng điện trong mạch cũng đồng thời cực đại khi bước nhảy điện áp được đặt vào. Sau đó điện áp sẽ giảm dần tới giá trị gần như bằng 0 với tốc độ được xác định bởi hằng số thời gian, bằng tích số giữa R và C.

Chạy thử 4.3

```
Program to determine voltage across
Resistor in an RC circuit
Enter R,C >> 1e3 1e-6
Enter number of time steps and end time >> 20 10e-3
Enter voltage step applied >> 10
       TIME     VOLTAGE
       0.0000   10.00
       0.0005   6.07
       0.0010   3.68
       0.0015   2.23
       0.0020   1.35
       0.0025   0.82
       0.0030   0.50
       0.0035   0.30
       0.0040   0.18
       0.0045   0.11
       0.0050   0.07
       0.0055   0.04
```

```
0.0060   0.02
0.0065   0.02
0.0070   0.01
0.0075   0.01
0.0080   0.00
0.0085   0.00
0.0090   0.00
0.0095   0.00
```

Hình 4.4 là một đồ thị được vẽ ra từ những kết quả này (được tạo ra bằng một bảng trong máy tính).



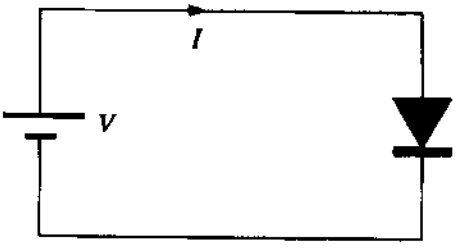
Hình 4.4: **Đáp ứng tức thời của mạch RC với điện áp ở lối vào tại thời điểm t = 0.**

Để sửa đổi chương trình sao cho có thể hiển thị điện áp sụt cả trên điện trở lẫn trên tụ điện, ta bổ sung những dòng sau đây vào chương trình 4.3.

```
for (t=0;t<tend;t+=tend/tsteps)
{
   vr=V*exp(-t/(R-C)),
   vc=V-vr
   printf("%8.4f %8.2f %8.2f\n",t,vr,vc);
}
```

Dòng đi qua một điốt

Dòng điện qua một điốt phụ thuộc vào điện áp đặt lên điốt và nhiệt độ môi trường. Một mạch bao gồm một điôt và nguồn điện áp đặt vào được mô tả trên hình 4.5. Nên nhớ rằng không bao giờ được nối trực tiếp một điốt vào nguồn điện áp mà cần có một điện trở nối tiếp để bảo vệ cho điốt và như vậy hình vẽ trên chỉ mang tính tượng trưng.



Hình 4.5: **Điốt với một nguồn điện áp đặt vào.**

Dòng điện đi qua điốt được xác định bằng phương trình sau:

$$I = I_o(e^{\frac{11600.V}{T}} - 1)$$

Chương trình 4.4 cho phép tính dòng điện đi qua một điốt ở nhiệt độ phòng (ngầm hiểu là 27°C hoặc 300K) khi nhập vào giá trị điện áp đặt vào (V) và dòng ngược bão hoà (I_O).

Chương trình 4.4

```
/* prog4_4.c                                                    */
/* Program to determine current flowing in a diode  */
#include <math.h>
#include <stdio.h>

#define TEMPERATURE 300 /*Room temperature in Kelvin*/
#define MICRO           1e-6

int  main (void)
{
float v,vend,Io,I;
int  tsteps;
```

```
    puts("Program to determine current flowing");
    puts("in a diode for an enter applied voltage");

    printf("Enter end voltage and number of time steps>>");
    scanf("%f %d",&Vmd,&tsteps);
    printf("Enter reverse saturation current>>");
    scanf("%f",&10);

    puts("VOLTAGE   CURRENT (uA)");

    for (v=0;v<Vend;v+=Vend/tsteps)
    {
          I=Io.exp (11600*v/TEMPERATURE-1);
          printf("%8.2f %12.2f\n",v,I/MICRO);
                    /* ^ display current in uA */
    }
    return(0);
}
```

Chạy thử 4.4 chỉ ra rằng trong một phạm vi rộng của dòng điện (giữa 1 μA và 1A) điện áp sụt trên điốt nằm trong khoảng từ 0,4 đến 0,7 vôn. Điện áp sụt trên điốt ở trạng thái dẫn nằm trong giới hạn từ 0,6 đến 0,7 V. Trong thí dụ chạy thử ta thấy rằng nếu điện áp đặt vào bằng 0,76 V thì dòng điện sẽ vượt quá 2 A.

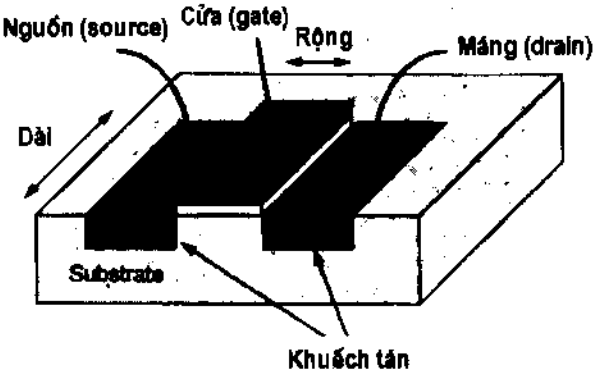
Chạy thử 4.4

```
Program to determine current flowing
in a diode for a give applied voltage
Enter end voltage and number of time steps>> 0.8 20
Enter reverse saturation current , 1e-12
ENTER điện áp cuối và số các khoảng thời gian >> 0.8 20}
ENTER dòng bão hoà ngược, 1e-12  }
 VOLTAGE   CURRENT (uA)
 0.00              0.00
 0.04              0.00
```

```
0.08            0.00
0.12            0.00
0.16            0.00
0.20            0.00
0.24            0.00
0.28            0.02
0.32            0.09
0.36            0.41
0.40            1.92
0.44            9.01
0.48           42.29
0.52          198.57
0.56          932.46
0.60         4378.63
0.64        20561.13
0.68        96550.87
0.78       453383.15
0.76      2128994.46
```

Đặc trưng Von-Ampe của một tranzito MOS

Khi dựa trên loại độ dẫn của vật liệu xuất phát để chế tạo ra linh kiện có thể chia các linh kiện MOS ra làm hai loại chính, đó là: kênh n (NMOS) và kênh p (PMOS). Hình 4.6 mô tả một tranzito MOS với ba cực ra: *cực cửa*, *cực máng* và *cực nguồn*. Chiều rộng và chiều dài của cửa xác định các thông số đặc trưng của linh kiện.



Hình 4.6: **Cấu trúc của một tranzito MOS.**

Tranzito loại NMOS được chế tạo trên một đế có độ dẫn loại p với mức độ pha tạp vừa phải. Vùng cực nguồn và cực máng được hình thành nhờ công nghệ khuếch tán để đưa vào các tạp chất loại n và mở rộng vùng nghèo điện tích không gian chủ yếu về phía miền p được pha tạp ít hơn.

Tranzito MOS là loại linh kiện hoạt động theo kiểu hỗ dẫn (transconductance), trong đó sự thay đổi của điện áp giữa cực cổng và cực nguồn sẽ gây ra sự thay đổi dòng cực máng. Hiện tượng này xảy ra khác với ở tranzito lưỡng cực (bipolar) trong đó sự thay đổi của dòng điện phát-gốc gây ra sự thay đổi trong dòng cực phát-góp. Không có hiện tượng dẫn điện trong tranzito MOS chừng nào chưa đạt đến điện áp ngưỡng, cụ thể là cho đến khi điện áp cửa-nguồn lớn hơn một giá trị ngưỡng cho trước. Thông thường giá trị ngưỡng khoảng 1 V khi điện áp nguồn cung cấp bằng 5 V. Các phương trình đối với dòng cực máng-nguồn I_{DS} có thể được dẫn ra như sau:

$$I_{DS} = 0 \qquad V_{GS} < V_{NG}$$

$$I_{DS} = \frac{\beta}{2}.\left[V_{GS} - V_{NG}\right]^2 \qquad 0 < V_{GS} - V_{NG} < V_{DS} \quad \text{(bão hoà)}$$

$$I_{DS} = \beta.\left[V_{GS} - V_{NG} - \frac{V_{DS}}{2}\right].V_{DS} \qquad V_{GS} - V_{NG} > V_{DS} \quad \text{(tuyến tính)}$$

ở đây:
$$\beta = \frac{\mu.\varepsilon}{T_{OX}}.\frac{W}{L}$$

μ độ linh động trung bình của hạt tải điện (cụ thể là điện tử đối với kênh n, và lỗ trống đối với kênh p);
ε hằng số điện môi của lớp ôxyt;
T_{ox} bề dày của lớp ôxyt (lớp giữa cực cửa và đế);
V_{GS} điện áp giữa các cực cửa-nguồn;
V_{DS} điện áp giữa các cực máng-nguồn.

Thông thường bề dày lớp ôxýt khoảng 700 ăngstrôm (70 nm) và độ linh động điện tử cỡ 8 x10^{-2} $m^2.V^{-1}.s^{-1}$.

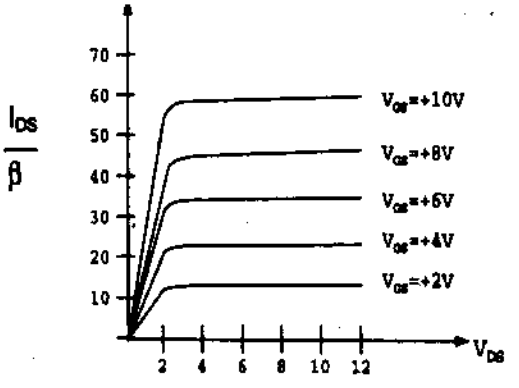
Chương trình 4.5 xác định các đặc trưng Von-Ampe của một tranzito hiệu ứng trường (FET). Trong chương trình sử dụng các vòng liên kết để thay đổi V_{GS} và V_{DS}, trong đó V_{GS} điều khiển vòng bên ngoài và V_{DS} điều khiển vòng phía trong. Thoạt đầu, lúc khởi tạo, V_{GS} được đặt bằng V_{NG}, vòng phía trong bắt đầu với điều kiện khởi tạo cụ thể là $V_{DS} = 0$, sau đấy cứ mỗi lần chạy xong một vòng lặp bên trong V_{DS} lại được tăng thêm 0,5 V. Sau khi các vòng phía trong đã thực hiện xong chương trình sẽ trở lại với vòng bên ngoài, và V_{GS} sẽ tăng từng 1 V một. Quá trình sẽ tiếp tục cho đến khi V_{GS} đạt giá trị bằng với điện áp nguồn nuôi.

Chương trình 4.5

```
/* prog4_5.c                                                    */
/* Program which will determine the drain                       */
/* current  in a FET with an entered threshold                  */
/* voltage, beta and supply voltage                             */
#include <stdio.h>

int     main(void)
{
float Id,Vgs,Vth,VDD,Vds,beta;
  /* Id is the drain current,Vgs the gate-source voltage */
  /* Vth is the threshold voltage of the FET, VDD the supply  */
  /* voltage and beta a constant for the FET                  */
printf("Enter Supply voltage and threshold>>");
scanf("%f %f",&VDD,&Vth);
printf("Enter beta (AIV^2)>>");
scanf("%f",&beta);

    for (Vgs=Vth;Vgs<=VDD;Vgs++)
         /* gate-source voltage incremented in IV steps */
    {
    printf("Vgs=%6.2f\n",Vgs);
    for (Vds=0;Vds<(Vgs-Vth);Vds,=0.5)
          /* drain-source voltage incremented in 0.5V steps */
    {
```

```
            id=beta.Vds-(Vgs-Vth-0.5*Vds);
            printf("%6.2f %6.2f\n",Vds,Id);
        }
        for (Vds=Vgs-Vth;Vds<VDD;Vds+=0.5)
            Id=0.5 * beta *(Vgs-Vth)*(Vgs-Vth);
            printf("%6.2f %6.2f\n",Vds,Id);
        printf("Press <RETURN> to continue");
        getchar();
    }
    return(0);
}
```

Kết quả chạy thử của chương trình với những tham số V_{DD} = 12 V, β = 1 (để tạo ra một đáp ứng bình thường) và V_{NG} = 1 V, được minh họa trên hình 4.7. Đồ thị mô tả sự phụ thuộc của dòng máng-nguồn (I_{DS}) vào điện áp máng-nguồn (V_{DS}) với các giá trị điện áp cửa-nguồn (V_{GS}) khác nhau. Cực nguồn được giả thiết là luôn nối với đất và tất cả điện áp khác được tính so với điểm này.



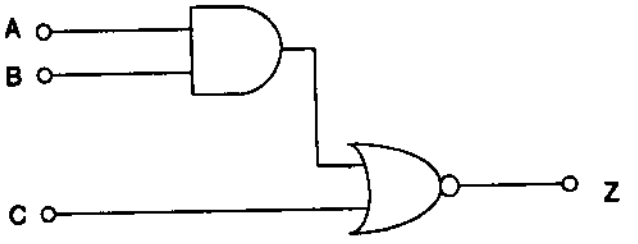
Hình 4.7: **Họ đặc trưng I_{DS} - V_{DS} của một tranzito MOS kênh n.**

Lôgic Boole

Chương trình 4.6 là một thí dụ cho thấy một hàm lôgic Boole có thể được phân tích ra sao và một bảng giá trị chân lý được tạo ra như thế nào. Vòng lặp `for` tạo ra tất cả các phép hoán vị nhị phân cần có đối với một bảng chân lý. Hàm Boole được sử dụng là:

$$Z = \overline{(A.B) + C}$$

Mạch lôgic thoả mãn phương trình này được mô tả trên hình 4.8.



Hình 4.8: **Một mạch digital.**

Chương trình 4.6

```
/* prog4_6.c                                                    */
/* Program to generate truth table for boolean function */
#include <stdio.h>

int main (void)
{
int A, B, C, Z;
   puts("Boolean function NOR (AND(A,B),C)");
   puts("   A    B     C    Z");
   for (A=0;A<=1;A++)
         for (B=0;B<=1;B++)
               for (C=0;C<=1;C++)
               {
                  Z=!( (A && B) || C);
                  printf("%4d %4d %4d %4d\n",A,B,C,Z);
               }
   return(0);
}
```

Chạy thử 4.5 chỉ ra cho một thí dụ làm mẫu.

Chạy thử 4.5

```
Boolean function NOR (AND(A,B),C)
A   B  C  z
0 0   0  1
0 0   1  0
0 1   0  1
0 1   1  0
1 0   0  1
1 0   1  0
1 1   0  0
```

Chương trình 4.7 thay thế các số nguyên 0 và 1 bằng macro SAI (FALSE) và ĐÚNG (TRUE), làm cho chương trình dễ đọc hơn.

Chương trình 4.7

```
/* prog4_7.c                                          */
/* Program to generate truth table for a              */
/* boolean function                                   */
#define FALSE 0
#define TRUE        1

int     main(void)
{
int A, B, C, Z;
  puts("Boolean function NOR (AND(A,B),C)");
  puts("A B C Z");
  for(A=FALSE;A<=TRUE;A++)
    for (B=FALSE:B<=TRUE;B++);
      for (C=FALSE;C<=TRUE;C++)
      {
        Z= ! ( (A && B) || C) ;
        printf("%4d %4d %4d %4d\n",A,B,C,Z);
      }
   return=(0) ;
}
```

4.2 LỆNH while ( )

Lệnh lặp `while` (chừng nào mà) cho phép một khối mã được thực hiện trong khi điều kiện được chỉ định là đúng. Lệnh này kiểm tra điều kiện ở chỗ bắt đầu của khối; nếu điều kiện này là TRUE (thoả mãn) thì khối mã được thực hiện, nếu không nó sẽ thoát ra khỏi vòng. Cú pháp của lệnh là:

```
while (condition)
{
   :     :
statement block
   :     :
}
```

Nếu khối lệnh chứa đựng một lệnh đơn thì dấu ngoặc nhọn có thể được bỏ đi (mặc dù cũng không gây thiệt hại gì nếu như ta giữ lại).

Các thí dụ:

(a) `while (i>10)` - sẽ lặp lại khối lệnh chừng nào i còn lớn hơn 10.

(b) `while (letter != 'q')` - câu lệnh này sẽ lặp lại *khối lệnh* chừng nào mà `letter` còn khác (không bằng) với ký tự ' q '.

(c) `while ((index <= 10) && (value ==3))` - câu lệnh này sẽ lặp lại *khối lệnh* chừng nào mà `index` còn nhỏ hơn hoặc bằng 10 và `value` (giá trị) bằng 3.

4.3 LỆNH do .. while ( )

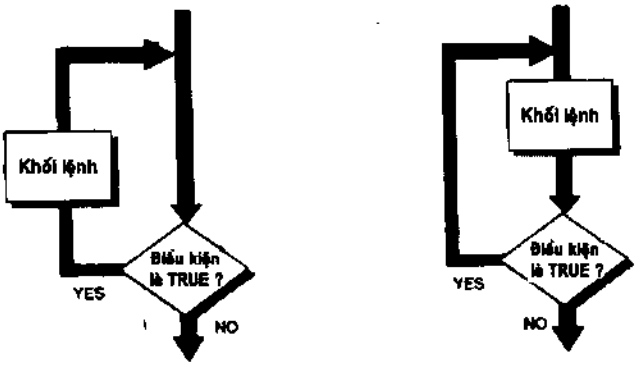
Lệnh `do..while()` tác động giống với lệnh `while()`, nếu bỏ qua một điểm khác là nó kiểm tra điều kiện ở phía dưới của vòng lặp. Lệnh này cho phép *khối lệnh* phải được chấp hành ít nhất một lần. Cú pháp của lệnh như sau:

```
do
{
   statement block
} while (condition);
```

Giống như với vòng lặp `for()` và `while()` dấu ngoặc nhọn là tùy chọn. Vòng lặp `do...while` đòi hỏi có một dấu chấm phẩy ở chỗ kết thúc của vòng, trong khi vòng lặp `while()` thì lại không cần.

Hình 4.9 mô tả tiến trình (lưu đồ) của các vòng lặp `do...while()` và `while()`. Trong cả hai vòng lặp một điều kiện ĐÚNG (TRUE) sẽ làm cho khối lệnh được lặp lại.

```
while (condition) do
{       {
        statement block statement block
}
        } while (conditions);
```



Hình 4.9: **Các vòng lặp do...while() và while().**

4.4 LỆNH NGỪNG break

Lệnh BREAK được sử dụng để thoát ra khỏi một vòng lặp. Nó có thể được sử dụng với các vòng `for()`, `while()` và `do...while()`. Một vài thí dụ được cho trong các chương trình 4.8, 4.9 và 4.10. Các vòng lặp sử dụng trong những chương trình này được mô tả như các vòng lặp vô tận `(while(1), do...while (TRUE)` và `for(; ;))` và chỉ có lệnh BREAK là có thể làm cho chương trình ngừng lại.

Chương trình 4.8

```
/* prog4_8.c                                          */
/* Program to determine resistance given              */
/* an applied voltage and current                     */
#include <stdio.h>
#include <ctype.h> /* required for tolower() function */

#define TRUE  1
int  main(void)
{
float voltage,current,resistance;
char ch;

   puts("Program to determine resistance given");
   puts("an applied voltage and current");

   do
   {
      printf("Enter a voltage and current >>");
      scanf("%f %f",&voltage,&current);

      resistance=voltage/current:
      printf("Resistance is %8.2f\n",resistance);

      printf("Do You wish to continue (y/n) >>");

      fflush(stdin);/*clear keyboard buffer */

      ch=tolower(getchar());
      if (ch=='n') break;
   } while (TRUE);

   puts("program end <BYE>");
   return(0);
}
```

Chương trình 4.9

```
/* prog4_9.c                                                    */
/* Program to determine reactance of an inductor  */
/* at an applied frequency                                      */

#include <stdio.h>
#include ctype.h>
                    /* required for tolower() function  */

#define PI    3.14159
#define TRUE  1

int    main (void)
{
float X_1,inductance,freq;

   puts("Program to determine reactance of an inductor");
 puts("at an applied frequency") ;

 while (TRUE)
 {
   puts("Enter an inductance and frequency (enter ");
   printf("a zero for any of the parameters to exit)>>");
   scanf("%f %f",&inductance,&freq);
   if ((iductance<=0) || (freq<=0)) break;
   X_1=2*PI*freq*inductance;
   printf("Reactance is %8.2f\n",X_1);
  }
  puts("Program end <BYE>") ;
  return(0);
}
```

Kết quả chạy thử 4.10 chỉ ra một thí dụ chạy mẫu với điều kiện để thoát ra.

Chạy thử 4.6

```
Program to determine reactance of an inductor
at an applied frequency
Enter an inductance and frequency (enter a
zero for any of the parameters to exit) >> 0 0
Program end <BYE>
```

Chương trình 4.10

```
/* prog4_10.c                                          */
#include <stdio.h>
int main(void)
{
int  i
  for (;;)
  {
        puts("Enter number to be cubed");
        scanf("%d",&i);
        if (i==0) break; /* stop infinite loop */
        printf ("The cube of %d  is %d\n ",i,i*i*i);
  }
  puts("Zero entered");
  return(0);
}
```

4.5 LỆNH TIẾP TỤC continue

Lệnh tiếp tục (continue) chỉ có thể được sử dụng ở bên trong một lệnh lặp. Lệnh này chuyển trạng thái điều khiển sang kiểm tra điều kiện đối với các vòng while() và do...while() và tới biểu thức (expression) trong một vòng for(). Mã dùng làm thí dụ sau đây sẽ xuất ra các giá trị của i từ 0 đến 9 nhưng sẽ không in ra 5 (nghĩa là 0, 1, 2, 3, 4, 6, 7, 8 và 9).

```
for (i=0;i<10;i++)
{
```

```
    if (i==5) continue;
    printf("Value of i is %d",i);
}
```

4.6 MỘT SỐ THÍ DỤ

4.6.1 PHƯƠNG TRÌNH BOOLE

Chương trình 4.11 sử dụng phương trình đã được xác định trong chương trình 4.6. Chương trình này cho phép người dùng nhập vào các trạng thái đầu vào cho A, B và C và chương trình sẽ xác định đầu ra cho các lối vào này. Nó sử dụng lệnh `while()` để ngăn việc nhập vào các giá trị nguyên, sai (thí dụ giá trị bất kỳ mà không phải là 0 hoặc 1).

Chương trình 4.11

```
/* prog4_11.c                                          */
/* Program which determines the output of the  */
/* boole equation NOR(C,AND(A.B))                      */
int  main (void)
{
int  A,B,C,Z;
   do
   {
      printf("Enter A input (0 or 1) >>");
      scanf("%d",&A);
      if ( (A!=0) && (A!=1) ) puts("Invalid input");
   } while ( (A!=0) && (A!=1)) ;
   do
   {
      printf("Enter B input (0 or 1) >>");
      scanf("%d",&B),
      if ( (B!=0) && (B!=1) ) puts("Invalid input");
   } while ( (B!=0) && (B!=1)) ;
   do
   {
      printf("Enter C input (0 or 1) >>");
      scanf("%d",&C);
```

```
        if ( (C!=0) && (C!=1) ) puts("invalid input");
    } while ( (C!=0) && (C!=1));
    Z=!(A && B) || C);
    printf("Output will be %d\n",Z);
    return(0);
}
```

Kết quả chạy thử 4.7 chỉ ra cho một thí dụ chạy mẫu.

Chạy thử 4.7

```
Enter A input(0 or 1) >> 21
Invalid input
Enter A input(0 or 1) >> 0
Enter B input(0 or 1) >> -1
Invalid input
Enter B input  (0 or 1) >> 1
Enter C input(0 or 1) >> 1
output will be 0
```

Chương trình 4.11 có thể vẫn còn cho phép nhập vào các giá trị không hợp lệ bởi vì `scanf()` cho phép nhập vào các kiểu dữ liệu khác với khuôn mẫu đã được chỉ định. Trong trường hợp này giá trị được nhập vào phải là một số nguyên, nhưng `scanf()` sẽ tiếp nhận các kiểu dữ liệu khác chẳng hạn như các giá trị dấu phẩy động, các ký tự, v. v... Thí dụ, nếu người dùng nhập một giá trị dấu phẩy động vào trong `scanf()` (chẳng hạn 1,0) thì giá trị này có thể không được dịch như một số nguyên hợp lệ. Tham số được quét cuối cùng không có khả năng dự đoán trước.

Hàm `scanf()` trả lại một giá trị liên quan đến số của các trường được quét hoàn chỉnh. Chẳng hạn, nếu trả lại là 0 thì không có các trường được quét. Mỗi trường được phân định bởi một dấu trống. Đoạn mã dưới đây sử dụng giá trị trả lại để xác định nếu số của những đầu vào không hợp lệ là 1. Nếu như số này hơn 1 hoặc 0 thì người dùng sẽ được nhắc lại để nhập vào giá trị hợp lệ. Biến `okay` được thiết lập là FALSE (Sai) nếu giá trị nhập vào là không hợp lệ. Biến này được kiểm

tra ở chỗ kết thúc vòng, nếu là FALSE (Sai) thì vòng lặp sẽ tiếp tục và người dùng lại được nhắc để nhập vào một giá trị.

```
int okay
    do
    {
          printf("Enter A input (0 or 1) >>");
          rtn=scanf("%d",&A);
          if ((rtn!=1) || ( (A!=0) && (A!=1)))
          {
               puts("invalid input");
               okay=FALSE;
          }
          else okay=TRUE;
    } while ( !okay )
```

Chương trình 4.12 sử dụng một vòng `do...while()` ở phía ngoài để lặp lại chương trình. Người dùng được nhắc xem liệu chương trình sẽ tiếp tục hay không. Nếu như ký tự `'y'` được nhập vào thì chương trình sẽ lặp lại, bằng không nó sẽ kết thúc. Vấn đề có thể xuất hiện khi `getchar()` được sử dụng sau `scanf()` bởi vì các ký tự dòng mới có thể được lưu lại trong bộ đệm bàn phím. Vì lý do này bộ đệm bàn phím được làm rỗng bằng cách sử dụng `fflush(stdin)`.

Chương trình 4.12

```
/* prog4_12.c                                          */
/* Program that detemines the output for the          */
/* boole equation, NOR(C,AND(A,B))                    */
#include <stdio.h>
#include <ctype.h>
#define TRUE  1
#define FALSE 0
int    main (void)
{
int    A,B,C,Z,okay,rtn;
char   ch;
  do
```

```
{
    do
    {
        printf("Enter A input (0 or 1) >>");
        rtn=scanf('%d',&A);
        if ((rtn!=1) ||( (A!=0) && (A!=1)))
        {
           puts("invalid input");
           okay=FALSE;
        }
        else okay=TRUE;
    } while ( !okay );
    do
    {
        printf("Enter B input (0 or 1) >>");
        rtn=scanf("%d",&B);
        if ((rtn!=1) || ( (B!=0) && (B!=1)))
        {
            puts("Invalid input");
            okay=FALSE;
        }
        else okay=TRUE;
    } while ( !okay );
    do
    {
        printf("Enter C input (0 or 1) >>");
        rtn=scanf("%d",&C);
        if ((rtn!=1) || ( (C!=0) && (C!=1)))
        {
            puts("Invalid input");
            okay=FALSE;
    }
    else okay= TRUE
    } while ( !okay );
    Z=!((A && B) || C);
    printf("Output will be %d\n",Z);
```

```
        printf("Do you wish to continue (y/n) >>");
        fflush(stdin);
           /* this will flush the keyboard buffer as */
           /* there may be characters still in it  */
        ch=getchar();
      } while (tolower(ch)=='Y');
      return(0);
}
```

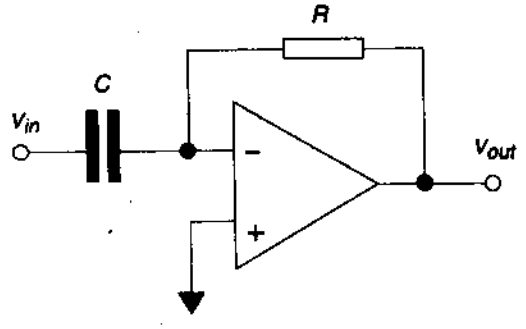
Kết quả chạy thử 4.8 chỉ ra cho một thí dụ chạy mẫu.

Chạy thử 4.8

```
Enter A input (0 or 1) >> 0
Enter B input (0 or 1) >> 0
Enter C input (0 or 1) >> 0
Output will be 1
Do you wish to continue (y/n) >> y
Enter A input (0 or 1) >> 0
Enter B input,(0 or 1) >> 0
Enter C input (0 or 1) >> 1
Output will be 0
Do you wish to continue (y/n)>> n
```

4.6.2 BỘ LỌC TÍCH CỰC RC

Một bộ lọc tích cực sử dụng một bộ khuếch đại thuật toán được minh hoạ trong hình 4.10.



Hình 4.10: **Bộ lọc tích cực dùng RC.**

Mạch này có hệ số khuếch đại nhỏ ở vùng tần số thấp và có hệ số khuếch đại lớn ở vùng tần số cao; bởi vậy nó hoạt động giống như một bộ lọc dải thông tần số cao.

Hệ số khuếch đại của mạch này được cho bởi:

$$|\text{HSKĐ}| = 2\pi.fRC = 20\log_{10}(2\pi.fRC) \quad [\text{dB}]$$

Chương trình 4.13 tính hệ số khuếch đại của mạch này ra đêxiben. Các giá trị thícb hợp để nhập vào đối với các điện trở nằm giữa 1 kΩ và 1MΩ và những giá trị điện dung thích hợp là một giá trị bất kỳ nhưng lớn hơn hoặc bằng với 1 μF. Vòng `do..while()` được đặt khi nhập các giá trị này để người dùng sẽ được nhắc nhở nếu như các giá trị không hợp lệ đã được nhập vào.

Hàm để xác định $\log_{10}$ được thực hiện trong thư viện toán học (math) chuẩn, hàm này có tên `log10()`. Để tính hàm này khi sử dụng hàm lôgarit tự nhiên (`log()`) có thể sử dụng hệ thức chuyển đổi sau:

$$\log_{10}(x) = \frac{\log(x)}{\log(10)}$$

Chương trình 4.13

```
/* prog4_13.c                                          */
/* Program to determine the gain of C and R            */
/* active filter for entered values of C and R         */
/* and a frequency span from 1Hz up to 1 GHz in        */
/* decade steps                                         */

#include <stdio.h>
#include <math.h>
#define TRUE     1
#define FALSE    0
#define   MILLI  1e-3
#define   KILO   1e3
#define   MEGA   1e6
#define   GIGA   1e9
#define   PI     3.14159
```

```
int      main(void)
{
float    resistance,capacitance,freq,gain,gain_dB;
int      rtn,okay;

     puts("Program to determine gain of a CR active filter");
     do
     {
        printf("Enter capacitance >>");
        rtn=scanf("%f",&capacitance);
        if ((rtn!=1) || (capacitance>MILLI))
        {
             puts("INVALID: re-enter");
             okay=FALSE;
        }
        else okay--MUE;
     } while (!okay );
     do
     {
        printf ("Enter resistance >>") ;
        rtn=scanf("%f",resistance) ;
        if ((rtn!=1)||((resistance>=MEGA)||(resistance<KILO)));
        {
            puts("INVALID: re-enter");
            okay=FALSE;
        }
        else okay=TRUE;
     } while (!okay);

     puts("FREQUENCY(HZ) GAIN(dB)");

     for (freq=1;freq<=GIGA;freq*=10)
     {   /* decade steps of frequency */ .
        gain=2*PI*freq*resistance*capacitance;
```

```
            gain_dB=20*log10(gain);
            printf("%10.1e %10.3f\n",freq,gain_dB);
        }
        return(0);
}
```

Kết quả chạy thử 4.9 chỉ ra một thí dụ chạy mẫu.

Chạy thử 4.9

```
Program to determine gain of CR active filter
Enter capacitance >> LOG
INVALID: re-enter
Enter capacitance >> 1e-6
Enter resistance >> -43
INVALID: re-enter
Enter resistance >> 1e4
FREQUENCY(Hz) GAIN(dB)
1e+00-24.036
1e+01 -4.036
1e+02 15.964
1e+03 35.964
1e+04 55.964
1e+05 75.964
1e+06 95.964
1e+07115.964
1e+08135.964
     1e+09  155.964
```

4.7 THỰC HÀNH

Q4.1 Xác định lỗi trong các chương trình sau:

(a)

Chương trình 4.14

```
/* prog4_14.c
```

```
/* Prints the square of the numbers 1 to 10     */
/* ie 1,4,9..100                                */
#include <stdio.h>

int main(void)
{
int i;
     for (i=1,i<10,i++)
          printf("The square of i is %d",i*i);
     return (0);
}
```

(b)

Chương trình 4.15

```
/* prog4_15.c                                   */
/* Prints values from 1 to 100 in power of 3    */
/* The step used is 0.3                         */
#include <stdio.h>

int   main(void)
{
int   i;

   while (i != 100)
   {
        printf("%d to the power of three is %d/n",i*i*i);
   i += 0.3;
   return (0);
}
```

(c)

Chương trình 4.16

```
/* prog4-16.c                                   */
/* Prints the square of the numbers 1 to 10     */
```

```
/* ie 1, 4, 9..100                                    */
#include <studio.h>

int     main(void)
{
int     i;
  for (i=1;i<=10;i++);
        printf("The square of i is %d",i*i);
  return(0);
}
```

(d)

Chương trình 4.17

```
/*  prog4_17.c                                         */
/*  Program to determine input resistance given the    */
/*  input voltage and current                          */
#include <stdio.h>
int   main(void)
{
char   input;
   puts("Program to determine the resistance given");
   puts("input voltage and current");

   while (input == 'Y')
   {
        printf("Enter voltage and current >>");
        scanf("%f %f",voltage,current);
        printf("The resistance is %f",voltage/current);
        puts("Do you wish to continue (y/n)")
        input = getchar();
   }
   return(0);
}
```

(e)

Chương trình 4.18

```
/* prog4_18.c                                    */
#include <stdio.h>
int  main(void)
{
  puts("Program to determine the resistance given");
  puts("input voltage and current");

  while (1)
  {
    printf("Enter voltage and current >> ");
    scanf("%f %f",voltage,current);
    printf("The resistance is %f",voltage/current);

    puts("Do you wish to continue(y/n)");
    ch=getchar();
    if ((ch=='n') && (ch=='N')) break;
  }
  puts("Program exited");
  return(0);
}
```

Q4.2 Xác định đầu ra từ những đoạn mã dưới đây (chú ý vòng lặp vô tận).

(a)

```
for (i=0;i<10;i+=2)
  printf("%d ",i);
```

(b)

```
for (i=1;i<120;i+=2)
  printf("%d ",i);
```

(c)

```
for (i=19;i<12;i--)
  printf("%d ",i);
```

(d)

```
i=6;
```

```
do
{
     i++;
     printf("%d",i)
} while (i<10);
```

(e)

```
i=1;
while (1)
{
     i++;
     printf("%d",i)
   if (i==4) break ;
}
```

(f)

```
i=2;
for (;;)
{
     i++;
     printf("%d",i)
     if (i==256) break;
}
```

(g)

```
i=10
while (i<10)
{
   i--;
   printf("%d ",i);
}
```

(h)

```
i=10;
do
{
   i--;
```

```
        printf("%d",i)
    } while (i<10);
```

(i)

```
i=1;
while (i>0)
{
   i++;
   printf("1%d ",i);
}
```

Q4.3 Xác định bảng chân lý cho những phương trình sau:

$$Z = \overline{\overline{A + B} + A\overline{B}}$$

$$Z = \overline{\overline{A + B + \overline{C} + B\overline{C}D} + A}$$

$$Z = \overline{\overline{A + B + D + C.D} + \overline{A}}$$

So sánh với chương trình 4.7.

Q4.4 Hãy viết một chương trình xác định điện áp sụt trên một cuộn cảm và một điện trở trên một mạch RL nối tiếp với một bước nhảy điện áp ở đầu vào được đặt tại thời điểm t = 0. Người dùng sẽ nhập vào các giá trị của các linh kiện (phần tử), thời gian kết thúc và số các khoảng thời gian. Giới hạn thích hợp đối với các giá trị được nhập vào được liệt kê trong bảng dưới đây. So sánh với chương trình 4.3.

Thông số	Cực tiểu	Cực đại
R	0 Ω	1 MΩ
L	0 H	100 mH
Thời gian kết thúc	1 μs	1 s
Số khoảng thời gian	10	100

Q4.5 Hãy viết một chương trình xác định độ lớn và góc của trở kháng của mạch RL nối tiếp. Chương trình phải tính độ lớn và góc của trở kháng và sẽ không cho phép nhận các giá trị nhập vào không

hợp lệ. Giới hạn thích hợp đối với các giá trị được nhập vào được dẫn ra trong bảng dưới đây. So sánh với chương trình 2.14.

Thông số	Cực tiểu	Cực đại
R	0 Ω	1 MΩ
L	0 H	100 mH
f	0 Hz	10 GHz

Q4.6 Hãy sửa đổi chương trình trong Q4.5 để người dùng có thể nhập vào một giá trị tần số bắt đầu và kết thúc và số các điểm tần số. Chương trình sẽ xác định độ lớn và góc của trở kháng đối với khoảng tần số được yêu cầu. So sánh với chương trình 4.11.

Q4.7 Hãy sửa đổi chương trình trong Q4.6 để cho ra một trong các kết quả ở một thời điểm và sau đó nhắc người dùng nhấn phím <ENTER> để tiếp tục chương trình. Chẳng hạn, 10 giá trị của bảng các kết quả có thể được in và chương trình dừng cho đến khi phím <ENTER> được nhấn.

Q4,8 Hãy sửa đổi chương trình trong Q4.6 để dùng cho một mạch RC nối tiếp.

Q4.9 Dòng cực máng, tính ra mili ampe, của một tranzito trường (FET) kiểu lớp nghèo điện tích được cho bởi:

$$I_D = 6.\left[1 + \frac{V_{GS}}{4}\right]^2 \quad \text{[mA]}$$

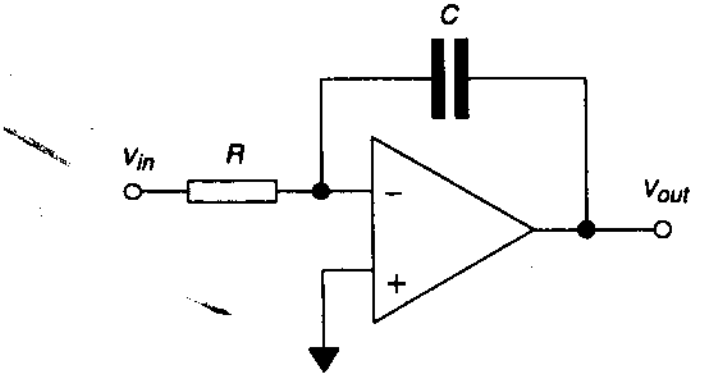
Hãy viết một chương trình xác định được dòng máng (ra mA) đối với điện áp cửa-nguồn (V_{GS}) thay đổi từ -5 tới 5 V theo các bước chênh lệch nhau nhau 0,5 V. Một thí dụ chạy thử được mô tả trong chạy thử 4.10. Hãy tham khảo thêm chương trình 4.3.

Chạy thử 4.10

```
VGS (V)      IDS (mA)
- 5.00         0.38
- 4.50         0.09
```

- 4.00	0.00
- 3.50	0.09
- 3.00	0.38
- 2.50	0.84
- 2.00	1.50
- 1.50	2.34
- 1.00	3.38
- 0.50	4.59
0.00	6.00
0.50	7.59
1.00	9.38
1.50	11.34
2.00	13.50
2.50	15.84
3.00	18.38
3.50	21.09
4.00	24.00
4.50	27.09
5.00	30.38

Q4.10 Một bộ lọc tích cực sử dụng bộ khuếch đại thuật toán được mô tả trên hình 4.11. Mạch này có hệ số khuếch đại lớn ở tần số thấp và hệ số khuếch đại nhỏ ở vùng tần số cao; bởi vậy nó hoạt động như một lọc dải thông tần số thấp.



Hình 4.11: **Bộ lọc tích cực RC.**

Hệ số khuếch đại của mạch này được tính theo:

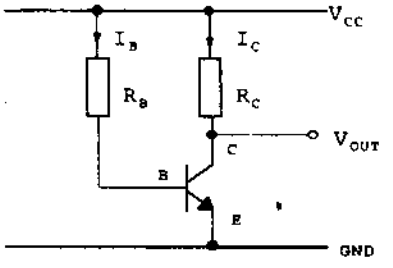
$$|\text{HSKĐ}| = \frac{1}{2\pi . fRC}$$

Hãy viết một chương trình có thể in ra một bảng tần số và hệ số khuếch đại ứng với tần số từ 0,1 Hz đến l0 GHz theo từng bước hơn kém nhau 10 lần bằng cách sử dụng

(1) Một vòng `for();`
(2) Một vòng `while();`
(3) Một vòng `do...while()`

Hãy sửa đổi các chương trình để hệ số khuếch đại được hiển thị ra đềxiben (dB). So sánh với chương trình 4.11.

Q4.11 Một mạch tranzito lưỡng cực (bipolar) thiên áp đơn giản bằng một điện trở ở cực gốc được mô tả trên hình 4.12.



Hình 4.12: **Một mạch tranzito lưỡng cực thiên áp chỉ bằng một điện trở ở cực gốc.**

Các phương trình sau đây có thể áp dụng cho thiên áp của mạch này:

$$I_C = \frac{V_{CC}/2}{h_C} \quad [\text{A}]$$

$$I_B = \frac{I_C}{h_{FE}} \quad [\text{A}]$$

$$R_B = \frac{V_{CC} - V_{BE}(ON)}{I_B} \quad [\Omega]$$

Hãy viết một chương trình tính giá trị của R_B, theo các số liệu của Ic và Vcc được nhập vào. Giả sử rằng V_{BE}(ON) là 0,65 V. Chương

trình sẽ chỉ chấp nhận những giá trị hợp lệ như một điện áp nguồn nuôi (Vcc) từ 5 đến 30 V và cực góp I_C giữa 0,1 và 10 mA. Một thông báo lỗi sẽ được hiển thị nếu như các giá trị nằm ngoài phạm vi này và chương trình nhắc lại những giá trị hợp lệ. Một đầu ra mẫu được chỉ ra trong chạy thử 4.11

Thông số	Cực tiểu	Cực đại
V_{CC}	5 V	30 V
I_C	0,1 mA	10 mA
h_{FE}	50	150

Chạy thử 4.11

```
ENTER Vcc >> 1000000
Đầu vào sai (5-> 15 V)
ENTER Vcc >> -1
Đầu vào sai (5-> ISV)
ENTER Vcc >> 15
ENTER Ic (1mA)>> 1
Điện trở gốc value XX ôm
Tương đương nhị phân là 00100001
```

Q4.12 Điện áp nhiễu RMS do một điện trở sinh ra được cho bởi:

$$V_N = \sqrt{4k.T.R.B}$$

ở đây:

k là hằng số Boltzmann (1.38×10^{-13} J/ K)
B là dải thông của tín hiệu (Hz);
R là điện trở tương đương (Ω)
T là nhiệt độ (K)

Chất lượng của một hệ thống truyền thông có thể được đo bằng cách sử dụng tỷ số tín hiệu/ nhiễu (SNR: signal noise ratio). Giá trị này được cho như là tỷ số giữa công suất tín hiệu và của nhiễu, và thường được biểu diễn bằng dB. Công thức này (các số hạng RMS là điện áp tín hiệu và nhiễu) là:

$$SNR(dB) = \log_{10} \frac{V_S}{V_N}$$

Hãy viết một chương trình hiển thị SNR (tính theo dB) cho các dải thông từ 1 Hz lên tới 10 GHz theo những bước chênh lệch nhau 10 lần. Tham khảo thêm chương trình 4.11. Giả thiết rằng:

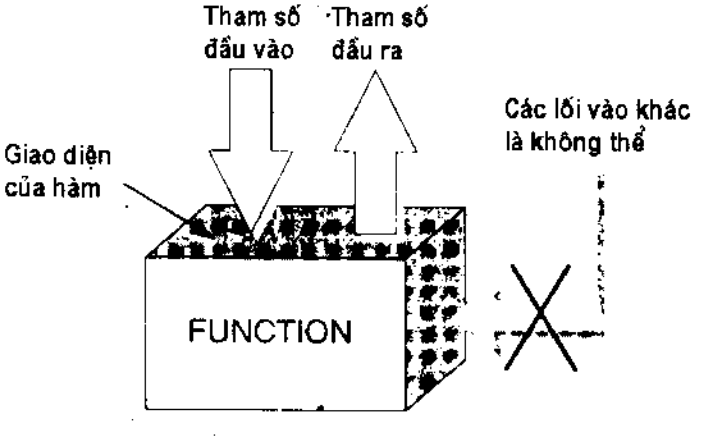
T = 27 ^{o}C;

R = 600 Ω

V_S = 0,25 mV.

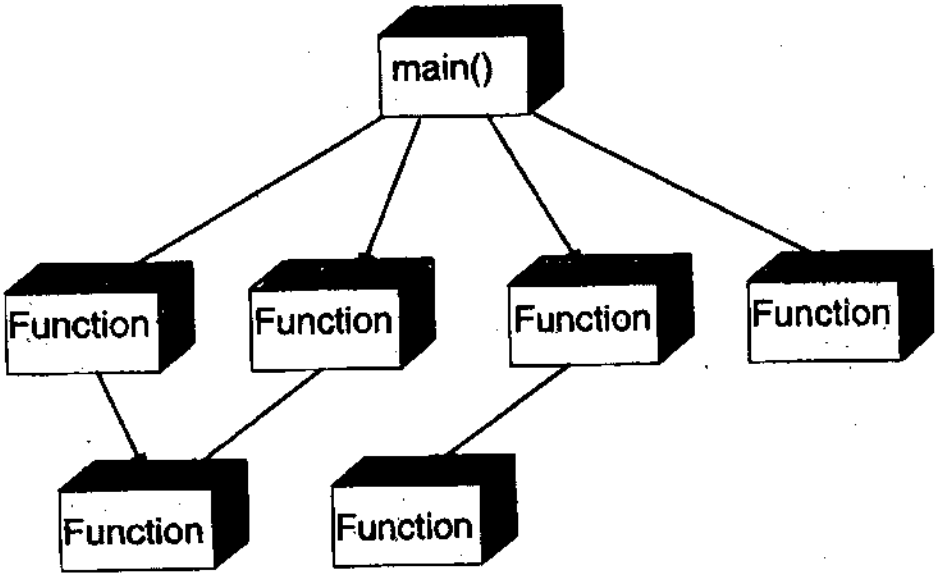
Chương 5
CÁC HÀM

Các hàm là các đoạn mã có thể nhận biết bằng một giao diện đã được định nghĩa. Các hàm được gọi từ bất kỳ chỗ nào của một chương trình và cho phép các chương trình lớn chia ra thành các phần công việc dễ quản lý hơn, mỗi một trong chúng có thể được kiểm tra một cách độc lập. Đồng thời các hàm tỏ ra là có ích trong các thư viện xây dựng các đoạn chương trình (routine) mà các chương trình khác sử dụng. Có tồn tại một vài thư viện chuẩn, như thư viện toán học và thư viện đầu vào/ đầu ra. Trong các chương trước, các chương trình đã được viết ra bằng cách sử dụng một số hàm chuẩn như `printf()`, `scanf()`, `get()`, `getchar()`, `sqrt()`, `exp()` và `log()`.



Hình 5.1: **Mô tả một hàm như một "hộp đen" lý tưởng.**

Một hàm có thể được quan niệm như một "hộp đen" với một tập hợp các đầu vào và đầu ra. Bằng chức năng của mình, nó xử lý các đầu vào theo cách "đọc chính tả" và cung cấp một vài đầu ra. Trong đa số các trường hợp, hoạt động thực tại của "hộp đen" là vô hình đối với phần còn lại của chương trình. Một chương trình cấu trúc theo kiểu môđun bao gồm một số "hộp đen" làm việc độc lập với tất cả các hộp khác, và từng cái sử dụng các biến khai báo bên trong nó (các biến cục bộ) và các tham số bất kỳ được gửi cho nó. Hình 5.1 minh họa một hàm mô tả bởi một "hộp đen" lý tưởng với các đầu vào và đầu ra, và hình 5.2 chỉ ra một hàm chính (main), gọi một vài hàm con (hoặc các môđun).



Hình 5.2: **Phân tích một chương trình theo thứ bậc.**

5.1 CHUYỂN GIAO THAM SỐ

Các kiểu dữ liệu và tên của các tham số chuyển giao vào một hàm được khai báo trong đầu mục hàm (giao diện của nó) và các giá trị hiện tại gửi được đối chiếu như các đối số. Chúng có thể nhập vào hoặc như

các giá trị (được gọi là "chuyển giao bằng giá trị") hoặc như các con trỏ (được gọi là "chuyển giao bằng đối chiếu"). Sự chuyển giao bằng giá trị kéo theo gửi một bản sao của nó vào trong hàm. Không thể thay đổi giá trị của một biến khi sử dụng phương pháp này. Các biến chỉ có thể được thay đổi nếu như chúng được chuyển giao bằng đối chiếu (điều này sẽ được bàn luận trong chương sau). Chương này xem xét các tham số được chuyển giao vào trong hàm như thế nào và một giá trị đơn được trả lại như thế nào.

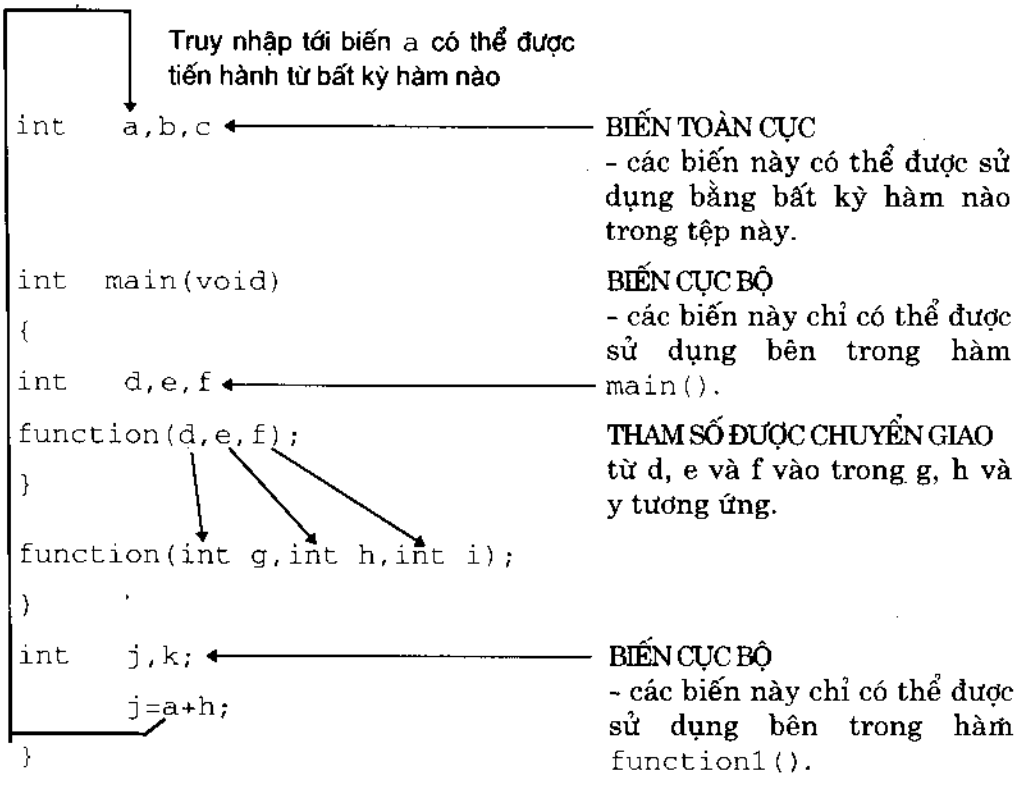
Một đối số và một tham số được định nghĩa như sau:

"Đối số" là giá trị hiện thời nhập vào hàm.
"Tham số" là biến định nghĩa trong đầu mục hàm.

Hình 5.3 minh hoạ một chương trình với hai hàm: `main()` và `function1()`. Hàm `main()` gọi hàm `function1()` và nhập ba tham số vào đó; chúng được chuyển giao vào như các giá trị. Một bản sao nội dung của `d` được chuyển vào trong `g`, `e` vào trong `h` và `f` vào trong `i`.

Các biến khai báo trong một hàm được mô tả như các biến cục bộ (local). Hình 5.3 chỉ ra rằng `d`, `e` và `f` là các biến cục bộ trong `main()`; `g`, `h`, `i`, `j` và `k` là cục bộ trong `function1()`. Các biến này sẽ không có mối liên kết với các biến cùng tên đã khai báo trong các hàm khác. Các biến cục bộ chỉ tồn tại bên trong một hàm, tại đó chúng đã được khai báo và không tồn tại mỗi khi chương trình rời bỏ hàm. Các biến đã khai báo ở đầu của tệp nguồn (và không ở bên trong hàm) đã được định nghĩa như các biến toàn cục. Các biến này cho phép các hàm, bên trong tệp tin nguồn truy nhập lên chúng. Cần thận trọng khi sử dụng các biến toàn cục vì nhiều lý do, một trong các lý do là làm cho các chương trình đó bị lộn xộn về cấu trúc và khó bảo trì.

Trong hình 5.3 hàm `function1()` chọn việc sử dụng biến `a` bởi vì biến này đã được khai báo như một biến toàn cục. Hàm này không thể được mô hình hoá như một "hộp đen" bởi vì nó có thể sửa đổi một biến khi không được chuyển giao tới nó. Trong một chương trình tương đối nhỏ khả năng này có thể không gây ra một vấn đề gì nhưng vì kích thước của chương trình bị tăng lên, thành thử việc điều khiển của các biến có thể trở nên khó khăn.

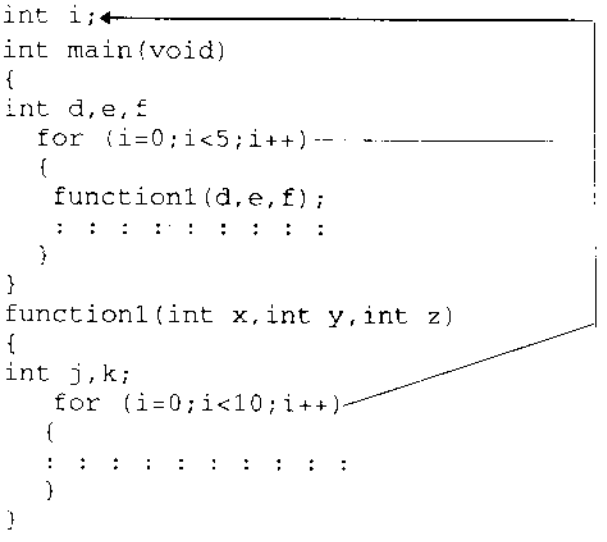


Hình 5.3: **Các biến cục bộ và toàn cục.**

Hình 5.4 minh hoạ một thí dụ trong đó các biến toàn cục có thể bị sử dụng sai như thế nào. Trong thí dụ này, một biến toàn cục `i` này được hai hàm sử dụng. Thoạt tiên, giá trị của `i` trong vòng lặp `main(0)` sẽ bằng 0. Hàm `function1()` được gọi bên trong vòng lặp, khi dùng biến `i` trong vòng lặp khác. Mỗi lần nó gia tăng thì biến toàn cục lại gia tăng giá trị. Khi chương trình rời bỏ hàm này, giá trị của `i` sẽ bị thay đổi (thí dụ nó sẽ bằng 10). Kết cục này làm lặp lại `for(0)` trong `main(0)` cho đến khi kết thúc. Nếu như biến `i` đã được khai báo là cục bộ bên trong `main(0)` và `function1()` thì vấn đề này đã không xảy ra. Thông thường các môđun được gọi sẽ được tự được chứa và sử dụng chỉ các tham số gửi tới chúng.

Chương trình 5.1 chứa một hàm có tên `print_values()`. Hàm này được gọi từ `main(0)` và các biến `a` và `b` được đưa vào trong các tham số `c` và `d` tương ứng; `c` và `d` là các tham số cục bộ và chỉ tồn tại

trong print_values(). Giá trị của c và d có thể bị thay đổi nhưng không ảnh hưởng lên các giá trị của a và b.



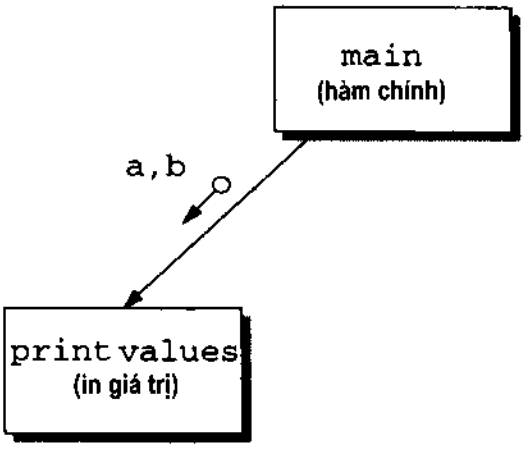
Hình 5.4: **Thí dụ về cách dùng các biến toàn cục.**

Chương trình 5.1

```
/* prog5_1.c                                              */
/* Simple program that shows parameter passing    */
#include <stdio.h>

void    print_values(int a, int b);
   /* ^ function prototype, to be discussed    */
int  main(void)
{
int  a=5,b=6;

print_values(a,b)
return(0)
}

void   print_values(int c, int d)
{
  print("The value passed are %d %d \n",c,d);
```

```
}
```

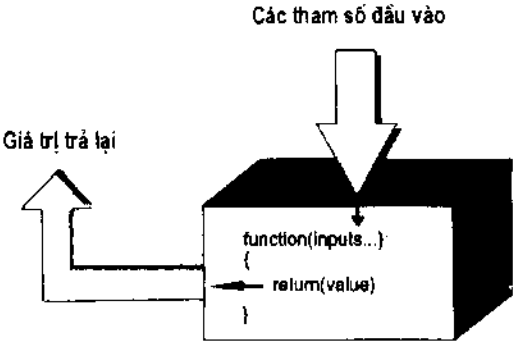
Hình 5.5 minh họa cho thấy chương trình này có thể được diễn tả bằng một giản đồ cấu trúc như thế nào. Giản đồ này là sự trình bày mức độ cao của chương trình và hiển thị sự lưu chuyển dữ liệu giữa các hàm. Trong trường hợp này, nó chỉ ra rằng `main(0)` là hàm (hoặc môđun) mức cao nhất và gọi hàm `print_values()`.



Hình 5.5: **Đồ thị cấu trúc chương trình 5.1.**

5.2 GIÁ TRỊ TRẢ LẠI

Lệnh `return` trả lại một giá trị đơn từ một hàm tới đoạn chương trình gọi như minh họa trong hình 5.5.



Hình 5.6: **Biểu diễn một hàm bằng hộp đen với giá trị trả lại.**

Các chương trình trong các chương trước đã sử dụng các hàm trả lại các giá trị. Trong số đó là các hàm toán học `sqrt()` (chương trình 2.10),

`atan()` (chương trình 2.14), `exp()` (chương trình 4.3), và `log10()` (chương trình 4.13). Tệp đầu mục *math.h* có chứa khai báo của các hàm này.

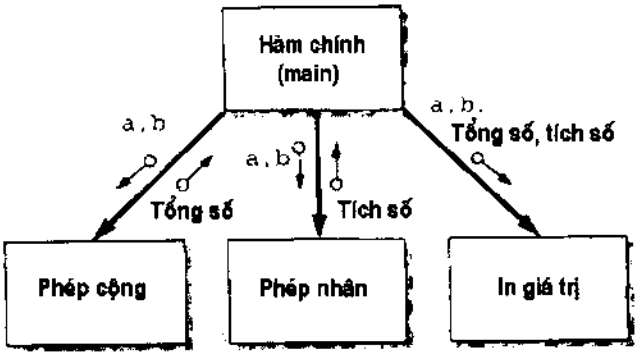
Nếu như không có lệnh `return` trong một hàm thì trong quá trình chấp hành, chương trình tự động trả lại tới đoạn chương trình gọi đang chấp hành ở dấu đóng ngoặc (nghĩa là sau lệnh cuối cùng bên trong hàm). Chương trình 5.2 chứa các hàm, sẽ cộng và nhân hai số. Hàm cộng `addition()` dùng `return` để gửi trả lại tổng hai giá trị tới `main()`.

Chương trình 5.2

```
/* Prog5_2.c                                                    */
#include <stdio.h>
/* Function prototyped, these discussed in the next section */
 int   addition(int c, int d);
 int   multiply(int c, int d);
 void  print_values(int c, int d, int sum,int mult);
 int   main(void)
 {
 int   a=5, b=6, summation, multi;

   Summation=addition(a,b)
   multi = multiply(a,b)
   print_values(a,b,summation,multi)
   return(0)
 }

 int     addition(int c,int d)
 {
   return(c+d)
 }
 int    multiply(int c,int d)
 {
   return(c*d);
 }
```

```
void     print_value(int d,int sum, int mult);
{
  print(*%d plus %d is %d \n",c,d,sum);
  print("%d multiplied %d is %d \n",c,d,mult);
}
```

Hình 5.7 mô tả một sơ đồ cấu trúc đơn giản của chương trình này. Hàm cộng `addition()` được gọi đầu tiên; các biến gửi đến là `a` và `b` và giá trị trả lại được đặt vào trong biến tổng số `summation`. Tiếp theo, hàm nhân `multiply()` được gọi; các biến được gửi đến cũng là `a` và `b` và và giá trị trả lại được đưa vào `multi`. Cuối cùng hàm `print_value()` được gọi; các giá trị gửi đến là `a` và `b`, `multi` và `summation`.



Hình 5.7: **Sơ đồ cấu trúc cơ sở dùng cho chương trình 5.2.**

Một hàm có thể có vài điểm trả lại, mặc dù bình thường thì cách tốt hơn cả là chỉ có một điểm trả lại. Điều này bình thường đạt được bằng cách cấu trúc lại mã. Một thí dụ về một hàm với hai lệnh `return` (trả lại) được chỉ ra dưới đây. Trong thí dụ này, phải quyết định giá trị nhập vào hàm là dương hay âm. Nếu như giá trị này lớn hơn hoặc bằng 0 thì nó trả lại cùng giá trị đó, nếu không nó trả lại một giá trị âm (`return(-value)`).

```
int magnitude(int value)
{
```

```
    if (value >= 0)
      return (value);
else
    return (-value);
}
```

5.3 KIỂU HÀM

Chương trình 5.2 chứa các hàm, trả lại các kiểu dữ liệu số nguyên. Có thể trả lại các kiểu dữ liệu khác được sử dụng trong ngôn ngữ C, bao gồm `float`, `double` và `char`, bằng cách chèn kiểu dữ liệu vào trước tên hàm. Nếu như không có dữ liệu được chỉ định, thì kiểu dữ liệu trả lại được mặc định là `int`. Sau đây là cú pháp chung của một hàm.

```
type_def function_name(parmeter list)
{
}
```

C là một ngôn ngữ linh hoạt trong cấu trúc. Nó cho phép sắp xếp các hàm theo thứ tự bất kỳ và ngay cả bên trong các tệp khác. Nếu như chương trình dịch tìm thấy một hàm mà đã không được định nghĩa (hoặc chọn làm mẫu - prototype) nó mặc định thừa nhận kiểu dữ liệu trả lại là `int`. Nó cũng thừa nhận rằng ở giai đoạn liên kết, bộ kết nối sẽ có khả năng tìm hàm cần có hoặc là trong chương trình đã được dịch hiện tại, trong các thư viện hoặc các mã đối tượng khác. Như vậy điều này quan trọng ở chỗ là kiểu dữ liệu mà hàm trả lại đã được định nghĩa khi chương trình dịch đang biên dịch hàm; nói khác đi nó chấp nhận rằng kiểu dữ liệu trả lại là `int`.

Những lời khai báo hàm (hoặc chọn làm mẫu - prototype) thông thường được chèn hoặc là ở đầu của mỗi tệp tin, định xứ bên trong một hàm, hoặc là bên trong một tệp đầu mục (các tệp có phần mở rộng là *.h*). Cách khai báo này cho phép chương trình dịch xác định kiểu trả lại và kiểu dữ liệu của tất cả các tham số chuyển giao cho hàm. Như vậy cho phép chương trình dịch kiểm tra các kiểu dữ liệu không hợp lệ được chuyển giao cho hàm khi có lỗi. Thí dụ sau đây minh hoạ các cách sử dụng sai hàm `printf()`, `sqrt()` và `scanf()`. Hàm `printf()` có cú

pháp sai bởi vì tham số đầu tiên đã là một lệnh khuôn mẫu (cụ thể là một xâu), hàm `sqrt()` sẽ được chuyển giao một giá trị dấu phẩy động và hàm `scanf()` đòi hỏi một xâu khuôn mẫu như trong tham số đầu tiên.

```
printf(23,"Value is %d\n");
b=sqrt("1233");
scanf(&val1,&val2);
```

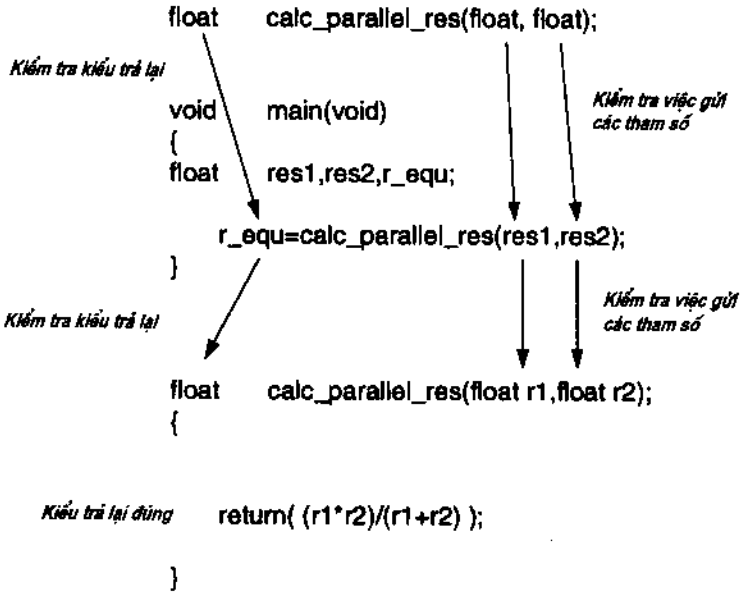
Nếu như không có đầu mục *stdio.h*, chương trình dịch không tạo ra các bất kỳ lỗi cho sự sử dụng sai của `printf()` và `scanf()`. Ứng dụng này cũng áp dụng cho tệp đầu mục và hàm `sqrt()`.

Khi một mẫu thử hàm được chèn ở đầu của tệp tin thì đây là một sự khai báo toàn cục của hàm ấy bên trong tệp mã nguồn. Còn một cách khác là phần khai báo có thể được chèn vào trong các danh sách khai báo biến bên trong một hàm; cách khai báo này sẽ làm cho định nghĩa định xứ chỉ tới hàm mà tại đó nó được định nghĩa.

Nếu một hàm không trả lại một giá trị thì định nghĩa kiểu dữ liệu cho giá trị trả lại sẽ là `void`. Đồng thời, nếu không có các tham số đã gửi tới hàm thì danh sách đối số có chứa một `void` . Như vậy, chương trình dịch sẽ đặt cờ báo lỗi hoặc cảnh báo nếu như có bất kỳ tham số đã chuyển giao vào hàm với một danh sách tham số `void` hoặc nếu như một giá trị được trả lại đã được sử dụng từ một kiểu dữ liệu trở về `void`.

Hình 5.8 là hoạt động của một mẫu thử hàm. Ở đầu tệp tin của mẫu thử khai báo các kiểu tham số của các đối số được chuyển giao (trong trường hợp này, hai `float`) và kiểu dữ liệu trả lại (trong trường hợp này là `float`). Chương trình dịch kiểm tra tất cả các đối số gửi cho hàm này để nhìn thấy nếu như chúng phù hợp với các kiểu này. Một tín hiệu cảnh cáo hoặc lỗi được phát sinh nếu như chúng không thích ứng. Kiểu trả lại cũng được kiểm tra.

Chương trình 5.3 có chứa hàm `power()` là mẫu thử ở đầu chương trình. Giá trị trả lại là `double`; tham số đầu tiên là `double` và tham số thứ hai là `int`. Hàm này sử dụng các hàm lôgarit để xác định giá trị của x đã được luỹ thừa bậc n.



Hình 5.8: **Sự kiểm tra chỉ đạo bởi trình dịch trên hàm các mẫu thử.**

Công thức sử dụng sẽ được dẫn ra dưới đây; hàm `log()` là lôgarit tự nhiên và `exp()` là hàm số mũ. Cả hai hàm này đòi hỏi tham số gửi tới là `double` và kiểu trả lại cũng là `double`.

$$y = x^n$$
$$\log(y) = \log(x^n)$$
$$\log(y) = n.\log(x)$$
$$y = \exp(n.\log(x))$$

Chương trình 5.3

```
/* prog5_3.c                                                    */
#include <math.h>
#include <stdio.h>

double power(double x,int n); /* declare function prototype */
int    main(void);
{
double x,val;
int    n;
 puts("Program to determine the result of a value");
```

```
    puts("raised to the power of an integer");

    printf("Enter x to the power of n >>");
    scanf("%f %d",&x,&n);

    val=power(x,n);
    printf("%f to power of %d is %f\n",x,n,val);
    return(0);
}

double power(double x,int n)
{
    return(exp(n*log(x));
}
```

Một vài chương trình dịch sẽ đưa ra thông báo nhắc nhở nếu như kiểu dữ liệu của giá trị trả lại khác với kiểu dữ liệu của biến mà nó được gán vào. Chẳng hạn, nếu như kiểu dữ liệu của biến x trong chương trình 5.3 được thay đổi thành một `float` thì đây là cách thực tế nhất để xếp lại giá trị trả lại vào kiểu `float`, như chỉ ra dưới đây:

```
int main(void)
{
float    x,val;
int   n;
  ::::::
  val=(float) power(x,n);
  ::::::
```

5.4 SỬ DỤNG BỘ TIỀN XỬ LÝ ĐỂ ĐỊNH NGHĨA MACRO HÀM.

Bộ tiền xử lý (pre-processor) tác động lên một tệp mã nguồn trước khi chương trình dịch được gọi. Một trong các chức năng của nó là tìm kiếm các tệp đầu mục cần thiết và chèn chúng vào trong tệp tin. Nó cũng tìm tất cả các macro `#define` và thay thế chúng bằng các định nghĩa cần có. Đây là cách thực tiễn cho phép định nghĩa hàm nối tiếp nhau (in-line) theo cách của các macro.

Chương trình 5.4 sử dụng bộ tiền xử lý để xác định giá trị của π và cũng để khai báo một macro có tên là `_sqr()`.

Chương trình 5.4

```
/* prog5_4.c                                          */
/* FILE before the pre-processor                      */
#include <stdio.h>
#include <math.h>
#define    PI    3.14159
#define    _sqr(_X) ((_X)*(_X))

int  main(void)
{
float   r,f,l,Z,X1;
  printf("Enter resistance, inductance and frequency>>");
  scanf("%f %f %f",&r,&l,&f);

  X1=2*PI*f*l;
  Z=sqrt(_sqr(r)+_sqrt(X1));

  printf("Magnitude of impedance is %f\n",Z);
  return(0);
  }
```

Chương trình 5.5 cho thấy bộ xử lý sửa đổi chương trình 5.4 như thế nào. Chú ý rằng nội dung của các tệp tin *stdio.h* và *math.h* đã bị cắt xén (truncate).

Chương trình 5.5

```
/*  prog5_5.c
/* FILE after the pre-processor has operated  */
:::::::::::::::::: contens of stdio:::::::::::::::::

:::::::::::::rest of the contents of stdio.h::::::::

:::::::::::::::::: contents of math.h::::::::
int abs   (int x);
```

```
int acos  (double x);
int asin  (double x);
int atan  (double x);
int tan2  (double y, double x);
int atof  (const char *s);
:::::::::::: rest of the contents of math.h:::::::::
int main(void)
{
float  r,f,l,Z,x1;

printf("Enter resistance, inductance and freqency>>")
scanf("%f %f %f",&r, &l, &f);

x1=2*3.14159*f*l;
z=sqrt(((r)*(r))+((x1)*(x1)));

printf("Magnitude of impedance is %f\n",Z);
return(0)
}
```

Như đã nhìn thấy từ chương trình 5.5 có thể dùng `#define` để tạo ra các macro đơn giản. Chẳng hạn, để tạo ra một hàm căn bậc hai có thể dùng:

```
#define _sqr(_A)'((_A.) * (_A))
```

Trong trường hợp này bộ xử lý thay thế mỗi lần xuất hiện (occurrence) của định nghĩa macro `_sqr(ARG)` bằng `((ARG)*(ARG))`. Thí dụ:

```
Z=sqrt(_sqr(x)+_sqr(y));
```

thay thế bằng:

```
Z=sqrt( ((x)*(x)) + ((y)*(y)) );
```

Khi khai báo các tham số đã sử dụng các hàm nối tiếp nhau (in-line), bằng cách sử dụng chỉ dẫn hướng `#define` và bình thường khi viết chương trình đều có đặt trước một dấu gạch dưới. Đây cũng là nét

tiêu biểu đối với những tên hàm có đặt trước một dấu gạch dưới (thí dụ `_tolower()` và `_toupper()`). Cần hết sức thận trọng khi sử dụng bộ tiền xử lý thay thế các hàm vì người lập trình hầu như không nhìn thấy hoạt động của chúng. Dấu sổ ngoặc sẽ được sử dụng để bảo đảm chắc rằng chúng thao tác đúng khi được chèn vào trong mã.

Một thí dụ cho thấy cái gì có thể dẫn đến lỗi được cho trong thí dụ sau đây, trong trường hợp này là một định nghĩa sai của `_sqr()`. Với định nghĩa sau các dấu ngoặc xung quanh `-X*-X` bị bỏ sót.

```
#define sqr(-x)    -X*-X
```

Như vậy mã sau đây :

```
Z = sqr(a)/sqr(b);
```

sẽ được thay thế bởi:

```
Z = a*a/ b*b ;
```

Điều chưa đúng ở đây là toán tử / đã đặt trước toán tử *. Nó sẽ được dịch là phép nhân với `a` chia cho `b` rồi lại nhân với `b`, trong khi lại phải lấy kết quả của `a` nhân với `a` đem chia cho kết quả của `b` nhân với `b`.

```
Z=(a*a)/(b*b);
```

Chương trình 5.6 là một thí dụ thực tế, cho thấy `#define` có thể gây ra lỗi như thế nào.

Chương trình 5.6

```
/* prog5_6.c                                          */
#include    <math.h>
#include    <stdio.h>

#define   PI     3.14159
#define _sqr(-X)   (_x*_x)

/* FILE before the preprocessor                       */
int   main(void)
{
float r1,r2,f,l,Z,X1;
```

```
    printf("Enter R1 and R2, inductance and frequency>>");
    scanf("%f %f %f",&r1,&r2,&l,&f);

    Xl=2*PI*f*l;
    Z=sqrt(_sqr(r1+r2)+_sqr(xl));

    printf("Magnitude of impedance is %f\n",Z)
    return(0);
}
```

Bộ tiền xử lý thay thế `_sqr(r1+r2)` bằng `r1+r2*r1+r2`. Tác động này là sai bởi vì phép nhân được đặt trước phép cộng. Chương trình 5.7 chỉ ra tệp tin đã được xử lý.

Chương trình 5.7

```
/* prog5_7.c                                          */
:::::::::      Contents of math.h and ::::::::::::
/* FILE after the pre-processor                       */
int main (void)
{
float r1,r2,f,I,Z,X1;

    printf("Enter R1 and R2, inductance and frequency>>");
    scanf("%f %f %f %f",&r1,&r2,&l,&f);

    Xl=2*3.14159*f*l;
    Z=sqrt((r1+r2*rl.r2)+(Xl*Xl));

    printf("Magnitude of impedance is %f\n",Z)
    return(0);
}
```

Một vài macro sử dụng chỉ dẫn hướng `#define` được chỉ ra sau đây. Dòng lệnh đầu tiên `_toupper()` biến đổi một ký tự thường thành ký tự hoa; Dòng lệnh thứ hai biến đổi một ký tự hoa thành ký tự thường. Các

macro này được tìm trong tệp đầu mục *ctype.h*. Thí dụ sau biến đổi một giá trị (có kiểu dữ liệu bất kỳ) thành độ lớn của nó.

```
#define _toupper(c)     ((C)+ 'A'-'a')
#define _tolower(c)     ((C)+ 'a'-'A')
#define _mag(_A)        (_A<0) ? (_A) : (_A)
```

Lệnh cuối cùng sử dụng một cách viết tắt (tiếng Anh: shorthand) được dùng để trình bày lệnh `if`. Các toán tử `?` và `:` có thể được giải thích như 'then' và 'else' tương ứng.

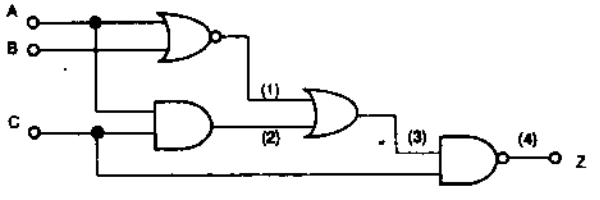
5.5 MỘT SỐ THÍ DỤ

5.5.1 LÔGIC TỔ HỢP

Trong thí dụ này, phương trình đại số Boole dưới đây được xử lý để xác định bảng giá trị chân lý của nó.

$$Z = \overline{\overline{(A+B+(A.C))}.C}$$

Hình 5.9 giới thiệu một sơ đồ của hàm Boole này.



Hình 5.9: **Mô tả hàm** $Z = \overline{\overline{(A+B+(A.C))}.C}$ **bằng sơ đồ.**

Bốn điểm được đánh số trên sơ đồ này:

(1) $\overline{A+B}$

(2) $A.C$

(3) $\overline{\overline{A+B}+(A.C)}$

(4) $\overline{\overline{(\overline{A+B})+(A.C)).C}}$

Bảng 5.1 giới thiệu bảng giá trị chân lý thông qua mức lôgic ở mỗi điểm trong sơ đồ. Bảng này cần có để kiểm tra các kết quả chương trình khi đối chiếu với các kết quả mong đợi.

Bảng 5.1: **Bảng giá trị chân lý.**

A	B	C	$\overline{A+B}$ (1)	A.C (2)	A+B+(A.C)(3)	$\overline{A+B+(A.C)}$.C	$\overline{A+B+(A.C)}$.C (4)
0	0	0	1	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	0	0	1
0	1	1	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	1	0	1	1	1	0
1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	0

Bảng 5.2 chỉ ra bảng giá trị chân lý kết quả:

Bảng 5.2: **Bảng giá trị chân lý.**

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Sự hoán vị các biến lối vào bảng chân lý (cụ thể là 000; 001; 010; 011; 111) được tạo ra bằng cách sử dụng 3 vòng lặp `for` lồng vào nhau. Vòng lặp bên trong thay đổi (toggle) C từ một số 0 sang một số 1, vòng tiếp theo thay đổi B và vòng lặp bên ngoài thay đổi A. Các hàm Boole sử dụng các toán tử lôgic && và ||. Gọi lại (recall) các toán tử này xử lý một giá trị bằng 0 như là FALSE và một vài giá trị khác như là TRUE.

Chương trình 5.8

```
/* prog5_8.c                                                    */
/* Example showing use of functions in returning */
/* values, in this case binary values                           */
#include <stdio.h>
#define  FALSE 0
#define  TRUE  1

  /* ANSI C function prototypes                  */
int     AND(int X,int y);
int     NAND(int x,int y);
int     NOR(int x,int y);
int     OR (int x, int y) ;
int     NOT(int x);

int     main (void)
{
int      a, b, c, z;

/* Go through all permutations of truth table    */
 puts(*     A      B      c      Result") ;
 puts(" ***************************");

 for (a=FALSE;a<=TRUE;a++)
  for (b=FALSE;b<=TRUE;b++)
   for (C=FALSE;c<=TRUE;C++)
       {
       Z=NAND (OR(NOR(a,b), AND(a,c)), c) ;
       printf("%6d %6d %6d %6d\n",a,b,C,Z);
       }
return(0);
}
int    AND(int x,int y)
{
  if(x && y ) return=(FALSE);
```

```
  else return(TRUE);
}
int     NAND(int x,int y)
{
  if  ( x && y ) return(FALSE);
else    return(TRUE);
}
int     OR(int x,int y)
{
  if ( x || y ) return(TRUE);
  else return(FALSE);
}
int    NOR(int x,int y)
}
  if ( x || y) return(FALSE)
  else return(TRUE);
}

int     NOT(int x)
{
  if (x) return(FALSE);
  else return(TRUE);
}
```

Chạy thử 5.1 giới thiệu một thí dụ làm mẫu khi cho chạy chương trình này. Chú ý rằng các kết quả đồng nhất tới bảng giá trị chân lý. được tạo ra bằng cách phân tích sơ đồ.

Cũng có thể sử dụng hàm nối tiếp nhau (in-line) cho các hàm Boole. Lệnh

```
#define _AND(_x,_y) ((_x && _y) ? TRUE : FALSE
```

là cách giới thiệu viết tắt (shorthand) khi giới thiệu lệnh `if`. Nếu lệnh này là TRUE thì sẽ dẫn đến một TRUE lôgic, nếu không thì là FALSE. Các toán tử `?` và `:` có thể được dịch tương ứng là "`then`" và "`else`".

Chạy thử 5.1

```
A     B     C     Kết quả
***   ***   ***   ********
0     0     0     1
0     0     1     0
0     1     0     1
0     1     1     1
1     0     0     1
1     0     1     0
1     1     0     1
1     1     1     0
```

Chương trình 5.9 tương đương với chương trình 5.8 nhưng sử dụng các chỉ dẫn hướng #define cho các hàm Boole cơ bản. Cần hết sức thận trọng khi sử dụng các lệnh #define để thay thế các hàm bởi vì nhiều phép toán biến đổi là không nhìn thấy được và như vậy có thể gây ra các lỗi, khó đánh dấu. Điều này quan trọng là sự thay thế #define được kiểm tra độc lập và sau đó là ở bên trong một chương trình trước khi nó có thể được chứng minh khả năng hoạt động đúng.

Chương trình 5.9

```
/* prog5_9.c                                              */
/* Example showing use of macros in yielding              */
/* values, in this case logical values                    */
#include <stdio.h>

#define  FALSE 0
#define  TRUE  1

#define    _AND(_x,_y)   ((_x &&_y) ? TRUE  :   FALSE)
#define    _OR (_x,_y)   ((_x 11_y) ? TRUE  :   FALSE)
#define    -NAND(_x,_y)  ((_x &&_y) ? FALSE :    TRUE)
#define    _NOR(_x,_y)   ((_x 11_y) ? FALSE :    TRUE)
#define    _NOT(_x,_y)   ((_x )      FALSE :    TRUE)

int    main(void)
```

```
{
int  a, b, c, z
/* Go through all permutations of truth table  */
   puts("     A     B     C     Result");
   puts(" ************************ ");

   for (a=FALSE;a<=TRUE;a++)
      for (b=FALSE;b<=TRUE;b++)
         for (c=FALSE;c<=TRUE;c++)
         {
            z=_NAND(_OR(_NOR(a,b),_AND(a,c)), c);
            printf("%6d %6d %6d %6d\n",a,b,c,z);
      }
   return(0)
}
```

Sau khi hàm macro nối tiếp nhau (in-line) đã được kiểm tra đầy đủ, chúng có thể được đặt trong tệp đầu mục được người dùng định nghĩa. Tệp tin này không nên đặt trong thư mục có chứa các tệp đầu mục chuẩn, nhưng sẽ được chèn vào trong một thư mục tệp đầu mục riêng cho người dùng hoặc đặt trong cùng thư mục như mã nguồn. Trong trường hợp này, các macro sẽ đặt vào trong tệp đầu mục *boole.h*. Kỹ thuật sử dụng các tệp đầu mục cho phép các tệp tin mã nguồn khác để truy nhập tới các chức năng in-line của bộ xử lý, các mẫu thử hàm hoặc các định nghĩa hằng.

Ở đây tốc độ của mã là quan trọng, các hàm nối tiếp nhau (in-line) tác động nhanh hơn hàm tương đương bởi vì chúng không kéo theo sự chuyển giao tham số. Chương trình 5.10 chỉ ra mã sửa đổi với tệp đầu mục bao gồm. Nội dung của tệp tin *boole.h* được chỉ ra dưới đây; nó bao gồm định nghĩa của hàm Boole bằng cách sử dụng các macro và đồng thời các sự khai báo hàm.

```
/* Nội dung của tệp Boole.h   */

#define FALSE 0
#define TRUE 1
```

```
/*   Definition using   macros                          */
 #define _AND(_x,_Y) ((_X && _Y) ? TRUE  :  FALSE
 #define _OR(_x,_Y)  ((_x || _Y)  ? TRUE   :  FALSE
 #define _NAND(_x,_Y)   ((_X && _Y)  ? FALSE  :  TRUE
 #define _NOR(_x,_Y) ((_x || _Y)  ? FALSE  :  TRUE
 #define _NOT(_x,_Y) ((_X )       ? FALSE  :  TRUE

 /*ANSI C Boolean function prototypes  */
 int     AND(int x,int y);
 int     NAND(int x,int y);
 int     NOR(int x,int y);
 int     OR (int x, int Y);
 int     NOT(int x);
```

Một tệp đầu mục mà người dùng đã định nghĩa được chèn vào trong chương trình bằng cách sử dụng chỉ dẫn hướng (dành cho bộ xử lý) `#include " "`.

Chương trình 5.10

```
/* Prog5_10.c                                          */
/* Example showing use of macros in yielding   */
/* values, in this case logical values         */

#include <stdio.h>
#include "boolean.h"

int  main(void)
{
int a,b,c,z;
/* Go through all permutations of truth table  */
puts("   A    B   C    Result");
puts("  ******************** ");

    for (a=FALSE;a<=TRUE;a++)
      for (b=FALSE;b<=TRUE;b++)
         for (c=FALSE;c<=TRUE;c++)
```

```
        {
Z= _NAND(-OR(_NOR(a,b),_AND(a,c)),c);
printf("%6d %6d %6d %6d\n",a,b,c,z);
        }
   return(0);
}
```

Chương trình 5.11 sử dụng các toán tử xử lý tới bit để thực hiện các hàm Boole. Phép toán VÀ đơn (&) là hàm AND xử lý tới bit và không được nhầm lẫn với lệnh AND lôgic (&&). Tương tự, phép toán với vạch thẳng đứng (|) là hàm OR xử lý tới bit và không được nhầm lẫn với hàm OR lôgic (||).

Chương trình 5.11

```
/* prog5_11.c                                          */
/* Example showing use of macros in yielding          */
/* values, in this case logical values               */

#include <stdio.h>
#include "boolean.h"

int   main (void)
int   a, b, c, z;
/* Go through all permutations of truth table     */
puts("  A          B     C     Result");
puts("  ***************************");
for (a=FALSE;a<=TRUE;a++)
for (b=FALSE;b<=TRUE;b++)
for (C=FALSE;c<--TRUE;c++)
Z=NAND(OR(NOR(a,b),AND(a,c)),c);
printf("%6d %6d %6d %6d\n",a,b,c,z),
}
    return (0);
}
int    AND(int x,int y)
}
    return(x&Y)
```

```
}
int    NAND(int x,int y)
{
     return( ! (x&y) );
}
int    OR(int X,int y)
{
     return( x|y );

}
int    NOR(inc x,int y)
{
     return ((x|y) );
}
int     NOT(int x)
{
     return(!(x));
}
```

5.5.2 TRỞ KHÁNG CỦA MẠCH RL NỐI TIẾP

Độ lớn trở kháng của mạch RL nối tiếp được cho bởi phương trình:

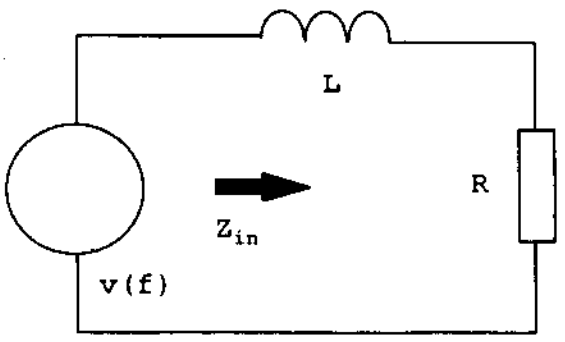
$$|Z| = \sqrt{R^2 + X_L^2} \quad [\Omega]$$

và góc pha của trở kháng này là:

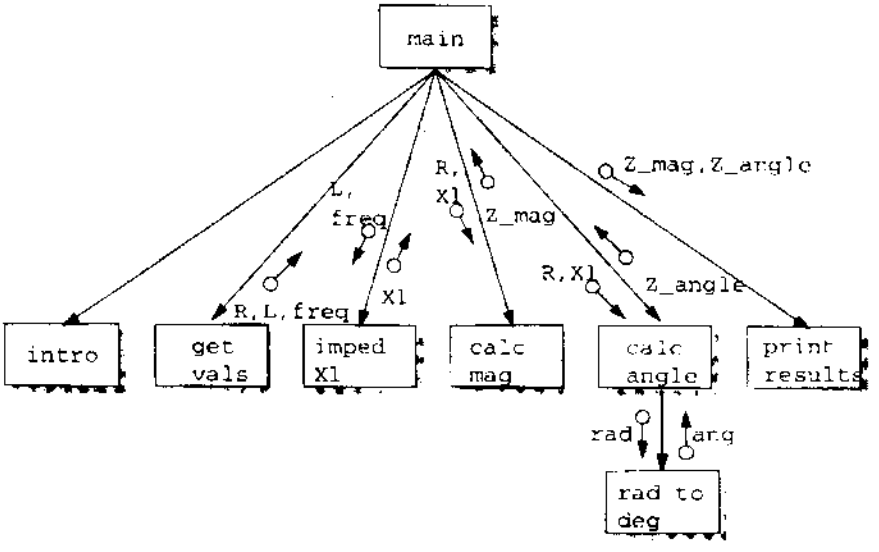
$$\langle Z = \tan^{-1} \frac{X_L}{R}$$

Hình 5.10 là sơ đồ mạch RL nối tiếp.

Hình 5.11 giới thiệu sơ đồ cấu trúc thể hiện thiết kế cơ bản cho bài toán này. Các đầu vào là điện trở (R), điện cảm (L) và tần số (f). Chương trình xác định độ lớn và góc pha của trở kháng. Để xác định các giá trị này, phải dùng công thức $X_L = 2\pi.fL$ để xác định điện kháng cuộn cảm.



Hình 5.10: Mạch RL nối tiếp.



Hình 5.11: **Biểu đồ cấu trúc của chương trình dùng cho mạch RL nối tiếp.**

Chương trình 5.12

```
/* prog5_12.c                                    */
/* Program to determine impedance of an          */
/* RL series circuit                             */

#include <stdio.h>
```

```
#include <math.h>

#define PI     3.14159
float    calc_mag(float x, float y);
float    calc_angle(float x, float y);
float    impedance_Xl(float L,float f);
float    rad_to_deg(float rad);
void     intro(void);
void     print_results(float mag,float angle);

int    main(void)
{
float r,l,Xl,Z_mag,z_angle,freq;
intro () ;
printf("Enter R,L and frequency >>>")
scanf("%f %f %f",&r,&l,&freq);
Xl=impedance_Xl(l,freq);
Z_mag=calc_mag(r,Xl) ;
Z_angle=calc_angle(r.Xl);
print_results(Z_mag,Z_angle);
return(0);
}
void   intro(void)
{
puts("Prog to determine impedance of series RL circuit");
}
void   print_results(float mag,float angle)
{
printf("Impedanceis %.2f  ohms Phase angle %.2f
        degrees\n",mag,angle);
}
float calc_mag(float x, float y)
{
float  z;
      /*determine magnitude for rectangular    */
      /*co-ordinates x+jy                      */
z=(float) sqrt(x"x+Y*Y); /*sqrt() returns a double */
```

```
return(z);
}
float cal_angle(float x, float y)
{
float ang;

   /* determine angle in degrees for rectangular */
   /* co-ordinates X+jy                          */

ang=(float) atan(y/x); */atan()returns a double */
   ang=rad_to_deg(ang);
return(ang);
}
float  impedance_Xl(float L,float f)
{
   /*determine impedance of inductor      */
return(2*PI*f*L);
}

float rad_to_deg(float rad)
{
   /*convert from radians to degrees  */
    return(rad*180/PI);
}
```

Chạy thử 5.2 đưa ra một thí dụ làm mẫu khi cho chạy chương trình này. Các tham số đầu vào sử dụng là điện trở 1 kΩ, điện cảm 1 mH và tần số sử dụng là 1 MHz.

Chạy thử 5.2

```
Program to determine impedance of series RL circuit
ENTER R,L and frequency >>> 1e-3   1e6
Impedance is 6362.27 ohms phase angle 80.96 degrees
```

Chương trình 5.13 là kết quả của việc bổ sung vào chương trình 5.12 bằng cách sử dụng các định nghĩa hàm nối tiếp nhau để thay thế

các hàm. Các hàm mạch được định nghĩa trong chương trình 5.1 có thể chèn vào trong một tệp tin có tên *ac.h*. Tệp này bao gồm trong bất kỳ chương trình nào khi sử dụng để tính toán trong mạch xoay chiều (A.C).

Chương trình 5.13

```
/* prog5_13.c                                          */
/* program to determine impedance of an                */
/* series RL circuit                                   */
#include <stdio.h>
#include <math.h>

/*define functions used, these functions could    */
/*go into a header file, e.g. circuit.h           */
#define  _sqr(_X)          ((_X)*( _X))
#define  _rad_to_deg(_X)       ((_X)*180/PI)
#define  _impedance_x1(_L,_f) (2*PI*_f*_L)
#define  _cal_mag(_X, _Y)     (sqrt((_sqr(_X)+ _sqr(_Y))))
#define  _calc_angle(_X,_Y)   (_rad_to_deg(atan(_Y/_X)))
#define   PI                  3.14159

void    intro (void) ;
void    print_results(float mag, float angle) ;

int     main (void)
float r,l,Xl,Z_mag,Z_angle,freq;

intro () ;

printf("Enter R,L and frequency >>>");
 scanf("%f %f &f",&r,&l,&freq);

 Xl=_impedance_Xl(1,freq);

 Z_mg=_calc_mag(r,xl);
 Z_angle=_calc_angle(r,xl);

 print_results (Z_mag,Z_angle);
 return(0);
```

```
}
void   intro (void)
{
  puts("Program to determine impedance of series RL circuit");
}
void   print_results(float mag,float angle)
{
 printf("impedance is %.2f ohms phase angle %.2f
                          degrees\n",mag,angle);
}
```

Chạy thử 5.3 là một thí dụ làm mẫu khi cho chạy chương trình này

Chạy thử 5.3

```
Program to determine impedance of series RL circuit
ENTER R,L and frequency >>> 1e-3   1e6
Impedance is 6362.27 ohms phase angle 80.96 degrees
```

5.5.3 ĐÁP ỨNG XOAY CHIỀU CỦA MẠCH RC NỐI TIẾP

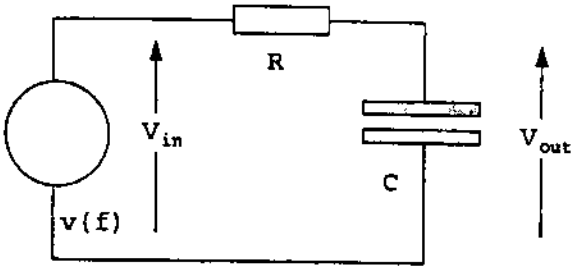
Trong mạch RC như mô tả trên hình 5.12, tỷ số giữa điện áp đầu ra và đầu vào có thể được sử dụng để xác định đáp ứng của mạch. Đáng chú ý là tỷ số này tính bằng đêxiben (dB).

Điện áp đầu ra của mạch này có thể xác định bằng công thức:

$$V_O = V_i . \frac{-jX_C}{R - jX_C} \quad [V]$$

ở đây:

$$X_C = \frac{1}{2\pi . fC} \quad [\Omega]$$



Hình 5.12: **Mạch RC nối tiếp với điện áp đầu vào và điện áp đầu ra.**

Như vậy độ lớn của tỷ số điện áp là:

$$\left|\frac{V_O}{V_I}\right| = \frac{X_C}{\sqrt{R^2 + X_C^2}}$$

Chương trình 5.14 xác định độ lớn của tỷ số điện áp này tương ứng với các giá trị của R, C và f. Một vấn đề có thể xuất hiện trong chương trình này là lỗi chia cho 0, khi mẫu số của công thức tính dung kháng bằng 0 (khi f hoặc C bằng 0). Giải quyết vấn đề này hàm `impedance_Xc()` xác định nếu như mẫu số bằng 0 và trả lại cờ không xác định (INFINITYFLAG) để gọi đoạn chương trình nếu như nó có. Hàm `RC-response()` lúc đó tìm kiếm điều này và trả lại giá trị 1 của hệ số biến áp. Giá trị -1 đã chọn cho dấu này là giá trị không thể xuất hiện trong các phép tính bình thường.

Chương trình 5.14

```
/* prog5_14.c                                              */
/* Program to determine the voltage ratio of an    */
/* RC series circuit with the capacitor on the     */
/* output                                                  */
#include <stdio.h>
#include <math.h>

#define INFINITYFLAG  -1
#define PI            3.14159
/* function prototypes          */
```

```
float RC_response(float r,float C,float f);
float impedance_Xc(float C, float f);
float dBs(float Gain);
float calc_mag(float x, float y);

int main(void)
{
float f_start,f_end,f_step,f,r,C, gain;

  printf("Enter start, end and step freq >>");
  scanf("%f %f %f",&f_start,&f_end,&f_step);

  printf("Enter resistance and capacitance >> ");
  scanf("%f %f",&r,&C);

  puts("Frequency    Voltage    gain ");
  puts(" (Hz)        ratio      dBs");
  puts(**************************** );

  for (f=f_start;f<=f_end;f+=f_step)
  {
  gain=RC_response(r,C,f);
  printf("%8.2f %8.3f %8.3f\n",f,gain,dBs(gain));
  return(0);
}
float RC_response(float r,float C,float f)
float Xc,response;
  /* Determine gain of RC circuit */

  Xc=impedance_xc(C,f);
  if (Xc==INFINITYFLAG) response=1.0;
  else response=Xc/(calc_mag(r,Xc));

  return(response);
}
float impedanc_.Xc(float C, float F)
```

```
{
  /* Calculate impedance of a capacitor */
  /* Beware when dividing by zero       */
   if ((F==0) ||(C==0)) return(INFINITYFLAG);
   else return(1/(2*PI*F*C));
{
float calc_mag(foat x, float y)
{
  /* calculate magnitude of complex value x+jy */
   return (sqrt(X*X+Y*Y));
}
float dBs(float gain)
{
  /* Convert Voltage ratio to dBs */
  return(20*log10(gain));
}
```

Kết quả chạy thử 5.4 giới thiệu một thí dụ làm mẫu của chương trình với các giá trị R = 1 kΩ và C = 1 μF. Tần số quét từ 0 Hz đến 1 kHz với mỗi bước 50 Hz. Chú ý rằng ở 0 Hz điện áp đầu ra là cực đại và khi tần số tăng thêm thì điện áp đầu ra giảm đi. Mạch này hiển thị các đặc tính của một bộ lọc dải thông thấp (LPF).

Chạy thử 5.4

```
Enter start, end and step freq >> 0 100 50
Enter resistance and capacitance >> 1000 1e-6
   Frequency   Voltage   Gain
   (Hz)        ratio     dBs
   0.00        1.000      0.000
   50.00       0.954     -0.409
   100.00      0.847     -1.445
   150.00      0.728      2.761
   200.00      0.623      4.115
   250.00      0.537      5.400
   300.00      0.469      6.583
   350.00      0.414     -7.661
   400.00      0.370      8.643
```

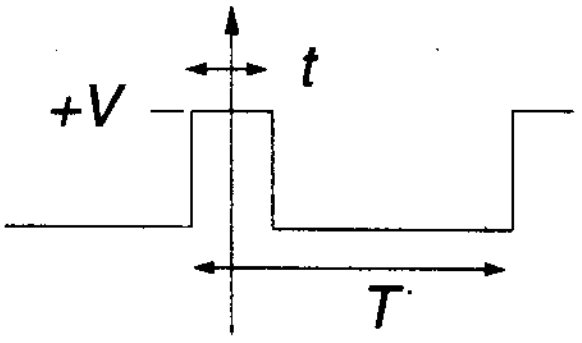
450.00	0.333	9.540
500.00	0.303	10.362
550.00	0.278	11.120
600.00	0.256	11.822
650.00	0.238	12.475
700.00	0.222	13.084
750.00	0.208	13.656
800.00	0.195	14.194
850.00	0.184	14.702
900.00	0.17	15.182
950.00	0.165	15.638
1000.0	0.157	16.072

5.5.4 SÓNG HÀI CỦA MỘT SÓNG XUNG LẶP LẠI

Tín hiệu biến đổi liên tục có phổ tần cũng biến đổi, trong khi một tín hiệu lặp đi lặp lại là một dãy tần số gián đoạn được gọi là sóng hài. Nếu như tín hiệu lặp lại có chu kỳ bằng T thì tần số cơ bản là 1/ T Hz. Tất cả các thành phần tần số khác tạo ra tín hiệu có tần số cơ bản hoặc tần số bội, thí dụ từ 2, 3, 4... lần. Chẳng hạn, một tín hiệu lặp lại với chu kỳ 10 ms có tần số từ 100 Hz, 200 Hz, 300 Hz, v. v... Nếu dạng sóng là một sóng hình vuông thì nội dung tần số của sóng chỉ có các sóng hài bậc lẻ (thí dụ 100 Hz, 300 Hz v. v...).

Dạng sóng xung hình chữ nhật được minh hoạ trên hình 5.13. Chu trình làm việc của sóng là tỷ số giữa thời gian có xung (t) và chu kỳ của dạng sóng (T). Chu trình làm việc được tính bởi công thức:

$$\text{Chu trình làm việc} = \frac{1}{T}$$



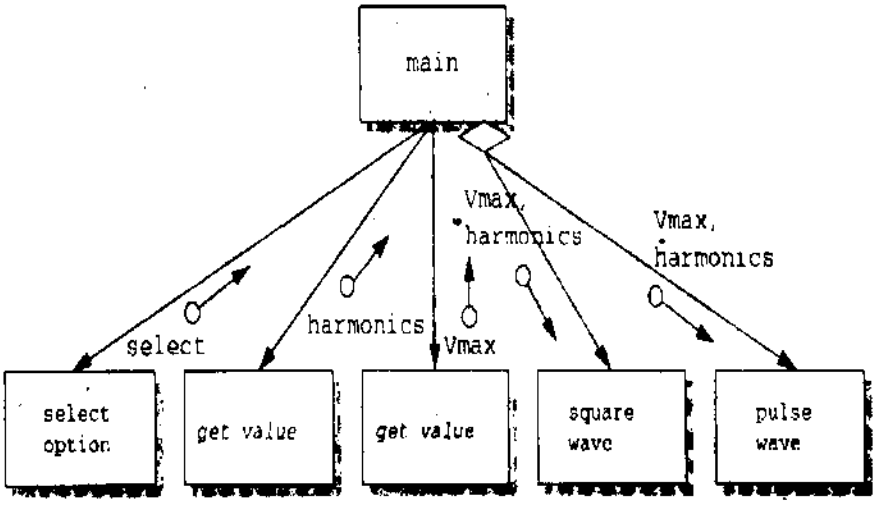
Hình 5.13: **Chu trình làm việc.**

Biên độ của các thành phần tần số tạo nên tín hiệu này được cho bởi:

$$v_n(t) = \frac{V.t}{T} + \sum_{n=1}^{n=\infty} V_n \cos(n2\pi.ft)$$

ở đây:

$$V_n = \frac{2Vt}{T} \frac{\sin N.x}{N.x}$$



Hình 5.14: **Biểu đồ cấu trúc của chương trình 5.15.**

và:

$$x = \frac{\pi.t}{T}$$

V_1 là biên độ của tín hiệu cơ bản, V_2 là biên độ của sóng hài bậc hai v.v... Tần số được chứa trong tín hiệu là:

$$f_1 = \frac{1}{T} \text{ Hz; } f_2 = \frac{2}{T} \text{ Hz; } f_3 = \frac{3}{T} \text{ Hz; v. v...}$$

Thiết kế cơ bản của một chương trình để xác định sóng hài được mô tả trên hình 5.14.

Chương trình 5.15 xác định biên độ sóng hài của sóng hình vuông hoặc sóng xung hình chữ nhật.

Chương trình 5.15

```
/* prog5_15.c                                          */
/* program to determine the harmonics of       */
/* repetitive rectangular or square pulses        */

#define   PI       3.14159
#define   TRUE     1

#include <math.h>
#include <stdio.h>

void      Square_wave(int harm,float Vmax);
void      Pulse_wave(int harm,float Vmax);
int Selectoption(void);

int       main(void)
{
int    select,harmonics;
float   Vmax;

/* infinite loop until break    */
```

```
  while (TRUE)
  {
      select=Selectoption();
      if (select==3)break;

     puts("Enter number of hamonics required");
     scanf("%d",&harmonics);

     puts("Enter max voltage");
     scanf("%f",&Vmax);

     switch (select)
 {
   case 1: Square_wave(harmonics,vmax);  break;
   case 2: Pulse_wave (harmonics,vmax);  break;
  }
   }
 return(0);
}
void   square_wave(int harm,float vmax)
{
float  v;
int    i;

  for (i=1;i<=harm;i+=2)
  {
    v=(4*vmax/i/PI);
    printf("Harmonic %d Amplitude %.3f\n",i,v);
   }
}
void   Pulse_wave(int ham,float Vmax)
{
float Duty,v;
int   I;
```

```
    puts("Enter duty cycle (0->1)");
    scanf("%f",&Duty);
    v=(vmax*Duty);
    printf("DC %.3f\n",v);
    for(i=1;i<=harm;i++)
    {
    V=(2*Vmax*Duty) * sin(I*PI*Duty) / (I*PI*Duty);
    printf("Harmonic %d Amplitude %.3f\n",i,v);
    }
}
int     Selectoption(void)
{
int     Select;
  do {
     puts("Do you wish");
     puts("1- Square wave");
     puts("2- Pulse train*);
     puts("3- Exit");
     scanf("%d",&Select);

  } while ( (Select<0) 11 (Select>3))
  return(Select);
}
```

Kết quả chạy thử 5.5 là một thí dụ được đưa ra làm mẫu.

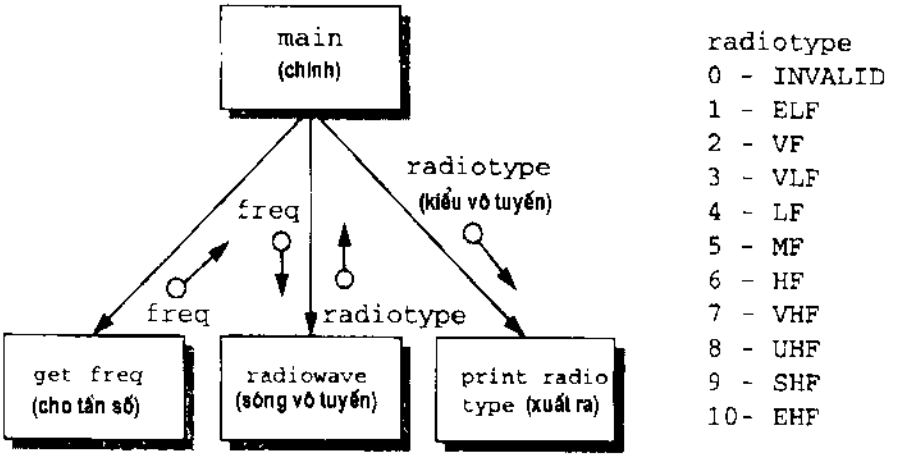
Chạy thử 5.5

```
Do you wish
1- Square wave
2- Pulse train
3- Exit
1

Enter number of harmonics required
```

```
10
Enter max voltage
4
Harmonic 1 Amplitude 1.274
Harmonic 3 Amplitude 0.425
Harmonic 5 Amplitude 0.255
Harmonic 7 Amplitude 0.182
Harmonic 9 Amplitude 0.142
**************************
DO you wish:
1-  Square wave
2-  Pulse train
3- Exit
2
Enter number of harmonics required
10
Enter max voltage
1
Enter duty cycle (0->1)
0.2
DC 0.200000
   Harmonic 1   Amplitude  0.374
   Harmonic 2   Amplitude  0.303
   Harmonic 3   Amplitude  0.202
   Harmonic 4   Amplitude  0.094
   Harmonic 5   Amplitude  0.000
   Harmonic 6   Amplitude  -0.062
   Harmonic 7   Amplitude  -0.086
   Harmonic 8   Amplitude  -0.076
   Harmonic 9   Amplitude  -0.042
   Harmonic 10 Amplitude   -0.000
Do you wish
1-  Square wave
2-  Pulse train
3- Exit
3
```

5.5.5 PHÂN LOẠI CÁC SÓNG VÔ TUYẾN

Chương trình 5.16 được dùng để phân loại một sóng vô tuyến được nhập vào chương trình. Hàm `radiowave()` trả lại cho một số nguyên trong phạm vi 0 tới 10. Hàm này được dùng để phân loại sóng vô tuyến. Nếu như giá trị trả lại bằng 0 thì đầu vào là KHÔNG HỢP LỆ (INVALID) ! ! ! Biểu đồ cấu trúc cơ bản của chương trình này được giới thiệu trên hình 5.15.



Hình 5.15: **Biểu đồ cấu trúc cho chương trình sóng vô tuyến.**

Chương trình 5.16

```
/* prog5_16.c                                          */
/* program to determine classification of radio wave */
#include <stdio.h>
        /* #define function returns */
#define INVALID    -1
#define ELF         0
#define VF          1
#define VLF         2
#define LF          3
#define MF          4
#define HF          5
#define VF          6
```

```
#define SHF         7
#define UHF         8
#define SHF         9
#define EHF         10

void    printf_radio_type(int type);
int     radiowave(float f);
float   get_freq (void);

int     main(void)
{
float   freq;
int     radiowave_type;

  freq=get_freq();

  radiowave_type=radiowave(freq);
  print_radio_type(radiowave_type);
  return(0);
}
float get_freq(void);
{
float f;
  printf("Enter frequency of radio wave>>");
  scanf("%f",&f);
  return(f);
}

void   print_radio_type(int type)
{
/* printf radio wave type */
   printf("Electromagnetic wave is ")
   switch(type)
  {
      case     ELF:  puts("Extralow frequency");      break;
```

```
        case    VF:     puts("Voice frequency");          break;
        case    VLF:    puts("Very low frequency");       break;
        case    LF:     puts("Low frequency");            break;
        case    MF:     puts("Medium frequency");         break;
        case    HF:     puts("High frequency");           break;
        case    VHF:    puts("Very high frequency");      break;
        case    UHF:    puts("Ultra high frequency");     break;
        case    SHF:    puts("Super high frequency");     break;
        case    EHF:    puts("Extra high frequency");     break;
        case    TOOLARGE: puts("Input wave too large") break;
        default:        puts("Invalid")
    }
}
int  radiowave(float f)
{
*/ determine classification of radiowave */
    if (f<0)                return(INVALID);
    else  if  (f<300)       return(ELF);
    else  if  (f<3e3)       return(VF);
    else  if  (f<30e3)      return(VLF);
    else  if  (f<300e3)     return(LF);
    else  if  (f<3e6)       return(MF);
    else  if  (f<30e6)      return(HF);
    else  if  (f<300e6)     return(VHF);
    else  if  (f<3e9)       return(UHF);
    else  if  (f<30e9)      return(SHF);
    else  if  (f<300e9)     return(EHF);
    else                    return(TOOLARGE);
  }
```

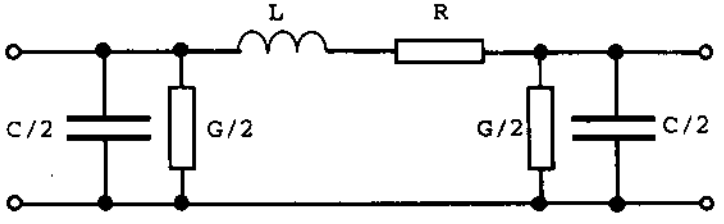
Kết quả chạy thử 5.6 là một một thí dụ được giới thiệu làm mẫu.

Chạy thử 5.6

```
Nhập vào tần số của sóng vô tuyến >> 1e-3
Sóng điện từ có tần số âm thanh
```

5.5.6 TRỞ KHÁNG ĐƯỜNG TRUYỀN

Một đường truyền được dùng để truyền các tín hiệu số hoặc tương tự từ máy phát tới máy thu. Nó có thể là một cáp đồng trục, cáp xoắn, v. v... Theo quan điểm mạch điện, đường truyền được tạo ra từ một dãy các điện trở và điện cảm nối tiếp, hình thành từ trở kháng nối tiếp của các dây dẫn, và một điện dẫn và điện dung của chất cách điện giữa các dây dẫn mắc song song, tạo ra trở kháng mắc sun với đường truyền. Nếu chiều dài cho trước của đường truyền này được chia ra các đoạn và chia tiếp mãi, trường hợp cuối cùng sẽ là một đoạn nhỏ vô cùng của các phần tử cơ bản: điện trở R, điện cảm L, điện dẫn G, và điện dung C. Mạch tương đương của các kết quả đó cho trong hình 5.16.



Hình 5.16: **Mạch tương đương của một đường truyền.**

Các tham số R, L, G và C được gọi là các hằng số đường truyền sơ cấp (primary). Đó là:

- ❒ Điện trở nối tiếp R — $\Omega.m^{-1}$
- ❒ Điện cảm nối tiếp L — $Henry.m^{-1}$
- ❒ Điện dẫn song song G — $Siemens.m^{-1}$
- ❒ Điện dung song song C — $Farad.m^{-1}$

Trở kháng đặc tính Z_0 là tỷ số giữa điện áp và dòng đối với mỗi sóng lan truyền dọc theo một đường truyền và được cho bởi công thức:

$$Z_0 = \frac{V}{I}$$

Từ công thức này Z_0 có thể được chỉ ra dưới dạng:

$$Z_0 = \sqrt{\frac{R + j\varpi.L}{G + j\varpi.C}}$$

Độ lớn của trở kháng này được tính bởi:

$$Z_0 = \sqrt{\frac{R^2 + (2\pi.fL)^2}{G^2 + (2\pi.fC)^2}}$$

Chương trình 5.17 xác định độ lớn của trở kháng đặc trưng cho một đường truyền với các hằng số đường dây sơ cấp được nhập vào. Một vấn đề có thể xuất hiện trong chương trình này là lỗi chia cho số 0 xảy ra khi mẫu số của công thức trở kháng đặc trưng bằng 0 (khi mà G và ωC bằng 0). Để giải quyết vấn đề này hàm `calc_Zo()` xác định nếu như mẫu số bằng 0 (`value2`) và trả lại một cờ không xác định (INFINITYFLAG) tới đoạn chương trình (routine) gọi. Nếu chương trình chính tìm ra cờ này thì nó hiển thị trở kháng không xác định (INFINITY). Giá trị -1 đã được chọn cho cờ này bởi vì giá trị này không thể xuất hiện trong các phép tính bình thường.

Chương trình 5.17

```
*/ prog5_17.c                                                    */
*/ program to determine impedance of a transmission line        */
*/ Impedance of TL is    root((Rl+wL)I(G+jwC))                  */
#include <stdio.h>
#include <math.h>

#define  MILLI              1e-3
#define  MICRO              1e-6
#define  INFINITYFLAG       -1
#define  PI                 3.14159

float    calc_Zo(float r,float l,float g,float c,float f);
float    calc_mag (float X, float y) ;
float    calc_imp(float f, float val);

int    main(void)
{
float Zmag,R,L,G,C,f;
```

```
puts("program  to determine  impedance of a transmission
                              line");
printf("Enter R,L(mH),G(mS),C(uF)and freq.>>"):
scanf("%f %f %f %f %f",&R,&L,&G,&C,&f);
Zmag=calc_Zo(R,L*MILLI,G*MILLI,C*MICRO,f);
if(Zmag==INFINITYFLAG)printf("Magnitude,is INFINITY ohms\n");
else      printf ("Magnitude is %.2f ohms\n", Zmg) ;
return(0);
}
float calc_Zo (float r, float 1, float g, float c, float f)
{
float value1,value2;

  value1=calc_mag(r,calc_imp(f,l));
  value2=calc_mag(g,calc_imp(f,c));
  */ Beware if dividing by zero */
  if (value2==0) return(INFINITYFLAG);
  else           return(sqrt(value1/value2));
}
float calc_mag(float X,float y)
{
   return(sqrt(x*x)+(Y*Y));
}
float calc_imp(float f, float val)
{
   return(2*PI*f*val);
}
```

Kết quả chạy thử 5.7 là một thí dụ được giới thiệu làm mẫu.

Chạy thử 5.7

```
Program to determine impedance of a tranmission line
Enter R, L(mH), G(mS), C(uF) and freq. >> 0 40 0 7 1000
Magnitude is 75.59 ohms
```

5.6 THỰC HÀNH

Q5.1 Chương trình 5.18 có chứa một lỗi. Một kết quả chạy thử làm mẫu từ chương trình này được giới thiệu trong chạy thử 5.8 (lỗi đã làm cho kết quả ở đầu ra trở nên không thể dự đoán và có thể cho kết quả đầu ra khác nhau trên các máy tính khác nhau). Xác định lỗi và sửa chữa để chương trình chạy đúng.

Chương trình 5.18

```
/* prog5_18.c                                          */
/* Program to determine impedance of an capacitor     */

#include  <stdio.h>
#include  <math.h>

#define   PI 3.14159

int   main(void)
{
float  f,C,Xc;
  puts("Program to determine impedance of an capacitor");
  printf("Enter f and C >>>");

  scanf("%f %f",&f,&C);

  Xc=calc_Xc(f,C);

  printf("Impedance is %f ohms\n",Xc);
  return(0)
}
calc_Xc(float f, float 1)
{
return(1/(2*PI*f*1));
}
```

Chạy thử 5.8

```
Program to determine impedance of a capacitor
Enter f and C >>> 1e3   1e-6
Floating point error:Divide by 0.
```

Q5.2 Chương trình 5.19 có chứa một lỗi; hãy xác định và sửa chữa lỗi. Chạy thử 5.9 là một thí dụ làm mẫu (như trong Q5.1, chương trình sẽ đưa một đầu ra không thể dự đoán).

Chương trình 5.19

```
/* prog5_19.c                                                    */
/* Program to determine Impedance of an inductor                 */
#define  PI   3.14159
#include <stdio.h>
#include <math.h>
float calc_X1(int f,int 1);
int    main(void)
float  f,L,X1;
   puts("Program to determine impedance of an inductor");
   printf("Enter f and L >>> ");
   scanf("%f %f",&f,&L);
   X1=calc_X1(f,L);
   printf("Impedance is %f ohrns\n",X1);
   return (0)
}
float calc_X1(int f,int 1);
{
   return(2*PI*f*1);
}
```

Chạy thử 5.9

```
Program to determine impedance of an inductor
Enter f and L >>>     1e3   1e-3
Impedance is 0.000000 ohms
```

Q5.3 Hãy sử dụng lại Q4.12 bằng cách sử dụng các hàm để xác định điện áp nhiễu và tỷ số giữa tín hiệu và nhiễu (SNR). Sử dụng các kết quả đó để xác định SNR cho dải thông từ 1 Hz lên tới 10 GHz trong các bước lớn hơn nhau mười lần.

Q5.4 Hãy viết một chương trình xác định trở kháng của mạch RC nối tiếp ứng với một giá trị tần số được nhập vào. Tham khảo thêm chương trình 5.12.

Q5.5 Hãy lặp lại Q5.4 với mạch LC nối tiếp.

Q5.6 Hãy viết các hàm lôgic Boole với bốn cổng số sau đây:

```
AND3(A,B,C)
OR3(A,B,C)
NAND3(A,B,C)
NOR3(A,B,C)
```

Hãy bổ sung các hàm này vào thư viện của các hàm Boole đã được tạo ra (thí dụ *boole.h*). Tham khảo thêm chương trình 5.9.

Q5.7 Hãy sử dụng các hàm được tạo ra trong Q5.6 để xác định bảng giá trị chân lý cho các phương trình sau.

$$Z = \overline{\overline{A + B + C} + (\overline{B}.C.D) + A}$$

$$Z = \overline{\overline{A.C.D}.(\overline{C}.D).B}$$

$$Z = \overline{\overline{A.B.C}.(A + B + C).(\overline{B}.C.D)}$$

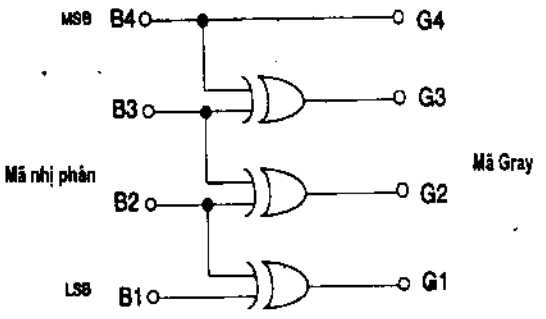
Q5.8 Hãy lặp lại Q4.3 bằng cách sử dụng các hàm được tạo ra trong Q5.6.

Q5.9 Hãy tạo ra một hàm, bổ sung một cổng EX-OR (hoặc-loại trừ) hai lối vào. Kết quả chạy thử 5.10 là một thí dụ được chọn làm mẫu.

Chạy thử 5.10

```
A     B     Z
0     0     0
0     1     1
1     0     1
1     1     0
```

Q5.10 Hãy sử dụng hàm được tạo ra trong Q5.9 để viết một chương trình chuyển đổi mã nhị phân 4 bit thành mã Gray. Hình 5.17 giới thiệu sơ đồ của một bộ biến đổi mã Gray 4 bit, và chạy thử 5.11 là một thí dụ được chọn làm mẫu.



Hình 5.17: **Bộ đổi mã nhị phân Gray.**

Chạy thử 5.11

Nhị phân (BINARY)				Gray			
B4	B3	B2	B1	G4	G3	G2	G
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	0	1	1	0	1

Q5.11 Hãy viết một hàm xác định bảng giá trị chân lý cho một cổng EX-OR có 3 lối vào.

Q5.12 Hãy sửa đổi chương trình 5.17 để các giá trị không hợp lệ của các linh kiện và tần số không thể nhập vào được. Giới hạn hợp lệ của các giá trị được chỉ ra dưới đây.

Tham số	Mức tối thiểu	Cực đại
R	0 Ω	1 MΩ
C	1 pF	100 mF
L	0 H	100 mH
f	1 Hz	10 GHz

Q5.13 Hãy viết lại chương trình 5.17 sao cho trở kháng đặc trưng được cho dưới dạng độ lớn và góc pha. Góc của trở kháng (tính bằng radian) có thể tính theo công thức:

$$\langle Z \rangle = \frac{1}{2}.\left[\tan^{-1}\left[\frac{\varpi.L}{R}\right] - \tan^{-1}\left[\frac{\varpi.C}{G}\right]\right]$$

Q5.14 Hãy viết một macro bằng cách sử dụng `#define` để xác định nhiều hơn hai số nguyên. Chẳng hạn:

```
#define _max(_A,_B)...
```

Kiểm tra macro bằng cách gọi chúng từ một chương trình. Chẳng hạn:

```
printf("maximum is %d",_max(c,d));
```

Khi hàm này đã được kiểm tra thì đặt nó vào một tệp đầu mục (thí dụ *main.h*) và đưa tệp này vào trong chương trình.

Q5.15 Hãy lặp lại Q5.14 cho trường hợp một hàm cực tiểu.

Chương 6
CON TRỎ

Nếu một công ty gửi một bản kê khai cho một người nhưng không thông báo cho người nhận biết địa chỉ chính xác để gửi trả lại thì sau đấy sẽ không thể nhận lại bản đã được kê khai (trừ khi người nhận bản khai đã biết được địa chỉ). Các biến gửi tới các hàm hoạt động theo cách hoàn toàn tương tự. Nếu như hàm không biết một biến đang ở đâu (nghĩa là địa chỉ của biến trong bộ nhớ) thì hàm không thể thay đổi được nội dung của nó.

Một chương trình sử dụng dữ liệu đã được cất giữ bởi các biến. Chúng được gán cho một vị trí duy nhất trong bộ nhớ, số các byte được sử dụng phụ thuộc vào kiểu dữ liệu của chúng. Chẳng hạn, kiểu dữ liệu `char` sử dụng 1 byte, kiểu `int` thường sử dụng 2 hoặc 4 byte, và `float` thường sử dụng 4 hoặc 8 byte. Mỗi vị trí trong bộ nhớ chứa một byte và có một địa chỉ duy nhất được cố kết với nó (cụ thể là địa chỉ nhị phân). Thông thường, địa chỉ này được chỉ rõ bằng một giá trị thập lục phân bởi vì có thể dễ dàng biến đổi thành địa chỉ nhị phân. Bản đồ bộ nhớ trên hình 6.1 chỉ ra cho thấy ba biến: `value1`, `value2` và `ch` có thể được phân bố như thế nào trong bộ nhớ. Sơ đồ này đã thừa nhận rằng kiểu `float` sử dụng 4 byte, còn `int` sử dụng 2 byte. Chương trình dịch, trong trường hợp này, đã xếp `value1` vào địa chỉ từ 100h tới 103h, `value2` ở 104h và 105h, và `ch` được xếp vào 106h. Chỗ bắt đầu của địa chỉ biến trong bộ nhớ có thể được mô tả như một con trỏ bộ nhớ vào biến. Một biến con trỏ được sử dụng để lưu giữ một địa chỉ bộ nhớ.

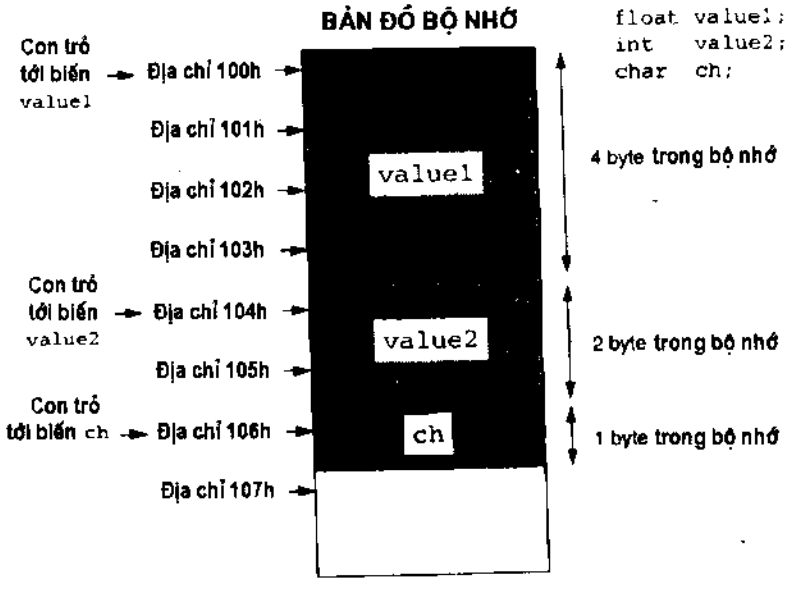
Nội dung của các biến gửi cho một hàm có thể bị thay đổi bằng cách chuyển giao con trỏ trong danh sách các tham số. Phương pháp này đòi

hỏi phải gửi địa chỉ ô nhớ ít hơn là sao chép (copy) giá trị của biến. Ký hiệu (&) dùng làm tiền tố (đặt trước) để chỉ rõ một con trỏ. Cách viết này có thể được hiểu như là muốn diễn đạt *địa chỉ của*:

```
&variable_name {địa chỉ của variable_name}
```

Một con trỏ tới biến sẽ lưu trữ địa chỉ tới byte đầu tiên của vùng phân bố cho biến. Một dấu sao (*) đặt trước một con trỏ được sử dụng để truy nhập nội dung của vị trí được chỉ tới. Số của byte truy nhập sẽ phụ thuộc vào kiểu dữ liệu của con trỏ. Toán tử * có thể được hiểu như là *ở địa chỉ*:

```
*ptr {giá trị lưu giữ ở địa chỉ chỉ rõ bởi ptr}
```

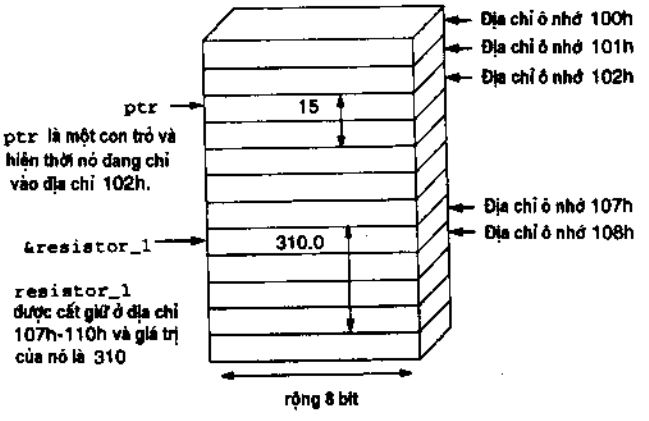


Hình 6.1: **Thí dụ về bản đồ bộ nhớ.**

Hình 6.2 giới thiệu một thí dụ về một bản đồ bộ nhớ. Biến `resistance_1` có giá trị `310,0` và được cất giữ ở một vùng ô nhớ bắt đầu từ `107h`. Nếu kiểu dữ liệu là một *short integer*[*)] thì nó sẽ chiếm 2 byte trong bộ nhớ (thí dụ `107h` và `108h`), nếu như kiểu dữ liệu là dấu

[*)] Một kiểu nguyên có giá trị trong vùng từ -32.768 đến 32.767 với yêu cầu về bộ nhớ là 2 byte (ND).

phẩy động thì nó có thể chiếm 4 byte trong bộ nhớ (thí dụ 107h tới 110h). Đồng thời bản đồ bộ nhớ chỉ ra rằng con trỏ ptr chỉ vào vị trí ô nhớ ở 102h. Giá trị cất giữ ở vị trí này là 15; *ptr truy nhập tới nội dung của nó. Sự khai báo của con trỏ xác định kiểu dữ liệu của con trỏ và như vậy xác định số byte được sử dụng để lưu trữ giá trị ở địa chỉ mà con trỏ chỉ.



Hình 6.2: **Một thí dụ về bản đồ bộ nhớ.**

6.1 CON TRỎ VÀ HÀM

Trong chương trước ta đã thấy rằng một giá trị đơn được chuyển giao ra ngoài một hàm qua đầu mục hàm. Để chuyển giao các giá trị ra ngoài qua danh sách tham số thì địa chỉ của biến đã được chuyển giao; cách chuyển giao này được so sánh với *gọi bằng tham chiếu* (call by reference). Để khai báo một con trỏ thì kiểu dữ liệu đã được chỉ rõ và tên con trỏ có tiền tố (được đặt trước) là một dấu sao. Sau đây là khuôn mẫu tổng quát:

```
type_def *ptr_name;
```

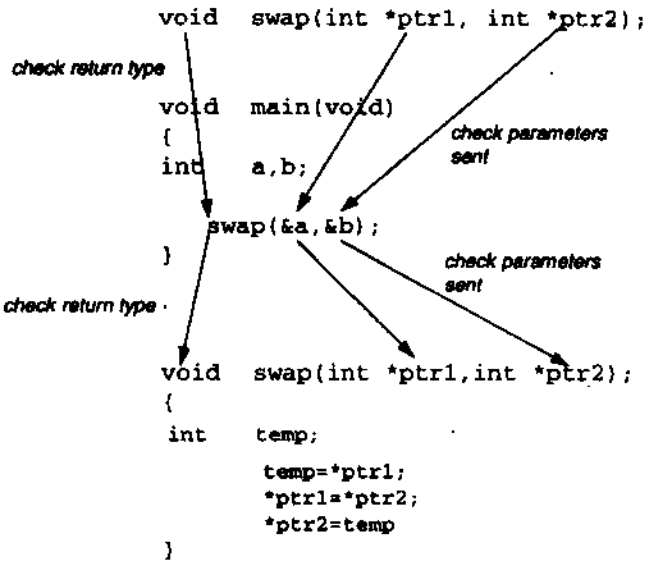
Trong trường hợp này ptr_name là tên của con trỏ. Nội dung của biến ở địa chỉ này có thể được truy nhập bằng cách sử dụng *ptr_name. Khi một hàm bị một biến sửa đổi thì một con trỏ được gửi tới địa chỉ. Chẳng hạn, khi biến được sửa đổi là value thì tham số chuyển giao là &value.

Chương trình 6.1 giới thiệu một thí dụ về hàm, tráo đổi (swap) nội dung của hai biến (a và b). Hình 6.3 cho thấy chương trình dịch kiểm tra như thế nào đối với các tham số chuyển giao tới hàm và kiểu trả lại. Hàm đặt làm mẫu (prototype), trong trường hợp này, chỉ rõ rằng tham số gửi là những con trỏ tới những giá trị nguyên và kiểu trả lại là void. Chương trình dịch kiểm tra các tham số gửi tới hàm là những con trỏ số nguyên và rằng không có gì được gán vào giá trị trả lại từ các hàm.

Chương trình 6.1

```
/* prog6_1.c                                                    */
#include <stdio.h>

void    swap(int *ptr1,int *ptr2);
int     main(void)
{
int     a,b;

  a=5; b=6
  swap(&a,&b); /* send addresses of a and b     */
  Printf("a= %d  b= %d \n",a,b);
  return(0);
}
void    swap(int *ptr1,int *ptr2)
{
    /* ptr1 and ptr2 are pointers (addresses) */
int     temp;

  temp = *ptr1;
  *ptr1 = *ptr2;
  *ptr2 = temp;
}
```

Một thí dụ về hàm C tiêu chuẩn, sử dụng cách gọi bằng tham chiếu là scanf() ở đây một con trỏ được chuyển giao đối với mỗi biến. Cách gọi này cho phép hàm thay đổi nội dung của nó.



Hình 6.3: **Kiểm tra chương trình dịch dùng cho chương trình 6.1.**

6.2 MỘT SỐ THÍ DỤ

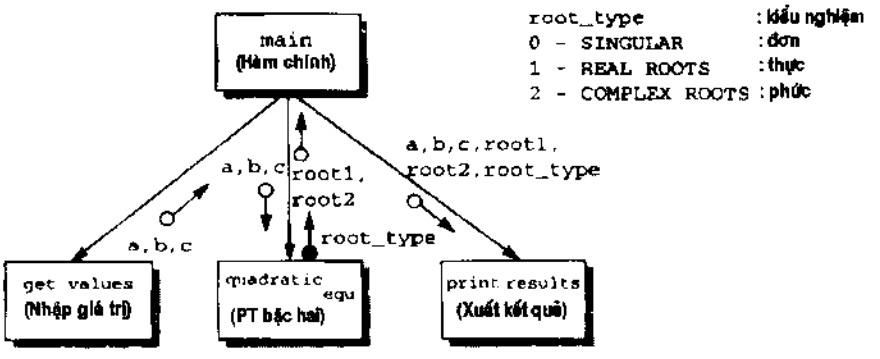
6.2.1 CÁC PHƯƠNG TRÌNH BẬC HAI

Chương trình 6.2 xác định các nghiệm của một phương trình bậc hai. Hàm `get_value()` có các biến `a`, `b` và `c`; các biến này được chuyển giao như những con trỏ. Với `get_value()` hàm `scanf()` không thể đòi hỏi ký hiệu VÀ (&) làm tiền tố; các tham số biến đã là những con trỏ.

Hàm để xác định nghiệm của một phương trình bậc hai là `quadratic_equ()`. Hàm này trả lại loại nghiệm (thí dụ: đơn, thực hoặc phức) qua đầu mục hàm và chuyển các nghiệm của phương trình qua danh sách tham số bằng cách sử dụng những con trỏ. Loại nghiệm được trả lại có thể được đối chiếu như một cờ trả lại; cờ này đã được đặt bằng cách sử dụng khai báo bằng `enum`. Có ba trạng thái có thể: `SINGULAR` (giá trị bằng 0) `REAL ROOTS` (giá trị bằng 1) và `COMPLEX_ROOTS` (giá trị bằng 2). Chương trình lúc đó sử dụng cờ để xác định xem các nghiệm đã được hiển thị như thế nào. Nếu là nghiệm đơn thì `printf_results()` xuất ra một giá trị đơn là `root1`; nếu là

nghiệm thực thì hai giá trị `root1` và `root2` được xuất ra; còn nếu các nghiệm là phức thì hàm sẽ xuất ra các nghiệm dưới dạng `root1 +/-j root2`.

Hình 6.4 giới thiệu một sơ đồ cấu trúc cơ bản của chương trình này. Cờ trả lại từ hàm `quadratic_equ()` được diễn tả bằng một mũi tên với một vòng tròn ở phía sau.



Hình 6.4: **Sơ đồ cấu trúc của chương trình 6.2.**

Chương trình 6.2

```
/* prog6_2.c                                                    */
#include <stdio.h>
#include <math.h>
enum (SINGULAR, REAL_ROOTS,COMPLEX_ROOTS) quad_roots;

void    get_values (float *ain, float *bin, float *cin);
int     quadratic-equ(float a, float b, float c, float *r1,
                float *r2);
void     print_results(int    r_type,float a,float b,float c,
                float r1, float r2);

int     main(void)
{
float    a,b.c,root1,root2;
int     root_type;
  get_values(&a,&b,&c);
  root_type=quadratic_equ(a,b,c,&root1,&root2);
```

```
  print_results(root_type,a,b,c,root1,root2);
  return(0);
}
void   get_values(float *ain,float *bin,float *cin)
{
  printf("Enter a, b and c >>");
  scanf("%f %f %f",ain,bin,cin);
  /*ain,bin and cin are pointers*/
}
int     quadratic_equ(float a, float b, float c,
                float *r1, float *r2)
{
   if (a==0)
   {
      *r1=-c/b;
      return(SINGULAR);
}
else if ((b*b)>(4*a*c))
{
      *r1=(-b+sqrt(b*b-4*a*c))/(2*a);
      *r2=(-b-sqrt(b*b-4*a*c))/(2*a);
      return(REAL_ROOTS);
}
  else  if ((b*b)<(4*a*c))
  {
  *r1=-b/(2*a);
  *r2=sqrt(4*a*c-b*b)/(2*a);
  return(COMPLEX_ROOTS);
  }
  else
  {
     *r1=-b/(2*a);
     return(SINGULAR);
   }
}
void   print_results(int r_type,float a,float b,
                  float c,float r1,float r2)
```

```
{
   printf("Quadratic equation %8.3f x^2 + %8.3f x +
   if (r_type==SINGULAR)
   printf("Singular root of %8.3f\n",r1);
   else if (r_type==REEL_ROOTS)
   printf("Real roots of %8.3f and %8.3f\n",r1,r2);
   else
   printf ("Complex root of %8.3f +/-j %8.3f\n",r1,r2)
}
```

Kết quả chạy thử 6.1 chỉ ra một thí dụ làm mẫu.

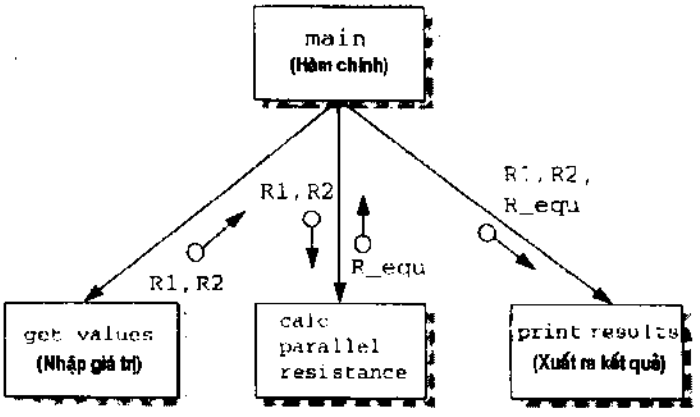
Chạy thử 6.1

```
Enter a, b and c >> 2 1 1
Quadratic equation    2.000 xA2 + 1.000 X + 1.000
Complex root of    -0.250 +/-j 0.661

Enter a, b and c >> 1 -2 -3

Quadratic equation     1.000 x^2 +  -2.000 x +-3.000
Real -roots of    3.000 and -1.000

Enter a, b and c >> 1 2 1

Quadratic equation 1.000 xA2 + 2.000 x + 1.000
Singular root of    -1.000
Tương đương nhị phân là 00100001
```

6.2.2 ĐIỆN TRỞ TƯƠNG ĐƯƠNG CỦA CÁC ĐIỆN TRỞ MẮC SONG SONG

Chương trình 6.3 sử dụng các con trỏ để xác định điện trở tương đương của hai điện trở mắc song song. Sơ đồ cấu trúc cơ bản, được giới thiệu trên hình 6.5 đã chỉ ra rằng `get_values()` trả lại các biến R1 và R2; để thay đổi giá trị của chúng, chúng được gửi như là các con trỏ. Sơ đồ này cũng chỉ ra rằng các biến gửi tới `calc_paralell_res()` là `R1`,

R2 và R_equ đều được trả lại. Sau đó các biến R1, R2 và R_equ được chuyển giao vào print_results().



Hình 6.5: **Sơ đồ cấu trúc của chương trình 6.3.**

Chương trình 6.3

```
/* prog6_3.c                                                    */
#include <stdio.h>
void   get_values(float *r1,float *r2);
void   get_parallel_res(float r1,float r2,float *r_e);
void   print_results(float r1,float r2,float r_e);

int   main(void)
{
float R1,R2,R_equ;

  get_values(&R1,&R2);
  get_parallel_res(R1,R2,&R_equ);
  print_results(R1,R2,R_equ);
  return(0);
}
void    get_values(float *r1,float *r2)
{
   do
```

```
{
    printf("Enter R1 >>");
scanf("%f",r1); /*r1 is already a pointer no need for &r1 */
    if (*r1<0) puts("INVALID: re-enter");
     } while (*r1<0);

     do
     {
       printf("Enter R2 >>");
       scanf (" %f", r2) ;
       if (*r2<0) puts("INVALID: re-enter");
    } while (*r2<0);
}
void get_parallel_res(float r1,float r2,float *r_e)
{
   *r_e=1/(1/r1+1/r2);
}
void   print_results(float r1,float r2,float r_);
{
printf("Parallel resistors %8.3f and %8.3f ohm\n",r1,r2);
printf("Equivalent resistance is %8.3f ohm\n",r_e);
}
```

Chạy thử 6.2 giới thiệu các lần thử nghiệm với từng loại nghiệm.

Chạy thử 6.2

```
Enter R1 >> 1000
Enter R2 >> 800
Parallel resistors 1000.000 and   800.000 ohm
Equivalent resistance is 444.444 ohm
```

6.2.3 TRỞ KHÁNG CỦA MẠCH RL

Chương trình 6.4 tính toán trở kháng vào của mạch RL nối tiếp khi tần số được quét.

Chương trình 6.4

```
/* prog6_4.c                                                    */
#include <stdio.h>
#include <math.h>

#define   PI   3.14159

void  get_values (float *fe,int *fn,float *r, float *1);
void    calc_impedance-XL,(float f,float r,float 1,float
                      *zmag,float *zang);

int   main (void)
{
float    f,fend,R,L,Zmag,Zangle;
int fsteps;
     get_values (&fend, &fsteps, &R, &L) ;

     for (f=0;f<fend;f+=fend/fsteps)
     {
     calc_impedance_XL(f,R,L,&Zmag,&Zangle);
     printf("%8.2f %8.3f %8.3f\n",f,Zmag,Zangle);
     }
     printf("Press ENTER to continue >>");
     getchar();
     return(0);
}
void   get_values(float *fe,int *fn,float *r,float *1)
{
     printf("Enter end frequency >>") ;
     scanf("%f",fe);
     printf("Enter number of frequency steps >>"),
     scanf("%d",fn);
     printf("Enter resistance >>");
     scanf("%f", r,;
     Printf("Enter inductance >>");
```

```
    scanf("%f",1);
}
void   calc_impedance_XL(float f,float r,float 1,
                  float *zmag,float *zang)
 {
 float  x1;

    X1=2*PI*f*1;
    *zmag=sqrt((X1*X1)+(r*r));
    *zang=atan(x1/r);
 }
```

Kết quả chạy thử 6.3 là một thí dụ được giới thiệu làm mẫu.

Chạy thử 6.3

```
Program to detemine impedance of an RL circuit
Enter fmax and number of freq steps >> 1e6 20
Enter R and L >> 1000 1e-3
 Frequency        Impedance
              Magnitude   Angle

      0.00    1000.00      0.00
  50000.00    1048.19     17.44
 100000.00    1181.01     32.14
 150000.00    1374.14     43.30
 200000.00    1605.97     51.49
 250000.00    1862.00     57.52
 300000.00    2133.79     62.05
 350000.00    2415.30     65.55
 400000.00    2704.91     68.30
 450000.00    2999.06     70.52
 500000.00    3296.91     72.34
 550000.00    3597.53     73.86
 600000.00    3900.29     75.14
 650000.00    4204.72     76.24
 700000.00    4510.48     77.19
 750000.00    4817.32     78.02
 800000.00    5125.06     78.75
 850000.00    5433.52     79.39
```

```
900000.00    5742.61    79.97
950000.00    6052.21    80.49
Press anykey to continue  >>>
```

6.2.4 CHƯƠNG TRÌNH QUAN SÁT BỘ NHỚ

Chương trình 6.5 hiển thị nội dung của bộ nhớ RAM trong khi chương trình đang được thực hiện và được gọi là chương trình quan sát bộ nhớ (*Viewer*). Mỗi địa chỉ trong bộ nhớ chứa một byte, sao cho con trỏ `char` được đặt để chỉ vào từng vị trí; khai báo `char*ptr` đã được sử dụng cho mục đích này. Con trỏ này không có vị trí trong bộ nhớ khi nó được khai báo, nhưng ở thời điểm bắt đầu chương trình nó được đặt vào ô nhớ đầu tiên `0x00(ptr=0x00)`.

Sau khi nội dung của mỗi ô nhớ đã được đọc, con trỏ gia tăng tới địa chỉ tiếp theo trong bộ nhớ. Khi sự khai báo là một con trỏ đặc trưng nó sẽ được tăng thêm một byte (vấn đề này sẽ được bàn luận nhiều hơn trong chương tiếp theo).

Chương trình đọc 64 byte từ bộ nhớ và hiển thị, chúng chứa đựng ký tự mã ASCII. Sau đó chương trình đợi người dùng nhập vào ký tự. Nếu như ký tự được nhập vào là 'x' thì chương trình sẽ thoát ra; một ký tự khác bất kỳ sẽ làm cho chương trình tiếp tục hiển thị nội dung bộ nhớ. Một vài ký tự điều khiển đặc biệt dùng để đẩy dòng (line feed), xuống dòng (new line), form-feed v. v... đều không được in ra. Dành cho mục đích này, một số ký tự trong các ký tự loại này được hiển thị bằng một dấu '?'. Các macro CR (Carriage Return), VT (Vertical Tab) và BS (Backspace) v. v... được sử dụng để định nghĩa các ký tự này.

Chương trình 6.5

```
/* prog6_5.c                                          */
#include     <stdio.h>
#define  BELL   7
#define  BS     8
#define  HT     9
#define  LF     10
#define  VT     11
```

```
#define  FF     12
#define  CR     13

int    main(void)

char   *ptr,ch=NULL;
int    i;

  puts("Memory viewer program Ver 1.00");
  puts("Enter an 'x' to exit the program");
  ptr=0x00;

do
{

  printf("%4p >> ",ptr); /* display memory pointer */
  for (i=0;i<64;i++)
  {

    if ((*ptr!=BELL) (*ptr!=CR) (*ptr!=VT)
      (*ptr!=BS) (*ptr!=FF) (*ptr!=LF))
    putchar(*ptr);
else putchar('?'); /* display control character */
ptr++; /* move one byte in memory.  */
}
  printf("\n");
  ch=getchar();

    while (ch!='x,); /* enter an lxl to exit */
  return(0);
}
```

Chương trình 6.6 có một vài điểm đáng chú ý. Chương trình này hiển thị 24 dòng nội dung bộ nhớ trước khi người dùng được nhắc để

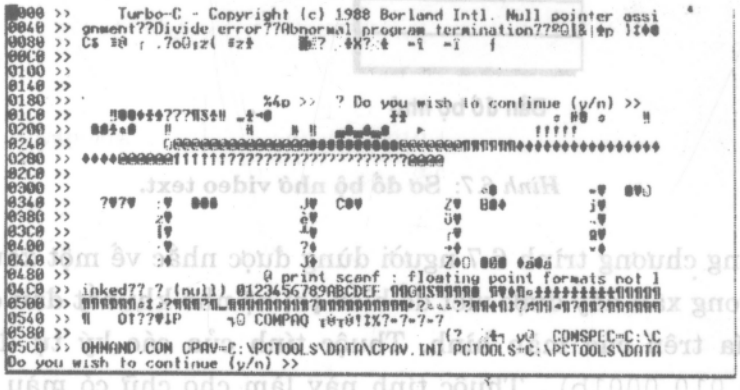
tiếp tục (số dòng đặt được bằng hằng số SCREEN_SIZE còn số ký tự trên một đường bằng COLUMN_SIZE).

Chương trình 6.6

```
/* prog6_6.c                                                    */
#include  <stdio.h>
#include  <ctype.h>
     /* required for tolowero function */

#define  BELL   7
#define  BS     8
#define  HT     9
#define  LF     10
#define  VT     11
#define  FF     12
#define  CR     13
#define  SCREEN-SIZE  24
#define  COLUMN-SIZE  64

int  main(void)
{
char *ptr,ch=NULL; /* initialise ch as a NULL character */
int i,i;
  puts("Memory viewer program Ver 1.01");
  ptr=0;

  do
  {
    for (j=0;j<SCREEN_SIZE;j++)
    {

      printf("%4p >> ",ptr);
      for (i=0;i<COLUMN_SIZE;i++)
      {
    if ((*ptr!=BELL) && (*ptr!=CR) (*ptr!=VT)
        && (*ptr!=BS) (*ptr!=FF) && (*ptr!=LF))
```

```
            putchar(*ptr);
        else {
            putchar('?');
        }
        ptr++;
    }
    printf("\n");
}
    printf("Do you wish to continue (y/n) >> ");
    ch=tolower(getcharo);

  while (ch=='Y');

return(0);
{
```

Hình 6.6 chỉ ra một thí dụ khi chạy chương trình trên máy tính PC tương thích IBM. Ký tự '?' đã được thay thế bằng các ký tự điều khiển đặc biệt, chẳng hạn như nạp dòng (line feed), dấu trống tab v. v...

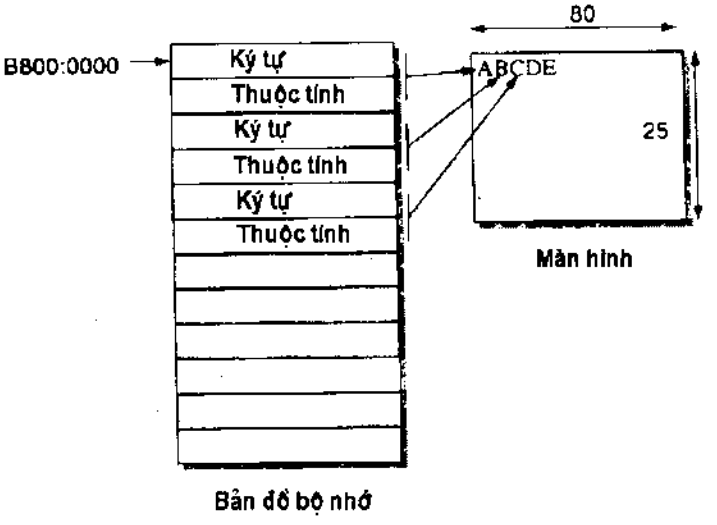


Hình 6.6: **Màn hình khi chạy thử chương trình 6.6.**

6.2.5 TRUY NHẬP BỘ NHỚ TEXT VIDEO CỦA MÁY TÍNH CÁ NHÂN.

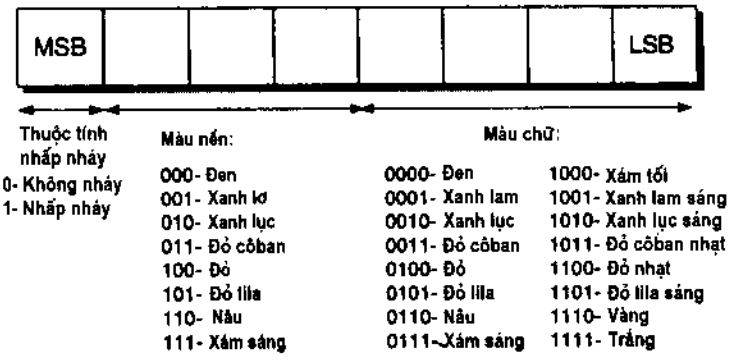
Các chương trình trong đoạn này chỉ làm việc với một hệ máy tính cá nhân.

Máy tính cá nhân (PC) sử dụng một vùng riêng trong bộ nhớ để lưu giữ tất cả các ký tự xuất hiện trên màn hình. Với màn hình màu, vùng nhớ này bắt đầu ở địa chỉ `B800:0000`. Mỗi ký tự có một thuộc tính text gắn với nó. Thuộc tính này quyết định các màu của chữ và của nền trên màn hình, cho dù có nhấp nháy hoặc không. Một màn hình tiêu biểu có 80 cột và 25 hàng. Như vậy, tổng cộng có 4000 byte được sử dụng để lưu giữ tất cả các ký tự (80*25 ký tự và 80*25 thuộc tính). Hình 6.7 cho thấy các ký tự và thuộc tính được sắp xếp trong bộ nhớ như thế nào. Các định nghĩa bit thuộc tính được chỉ ra trên hình 6.8.



Hình 6.7: **Sơ đồ bộ nhớ video text.**

Trong chương trình 6.7 người dùng được nhắc về một xâu text. Các ký tự trong xâu này được viết để hiển thị bộ nhớ khi bắt đầu ở vị trí bên trái phía trên của màn hình. Thuộc tính của các ký tự được cho là `0x21(0 010 0001b)`. Thuộc tính này làm cho chữ có màu xanh lam, còn nền có màu lục (lá cây) và không nhấp nháy. Sau khi mỗi ký tự và thuộc tính đã được viết vào bộ nhớ, con trỏ được gia tăng thêm 2 để di chuyển tới ký tự tiếp theo trong bộ nhớ. Hàm `Strlen()` xác định số của các ký tự trong xâu (vấn đề này sẽ được bàn luận chi tiết hơn trong chương 8).



Hình 6.8: **Các thuộc tính của ký tự.**

Máy tính cá nhân mặc định là có con trỏ địa chỉ 16 bit (trong hầu hết các trường hợp). Để truy nhập tới phạm vi cao hơn của bộ nhớ PC cần có một con trỏ địa chỉ 32 bit. Khi đó con trỏ được khai báo như là một con trỏ far (far pointer). Kết quả này đạt được nhờ khai báo nó như `char far *ptr`.

Chương trình 6.7

```
/* prog6_7.c                                                    */
#include <stdio.h>
#include <string.h>
int main(void)
  {
  char far *ptr;
  char     str[BUFSIZ];
  int      i;
  ptr=(char far *)0xb8000000; /* memory address B800:0000 */

  printf("\n Enter string for diret memory access >>");
  gets(str);

  for(i=0;i<strlen(str);i++)
  {
    *ptr=str[I];     /* write character                        */
    /*(ptr+1)=0x21;   /* character attribute 0 010 0001         */
    /* B XXX YYYY B - Blink, XXX - B/G Colour YYYY - F/G Colour */
```

```
    /* Colour 0-Black, 1 Blue, 2 Green, etc.      */
    ptr+=2;
  }
  puts ("") ;
  return(0);
}
```

Khi chương trình này chạy, xâu được nhập vào sẽ xuất hiện ở góc bên trái phía trên màn hình. Ký tự có màu xanh lam trên nền màu xanh lục.

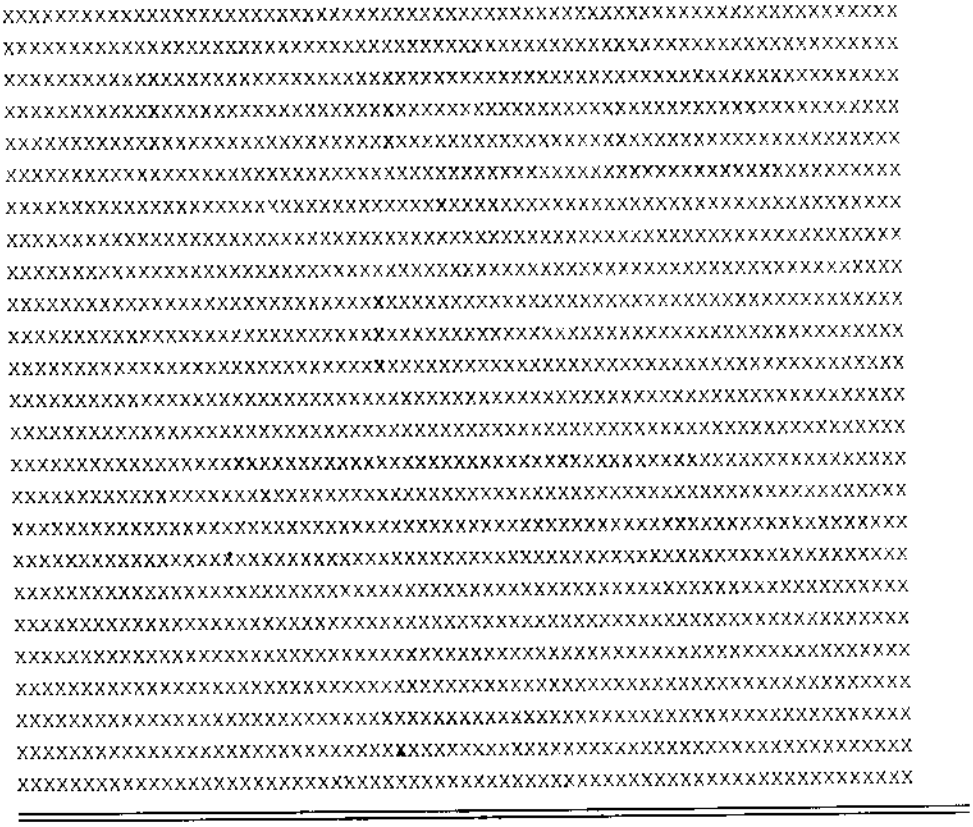
Chương trình 6.8 làm đầy bộ nhớ text video bằng ký tự đơn. Ký tự sử dụng trong trường hợp này là `'X'` và thuộc tính text là 0x70 (0111 0000 b). Thuộc tính này tạo ra màu trắng cho nền còn chữ có màu đen. Số của các hàng và cột được xác lập bằng các macro `COLUMNS` và `ROWS`. Thí dụ làm mẫu được giới thiệu trong chạy thử 6.4.

Chương trình 6.8

```
/* prog6_8.c                                          */
#define     COLUMNS    80
#define     ROWS       25

#include <stdio.h>

int main(void)
{
char far *ptr;
int      i;

  ptr=(char far *)0xb8000000; /*Memory address B800:0000 */
  for          (i=0;i<COLUMNS*ROWS;i++)
  {
    *ptr='X';       /* fill with an'X'                 */
    *(ptr+1)=0x70, /* 0 111 0000                       */
    ptr+2;         /* move onto next character in memory  */
}
  getchar();
```

```
    puts("");
    return(0);
}
```

Chạy thử 6.4

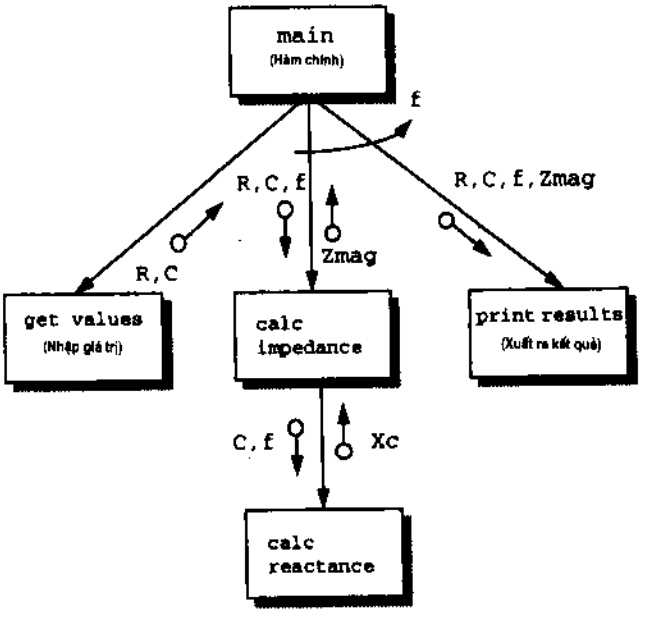


6.3 THỰC HÀNH

Q6.1 Hãy viết một chương trình xác định trở kháng, tương ứng với giới hạn tần số được nhập vào, cho những mạch sau đây:

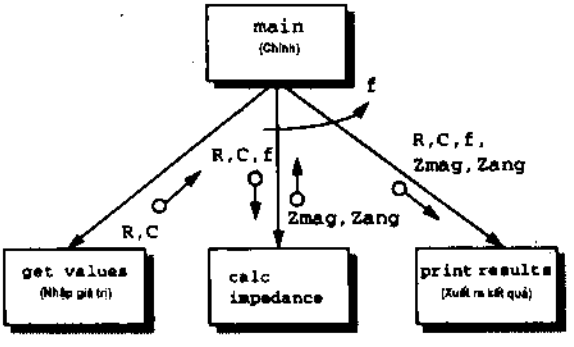
(a) Mạch RC nối tiếp;
(b) Mạch RC song song;
(c) Mạch RL song song

Hình 6.9 giới thiệu sơ đồ cấu trúc cơ bản cho mạch RC nối tiếp. Hãy tham khảo thêm chương trình 6.4.



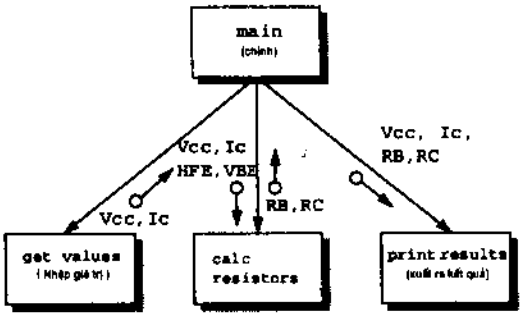
Hình 6.9: **Sơ đồ cấu trúc dùng cho Q6.1.**

Q6.2 Hãy lặp lại Q6.1, nhưng xác định độ lớn và góc pha của mạch. Hình 6.10 giới thiệu sơ đồ cấu trúc mức cao (top level) dùng cho chương trình này.



Hình 6.10: **Sơ đồ cấu trúc cho Q6.2.**

Q6.3 Hãy viết một chương trình xác định các giá trị điện trở cho mạch tranzito thiên áp chỉ bằng một điện trở cực gốc được chỉ ra trong Q4.11. Hình 6.10 giới thiệu sơ đồ cấu trúc của chương trình này.



Hình 6.11: **Sơ đồ cấu trúc dùng cho Q6.3.**

Q6.4 Hãy sửa đổi các chương trình trong chương 5 để tạo ra cấu trúc hoàn hảo hơn. Đáng chú ý là có thể thêm vào các hàm `get_values()` và `printf_results()`. Đồng thời sửa đổi các chương trình này để giá trị tần số bằng 0, không hợp lệ không thể nhập được vào.

(a) Chương trình 5.17;
(b) Chương trình 5.14.

Q6.5 Hãy sửa đổi chương trình 6.8 để người dùng được nhắc nhở địa chỉ khi khởi động chương trình quan sát bộ nhớ.

Nếu như bạn đang có máy tính PC thì hãy giải các bài tập thực hành sau đây:

Q6.6 Hãy viết một chương trình làm đầy màn hình text bằng một ký tự đơn 'z' với nền màu trắng và chữ màu xanh lam bằng cách sử dụng truy nhập video trực tiếp.

Q6.7 Hãy viết một chương trình làm đầy màn hình bằng ký tự được nhập vào và màu cho chữ và cho nền được nhập vào. Thí dụ làm mẫu được giới thiệu trong chạy thử 6.5.

Chạy thử 6.5

```
Nhập vào một ký tự để hiển thị >> c
Chọn màu tiền cảnh
(0)- BLACK(đen) (1)- BLUE(xanh lơ) (2)- GREEN(xanh lục) (3) -
CYAN (đỏ côban)(4)- RED(đỏ)
```

```
>> 4
Chọn màu nền
(0)-BLACK(đen) (1)-BLUE(xanh lơ) (2)-GREEN(xanh lục) (3)-CYAN(đỏ côban) (4)-RED
   (đỏ)
>> 3
```

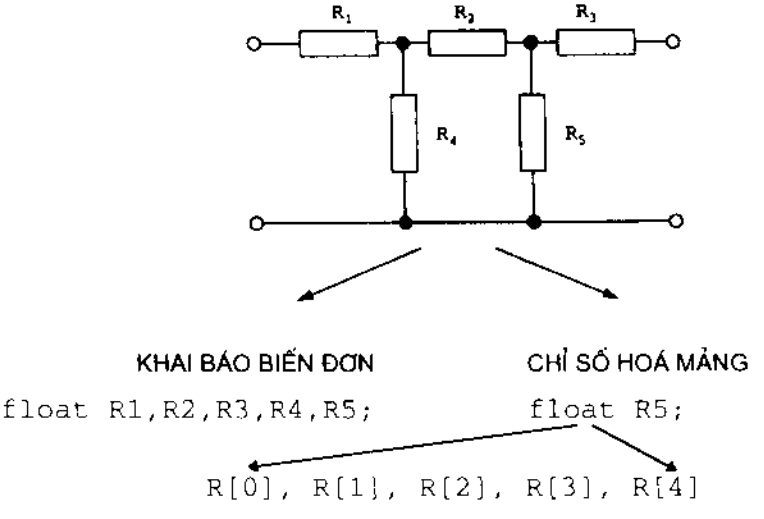
Q6.8 Ngày tháng trong bộ nhớ ROM BIOS trên máy tính cá nhân được lưu trữ ở một địa chỉ khi khởi động tại F000:FFF5 (0xF000FFF5) và sử dụng 8 byte. Hãy viết một chương trình đọc dữ liệu về ngày tháng năm này. Một kết quả chạy thử làm mẫu được chỉ ra trong chạy thử 6.6 (trong trường hợp này 8 ký tự đọc được là 11/11/92.

Chạy thử 6.6

```
BIOS RAM Ngày tháng ghi là 11/11/98
```

Chương 7
MẢNG

Một mảng bao gồm (hay còn gọi là cất giữ) không ít hơn một giá trị, có một kiểu dữ liệu chung, dưới một tên chung. Mỗi giá trị có một vị trí (slot) duy nhất và được so sánh bằng cách sử dụng kỹ thuật chỉ số hóa. Hình 7.1 giới thiệu một mạch với 5 điện trở, có thể được khai báo với chương trình bằng 5 khai báo dấu phảy động đơn giản.



Hình 7.1: **Các biến đơn giản dựa vào sự chỉ số hóa mảng.**

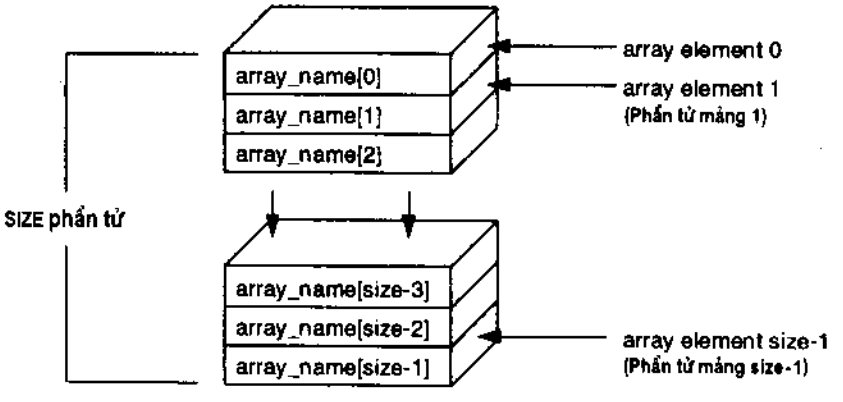
Nếu như những biến điện trở cần được chuyển giao vào trong một hàm thì toàn bộ 5 biến cần phải được chuyển giao qua danh sách tham số. Một phương pháp tinh xảo hơn sử dụng các mảng để cất giữ tất cả các

giá trị dưới một tên chung (trong trường hợp này là R). Nhờ vậy một biến mảng đơn lúc đó có thể được chuyển giao vào trong bất kỳ hàm nào sử dụng nó.

Sự khai báo của một mảng chỉ rõ kiểu dữ liệu, tên và số lượng của các phần tử trong mảng được chỉ rõ trong những dấu ngoặc vuông ([ ]). Sau đây là khuôn mẫu chuẩn dùng cho khai báo mảng.

```
data_type array_name[size];
```

Hình 7.2 chỉ ra rằng phần tử đầu tiên của mảng đã được chỉ số hoá (đánh số) là 0 và phần tử cuối cùng được đánh số là [SIZE-1]. Chương trình dịch phân bố (cấp phát) bộ nhớ cho phần tử đầu tiên `array_name[0]` tới phần tử mảng cuối cùng `array_name[SIZE_1]`. Số byte được phân bố trong bộ nhớ sẽ là số các phần tử trong mảng nhân với số byte được sử dụng để cất giữ kiểu dữ liệu của mảng.



Hình 7.2: **Các phần tử của mảng.**

Dưới đây xin dẫn ra một vài thí dụ về khai báo mảng.

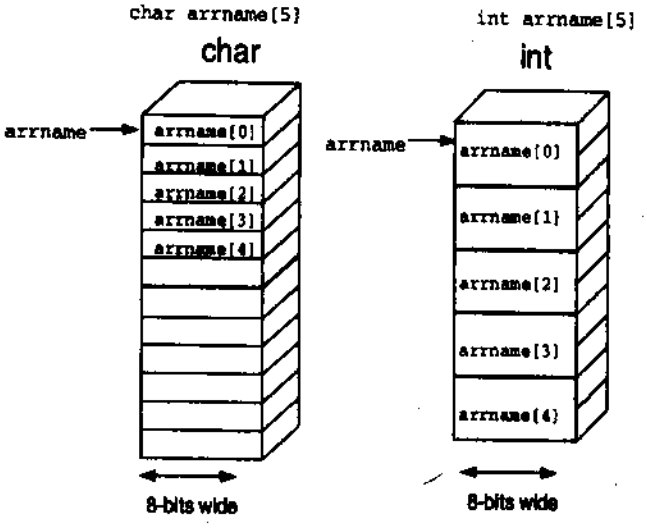
```
int circuit(10);
    /* allocates space for circuit(0)    */
    /* to circuit[9]                     */
float impedance(50);
    /* allocates for impedance(0)to      */
    /* impedance(49)                     */
```

7.1 CON TRỎ VÀ MẢNG

Giữa con trỏ và mảng có một mối quan hệ rất chặt chẽ. Một biến con trỏ cất giữ một địa chỉ bộ nhớ, có thể được sửa đổi, trong khi một tên mảng cất giữ một địa chỉ cố định, đặt cho phần tử đầu tiên trong mảng. Địa chỉ của phần tử đầu tiên của mảng có tên là `arrname` như vậy sẽ là `&arrname[0]`. Bảng 7.1 chỉ ra những thí dụ cho thấy các mảng và con trỏ sử dụng những ký hiệu chỉ số hóa khác nhau như thế nào và nó có thể để trao đổi lẫn nhau như thế nào.

Bảng 7.1: **Mối quan hệ giữa mảng và con trỏ.**

Mảng được dùng	Con trỏ được dùng
`float arr[10];`	`float *arr;` `arr=(float *) malloc(10*sizeof(float));`
`arr[0]`	`*(arr)`
`arr[1]`	`*(arr+1)`
`arr[2]`	`*(arr+2)`
`arr[9]`	`*(arr+9)`



Hình 7.3: **Các phần tử mảng.**

Hình 7.3 mô tả hai loại khai báo mảng đối với arrname. Mỗi loại có năm phần tử: thứ nhất là arrname[0] và cuối cùng arrname[4]. Số byte được phân bố (cấp phát) tới mỗi phần tử phụ thuộc vào việc khai báo kiểu dữ liệu. Một mảng char sử dụng một byte cho mỗi phần tử, trong khi mảng int thông thường chiếm 2 hoặc 4 byte. Tên mảng arrname được đặt cho địa chỉ của phần tử đầu tiên của mảng. Mỗi phần tử bên trong mảng được so sánh với địa chỉ này.

Bảng 7.2 giới thiệu một số thí dụ về các lệnh mảng và con trỏ.

Bảng 7.2: **Thí dụ về các lệnh mảng và con trỏ.**

Lệnh	Mô tả
`int tmp[100];`	`declare an array named tmp with 100 elements`
`tmp [1]=5;`	`assign 5 to the second element of array tmp`
`*(tmp+1) =5;`	`equivalent to previous statement`
`ptr=&tmp[2]`	`get address of third element`

7.2 CHUYỂN GIAO MẢNG TỚI HÀM

Khi biên dịch, chương trình dịch dự trữ không gian bộ nhớ đủ cho tất cả các phần tử trong một mảng và khởi tạo tên mảng cho chỗ bắt đầu của không gian này. Để một hàm có thể sửa đổi, mảng địa chỉ cơ sở được chuyển giao qua danh sách các tham số. Hàm không tự biết số cực đại của các phần tử trong mảng; trừ khi tham số liên quan với số cực đại của các phần tử trong mảng cũng được chuyển giao. Như vậy có khả năng chạy ra khỏi chỗ kết thúc của mảng và bộ nhớ truy nhập không được phân bố tới mảng.

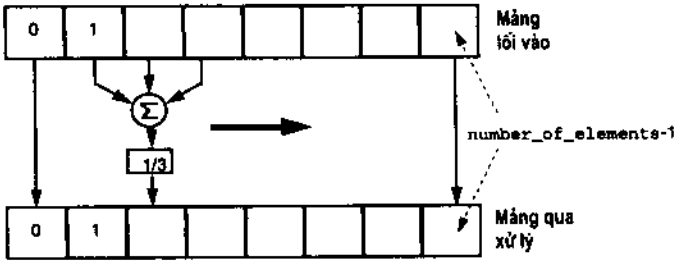
Cách viết được sử dụng để ký hiệu một mảng đang được chuyển giao vào một hàm là những dấu ngoặc vuông, các dấu ngoặc này báo hiệu cho biết đây là một địa chỉ cố định chứ không phải là biến con trỏ. Dưới đây là một thí dụ về sự chuyển giao mảng.

```
float maximum=(int number_of_elements,float arrayname */
{
```

```
float max;
/* number_of_elements maximum number of elements in the array*/
/* this function determines maximum value in an  array        */

   max=arrayname(0);
   for(i=1;i<number_of_elements;i++)
         if (max<arrayname(i)) max=arrayname(i);
   return(max);
}
```

Chương trình 7.1 là chương trình mức trung bình chạy (running avereage) 3 điểm. Loại chương trình này có đáp ứng (responce) của bộ lọc dải thông thấp và có thể lọc lựa các mẫu dữ liệu. Hình 7.4 minh họa đầu ra của hàm trung bình của ba phần tử trong mảng đầu vào như thế nào; kết quả này đạt được bằng cách tạo ra mức trung bình chạy.

Giá trị đầu tiên và cuối cùng của mảng được xử lý sẽ chiếm cùng giá trị như mảng đầu vào bởi vì không có ba giá trị trên đó mức trung bình chiếm.



Hình 7.4: **Các phần tử của mảng.**

Chương trình 7.1

```
/* prog7_1.c                                                  */
#include <stdio.h>

#define MAX 150     /* maximum values in the array */
#define TRUE 1
#define FALSE 0
```

```
void    filter(int n,float array_in[],float array_out[]);
void    get_values(int *n,float array[]);
void    print_values(int n,float array_in[],float array_out[]);

int   main(void)
{
float  input[MAX],output[MAX];/*input values, processed values */
int nvalue.
   get_values(&nvalues,input);
   filter(nvalues,input,output);
   print_values(nvalues,input,output);
   return(0);
 }

void   filrer(int n,float array_in[],float array_out[])
{
int   i;
  array_out[0]=array_in[0];
  array_out[n-1]=array_in[n-1];

  for (i=1;i<n-1;i++)
     array_out[i]=(array_in[i-1]+array_in[i]+array_in[i+1]))/3;
}

void     get_values(int *n,float array[])
/* *n    is the number of elements in the array */
{
int      i,rtn,okay;

   do
   {
   printf("Enter number of values to be processed >>");
   rtn=scanf("%d",n);
   if ((rtn!=1) || (*n<0) || (*n>MAX))
   {
```

```
      printf("Max elements are %d, re-enter\n",MAX);
      okay=FALSE;
    }
    else okay=TRUE;
  } while (!okay);

    for (i=0;i<*n;i++)
  {
   puts("Enter value >> ");
   scanf("%f",&array[i]);
  }
}

void  print_values(int n,float array_in[],float array_out[])
{
int   i;
      printf("Input     Output\n");
      for (i=0;i<n;i++)
        printf(" %6.3f%6.3f \n",array_in[i],array_out[i]);
}
```

Kết quả chạy thử 7.1 chỉ ra một thí dụ cho chạy chương trình với 10 giá trị nhập vào.

Chạy thử 7.1

```
Enter number of values to be processed >> 10
Enter value >> 3
Enter value >> -2
Enter value >> 4
Enter value >> 10
Enter value >> 3
Enter value >> 2
Enter value >> 1
Enter value >> 0
Enter value >> 19
```

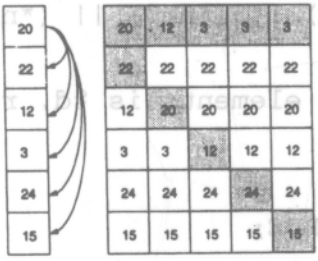
```
Enter value >> 14
 Input      Output
 3.000       3.000
-2.000       1.667
 4.000       4.000
10.000       5.667
 3.000       5.000
 2.000       2.000
 1.000       1.000
 0.000       6.667
19.000      11.000
14.000      14.000
```

Chương trình 7.2 là một thí dụ của chương trình phân loại (sorting) trong đó một mảng được chuyển giao tới hàm `sort()`, hàm này phân bố các giá trị từ nhỏ nhất tới lớn nhất. Thoạt đầu giải thuật kiểm tra giá trị đầu tiên trong mảng với tất cả các giá trị khác. Nếu giá trị trong vị trí đầu tiên lớn hơn giá trị mảng đã lấy mẫu thì hai giá trị được tráo đổi.

Hình 7.5 chỉ ra một thí dụ cho thấy một mảng 6 phần tử có thể được phân loại để xác định giá trị nhỏ nhất như thế nào. Trong lần lặp lại đầu tiên giá trị 20 được so sánh với 22. Bởi vì 20 nhỏ hơn 22 nên các giá trị không được tráo đổi. Tiếp theo, giá trị 20 được so sánh với 12 (phần tử thứ ba), bởi vì giá trị này nhỏ hơn nên các giá trị được tráo đổi. Bây giờ chọn 12 là phần tử đầu tiên. Quá trình này tiếp tục cho đến khi giá trị cuối cùng (15) được kiểm tra. Ở cuối của quá trình lặp giá trị nhỏ nhất (3) sẽ là phần tử đầu tiên trong mảng. Bây giờ, phần tử đầu tiên chứa giá trị nhỏ nhất nên phép toán có thể tiếp tục trên phần tử thứ hai. Phép toán này được kiểm tra dựa vào các phần tử thứ ba, thứ tư, thứ năm và thứ sáu v. v... Số những lần lặp lại cần có để hoàn thành quá trình này do đó sẽ bằng 15 (=5+4+3+2+1).

Chương trình 7.2

```
/* prog7_2.c                                          */
#include <stdio.h>
```



Hình 7.5: **Các phần tử mảng.**

```
#define   MM 150
#define   TRUE 1
#define   FALSE0

void      get_values(int *n,float array[]);
void      print_values(int n,float array_in[]);
void      sort(int n,float input[]);
void      order(float *val1,float *val2);

int       main(void)
{
float     array(max];
int       nvalues;
   get_values(&nvalues,array);
   sort(nvalues,array);
   print_values(nvalues,array);
   return(0);
}
void   get_values(int *n,float array[])
/* *n stores the number of value in the array */
{
int   i,rtn,okay;
   do
   {
```

```
        printf("Enter number of values to be processed >>");
        rtn=scanf("%d",n);
        if ((rtn!=1) || (*n<0) || (*n>MAX))
        {
        printf("Max elements is %d, re-enter\n",MAX);
        okay=FALSE;
        }
        else okay=TRUE;
    } while (!okay);

    for (i=0;i<*n;i++)
    {
        printf("Enter value >>");
        scanf("%f",&array[i]);
    }
}
void   print_values(int n,float array_in[])
{
int    i;

    printf("Ordered values\n");
    for (i=0;i<n;i++)
      printf("%8.3f ",array_in[i]);
}
void   sort(int n,float input[])
/* order array input to give smallest to largest */
{
int   i,j;

    for (i=0;i<n-1;i++)
      for (j=n-1;i<j;j--)
         order(&input [i],&input[j]);
}
void  order(float *val1,float *val2)
/* val1 is the smallest  */
```

```
{
float temp;

    if (*val1 > *val2)
    {
        temp  = *val1;
        *val1 = *val2;
        *val2 = temp;
    }
}
```

Kết quả chạy thử 7.2 giới thiệu một thí dụ khi tiến hành chạy chương trình với 10 giá trị được nhập vào.

Chạy thử 7.2

```
Enter    number of values be entered >> 10
Enter    value >> 3
Enter    value >> -2
Enter    value >> 4
Enter    value >> 10
Enter    value >> 3
Enter    value >> 2
Enter    value >> I
Enter    value >> 0
Enter    value >> 19
Enter    value >> 14
Ordered values
  -2.000 0.000  1.000     2.000  3.000  3.000
  4.000   10.000 14.000 19.000
```

7.3 KHỞI TẠO MẢNG

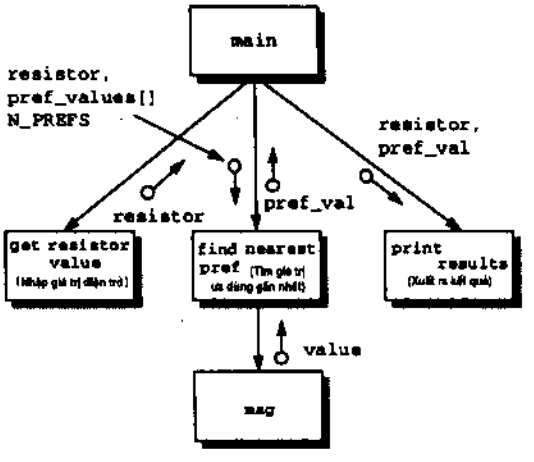
Khởi tạo mảng hay còn gọi là sự thiết lập trạng thái ban đầu của một mảng bằng các giá trị (hoặc các ký tự) được định nghĩa giữa một

cặp dấu ngoặc nhọn ({ }). Dưới đây là khuôn mẫu chuẩn để khởi tạo một mảng:

```
type arrname[nvalues]={val_0,val_1,.... val_n-1};
```

Chương trình 7.3 xác định giá trị điện trở thường được dùng nhất trong phạm vi 10 tới 100 Ω. Mảng đã được khởi tạo `pref_values[]` chứa các giá trị được ưa dùng bằng: 10; 12; 15; 18; 22; 27; 33; 39; 47; 56; 68; 82 và 100 Ω.

Hàm `find_nearest_pref()` xác định giá trị gần nhất được ưa dùng. Các phép toán của hàm sử dụng độ chênh lệch giữa giá trị được nhập vào và một giá trị chỉ số trong mảng các giá trị được ưa dùng. Nếu độ chênh lệch ít hơn độ chênh lệch giữa giá trị gần nhất trước đó và giá trị nhập vào thì giá trị được ưa dùng hiện thời sẽ được chọn trên giá trị mảng đã được chỉ số hoá hiện thời. Hình 7.6 chỉ ra một biểu đồ cấu trúc cơ bản dùng cho chương trình này.



Hình 7.6: **Biểu đồ cấu trúc dùng cho chương trình 7.3.**

Chương trình 7.3

```
/* prog7_3.c                                                    */
#define        TRUE        1
#define        FALSE       0

#include <stdio.h>
```

```
#define  N_PREF 13

void     get_resistor_value(float *r);
void     find_nearest_pref(float r,float pref_arr[],int n_prf,
                    float *p_val);
void   print_results(float r,float pref_r);
float mag (float val);

int     main(void)
{
float
  pref_values [N_PREF]=(10,12,15,18,22,27,33,39,47,56,68,82,100);
float  resistor,pref_val;
    get_resistor_value(&resistor);
    find_nearest_pret(resistor, pref_:values, N_PREF, &pref_val)
    print_results(resistor,pref_val);
    return(0);
}
void   get_resistor_value(float *r)
{
int     rtn,okay;
  /* get a value between 10 and 100 ohms  */
  do
  {
      printf("Enter a resistance (10-100 ohm) >>");
       rtn=scanf("%f",r);
      if ((rtn!=1) || (*r<10) || (*r>100))
      {
           puts("Invalid value, re-enter");
           okay=FALSE;
       }
        else okay=TRUE;
    } while (!okay);
}
```

```
void find_nearest_pref(float r,float pref_arr[],int n_prf,
                                        float *p_val)
{
int    i;
   *p_val=pref_arr[0];
   for (i=1;i<n_prf;i++)
   {
   if (mag(r-pref_arr[i]<mag(*p_val-pref_arr[i]))
        *p_val=pref_arr[i];
   }
}
void   print_results(float r,float pref_r)
{
printf("value entered %8.3f ohm, pref value is %8.3f ohm\n",
               r,pref_r);
}
float  mag(float val)
{ /* Determine the magnitude of val */
    if (val<0.0) return(-val);
    else return(val);
}
```

Kết quả chạy thử 7.3 chỉ ra một thí dụ làm mẫu.

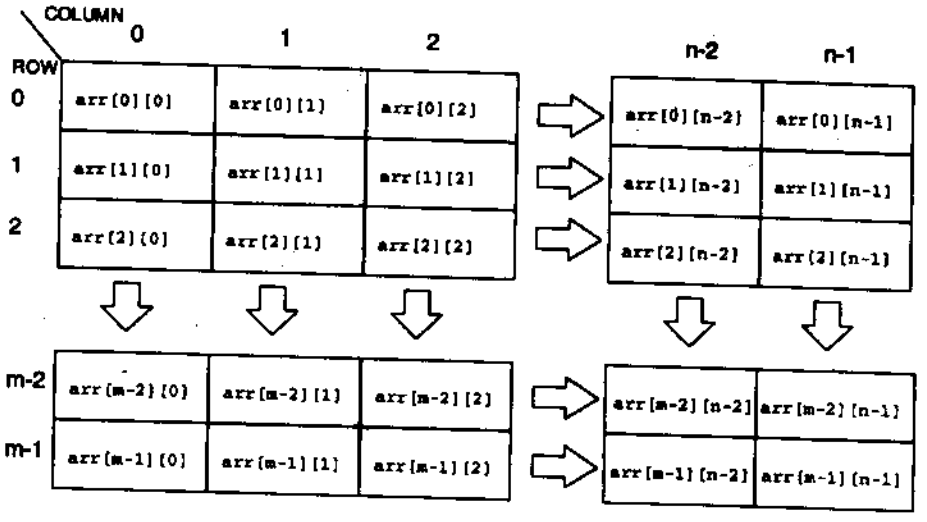
Chạy thử 7.3

```
Enter a resistance (10-100 ohm) >> 3
Invalid value, re-enter
Enter a resistance (10-100 ohm) >> 45
value entered    45.000 ohm, pref value is    47.000 ohm
```

7.4 MẢNG NHIỀU CHIỀU

Cách khai báo mảng đã sử dụng cho tới thời điểm này được áp dụng cho các *mảng tuyến tính* hay còn gọi là *mảng một chiều*. Nhiều ứng dụng đòi hỏi việc chỉ số hóa của dữ liệu dưới dạng *nhiều chiều*. Hình 7.7

cho thấy các hàng và cột tạo ra một mảng hai chiều như thế nào. Chỉ số mảng thứ nhất chỉ rõ số hàng, còn thứ hai là số cột.

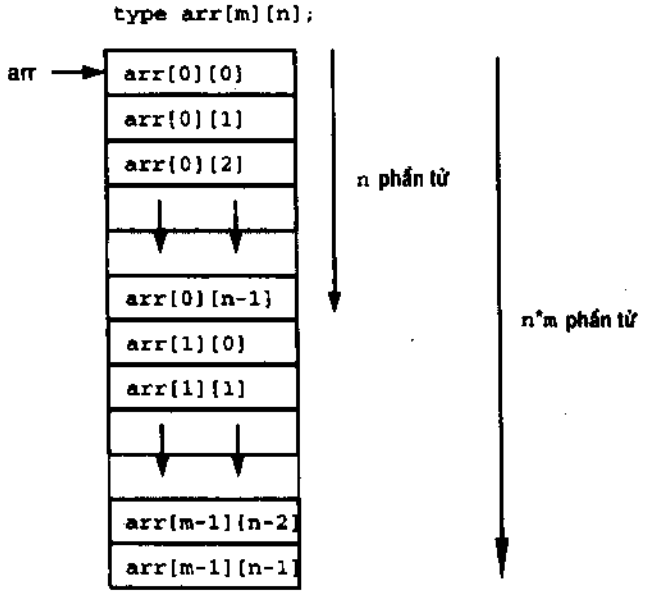



```
type arr[m][n];
```

Hình 7.7: **Các phần tử mảng.**

Hình 7.8 cho thấy một mảng hai chiều được phân bố trong bộ nhớ như thế nào. Trong trường hợp này một mảng `arr` được sắp xếp với `m` cột và `n` hàng. Chương trình dịch phân bố một vùng cho `m*n` giá trị. Một vùng gồm các phần kề sát nhau trong bộ nhớ được phân bố (cấp phát) và con trỏ cơ sở mảng chỉ vào chỗ bắt đầu của vùng, vùng bộ nhớ này được gọi là *đống* (heap).

Địa chỉ cơ sở của mảng sẽ là `&arr [0][0]`; ô nhớ của phần tử thứ hai sẽ là `&arr [0][0]+1`; địa chỉ của phần tử `arr[1][0]` sẽ là `&arr [0][0]+n`; của phần tử `arr[2][0]` sẽ là `&arr[0][0]+2*n+3`. Bảng 7.3 chỉ ra những thí dụ về các phần tử mảng và những địa chỉ của chúng so với địa chỉ cơ sở của mảng.



Hình 7.8: **Phân bố bộ nhớ cho mảng 2 chiều.**

Bảng 7.3: **Những thí dụ về các phần tử mảng và địa chỉ của chúng so với địa chỉ cơ sở.**

Phần tử mảng	Địa chỉ so với địa chỉ cơ bản
`arr[0][0]`	`&arr[0][0]`
`arr[0][1]`	`&arr[0][0]+1`
`arr[0][n-1]`	`&arr[0][0]+(n-1)`
`arr[1][0]`	`&arr[0][0]+n`
`arr[3][5]`	`&arr[0][0]+(3*n)+5`
`arr[m-1][n-1]`	`&arr[0][0]+(m-1)*n+(n-1)`

Chương trình 7.4 hiển thị sự phân bố bộ nhớ của mảng 3 nhân 4. Kết quả chạy thử 7.4 chỉ ra rằng địa chỉ cơ sở của mảng là `FFAEh`. Bởi vì mảng thuộc kiểu `float` nên mỗi phần tử chiếm 4 byte trong bộ nhớ (giá trị này có thể khác nhau trên những hệ thống khác nhau). Địa chỉ offset của phần tử thứ hai `[0][1]` như vậy sẽ là 4 byte kể từ địa chỉ cơ

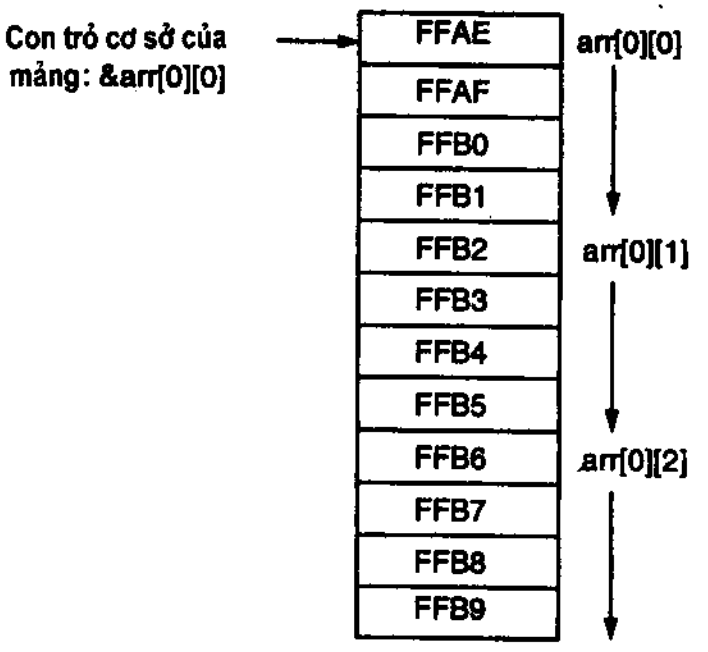
sở. Hình 7.9 phác thảo sự phân bố bộ nhớ dùng cho kết quả chạy thử này.

Chương trình 7.4

```
/* prog7_4.c                                                    */
#include   <stdio.h>
#define  ROWS        3
#define  COLUMNS     4
int   main(void)
{
float arr(ROWS)(COLUMNS);
int   row,col;

    for (row=0;row<ROWS;row++)
          for (col=0;col<COLUMNS;col++)
            printf("Address of element (%d][%d] is %p(hex)\n",
                        row,col,&arr[row][col]);
    return(0);
}
```

Chạy thử 7.4

```
Address  of   element   [0][0]  is   ffae(hex)
Address  of   element   [0][1]  is   ffb2(hex)
Address  of   element   [0][2]  is   ffb6(hex)
Address  of   element   [0][3]  is   ffba(hex)
Address  of   element   [1][0]  is   ffbe(hex)
Address  of   element   [1][1]  is   ffc2(hex)
Address  of   element   [1][2]  is   ffc6(hex)
Address  of   element   [1][3]  is   ffca(hex)
Address  of   element   [2][0]  is   ffce(hex)
Address  of   element   [2][1]  is   ffd2(hex)
Address  of   element   [2][2]  is   ffd6(hex)
Address  of   element   [2][3]  is   ffda(hex)
```



Hình 7.9: **Các phần tử của mảng.**

Khi chuyển giao một mảng hai chiều vào một hàm thì kích thước của các cột cần được chỉ định rõ. Kích thước này sẽ cho những thông tin về địa chỉ offset dùng cho mỗi hàng (xem hình 7.8). Mã mẫu giới thiệu sau đây cho thấy kích thước cột được chỉ định rõ trong dấu ngoặc như thế nào và cũng không cần thiết phải chỉ rõ kích thước hàng.

```
void   pass_arr(float a[][6]);

void   main(void)
{
float  arr[4][6];
    pass_arr(arr);
}

void      pass_arr(float a[][6])
{
    }
```

Những mảng nhiều chiều hơn là không thông dụng nhưng có thể được thiết lập bằng cách sử dụng một phương pháp tương tự với phương pháp đã sử dụng cho mảng hai chiều. Số hạng chỉ số hóa bổ sung được phụ thêm vào mỗi chiều. Những thí-dụ sau đây sử dụng các mảng 3 và 4 chiều.

```
void  pass_arr1(float a[][5][6]);
void  pass_arr2(int a[][5][5][5]);

int   main(void)
{

float arr_3d[4][5][6];
int   temp[5][5][5][5];

    pass-arr1(arr_3d);
    pass-arr2(temp);
    return(0);
}
void     pass_arr1(float a[][5][6])
{
}

void     pass_arr2(float a[][5][5][5])
{
}
```

7.5 PHÂN BỐ ĐỘNG

Khi một biến được khai báo bên trong một chương trình thì chương trình dịch sẽ gán bộ nhớ cho biến đó. Không có gì bảo đảm là bộ nhớ này sẽ thích ứng trong thời gian chạy chương trình. Điều cũng có thể xảy ra là số lượng của bộ nhớ cần có để cất giữ dữ liệu có thể chưa được biết chừng nào chưa cho chương trình chạy. Để vượt qua vấn đề này một kỹ thuật có tên là *gán bộ nhớ trong thời gian chương trình chạy* hay còn gọi là *phân bố bộ nhớ động* đã được sử dụng. Kỹ thuật này kéo theo sự khai

báo con trỏ và sau đó là phân bố bộ nhớ cho con trỏ đó khi chương trình chạy. Thoạt đầu, con trỏ không có bộ nhớ được phân bố và một hàm phân bố bộ nhớ, như `malloc()` đã được sử dụng để phân bố bộ nhớ cần có.

Để khai báo một con trỏ thì kiểu dữ liệu được chỉ rõ và tên con trỏ được đặt trước bằng một dấu sao. Sau đây là khuôn mẫu chung:

```
type_def *ptr_name;
```

Chương trình 7.5 cho thấy một con trỏ được khai báo như thế nào khi sử dụng lệnh `int*b`. Chương trình dịch sẽ không dự trữ không gian cho biến số nguyên. Để gán bộ nhớ ở trạng thái động (dynamically) trong thời gian chạy thì hàm `malloc()` đã được sử dụng; hàm này đã được đặt làm mẫu (prototype) trong tệp *stdlib.h*.

Phép toán gộp lại (casting) kiểu (`int*`) thông báo cho trình dịch biết địa chỉ bắt đầu của sự phân bố bộ nhớ được chỉ vào kiểu `int`. Hàm `Sizeof()` trả lại số byte của kiểu dữ liệu, thí dụ hai hoặc bốn byte cho một số nguyên. Hàm này cho phép có được tính dễ di chuyển (portability) giữa những hệ thống khác nhau, sử dụng số byte khác nhau để cất giữ một kiểu dữ liệu riêng biệt.

Chương trình 7.5

```
/* prog7_5.c                                                    */
*include <stdio.h>
*include <stdlib.h>
int main(void)
{
int a,*b;
   b=(int *) malloc(sizeof(int));
    /*allocate space for integer */
   a=5;
   *b=6;
   printf("Address of variable a is %x, value is %d\n",&a,a);
   printf("Address of pointer b is %x, value pointed to is
                        %d\n",b,"b),
   return(0);
```

```
}
```

Kết quả chạy thử 7.5 chỉ ra một thí dụ làm mẫu. Trong trường hợp này, chương trình dịch đã gán a vào địa chỉ FFDAh (1111 1111 1101 1010b) và số nguyên cất giữ ở ô nhớ đã được chỉ vào bởi địa chỉ bắt đầu của nó là 5. Con trỏ b đã được gán địa chỉ 061Eh bằng hàm malloc() và giá trị số nguyên đã cất giữ ở tại địa chỉ này là 6. Con trỏ b có địa chỉ ban đầu là NULL (địa chỉ không hoặc con trỏ không được phân bổ).

Chạy thử 7.5

```
Address of variable a is ffda, value is 5
Address of pointer b is 61e, value pointed to is 6
```

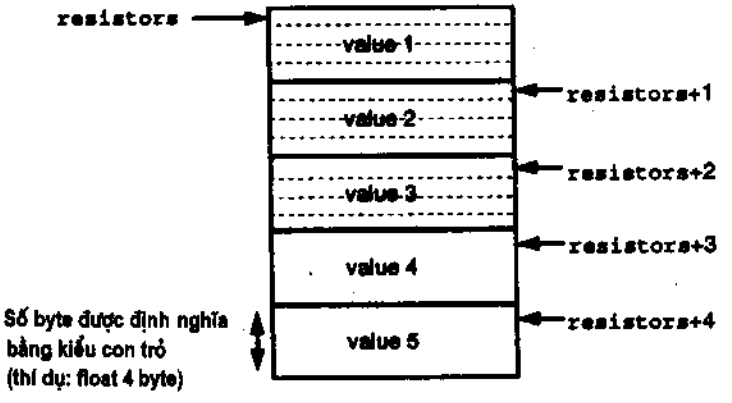
Khi sử dụng các con trỏ cần phải chú ý để có đủ bộ nhớ gán cho chúng để có thể cất giữ đủ tất cả dữ liệu cần có, nói khác đi dữ liệu có thể được viết vào vùng bộ nhớ, chưa được gán cho mục đích khác.

7.6 SỐ HỌC CON TRỎ

Con trỏ có thể được tác động theo kiểu số học. Số của những ô nhớ mà con trỏ được di chuyển tới hoặc so sánh sẽ phụ thuộc vào kiểu dữ liệu của con trỏ. Hình 7.10 chỉ ra một thí dụ về một con trỏ có tên là resistors với năm giá trị được phân bổ. Nếu như con trỏ này được định nghĩa là một con trỏ dấu phẩy động thì mỗi biến sẽ chiếm 4 byte trong bộ nhớ. Các giá trị được cất giữ trong bộ nhớ cũng có thể được so sánh với những con trỏ cơ sở resistors. Nội dung sẽ được so sánh như sau:

Con trỏ so sánh	Giá trị cất giữ
*resistors	value1
*(resistors+1)	value2
*(resistors+2)	value3
*(resistors+3)	value4
*(resistors+4)	value5

Chương trình 7.6 xác định điện trở tương đương của một số điện trở mắc song song và chứa một thí dụ về sự gia tăng giá trị của con trỏ. Chương trình sử dụng sự phân bố bộ nhớ động để gán bộ nhớ cho tất cả các điện trở được nhập vào. Nếu như không có đủ bộ nhớ sẵn sàng để dùng thì hàm `malloc()` sẽ trả lại một NULL (một con trỏ không). Điều này có nghĩa là hàm đã không có khả năng phân bố bộ nhớ cho dữ liệu cần thiết. Nếu như có đủ bộ nhớ thì hàm `get_values()` nhắc người dùng đối với từng điện trở. Con trỏ `resistors` được gia tăng giá trị sau mỗi lần giá trị được nhập vào.



Hình 7.10: **Các phần tử của mảng.**

Chương trình 7.6

```
/* prog7_6.c                                                    */
#include <stdio.h>
#include <stdlib.h>

void  get_resistors(int n,float *res);
void  calc_resistance(int n,float *res,float *r_e);
int             main(void)
{
float  *resistors,R_equ;
int    nres;
  printf("How many resistors in parallel >>");
  scanf("%d",&nres);
```

```
    resistors=(float *) malloc(nres*sizeof(float));
    if (resistors==NULL)
    puts("Cannot allocate enough memory");
    else
    {
    get_resistors(nres,resistors);
    calc_resistance(nres,resistors,&r_equ);
    printf("Equivalent resistance is %8.3f ohm\n", R_equ);
    }
    return(0);
}
void   get_resistors(int n,float *res)
{
int    i;
   for (i=1;i<=n;i++)
   {
      printf("Enter resistor %d >>",i);
      scanf("%f",res);
      res++; /* increment pointer to next value  */
}
}
void   calc_resistance(int n, float *res,floar r_e)
{
float r=0;
int    i;
    for   (i=0;i<n;i++)
    {
    r+=1/(*res);
    res++; /*increment pointer to next value  */
    }
    *r_e=1/r;
}
```

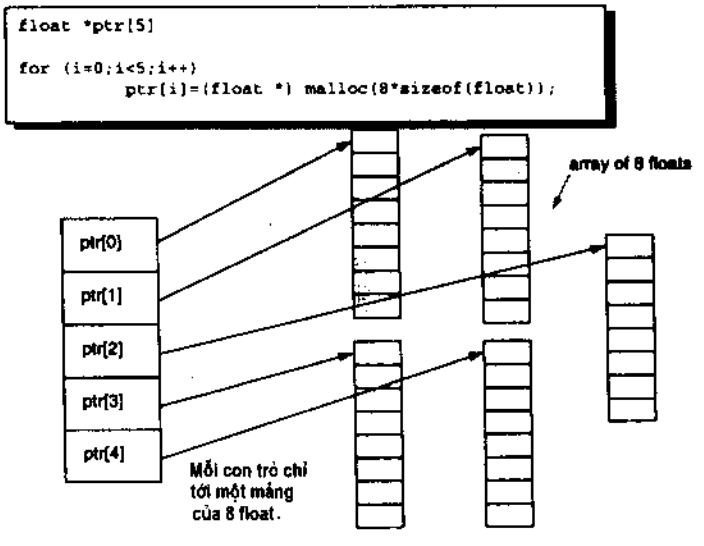
Kết quả chạy thử 7.6 chỉ ra một thí dụ khi chạy chương trình với ba điện trở mắc song song.

Chạy thử 7.6

```
How many resistors in parallel >> 3
Enter resistor 1 >> 100
Enter resistor 2 >>   75
Enter resistor 3 >>   30
Equivalent resistance is  17.647 ohm
```

7.7 MẢNG CON TRỎ

Một mảng của các con trỏ có thể được thiết lập bằng cách sử dụng kỹ thuật *phân bố bộ nhớ động*. Kỹ thuật này tỏ ra có ích trong những ứng dụng mà số lượng của bộ nhớ cần có khi biên dịch không được biết. Mỗi phần tử trong mảng là một con trỏ tới một vùng của bộ nhớ, được phân bố trong thời gian chạy sử dụng một hàm phân bố bộ nhớ, như `malloc()`. Hình 7.11 chỉ ra một mảng của 5 con trỏ; mỗi con trỏ trong số này chỉ vào một mảng của 8 dấu phẩy động.



Hình 7.11: **Mảng của những con trỏ.**

Chương trình 7.7 sử dụng một mảng các con trỏ để cất giữ một số dãy nhị phân. Mỗi dãy có độ dài cố định là 5 digit nhị phân; con số này

được thiết lập bằng hằng số SEQ_LENGTH. Có thể có một số cực đại là 100 dãy được cất giữ, nếu như có đủ bộ nhớ, bởi vì đây là số cực đại của các con trỏ trong mảng bin_seq[]. Giá trị cực đại này được thiết lập bằng hằng số MAXSEQUENCES.

Hàm malloc() phân bố bộ nhớ cho một trong những con trỏ mỗi khi người dùng yêu cầu chương trình cất giữ dãy khác. Hàm get_seq() sử dụng con trỏ này và tác động trên nó nếu như đây là một mảng tuyến tính (một chiều).

Mảng của những con trỏ gửi vào trong show_seq() là *seq[]. Con trỏ *s_ptr được khởi tạo cho chỗ bắt đầu của mỗi dãy và các digit nhị phân được in bằng cách sử dụng biện pháp chỉ số hóa mảng.

Chương trình 7.7

```
/* prog7_7.c                                                    */
#include      <stdio.h>
#include      <stdlib.h>
#define       TRUE      1
#define       FALSE     0
#define       MAXSEQUENCES  100  /* max no of sequences */
#define       SEQ_LENGTH    5  /* sequence length   */

void   get_seq(int seq[]);
void   show_seq(int n_seq,int *seq[]);
int      main(void)
{
int      n_seq=0;
int      *binseq[MAXSEQUENCES];
  do
  {
   printf("Do you wish to add a binary sequence >>");
   fflush(stdin);/* flush keyboard */
   if (getchar()=='n') break;

   bin_seq(n_seq]=(int *) malloc(SEQ_LENGTH*sizeof(int));
```

```
    get_seq(bin_seq[n_seq]);
    n_seq++;

    show_seq(n_seq,bin_seq) ;
    if(n_seq==MAXSEQUENCES)puts("Maximum sequences reached");

  } while (n_seq!=MAXSEQUENCES);
  return(0);
}
void   get_seq(int seq[])
{
int    in,i,rtn,okay;

  for (i=0;i<SEQ_LENGTH;i++)
  {
    do
    {
    print("Enter sequence number %d >>",i);
    rtn=scanf("%d",&in);
    if ((rtn!=1) || ( (in!=0  && (in!=1)))
    {
    puts("INVALID re-enter 1 or 0");
    okay=FALSE;
    }
    else okay=TRUE;

   } while (!okay);
   seq[i]=in;
  }
}
void   show_seq(int n_seq,int *seq[])
{
int    *s_ptr,n,i;

   /* s_ptr points to the start of each of the sequences  */
```

```
    for (n=0;n<n_seq;n++)
    {
      s_ptr=seq[n]; /*initialise array to start of sequence */
      printf("Binary seq %d >>",n);

      for (i=0;i<SEQ_LENGTH;i++)
      {
      printf("%d",s_ptr[i]);
      }
      puts("");
   }
}
```

Kết quả chạy thử 7.7 chỉ ra một thí dụ làm mẫu.

Chạy thử 7.7

```
Do you wish to add a binary sequence >>y
Enter sequencenumber 0 >>1
Enter sequencenumber 1 >>0
Enter sequencenumber 2 >>1
Enter sequencenumber 3 >>0
Enter sequencenumber 4 >>1

Binary seq 0 >>10101
Do you wish to add a binary sequence >>y
Enter number 0 >>1
Enter number 1 >>1
Enter number 2 >>1
Enter number 3 >>1
Enter number 4 >>1
Binary seq 0 >>10101
Binary seq 1 >>11111
Do you wish to add a binary sequence >>n
```

Chú ý là không có công đoạn kiểm tra bên trong chương trình để xác định xem liệu có đủ bộ nhớ cho đến từng phần tử. Đoạn chương trình đã được sửa đổi sau đây xác định kết quả của việc phân bố bộ nhớ và hiển thị thông báo lỗi nếu không thể cấp phát đủ bộ nhớ cần có.

```
int okay_TRUE;
  do
  {
   printf("Do you wish to add a binary sequence >>");
   fflush(stdin);

   if (getchar()=='n') break;

   bin_seq[n_seq]=(int *) malloc(SEQ_LENGTH*sizeof(int));

   if (bin_seq(n_seq]==NULL)
   {
        puts("Cannot allocate memory");
        okay=FALSE;
   }
  } while (okay);
```

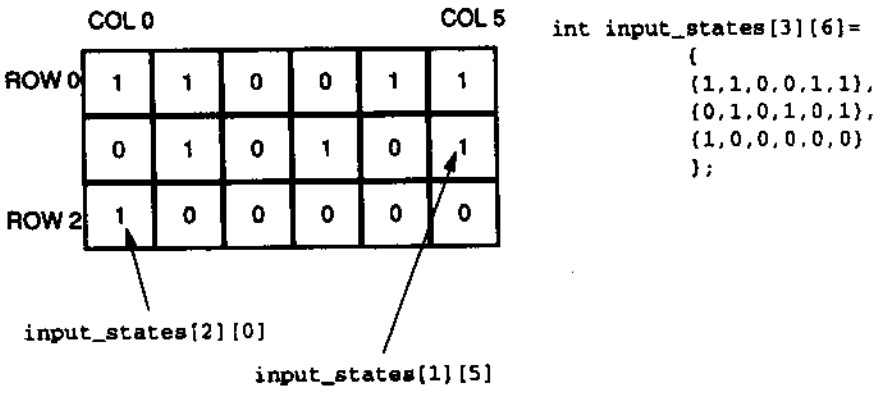
7.8 MỘT SỐ THÍ DỤ

7.8.1 MẠCH ĐIỆN BOOLE

Chương trình 7.8 sử dụng mảng hai chiều để thiết lập một dãy đầu vào 3 bit áp dụng cho một mạch số. Phương trình mạch điện Boole được sử dụng là:

$$Z = (in_1.in_2) + in_3$$

Hình 7.12 chỉ ra mảng `input_states[]` được khởi tạo bằng cách chỉ ra các nhóm của các digit nhị phân được sắp xếp vào các hàng. Các digit này được khởi tạo bằng cách chia nhóm chúng bằng các dấu ngoặc nhọn.



Hình 7.12: **Các phần tử của mảng.**

Chương trình 7.8

```
/* prog7_8.c                                                    */
#include <stdio.h>
#define NO_STATES     6
void   process_states(int nstates,int infl(NO STATES],int out[]);
void   print_results(int nstates,int int[][NO-STATES],int out[]);

int       main(void)
{
int       input-states[3][NO-STATES]={
         {1,1,0,0,1,1},{0,1,0,1,0,1},{1,0,0,0,0,0}};
int       output_state[NO_STATES];

   process_states(NO-STATES,input_states,output_state);
   print_results(NO-STATES,input_states,output-state);
   return(0);
}
void      process_states(int states,int n[][NO_STATESI],int out[])
{
int       i;
   for    (i=0;i<nstates;i++)
      out[i1=(in{0][i] & in[1][i]) | in[2] [i];
}
```

```
void  print_results(int nstates,int in[][NO_STATES],int out[])
{
int       i;
  puts(" A B C Z");
  for (i=0;i<nstates;i++)
     printf("%3d%3d%3d%3d\n",in[0][i], in[1][i],in[2][i], out[i]);
}
```

Kết quả chạy thử 7.8 chỉ ra một thí dụ làm mẫu.

Chạy thử 7.8

```
 A  B  C  Z
 1  0  1  1
 1  1  0  1
 0  0  0  0
 0  1  0  0
 1  0  0  0
 1  1  0  1
```

7.8.2 TRỞ KHÁNG CỦA MẠCH RL

Chương trình 7.9 sử dụng một mảng để xác định trở kháng của mạch RL tương ứng với một khoảng tần số.

Chương trình 7.9

```
/* prog7_9.c                                                    */
#include    <math.h>
#include    <stdio.h>
/* maximum array size */
#define   MAXVALUES 30
#define   TRUE      1
#define   FALSE     0
#define   PI      3.14159

/* define prototypes  */
void   get_param(float *r,float *l,int *fn,float *fl,float *f2);
```

```
void   generate_imp(float r,float 1,int fn,float fl, float f2,
                    float imp[];);
float     calc_zrl(float f.float r,float 1);
void      print_imp(int f_points,float fl.float f2,float imp[]);

int       main (void)
{
float     R, L, start_f, end_f, impedance [MAXVALUES];
int       freq_points;

/* start_f and end_f are the start and end freq points  */
/* freq_points is the number of freq points to sweep */
/* impedance[] is an array containing the impedance values*/

    get_param (&R,&L,&freq_points,&start_f,&end_f);
    generate_imp(R,L,freq_points,start-f,end_f,impedance);
    print_imp(freq_points,start_f,end_f,impedance);
    return(0);
}
void   get_param(float *r,float *1,int *fn,float *fl, float *f2)
{
int     rtn, okay;
    printf("Enter r and 1 >> ");
    scanf("%f %f",r,1);
    do
    {
        printf("Enter number of frequency points>>");
        rtn=scanf("%d",fn);
        if ( (rtn!=1) || (*fn<0)|| (*fn>MAXVALUES) )
        {
            puts("Invalid input");
            okay=FALSE;
        }
        else okay=TRUE;
   } while (!okay);
```

```
  printf("Enter start and end frequencies >>");
  scanf("%f %f",f1,f2);
}
void   generate_imp (float r, float l, int fn, float f1,float f2,
                     float imp[]
{
float fstep,f;
int      i;
     fstep=(f2-f1)/fn;
     f=f1;
     for (i=0;i<fn;f+=fstep;i++)
        imp[i]=calc_Zrl(f,r,l);
}
float calc_Zrl(float f,float r,float l)
{
float X1;
   X1=2*PI*f*l;
   return(sqrt(r*r+X1*X1));
}
void   print_imp(int fn,float f1,float f2,float imp[])
{
float fstep,f;
int      i;

   f=f1;
   fstep=(f2-f1)/fn;

   puts("Frequency      Impedance");

   for (i=0;i<fn;i++;f+=fstep)
        printf("%10.3f %10.3f\n",f,imp[i]);
}
```

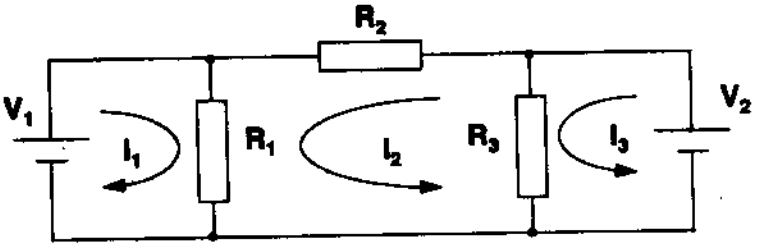
Kết quả chạy thử 7.9 chỉ ra một thí dụ làm mẫu.

Chạy thử 7.9

```
Enter r and l >> 1000 1e-3
Enter number of frequency points >> 20
Enter start and end frequencies >> 1 10e5

Frequency       Impedance
     0.000       1000.000
 50000.000       1048.187
100000.000       1181.010
150000.000       1374.141
200000.000       1605.969
250000.000       1862.096
300000.000       2133.790
350000.000       2415.803
400000.000       2704.912
450000.000       2999.063
500000.000       3296.908
550000.000       3597.530
600000.000       3900.286
650000.000       4204.715
700000.000       4510.479
750000.000       4817.324
800000.000       5125.055
350000.000       5433.522
900000.000       5742.606
950000.000       6052.212
```

7.8.3 PHÂN TÍCH MẠCH DC

Mạch điện một chiều được chỉ ra trên hình 7.13 sẽ được phân tích để xác định các dòng điện I_1, I_2 và I_3. Đây là một mạch đơn giản để phân tích bởi vì các dòng điện có thể được tính một cách đơn giản bằng cách xác định I_1 theo V_1/ R_1, và I_3 theo V_3/ R_3. Các giá trị này sẽ được sử dụng để kiểm tra kỹ thuật ma trận được sử dụng trong chương trình.



Hình 7.13: **Mạch điện một chiều (DC).**

Do đó dẫn đến các phương trình sau:

$$R_1 I_1 + R_1 I_2 + 0 I_3 = 0$$
$$R_1 I_1 + (R_1 + R_2 + R_3) I_2 - R_3 I_3 = 0$$
$$0 I_1 - R_3 I_2 + R_3 I_3 - V_2 = 0$$

Đây là dạng một hệ phương trình 3 biến:

$$a_1 x + b_1 y + c_1 z + d_1 = 0$$
$$a_2 x + b_2 y + c_2 z + d_2 = 0$$
$$a_3 x + b_3 y + c_3 z + d_3 = 0$$

Lời giải có thể tìm được bằng cách sử dụng các định thức. Có thể chỉ ra rằng:

$$x = -\frac{\begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

$$y = -\frac{\begin{vmatrix} b_1 & a_1 & d_1 \\ b_2 & a_2 & d_2 \\ b_3 & a_3 & d_3 \end{vmatrix}}{\begin{vmatrix} c_1 & b_1 & a_1 \\ c_2 & b_2 & a_2 \\ c_3 & b_3 & a_3 \end{vmatrix}}$$

$$z = -\frac{\begin{vmatrix} b_1 & a_1 & d_1 \\ b_2 & a_2 & d_2 \\ b_3 & a_3 & d_3 \end{vmatrix}}{\begin{vmatrix} c_1 & b_1 & a_1 \\ c_2 & b_2 & a_2 \\ c_3 & b_3 & a_3 \end{vmatrix}}$$

Trong trường hợp này:

$$\begin{aligned} a_1 &= R_1 \\ b_1 &= R_1 \\ c_1 &= 0 \\ d_1 &= -V_1 \\ a_2 &= R_1 \\ b_2 &= R_1 + R_2 + R_3 \\ c_2 &= -R_3 \\ d_2 &= 0 \\ a_3 &= 0 \\ b_3 &= -R_3 \\ c_3 &= R_3 \\ d_3 &= -V_2 \end{aligned}$$

Chương trình 7.10 xác định dòng điện trong mạch bằng cách sử dụng phương pháp định thức.

Chương trình 7.10

```
/ *prog7_10.c                                    */
```

```
#include      <stdio.h>
#define  TRUE         1
#define  FALSE        0
void      fill_matrix (float, float, float, float, float, float [] [4]);
void     getvalues (float *, float *, float *, float * , float *)

void    solve_soln(float [][4],float *,float*,float *);
void   get_column(int float [][4],float []);
float  deter(float [],float [],float []);
void   print_values(float i1,float i2,float i3);

      int main (void)
      {
      float   R1,R2,R3,V1,V2,I1,I2,I3;
      float   matrix [3] [4]

    getvalues(&R1,&R2,&R3,&V1,&V2);
    fill_matrix(R1,R2,R3,V1,V2,matrix);
    solve_soln(matrix,&I1,&I2,&I3);
    print-values (I1, I2, I3)
    return(0);

       void    getvalues(float *r1,float *r2,float
     *r3,float *V1,float *V2)

  int        rtn,okay,;
    do
    {
      printf("\nEnter R1, R2 and R3 >> ");
      rtn=scanf("%f %f %f",r1,r2,r3);
      if flrtni=3 || (*r1<0) ||(*r2<0) || (*ı3<0)
      {
       okay=FALSE;
       puts("Invalid values, re-enter");
       }
```

```
      else      okay=TRUE;
    } while (!okay);
    do
    {
      printf ("Enter V1, V2 >>");
      rtn=scanf("%f %f",v1,v2);
      } while (rtn!=2)
}
  void   fill_matrix(float r1,float r2,float3,float v1,
  float v2, float mat [] [4])
{
  /*fills matrix with required values(see notes)*/
  mat[0][0]=r1;
  mat[0][1]=r1;
  mat[0][2]=0;
  mat[0][3]=-v1;

   mat[1][0]=r1;
   mat[1][1]=r1+r2+r3;
   mat[1][2]=-r3;
   mat[1][3]=0;

   mat[2][0]=0;
   mat[2][1]=-r3;
   mat[2][2]=r3;
   mat[2][3]=-v2;

 void  solve_soin(float mat[][4], float *i1,float *i2, float *i3)
 {
 float a_col[3],b_col[3],c_col[3],d_col[3];

   get_column(0,mat,a_col);
        /*scan matrix for column */
   get_column(1,mat,b_col);
   get_column(2,mat,c_col);
```

```
    get_column(3,mat,d_col);

    *i1=-deter(b_col,c_col,d_col)/deter(a_col,b_col,c_col);
    *i2=-deter(a_col,c_col,d_col)/deter(b_col,a_col,c_col);
    *i3=-deter(b_col,a_col,d_col)/deter(c_col,b_col,a_col);

 void    get_column(int col,float mat[][4],float column[])
{
       int     row;

    for (row=0;row<3;row++)
     column[row]=mat[row][col];

 float deter(float col1[],float col2[],float col3[])
{
 float det;
  /* calculate the determinant of a matrix of the form: */
   col1[0] col2[0] col3[0]
   col1[1] col2[1] col3[1]
   col1[2] col2[2] col3[3]
   det=col1[0]*(col2[1]*col3[2]-col2[2]*col3[1])-
   col2[0]*(col1[1]*col3[2]-col1[2]*col3[1])+
   col3[0]*(col1[1]*col2[2]-col1[2]*col2[1]);
   return(det);

 void  print_values(float i1,float i2,float i3)

   printf("i1 = %6.2fA i2 = %6.2fA i3 = %6.2fA\n",i1,i2,i3);
 }
```

Kết quả chạy thử 7.10 chỉ ra một thí dụ làm mẫu.

Chạy thử 7.10

```
Enter R1, R2 and R3 >> 10 5 10
Enter V1, V2 >> 5 10
```

```
I1 =     0.50A I2        0.50A I3        2.50A
```

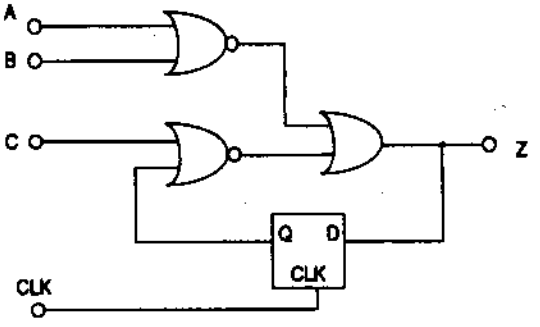
Kết quả này chỉ ra rằng dòng qua điện trở R_1 là 1A ($I_1 + I_2$), qua R_2 là 0.5 A và qua R_3 là 2A ($I_3 - I_2$). Các dòng điện này có thể được ước tính bằng cách sử dụng luật Ôm, và do đó chương trình chuyển qua phép thử đầu tiên.

7.8.4 MÔ PHỎNG LÔGIC

Thí dụ trong đoạn này mô phỏng một mạch lôgic giữ nhịp (đồng hồ) với phản hồi từ đầu ra. Một dãy các bit được thiết lập và đầu ra tổng cộng được xác định. Phương trình đại số Boole cho bởi:

$$Z_n = \overline{A + B} + \overline{(C + Z_{n-1})}$$

ở đây Z_n là trạng thái đầu ra hiện tại và Z_{n-1} là trạng thái đầu ra trước đấy. Hình 7.14 mô tả phương trình bằng một sơ đồ điện.

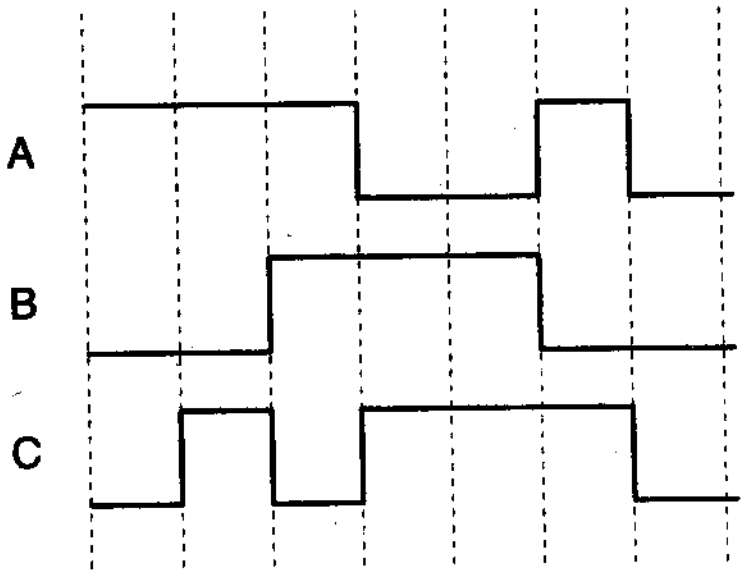


Hình 7.14: **Biểu diễn hàm Boole bằng sơ đồ.**

Hình 7.15 minh họa dãy đầu vào được mô phỏng.

Chương trình 7.11

```
/* prog7_11.c

#include <stdio.h>
#define  NO-STATES 7
```



Hình 7.15: **Giản đồ phân chia thời gian.**

```
void      print_table(int [],int [],int [],int[]);
void      process_states(int [],int [],int [],int[]);
int       main(void)
{
int       A[NO_STATES]=(1,1,1,0,0,1,0);
int       B(NO_STATES]={0,0,1,1,1,0,0};
int       C[NO_STATES]=(0,1,0,1,1,1,0};
int       Q[NO_STATESI;

  process_states(A,B,C,Q);
  print_table(A,B,C,Q);
  return(0);
}

void   process_states(int a[],int b[],int c[],int q[])
{
int i;
```

```
    q[0]=!(a[0] | b[0] | (c[0] & 0);
          /* initial state Zn-1=0*/

    for (i=1;i<NO-STATES;i++)
     q[i]=!(a[i] | b[i] | !(c[i] | q[i-1]);
}
void   print_table(int a[],int b[],int c[],int q[])
{
 int     i;
  printf("\n A B C Q\n");
  for (i=0;i<NO-STATES;i++)
   printf("%3d%3d%3d%3d\n",a[i],b[i],c[i],q[i]);
 }
```

Kết quả chạy thử 7.11 chỉ ra một thí dụ làm mẫu.

Chạy thử 7.11

```
 A B C  Q
 1 0 0  0
 1 0 1  0
 1 1 0  1
 0 1 1  0
 0 1 1  0
 1 0 1  0
 0 0 0  1
```

Chú ý rằng số các trạng thái có thể được chuyển giao như một tham số tới các hàm `process_states()` và `print_table()` như chỉ ra dưới đây.

```
          :     :
    process_states(NO_STATES,A,B,C,Q);
    print_table(NO_STATES,A,B,C,Q);

}
void  process_states(int n,int a[],int b[],
       int c[], int q[])
```

```
{
int i;

        for (i=1;i<n;i++)
           q[i]=!(a[i] | b[i] ) ! (c[i] | q[i-1]
```

7.9 THỰC HÀNH

Q7.1 Hãy viết một hàm trả lại giá trị lớn nhất trong một mảng.

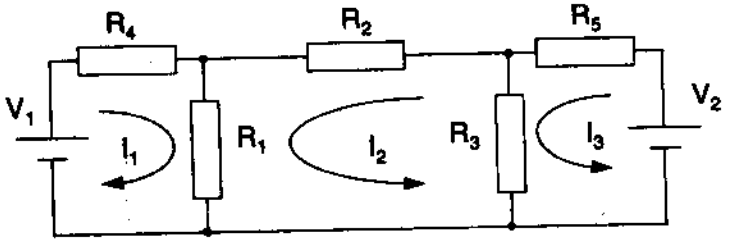
Q7.2 Hãy lặp lại Q7.1 đối với một hàm cực tiểu.

Q7.3 Hãy sửa đổi chương trình 7.9 để xác định góc pha của trở kháng và cất giữ nó trong một mảng.

Q7.4 Hãy sửa đổi chương trình 7.9 để xác định trở kháng của mạch RC nối tiếp.

Q7.5 Hãy viết một hàm sắp xếp một mảng theo những giá trị. Hãy tham khảo thêm chương trình 7.2.

Q7.6 Hãy sửa đổi chương trình 7.10 để xác định các dòng điện của mạch điện được minh hoạ trên hình 7.16.



Hình 7.16: **Mạch điện một chiều.**

Q7.7 Hãy sửa đổi chương trình 7.10 để xác định lời giải cho các hệ phương trình sau, các phương trình này được xác định bởi các định luật Kirchoff.

(a)

$2i_1+i_2+i_3=1,67$

$3i_1+4,5\ i_2-1,5i_3= 0$

$2,25i_1+1,5\ i_2+5,23\ i_3=0$

[*ANS*: $i_1=3,94A, i_2=-3,54A, i_3=2,7$[A]

(b)

$14i_1-5i_2-6i_3=10$

$-5i_1+14i_2-2i_3= 3$

$-6i_1-2i_2+18i_3=5$

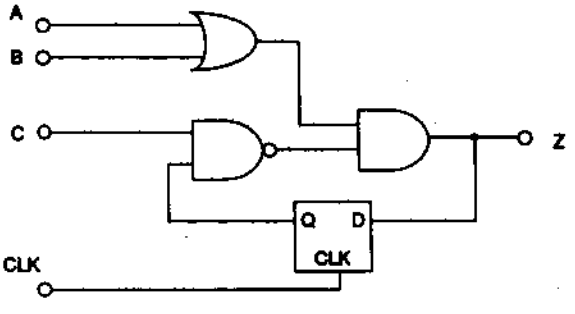
[*ANS*:$i_1=1,358A, i_2=0,816A, i_3 =0,82$[A]

Q7.8 Với mạch số được cho trong hình 7.17 hãy xác định bảng trạng thái Boole đối với dãy đã cho (giả thiết rằng thoạt đầu Z bằng 0). Hãy tham khảo chương trình 7.11.

Dãy:

Chỗ bắt đầu của dãy

```
A     1001110100111010100O
B     0001010110001111O100
C     1110101010101010100l
```



Hình 7.17: **Mạch số.**

Q7.9 Hãy sửa đổi chương trình trong Q7.8 sao cho người dùng có thể nhập vào các dãy A, B và C từ bàn phím. Sử dụng một hàm để cho dãy được nhập vào. Chương trình 7.12 đưa một nét phác thảo của chương trình.

Chương trình 7.12

```
#define    MAX-STATES 100
          :     :
     int main(void)
 {
int A[MAX_STATES],B[MAX_STATES],C[MAX_STATES],
Q[MAX-STATES];

  get_sequence(&n_sequence,A,B,C);
  process_sequence(A,B,C,Q);
  print_sequence(A,B,C,Q);
  return(0);
   }
    void     get_sequence(....)
    {
        get_seq(n,"Enter A sequence",a);
        get_seq(n,"Enter B sequence",b);
        get_seq(n,"Enter C sequence",c);
     }
        :     :     :     :
        :     :     :     :
```

Q7.10 Sửa đổi chương trình 7.3 để xác định giá trị điện trở gần nhất thường được dùng giữa 10 và 100 Ω đối với tập hợp các giá trị thường được dùng như cho trong bảng 7.4.

Table 7.4: **Các giá trị điện trở thường gặp.**

10	16	27	43	68
11	18	30	47	76
12	20	33	51	82
13	22	36	56	91
15	24	39	62	100

Q7.11 Sửa đổi chương trình trong Q7.10 sao cho người dùng có thể nhập vào giá trị bất kỳ của điện trở và chương trình sẽ xác định giá trị điện trở gần nhất thường được dùng. Chạy thử 7.12 đưa một thí

dụ làm mẫu. Gợi ý: viết cho một hàm sắp xếp theo thứ tự (scale) các giá trị nhập vào giữa 10 và 100 Ω, sau đó chuyển giao các giá trị được sắp xếp theo thứ tự tới hàm giá trị thường được dùng.

Chạy thử 7.12

```
Enter resistor value >> 42130
Nearest preferred value is 43000 ohms
```

Q7.12 Hãy viết một chương trình với mảng có kích thước 5 x 3 và xác định các địa chỉ của mỗi phần tử trong mảng. Hãy tham khảo chương trình 7.4. Lặp lại thủ tục đối với mảng 3 x 4 x 2.

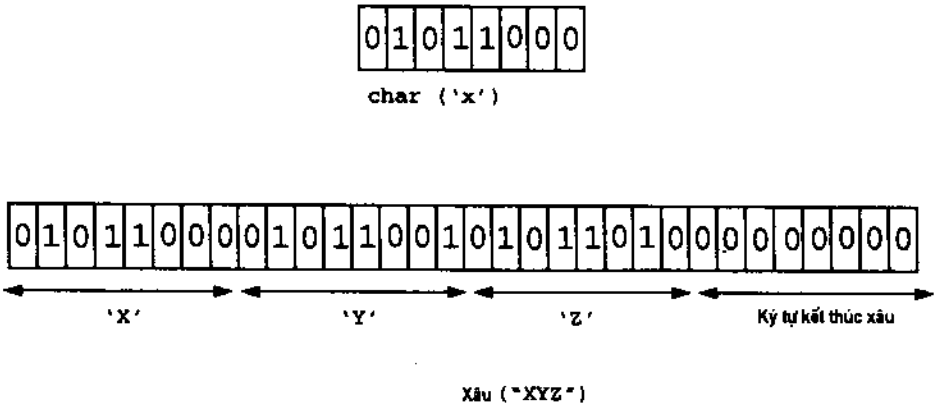
Q7.13 Hãy viết một chương trình xác định trở kháng vào của một số cuộn cảm mắc song song tương ứng với một tần số làm việc. Hãy sử dụng cách phân bố bộ nhớ động. Tham khảo thêm chương trình 7.6.

Q7.14 Lặp lại Q7.13 cho trường hợp các tụ điện mắc song song.

Chương 8
XÂU

Xâu là một mảng một chiều chứa các ký tự. Các xâu có xu hướng thay đổi về độ dài vì thế cần phải khai báo xâu với số cực đại của các ký tự cần có. Một vài thí dụ về các xâu:

- Tên các máy tính, thí dụ "IBM C", "Quả táo Mac", "MicroVAX",
- Tên các bộ vi xử lý, thí dụ "Intel i80386", "Motorola MC6800";
- Tên các phần tử trong mạch điện tử, thí dụ "Điện trở 1", "Vòng khoá pha", "Bộ khuếch đại vi sai".



Hình 8.1: **Thí dụ về bộ nhớ dùng cho các kiểu dữ liệu khác nhau.**

Tất nhiên là số các ký tự được sử dụng để mô tả các tên này có thể thay đổi. Hình 8.1 cho thấy là một ký tự đơn lẻ được lưu trữ bằng cách sử dụng 8 bit. Một xâu có thể thay đổi kích thước của nó và chứa đựng

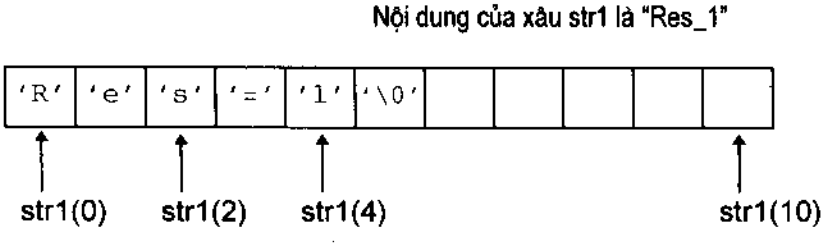
một số các ký tự đơn lẻ. Ký tự mã ASCII NULL hoặc ký tự kết thúc (`'\0'`) định nghĩa chỗ kết thúc một xâu. Trên hình 8.1 xâu "XYZ" được giới hạn ở ký tự NULL (nghĩa là 0000 0000b).

Nếu như dấu ngoặc kép được sử dụng để định nghĩa một xâu, thí dụ. "Cap1", thì một ký tự NULL được tự động nối vào nó. Nếu một xâu được nạp, như một mảng, với những ký tự đơn thì khi đó ký tự NULL cần phải được chèn vào sau ký tự cuối cùng. Chẳng hạn nếu như xâu được nạp là "Cap1" thì các phần tử của mảng sẽ là:`'C','a','p','1','\0'`.

8.1 NHẬP VÀO MỘT XÂU

Xâu là mảng ký tự với kích thước cực đại. Tên xâu là địa chỉ bộ nhớ dùng cho ký tự đầu tiên trong xâu. Đối với một xâu `char name [SIZE]` đã được khai báo thì tên mảng `name` là một địa chỉ cố định ở chỗ bắt đầu của xâu và `SIZE` là số các ký tự được dữ trữ trong bộ nhớ dành cho xâu. Hình 8.2 chỉ ra một thí dụ về sự sắp xếp xâu trong bộ nhớ; xâu đã được khai báo chứa nhiều nhất là 11 ký tự. Số lớn nhất của các ký tự có thể hiển thị được trong xâu được lưu trữ sẽ chỉ bằng 10 bởi vì ký tự NULL được dùng để kết thúc xâu.

```
char str1[11];
```



Hình 8.2: **Thí dụ về sự sắp xếp xâu.**

Ký tự đầu tiên của một xâu được khai báo là `char str1[SIZE]` được đánh số là `str1[0]` và cuối cùng là `str1[SIZE-1]`. Giống với các mảng, có khả năng vượt tràn (overrun) qua chỗ kết thúc của xâu (đặc biệt là khi không có ký tự kết thúc). Điều này có thể làm cho dữ liệu bị đọc ra hoặc viết vào những vùng của bộ nhớ không được gán cho mục

đích này. Xâu đã được định kích thước luôn chứa ít nhất số lượng cực đại của các ký tự +1 được nhập vào. Nếu như một xâu được đọc từ bàn phím thì số lớn nhất của các ký tự có thể được nhập vào bị hạn chế bởi bộ đệm bàn phím. Một macro BUFSIZ, được định nghĩa trong *stdio.h*, có thể được sử dụng để xác định kích thước của nó.

Chương trình 8.1 chỉ ra rằng xâu đã được khai báo instr có thể chứa một số cực đại của các ký tự có thể biển thị BUFSIZ-1 (một ký tự dùng cho NULL). Hàm get() đọc một dòng văn bản cho đến khi phím <RETURN> được nhấn. Hàm này chấp nhận dấu trống giữa các từ, trong khi scanf() giới hạn mỗi xâu bằng các dấu trống (hãy xem chương trình 2.9). Chương trình sẽ tiếp tục nhắc (prompting) xâu văn bản chừng nào mà ký tự đầu tiên của xâu nhập vào là ký tự 'X'.

Chương trình 8.1

```
/* prog8_1.c                                              */
#include <stdio.h>

int   main(void)
{
char instr[BUFSIZ];

    do
    {
      printf("Enter a name (enter 'X' to exit )>>");
      gets(instr);
      printf("Name entered is %s\n",instr);
    } while (instr[0]!='X');
    return(0);
}
```

Kết quả chạy thử 8.1 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.1

```
Enter a name (enter X to exit )>> resistor_1
Name entered  is resistor_1
Enter a name (enter X to exit )>> inverting amplifier
```

```
Name entered  is inverting amplifier
Enter a name (enter X to exit )>> X
Name entered  is X
```

8.2 GÁN XÂU

Giống như với các mảng, xâu có thể được nạp bằng các ký tự khi sử dụng cách đánh số mảng như sau:

```
name [0] = 'R';
name [1] = 'e';
name [2] = 's';
name [3] = '_';
name [4] = '1';
name [5] = '\0' /* Kết thúc xâu bằng một Null */
```

Cách đánh số này sẽ nạp ký tự `'R'` vào vị trí đầu tiên, `'e'` vào vị trí thứ hai, vân vân. Chương trình 8.2 chỉ ra một thí dụ cho thấy các ký tự riêng lẻ được nạp như thế nào vào trong một mảng ký tự. Địa chỉ được chuyển giao tới `printf()` đầu tiên chỉ tới chỗ bắt đầu của xâu này. Có thể in một phần của xâu bằng cách chuyển giao một địa chỉ cơ sở khác. Chẳng hạn, để in "s_1" thì địa chỉ được chuyển giao tới `printf()` là `&instr[2]`.

Hàm `strcpy()` được chứa trong thư viện chuẩn. Nó sao chép xâu đối số thứ hai vào trong xâu đối số đầu tiên.

Chương trình 8.2

```
/* prog8_2.c                                              */
#include <stdio.h>
#include <string.h>

int     main(void)
{
char    instr[BUFSIZ];

  instr[0]='R';
  instr[1]='e';
```

```
    instr[2]='s';
    instr[3]='_';
    instr[4]='1';
    instr[5]='\0';  /* kết thúc xâu bằng một null */
    printf("String is %s\n",instr);
    printf("Part of string is %s\n",&instr[2]);
    strcpy(instr,"Res_2");
    printf("String is %s\n",instr);
    return (0) ;
}
```

Kết quả chạy thử 8.2 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.2

```
String is Res_1
Part of string is s_1
String is Res_2
```

Chương trình 8.3 chỉ ra một thí dụ cho thấy `malloc()` phân bố bộ nhớ dành cho một xâu như thế nào. Nó gán cho một số byte của không gian trống và trả lại một con trỏ vào chỗ bắt đầu của bộ nhớ được phân bố. Hàm `strcat()` đính xâu đối số thứ hai vào với xâu đối số thứ nhất và đặt kết quả vào đối số thứ nhất.

Chương trình 8.3

```
/* prog8_3.c                                                    */
*include <stdio.h>
*include <stdlib.h>
*include <string.h>
int   main(void)
{
char  *str1,*str2;
         /* dynamic string allocation */
  str1=(char *)malloc(BUFSIZ*sizeof(char));
```

```
    str2=(char *)malloc(BUFSIZ*sizeof(char));

    printf("Enter a string of text >>")
    gets(str1);
    printf("Enter a second string of text >>"),
    gets(str2);
    printf("STR1:     %s\n",str1);
    printf("STR2:     %s\n",str2);
    strcat(str1,str2);

    printf("STR1+STR2: %s",str1);
    return(0);
}
```

Kết quả chạy thử 8.2 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.3

```
Enter a string of text >> resistor
Enter a string of text >> resistor 1 is 43
Enter a second string of text >> resistor 2
   str1:  resistor 1 is 43 ohms
   str2:  resistor 2 is 56 ohms
str1,str2: resistor 1 is 43 ohms resistor 2
```

Chương trình 8.4 chứa một hàm `nochars()`, hàm này quét một xâu và xác định số lần xuất hiện của một ký tự cho trước. Nó sử dụng `get()` để đọc xâu khi chấp nhận các dấu trống giữa các từ.

Hàm `nochars()` sử dụng số học con trỏ để đọc từng ký tự trong xâu đã được chuyển giao cho đến khi gặp một ký tự NULL. Hàm `getchar()` được sử dụng để tìm kiếm ký tự.

Chương trình 8.4

```
/* prog8_4.c                                          */
#include <stdio.h>

   /* Find the number of occurrences in a tring */
```

```
int nochars (char *str, char ch) ;

int main(void)
{
 charstr1[BUFSIZ],ch;
  printf("Enter a string >>");
  gets(str1);
  printf("Enter character to find >>");
  ch=gecchar();
  printf(*Number of occurrences is %d\n",nochars(str1,ch);
  return(0);
}

int nochars(char *string,char c)
{
int no_occ=0;
/* repeat until end of string (Null) */
while (*string!='\0')
{
    if (c== *string) no_occ++;
    string++; /* increment pointer to the next character */
  }
  return(no_occ); /*no. of characters found */
}
```

Kết quả chạy thử 8.4 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.4

```
Enter a string>> resistor 1 is 100 omhs
Enter a letter to find >> s
Number of occurrences = 4
```

Hàm sau đây thực hiện `nochars()` bằng cách sử dụng ký hiệu đánh số (chỉ số hóa) mảng. Số các ký tự trong xâu được xác định bằng cách sử dụng `strlen()`. Sau đó vòng lặp `for()` được sử dụng để đánh số cho từng ký tự trong mảng và chúng được kiểm tra theo các ký tự tìm kiếm.

```
int nochars(char str[],char c)
{
int i=0,no_occ0,size;
    size=strlen(str);
    for (i=0;i<size;i++)
    if (str[i]==c) no_occ++;
     return(no_occ);
}
```

8.3 HÀM XÂU CHUẨN

Có vài hàm quản lý (handling) xâu trong thư viện chuẩn; hầu hết chúng được đặt làm mẫu (prototype) trong tệp *string.h*. Bảng 8.1 liệt kê các hàm này. Tất cả các hàm xâu đều trả lại một giá trị; chẳng hạn, `strlen()` trả lại cho một giá trị nguyên liên quan đến chiều dài của một xâu và các hàm `strcat()`, `strupr()`, `strlwr()` và `strcpy()` đều trả lại những con trỏ cho xâu kết quả. Con trỏ này có thể được sử dụng, nếu cần, nhưng xâu kết quả cũng được chuyển giao ngược trở lại như là đối số đầu tiên của các hàm này. Hàm `strcmp()` trả lại chỉ một số 0 nếu như cả hai xâu là đồng nhất.

Bảng 8.1: **Các hàm điều khiển xâu quan trọng.**

Những hàm biến đổi	Đầu mục	Mô tả
`int strcmp(char *str1,char *str2);`	*string.h*	Chức năng: So sánh hai mảng `str2` và `str2` Trả lại: Một số 0 được đưa trở lại nếu hai xâu đồng nhất, một giá trị âm nếu `str1` nhỏ hơn `str2`, hoặc một số dương nếu `str1` lớn hơn `str2`.
`int strlen(char *str);`	*string.h*	Chức năng: Xác định số các ký tự có trong xâu. Trả lại: Số các ký tự có trong xâu.

`char*strcat(char*str1, char*str2);`	*string.h*	Chức năng: Nối str2 vào str1. Xâu kết quả `str1` sẽ chứa `str1` và `str2`. Trả lại: Một con trỏ vào xâu kết quả
`char*strlwr(char*str1);`	*string.h*	Chức năng: Biến đổi các chữ hoa thành một xâu các chữ thường. Trả lại: Một con trỏ vào xâu kết quả
`char*strupr(char*str1);`	*string.h*	Chức năng: Biến đổi các chữ thường thành một xâu các chữ hoa. Trả lại: Một con trỏ vào xâu kết quả
`char*strcpy(char*str, char*str2);`	*string.h*	Chức năng: Sao chép `str2` vào `str1`. Trả lại: Một con trỏ vào xâu kết quả
`int sprintf(char *str, char *format_str, arg1,...);`	*stdio.h*	Chức năng: Tương tự với `prinrf()` nhưng lối ra dẫn tới xâu `str`. Trả lại: Số các ký tự lối ra.
`int scanf(char *str, char *format_str, arg1,...);`	*stdio.h*	Chức năng: Tương tự với `scanf()` nhưng lối vào dẫn từ xâu `str`.. Trả lại: Số các trường được quét xong.

Chương trình 8.5 chỉ ra `sscanf()` quét một xâu với những kiểu dữ liệu khác nhau như thế nào. Trong trường hợp này, người dùng nhập vào một xâu text (văn bản). Từ đầu tiên của xâu được đọc như là một xâu `(res_name)` và từ thứ hai như một dấu phẩy động `(res_values)`. Hàm `sscanf()` trả lại số các trường đã được quét xong; nếu như số các trường đã được quét bằng 2 thì biến `okay` được đặt là `TRUE` (*đúng*) và

như vậy vòng lặp do{}while()sẽ chấm dứt. Nếu như số này không bằng tới 2 thì biến okay được đặt là FALSE và một thông báo lỗi được hiển thị, lúc đó người dùng sẽ được nhắc để nhập lại giá trị.

Chương trình 8.5

```
/* prog8_5.c                                                    */
#define       TRUE   1
#define       FALSE  0
#include      <stdio.h>
#include      <string.h>

int   main(void)
{
char      res_name [BUFSIZ], instr[BUFSIZ], outstr[BUFSIZ];
float     res_value;
int       rtn,okay;
  do
  {
     printf("Enter resistor identifier and the value >>");
     gets(instr);
     rtn=sscanf(instr,"%s %f",res_name,&res value);
     if (rtn!=2)
     {
           okay=FALSE;
           prinrf("Invalid input <%s>\n",instr);
     }
     else okay=TRUE;
  } while (!okay);
  printf("Resistor name is %s, value is %8.3f ohm\n",
        res_narne,res_value);
  sprintf(outstr,"Resistor name is %s, value is %8.3f ohm\n"
        res_name,res_value);
  puts(outstr);
  return(0);
}
```

Kết quả chạy thử 8.5 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.5

```
Enter resistor identifier and the value >> resistor_1 1430
Resistor name is resistor_1, value is 1430.000 ohm
Resistor name is resistor_1, value is 1430.000 ohm
```

Chương trình 8.6 sử dụng hàm xâu `strcinpo()`. Vòng lặp `do{} while()` tiếp tục chạy chừng nào người dùng nhập vào từ `"exit"`.

Chương trình 8.6

```
/* prog8_6.c                                                    */
#include  <stdio,h>
#include <string.h>

int     main(void)
{
char    str1[BUFSIZ],str2[BUFSIZ],instr[BUFSIZ];

  do
  {
    printf("Enter a string >> ");
    gets(str1);
    printf("Enter a string >>");
    gets(strr2);
    if(!strcmp(str1,str2))Puts(">>> Strings  are identical<<<");
    else puts(">>>Strings differ<<<");

    printf("Do you wish to continue (type 'exit' to quit)>>");
    gets(instr);
  } while  strcmp(instr,"exit"));
  return(0);
```

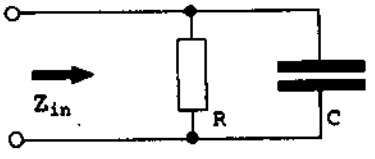
Kết quả chạy thử 8.6 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.6

```
Enter a string >> this is a string
Enter a string >> this is another
>>> Strings differs <<
DO you wish to continue (type 'exit' to quit)>>
Enter a string >> input is zero
Enter a string >> input is zero
>>> Strings are identical <<<
DO you wish to continue (type 'exit' to quit)>>exit
```

8.4 TRỞ KHÁNG CỦA MẠCH RC SONG SONG

Chương trình trong đoạn này xác định trở kháng của một mạch RC mắc song song. Hình 8.3 giới thiệu sơ đồ của mạch này.



Hình 8.3: **Mạch RC song song.**

Trở kháng của mạch này có thể xác định bằng cách sử dụng tích số của trở kháng chia cho tổng số của trở kháng. Như vậy:

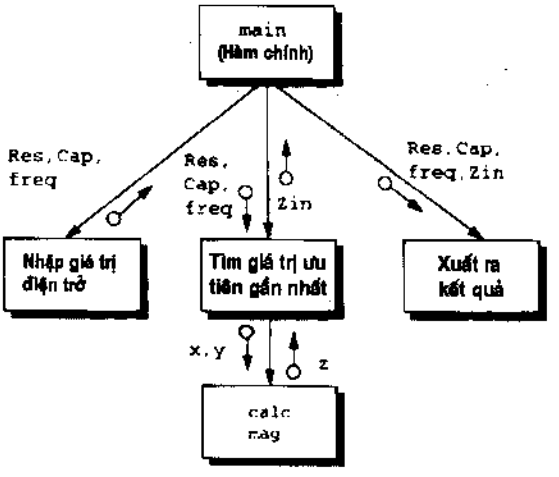
$$Z = \frac{R.\frac{1}{j\varpi.C}}{R+\frac{1}{j\varpi.C}} = \frac{R}{j\varpi.C(R+\frac{1}{j\varpi.C})} = \frac{R}{j\varpi.CR+1}$$

Như vậy độ lớn của trở kháng bằng:

$$|Z| = \frac{R}{\sqrt{1+(\varpi.RC)^2}}$$

Biểu đồ cấu trúc của chương trình xác định đại lượng này được mô tả trên hình 8.4. Hàm get_parameters() có ba biến (Res,Cap và

freq); hàm `parallel_impedance()` xác định trở kháng vào và trả lại cho nó biến `Zin`. Cuối cùng, hàm `print_impedance()` hiển thị các tham số đầu vào và trở kháng được tính.



Hình 8.4: **Sơ đồ cấu trúc của chương trình 8.7.**

Trong những chương trình trước các giá trị đã được nhập vào trong chương trình sử dụng hàm `scanf()`. Hàm này có thể không đáng tin cậy khi nào khuôn mẫu của dữ liệu nhập vào khác với khuôn dạng cần có. Chẳng hạn, người dùng có thể đã nhập vào một xâu ký tự hoặc một giá trị thực trong khi lại cần có một giá trị nguyên. Một phương pháp cải tiến cách nhập giá trị vào bằng cách đưa thêm công đoạn đọc đầu vào như một xâu (bằng `get()`) và quét các giá trị này theo những tham số cần có (bằng `sscanf()`). Một đoạn mã thực hiện được yêu cầu này được dẫn ra sau đây.

Trong trường hợp này, người dùng được nhắc nhập vào một giá trị điện trở và xâu vào từ bàn phím được nạp vào trong `inline`. Sau đó lại được quét để tìm một giá trị dấu phẩy động và đặt vào trong con trỏ `r`. Nếu một giá trị dấu phẩy động đơn được quét, thì `rtn` sẽ bằng 1 và giá trị nhập vào là không âm, và như vậy vòng lặp sẽ kết thúc, nếu không người dùng sẽ lại được nhắc nhập vào một giá trị.

```
void   get_parameters(float *r,float *c,float *f)
{
```

```
char    inline[BUFSIZ];
int     rtn,okay;

    do
    {
        printf("Enter resistance >>")
        gets(inline);
        rtn=sscanf(inline,"%f",r);
        if ((rtn!=1) || (*r<0))
        {
          printf("Invalid input <%s>\n",inline);
          okay=FALSE;
        }
        else okay=TRUE;
} while (!okay);
```

Chương trình hoàn thiện được thể hiện trong chương trình 8.7.

Chương trình 8.7

```
/* prog8_7.c                                          */
#include    <stdio.h>
#define     TRUE    1
#define     FALSE0
#define     PI      3.14159

 void    get_parameters(float *r,float *c,float *f);
 void    parallel_impedance(float r,float c,float f,float Z);
 void    print-impedance(float r,float c,float f,float Z);
 float   calq_mag(float x,float y);

int      main(void)
{
float    Res,Cap,freq,Zin;
 get_parameters(&Res,&Cap,&freq);
 parallel_impedance(Res,Cap,freq,&Zin);
 print_impedance(Res,Cap,freq,Zin);
```

```
  return(0);
}
void   get_parameters(float *r,float *c,float *f)
{
char       inline[BUFSIZ];
int        rtn,okay;
 do
 {
    printf("Enter resistance >>");
    gets(inline);
    rtn=sscanf(inline,"%f",r);
    if ((rtn!=1) || (*r<0))
    {
       printf("Invalid input <%s>\n",inline);
       okay=FALSE;
     }
     else okay=TRUE;
   } while (!okay);
   do
   {
      printf("Enter capacitance >>");
      gets(inline);
      rtn=sscanf(inline,"f",c);
      if ((rtn!=1) || (*c<0))
      {
         printf("Invalid input <%s>\n",inline);
         okay=FALSE;
      }
      else okay=TRUE;
    } while (!okay);
    do
    {
       printf("Enter frequency  >>");
       gets(inline);
       rtn=sscanf(inline,"%f",f);
```

```
        if ((rtn!=1) || (*f<0);
        {
          printf("Invalid input <%s>\n",inline);
          okay=FALSE;
        }
        else okay=TRUE;
    } while (!okay);
}
void   print_impedance(float r,float c,float f,float Z)
{
    printt("R=%f ohms C=%f uF f=%f Hz\n",r,c,f);
    printf("Zin = %f ohms\n",Z);
}
void  parallel-impedance(float r,float c,float f,float *Z)
{
    *z=r/(calc_mag(1,2*PI*f*r*c));
}
float calc_mag (float x, float y)
{
  return(x*x+y*y);
}
```

Chạy thử 8.7 chỉ ra rằng người dùng có thể nhập vào một giá trị trong khuôn mẫu không đúng và chương trình sẽ nhắc lại để nhập vào giá trị khác. Chú ý là người dùng đã nhập vào các xâu `"none"` và `"fred"`; chương trình sao chép bằng các xâu đó và nhắc lại việc nhập vào. Nếu như `scanf()` đã được sử dụng thay cho `gets()` và `sscanf()` thì khó mà đoán trước chương trình sẽ diễn biến như thế nào. Chẳng hạn, khi chạy chương trình trên một máy tính cá nhân PC mà ta sử dụng `scanf()` sẽ dẫn đến hiện tượng nhắc nhở luân hồi thông báo `Enter resistance >>` (*nhập vào giá trị điện trở*) khi ta vô tình nhập vào một xâu ký tự, trong khi lẽ ra phải là một giá trị thực.

Chạy thử 8.7

```
Enter resistance      >>non.
Invalid input <none>
```

```
Enter resistance      >>fred
Invalid input       <fred>
Enter resistance      >>-100
Invalid input <-100>
Enter resistance >>1000
Enter capacitance >>1e-6
Enter frequency >>1e3
R=1000.000 ohms C= 1.00 uF  f=1000.000 Hz
Zin = 24.704565 ohs
```

Hàm `gets_parameters()` chứa mã, và mã này đã được lặp lại ba lần. Một chương trình được hoàn thiện thay thế mã lặp lại bằng những lời gọi hàm đơn. Một hàm có tên `gets_float` có một giá trị dấu phẩy động trong phạm vi được chỉ rõ. Một mẫu của hàm này được chỉ ra dưới đây. Đối số đầu tiên là thông báo nhắc nhở, hai đối số tiếp theo là các giá trị cực tiểu và cực đại, và đối số cuối cùng là một con trỏ vào giá trị.

```
void    get_float(char msg[],float min,float max,float *val);
```

Chương trình 8.8 chứa hàm này. Hàm `get_parameters()` bây giờ chứa ba lời gọi tới `gets_float()`.

Chương trình 8.8

```
/* prog8_8.c                                                    */
#include <stdio.h>
#define   TRUE1
#define   FALSE   0
#define   PI      3.14159
#define   MICRO   1e-6
void   get_parameters(float r,float lc,float f);
void   parallel_impedance(float r,float c,float E,float *Z);
void   print_impedance(float r,float c,float f,float Z);
float  calc_mag (float X,float y);
void   get_float(char msg[],float min,float max,float *val);

int    main(void)
{
```

```
float  Res,Cap,freq,Zin;
   get_parmeters(&Res,&Cap,&freq);
   parallel_impedance(Res,Cap,freq,&Zin);
   printf_impedance(Res,Cap,freq,Zin);
   return(0);
}
void  get_parameters(float *r,float *c,float *f)
{
   get_float("Enter resistance >> ",0,10e6,r);
   get_float("Enter capacitance >> ",0,1,c);
   get_float("Enter frequency >> ",0,10e7,f);
}
void   get-float(char mag[],float min,float max,float *val)
{
char   inline(BUFSIZ);
int    rtn,okay;

  do
  {
       printf("%s",msg);
       gets(inline);
       rtn=sscanf(inline,"f",val);
       if ((rtn!=1) || (*val>min) || (*val>max))
       {
         okay=FALSE;
         printf("Invalid input <%s>\n",inline);
       }
       else okay=TRUE;
    } while (!okay);
}

void   print_impedance (float r, float c, float f, float Z)
{
    printf ("R=%f ohms C=%f uF f=%f HZ\n", r, c/MICRO,f ) ;
    printf("Zin = %f ohms\n",Z);
```

```
}
void   parallel_impedance(float r,float c,float f,float *Z)
{
      *Z=r/(calc_mag(1,2*PI*f*r*c));
}
float calc_mag(float x, float y)
{
   return(x*x+Y*Y);
}
```

8.5 LỰA CHỌN MẠCH ĐIỆN

Chương trình 8.9 giới thiệu một hệ thống thực đơn cơ bản, có thể được sử dụng trong một chương trình để hiển thị một bảng chân lý Boole. Một mảng của các xâu được thiết lập để cất giữ các thực đơn tùy chọn. Khai báo `char *menu_options[5]` khai báo một mảng của 5 con trỏ; mỗi một con trỏ chỉ tới một trong các thực đơn tùy chọn. Con trỏ đầu tiên `*menu_options[0]` trỏ vào xâu `"1-AND function"`, con trỏ thứ hai vào `menu_options[1]` tới `"2-OR function"`, v. v... Sau đây xin dẫn ra một danh sách các xâu trong mảng con trỏ.

```
menu_options[0] "1-AND function"
menu_options[1] "2-OR function"
menu_options[2] "3-NOR fmction"
menu_options[3] "4-NAND function"
menu_options[4] "5-EXIT"
```

Một mảng con trỏ được thiết lập và được khởi tạo cho thực đơn tuỳ chọn.

Chương trình 8.9

```
/* prog8_9.c                                                    */
#define LOGIC_FUNCTIONS 5
#define TRUE                1
#define FALSE               0

#include <stdio.h>
```

```
int     display_menu(char *menu(LOGIC_FUNCTIONS]);
void    get_int(int *opt,int min,int max, char msg[]);
enum    logic (AND=1,OR,NOR,NAND,EXIT);

int      main(void)
{
char   *memu_options(LOGIC_FUNCTIONS)={
       "1-AND function","2-OR function",
       "3-NOR function","4-NAND fmction","5-EXIT"};
enum   logic option;
   do
   {
      option=display_menu(menu_options);
      printf("Option selected is %d (%s)\n",
                    option,menu_options[option-1]);
   } while (option!=EXIT);
   return(0);
 }
int display_menu(char *menu[LOGIC_FUNCTIONS])
{
int i,opt;
  /* display menu and get circuit option */
  for (i=0;i<LOGIC_FUNCTIONS;i++)
     puts(menu[i]);
   get_int(&opt,AND,EXIT,"Enter logic device >>");
   return(opt);
}
void   get_int(int *opt,int min,int max,char msg[])
{
char   inline[BUFSIZ];
int     rtn,okay;
  /*get an integer value between min and max */
  do
  {
    printf ("%s", msg) ;
```

```
    gets(inline);
    rtn=sscanf(inline,"%d",opt);
    if ((rtn!=1) || (*opt<min) || (*opt>max))
    {
      okay=FALSE;
      printf("Invalid input <%s>\n",inline);
    }
    else okay=TRUE;
  } while (!okay);
}
```

Kết quả chạy thử 8.8 chỉ ra một thí dụ chạy mẫu chương trình.

Chạy thử 8.8

```
1-AND function
2-OR function
3-NOR function
4-NAND function
5-EXIT
Enter logic device >>1
option selected is 1 (I-AND function)
1-AND function
2-OR function
3-NOR function
4-NAND function
5-EXIT
Enter logic device >>3
Option selected is 3 (3-NOR function)
1-AND function
2-OR function
3-NOR function
4-NAND function
Enter logic device >>5
Option selected is 5 (5-EXIT)
```

8.6 THIẾT LẬP MỘT MẢNG CỦA CÁC XÂU

Cách đơn giản nhất để thiết lập một mảng của các xâu là định nghĩa một kiểu dữ liệu xâu mới. Trong chương trình 8.10 một kiểu dữ liệu mới có tên là `string` được định nghĩa bằng cách sử dụng lệnh `typedef char string [BUFSIZ]`. Khi đó, một mảng của các xâu lúc đó được thiết lập bằng cách sử dụng khai báo `string database [MAX_COMP_NAMES]`. Mỗi xâu trong mảng có thể chứa nhiều nhất là `BUFSIZ` ký tự (chú ý rằng con số này có thể được thay đổi tới kích thước bất kỳ nếu như có yêu cầu) và có thể được truy nhập bằng cách sử dụng cách chỉ số hóa mảng như thông thường.

Chương trình 8.10

```
/* prog8_10.c                                    */
#include <stdio.h>
#include <string.h>

#define MAX_COMP_NAMES 100

typedef char string[BUFSIZ];
void get_component_name(int *comp, string data[]);
void      print_component_names(int comp, string data[]);
int      main(void)
{
int components=0;
string  database(MAX_COMP_NAMES);
   do
   {
       get_component_name(&components,database);

       if (scrcmp(database(components-1],"exit")) break;
       print_component_names(components,database);
   } while (components<MAX_COMP_NAMES);
   return(0);
}
void    get_component_name(int *comp, string data())
{
```

```
int     i;
  .printf("Enter component name >>");
   gets(data[*comp]);
   (*comp)++;
}
void   printf_component_names(int comp, string data{]
{
int       i;
   puts("Component Name ");
   for (i=0;i<comp;i++)
      printf("%d %s\n",i,data[i]);
}
```

Một kết quả chạy làm mẫu được giới thiệu trong chạy thử 8.9. Chú ý rằng xâu `"exit"` sẽ kết thúc chương trình.

Chạy thử 8.9

```
Enter component name >>resistor 1
0         resistor 1
Enter component name >>resistor 2
Component Names
   0      resistor 1
   1      resistor 2
Enter component name >>capacitor 1
Component Names
   0      resistor 1
   1      resistor 2
   2      capacitor 1
Enter component name >>capacitor 4
Component Names
   0      resistor 1
   1      resistor 2
   2      capacitor 1
```

```
   3       capacitor 4
Enter component name >>inductor
component Names
   0       resistor 1
   1       resistor 2
   2       capacitor 1
   3       capacitor 4
   4       inductor 5
Enter component name >> exit
```

Một mảng của các xâu cũng có thể được khởi tạo bằng cách sử dụng dấu ngoặc nhọn. Chương trình 8.11 và kết quả chạy thử 8.10 cho ta thấy mảng này được thiết lập như thế nào.

Chương trình 8.11

```
/* prog8_11.c                                                    */
#include <stdio.h>
#define  MAX_CMP_NAMES  5
typedef char string[BUFSIZ];

int main(void)
{
string database(MAX_COMP_NAMES)=("Res 1","Res 2",
                              "Res 3","Res 4","Cap 1");
int      i
    for (i=0;i<MAX_COMP_NAMES;i++)
       printf(*%s\n",database[i];
    return(0);
}
```

Chạy thử 8.10

```
Res     1
Res     2
Res     3
Res     4
Cap     1
```

8.7 THỰC HÀNH

Q8.1 Hãy viết một chương trình có khả năng tiếp nhận một xâu text (văn bản) và xác định giá trị của hai điện trở được nhập vào. Khuôn mẫu của xâu được nhập vào là:

RES_NAME1 COMPONENT_VALUE1 RES_NAME2 COMPONENT_VALUE2

Chương trình sẽ xác định điện trở tương đương của các điện trở mắc nối tiếp.

Kết quả chạy thử 8.11 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.11

```
Enter string: R1 244 R2 300
Two resistors are R1=244; R2=300, total is 544 ohms
Enter string: R15 3000 R20 534
Two resistors are R15=3000,R20=535,total is 3534 ohms
Enter string: quit
```

Q8.2 Hãy viết một chương trình khai báo bảy xâu sau đây:

```
"RL series", "RC series", "LC series", "RL parallel",
"RC parallel", "LC parallel", "EXIT"
```

Lưu trữ các xâu này như một mảng đơn của các xâu được đặt tên là menu (thực đơn) bằng cách khai báo một mảng của xâu. Chương trình sẽ hiển thị các xâu này như những thực đơn tùy chọn bằng cách sử dụng một vòng lặp `for()`. Kết quả chạy thử 8.12 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.12

```
Menu Options
RL series
RC series
LC series
RL parallel
RC parallel
LC parallel
```

```
QUIT
```

Q8.3 Hãy sửa đổi chương trình trong Q8.2 sao cho người dùng có thể nhập vào thực đơn tùy chọn (option). Chương trình sẽ hiển thị một thông báo trên thực đơn tùy chọn đã được lựa chọn. Kết quả chạy thử 8.13 chỉ ra một thí dụ làm mẫu.

Chạy thử 8.13

```
Menu options
  RL  series
  RC  series
  LC  series
  RL  parallel
  RC  parallel
  LC  parallel
  QUIT
Enter option >> RL series
  >>> RL series circuit selected
Menu options
  RL  series
  RC  series
  LC  series
  RL  parallel
  RC  parallel
  LC  parallel
  QUIT
Enter option >> QUIT
```

Q8.4 Viết một hàm có tên `get_int()` có khả năng đọc một xâu từ bàn phím. Khi đó hàm này sẽ quét một giá trị nguyên (integer) trong phạm vi giữa giá trị cực đại và cực tiểu đã được xác định.

Q8.5 Hãy sửa đổi một vài chương trình trong các chương trước sao cho các tham số của chương trình có thể nhập vào được bằng cách sử dụng hàm `get_float()`.

Q8.6 Hãy viết một hàm có khả năng viết bằng chữ hoa tất cả các ký tự có trong một xâu

Q8.7 Lặp lại Q8.6 nhưng làm cho ký tự được viết bằng chữ thường (không phải chữ hoa).

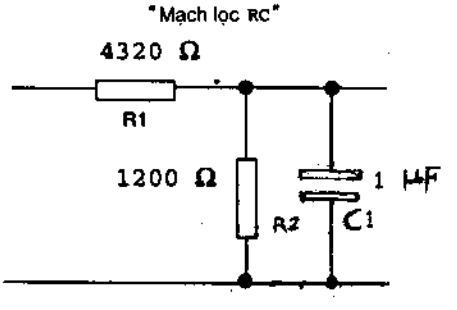
Q8.8 Viết một hàm có khả năng xác định số lượng các từ trong một xâu.

Chương 9
CÁC CẤU TRÚC

Cấu trúc là một đối tượng có khả năng nhận biết được, trong đó có chứa các mục (term) đặc trưng cho cấu trúc. Các mục này được liên kết thành một nhóm chung. Chẳng hạn, một mạch điện có một số tính chất có thể đặc trưng cho nó. Chẳng hạn:

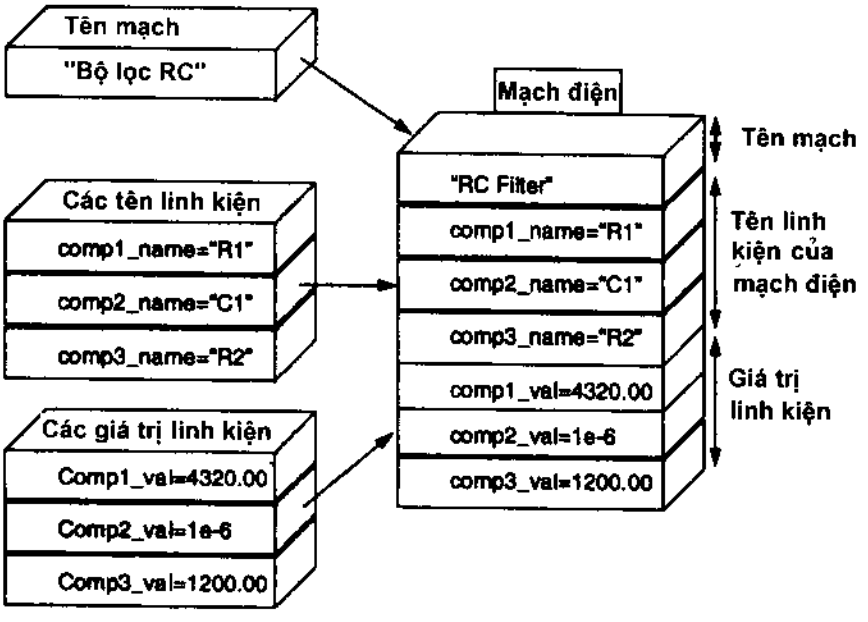
- tên mạch điện;
- các phần tử của mạch với tên gọi mang tính đặc trưng;
- các phần tử của mạch với giá trị đã biết.

Hình 9.1 chỉ ra một mạch lọc RC cơ bản. Tên của mạch này là `"mạch lọc RC"`, các phần tử (linh kiện) của mạch có tên `"R1"`, `"R2"` và `"C1"` và các giá trị của các linh kiện này tương ứng là `4320Ω`, `1200Ω` và 1μF. Tên mạch điện và tên của các phần tử là các xâu ký tự, trong khi giá trị các phần tử là dấu phẩy động. Một cấu trúc nhóm lại các tính chất này thành một thực thể đơn. Có thể hình dung nhóm này như được so sánh với các trường và mỗi trường được hình thành từ những thành viên (member).



Hình 9.1: Mạch lọc RC

Hình 9.2 cho thấy các trường có thể được liên kết như thế nào để tạo ra một cấu trúc đơn. Trong trường hợp này, có ba trường: tên mạch điện, tên các phần tử và giá trị của các phần tử.



Hình 9.2: **Nhóm lại các trường bên trong một cấu trúc.**

Một cấu trúc là một kiểu, hỗn hợp của các phần tử riêng biệt và thường thì các phần tử này có kiểu dữ liệu khác nhau. Sau đây sẽ đưa ra một thí dụ về cách khai báo một cấu trúc (trong trường hợp này các xâu đã được khai báo với nhiều nhất là 100 ký tự).

```
struct
{
  char    title(100);
  char comp1_name[100], comp2_name[100], comp3_name[100];
  float comp1_val, comp2_val, comp3_val; )
} circuit;
```

Sau đây là một thí dụ về một cấu trúc có khả năng lưu trữ một phần tử đơn của mạch điện. Biến cấu trúc được khai báo, trong trường

hợp này, là component (phần tử). Phần tử có ba trường với các kiểu dữ liệu khác nhau: cost (dấu phảy động: float), code (số nguyên) và name (mảng ký tự).

```
struct
{
float  cost;
int    code;
char   name[BUFSIZ];
} Component;
```

Ký hiệu điểm (.) truy nhập tới từng thành viên bên trong cấu trúc. Chẳng hạn:

```
Component.cost=12.3
Component.code=31004;
strcpy(Component.name,"Resistor 1");
```

Chương trình 9.1 là một chương trình cơ sở dữ liệu đơn giản. Cơ sở dữ liệu cất giữ một bản ghi của một phần tử đơn lẻ của mạch điện bao gồm: tên (Component.name), giá (Component.cost) và số mã (Component.code). Tên là một xâu, mã là một số nguyên có dấu và giá thành là một giá trị với dấu phẩy động.

Chương trình 9.1

```
/* prog9_1.c                                          */
#include <stdio.h>
#include <string.h>

int main(void)
{
struct

    float   cost;
    int     code;
    char    name[BUFSIZ];
} Component;
      Component.cost=30.1;
      Component.code=32201;
```

```
        strcpy(Component.name,"Resistor 1");
        printf("Name %s Code %d Cost %f\n";
            Component.name,Component.code,Component.cost);
        return(0);
}
```

Kết quả chạy thử 9.1 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.1

```
Name resistor 1 Code 32201 Cost 30.10000
```

Chú ý rằng cấu trúc có thể đã được thiết lập khi khởi tạo trạng thái ban đầu bằng cách sử dụng các dòng lệnh sau đây:

```
struct
{
   float  cost;
   int    code;
   char   name[BUFSIZ];
} Component={30.1, 32201, "Resistor 1"};
```

Chương trình 9.2 có chứa một hàm để xuất ra (in) cấu trúc (`print_component()`). Để chuyển giao một cấu trúc vào trong một hàm, kiểu dữ liệu của tham số chuyển giao phải được định nghĩa. Muốn thế, từ khóa `typedef` được sử dụng để định nghĩa một kiểu dữ liệu mới, trong trường hợp này có tên là `CompType`. Chương trình cũng sử dụng dấu ngoặc nhọn để khởi tạo các trường bên trong cấu trúc.

Chương trình 9.2

```
/* prog9_2.c                                          */
 #include <stdio.h>
 #include <string.h>

 typedef struct
 {
     float    cost;
```

```
        int       code;
        char      name[BUFSIZ];
}       CompType;
void print_component(CompType Comp);
int     main(void)
{
CompType component={30.1,32201,"Resistor 1');
     print_component(Component);
     return(0);
}
void    print_component(comptype Comp)
{
      printf("Name %s Code %d Cost %f\n",
          comp.name,Co.p.code,Comp.cost),
}
```

Kết quả chạy thử 9.2 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.2

```
Name resistor 1 Code 32201 Cost 30.10000
```

Chương trình 9.3 sử dụng một hàm để đưa dữ liệu vào trong cấu trúc (get_component()) tham số được chuyển giao vào trong hàm này sẽ là một con trỏ chỉ vào địa chỉ cơ sở của cấu trúc. Muốn thế, một ký hiệu "VÀ" này được chèn vào trước tên của cấu trúc. Toán tử con trỏ cấu trúc (->) được sử dụng với con trỏ cấu trúc để truy nhập lên một thành viên của trường.

Chương trình 9.3

```
/* prog9_3.c                                              */
#include <stdio.h>
#include <string.h>

typedef struct
{
        float  cost;
```

```
        int    code;
        char   name[BUFSIZ];
 } CompType;
void   get_component(comptype *Comp);
void   print_coponent(CompType Comp);
int    main(void)
{
CompType Component;
     get_component(&Componenr);
     print_component(Component);
     return (0);
}

void   get_component(CompType *Comp)
{
    Comp->cost=30.1;
    Comp->code=32201;
    strcpy(Comp->name,"Resistor 1");
 }
void   print_componenr(CompType Comp)
{
       printf("Name %s Code %d cost %f\n",
            Comp.name,Comp.code,Comp.cost);
}
```

Kết quả chạy thử 9.3 chỉ ra kết quả giống hệt với lần chạy vừa qua.

Chạy thử 9.3

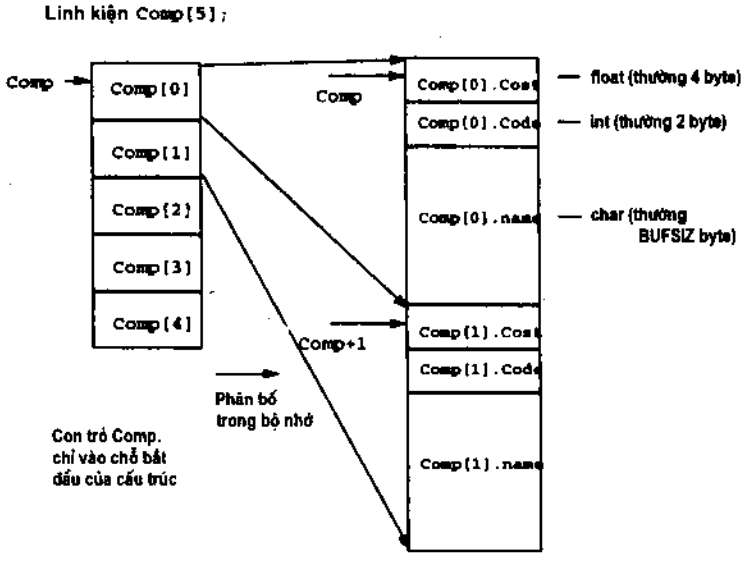
```
Name resistor 1 Code 32201 Cost 30.10000
```

9.1 MẢNG CỦA CẤU TRÚC

Một mảng của các cấu trúc có thể được thiết lập theo cách giống với việc đánh số (chỉ số hóa) mảng thông thường. Khi một mảng được khai

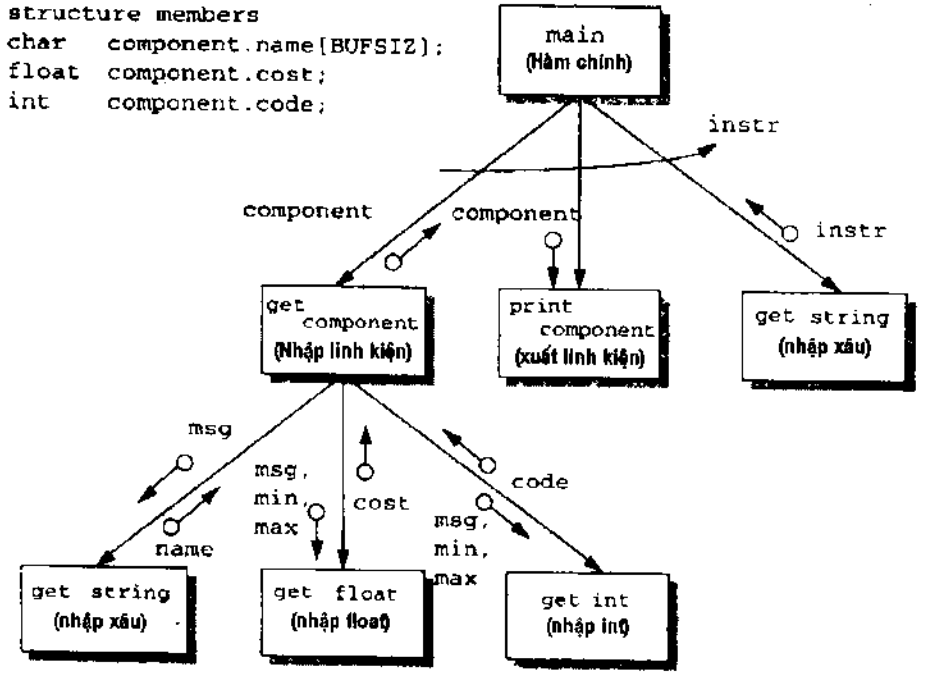
báo, chương trình dịch sẽ gán đủ bộ nhớ để giữ tất cả các phần tử của mảng. Nếu như có đủ bộ nhớ, tên mảng cấu trúc được gán cho địa chỉ bắt đầu của mảng (địa chỉ cơ sở). Bộ nhớ này bao gồm các ô nhớ kề sát nhau và chỉ một con trỏ được sử dụng để so sánh tất cả các phần tử bên trong cấu trúc.

Chương trình 9.4 tương tự với chương trình 9.3, nhưng sử dụng một mảng của các cấu trúc để cất giữ tới 5 linh kiện (phần tử mạch điện). Hình 9.3 cho thấy cấu trúc này được sắp xếp như thế nào trong bộ nhớ. Trong chương trình này cấu trúc không được chuyển giao với ký hiệu "VÀ" đặt trước (tiền tố) bởi vì tên cấu trúc là địa chỉ cơ sở của cấu trúc (tương tự với chuyển giao mảng). Mỗi hàm, sử dụng cấu trúc `Components`, có thể xác định xem mỗi phần tử bên trong cấu trúc được truy nhập như thế nào khi biết kiểu dữ liệu của mỗi các thành viên (thí dụ số byte mà các thành viên chiếm trong bộ nhớ). Chẳng hạn, khi so sánh với hình 9.3, phần tử mảng cấu trúc thứ hai sẽ là `4+2+BUFSIZ` byte tính từ con trỏ cơ sở (`Comp[1]` hoặc `Comp+1`), thứ ba là `2*(4+2+BUFSIZ)` byte tính từ con trỏ cơ sở (`Comp[2]` hoặc `Comp+2`) v. v...



Hình 9.3: **Sắp xếp của một mảng các cấu trúc.**

Hình 9.4 minh hoạ biểu đồ cấu trúc của chương trình này. Nó sử dụng hàm `get_float` và `get_int()` để lọc bất kỳ giá trị đầu vào nào không hợp lệ đối với chi phí và mã của một phần tử. Các hàm này đã được giới thiệu trong chương 8 và đã được sử dụng lại bởi vì chúng được kiểm tra kỹ và dễ dàng chuyển vào một chương trình bất kỳ. Giá tiền của các linh kiện bị giới hạn giữa 0 và 1.000 còn mã thì giữa 0 và 32.767. Hàm `get_string()` cũng được bổ sung để nhận được tên của các phần tử.



Hình 9.4: **Biểu đồ cấu trúc dùng cho chương trình 9.4.**

Chương trình 9.4

```
/*  prog9_4.c                                          */
#include        <string.h>
#include        <stdio.h>

#define  TRUE             1
#define  FALSE      0
#define MAXCOMPONENTS     5
```

```
typedef struct
{
      float    cost;
      int code;
      char   name[BUFSIZ];
} CompType;

void      get_component(int *n,CompType Comp[]);
void      print_component(int n,CompType comp[]);
void      get_int (char msg [] , int min, int max, float *val)
void      get_float(char msg[],float min,float max,float *val);
void      get_string(char msg[],char ins[]);

int       main(void)
{
int       NumComponents=0;
char      instr[BUFSIZ];
CompType Components[MAXCOMPONENTS];
    do
    {
       get_component(&NumComponmts,Component);
       print_component(NumComponents,Component);
       get_string ("Do you wish to continue (type 'exit' to leave)",
                  instr);
    } while (strcmp(instr,"exit")!=0) ;
    return(0);
}
void   get_component(int *n,CompType Comp[1]
{
int   i;

      i=*n;
      get_string("Enter name of component >> ",Comp[i] name);
      get_float("Enter cost of component >> ",0.0,1000.0,
```

```
                &Comp[i].cost);
     get int("Enter component code >> ,0,32767,&Comp[i] code);
     (*n)++;
}
void   print_component(int n,CompType Comp[])
{
int      i;
   for (i=0;i<n;i++)
      printf("Name %12s Code %8d Cost %8.2f\n",
         comp[i].name,Comp[i].code,Comp[i].cost):
}
void    get_string (char msg[], char ins [])
{
   printf("%s",msg);
   gets(ins);
   }
void     get_float(char msg [], float min, float max, float *val)
(
char    inline[BUFSIZ];
int     rtn,okay;

  do
  {
     printf("%S",msg);
     gets(inline);
     rtn=sscanf(inline,"%f",val) ;
     if((rtn!=1) || (*val<min) || (*val>max))
     {
          okay=FALSE;
          printf("Invalid input <%s>\n",inline);
     }
     else okay=TRUE:
   } while (!okay);
}
void   get_int(char msg(],int min,int max,int *val)
```

```
{
char   inline[BUFSIZ];
int    rtn,okay;
  /*get an integer value between min and max */
  do
  {
    printf("%s"msg);
    gets(inline);
    rtn=sscanf(inline,"%d", val);
    if((rtn!=1) || (*val<min) || (*val>max))
    {
          okay=FALSE;
          printf("Invalid input <%s>\n",inline) ;
    }
    else  okay=TRUE;

  } while (!okay);
}
```

Kết quả chạy thử 9.4 chỉ ra một thí dụ mẫu.

Chạy thử 9.4

```
Enter name of component >> resistor 1
Enter cost of component >> 1.23
Enter component code >> 32455
Name resistor 1 code    32455 Cost   1.23
Do you wich to continue (type 'exit' to leave)y
Enter name of component >> capacitor 1
Enter cost of component >> 2.58
Enter component code  >> 32456
Name resistor 1 Code 32455 Cost   1.23
Name capacitor 1 Code 32456 Cost  2.68
DO you wish to continue (type 'exit' to leave)y
Enter name of component >> inductor 2
Enter cost of component >> 6.32
Enter component code >> 32457
```

```
Name resistor 1 Code  32455 Cost 1.23
Name capacitor 1 Code 32456 Cost 2.68
Name inductor 2 Code  32457 Cost 6.32
Do you wish to continue (type 'exit' to leave)exit
```

Chương trình 9.4 có một vài điểm không bình thường. Một trong các điểm bất thường là chương trình có khả năng chấp nhận nhiều hơn 5 linh kiện. Điều này có thể gây ra các vấn đề khi bộ nhớ chỉ đủ cho 5 linh kiện dẫn đến chương trình bị treo hoặc diễn biến theo hướng không dự đoán trước được. Vấn đề này sẽ còn được đề cập trong chương trình tiếp theo.

Chương trình 9.5 có sử dụng hàm `get_component()` để kiểm tra số các phần tử trong cấu trúc mảng. Nếu có nhiều phần tử hơn số đã khai báo thì nó sẽ trả lại `DB-FULL`, nếu không nó sẽ trả lại `!DB_FULL`. Giá trị đưa trở lại này được kiểm tra trong phần `main()`. Nếu giá trị trả lại là `DB_FULL` thì thông báo `Database is full, now exiting` sẽ được hiển thị, nếu không nó sẽ xuất ra nội dung của cơ sở dữ liệu.

Chương trình 9.5

```
/* prog9_5.c                                        */
#include        <string.h>
#include        <stdio.h>
#define TRUE            1
#define FALSE           0
#define DB_FULL         1
#define MAXCOMPONENTS   5

typedef struct
{
      float   cost;
      int     code;
      char    name [BUFSIZ];
 } CompType;
 int    get_component(int *n,CopmType Comp[]);
 void   print_component(int n.CompType Comp[]);
```

```
 void   get_inc(char msg[],int min,int max,int *val);
 void   get_float(char msg[],float min,float max,float *val);
 void   get-string(char msg[],char ins[]);

int  main(void)
{
int      NumComponents=0;
char     instr[BUFSIZ];
CompType   Component[MAXCOMPONENTS];
 do
 {
      if   get_component(&NumComponents, Component)==DB_FULL)
      {
      puts("Database is now full");
      break;
   }
     print component(NumComponents,Component);
     get_string("Do you wish to continue (type 'exit' to leave)",
                 instr);
   } while (strcmp(instr,"exit")!=0);
   return (0);
}
int get_component(int *n,CompType Comp[])
{
int  i;
    i=*n;
    if (i>=MAXCOMPONENTS) return(DB_FULL);
    get_string("Enter name of component >>" Comp[i].name);
    get_float("Enter cost of component  >>",0,1000,&COMP[i].cost);
    get_int ("Enter component code >> " 1,0,1000,&Comp[i].code);
    ("n)++
    return (!DB_FULL);

 void  print_component(int n,CompType Comp[])
 {
```

```
  : : : : see Program 9.4
{
void get_string(char msg[],char ins[])
  : : : : see Program 9.4
void get_float(char msg[],float min,float max,float *val)
  : : : : see Program 9.4
  }
void get_int(char msg[],int min,int max, int*val)
{
  : : : : see Program 9.4
}
```

Chương trình 9.6 chỉ ra một thí dụ cho thấy một mảng cấu trúc có thể được khởi tạo như thế nào ở điểm đã khai báo. Một macro MAXSTRING đã được thiết lập để định nghĩa kích thước mảng cực đại (bình thường thì BUFSIZ được sử dụng khi nhập vào từ bàn phím).

Chương trình 9.6

```
/* prog9_6.c                                              */
#include <stdio.h>
#define  MAXSTRING    100
#define  COMPONENTS    5

typedef struct
{
 char name [MAXSTRING];
   float value;
   char unit[MAXSTRING];
} Compname;

int main(void)

{
Compname comp[COMPONENTS]={
     {"R1",43,"Kohm"),("C2",1.02,"uF"),
     ("R2",100,"Mohm"),("C1",2.6,"uF"},
     ("R3',1.2,"Kohm"}};
```

```
int
  for (i=0;i<COMPONENTS;i++)
     printf("%10s %8.2f %10s\n",
   comp[i].name,comp[i].value,comp[i].unit);
  return(0);
}
```

Kết quả chạy thử 9.5 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.5

```
R1        43.00      Kohm
C2         1.02      uF
R2       100.00      Mohm
C1         2.60      uF
R3         1.20      Mohm
```

9.2 SẮP XẾP CẤU TRÚC THEO KIỂU ĐỘNG

Mỗi lần một kiểu dữ liệu mới được tạo ra đối với một cấu trúc thì cách đơn giản để phân bố bộ nhớ cho cấu trúc là tiến hành theo kiểu động. Cách phân bố này đạt được bằng cách khai báo tên cấu trúc như một con trỏ, và sau đó bộ nhớ được phân bố bằng cách sử dụng một hàm phân bố bộ nhớ, như `malloc()`. Giá trị trả lại từ hàm này lúc đó được gán cho con trỏ cấu trúc. Nếu giá trị trả lại từ `malloc()` là `NULL` (không) nghĩa là không có đủ bộ nhớ cho cấu trúc. Chương trình 9.7 chỉ ra một thí dụ về một mảng của các cấu trúc được phân bố động như thế nào.

Chương trình 9.7

```
/* prog 9_7.c                                          */
#include        <string.h>
#include        <stdio.h>
#include <alloc.h> /* required for malloc() */

#define  TRUE           1
```

```
#define  FALSE      0
#define  DB_FULL          1
#define MAXCOMPONEWS      5

typedef struct
{
      float     cost;
      int code;
      char      name[BUFSIZ];
} CompType;

int      get_component(int *n,CompType Comp[];
void     print_component(int n,CompType comp[]);
void     get_int (char msg [],int min, int max, int *val)
void     get_float(char msg[],float min,float max,float *val);
void     get_string(char msg[],char ins[]);

int       main(void)
{
int       num_components=0;max_comp;
char      instr[BUFSIZ];
CompType *component;
   do
   {
if(get_component(&Num_components,component)==DB_FULL);
{

        puts(Database is now full*);
        break;
    }
print_component(Num_components,component);
get_string("Do you wish to continue (type 'exit' to leave)",instr);
} while (strcmp(instr,"exit")!=0);
return(0);
}
```

```
int   get_component(int *n,CompType Comp[])
{
int  i;
   i=*n;
   if (i>=MAXMCOMPONENTS) return(DB.FULL);

get_string("Enter name of component >> ",Comp[i]. name);
get_float("Enter cost of component >> ",0.0,1000.0,
              &Comp[i].cost);
get_int("Enter component code >> ",0,1000,&comp[i]. code);
(*n)++
return(!DB_FULL);
}
void   print_component(int n,CompType Comp[])
{
:  :  :  : see Program 9.4
}
void     get_string(char msg[],char ins[]
{
:  :  :  : see Program 9.4
}
void   get_float(char msg[],float min, float max,float *val)
{
:  :  :  : see Program 9.4
}
void     get_int(char msg[],int min,int max,int *val)
{
:  :  :  : see Program 9.4
}
```

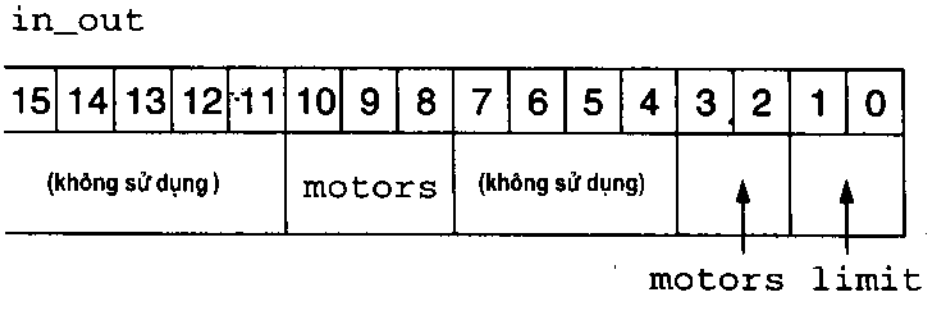
9.3 TRƯỜNG BIT

Cấu trúc có thể được sử dụng để xác định các trường bit trong giới hạn integer (có dấu hoặc không dấu). Các trường bit này có thể thay đổi độ rộng từ 1 đến 16 bit (hoặc nhiều hơn tuỳ thuộc vào sự bổ sung của

trình dịch). Dấu hiệu để nhận dạng trường bit được định nghĩa sau kiểu dữ liệu và độ rộng của trường bit được chỉ định rõ sau dấu hai chấm. Các quy ước này được chỉ ra trong thí dụ dưới đây. Nếu không có dấu nhận dạng trường bit thì các bit được định nghĩa trong trường bít ấy không được sử dụng.

```
struct
{
  int limit       :2;
        /* Bit fields 0 and 1 used for limit   */
  int sensors     :2;
        /* Bit fields 2 and 3 used for sensors *1
  int             :4;
        /* Bit fields 4,5,6 and 7 are =used*/
  unsigned int     motors     :3;
       /*Bit fields 8,9 and 10 are used for motors*/
} in-out;
```

Sự sắp xếp các bit bên trong cấu trúc `in-out` được mô tả trên hình 9.5.



Hình 9.5: **Sắp xếp trường bit dùng cho cấu trúc trong - ngoài.**

Các trường bit được truy nhập khi sử dụng toán tử điểm, thí dụ các trường bit 8 - 10 được truy nhập bởi `in_out.motors`. Nếu kiểu trường bit là có dấu thì bit có giá trị cao nhất của trường là cờ dấu bởi vì các số nguyên có dấu được cất giữ trong ký hiệu lấy bù theo 2. Chẳng hạn, trong trường bit số nguyên có dấu, 3 bit giá trị -1 (thập phân) sẽ được

cất giữ như 111 (nhị phân), trong khi một trường **không dấu** 3 bit 111 (nhị phân) tương ứng với 7 (thập phân).

Như vậy, phạm vi của một trường có dấu, 3 bit từ 011 (3) tới (-4) còn của một trường 3 bit không dấu từ 111 (7) đến 000 (0). Bảng 9.1 chỉ ra các phạm vi thập phân khác nhau dùng cho một khuôn mẫu nhị phân, 3 bit.

Bảng 9.1: **Các giá trị thập phân 3 bit có dấu và không dấu.**

Khuôn mẫu nhị phân 3 bit	Giá trị thập phân có dấu (int)	Giá trị thập phân không dấu (unsigned int)
011	3	3
010	2	2
010	1	1
000	0	0
111	-1	7
110	-2	6
101	-3	5
100	-4	4

Bảng 9.2 chỉ ra một thí dụ về dãy các tác động trên các trường bit của in_out (trong - ngoài). Dãy này được giả thiết là thoạt đầu tất cả các trường bit đều được đặt bằng 0.

Bảng 9.2: **Các tác động mẫu trên cấu trúc trong - ngoài.**

Lệnh	Mẫu bit của in_out sau lệnh
in-out.limit=1;	000000 000 000 00 01
in-out.sensors=-2;	000000 000 000 10 01
in-out.motors=7;	000000 111 000 10 01

Chương trình 9.10 và 9.11 cho ta các thí dụ về các chương trình sử dụng các trường bit.

9.4 CẤU TRÚC TIME

Một thí dụ về một cấu trúc đã được định nghĩa sẵn trong C là cấu trúc thời gian, tiếng Anh: *time* và được viết tắt là `tm`. Cấu trúc này được định nghĩa trong *time.h* và có thể được sử dụng để lưu trữ số liệu về ngày tháng và thời gian. Một số hàm thời gian chuẩn đã sử dụng cấu trúc này, như `gmtime()`, `asctime()` và `localtime()`. Khuôn mẫu cơ bản của cấu trúc này được chỉ ra dưới đây.

```
struct    tm
{
   int    tm-sec;     /* seconds (0-59)              */
   int    tm_min;     /* minutes (0-59)              */
   int    tm_hour;    /* hours   (0-23)              */
   int    tm-mday;    /* day of the month(1-31)      */
   int    tm-wday;    /* day of week (0-6) Sunday is 0   */
   int    tm-mon;     /* month of the year (0-11) */
   int    tm-year;    /* calendar year (year minus 1990) */
   int    tm_yday     /* day of the year (0-364)  */
}
```

Chương trình 9.8 là một chương trình đơn giản trong đó người dùng nhập vào tên tháng dưới dạng một số nguyên (cụ thể là 1 đến 12) và ngày hiện tại của tháng. Chương trình sẽ hiển thị ngày tháng trong khuôn mẫu DD-MMM, trong đó MMM là chữ viết tắt của 3 chữ cái đầu tiên của tên tháng (thí dụ `JAN`, `FEB`, v. v...). Chương trình kiểm tra con số chỉ các ngày trong tháng và sẽ hiển thị một thông báo lỗi nếu như ta đã nhập vào số chỉ ngày vượt quá số các ngày trong tháng đã được chỉ định. Nó cũng sẽ hiển thị một lỗi nếu như vào số chỉ ngày lại nhỏ hơn hơn 1. Số các ngày trong mỗi tháng được thiết lập bằng cách sử dụng một mảng `days_in_month[ ]` (thí dụ 31 cho tháng giêng, 28 cho tháng hai, v. v...). Chú ý là chương trình đã giả thiết rằng năm đang tính không phải là một năm nhuận*)

*) Cứ bốn năm lại có một năm mà tháng hai có 29 ngày, năm đó được gọi là năm nhuận (ND)

Chương trình 9.8

```
/* prog9_8.c                                                    */
#include <stdio.h>
#include <time.h>

#define NMONTHS    12 /* Number of months in a year    */
#define TRUE       1
#define FALSE      0

void      get_int(char msg[],int min,int max,int *val);

int main(void)
{
struct    tm     t;
int days_in_month[NMONTHS]={31,28,31,30,31,
          30,31,31,30,31,30,31};
char *months(NMONTHS)={"JAN","FEB","MAR","APR","MAY",
          "JUN","JUL","AUG","SEP","OCT","NOV","DEC");
    get_int("Enter current month (1-12) >>",1,12,&t.tm_mon);
    get-int("Enter day of the month >>",
        1,days_in_month[t.tm_mon-1],&t.tm_mday);
    printf("Date is %02d-%s\n",t.tm_mday,months(t.tm_mon-1]);
    return(0);
}
void    get_int(char msg[],int min,int max,int *val)
: : : :  see Program 9.4
}
```

Chạy thử 9.6 chỉ ra một kết quả chạy chương trình này làm thí dụ. Người dùng đã nhập vào một giá trị bằng 2 cho tháng (nghĩa là tháng hai) và sau đó nhập vào ngày của tháng là 31. Con số này là không hợp lệ, vì tháng hai không có ngày 31, và người dùng được nhắc là phải nhập một ngày khác của tháng. Chương trình có khả năng kiểm tra con số chỉ ngày trong tháng như giá trị `max` gửi tới `get_int()` được đặt bằng mảng `day_in_month[]`. Bởi vì giá trị được nhập vào là 2 nên giá trị

cực đại được gửi tới sẽ là 28 (chú ý rằng 1 được trừ đi từ số chỉ tháng được nhập vào trong khi 0 lại được chấp nhận là tháng đầu tiên).

Chạy thử 9.6

```
Enter current month (1-12)>>2
Enter day of the month >>31
Invalid input <31>
Enter day of the month >>-3
Invalid input <-3>
Enter day of the month >>23
Date is 23-FEB
```

Chương trình 9.9 chứa các hàm C tiêu chuẩn, có thể được sử dụng để tạo ra một cấu trúc tm, đó là `time()` và `gmtime()`. Hàm `time()` xác định thời gian hệ thống hiện thời còn hàm `gmtime()` biến đổi thời gian này thành một khuôn mẫu cấu trúc `tm`. Cú pháp của các hàm như sau:

```
time_t:      time(time_t *timer)
struct tm    *gmtime(time_t *timer)
```

Hàm `time()` xác định số giây đã chuyển giao kể từ 00:00:00 giờ GMT của ngày 1 tháng giêng năm 1970. Giá trị quay lại được trả lại như một con trỏ (*time) và cũng qua đầu mục hàm (function header), kiểu dữ liệu của giá trị trả lại là `time_t`, cũng đã được định nghĩa trong *time.h*. Giá trị đưa trở lại này được chuyển giao tới `gmtime` làm đầy cấu trúc `ttt`. Biến `ttt` được khai báo như một con trỏ chỉ tới cấu trúc `tm` và hàm `gmtime()` dự trữ đủ không gian trong bộ nhớ dành cho cấu trúc và trả lại một con trỏ tới chỗ bắt đầu của nó.

Chương trình 9.9

```
/* prog9_9.c                                        */
#define   MAXSTRING  30
  /* Khuôn mẫu mảng là ngày tháng hiện tại    */
  /* và được viết dưới dạng XX-YY-ZZ          */
int main(void)
{
time_t      timer;
```

```
struct tm  *ttt;
char       str[MAXSTRING];

    time(&timer); /* get number of seconds since 0:0:0 1/1/70  */
    ttt=gmtime (&:timer); /* convert seconds into tm format */

    sprintf(str,"Current Date is t02d-%02d-%02d',
  ttt-gtm_mday,ttt->tm_mon,ttt->tm_year);

    puts(str);

    return(0);
}
```

Kết quả chạy thử 9.7 chỉ ra một thí dụ làm mẫu; trong trường hợp này ngày tháng hiện tại là 5 tháng 9 1994.

Chạy thử 9.7

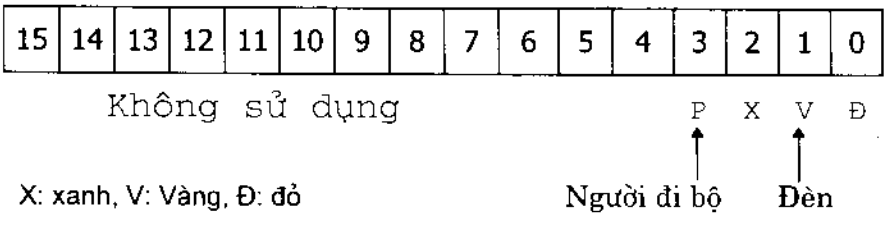
```
Current Date is 05-09-94
```

9.5 MỘT SỐ THÍ DỤ

9.5.1 SA BÀN ĐÈN GIAO THÔNG

Chương trình 9.10 tạo ra một sa bàn đèn giao thông cơ bản với các đèn điều khiển người đi bộ. Một cấu trúc có tên là `traffic_bit_field` lưu trữ các bit dùng cho trạng thái đèn giao thông: đỏ, vàng và xanh, trong đó 0 - `OFF` (tắt) còn 1 - `ON` (bật) và tín hiệu cho người đi bộ: 0 - không được đi, còn 1 - đi được. Trường bit đèn giao thông được chỉ ra trên hình 9.6. Các bit 0 - 2 được sử dụng cho màu đèn còn bit 3 là hiệu lệnh cho người đi bộ.

`traffic_bit field`: Trường bit giao thông



Hình 9.6: **Sự sắp xếp trường đối với cấu trúc traffic _bit_field.**

Một hàm `show_sequence()` được sử dụng để che từng bit bên trong cấu trúc và hiển thị trạng thái của chúng. Bốn mặt nạ bit đã được thiết lập, đó là: đỏ, vàng và xanh (lá cây). Mặt nạ bit đỏ tác động lên bit đầu tiên trường đèn, vàng lên bit thứ hai còn xanh cho bit thứ ba. Chẳng hạn, nếu như `traffic_lights & đỏ` là TRUE thì bit red (đỏ) được đặt, nếu không nó sẽ là FALSE (số 0 là FALSE trong khi một TRUE là một giá trị khác bất kỳ). Bảng 9.3 chỉ ra sơ đồ cần có.

Bảng 9.5: **Đèn giao thông và sơ đồ điều khiển đi bộ.**

Đèn giao thông 1	Đèn giao thông 2	Người đi bộ
Đỏ	Đỏ	Không được đi
Đỏ	Đỏ và vàng	Không được đi
Đỏ	Xanh	Không được đi
Đỏ	Vàng	Không được đi
Đỏ	Đỏ	Đi bộ qua
Đỏ và vàng	Đỏ	Không được đi
Xanh	Đỏ	Không được đi
Vàng	Đỏ	Không được đi
Đỏ	Đỏ	Đi bộ qua
Đỏ	Đỏ và vàng	Không được đi
	vân vân...	

Chương trình 9.10

```
/* prog9_10.c                                          */
#include <stdio.h>

#define RED          1 /* 1st bit fielf */
#define AMBER        2 /* 2nd bit fielf */
#define GREEN        4 /* 3rd bit field */

#define FALSE        0
#define TRUE         1

typef  struct
{
```

```
unsigned int lights :3; /*GAR */
unsigned int pedestrian :1; /*P */
} traffic_bit_field; /* xx xxxx xxxx PGAR

void   show_sequence(traffic_bit-field traf);
int    main(void)
traffic-bit_field traffic;
char   ch;
  do
  {
       traffic.lights=RED;
       traffic.pedestrian=FALSE;      /* 0 001 */
       show_sequence(traffic);
       ch=getchar(); /* press any key to continue"/

       traffic.lights=RED+AMBER;      /* 0 011 */
       show_sequence(traffic);
       ch=getchar(); /* press any key to continue*/

       traffic.lights=GREEN;          /* 0 100 */
       show_sequence(traffic);
       ch=getchar() ; /* press any key to continue*/

       traffic.lights=AMBER;            /* 0 010 */
       show_sequence(traffic);
       ch=getchar(); /* press any key to continue */

       traffic.lights=RED:
       traffic.pedestrian=TRUE;
       show_sequence (traffic);
       printf("Press 'x' to exit >> "); ch=getchar();
   } while (ch!='x');
   return(0);
}
void   show-sequence(traffic_bit_field traff)
```

```
{
    if (traff.lights & RED)     printf("RED");
    if (traff.lights & AMBER)   printf("AMBER");
    if (traff.lights & GREEN)   printf("GREEN");

     if (traff.pedestrian==FALSE) puts("<DONT WALK>");
       else puts("<WALK>");
}
```

Kết quả chạy thử 9.8 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.8

```
RED <DONT WALK>
RED AMBER <DONT WALK>
GREEN <DONT WALK>
AMBER <DONT WALK>
RED <WALK>
Press X, to exit >>
RED <DONT WALK>
RED AMBER <DONT WALK>
GREEN <DONT WALK>
AMBER <DONT WALK>
RED <WALK>
Press 'x' to exit >> x
```

Chương trình 9.11 có một cấu trúc được cải thiện và sử dụng một mảng để cất giữ sa bàn đèn giao thông và người đi bộ.

Chương trình 9.11

```
/* prog_11.c                                          */
/* program to display a traffic light sequence  */
/* with a pedestrian crossing                         */

#include  <stdio.h>

#define    RED       1
```

```
#define    AMBER     2
#define    GREEN     4
#define    NO_SEQ    6

#define    FALSE     0
#define    TRUE      1

typedef struct
{
  unsigned int lights        :3;
  unsigned int pedestrian        :1;
} traffic_bit_field;
void   show_sequence(traffic_bit_field traf);

int    main(void)
traffic_bit_field traffic;
char     ch;
int   lights_seq[NO_SEQ]=(RED,RED.AMBER,GREEN,AMBER,RED,RED);
int   pedestrian_seq[NO-SEQ]=(FALSE,FALSE,FALSE,FALSE,TALSE,TRUE);
int   i;
   puts("Traffic lights seqence program")
   puts("Enter 'X' to exit the program")
   do
   {
     for (i=0;i<NO-SEQ;i++)
     {
          traffic.lights=lights_seqt[i];
          traffic.pedestrian=pedestrian_seq[i];
          show_sequence(traffic);
          ch=getchar();
          if (ch=='X') break;
      }
    } while (ch!='x');
    return(0);
}
```

```
void show_sequence(traffic_bit_field traff)
{
     if (traff.lights & RED)          printf("RED")
     if (traff.lights & AMBER)  printf("AMBER")
     if (traff.lights & GREEN)  printf("GREEN")

     if (traff.pedestrian==FALSE)  puts("<DONT WALK>");
     else                           puts("<WALK>");
}
```

Kết quả chạy thử 9.9 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.9

```
RED <DONT WALK>
RED AMBER <DONT WALK>
GREEN <DONT WALK>
AMBER <DONT WALK>
RED <DONT WALK>
RED <WALK>
RED <DONT WALK>
RED AMBER <DONT WALK>
GREEN <DONT WALK>
AMBER <DONT WALK>
RED <DONT WALK>
RED <WALK>
```

9.5.2 TRỞ KHÁNG LỐI VÀO CỦA MẠCH RLC

Chương trình 9.12 sử dụng một cấu trúc để lưu trữ độ lớn và góc của trở kháng mạch RLC mắc nối tiếp. Cấu trúc này có tên là `polar` và nó lưu trữ một giá trị độ lớn và một giá trị góc.

Chương trình 9.12

```
/* prog9_12.c                                          */
/* RLC circuit                                        */
/*  Program to calculate the input impedance of       */
```

```
/* a series RLC circuit                                    */
typedef struct
{
float mag,angle;
} polar;

#include <stdio.h>
#include <math.h>
#define      PI      3.14157
#define      TRUE    1
#define      FALSE   0

#define  -calcXL(-f,-L)     (2*PI(-f)*(-L)
#define  -calcXC(-f,-C)     (1/   (2*PI*(-f)*(-c)))
#define   _calc_mag(-x.-y)     (sqrt((-x)*(-x)+(-y)*(-y)))
#define  _calc_angle(-x,-y)atan2((-y),(-x))
#define  -RadToDeg(_rad)    ((-rad)/(PI)*180)
void     get_values(float *f,float *r,float *1, float *C);
void     calc_impedance(float f.float r,float 1,float c, polar*z)
void     get_float(char msg[],float min,float max.float *val);

int main(void)
{
float    R,L,C,freq;
polar    Zin;

     get_values(&freq,&R,&L,&C);
     calc_impedance(freq,R,L,C,&Zin);

     printf("Zin mag is %.2f angle %.2f degrees\n",
             Zin.mag, _RadToDeg(Zin.angle));
     return(0);
}
void   get_values(float *f,float *r,float *1,float *c)
{
```

```
    get_float("Enter frequency >>",0,100e6,f);
              /* max freq 100 MHz          */
    get_float("Enter resistance >>,0,100e6,r);
              /* max resistance 100 M      */
    get_float ("Enter inductance 0>>",0,1,1);
              /* max inductance 1 mH
    get_float("Enter capacitance >>",0,10e-3,c);
                   /* max cap 1 uF              */
}
void   calc_impedance(float f,float r,float 1,float c, polar *z)
{
float reactance;

    reactance=_calcXL(f,1) - _calcXC(f,c);
    z->mag =_calc_mag(r,reactance);
    z->angle =_calc_angle (r,reactance) ;
}
void      get_float(char msg[1,float min,,loat max,float *val)
{
:::: see Program 9.4
}
```

Kết quả chạy thử 9.10 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.10

```
Enter frequency >> 10e3
Enter resistance >> 3200
Enter inductance >> 10e-3
Enter capacitance >> 1e-6
Zin mg is 3258.07 angle 10.83 degrees
```

9.5.3 CÁC VI MẠCH HỌ 74

Các vi mạch lôgic họ 74 là một tập hợp các mạch tổ hợp số TTL. Các vi mạch này có thể được xác định phân biệt bằng tiền tố 74 trong tên gọi, được kế tiếp bằng 2 hoặc 3 con số, xác định chức năng của vi mạch.

Chương trình 9.13 hiện thị 8 vi mạch khác nhau của họ 74. Chú ý rằng số hiệu các vi mạch được lưu trữ trong khuôn mẫu `long integer` như giá trị cực đại của số nguyên có dấu 2 byte là 32.767 và một số nguyên không dấu là 65.535. Một chữ cái L được nối vào một giá trị hằng nếu nó là một số `long integer`.

Lệnh `printf()` sử dụng xâu khuôn mẫu "%-61d %-20s\n"; dấu trừ có nghĩa là các đối số được căn lề bên trái (justification), trong khi sự căn lề mặc định là bên phải.

Chương trình 9.13

```
/* prog9_13.c                                          */
#include <stdio.h>

#define  NO-DEVICES      8
#define  MAXSTRING      100

typedef struct
{
  long  int product_no;
  char       name[MAXSTRING];
} Device;
void      show_devices(Device dev[]);
int       main(void)
{
Device    device[NO-DEVICES]={
     (7408L,"2-input AND gate"),(7432L,"2-input OR gate"),
     (7400L,"2-input NAND gate"),(7402L'"2-input NOR gate"),
     (7404L,"Inverting buffer"),(7474L, "D-type flip-flop"),
     {7470L, "J-K flip-flop" @,(74121L,"Monostable"));
  show_devices(device);
  return=(0);
}
void    show_devices(Device dev[])
{
int   i;
```

```
    for (i=0;i<NO_DEVICES;i++)
      printf("%-6ld %-20s\n",dev[i].product_no,dev[i].name);
}
```

Kết quả chạy thử 9.11 giới thiệu một thí dụ làm mẫu.

Chạy thử 9.11

```
7408    2-input AND gate
7432    2-input OR gate
7400    2-input NAND gate
7402    2-input NOR gate
7404    Inverting buffer
7474    D-type flip flop
7470    J-K flip flop
74121   Monostable
```

9.6 THỰC HÀNH

Q9.1 Hãy viết một chương trình chứa một cơ sở dữ liệu cố định, bao gồm danh sách các vi mạch CMOS thuộc họ 4000. Đặc tính của một vài vi mạch cơ bản được giới thiệu trong bảng 9.4. So sánh với chương trình 9.13.

Bảng 9.4: **Một số vi mạch thuộc họ 4000.**

Vi mạch	Chức năng
4081	Cổng AND hai lối vào
4011	Cổng NAND hai lối vào
4071	Cổng OR hai lối vào
4001	Cổng NOR hai lối vào
4070	Cổng EX-OR hai lối vào
4013	Flipflop loại D
4027	Flipflop loại JK
4043	Flipflop loại RS

Q9.2 Hãy sửa đổi chương trình trong Q9.1 sao cho người dùng có thể nhập vào một số hiệu của vi mạch và chương trình sẽ hiển thị chức năng của nó. Kết quả chạy thử 9.12 chỉ ra một thí dụ làm mẫu.

Chạy thử 9.12

```
Enter a 4000 series device >> 4013
    D-type flip-flop
Enter a 4000 series device >> 4043
>>>> RS-flip-flop
```

Q9.3 Hãy viết một chương trình cơ sở dữ liệu, dựa trên cơ sở chương trình 9.5, cho phép đưa ra một thực đơn lựa chọn như người dùng muốn nhập vào một linh kiện mới, để liệt kê tất cả các linh kiện đã có trong cơ sở dữ liệu, hoặc để thoát ra khỏi chương trình. Một kết quả dùng làm thí dụ được giới thiệu trong chạy thử 9.13.

Chạy thử 9.13

```
Do you wish to
(1) Input a component
(2) List all components
(3) Exit from program
Enter option >>>
```

Q9.4 Hãy thay đổi chương trình 9.8 sao cho có thể đưa vào cả các năm nhuận. Người dùng sẽ nhập vào tên tháng, năm hiện tại và ngày trong tháng. Chú ý rằng năm nhuận có thể chia hết cho 4 (toán tử mô đun % có thể có ích bởi vì year %4 dẫn đến giá trị bằng 0. Một kết quả dùng làm thí dụ được giới thiệu trong chạy thử 9.14.

Chạy thử 9.14

```
Enter month              >> 2
Enter year (e.g. 1994) >> 1993
Enter day of month (1-31) >> 29
invalid input <29>
Enter day of month (1-31) >> 28
Date is 28-FEB-1993
```

Q9.5 Hãy sửa đổi chương trình trong Q9.4 sao cho chương trình hiển thị được phạm vi có nghĩa của các ngày khi ngày trong tháng được nhắc nhập vào. Chẳng hạn, nếu tháng hai được nhập vào thì thông báo nhắc sẽ hiển thị "Nhập vào ngày trong tháng (1-28)": `Enter day of month (1-28)`. Một kết quả chạy thử mẫu được giới thiệu trong chạy thử 9.15.

Chạy thử 9.15

```
Enter month                 >> 4
Enter year (e.g. 1994) >> 1993
Enter day of month (1-30) >> 31
Invalid input <31>
Enter day of month (1-30) >> 28
Date is 28-APR-1993
```

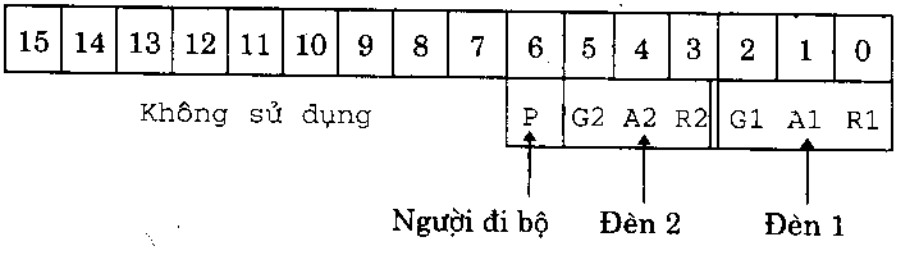
Q9.6 Hãy sửa đổi chương trình 9.9 sao cho ngày tháng được in trong khuôn mẫu DD-MMM-YYYY, trong đó DD là ngày trong tháng, MMM là viết tắt 3 ký tự đầu của tên tháng và YYYY là 4 con số mô tả tên năm. Chạy thử 9.16 chỉ ra một thí dụ mẫu. Gợi ý: sử dụng mảng của các mảng đã được sử dụng trong chương trình 9.8 để diễn tả tên tháng.

Chạy thử 9.12

```
Current data is 05-OCT-1998
```

Q9.7 Hãy sửa đổi chương trình 9.1 sao cho nó hiển thị được sự nối tiếp của hai đèn giao thông với sự điều khiển người đi bộ một mình.

`traffic_bit_fielf`: Trường bit giao thông



Hình 9.7: **Sự sắp xếp trường bit trong cấu trúc traffic_bit_field.**

Bảng 9.5 chỉ ra một sơ đồ làm mẫu và hình 9.7 chỉ ra một mẫu bit có thể dùng cho cấu trúc.

Bảng 9.5: **Đèn giao thông và sơ đồ điều khiển người đi bộ.**

Đèn giao thông 1	Đèn giao thông 2	Người đi bộ
Đỏ	Đỏ	Không được đi
Đỏ	Đỏ và vàng	Không được đi
Đỏ	Xanh	Không được đi
Đỏ	Vàng	Không được đi
Đỏ	Đỏ	Đi bộ qua
Đỏ và vàng	Đỏ	Không được đi
Xanh	Đỏ	Không được đi
Vàng	Đỏ	Không được đi
Đỏ	Đỏ	Đi bộ qua
Đỏ	Đỏ và vàng	Không được đi
	vân vân...	

Q9.8 Hãy viết một chương trình xác định trở kháng (độ lớn và góc pha) của một mạch RL mắc song song. Mỗi giá trị trở kháng được nạp vào qua các giá trị độ lớn và góc pha, và trở kháng của mạch song song sẽ được in ra dưới dạng phức, nghĩa là các giá trị độ lớn và góc pha. So sánh với chương trình 6.4 và 9.12.

$$Z = \frac{R.j\varpi.L}{R + j\varpi.L}$$

$$= \frac{\varpi.RL\langle 90^o \rangle}{\sqrt{R^2 + (\varpi.L)^2}\left\langle \tan^{-1}\frac{\varpi.L}{R}\right\rangle}$$

$$|Z| = \frac{\varpi.RL}{\sqrt{R^2 + (\varpi.L)^2}}$$

$$\langle Z \rangle = 90^o - \tan^{-1}\frac{\varpi.L}{R}$$

Q9.9 Hãy viết một chương trình bằng cách sử dụng các cấu trúc với các giá trị trở kháng phức để xác định trở kháng tương đương (Z_{td}) của hai trở kháng Z_1 và Z_2 mắc song song. Công thức để tính trở kháng của mạch mắc song song được cho như sau:

$$Z_{td} = \frac{Z_1 . Z_2}{Z_1 + Z_2} \quad [\Omega]$$

Chương 10
NHẬP/ XUẤT TỆP

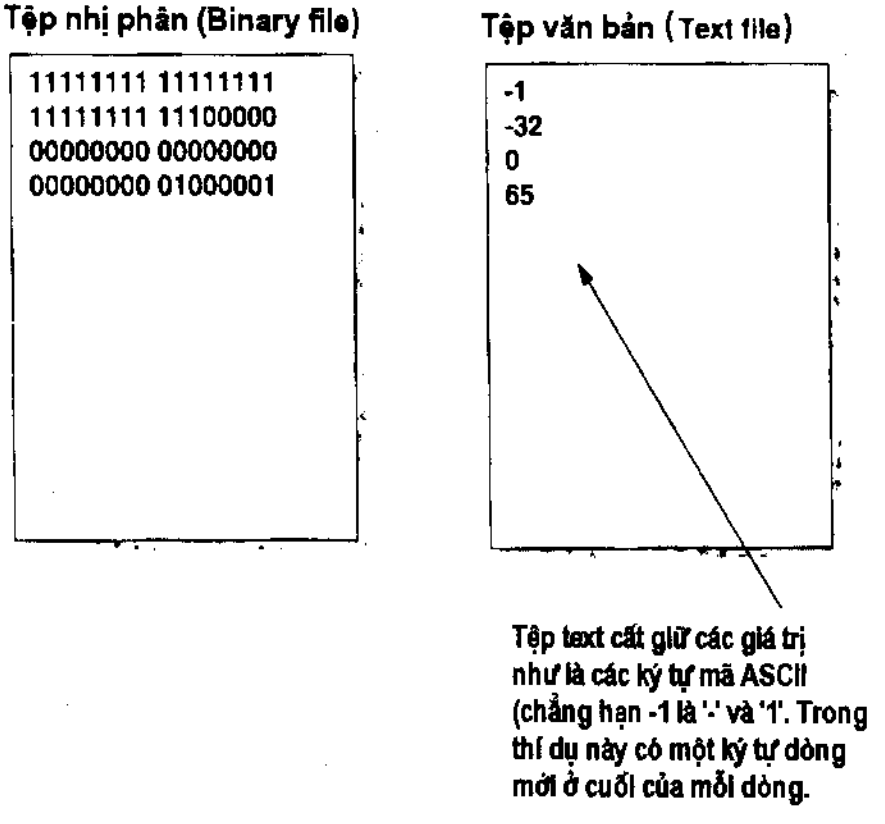
Các tệp lưu trữ dữ liệu dưới dạng có thể sử dụng lâu dài. Bình thường các tệp được đặc trưng bằng một *tên gọi*, có kèm theo một *phần mở rộng* tên tệp (còn gọi là *đuôi*) để xác định loại tệp. Có thể kể ra một vài thí dụ về tệp:

- Tệp mã nguồn ngôn ngữ C, Pascal, FORTRAN, BASIC, COBOL (thí dụ tệp: file.C; file.pas; file.ftn; file.bas; file.cob);
- Các tệp tài liệu cuả bộ soạn thảo văn bản (thí dụ temp1.doc);
- Các tệp mã đối tượng (thí dụ file1.obj; f ile2.cob);
- Các tệp chấp hành (thí dụ file1. exe; file2);
- Các tệp dữ liệu vào (thí dụ in.dat);
- Các tệp đầu ra (thí dụ out.dat);
- Các tệp trợ giúp (thí dụ prog.hlp);

v. v...

Có hai kiểu tệp có thể thao tác được, đó là *tệp nhị phân* và *tệp text* (hay còn gọi là *tệp văn bản*). Một tệp text sử dụng các ký tự ASCII khi đọc ra và viết vào một tệp; một tệp nhị phân sử dụng các digit nhị phân mà máy tính sử dụng để cất giữ các giá trị. Bình thường thì không thể quan sát một tệp nhị phân nếu không có một chương trình đặc biệt dùng làm công cụ, nhưng một tệp text có thể được hiển thị bằng một chương trình soạn thảo văn bản. Để đọc và viết các tệp text người ta sử dụng các hàm `fscanf()` và `fprintf()`, trong khi với các tệp nhị phân thì sử dụng các hàm `fread()` và `fwrite()`.

Hình 10.1 chỉ ra một thí dụ về hai tệp chứa bốn giá trị nguyên. Tệp nhị phân cất giữ các số nguyên bằng cách sử dụng hai byte trong cách viết có dấu, lấy bù theo 2, trong khi với các tệp text thì các ký tự mã ASCII được sử dụng để mô tả các giá trị. Thí dụ, giá trị -1 được mô tả bằng 11111111 11111111 trong cách lấy bù theo 2. Mẫu nhị phân này được cất giữ vào tệp nhị phân. Tệp text sử dụng các ký tự ASCII để biểu diễn -1 (sẽ là '-' và '1') và mẫu bit cất giữ đối với tệp text như vậy sẽ là 0010 1101 (mã ASCII '~') và 0011 0001 (mã ASCII '!'). Nếu cần có một dòng mới sau mỗi số thì một ký tự dòng mới được chèn vào sau nó. Chú ý là không có ký tự dòng mới trong mã ASCII nên muốn diễn tả một dòng mới ta phải dùng hai ký tự: quay về đầu đầu dòng (CR) và nạp dòng (LF). Trong ngôn ngữ C ký tự dòng mới được biểu thị bằng '\n'.



Hình 10.1: **Tệp nhị phân và tệp text.**

Số byte được sử dụng để cất giữ từng phần tử sẽ phụ thuộc vào kiểu dữ liệu của biến. Chẳng hạn, kiểu `long int` sẽ được cất giữ trên bốn byte, giá trị dấu phảy động cũng *có thể* được cất giữ trên bốn byte (trên một vài hệ thống). Khuôn mẫu dấu phẩy động khác với khuôn mẫu số nguyên; khuôn mẫu dấu phẩy động chuẩn sử dụng một bit dấu, một bit giá trị và một bit số mũ. Việc kết thúc tệp được thể hiện bằng ký tự EOF (tiếng Anh: *end of file*).

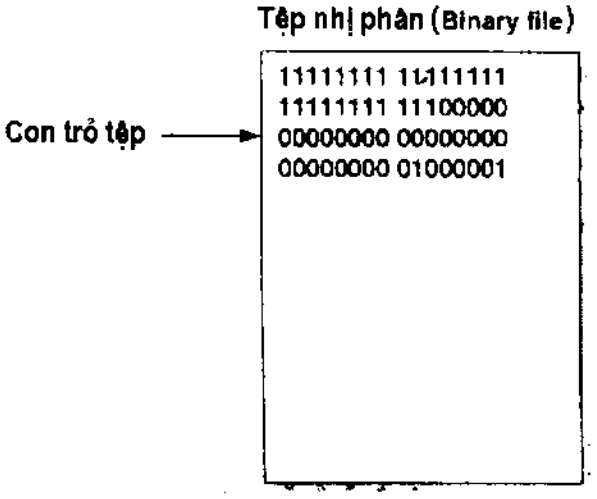
Có 11 hàm chính được sử dụng để xuất/ nhập tệp và được liệt kê ở dưới đây. Các hàm `fprint()` và `fscanf()` tương tự như các hàm `printf()` và `scanf()` nhưng đầu ra của chúng dẫn tới một tệp.

```
file_ptr=fopen(file name,"atrbutes");
fclose (file_ptr);
fprintf(file_ptr,"format",arg1,arg2,..);
fscanf(file_ptr,"format",arg1,arg2...);
fgets(str,n,file_ptr);
fputs(str,file_ptr);
fputc(ch,file_ptr);
ch=fgetc(file_ptr);
fwrite(ptr,size,n,file_ptr);
fread(ptr,size,n,file_ptr);
feof(file_ptr);
```

Một con trỏ tệp cất giữ vị trí hiện thời trong một tệp khi đọc hoặc viết tệp. Tất cả các tác động bên trong tệp được thực hiện bằng cách so sánh với con trỏ. Kiểu dữ liệu của con trỏ này được định nghĩa trong *stdio.h* và có tên là FILE.

```
#include <stdio.h>

int     main(void)
{
FILE     *fileptr;
}
```

Con trỏ tệp di chuyển khi mỗi phần tử được đọc/ viết. Hình 10.2 chỉ ra một con trỏ tệp đang chỉ vào vị trí hiện thời bên trong tệp.



Hình 10.2: **Con trỏ tệp.**

10.1 MỞ MỘT TỆP (fopen())

Con trỏ tệp được gán bằng cách sử dụng hàm `fopen()`. Tham số đầu tiên là tên tệp và thứ hai là một xâu, xác định các thuộc tính của tệp; các thuộc tính này được liệt kê trong bảng 10.1.

Bảng 10.1: **Các thuộc tính của tệp.**

Thuộc tính	Chức năng
`"r"`	mở chỉ để đọc
`"w"`	tạo ra tệp để viết
`"a"`	gắn vào; mở để viết vào chỗ cuối của tệp hoặc tạo ra để viết nếu như tệp chưa có (không tồn tại).
`"r+"`	mở một tệp đang tồn tại để cập nhật (đọc và viết)
`"w+"`	tạo ra một tệp mới để cập nhật
`"a+"`	mở để nối vào; mở (hoặc tạo ra nếu như tệp không tồn tại) để cập nhật vào cuối tệp

Chế độ mặc định để mở các tệp là *text*, nhưng thuộc tính `t` có thể được nối vào xâu thuộc tính để chỉ rõ một tệp text (văn bản). Thí dụ, thuộc tính "wt" mở một tệp văn bản để viết. Một tệp nhị phân được chỉ rõ bằng cách nối chữ `b` vào chuỗi thuộc tính. Chẳng hạn, "rb" sẽ mở một tệp nhị phân để đọc.

Khuôn mẫu của hàm `fopen()` là:

```
file-ptr=fopen("Filename","attrib");
```

Nếu như hàm `fopen()` được thực hiện hoàn chỉnh thì một con trỏ tệp sẽ được trả lại, và con trỏ này được khởi tạo ở chỗ bắt đầu của tệp. Nếu không có khả năng mở tệp thì một giá trị NULL (không) sẽ được trả lại. Có thể có nhiều lý do làm cho một tệp không thể mở được, cụ thể là:

- Không có tệp với tên như ta chỉ định;
- Tệp được bảo vệ chống đọc ra và/ hoặc viết vào;
- Tệp là một thư mục.

Điều quan trọng là một chương trình không đọc từ một tệp mà không thể mở được bởi vì nó có thể làm cho chương trình diễn biến theo hướng không dự đoán trước được. Một phép thử đối với điều kiện này được dẫn ra dưới đây:

```
int     main (void)
{
FILE     *in;
      if ((in=fopen("in.dat","r"))==NULL)
      {
      printf("File IN.DAT could not be opened");
      }
      ::::::::::::::::::::::::
      return(0);
}
```

10.2 ĐÓNG MỘT TỆP (fclose())

Mỗi lần một tệp được sử dụng thì nó phải được đóng trước khi chương trình kết thúc. Một tệp không được đóng đúng cách có thể gây ra các vấn đề trong hệ thống tệp. Khuôn mẫu chuẩn được dẫn ra dưới đây; giá trị trả lại (rtn) đưa trả lại 0 khi mọi việc tốt đẹp, một khả năng khác là trả lại EOF nếu như có bất kỳ một lỗi nào đấy xuất hiện. Macro EOF được định nghĩa trong *stdio.h*.

```
rtn=fclose(file_ptr)
```

10.3 XUẤT TEXT VÀO MỘT TỆP (fprintf())

Hàm `fprintf()` được sử dụng với các tệp text và có khuôn mẫu tương tự với `printf()` nhưng đầu ra được dẫn tới một tệp. Một thí dụ của hàm `fprintf()` được dẫn ra dưới đây. Giá trị trả lại (`rtn`) đưa trở lại số byte gửi cho tệp; khi mắc lỗi nó trả lại EOF.

```
rtn=fprintf(file-ptr,"%s %d",strl,valuel);
```

10.4 ĐỌC TEXT TỪ MỘT TỆP (fscanf())

Hàm `fscanf()` được sử dụng với các tệp text và có một khuôn mẫu tương tự với `scanf()` nhưng đầu vào được đọc từ một tệp. Một thí dụ về hàm `fscanf()` được dẫn ra dưới đây.

```
rtn=fscanf(file_ptr,"%s %d %d",strl,&vall,&val2);
```

Hàm này trả lại số các trường đã quét hoàn chỉnh (`rtn`), nếu như có việc thử đọc EOF (từ kết thúc tệp) thì giá trị EOF sẽ được trả lại.

10.5 TÌM CHỖ KẾT THÚC CỦA MỘT TỆP (feof())

Hàm `feof()` phát hiện ký tự kết thúc tệp. Nó trả lại giá trị khác không (nghĩa là một TRUE) nếu như con trỏ tệp ở vào chỗ kết thúc của một tệp, nếu không thì một số 0 sẽ được trả lại. Hàm được dẫn ra dưới đây sử dụng hàm `feof()` để tìm ra chỗ kết thúc của tệp và đồng thời kiểm tra giá trị trả lại từ `fscanf()` sao cho việc đọc không trọn vẹn từ tệp được bỏ qua.

```
int   get_values(int maxval, char fname[],int *n,float arr[])

FILE  *in;
int   i=0,rtn;
     if ((in=fopen(fname,"r"))==NULL)
     {
        printf("cannot open %s\n",fname);
        return(NOFILE);
     }
     while (1(feof(in)))
     {
           rtn=fscanf(in,"%f",&arr[i]);
           if (rtnl=EOF) i++;
           if (i==maxval) break;
     }
     fclose(in);

     *n=i,
     return(!NOFILE);
}
```

10.6 NHẬN XÂU TEXT TỪ MỘT TỆP (fgets())

Hàm `fgets()` tương tự với `get()` và được sử dụng để nhận một xâu text từ một tệp tới một ký tự dòng mới. Khuôn mẫu chuẩn được chỉ ra dưới đây.

```
rtn=fgets(str,n,file_ptr)
```

Hàm này đọc từ tệp được chỉ định rõ bởi `file_ptr` vào `str` với nhiều nhất là n ký tự. Hàm trả lại (`rtn`) chỉ vào xâu được chỉ vào bằng `str` hoặc sẽ trả lại một NULL (không) nếu như gặp các ký tự kết thúc tệp EOF (hoặc một lỗi).

10.7 ĐẶT XÂU TEXT VÀO MỘT TỆP (fputs ())

Hàm `fputs()` tương tự với `puts()` và được sử dụng để viết một xâu text vào một tệp. Hàm này không nối đầu ra với một dòng mới. Khuôn mẫu chuẩn được chỉ ra dưới đây.

```
rtn=fputs(str,file_ptr)
```

Hàm này xuất `str` vào tệp được chỉ định rõ bởi `file_ptr`. Hàm trả lại (`rtn`) đưa trở lại ký tự cuối cùng đã được viết; nếu như có lỗi thì hàm sẽ trả lại các ký tự kết thúc tệp EOF.

10.8 ĐẶT KÝ TỰ VÀO MỘT TỆP (fputc())

Hàm `fpute()` tương tự với `putchar()` và được sử dụng để viết một ký tự đơn vào một tệp. Khuôn dạng chuẩn được chỉ ra dưới đây.

```
rtn=fputc(ch,file_ptr)
```

Hàm này viết `ch` vào tệp được chỉ định rõ bởi `file_ptr`. Hàm trả lại (`rtn`) đưa trở lại ký tự cuối cùng đã được viết; nếu như có lỗi thì hàm sẽ trả lại các ký tự kết thúc tệp EOF.

10.9 NHẬN KÝ TỰ TỪ MỘT TỆP (fgetc())

Hàm `fgetc()` tương tự với `getchar()` và được sử dụng để đọc một ký tự đơn từ một tệp. Khuôn dạng chuẩn được chỉ ra dưới đây.

```
ch=fgetc(file_ptr)
```

Hàm này đọc `ch` từ tệp được chỉ định rõ bởi `file_ptr`. Nếu như có lỗi trong quá trình nhận thì các ký tự EOF sẽ được trả lại.

10.10 TỆP NHỊ PHÂN

10.10.1 ĐỌC DỮ LIỆU NHỊ PHÂN TỪ MỘT TỆP (fread())

Hàm `fread()` được sử dụng để đọc dữ liệu trong khuôn mẫu nhị phân. Khuôn mẫu chuẩn là:

```
rtn=fread(ptr.size,n,file_ptr)
```

Hàm này đọc n mục dữ liệu vào trong khối dữ liệu được chỉ định rõ bởi ptr, mỗi mục có chiều dài SIZE byte, từ tệp đầu vào được chỉ định rõ bởi file_ptr. Giá trị trả lại rtn chỉ rõ số các khối (hoặc các mục) đã được đọc hoàn chỉnh. Nếu như không có lỗi xuất hiện trong khi đọc thì số các mục đã được chỉ rõ (n) sẽ chính là giá trị trả lại (rtn). Nếu không thì có nghĩa là một lỗi hoặc ký tự kết thúc tệp đã xuất hiện.

Hàm fread() được sử dụng trong hàm chỉ ra dưới đây. Trong trường hợp, một giá trị dấu phẩy động được đọc từ tệp và giá trị trả lại được kiểm tra để xem liệu nó đã được đọc hợp lệ. Hàm sizeof() được sử dụng để xác định số byte đã được sử dụng để cất giữ một dấu phẩy động (float).

```
int   read_data(char fname[],float arr[],int *nov)
{
FILE    *in;
int     i;

   *nov=0; /* number of values in the array */
   if ((in=fopen(fname,"rb"))==NULL)
   {
   printf("Cannot open %s\n",fname);
   return(NOFILE);
   }
   while (!feof(in))
   if (fread(&&=[i],sizeof (float),1,in)==1) /*read 1 value*/
   (*nov)++;
   }
   fclose(in);
   return(!NOFILE);
}
```

10.10.2 VIẾT DỮ LIỆU NHỊ PHÂN VÀO MỘT TỆP (fwrite())

Hàm fwrite() được sử dụng để viết dữ liệu trong khuôn nhị phân. Khuôn mẫu chuẩn là:

```
rtn=fwrite(ptr,size,n,file_ptr)
```

Hàm này viết n mục dữ liệu từ khối dữ liệu được chỉ định rõ bằng ptr, mỗi khối có chiều dài SIZE byte, vào tệp đầu ra được chỉ định rõ bằng file_ptr. Giá trị được trả lại rtn chỉ rõ rằng số các khối đã được hoàn chỉnh. Hàm fwrite() được sử dụng trong hàm chỉ ra dưới đây.

```
int   dump_data(char fname[],float arr[],int nov)
{
FILE    *out;
int     i;

    if ((out=fopen(fname,"wb"))==NULL)
    {
        printf("cannot open %s\n",fname);
        return(NOFILE);
    }
    for (i=0;i,nov;i++)
        fwrite (&arr[i],sizeof (float),1,out);
    fclose(out);
    return(!NOFILE);
}
```

10.11 MỘT SỐ THÍ DỤ

10.11.1 PHƯƠNG TRÌNH ĐẠI SỐ BOOLE

Chương trình 10.1 sử dụng một tệp text để cất giữ các kết quả của quá trình mô phỏng mạch Boole. Phương trình Boole được sử dụng là: Z=(A.B) +C

Chương trình 10.1

```
/* prog10_1.c                                          */
/* Program to determine truth table for boolean        */
/* function Z=(A and B) or C                           */
/* output truth table is stored to a file              */
/* and then read back from this file                   */
```

```
#include <stdio.h>

#define   MAX-STATES     8
#define   STRINGSIZE    100

void    process_eq(int tt[]);
void    dump_table(char fname[],int tt[]);
void    read_table(char fname[]);

int main(void)
int t_table(MAX_STATES];
     process_eq(t_table);
     dump_table("OUT.DAT",t_table);
     read_table("OUT.DAT") ;
     return(0);
}
/* determine truth table */
 void process_eq(int tt[])
{
int     state=0,A,B,C;
     puts ("Processing states") ;
     for (A=0;A<=1;A++)
        for (B=0;B<=1;B++)
        for (C=0;C<=1;C++)
        {
              tt[state]=(A & B) | C;
              state++;
            }
}
/* output truth table to a file */
void    dump_table(char fname[],int tt[])
int     state=0,A,B,C;
FILE    *out;
    if ((out=fopen(fname,"w"))==NULL)
```

```
    {
        printf("Cannot open %s \n",fname);
        return;
    }
     puts("Writing to file");
     fprintf(out,"A 3 C Z\n");
     for (A=0;A<=1;A++)
            for (B=0;B<=1;B++)
            for (C=0;C<=1;C++)
            {
               fprintf(out,"%d %d %d %d\n",A,B,C,tt[state]);
               state++;
                 }
   fclose(out);
   puts("File written");
}
/* read truth table from a file */
 void  read_table(char fname[])
{
int     A,B,C,Z;
FILE    *in;
char    header[STRINGSIZE];
   if ((in=fopenifname,"r"))==NULL)
   {
        printf("Cannot open %s\n",fname);
        return;
}
puts("Reading from file");

/* get file header is "A B C Z" */
fgets(header,STRINGSIZE,in); printf("&s",header);
while (!feof(in))
{
      fscanf(in,"%d %d %d %d\n",&A,&B,&C,&Z);
      prinrf("%d %d %d %d\n",A,B,C,Z);
```

```
}
fclose(in);
}
```

10.11.2 CHƯƠNG TRÌNH LẤY TRUNG BÌNH

Chương trình 10.2 sử dụng các tệp text để xác định giá trị trung bình của số các giá trị dấu phẩy động được chứa trong một tệp. Hàm `get_values()` được sử dụng để đọc các giá trị từ một tệp, mà trong trường hợp này là IN.DAT. Tệp này có thể được tạo ra bằng cách sử dụng một chương trình soạn thảo văn bản.

Chương trình 10.2

```
/* prog10_2.c                                                    */
/* Program to  determine the average of a file    */
/* Containing a number of floating point values   */

#include <stdio.h>

#define  NOVALUES    100 /* max. number of entered values */
#define  NOFILE    0
int     get_values (int maxvals, char fname [],int *n,float arr[])
float   calc_average(int nval,float arr[1];
void    display_average (int nval, float arr [], float aver);
int     main(void)
{
float    values[NOVALUES],average;
int      nvalues;

if    (get_values(NOVALUES,"IN.DAT",&nvalues,values)==NOFILE)
return(0);
average=calc_average(nvalues,values);
display_average(nvalues,values,average);
return(0);
}
int   get_values (int maxvals, char fname[],int *n,float arr[])
{
```

```
FILE      *in;
int       i=0,rtn;
if ((in=fopen(fname,"r"))==,NULL)
{
printf("Cannot open %s\n",fname);
return(NOFILE);
}
while (!feof(in))
{
rtn=fscanf(in,"%f",&arr[i]);
if (rtn!=EOF) i++;
if (i==maxvals) break;
}
fclose(in);
*n=i;
return(!NOFILE);
}
float calc_average(int nval,float arr[])
{
int      i;
float    running_total=0;
for (i=0;i<nval;i++)
running_total+=arr[i];
/* note there is no test for a divide by zero */
return(running_total/nval);
}
void    display_average(int nval,float arr[],float aver)
{
int     i;
puts("INPUT VALUES ARE:");
for (i=0;i<nval;i++)
printf("%8.3f\n",arr[i]);
printf("Average is %8.3f\n",aver);
}
```

Một thí dụ về nội dung của tệp IN.DAT được chỉ ra dưới đây.

```
 3.240
 1.232
 6.543
-1.432
```

Một kết quả chạy thử khi sử dụng tệp này được chỉ ra trong chạy thử 10.2.

Chạy thử 10.2

```
INPUT VALUES ARE:
 3.240
 1.232
 6.543
-1.432
Average is      2.396
```

10.11.3 CHƯƠNG TRÌNH ĐỌC/ VIẾT NHỊ PHÂN

Chương trình 10.3 là một thí dụ cho thấy một xâu các giá trị dấu phẩy động được viết vào một tệp nhị phân như thế nào khi sử dụng hàm `fwrite()` và sau đó đọc ngược trở lại bằng cách sử dụng `fread()`. Chú ý là các cờ `NOFILE` được trả lại từ `dump_data()` và `reead_data()` được bỏ qua bằng `main()`.

Chương trình 10.3

```
/* prog10_3.c                                    */
/* FILEB.C                                       */
/* Writes and reads an array of floats           */
/* to and from a binary file                     */

#include   <stdio.h>
#define  NOFILE   0   /* error flag is file does not exist */
#define  MAXSTRING  100 /* max.number of char's in filename  */
```

```
#define  MAXVALUES  100 /1 max. number of floats in array     */

void      get_filename (char frame {]);
void      get_values(int maxvals, float vals[],int *nov);
int       dump_data(char fname[],float arr[],int nov);
int       read_data(char fname[],float arr[],int nov);
void      print_values(float arr[],int nov);

int       main (void)
{
char      fname[MAXSTRING];
float     values[MAXVALUES];
int       no_values; /*number of values in the array */
   get_filename(fname);
   get_values(MAXVALUES,values,&no-values);
   dump_data(fname,values,no_values);
   read_data(fname, values, &no_values);
   print_values(values,no_values) ;
   return(0)
}
void     get_filename(char fname[])
{
   printf("Enter file name >>");
   scanf("%s",fname);
}
void      get_values(int maxvals, float vals[],int *nov)
{
int       i;
   do
   {
        printf("Number of values to be entered >>")
        scanf("%d",nov);
        if (*nov>maxvals)
            printf("Too many values: MAX: %d\n",MAXVALUES);
   } while (*nov>MAXVALUES);
```

```
    for (i=0;i<*nov;i++)
    {
        printf("Enter value %d >>"i);
        scanf("%f"&vals[i];
    }
}
int dump_data(char fname[],float arrt[],int nov)
{
FILE     *out;
int      i;

        /* open for binary write */
    if ((out=fopen(fname,"wb"))==NULL)
    {
        printf("Cannot open %s\n",fname);
        return(NOFILE); /* unsuccessful file open */
    }
    for (i=0;i<nov;i++)
         fwrite(&arr [i],sizeof (float),1,out);

    fclose(out);
    return(!NOFILE);
}
int    read_data(char fname[],float arr[],int *nov)
{
FILE    *in;
    *nov=0;  /* number of values in the array */

        /* open for binary read */
    if ((in=fopen(fname,"rb"))==NULL)
    {
        Printf("Cannot open %S\n",fname);
        return(NOFILE); /* unsuccessful file open */
    }
    while (!feot(in))
```

```
{
      if (fread(&arr[*nov],sizeof (float,1,in)==1)
         (*nov) ++;
}
      fclose(in);
      return(!NOFILE);
}
void  print_values(float arr[],int nov)
{
int      i;
     printf("values are:\n");
     for (i=0;i<nov;i++)
          printf("%d %8.3f\n",i,arr[i]);
}
```

Một kết quả làm mẫu được giới thiệu trong chạy thử 10.3.

Chạy thử 10.3

```
Enter file name >>number.dat
Number of values to be entered >>5
Enter value 0 >>1.435
Enter value 1 >>0.432
Enter value 2 >>-54.32
Enter value 3 >>-1.543
Enter value 4 >>100.01
Values are:
0     1.435
1     0.432
2   -54.320
3    -1.543
4   100.010
```

10.12 THỰC HÀNH

Q10.1 Hãy viết một chương trình xác định các giá trị trung bình, lớn nhất và nhỏ nhất của một tệp text chứa các giá trị dấu phẩy động. Một tệp mẫu được chỉ ra dưới đây.

```
3.24
13.443
100.3
85.03
43.239
-30.032
```

Q10.2 Hãy viết một chương trình xác định bảng giá trị chân lý dùng cho hàm sau. Đầu ra sẽ được gửi tới một tệp (chẳng hạn OUT.DAT).

$$Z = \overline{(A+B)}.(\overline{C}.D)$$

Q10.3 Hãy sửa đổi các chương trình sau đây để đầu ra được gửi cho một tệp text:

(a) Chương trình 5.14 (xem chạy thử 5.4 đối với khuôn mẫu tệp đầu ra).

(b) Chương trình được chỉ rõ trong Q5.7.

(c) Chương trình chỉ rõ trong Q5.10 (xem chạy thử 5.11 đối với khuôn mẫu tệp đầu ra).

(d) Chương trình 6.4 (xem chạy thử 6.3 đối với khuôn mẫu tệp đầu ra).

Q10.4 Hãy lặp lại Q10.3, nhưng đầu vào được đọc vào từ một tệp (thí dụ IN.DAT) ít hơn là từ bàn phím.

Q10.5 Hãy sửa đổi chương trình 7.1 để chương trình đọc các giá trị từ một tệp dữ liệu đầu vào và xuất ra các kết quả vào một tệp đầu ra.

Q10.6 Viết một chương trình đọc một tệp text chứa các ký tự ASCH biểu diễn các digit nhị phân (thí dụ các ký tự '0' và '1'). Mỗi dòng của

tệp chứa năm digit. Chương trình sẽ đọc tất cả các dãy và hiển thị chúng trong khuôn mẫu nhị phân. Một dãy dùng làm thí dụ minh hoạ được giới thiệu trong chạy thử 10.4.

Chạy thử 10.4

```
Input file name > BINARY.DAT
Scanning file.....
Binary sequences
1  10010
2  00011
3  10111
4  11111
5  10010
Program end.  BYE
```

Nội dung của tệp BINARY.DAT được dẫn ra dưới đây.

```
1 0 0 1 0
0 0 0 1 1
0 0 1 1 1
0 1 1 1 1
0 0 0 1 0
```

Q10.7 Hãy sửa đổi Q10.6 để đọc vào dãy nhị phân của các chiều dài biến. Một thí dụ làm được giới thiệu trong chạy thử 10.5.

Chạy thử 10.5

```
Input file name > BINARY.DAT
Scanning file.....
Binary sequences
1  100101010001
2  00
3  101110101
4  111110001000
5  1001011
Program end.  BYE
```

Nội dung của tệp BINARY.DAT được dẫn ra dưới đây.

```
1 0 0 1 0 1 0 1 0 0 0 1
0 0
1 0 1 1 1 0 1 0 1
1 1 1 1 1 0 0 0 1 0 0 0
1 0 0 1 0 1 1
```

Q10.8 Hãy sửa đổi chương trình trong Q10.7 sao cho chỉ có các ký tự '0' và '1' được hiển thị đối với một tệp chứa cả các ký tự khác.

Chạy thử 10.6

```
Input file name > BINARY.DAT
Scanning file.....
Binary sequences
1  100101010001
2  00
3  101110101
4  111110001000
5  1001011
Program end.  BYE
```

Nội dung của tệp BINARY.DAT được dẫn ra dưới đây.

```
1 X 0 0 1 0 1 C 0 1 0 0 0
0 P 0
1 0 1 1 1 0 1 0 1
1 1 Q 1 1 1 0 0 0 1 0 0 0
1 0 0 1 1 0 1 1
```

Q10.9 Hãy viết một chương trình đếm số các ký tự trong một tệp.

Q10.10 Hãy viết một chương trình có khả năng đếm số lần xuất hiện của chữ cái 'a' trong một tệp. Hãy sử dụng hoặc ch=fgetco hoặc fscanf(in,"%c",&ch) để đọc các ký tự riêng lẻ trong tệp.

Q10.11 Hãy viết một chương trình trong đó người dùng nhập vào ký tự bất kỳ và chương trình sẽ xác định số lần xuất hiện của ký tự đó trong tệp được chỉ rõ. Chẳng hạn:

Chạy thử 10.7

```
Enter filename: frod.dat
Enter character to search for: i
There are 14 occurences of character i in file
fred.dat.
```

Q10.12 Viết một chương trình xác định số các từ trong một tệp.

Q10.13 Viết một chương trình mà sẽ xác định số các dòng trong một tệp. Một phương pháp có thể đếm các ký tự dòng mới được giới thiệu dưới đây:

```
ch=getc(in);
if (ch=='\n') no_lines++;
```

Q10.14 Các chương trình 10.4 và 10.5 đều có chứa một lỗi. Cả hai chương trình sẽ sao chép nội dung của tệp IN.DAT vào trong OUT.DAT. Chương trình 10.4 sử dụng `fgetc()` và `fputc()`, trong khi 10.5 sử dụng `fgets()` và `fputs()`.

Chương trình 10.4

```
/* prog10_4.c                                    */
/* Program to copy from IN.DAT to OUT.DAT        */
#include <stdio.h>

int     main(void)
{
int     ch;
FILE    *in,*out;

    if ((in=fopen("in.dat","r"))==NULL)
    {
        puts("Cannot open IN.DAT");
        return(1);
    }
```

```
    if ((out=fopen("out.dat","w"))==NULL)
    {
        puts("Cannot open OUT.DAT");
        return(1);
    }
    while (!feof(in))
    {
        ch=fgetc(in);
        if(ch==EOF) fputc(ch,out);
        }
    fclose(in);
    fclose(out);
    return(0);
}
```

Chương trình 10.5

```
/* prog10_5.c                                          */
/* Program to COPY from IN.DAT to OUT.DAT              */
#include <stdio.h>

int    main(void)
{
char      str[BUFSIZ];
FILE   *in,*out;
if ((in=fopen("in.dat","r"))==NULL)
{
puts("Cannot open IN.DAT");
return(1);
}
if ((out=fopen("out.dat","w"))==NULL)
{
puts("Cannot open OUT.DAT");
return(1);
}

while (feof (in) )
{
```

```
fgets(str,BUFSIZ,in);
fputs(scr,out);
}
fclose(in);
fclose(out);
return(0);
}
```

Q10.15 Hãy viết một chương trình có thể xoá đi các dòng trắng trong một tệp đầu vào và viết tệp đã qua xử lý vào một tệp ở đầu ra. Các tệp đầu vào và đầu ra dùng làm thí dụ về mối liên hệ giữa tần số (frequency) và trở kháng (Impedance) được chỉ ra dưới đây.

Tệp đầu vào (có một dòng trắng !):

Frequency	Impedance (mag)
100	101.1
150	165.1
200	300.5

Tệp đầu ra:

Frequency	Impedance (mag)
100	101.1
150	165.1
200	300.5

Chương 11
LẬP TRÌNH HỆ THỐNG

Hệ điều hành cho phép người dùng truy nhập phần cứng một cách dễ dàng để sử dụng. Hệ điều hành tiếp nhận các lệnh từ bàn phím và hiển thị lên màn hình. Hai hệ điều hành phổ biến nhất là hệ điều hành DOS và hệ điều hành UNIX. Hệ điều hành đĩa (Disk Operating System) mà thường được gọi tắt là DOS, có nguồn gốc từ mục đích đặt ra lúc ban đầu, khi tiến hành xây dựng hệ điều hành. DOS cung cấp một bộ điều khiển cho máy tính để truy nhập lên các ổ đĩa. Ngôn ngữ của hệ thống điều hành đĩa bao gồm một bộ lệnh, được người dùng trực tiếp nhập vào và được dịch để thực hiện nhiệm vụ quản lý các tệp tin, chương trình chấp hành và cấu hình hệ thống. DOS không phải là một hệ điều hành đa nhiệm (multi-tasking), với DOS ở một thời điểm ta chỉ có thể cho chạy một chương trình, trong khi UNIX lại là hệ điều hành đa nhiệm.

Các chức năng chính của một hệ điều hành là: chạy các chương trình, sao chép hoặc xoá các tệp tin, tạo ra các thư mục, di chuyển các tệp tin bên trong một cấu trúc thư mục hoặc liệt kê danh sách các tập tin. over

11.1 LỜI GỌI HỆ THỐNG

Một chương trình có thể gọi hệ điều hành theo một số cách khác nhau. Một phương pháp có thể kể ra là sử dụng `system()`, hàm này tiếp nhận một xâu lệnh và chuyển giao cho hệ điều hành. Một nhược điểm của `system()`, là việc điều khiển chương trình được thực hiện qua một chương trình trong hệ điều hành. Chương trình điều khiển ở mức

độ ít hơn qua chương trình gọi cho đến khi nó trở lại từ hệ điều hành. Các ví dụ về lời gọi hệ thống khi sử dụng hệ điều hành đĩa và các hệ điều hành UNIX được cho trong bảng 11.1.

Bảng 11.1: **Những lời gọi hệ thống.**

	DOS	UNIX
Liệt kê thư mục	`system("dir")`	`system("ls")`
Thay đổi thư mục	`system("cd temp")`	`system("cd temp")`
Di chuyển tệp	`system("rename file1 file2")`	`system("mv file1 file2")`
Sao chép tệp	`system("copy file1 file2")`	`systein("cp file1 file2")`

Một phương pháp khác để gọi hệ điều hành là sử dụng các đoạn chương trình (routine) trong thư viện chuẩn. Ưu điểm của phương pháp này là thông tin phụ thêm có thể được trả lại từ hệ điều hành và chương trình điều khiển ở mức độ nhiều hơn qua lời gọi hệ điều hành. Nó cũng làm cho chương trình dễ di chuyển (portable) hơn khi muốn sử dụng chương trình trên hầu hết các hệ điều hành. Một vài chức năng tiêu biểu và tính tương thích hệ điều hành của chúng được cho trong bảng 11.2.

Bảng 11.2: **Chức năng hệ thống thư viện chuẩn.**

Hàm	Mô tả	Để mục tệp	DOS	UNIX	ANSI C	Các cờ
chdir(directory)	thay đổi địa chỉ	*dir.h*				EACCES
mkdir (directory)	tạo thư mục	*dir.h*				ENOENT EACCES
chmod (path,,mode)	thay đổi loại tệp	*io.h*				ENOENT EACCES
creat(name,mode)	tạo ra một tệp	*io.h*				ENOENT EMFILE

			EACCES
rename(from,to)	đổi tên tệp	*stdio.h*	ENOENT EACCES
system(command)	issue system command	*stdlib.h*	ENOENT ENOMEM E2BIG ENOEXEC
rmdir(directory)	xoá đi một thư mục	*dir. h*	EACCES ENOENT

Một biến toàn cục `errno` được thiết lập trong *ermo.h* và *stdlib.h* và được thiết lập nếu như mắc lỗi khi gọi hệ điều hành. Các cờ chính được sử dụng là:

`ENEXEC`	- lỗi chấp hành;
`EACCES`	- không cho phép truy nhập;
`ENOEXIST`	- không có tệp này;
`E2BIG`	- quá nhiều tham số cho lời gọi hàm; (to function call;)
`ENOMEM`	- không đủ bộ nhớ.

Chương trình 11.1 sử dụng chức năng `chdir()` để thay đổi thư mục hiện tại và sử dụng lệnh `DIR` của DOS để liệt kê nội dung thư mục. Nếu như một lỗi xuất hiện khi chức năng đang được thực hiện nó sẽ trở lại giá trị -1. Trên một lỗi biến `errno` được kiểm tra để xác định xem liệu có đang tồn tại thư mục.

Chú ý là nếu một kiểu hệ điều hành khác được sử dụng, thì lệnh `DIR` sẽ được thay thế bằng một lệnh tương đương với nó (thí dụ ls đối với trường hợp hệ điều hành UNIX).

Chương trình 11.1

```
/* prog11_1.c                                    */
/* program to change directory                   */
#include <errno.h>                               */
```

```
#include <stdio.h>
#include <string.h>  /* required for strcmp() */
#include <dir.h>     /* required for chdir()  */
#include <stdlib.h>  /* required for system() */
int   main(void)
{
char    directory[BUFSIZ];
  while (1)
  {
  puts("Enter a directory");
  gets(directory);
  if (!strcmp(directory,"exit")) break;
        /* exit program if EXIT is entered */
  if (chdir(directory)==-1)
        if (errno==ENOENT) puts("Pathname not found");
  else
  {
    puts("Contents of this directory are:-");
    system("dir");
  }
 }
 return(0);
}
```

Chương trình 11.2 cho thấy một thư mục có thể được tạo ra như thế nào bên trong một chương trình.

Chương trình 11.2

```
/* prog11_2.c                                  */
/* program to make a directory                 */
#include <stdio.h>
#include <errno.h>
#include <dir.h>
int    main(void)
{
char    directory[BUFSIZ];
```

```
    puts("Enter a directory");
    gets(directory);
        /* a -1 returned from mkdir()  */
        /* indicates an error          */
    if (mkdir(directory)==-1)
    {
         if(errno==EACCES) puts ("Permisson not allowed");
         if(errno==ENOENT)puts("Path does not exist");
    }
    else
         puts("Directory made");
    return(0);
}
```

Chương trình tiếp theo dây sao chép một tệp sang một tệp khác. Một lần nữa chương trình lại sử dụng cách gọi hệ thống DOS.

Chương trình 11.3

```
/* prog11_3.c                                          */
/* Program to copy one file to another                 */
/* Program uses the systems() function                 */
#include <stdio.h>
#include <string.h>
#include <srdlib.h>
#include <errno.h>
int     main (void)
{
char    str[2*BUFSIZ],file1[BUFSIZ],file2[BUFSIZ];

  puts("Enter file which is to be copied>>");
  gets(file1);
  puts("Enter file which is to be copied to>>");
  gets(file2);
  strcpy(str,"copy");
  scrcat(str,file1);
  strcat(str," ");
```

```
    strcat(str,file2);

    if (system(str)==-1)
    {
      if (errno==ENOENT) puts("File does not exist");
      if (errno==ENOMEM) puts("Not enough memory");
      if (errno==ENOEXEC) puts("Not an executable file");
    }
    return(0);
}
```

Khối mã:

```
strcpy(str,"copy ");
stcat(str,file1);
stcat(str," ");
stcat(str,file2);
```

sẽ được thay thế một cách đơn giản bằng:

```
sprintf(str,"copy %s %s",file1,file2);
```

11.2 CHUYỂN GIAO THAM SỐ

Các đối số có thể được chuyển giao vào trong một chương trình C bằng cách sử dụng các tham số được chỉ định rõ trong phần đầu mục hàm `main()`. Khuôn mẫu chuẩn được sử dụng như sau:

```
main(int argc, char largv[])
```

`argc` chứa số đối số dòng lệnh.

`argv` là một xâu các chuỗi có chứa tất cả lệnh các đối số dòng lệnh.

Đối số đầu tiên chứa tên lệnh (`argv[0]`). Chương trình 11.4 liệt kê ra các đối số dòng lệnh.

Chương trình 11.4

```
/* prog11_4.c                                    */
#include <stdio.h>
```

```
int main(int argc,char *argv[])
/* argc contains number of command line          */
/* arguments. argv[] is a pointer to each argument */
/* argv[0] points to command name, argv[i] points  */
/* to first string, etc.                          */
{
int i;
   for (i=0;i<argc;i++)
        printf("Arg %d is %s\n",i,argv[i]);
   return(0);
}
```

Một kết quả chạy chương trình này được dẫn ra trong chạy thử 11.1. Dấu nhắc dòng lệnh được chỉ ra như một ký hiệu lớn hơn (>). Tên chương trình được lưu trữ trong `argv[0]` và phần còn lại của đối số đã cất giữ trong `argv[1]` đến `argv[argc-1]`. Tất cả các đối số dòng lệnh được xử lý như là các xâu. Nếu chương trình thực hiện những phép toán dưới dạng số thì chúng phải được biến đổi bằng cách sử dụng một hàm biến đổi xâu (như `sscanf()`).

Chạy thử 11.1

```
> argl list 100 4.31 -f
Arg 0 is C:\TC\ARG1.EXE
Arg 1 is list
Arg 2 is 100
Arg 3 is 4.31
Arg 4 is -f
```

11.3 MỘT SỐ VÍ DỤ

Chương trình 11.5 xác định trở kháng của một tụ điện. Các tham số được chuyển giao vào trong chương trình qua các tham số.

Chương trình 11.5

```
/* progll_5.c                                    */
```

```
/* parameters passed via arguments                    */
/* Format is calcimp frequency capacitance            */

#include <stdio.h>
#include <math.h>
#define PI   3.14159
float calc_Xc(float f , float c)
int   main(int argc,char *argv[])
{
float f,C,Xc;

   if (argc!=3)
   {
     puts("Program to determine impedance of a capacitor");
     puts("Format for data is: ");
     puts("calcimp FREQUENCY CAPACITANCE");
     exit(0);
   }
   sscanf(argv [1],"%f",&f) ;
   sscanf(argv [2],"%f",&C) ;

   Xc=calc_Xc(f,C);
   printf ("Impedance is %f ohms\n",XC) ;
}
float calc_xc(float f, float c)
{
   return(1/(2*PI*f*c));
}
```

Một kết quả chạy thử được dẫn ra trong chạy thử 11.2 dưới đây.

Chạy thử 11.2

```
C:\SRC\ARGS> calcimp
Program to determine impedance of a capacitor
Format for data is:
```

```
calcimp FREQUENCY CAPACITANCE

c:\SRC\ARGS> calcimp 1e6 1e-6
Impedance is 0.159155 ohms
```

Chương trình 11.6 xác định biên độ của trở kháng đường truyền. Tham số được chuyển giao vào chương trình thông qua các tham số dòng lệnh. Các tham số này được biến đổi sang khuôn mẫu dấu phẩy động bằng cách sử dụng hàm `sscanf()`.

Chương trình 11.6

```
/* prog11_6.c                                    */
/* Program to determine impedance of a           */
/* Transmission line                             */
/* Impedance of TL is                            */
/*     root((R+jwL)/(G+jwC))                     */
/* arguments are passed via program prompt       */
#include <stdio.h>
#include <math.h>
#define   MILLI       1e-3
#define   MICRO       1e-6
#define   BIGVALUE    1e38
#define   PI          3.14159

float calc_mag(float r,float l,float g,float c,float f);
float rect_to_polar(float x,float y);
float calc_imp(float f, float val);

int main(int argc, char *argv[])
float Zmag,R,L,G,C,f;

    puts("Prog to determine impedance of a trasmission line");
    if (argc!=6)
    {
      puts("Data format is:");
      puts(" trans R L G C freq ");
      return(1);
```

```
    }
      sscanf(argv[1],"%f",&R);sscanf(argv[2],"%f",&L);
      sscanf(argv[3],"%f",&G);sscanf(argv[4],"%f',&C);
      sscanf(argv[5],"%f",&f);

      Zmag=calc_mag(R,L*MILLI,G*MILLI,C*MICRO,f);
      printf("Magnitude is %.2f ohms\n",Zmag);
      return(0);
}
float calc_mag(float r,float 1,float g,float c,float  f)
{
float value1,value2;

  value1=rect_to_polar(r,calc_imp(f,1));
  value2=rect_to_polar(g,calc_imp(f,c));

  /* Beware if dividing by zero */
  if (value2==0) return(BIGVALUE);
  else return(sqrt(value1/value2));
}
float rect_to_polar(float x,float y)
{
  return(sqrt((x*x)+(Y*Y)));
{
float calc_imp(float f, float val)
{
  return(2*PI*f*val);
}
```

Kết quả chạy thử 11.3 giới thiệu một thí dụ làm mẫu.

Chạy thử 11.3

```
> trans
Program to determine impedance of a transmission line
Data format is:
  trans R L G C freq
> trans 10 10 8 5 100
```

```
Program to determine impedance of transmission line
Magnitude is 37.07 ohms
```

11.4 THỰC HÀNH

Q11.1 Hãy viết một chương trình có thể đọc một tệp dữ liệu chứa giá trị dấu phẩy động và in ra giá trị trung bình, cực đại và cực tiểu. Tên của tệp dược chuyển giao vào cho chương trình qua các tham số dòng lệnh. Một kết quả chạy chương trình được giới thiệu trong chạy thử 11.4.

Chạy thử 11.4

```
> getaver in.dat
The average value is 43.41
The maximum found is 100.34 and ninimum is -4.00
>
```

Q11.2 Hãy viết một chương trình cho các hàm toán học TANG, COS, SIN và hàm mũ (EXP). Chương trình sẽ nhận giá trị qua đối số dòng lệnh. Kết quả chạy thử 11.5 giới thiệu một thí dụ làm mẫu.

Chạy thử 11.5

```
> TAN 3.24
The tan of 3.24 is XXXX
> SIN 3.54
The sin of 3.54 is XXXXX
```

Q11.3 Hãy sửa đổi chương trình trong Q11.2 để người dùng có thể chỉ rõ hoặc là giá trị được tính theo độ (/d) hoặc theo radian (/r) hoặc đòi hỏi trợ giúp (/?). Chạy thử 11.6 giới thiệu một thí dụ làm mẫu.

Chạy thử 11.6

```
> TAN 30.24 /d
The tan of 3.24 degrees is XXXX
> SIN 30.54 /r
The sin of 30.54 radians is XXXXX
> SIN /?
FORMAT:   SIN VALUE [EXTENSION]
This is a program which will determine the
Sine if a value, where value is in degrees (/d)
or radians (/r). /? will print this page
```

Q11.4 Hãy sửa đổi một số chương trình ở chương trước để người dùng có thể yêu cầu trợ giúp khi chương trình đang chạy bằng cách sử dụng /? trên dòng lệnh. Kết quả chạy thử 11.7 giới thiệu một thí dụ làm mẫu.

Chạy thử 11.7

```
C:> RESON_LC /?
FORMAT:RESON-LC R C F [EXTENSION]
This is a program which will determine the
resonant frequency or a parallel RLC circuit
/? will print this page
C:>
```

Q11.5 Hãy viết một chương trình với đối số dòng lệnh, có khả năng xác định một giá trị thường được sử dụng đối với một điện trở. Chạy thử 11.8 giới thiệu một thí dụ làm mẫu.

Chạy thử 11.8

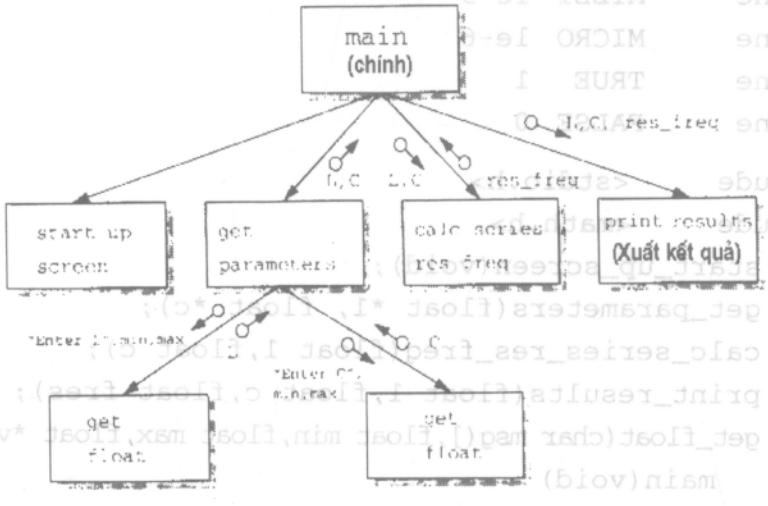
```
C:> R-VALUE 1.43 K
Nearest value is 1.4K
C:>
```

Chương 12
CÁC ĐỒ ÁN

Mục tiêu của chương này là tìm hiểu thêm về các kỹ thuật được sử dụng trong lập trình bằng ngôn ngữ C thông qua các bài toán cụ thể, mà ta gọi chung là các *đồ án làm việc*. Các mục 12.1 tới 12.5 có các chương trình mẫu dùng để minh hoạ với cấu trúc mã rõ ràng. Các chương trình viết trong phần thực hành có cấu trúc tương tự nhau.

12.1 TẦN SỐ CỘNG HƯỞNG CỦA MẠCH RLC NỐI TIẾP

So sánh với chương trình 2.9 để hiểu thêm về cơ sở của bài toán này. Một biểu đồ cấu trúc của thiết kế cho trong hình 12.1.



Hình 12.1: **Biểu đồ cấu trúc của chương trình 12.1.**

Chương trình 12.1 xác định tần số cộng hưởng của một mạch RLC nối tiếp theo các giá trị được nhập vào của điện cảm và điện dung. Các giá trị của L và C được giới hạn tới 1 nH và 1 pF, tương ứng, bằng các tham số được chuyển giao vào `get_float()`. Đầu mục chương trình được bổ sung vào trong chương trình để giới thiệu một vài thông tin về tác giả của chương trình, bộ nhớ truy nhập ngẫu nhiên, ngày tháng tạo ra, số hiệu phiên bản hiện hành, tên tệp tin, v. v...

Chương trình 12.1

```
/*****************************************************/
/* File:       Prog12_1.c                            */
/* Title:      Series Resonant Frequency Program     */
/* Function:   To determine the resonant frequency   */
/* of a series RLC circuit                           */
/* Author(s):              Bill Buchanan             */
/* Version:                1.00                      */
/* Created:                18-JAN-94                 */
/* Last Modified:          18-JAN-94                 */
/* Recent Modifications:   NONE                      */
/*****************************************************/

#define     PI 3.14159265358979323846
#define     MILLI 1e-3
#define     MICRO 1e-6
#define     TRUE  1
#define     FALSE 0

#include     <stdio.h>
#include     <math.h>
void start_up_screen(void);
void get_parameters(float *l, float *c);
float calc_series_res_freq(float l,float c);
void print_results(float l,float c,float fres);
void get_float(char msg[],float min,float max,float *val);
int     main(void)
{
float   L,C,res_freq;
```

```
  start_up_screen();
  get_parameters(&L,&C);
  res_freq=calc_series_res_freq(L,C);
  print_results(L,C,res_freq);
  return(0);
}
void    start_up_screen(void)
{
  puts("");
  puts("\tProgram to determine the resonant frequency");
  puts("\tof a series RLC circuit. Note that values");
  puts("\tfor inductance and capacitance are entered");
  puts("\tas milliHenries and microFarads respectively");
  puts("");
}
void    get_parameters(float *1, float *c)
{
  /* Values of L and C are entered as mH and uF respectively */
  /* and then scaled using the tokens MILLI and MICRO*/
  get_float("Enter inductance(mH) >>",1e-6,1e6,1);/* min 1 nH*/
  get_float("Enter resistance(uf) >>",1e-6.1e6.c);/* min 1 pF*/
  *1=(*1)*MILLI;
  *c=(*c)*MICRO;
}
float calc series_res_freq(float 1,float c)
}
float freq;
   freq=1/(2*PI)*sqrt(1/(1*c));
   return(freq);
}
void    print_results(float 1,float c,float fres)
{
   printf("Circuit values are %.2f mH, %.2f uF\n",
```

```
                1/MILLI,c/MICRO);
    printf("Resonant frequency is %.2f Hz\n",fres);
}
void   get_float(char msg[],float min,float max,float *val)
{
char   inline[BUFSIZ];
int    rtn,okay;
    /* get floating point value in the range min to max */
    do
    {
        printf("%s",msg);
        gets(inline);
        rtn=sscanf(inline,"%f",val);
        if ((rtn!=1) || (*val<min) || (*val>max))
        {
          okay=FALSE;
          printf("Invalid input <%s>\n",inline);
        }
        else okay=TRUE;
   ) while (!okay);
}
```

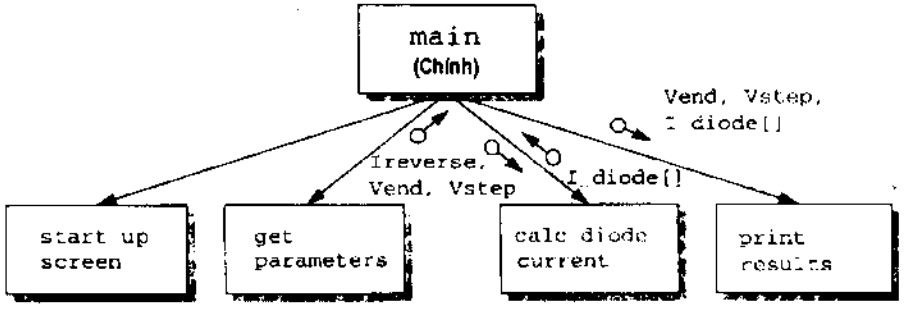
Kết quả chạy thử 12.1 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.1

```
  Program to determine the resonant frequency
  of a series RLC circuit. Note that values
  for inductance and capacitance are entered
  as millihenries and microfarads respectively
Enter inductance(mH) >> 1
Enter resistance(uF) >> 1
Circuit values are 1.00 mH, 1.0 uF
Resonant frequency is 5032.92 Hz
```

12.2 DÒNG ĐIỆN QUA ĐIỐT

So sánh với chương trình 4.4 để hiểu thêm về cơ sở của bài toán này. Biểu đồ cấu trúc mức thứ nhất của bài toán này được minh hoạ trên hình 12.2. Chú ý rằng hình vẽ không chỉ ra môđun con (sub-module) gọi `get_float()` từ `get_parameters()`.



Hình 12.2: **Sơ đồ cấu trúc cuả chương trình 12.2.**

Chương trình 12.2 xác định dòng điện đi qua một điốt.

Chương trình 12.2

```
/*****************************************************/
/* File:        prog12_2.c                           */
/* Title:       Diode Current Program                */
/* Function:  Program to determine the current       */
/* flow in a diode given an applied voltage          */
/* Author(s)      Bill Buchanan                      */
/* Version:       1.00                               */
/* Created:        18-JAN-94                         */
/* Last Modified: 18-JAN-94                          */
/* Recent Modifications: NONE                        */
/*****************************************************/
#include <stdio.h>
#include <math.h>

#define     TEMPERATURE    300 /*Room temperature*/
#define     MICRO  1e-6                          */
#define     MAXCURRENTVALUES 100 /* Maximum values in I_diode */
```

```
#define  TRUE    1
#define  FALSE   0

void   start_up_screen(void);
void   get_parameters(float *Io,float Vend,-'int *vn);
void   get_diode_current(floac Io,float Vend,int Vn,float I[]);
void   show_diode_current(float Vend,int Vn,float I[]);
void   get_float(char msg[],floar min,float max,float *val);

int      main(void)
{
float   I_reverse_sat,Vend;
int     Vsteps;
float    I_diode[MAXCURRENTVALUES];
  start_up_screen();
  get_parameters(&I_reverse_sat,&Vend,&vsteps);
  get_diode_current(I_reverse_sat,Vend, Vsteos,I_diode);
  show_diode_current(Vend,Vsteps,I_diode);
        return(0);
}
void     start_up_screen(void)
{
  puts("");
  puts("\tProgram to determine the current flow in a");
  puts("\tdiode given an applied voltage. The program");
  puts("\tassumes a temperature of 27 degrees C");
  puts("");
}
void    get_parameters(float *Io,float *Vend,int *Vn)
float temp; /* used to get number of voltage steps */

  get_float("Enter reverse saturation current >>",0, 1e-6,Io);
  get_float("Enter end voltage          >>",0,10,Vend);
  get_float("Enter number of voltage steps >>",1,
                          MAXCURRENTVALUES,&temp);
  *Vn=temp;
```

```
}
void get_diode_current(float 70,float Vend,int Vn,float I[])
{
int     i=0;
float   v;

  for (v=0;v<Vend;v+=Vend/Vn,i++)
    I[i]=Io*exp(11600*v/TEMPERATURE-1);
}
void  show_diode_current(float Vend.int Vn,float I[])
int i=0;
float   v;
  puts("VOLTAGE   CURRENT(uA)");
  for (v=0;v<Vend;v+=Vend/Vn,i++)
   printf("%12.2f %12.2f\n",v,1lil/MICRO);
  puts("");
}
void  get_floar(char msg[1,float min,float max,float *val)
{
char    inline[BUFSIZ];
int     rtn,okay;

  do
  {
    printf("%s",msg);
    gets(inline);
    rtn=sscanf (inline,"%f",val) ;
    if ((rt-n! =1)||(*val<min)||(*val>max))
    }
      okay=FALSE;
      printf("Invalid input <%s>\n",inline);
    }
    else okay=TRUE,
  } while (!okay);
}
```

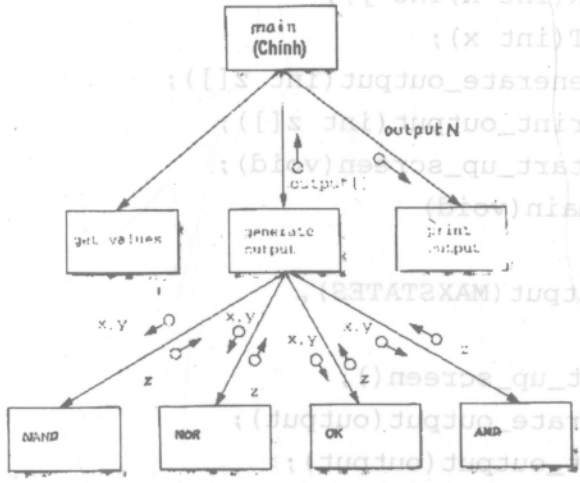
Kết qủa chạy thử 12.2 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.2

```
/* Program to determine the current flow in a
diode given an applied voltage. The program
assumes a temperature of 27 degrees C.
Enter reverse saturation current >> -1
Invalid saturation current please re-enter
Enter reverse saturation current >> 1e-12
Enter end voltage                >> -43
Invalid end voltage please re-enter
Enter end voltage     >>-0.8
Enter number of voltage steps     >>120
Invalid number of steps, max is 100
Enter number of voltage steps     >>20
  VOLTAGE     CURRENT(uA)
     0.00            0.00
     0.04            0.00
     0.08            0.00
     0.12            0.00
     0.16            0.00
     0.20            0.00
     0.24            0.00
     0.28            0.02
     0.32            0.09
     0.36            0.41
     0.40            1.92
     0.44            9.01
     0.48           42.29
     0.52          198.57
     0.56          932.46
     0.60         4378.63
     0.64        20561.13
     0.68        96550.87
     0.72       453383.15
     0.76      2128994.46
```

12.3 MẠCH BOOLE

So sánh với mục 5.5 để hiểu thêm về cơ sở của bài toán này.



Hình 12.3: **Sơ đồ cấu trúc cuả chương trình 12.3.**

Chương trình 12.3 xác định bảng chân lý của một mạch Boole.

Chương trình 12.3

```
/*****************************************************/
/* File:       progl2_3.c                            */
/* Title:     Boolean circuit analysis               */
/* Function: Program to determine the truth table    */
/* for the function NAND(OR(NOR(A,B),AND(A,C)),C)    */
/* Author(s):    Bill Buchanan                       */
/* Version:      1.00                                */
/* Created:      18-JAN-94                           */
/* Last Modified:          18-JAN-94                 */
/* Recent modifications:    NONE                     */
/*****************************************************/
#include    stdio.h>
#define     MAXSTATES 8
#define     TRUE      1
#define     FALSE     0
```

```
int AND(int x,int y);
int NAND(int x,int y);
int OR(int x,int y);
int NOR(inc X,int y);
int NOT(int x);
void generate_output(int z[]);
void print_output(int z[]);
void start_up_screen(void);
int  main(void)
{
int  output(MAXSTATES),

  start_up_screen();
  generate_output(output);
  print_output(output);
  return(0);
}
void   generate_output(int z[])
{
int    A,B,C,state=0;
   for (A=FALSE;A<=TRUE;A++)
         for (B=FALSE;B<=TRUE;B++)
               for (C=FALSE;C<=TRUE;C++)
         {
           z[state]=NAND(OR(NOR(A,B),AND(A,C)),C);
           state++;
         }
}
void print output(int z[]);
{

int     A,B,C,state=0;
   puts("Boolean function NOR( AND(A,B),C)");
   puts("    A    B    C    Z");
   for (A=FALSE;A<=TRUE;A++)
```

```
        for (B=FALSE;B<=TRUE;B++)
              for (C=FALSE;C<=TRUE;C++)
              {
                printf("%4d %4d %4d %4d\n",A,B,C,z[state]);
                state++;
              }
   puts("");
}
int AND(inr x,int y)
{
   return(x&y);
}
int NAND(inr x,int y)
{
   return( !(x&y) );
}
int OR(int x,int y)
{
   return( x|y );
}
int NOR(int x,int y)
{
return ( !(x|Y) );
     /* possible also with bit-masking ie return((-(x|y))&1);*/
}
int NOT (int x)
{
    return(!(x));
}
void start_up_screen (void)
{
   puts("") ;
   puts("\tProgram to determine truth table for the");
   puts ("\ tboolean function NAND(OR(NOR(A,3), AND (A, C)),C)");
   puts("");
```

```
}
```

Kết quả chạy thử 12.3 giới thiệu một thí dụ làm mẫu

Chạy thử 12.3

```
   Program to determine truth table for the
   boolean function NAND(OR(NOR(A,B),AND(A,C)),C)

Boolean function NAND(OR(NOR(A,B),AND(A,C)),C)
  A     B     C     Z
  0     0     0     1
  0     0     1     0
  0     1     0     1
  0     1     1     1
  1     0     0     1
  1     0     1     0
  1     1     0     0
  1     1     1     0
```

12.4 BỘ BIẾN ĐỔI THẬP PHÂN SANG NHỊ PHÂN

Chương trình 12.4 hiển thị số nhị phân tương đương của một số nguyên có dấu. So sánh với mục 1.7 để hiểu rõ thêm cơ sở của vấn đề này.

Chương trình 12.4

```
/*************************************************/
/* File: prog12_4.c                              */
/* Title:      Signed Decimal To Binary Program  */
/* Function:   Program to convert from signed    */
/* decimal to 2's complement binary              */
/* Author(s):    Bill Buchanan                   */
/* Version:      1.00                            */
/* Created:      18-JAN-94                       */
/* Last modified:  18-JAN-94                     */
/* Recent modifications:  NONE                   */
/*************************************************/
```

```
#include <stdio.h>
#include <math.h>
#define TRUE   1
#define FALSE  0
  void  start_up_screen(void);
  void  orint_binary(int dec);
  void  get_int(char msg[],int min,int max,int *val);

int     main(void)
{
int  decimal;

 start_up_screen();

 do
 {
  get_int("Enter a decimal value >> -32768,32767,&:decimal);
  if (decimal!=0) print_binary(decimal);
  } while (decimal!=0);
  return(0);
  }
void    start_up_screen(void)
{
  puts("");
  puts("\tProgram to determine the binary equivalent of");
  puts("\tof a signed decimal value. Note that this program");
  puts("\twill determine the number of bits used to store an");
  puts("\tinteger value. To end program enter a decimal value");
  puts("\tof zero.");
  puts("");
  }
void    print_binary(int dec)
{
unsigned bytes,bits,i;
  bytes=sizeof(inc);
      /* Determine number of bytes in an integer */
```

```
  bits=8*bytes;
     /* Determine the number of bits  */

   puts (" ") ;

   printf ("The binary equivalent of %10d is",dec);
   for (i=pow(2,bits-1);i>0;i>>=1)
     if (dec & i) prinrf("1"); else printf("0");

   puts (" ");
}
void get_int(char msg[],inz min,int max,int *val)
{
char inline[BUFSIZ];
int  rtn,okay;
  do
  {
   printf("%s",msg);
   gets(inline);
   rtn=sscanf(inline,"%d",val);
  if ((rtn!=1 || (*val<min) || (*val>max))
  {
      okay=FALSE;
      printf("Invalid input <%s>\n",inline);
  }
  else okay=TRUE;
  } while (!okay);
}
```

Kết quả chạy thử 12.4 giới thiệu một thí dụ làm mẫu

Chạy thử 12.4

```
Program to determine the binary equivalent of
of a signed decimal value. Note that this program
will determine the number of bits used to store an
```

```
integer value.To end program enter a decimal value
 of zero.
Enter decimal value >> -1
The binary equivalent of -1 is 1111111111111111
Enter decimal value >> 43
The binary equivalent of 43 is 0000000000001001011
Enter decimal value >> 1024
The binary equivalent of 1024 is 0000010000000000
Enter decimal value >> -453
The binary equivalent of -453 is 1111111000111011
Enter decimal value >>  22
The binary equivalent of 22 is 0000000000010110
winter decimal value >> 10000
The binary equivalent of 10000 is  0010011100010000
```

12.5 CHƯƠNG TRÌNH MÃ MÀU ĐIỆN TRỞ

Thông thường giá trị của các điện trở được xác định bởi một hệ thống các vòng màu hay còn gọi là mã màu. Mã màu của các điện trở được cho trên bảng 12.1.

Bảng 12.1: **Mã màu của điện trở.**

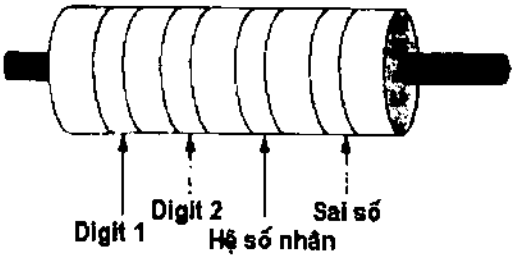
Số	Màu	Hệ số nhân	Số số 0
	Bạc nhũ	0.01	2
	Vàng nhũ	0.1	1
0	Đen	1	0
1	Nâu	10	1
2	Đỏ	100	2
3	Cam	1 k	3
4	Vàng	10 k	4
5	Xanh lục	100 k	5
6	Xanh lam	1 M	6
7	Tím	10 M	7
8	Xám		
9	Trắng		

Hai vòng màu đầu tiên chỉ ra hai con số*) , vòng thứ ba là một hệ số nhân còn vòng thứ tư chỉ sai số (độ chính xác). Bảng 12.2 đưa ra một danh sách mã màu của các vòng sai số.

Bảng 12.2: **Vòng màu sai số của điện trở.**

Màu	Sai số
Đỏ	2 %
Vàng nhũ	5 %
Bạc nhũ	10 %
Không có	20 %

Hình 12.4 mô tả một điện trở với 4 vòng màu:



Hình 12.4: **Mã màu của điện trở với 4 vòng màu.**

Thí dụ:

Đỏ ,đỏ, nâu, bạc nhũ	220 Ω, ±10 %
Lục, lam ,vàng, vàng nhũ	560 kΩ, ±5 %

Program 12.5

```
/************************************************/
/* File:        prog12_5.c                      */
/* Title:       Resistor Colour Code Program    */
/* Function:    Program to determine the colour */
```

*) Mã vòng màu được trình bày ở đây chỉ đúng với các điện trở có bốn vòng màu. Hiện nay, do công nghệ phát triển, nhiều loại điện trở có độ chính xác cao được chế tạo với giá thành không tăng hơn. Các điện trở này được ghi giá trị bằng năm vòng màu: ba vòng đầu là các con số chỉ giá trị, vòng thứ tư là hệ số nhân, vòng thứ năm là độ chính xác (ND).

```
/* bands tor a 4-band resistor                          */
/* Author(s):    Bill Buchanan                          */
/* Version:      1.01                                   */
/* Created:      20-JAN-94                              */
/* Author(s):     Bill                                  */
/* Version:       1.01                                  */
/* Created:       20-JAN-94                             */
/* Last Modified: 22-JAN-94                             */
/* Recent Modifications: NONE                           */
/******************************************************/
#include <stdio.h>
#define   TRUE 1
#define   FALSE 0

typedef struct
{
     int first_digit,second_digit,no_zeros;
     int tolerance;
} res_colour_band;
enum cols {SILVER=-2,GOLD,BLACK,BROWN,RED,ORANGE,YELLOW,
                GREEN,BLUE,VIOLET,GREY,WHITE};
char
*colour [12]=("SILVER","GOLD","BLACK","BROWN","RED","ORANGE",
 "YELLOW","GREEN","BLUE","VIOLET","GREY","WHITE");
void    get_res_codes(res_colour_band *c_bands);
void    show_colours (res_colour_band c_bands)
void    get_int(char msg[],int min,int max;int *val);
int     main (void)
{
res_colour_band resistor;

  get_res_codes (&resistor)
  show_colours(resistor);
  return=(0);
}
```

```
void     get_res_codes(res_colour_band *c_band)
{
  get_int("Enter 1st colour band >>"
  BLACK,WHITE,&c_band->first_digit);

  get_int("Enter 2nd colour band >>",
          BLACK,WHITE,&c-band->second_digit)
  get_int("Enter number of zeros >>",
     SILVER,VIOLET,&c_band->no_zeros)
  get_int("Enter tolerance >>",0,100,&c_band->tolerance);
}
void     show_colours(res_colour_band c_bands)
{
  printf("Resistor colour bands %s %s %s ",
     colours[c_bands.first_digit+2],
     colours[c-bands.second_digit+2],
     colours(c_bands.no_zeros+2]);

  if (c_band,tolerance>=20)        printf("NONE");
  else if (c_bands.tolerance>=10)  printf("SILVER");
  else if (c_bands.tolerance>=5)   printf("GOLD");
  else printf("RED\n");
}
void     get_int(char msg[],int min,int max,int *val)
{
char          inline[BUFSIZ];
int           rtn,okay;
  do
  {
     printf("%s",msg);
     gets(inline);
     rtn=sscanf(inline,"%d",val);
     if ((rtn!=1)  ||   (*val<min)   ||   (*val>max))
     {
          okay=FALSE;
```

```
            printf("Invalid input <%s>\n",inline);
            }
            else okay=TRUE;
        } while (!okay);
}
```

Kết quả chạy thử 12.5 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.5

```
Enter 1st coloour band >> 4
Enter 2nd coloour band >> 5
Enter number of zeros >>  4
Enter tolerance >>        5
Resistor colour bands YELLOW GREEN YELLOW GOLD
```

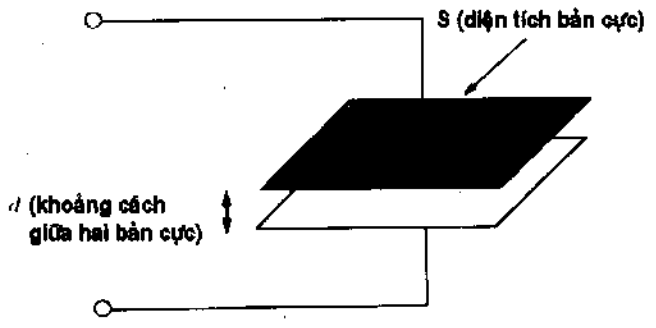
12.6 THỰC HÀNH

Q12.1 Một tụ điện phẳng song song, được mô tả trên hình 12.5, bao gồm hai bản cực phẳng song song ngăn cách nhau bằng chất điện môi là không khí. Viết một chương trình xác định điện dung của tụ điện trên. Chương trình sẽ nhắc để nhập vào diện tích bản cực và chiều dày của lớp điện môi (hay khoảng cách giữa hai bản cực).

Điện dung của một tụ điện với bản cực phẳng được ngăn các bởi vật liệu cách điện có hằng số điện môi bằng ε_r được cho bởi công thức sau đây:

$$C = \frac{\varepsilon_0 \varepsilon_r . S}{d} \quad [F]$$

ở đây hằng số điện môi của không gian tự do $\varepsilon_e = 8{,}854 \times 10^{-12}$ $F.m^{-1}$.



Hình 12.5: **Tụ điện phẳng song song.**

Diện tích bản cực được tính bằng mm^2 còn khoảng cách giữa các bản cực tính bằng mm. Điện dung hiển thị sẽ được tính bằng đơn vị thích hợp nhất (mF, μF, nF hoặc pF). Chẳng hạn, nếu điện dung lớn hơn 0,001 F thì đơn vị được hiển thị ra mF; nếu lớn hơn 0,000001 thì hiển thị ra μF v. v... Bảng 12.3 chỉ ra vùng giới hạn của các giá trị được nhập vào.

Bảng 12.3: **Các giá trị cực tiểu và cực đại.**

	Cực tiểu	Cực đại
d	0,1 μm	1000 nm
S	0,001 mm^2	10.000 m^2
ε_r	1	12

Kết quả chạy thử 12.6 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.6

```
*************************************************************
* CAPACITOR Version 1.00
* Author: Bill Buchanan
* Description:
*  Program to determine the capacitance of a
*  parallel plate capacitor given the distance
*  between the plates, the area of the plates
```

```
* and the dielectric constant of the material
* separating the plates
************************************************************
Enter plate separation (mm) >> 0.1
Enter area of plates (mm2) >> 1000
Enter the dielectric constant of material  >> 10
The capacitance is 885.40 pF
Enter plate separation (mm) >> 0.01
Enter area of plates (mm2) >> 10000
Enter the dielectric constant of material  >> 10
The capacitance is 88.54 nF
```

Q12.2 Cực cửa của tranzito trường (FET) được cách ly với đế bằng một lớp ôxyt silic (SiO_2). Điện dung của tiếp giáp cửa-nguồn có thể tính gần đúng và thông qua điện dung tụ phẳng song song có thể tính gần đúng như sau:

$$C = \frac{\varepsilon_o \varepsilon_r . WL}{T_{ox}} \quad [F]$$

ở đây:

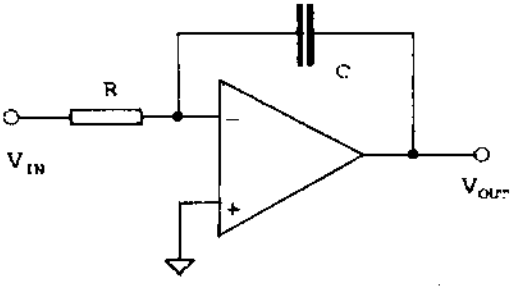
W - chiều rộng của cửa (gate)
L - chiều dài của cửa (gate)
ε_r - hằng số điện môi của ôxyt silic (SiO_2)
Tox - tiết diện ngang của dây dẫn ?

Hãy viết một chương trình hiển thị điện dung nguồn- cửa ứng với các giá trị được nhập vào của W; L; ε_r và T_{OX}. Vùng giá trị hợp lệ của các giá trị nhập vào được chỉ ra trên bảng 12.4.

Bảng 12.4: **Các giá trị cực tiểu và cực đại.**

	Cực tiểu	Cực đại
W	0,1 μm	100 μm
L	0,1 μm	100 μm
ε	2	3
T_{OX}	0,1 μm	100 μm

Q12.3 Hình 12.6 mô tả một bộ lọc tích cực sử dụng bộ khuếch đại thuật toán. Mạch này có hệ số khuếch đại cao ở tần số thấp và hệ số khuếch đại cao thấp ở vùng tần số cao. Như vậy nó hoạt động giống như một bộ lọc dải thông thấp.



Hình 12.6: **Bộ lọc tích cực dùng mạch RC.**

Hệ số khuếch đại của mạch này cho bởi:

$$|\text{HSKĐ}| = \frac{1}{2\pi . fRC}$$

Hãy viết một chương trình trong đó người dùng nhập vào một giá trị của: tần số, điện trở và điện dung. Chương trình sẽ xác định hệ số khuếch đại (tính theo dB) tương ứng với các giá trị được nhập vào. Bảng 12.5 đưa các vùng giá trị hợp lệ đối với các linh kiện.

Bảng 12.5: **Các giá trị cực tiểu và cực đại.**

	Cực tiểu	Cực đại
f	1 Hz	100 MHz
R	0 Ω	10 MΩ
C	1 pF	1 mF

Q12.4 Điện trở của một dây dẫn hình trụ là một hàm của điện trở suất của vật liệu, tiết diện ngang và chiều dài của dây dẫn, và được tính theo công thức:

$$R = \frac{\rho . l}{S} \quad [\Omega]$$

ở đây:

ρ	- điện trở suất của vật liệu dẫn	[Ω.m];
l	- chiều dài của dây dẫn	[m];
S	- tiết diện ngang của dây dẫn	[m^2];

Hãy viết một chương trình xác định điện trở của một dây dẫn làm bằng một trong bốn kim loại: đồng, nhôm, bạc và măng gan. Bảng 12.6 liệt kê các giá trị điện trở suất của các kim loại này. Chương trình sẽ xác định điện trở tương ứng với một kim loại được chọn, chiều dài và tiết diện ngang của dây dẫn, và sẽ hiển thị kết quả theo đơn vị thích hợp nhất (nΩ, μΩ, mΩ, Ω, kΩ, v. v...).

Bảng 12.6: **Điện trở suất của một số kim loại.**

Kim loại	Điện trở suất [Ω.m]
Đồng	$17x10^{-9}$
Nhôm	$25,4x10^{-9}$
Bạc	$16x10^{-9}$
Măng gan	$1400x10^{-9}$

Kết quả chạy thử 12.7 chỉ ra một thí dụ làm mẫu. Trong trường hợp này đã sử dụng một dây dẫn nhôm với bán kính 1 mm và chiều dài bằng 1000 m. Điện trở tính được bằng 8,09 Ω.

Chạy thử 12.7

```
Type of conductor >>
(c)opper
(a)luminium
(s)ilver
(m)anganese
Option >>    a
Enter radius and length of conductor >> 1e-3 1000
Resistance of conductor is 8.09e+00 ohm
```

Q12.5 Hãy viết một chương trình xác định bảng giá trị chân lý của các phương trình sau:

$$Z = \overline{\overline{A + B} + B\overline{C}}$$

$$Z = \overline{\overline{(A + B)}.(\overline{B} + C)}$$

$$Z = \overline{\overline{A + B + C} + B\overline{C} + A}$$

$$Z = \overline{\overline{A + B + D} + (C.D)} + \overline{A}$$

$$Z = \overline{\overline{A + B + C} + (\overline{B}.C.D)} + A$$

Q12.6 Một mạch lọc Butterworth tạo ra một đáp tuyến dải thông bằng phẳng. Độ rộng của đáp tuyến được cho bởi:

$$\frac{V_{OUT}}{V_{IN}} = \frac{1}{\sqrt{1 + (f / f_C)^{2n}}}$$

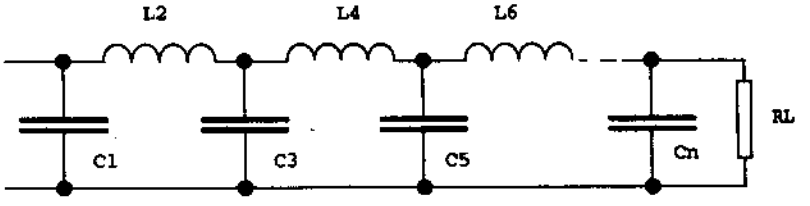
ở đây n là bậc của bộ lọc và f_C là tần số cắt (hoặc -3 dB). Hãy viết một chương trình hiển thị đáp ứng đối với n = 1; 2; 4; 8; 16 và 32. Một kết quả chạy thử mẫu với tần số cắt bằng 1 kilô hec (kHz) được giới thiệu trong chạy thử 12.8. Chương trình chạy thử đã hiển thị 20 bước tần số (không kể 0 Hz) trong phạm vi từ 0 Hz tới giá trị bằng hai lần tần số cắt. Chú ý rằng ở tần số cắt hệ số khuếch đại bằng 0,707 đối với tất cả các bậc bộ lọc.

Chạy thử 12.8

```
Enter cut-off frequency >>1e3
Frequency    n=1    n=2    n=4    n=8   n=16   n=32
    0.000  1.000  1.000  1.000  1.000  1.000  1.000
  100.000  0.995  1.000  1.000  1.000  1.000  1.000
  200.000  0.981  0.999  1.000  1.000  1.000  1.000
  300.000  3.958  0.996  1.000  1.000  1.000  1.000
  400.000  0.928  0.987  1.000  1.000  1.000  1.000
  500.000  0.894  0.970  0.998  1.000  1.000  1.000
  600.000  0.857  0.941  0.992  1.000  1.000  1.000
  700.000  0.819  0.898  0.972  0.998  1.000  1.000
  800.000  0.781  0.842  0.925  0.986  1.000  1.000
  900.000  0.743  0.777  0.836  0.919  0.983  0.999
```

1000.000	0.707	0.707	0.707	0.707	0.707	0.707
1100.000	0.673	0.637	0.564	0.423	0.213	0.047
1200.000	0.640	0.570	0.434	0.227	0.054	0.003
1300.000	0.610	0.509	0.330	0.122	0.015	0.000
1400.000	0.581	0.454	0.252	0.068	0.005	0.000
1500.000	0.555	0.406	0.194	0.039	0.002	0.000
1600.000	0.530	0.364	0.151	0.023	0.001	0.000
1700.000	0.507	0.327	0.119	0.014	0.000	0.000
1800.000	0.486	0.295	0.095	0.009	0.000	0.000
1900.000	0.466	0.267	0.077	0.006	0.000	0.000
2000.300	0.447	0.243	0.062	0.004	0.000	0.000

Q 12.7 Một bộ lọc Butterworth hình π được mô tả trên hình 12.7.



Hình 12.7: **Bộ lọc thông thấp Butterworth hình π.**

Những linh kiện có thể được tính toán bằng cách sử dụng các giá trị chuẩn hóa được cho trong bảng 12.7 và sắp xếp chúng theo trình tự (scale) đối với điện trở tải cần có và tần số cắt.

Bảng 12.7: **Các giá trị linh kiện đã được chuẩn hoá.**

n	C1	L2	C3	L4	C5
2	1,4142	1,4142			
3	1,0	2,0	1,0		
4	0,7654	1,8478	1,8478	0,7654	
5	0,6180	1,6180	2,0	1,6180	0,6180

Các giá trị đã được chuẩn hóa này sau đó được xếp theo trình tự để tạo ra giá trị hiện tại bằng cách sử dụng công thức:

$$C_n\ (hiện\ tại) = \frac{C_n(table)}{2\pi . fR_L}$$

$$L_n\ (hiện\ tại) = \frac{R_L\ R_n(table)}{2\pi . f}$$

ở đây n là bậc của bộ lọc, R_L là điện trở tải và f là tần số cắt. Chẳng hạn, đối với một bộ lọc bậc hai các giá trị linh kiện sẽ là:

$$C_1 = \frac{1{,}4142}{2\pi . f . R_L}$$

$$L_2 = \frac{1{,}4142 . R_L}{2\pi . f}$$

Một thí dụ làm mẫu tiến hành trên một bộ lọc bậc năm với tần số điểm cắt là 1 MHz được minh hoạ trong chạy thử 12.9.

Chạy thử 12.9

```
Butterworth Low Pass Filter Program
***********************************
Enter cut-off frequency >>        1e6
Enter order of filter (2-5) >>    5
Enter load impedance (ohms) >>    75
c1= 1.31e-g F
L2= 19.30e-6 H
C3= 4.24e-9 F
L4= 19.30e-6 H
CS= 1.31e-9 F
```

Hãy sửa đổi chương trình sao cho có thể hiển thị các giá trị linh kiện với đơn vị đo thích hợp nhất. Một kết quả làm ví dụ đã qua sửa đổi được giới thiệu trong chạy thử 12.10.

Chạy thử 12.10

```
Butterworth Low Pass Filter Program
***********************************
Enter cut-off frequency >>     1e6
Enter order of filter (2-5) >> 5
```

```
Enter load impedance (ohms) >> 75
C1=  1.31 nF
L2= 19.30 uH
C3=  4.24 nF
L4= 19.30 uH
C5=  1.31 nF
```

Q12.8 Bộ lọc Butterworth dải thông thấp trên hình 12.7 có thể được sửa đổi để trở thành một bộ lọc tần số cao bằng cách thay thế các tụ điện bằng những cuộn cảm và các cuộn cảm bằng những tụ điện, và sử dụng các giá trị linh kiện sau:

$$C_n(\textit{hiện tại}) = \frac{1{,}4142}{2\pi . fR_L}$$

$$L_n(\textit{hiện tại}) = \frac{1{,}4142 . R_L}{2\pi . f}$$

Một thí dụ làm mẫu được minh hoạ trong chạy thử 12.11.

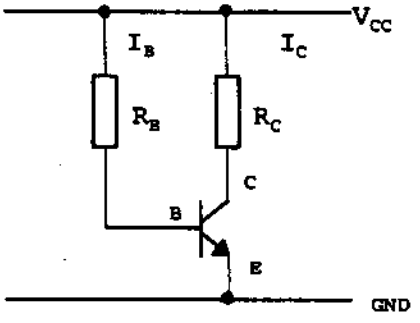
Chạy thử 12.11

```
Butterworth High Pass Filter Program
************************************
Enter cut-off frequency >> 6e6
Enter order of filter (2-5) >> 3
Enter load impedance (ohms) >> 50
C1= 5.10 nF
L2= 0.69 uH
C3= 5.10 nF
```

Q12.9 Hình 12.8 chỉ ra một mạch tranzito lưỡng cực thiên áp đơn giản bằng chỉ một điện trở cực gốc.

Đáng chú ý là, để tạo ra tín hiệu lối ra cực đại, điện áp cực góp được thiên áp (định thiên) ở giá trị bằng một nửa điện áp nguồn nuôi, trong trường hợp này là V_{CC}. Như vậy khi một dòng cực góp (I_C) đã được chỉ định thì điện trở cực góp R_C có thể được xác định (xem phương trình 1). Sau đó, dòng cực gốc được tính bằng

cách đem chia dòng cực góp cho hệ số khuếch đại h_{FE} (xem phương trình 2). Nếu tranzito mở (ON) thì tiếp giáp gốc-phát sẽ có một điện áp sụt trên điốt silic ở trạng thái dẫn ($V_{BE}(ON)$).



Hình 12.8: **Một mạch tranzito lưỡng cực thiên áp đơn giản chỉ bằng một điện trở cực gốc.**

Khi sử dụng giá trị gần đúng của điện áp này (- 0,65 V) thì điện trở cực gốc có thể được xác định (phương trình 3). Các phương trình mạch có thể sử dụng:

$$R_C = \frac{V_{CC}/2}{I_C} \quad [\Omega] \quad (1)$$

$$I_B = \frac{I_C}{h_{FE}} \quad [A] \quad (2)$$

$$R_B = \frac{V_{CC} - V_{BE}(ON)}{I_B} \quad [\Omega] \quad (3)$$

Hãy viết một chương trình xác định R_B tương ứng với các giá trị đã được nạp vào của I_C và V_{CC}. Giả sử rằng đó $V_{BE}(ON)$ bằng 0,65 V và h_{FE} bằng 100. Các giá trị của V_{CC} được nạp sẽ bị giới hạn giữa 5 và 30 V, còn đối với I_C thì 0,1 và 10 mA. Kết quả chạy thử 12.12 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.12

```
Enter Vcc (5->ISV)>> 15
Enter Ic (mA)      >> 1
Collector resistance is 15000 ohms
```

```
Base resistance is     1435000 ohms
```

Q12.10 Một bộ khuếch đại cực phát chung tự thiên áp được cho trên hình 12.9. Hãy viết một chương trình xác định tất cả các giá trị điện trở khi người dùng nhập vào giá trị của dòng cực góp, điện áp cực góp và điện áp nguồn nuôi. Người dùng được nhắc nhở tất cả các giá trị ở ngoài phạm vi cho phép. Một kết quả chạy thử được giới thiệu trong chạy thử 12.13. Các công thức tính gần đúng giá trị:

$$R_C = \frac{V_{CC} - V_C}{I_C} \quad [\Omega]$$

$$V_E = \frac{V_{CC}}{10} \quad [V]$$

$$R_E = \frac{V_E}{I_C} \quad [\Omega]$$

$$I_B = \frac{I_C}{h_{FE}} \quad [A]$$

$$V_B = V_{BE} + V_E \quad [V]$$

$$I_1 = 10.I_B \quad [A]$$

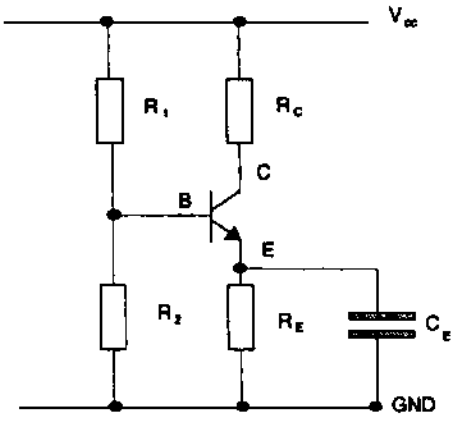
$$R_1 = \frac{V_{CC} - V_B}{I_1} \quad [\Omega]$$

$$R_2 = \frac{V_B}{I_1 - I_B} \quad [\Omega]$$

ở đây:

V_{CC}	-	Điện áp nguồn nuôi	[V]
V_C	-	Điện áp cực góp	[V]
I_C	-	Dòng cực góp	[A]
V_E	-	Điện áp cực phát	[V]
V_B	-	Điện áp cực gốc	[V]

Hãy sử dụng các giá trị gần đúng: hệ số khuếch đại là h_{FE} = 100 và V_{CC} (ON) = 0,65 V.



Hình 12.9: **Tầng khuếch đại cực phát chung.**

Chạy thử 12.13

```
Enter Vcc (V):15
Enter Collector Current (mAj: 1
Enter Collector Voltage (V) . 7.5
R1= 128000 ohm R2=24444 ohm R3= 1500 ohm R4=7500 ohm
```

Q12.11 Hãy sửa đổi chương trình trong Q 12.10 để các điện trở được hiển thị với đơn vị được lựa chọn thích hợp nhất. Kết quả chạy thử 12.14 giới thiệu một thí dụ làm mẫu.

Chạy thử 12.14

```
Enter Vcc (V) : 15
Enter Collector Current (rnA)
Enter Collector Voltage (V) : 7.5
R1= 128.0 Kohm R2=24.4 Kohm R3= 1.5 Kohm R4=7.5 Kohm
```

Q12.12 Hãy sửa đổi chương trình trong Q12.11 để có thể xác định các giá trị được ưa dùng nhất. Một danh sách các giá trị được ưa

dùng trong thực tế được cho trong bảng 11.8. Kết quả chạy thử 12.15 giới thiệu một thí dụ làm mẫu.

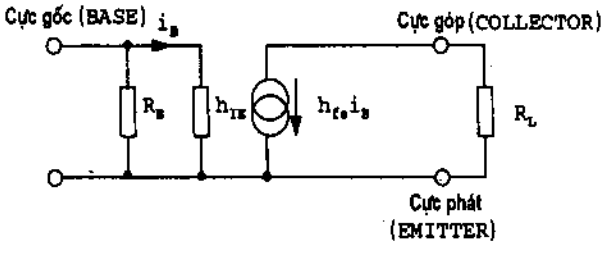
Bảng 12.8: **Các giá trị điện trở thường được dùng.**

10	16	27	43	68
11	18	30	47	76
12	20	33	51	82
13	22	36	56	91
15	24	39	62	100

Chạy thử 12.15

```
Enter Vcc (V):15
Enter Collector Current (mA): 1
Enter Collector Voltage (V): 7.5
R1= 130.0 Kohm R2=24.0 Kohm R3=1.5 Kohm R4=7.5 Kohm
```

Q12.13 Hãy sửa đổi chương trình trong Q12.12 để người dùng có thể đồng thời nhập vào hệ số khuếch đại dòng xoay chiều h_{fe} AC và trở kháng lối vào đối với dòng xoay chiều. Một sơ đồ tương đương thông số h đối với một mạch thiên áp một điện trở cực gốc được cho trên hình 12.10.



Hình 12.10: **Mạch tương đương thông số h cho mạch thiên áp dùng một điện trở cực gốc.**

William Buchanan

LẬP TRÌNH C TRONG KỸ THUẬT ĐIỆN TỬ

Người dịch: **Ngô Diên Tập**
Phạm Huy Quỳnh

Chịu trách nhiệm xuất bản:	PGS. PTS. Tô Đăng Hải
Biên tập:	Đặng Đình Thạch
Trình bày:	PTS Ngô Diên Tập
Sửa bản in:	Quang Ngọc
Chế bản:	Ngô Thanh Tùng
Vẽ bìa:	Phạm Huy Quỳnh

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 Phố Trần Hưng Đạo, HÀ NỘI, 1999

In 1.000 cuốn, khổ 16 x 24 cm, tại Xí nghiệp in Hàng không
Số xuất bản số: 41 - 72, ngày 04 tháng 3 năm 1999.
In xong và nộp lưu chiểu tháng 04 năm 1999

Giá: 42.000đ