MỤC LỤC

MỤC LỤC 1
Chương 1 KHÁI NIỆM VỀ HỆ THỐNG MÁY TÍNH 7
1.1. Lịch sử phát triển của máy tính điện tử 7
1.2. Khái niệm hệ thống máy tính 7
1.3. Cấu trúc cơ bản của hệ thống máy tính 9
1.4. Biểu diễn thông tin trong máy tính điện tử 10
1.4.1. Các hệ đếm 10
1.4.2. Chuyển đổi từ hệ thập phân sang các hệ khác 10
1.4.3. Chuyển từ hệ nhị phân sang các hệ khác 12
1.4.4. Chuyển từ hệ bát phân và hexa sang hệ thập phân và nhị phân 12
1.5. Các phép tính số học trên hệ nhị phân 13
1.5.1. Phép cộng 13
1.5.2. Phép trừ 14
1.5.3. Phép nhân 15
1.5.4. Phép chia 15
1.6. Khái niệm thuật toán 16
Chương 2: GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH C 17
2.1. Giới thiệu 17
2.1.1. Lịch sử ngôn ngữ C 17
2.1.2. Đặc điểm của ngôn ngữ C 17
2.1.3. Ứng dụng của ngôn ngữ C 17
2.2. Tập ký tự 17
2.3. Định danh 18
2.3.1. Từ khóa 18
2.3.2. Quy tắc đặt tên 18
2.4. Các kiểu dữ liệu 19
2.4.1. Kiểu dữ liệu cơ bản 19
2.4.2. Kiểu mở rộng 20
2.4.3. Địnhnghĩa kiểu dữ liệu bằng typedef 20
2.5. Biến 21
2.5.1. Khái niệm biến 21
2.5.2. Khai báo biến 21
2.5.3. Vị trí của khai báo biến 21
2.5.4. Lấy địa chỉ của biến 22
2.6. Hằng 22
2.6.1. Hằng kiểu số nguyên 23

2.6.2. Hằng kiểu long ... 23
2.6.3. Hằng dấu phẩy động ... 23
2.6.4. Hằng ký tự ... 23
2.6.5. Hằng chuỗi ... 24
2.6.6. Khai báo hằng ... 24
2.7. Các phép toán ... 24
2.7.1. Phép toán số học ... 24
2.7.2. Phép toán so sánh ... 26
2.7.3. Phép toán logic ... 26
2.7.4. Phép toán xử lý bit ... 27
2.7.5. Một số phép toán khác ... 28
2.8. Phép toán xử lý bit ... 29
2.8.1. Phép toán AND ... 29
2.8.2. Phép toán OR ... 29
2.8.3. Phép toán XOR ... 29
2.8.4. Phép toán NOT ... 30
2.8.5. Phép toán dịch trái/phải ... 30
2.9. Biểu thức ... 30
2.9.1. Khái niệm biểu thức ... 30
2.9.2. Phép gán ... 31
2.9.3. Chuyển đổi kiểu dữ liệu ... 31
2.10. Cấu trúc cơ bản của chương trình C ... 32
2.10.1. Cấu trúc chương trình ... 33
2.10.2. Một số chú ý khi viết chương trình ... 34
2.11. Làm quen với môi trường soạn thảo Turbo C ... 35
2.11.1. Mở Turbo C ... 35
2.11.2. Soạn thảo chương trình mới ... 35
2.11.3. Lưu chương trình vào bộ nhớ ... 36
2.11.4. Mở chương trình từ bộ nhớ ... 36
2.11.5. Dịch và thực hiện chương trình ... 36
2.11.6. Thoát khỏi môi trường C ... 36
Chương 3 CÁC LỆNH XUẤT NHẬP DỮ LIỆU ... 37
3.1. Giới thiệu chung ... 37
3.2. Các lệnh xuất dữ liệu cơ bản ... 37
3.2.1. Chuỗi điều khiển ... 37
3.2.2 Số nguyên hệ 10 có dấu và không dấu ... 39
3.2.3. Các lệnh xuất dữ liệu lên màn hình ... 41

3.2.4 Xuất dữ liệu ra máy in 44
3.3. Các lệnh nhập dữ liệu 44
3.3.1. Chuỗi điều khiển 44
3.3.2. Dòng vào chuẩn (Stdin) 46
3.4. Một số hàm xuất/nhập bổ sung 50
3.4.1. Các hàm xuất dữ liệu bổ sung 50
3.4.2. Các hàm nhập dữ liệu bổ sung 52
3.4.4. Xóa màn hình và di chuyển con trỏ 53
Chương 4 CÁC LỆNH ĐIỀU KHIỂN 55
4.1. Giới thiệu chung 55
4.2. Câu lệnh if 55
4.2.1. Cú pháp câu lệnh if 55
4.2.2. Lưu đồ 55
4.2.3. Giải thích lưu đồ 56
4.3. Câu lệnh if.. else 59
4.3.1. Cú pháp câu lệnh if … else (dạng đầy đủ) 59
4.3.2. Lưu đồ 59
4.3.3. Giải thích lưu đồ 59
4.4. Biểu thức điều kiện: 60
4.6. Câu lệnh switch 62
4.6.1. Cú pháp câu lệnh switch 62
4.6.2. Lưu đồ thuật toán câu lệnh switch 62
4.6.3. Giải thích lưu đồ 63
4.6.4. Một số chú ý 63
4.6.5. Một số ví dụ 63
4.7. Câu lệnh for 65
4.7.1. Cú pháp câu lệnh for 65
4.7.2. Lưu đồ thuật toán câu lệnh for 65
4.7.3. Giải thích lưu đồ 66
4.7.4. Một số chú ý 66
4.7.5. Một số ví dụ 66
4.8. Câu lệnh while 68
4.8.1. Cú pháp câu lệnh while 68
4.8.2. Lưu đồ thuật toán 68
4.8.3. Giải thích lưu đồ 69
4.8.4. Một số chú ý 69
4.8.5. Một số ví dụ 69

4.9. Câu lệnh do … while 70
4.9.1. Cú pháp câu lệnh do … while 70
4.9.2. Lưu đồ thuật toán 70
4.9.3. Giải thích lưu đồ...70
4.9.4. Ví dụ 71
4.10. Nhóm các lệnh điều khiển khác 71
4.10.1. Câu lệnh break...71
4.10.2. Câu lệnh continue..72
4.10.3. Câu lệnh return..72
4.10.4. Câu lệnh nhảy không điều kiện goto 73
Chương 5 HÀM 75
5.1. Giới thiệu chung..75
5.2. Định nghĩa và lời gọi hàm 75
5.2.1. Định nghĩa..76
5.2.2. Lời gọi hàm...76
5.2.3. Thí dụ về cách gọi hàm...77
5.3. Nguyên tắc hoạt động của hàm...78
5.4. Cách sử dụng các tham số trong hàm:...78
5.5. Chuyển giao tham số của hàm 79
5.5.1. Truyền bằng giá trị 79
5.5.2. Truyền bằng tham chiếu 79
5.6. Sự trả về từ hàm...80
5.7. Hàm đệ quy..80
5.8. Một số thí dụ..82
Chương 6 CON TRỎ 88
6.1. Giới thiệu...88
6.2. Biến con trỏ 88
6.2.1. Khái niệm con trỏ. 88
6.2.2. Khai báo biến con trỏ 88
6.2.3. Gán giá trị cho biến con trỏ 89
6.3. Cấp pháp bộ nhớ 91
6.3.1. Cấp phát bộ nhớ để lưu dữ liệu...91
6.3.3 Giải phóng vùng nhớ động 92
6.4. Các phép toán trên biến con trỏ 92
6.4.1. Phép cộng/trừ biến con trỏ với số nguyên 92
6.4.2 Phép gán và phép so sánh 93
6.4.3. Sự chuyển kiểu..94

6.4.4 Con trỏ hằng và con trỏ đến đối tượng hằng 94
6.4.5. Con trỏ NULL 95
6.5. Con trỏ và hàm 95
6.5.1. Khái niệm và cách khai báo con trỏ hàm 95
6.5.2. Đối con trỏ hàm 96
6.5.3. Thí dụ 96
Chương 7 MẢNG VÀ CHUỖI 99
7.1. Giới thiệu chung 99
7.2. Phần tử mảng và các chỉ số mảng 99
7.3. Cách khai báo một mảng 99
7.3.1. Khái niệm 100
7.3.2. Khai báo mảng một chiều 100
7.3.3. Mảng nhiều chiều 102
7.4. Sử dụng mảng một chiều và mảng hai chiều 105
7.4.1. Sử dụng mảng một chiều 105
7.4.2. Sử dụng mảng hai chiều để cộng ma trận 108
7.5. Con trỏ và mảng 112
7.5.1. Mối quan hệ giữa con trỏ và mảng 112
7.5.2. Con trỏ và mảng một chiều 113
7.5.3. Con trỏ và mảng nhiều chiều 115
7.5.4. Một số thí dụ 116
7.6. Các biến và hằng kiểu chuỗi 119
7.7. Các thao tác nhập/xuất chuỗi 119
7.7.1. Thao tác nhập/xuất chuỗi đơn giản 119
7.7.2. Thao tác nhập/xuất chuỗi có định dạng 120
7.8. Sử dụng các hàm về chuỗi 121
7.8.1. Hàm strcat() 121
7.8.2. Hàm strcmp() 122
7.8.3. Hàm strchr() 122
7.8.4. Hàm strcpy() 122
7.8.5. Hàm strlen() 123
7.8.6. Sắp xếp chuỗi sử dụng các hàm trong thư viện 123
7.8.7. Sử dụng hàm để chuyển một mảng ký tự về chữ hoa 125
Chương 8 CẤU TRÚC VÀ DANH SÁCH LIÊN KẾT 127
8.1. Giới thiệu chung 127
8.2. Kiểu cấu trúc 127
8.2.1. Khái niệm cấu trúc 127

8.2.2. Định nghĩa kiểu cấu trúc 128
8.2.3. Khai báo theo kiểu cấu trúc đã định nghĩa 129
8.2.4. Truy xuất đến các thành phần của cấu trúc 130
8.2.5. Cấu trúc lồng nhau 131
8.3. Mảng cấu trúc 132
8.3.1. Khai báo biến mảng cấu trúc 132
8.3.2. Truy xuất các phần tử mảng cấu trúc 132
8.3.3. Khởi tạo giá trị cho các phần tử của mảng cấu trúc 132
8.4. Con trỏ cấu trúc 132
8.4.1. Khai báo con trỏ cấu trúc 132
8.4.2. Sử dụng con trỏ cấu trúc 133
8.4.3. Con trỏ và mảng cấu trúc 134
8.5. Danh sách liên kết 135
8.5.1. Khái niệm cấu trúc tự trỏ 135
8.5.2. Khái niệm danh sách liên kết 135
8.5.3. Các phép toán trên danh sách liên kết 136
8.6. Một số kiểu dữ liệu tự tạo khác 140
8.6.1. Kiểu liệt kê (enum) 140
8.6.2. Kiểu hợp (union) 140
Chương 9 TỆP TIN 142
9.1. Giới thiệu chung 142
9.2. Một số khái niệm về tệp tin 142
9.3. Thao tác trên tệp tin văn bản 143
9.3.1. Hàm đóng/mở file 143
9.3.2. Một số hàm có chức năng điều khiển 145
9.3.3. Các hàm ghi dữ liệu 146
9.3.4. Các hàm đọc dữ liệu từ file 147
9.4. Thao tác trên tệp tin nhị phân 149
9.4.1. Các hàm ghi dữ liệu 149
9.4.2. Các hàm đọc dữ liệu 149
9.4.3. Di chuyển con trỏ tập tin - Hàm fseek() 150

Chương 1   KHÁI NIỆM VỀ HỆ THỐNG MÁY TÍNH

1.1. Lịch sử phát triển của máy tính điện tử

Lịch sử máy tính điện tử gắn liền với chặng đường phát triển của IBM-PC. Do đó ta có thể tóm tắt quá trình phát triển của máy tính điện tử như sau:

1979-1980: IBM cho ra đời máy Datamaster dùng vi xử lý 16 bit 8086 của Intel

1980: Đưa ra khái niệm: Personal Computer (PC). Chiếc IBM-PC đầu tiên dùng vi xử lý 8bit 8085 của Intel.

1981-1982: Dù Intel có vi xử lý 16bit nhưng giá thành còn cao, Để đáp ứng thị trường máy rẻ tiền, Intel đưa ra vi xử lý 8 bit 8088 mà trong nó là vi mạch 16bit 8086.

1984: Khi vi xử lý 16bit đã quen thuộc thị trường, Intel đưa ra vi xử lý 80286, là vi xử lý 16bit hoàn thiện, có thêm 4bit bus địa chỉ, quản lý 16MB bộ nhớ.

1987: Thế hệ PC mới ra đời với vi xử lý 80386. Bắt đầu từ đây IBM công khai cấu tạo máy và nội dung chương trình hệ điều hành vào ra cơ sở (BIOS), điều này giúp các hãng khác có thể sản xuất các máy tính tương thích và các bản mạch cắm tương thích khiến cấu trúc IBM-PC trở thành một cấu trúc chuẩn công nghiệp.

1990: 80486 ra đời với nhiều chức năng hơn, cụ thể là 8 Kbyte bộ nhớ đệm mã lệnh (code cache) và một bộ đồng xử lý toán học. Tần số làm việc đặc trưng của máy vi tính trong thời kỳ này là 66MHz.

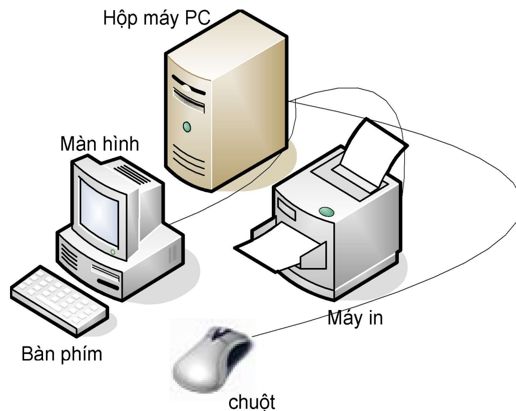
1993: Vi xử lý Pentium đầu tiên ra đời mở ra một kỷ nguyên mới với 64bit bus dữ liệu, 32bit bus địa chỉ, 8KB bộ đệm dữ liệu, 8KB bộ đệm mã lệnh. Bộ đồng xử lý toán học của Pentium làm việc nhanh gấp 10 lần so với 80486. Khi này các nhà sản xuất phần cứng lớn thoả thuận một chuẩn khe cắm mới PCI-bus (Peripheral Components Interconnect), và do đó bản mạch chính máy vi tính cá nhân chỉ còn lại vài vi mạch, tất cả các vi mạch ngoại vi của cấu trúc IBM-PC cũng như vi mạch điều khiển PCI được tích hợp vào một vi mạch duy nhất, có tên là PCI-chipset.

1995: Khả năng đa môi trường (multimedia) của máy vi tính cá nhân càng ngày càng hoàn thiện khi Pentium MMX , Pentium Pro, Pentium II lần lượt ra đời. Tần số đồng hồ cao nhất 300 MHz. Một chuẩn giao diện ngoại vi mới ra đời từ sự thoả thuận từ nhiều hãng lớn là bus tuần tự đa dạng USB (Universal Serial Bus).

Từ năm **2000**: Một cấu trúc vi xử lý 64 bit ra đời. Intel cho ra đời nhiều vi mạch tổng hợp thích hợp với vi xử lý của chính hãng. Chipset đảm nhiệm hầu hết các chức năng điều khiển trên máy và có bộ điều khiển hiển thị cấy ở bên trong. Thị trường **máy tính cá nhân** cũng như thị trường vi xử lý và vi mạch tổng hợp được chia thành nhiều phần đáp ứng nhu cầu đa dạng trong xã hội.

1.2. Khái niệm hệ thống máy tính

Máy tính (computer) là thiết bị điện tử thực hiện công việc nhận thông tin vào, xử lý thông tin theo chương trình nhớ sẵn bên trong bộ nhớ máy tính sau đó cho kết quả thông tin đầu ra.



Hình 1.1 Hệ thống máy tính điển hình

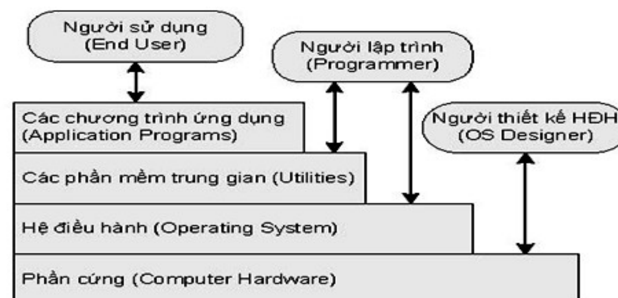
Chương trình (program) là dãy các lệnh nằm trong bộ nhớ để yêu cầu máy tính thực hiện công việc cụ thể. Như vậy, một máy tính bất kỳ đều làm việc theo chương trình.

Phần cứng (hardware) bao gồm các đối tượng hữu hình như các vi mạch (IC), các bảng mạch in, cáp nối, nguồn điện, bộ nhớ, máy đọc bìa…

Phần mềm (software) bao gồm các thuật toán và các biểu diễn cho máy tính của chúng, đó chính là các chương trình (program). Cái cơ bản nhất của phần mềm là tập các chỉ thị tạo nên chương trình chứ không phải là môi trường vật lý được sử dụng để ghi chương trình. Ví dụ hệ điều hành DOS, Window, Unix …

Phần sụn là trung gian giữa phần cứng và phần mềm, hay nói cách khác, phần sụn chính là phần mềm được nhúng vào các phần cứng trong quá trình chế tạo các phần cứng này. Phần sụn được sử dụng đối với các chương trình hiếm khi hoặc không bao giờ cần thay đổi hay trong trường hợp các chương trình không được phép bị mất khi tắt điện.

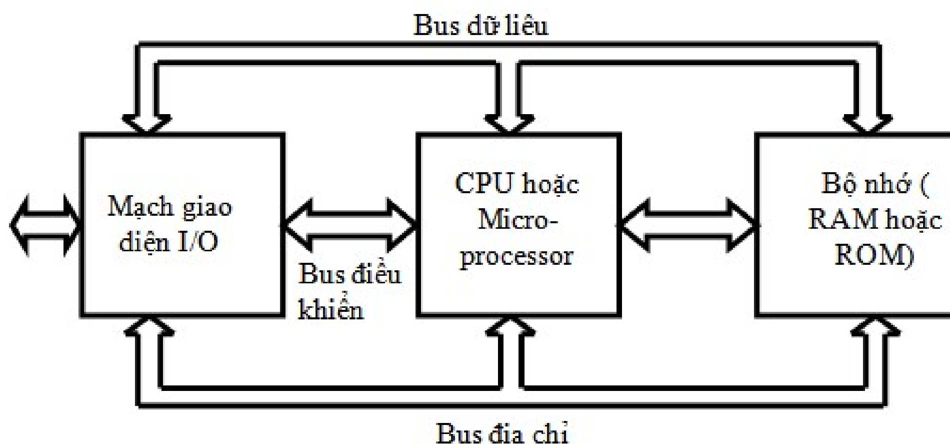
Hình sau mô tả mô hình phân lớp của máy tính sẽ giúp bạn đọc hiểu hơn về các khái niệm cơ bản nêu trên.



Hình 1.2 Mô hình phân lớp của máy tính

1.3. Cấu trúc cơ bản của hệ thống máy tính

Các phần chính của 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. Các phần tử đó được nối với nhau qua đường bus chính: bus địa chỉ, bus điều khiển và bus dữ liệu. Hình 1.3 mô tả một hệ thống cơ bản. Các các thiết bị ngoại vi như bàn phím, màn hình, ổ đĩa, có thể nối trực tiếp với các bus: dữ liệu, địa chỉ và điều khiển, hoặc ghép với các mạch vào ra.



Hình 1.3 Sơ đồ cấu trúc một hệ thống máy tính đơn giản

Khối xử lí trung tâm CPU (**C**entral **P**rocessing **U**nit): dùng để thu thập và cho chạy các lệnh. Bên trong CPU gồm các mạch điều khiển logic, mạch tính số học và logic,.. Nếu CPU được xây dựng trên một hoặc vài vi mạch, thường được đóng trong một chip, thì nó được gọi là bộ vi xử lí µP (microprocessor). Một máy tính dùng µP làm bộ xử lí thì được gọi là máy vi tính (microcomputer) hay máy tính cá nhân PC (**P**ersonal **C**omputer).

Bộ nhớ: dùng để lưu trữ các lệnh và dữ liệu cho bộ xử lí. Nó bao gồm hai loại: bộ nhớ trong (được tạo bởi các vi mạch nhớ bán dẫn) và bộ nhớ ngoài (được tạo bởi các môi trường nhớ khác như đĩa từ, đĩa quang). Bộ nhớ thường được chia thành từng ô nhớ nhỏ như từ hay byte (1 byte=8 bit, 1 từ=2 byte). Mỗi ô nhớ đó cũng như một thiết bị vào/ra được gán cho một địa chỉ (address) để CPU có thể định vị khi cần đọc hay viết dữ liệu lên nó.

Các thiết bị ngoại vi: gồm các thiết bị vào/ra (I/O: input/output) dùng để nhập hoặc xuất các dữ liệu. Bàn phím, chuột, máy quét,…thuộc loại thiết bị vào. Màn hình, máy in,…thuộc loại thiết bị ra.Các ổ đĩa ở bộ nhớ ngoài có thể được coi vừa là thiết bị vào vừa là thiết bị ra. Các thiết bị ngoại vi này liên hệ với CPU qua các mạch ghép nối vào/ra (I/O interface).Mạch này cho phép nối hai bộ phận độc lập nhằm làm cho chúng có thể tương hợp và thông tin được với nhau.

Bus hệ thống: là một tập hợp các đường dây mà qua đó CPU có thể liên kết với các bộ phận khác.

1.4. Biểu diễn thông tin trong máy tính điện tử

1.4.1. Các hệ đếm

Trong cuộc sống hàng ngày chúng ta thường dùng các hệ cơ số 10 để biểu diễn các giá trị số. Tuy nhiên đối với máy tính lại khác, do máy tính được cấu tạo từ các mạch điện tử và các mạch này chỉ có hai có hai trạng thái có điện và không có điện. Do đó để biễu diễn một giá trị số trong máy tính người ta sử dụng hệ đếm cơ số hai hay hệ đếm nhị phân (Binary). Trong hệ đếm này chỉ tồn tại hai chữ số 1 và 0 tương ứng với hai trạng thái có điện và không có điện của các mạch điện tử.

Nếu dùng hệ cơ số hai để biểu diễn các số có giá trị lớn sẽ gặp bất tiện là số hệ hai thu được quá dài, thí dụ: 55 = 110111 hay 253 = 11111101.

Do đó, để viết kết quả biễu diễn các số cho gọn lại người ta sử dụng các các hệ đếm khác như hệ cơ số 16 (thập lục phân hay hexa) và hệ cơ số 8 (bát phân). Bảng sau đây trình bày một số hệ đếm cơ bản:

Bảng 1.1 Các hệ đếm cơ bản

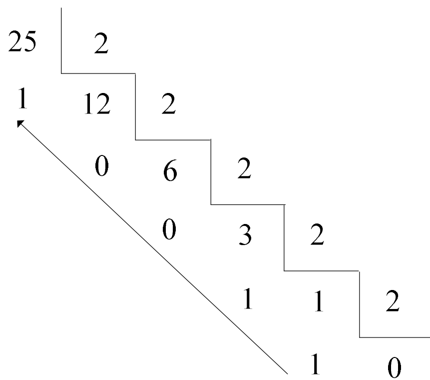
Các hệ đếm	Cơ số	Số ký số và ký tự	Dạng ký số và ký tự
Nhị phân (Binary)	2	2	0,1
Bát phân (Octal)	8	8	0,1,2,3,4,5,6,7
Thập phân (Decimal)	10	10	0,1,2,3,4,5,6,7,8,9
Thập lục phân (Hexadecimal)	16	16	0,1,2,3,4,5,6,7,8,9 A,B,C,D,E,F

1.4.2. Chuyển đổi từ hệ thập phân sang các hệ khác

1.4.2.1. Chuyển đổi từ hệ thập phân sang hệ nhị phân

Quy tắc: Để thực hiện việc đổi từ hệ thập phân sang nhị phân, ta áp dụng phương pháp chia lặp như sau: Lấy số thập phân chia cho cơ số để thu được một thương số và số dư. Số dư được ghi lại để làm một thành tố của số nhị phân. Sau đó, số thương lại được chia cho cơ số một lần nữa để có thương số thứ 2 và số dư thứ 2. Số dư thứ hai là con số nhị phân thứ hai. Quá trình tiếp diễn cho đến khi số thương bằng 0. Sau đó viết các số dư theo chiều từ phép chia cuối cùng đến phép chia đầu tiên.

Thí dụ: Chuyển số 25 trong hệ 10 sang hệ nhị phân:



Kết quả là: $(25)_{10} = (11001)_2$

Ngoài ra, cần xét đến quy tắc chuyển đổi từ một số thập phân sang số nhị phân như sau: Lấy số cần đổi nhân với 2, tích gồm phần nguyên và phần lẻ. Lấy phần lẻ nhân tiếp với 2 cho đến khi nào tích thu được bằng 1 thì dừng lại. Chọn riêng phần nguyên của các tích thu được và viết theo thứ tự từ phép nhân đầu tiên đến phép nhân cuối cùng. Thí dụ: chuyển số thập phân 0.125 sang hệ nhị phân

0.125 x 2 = 0 .25

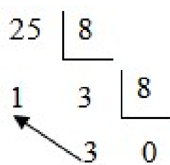
0.25 x 2 = 0 .50 => $(0.125)_{10} = (0.001)_2$

0.50 x 2 = 1 .0

1.4.2.2. Chuyển đổi từ hệ thập phân sang hệ bát phân

Quy tắc: Quy tắc chuyển đổi từ hệ thập phân sang hệ bát phân cũng thực hiện tương tự như với hệ nhị phân. Ta cũng thực hiện phép chia số thập phân cho cơ số (cơ số 8) được thương và số dư. Số dư sẽ là thành phần của số bát phân. Tiếp tục lấy thương chia cho cơ số và thực hiện cho tới khi nào thương bằng 0. Số bát phân sẽ là tập dãy các số dư được viết theo chiều từ phép chia cuối đến phép chia đầu.

Thí dụ: chuyển số 25 sang hệ bát phân:



Kết quả: $(25)_{10} = (31)_8$

1.4.2.3. Chuyển đổi từ hệ thập phân sang hệ thập lục phân

Quy tắc: Việc chuyển từ hệ thập phân sang hệ hexa (16) cũng được thực hiện hoàn toàn tương tự như quy tắc chuyển từ hệ thập phân sang hệ bát phân và nhị phân. Tức là ta cũng thực hiện các phép chia liên tiếp số thập phân đó và thương của nó cho cơ số 16 cho đến khi thương bằng 0. Và kết quả hệ 16 là dãy số dư được viết ngược từ phép chia cuối đến phép chia đầu.

Tuy nhiên, cần lưu ý rằng trong phép chia cho 16, các số dư là từ 0 đến 15. Để thuận tiện biểu diễn, từ số 10 đến 15 sẽ được thay bằng các ký tự chữ cái từ a đến f (hoặc A đến F).

Thí dụ: Chuyển số 45 sang hệ Hexa:

```
        45 | 16
           -----
D <= 13    2  | 16
              -----
            2    0
```

Kết quả: $(45)_{10} = (2D)_{16}$

1.4.3. Chuyển từ hệ nhị phân sang các hệ khác

1.4.3.1. Chuyển từ hệ nhị phân sang hệ thập phân

Để chuyển từ hệ nhị phân sang thập phân ta tính tổng các tích số giữa các hệ số với các trọng số 2^i tại từng vị trí thứ i.

Thí dụ:

$(1101)_2 = 1.\ 2^3 + 1.2^2 + 0.\ 2^1 + 1.\ 2^0$

$= 8 + 4 + 0 + 1 = (13)_{10}$

$(10110.11)_2 = 1.2^4 + 0.2^3 + 1.2^2 + 1.2^1 + 0.2^0 + 1.2^{-1} + 1.2^{-2}$

$= 16 + 0 + 4 + 2 + 0 + 0,5 + 0,25 = (22,75)_{10}$

1.4.3.2. Chuyển từ hệ nhị phân sang hệ bát phân

Để chuyển từ hệ nhị phân sang hệ bát phân, ta nhóm từng nhóm 3 bit, bắt đầu từ phải sang trái. Sau đó tính giá trị của từng nhóm đó (cho giá trị từ 0 đến 7),ghi các giá trị liền nhau theo trình tự cũ, ta sẽ có giá trị bát phân tương ứng.

Thí dụ:

Hệ nhị phân: 11 001 101 011

Hệ bát phân: 3 1 5 3

=> $(11001101011)_2 = (3153)_8$

1.4.3.3. Chuyển từ hệ nhị phân sang hệ hexa

Để chuyển từ hệ nhị phân sang hệ hexa, ta cũng thực hiện nhóm các bít thành các nhóm 4 bit, bắt đầu từ phải sang trái. Tính giá trị của từng nhóm bít đó (cho giá trị từ 0 đến 15), ghi các giá trị lần lượt theo thứ tự cũ ta được số hexa tương ứng.

Thí dụ:

Hệ nhị phân: 110 1001 1101 0000

Hệ Hexa: 6 9 D 0

=> $(110100111010000)_2 = (69D0)_{16}$

1.4.4. Chuyển từ hệ bát phân và hexa sang hệ thập phân và nhị phân

1.4.4.1. Chuyển từ hệ bát phân và hệ hexa sang hệ thập phân

Quy tắc để chuyển từ hệ bát phân và hệ hexa sang hệ thập phân là tương tự nhau và tương tự với quy tắc chuyển từ hệ nhị phân sang hệ thập phân. Giá trị thập phân sẽ là tổng của các tích số giữa các hệ số với trọng số 8^i (với hệ bát phân) hoặc 16^i (với hệ hexa) ứng với từng vị trí thứ i.

Thí dụ:

$$(407)_8 = 4.\ 8^2 + 0.\ 8^1 + 7.\ 8^0$$
$$= 256 + 0 + 7$$
$$= (263)_{10}$$
$$(4CA0)_{16} = 4.\ 16^3 + 12.\ 16^2 + 10.\ 16^1 + 0.\ 16^0$$
$$= 16384 + 3072 + 160 + 0$$
$$= 19616$$

1.4.4.2. Chuyển từ hệ bát phân và hexa sang hệ nhị phân

Quy tắc chuyển từ hệ bát phân và hệ hexa sang hệ nhị phân đơn giản là cách tính ngược lại so với cách chuyển từ hệ nhị phân sang hai hệ này. Tức là lấy từng chữ số trong số bát phân (hoặc số hexa) chuyển sang giá trị nhị phân tương ứng là một nhóm 3 bit (hoặc nhóm 4 bit). Giữ nguyên thứ tự các chữ số, ta được giá trị số nhị phân tương ứng.

Thí dụ:

$$(407)_8 = (100\ \ 000\ 111)_2$$
$$= (100000111)_2$$
$$(20A)_{16} = (0010\ \ 0000\ \ 1010)_2$$
$$= (1000001010)_2$$

1.5. Các phép tính số học trên hệ nhị phân

Phép tính số học dùng trong hệ nhị phân cũng tương tự như các phép tính được áp dụng trong các hệ khác. Các phép cộng, trừ, nhân và chia cũng được áp dụng với các giá trị số nhị phân.

1.5.1. Phép cộng

Phép tính đơn giản nhất trong hệ nhị phân là tính cộng. Cộng hai đơn vị trong hệ nhị phân được làm như sau:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10 (nhớ 1 lên hàng thứ 2)

Cộng hai số "1" với nhau tạo nên giá trị "10", tương đương với giá trị 2 trong hệ thập phân. Số "1" sẽ được nhớ vào phép cộng của các bit liền kề bên trái.

Thí dụ: thực hiện phép cộng:

11001011

+ 1011101

= 100101000

Tương tự, nếu ta cộng 3 bit "1" với nhau ta sẽ được giá trị là "11" và số một có trọng số cao hơn(phía bên trái) sẽ được nhớ vào phép cộng của các bit tiếp theo.

1.5.2. Phép trừ

Phép tính trừ theo quy tắc tương tự như phép cộng:

0 − 0 = 0

0 − 1 = −1 (mượn)

1 − 0 = 1

1 − 1 = 0

Phép trừ thực chất cũng là phép cộng với một số âm của số đó.

Thí dụ:

11011010

- 1001110

= 10001100

Phép toán trên tương đương với phép toán trong hệ thập phân

218 -78 = 140

Ta có thể hiểu phép toán trên là phép cộng:

218 + (-78) =140

Việc biểu diễn một số âm trong máy tính được thực hiện bởi phương pháp bù 2. Theo phương pháp này, bit cực trái (là bit nằm bên trái cùng của byte) được sử dụng làm bit dấu (*sign bit* - là bit tượng trưng cho dấu của số) với quy ước: nếu bit dấu là 0 thì số là số dương, còn nếu nó là 1 thì số là số âm. Ngoài bit dấu này ra, các bit còn lại được dùng để diễu diễn độ lớn của số.

Quy tắc biểu diễn số nguyên âm theo phương pháp bù 2 như sau:

- Biểu diễn phần nguyên dương dưới dạng nhị phân.
- Đảo tất cả các bit nhận được trong bước 1
- Cộng thêm 1 vào kết quả của bước 2
- Bit ngoài cùng bên trái là bit 1 (bit dấu của số âm)

Thí dụ: Biểu diễn số -7 theo phương pháp bù 2

Bước 1: Biểu diễn số 7 dưới dạng nhị phân: 0000 0111

Bước 2: Đảo tất cả các bít trong dãy bit trên: 1111 1000

Bước 3: Cộng thêm 1 vào kết quả của bước 2:1111 1001.

Như vậy kết quả biểu diễn số -7 trong máy tính là: 1111 1001.

Như vậy, trong thí dụ về phép trừ 218 cho 78, ta có thể biểu diễn số -78 theo phương pháp bù 2, sau đó thực hiện phép cộng 218 (dạng nhị phân) với -78(dạng nhị phân).

Tuy nhiên, cần lưu ý rằng khi thực hiện phép tính cộng với số âm biểu diễn theo phương pháp bù 2, ta thực hiện như phép cộng nhị phân bình thường nhưng trong trường hợp khi đã thực hiện phép cộng đến bit cực trái mà vẫn phát sinh bit nhớ thì ta bỏ bit nhớ này đi.

Quay lại thí dụ phép trừ trên, ta biểu diễn số -78 theo phương pháp bù 2:

Bước 1: $(78)_{10} = (0100\ 1110)_2$

Bước 2: đảo bít: 1011 0001

Bước 3: Cộng thêm 1: 1011 0010

Như vậy số -78 có dạng nhị phân là: 1011 0010. Giờ ta sẽ thực hiện phép cộng nhị phân của số 218 và -78:

```
        11011010
   -    10110010
   --------------------
   =    10001100       (Đã bỏ bít nhớ ngoài cùng bên trái)
```

1.5.3. Phép nhân

Phép tính nhân trong hệ nhị phân cũng tương tự như phương pháp làm trong hệ thập phân.

Thí dụ:

```
        10011       = (19)10
     x      101     = (5)10
   ------------------
          10011
     +  00000
     + 10011
  -------------------
     = 1011111   = (95)10
```

1.5.4. Phép chia

Phép chia nhị phân cũng tương tự như phép chia thập phân:

Thí dụ:

```
        110010 |101
   -    101     1010
     ----------
        0010
```

00

101

\- 101

00

Như vậy kết quả của phép chia 110010 / 101 (tương ứng là 50 / 5 trong hệ 10) là 1010 (tương ứng giá trị 10 trong hệ 10).

1.6. Khái niệm thuật toán

Thuật toán là tập hợp các quy tắc logic chặt chẽ, xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn lần thực hiện ta sẽ thu được kết quả mong đợi.

Việc nghiên cứu về thuật toán có vai trò rất quan trọng trong khoa học máy tính vì mọi vấn đề, mọi bài toán để thực hiện được trên máy tính điện tử đều cần phải có một thuật toán cho nó. Trong máy tính điện tử, mỗi thuật toán thường được thể hiện bởi một thủ tục gồm một số hữu hạn các câu lệnh để đạt được lời giải cho bài toán đó.

Cấu trúc dữ liệu và thuật toán có mối liên hệ gắn bó chặt chẽ với nhau trong một chương trình máy tính. Hay có thể nói, chương trình chính là sự kết hợp của thuật toán với cấu trúc dữ liệu để giải quyết một bài toán.

Thông thường trong mỗi ngôn ngữ lập trình đều cung cấp sẵn các cấu trúc dữ liệu xác định. Ngoài ta, nhiều ngôn ngữ lập trình cũng cho phép người sử dụng tự tạo ra các kiểu dữ liệu mở rộng để có thể đáp ứng các bài toán phức tạp.

Chương 2: GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH C

2.1. Giới thiệu

2.1.1. Lịch sử ngôn ngữ C

Dennis Ritchie phát triển ngôn ngữ C 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 một phương châm dễ dàng trong việc bảo trì, hiệu quả và dễ di chuyển (portable).

C có nguồn gốc từ ngôn ngữ BCPL do Martin Richards phát triển. BCPL sau đó đã được Ken Thompson phát triển thành ngôn ngữ B, và sau đó là sự kế thừa phát triển lên ngôn ngữ C.

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 hoa kỳ) được thành lập để chuẩn hóa ngôn ngữ này và xây dựng lên tiêu chuẩn ANSI-C. Hiện nay, hầu hết các sản phẩm C đều tuân theo tiêu chuẩn này.

2.1.2. Đặc điểm của ngôn ngữ C

C là một ngôn ngữ lập trình tương đối nhỏ gọn vận hành gần với phần cứng và nó giống với ngôn ngữ Assembly hơn hầu hết các ngôn ngữ bậc cao. Hơn thế, sự khác nhau quan trọng giữa nó với ngôn ngữ bậc thấp như Assembly là việc mã C có thể được dịch và thi hành trong hầu hết các máy tính, hơn hẳn các ngôn ngữ hiện tại trong khi đó thì Assembler chỉ có thể chạy trong một số máy tính đặc biệt. Vì lý do này C được xem là ngôn ngữ bậc trung.

Khi thực thi chương trình C cũng rất nhanh như Assembly, nên đáp ứng được yêu cầu thời gian thực của một số chương trình ứng dụng. Lập trình viên có thể tạo ra và bảo trì thư viện hàm mà chúng sẽ được tái sử dụng cho chương trình khác. Do đó, những dự án lớn có thể được quản lý dễ dàng mà tốn rất ít công sức.

Ngoài ra, ngôn ngữ C có một thư viện hàm khổng lồ, chương trình ngắn gọn, dễ học nên cũng càng ngày càng được ứng dụng mạnh mẽ.

2.1.3. Ứng dụng của ngôn ngữ C

Ban đầu ngôn ngữ C được ứng dụng chủ yếu trong lập trình hệ thống. Các chương trình dịch, các hệ điều hành và các tiện ích đều được viết bằng ngôn ngữ C. Ngôn ngữ này tỏ ra ưu việt hơn trong lĩnh vực này và ngày nay đã được ứng dụng trong nhiều lĩnh vực và các hệ thống máy tính khác nhau. Trong lĩnh vực kỹ thuật điện tử, C được sử dụng để lập trình vi điều khiển, lập trình các hệ thống nhúng, lập trình hệ thống,…

2.2. Tập ký tự

Trong ngôn ngữ lập trình C, tập ký tự được sử dụng bao gồm:

26 chữ cái hoa: A B C… Z

26 chữ cái thường: a b c … z

10 chữ số: 0 1 2 … 9

Các kí tự toán học như: + - * / = ( )

Các ký hiệu đặc biệt như: . , : ; _ {} ? ! \ & | % # $ > < ~ ‘ ’ “ ” …

Dấu cách ‘ ’ là khoảng trống được dùng để phân cách giữa các *từ* trong chương trình và không có ý nghĩa gì trong mỗi câu lệnh. Giữa các từ có thể có một hay nhiều dấu cách ‘ ’.

2.3. Định danh

2.3.1. Từ khóa

Tất cả các ngôn ngữ lập trình đều dành một số từ nhất định cho mục đích riêng. Những từ này có một ý nghĩa đặc biệt trong ngữ cảnh của từng ngôn ngữ, và được xem là “từ khóa”. Từ khóa được dùng để khai báo, định nghĩa các kiểu dữ liệu, định nghĩa các toán tử, các hàm và viết các câu lệnh…

Khác với các ngôn ngữ khác, các từ khóa sử dụng trong C phải được viết bằng chữ thường. Dưới đây là một số từ khóa thông dụng của ngôn ngữ C:

break	**case**	**char**	**continue**	**do**
else	**float**	**for**	**goto**	**if**
int	**long**	**return**	**short**	**sizeof**
struct	**switch**	**typedef**	**unsigned**	**while**

Mỗi từ khóa có một cú pháp sử dụng riêng mà khi viết chương trình ta cần phải tuân theo cú pháp đó. Một số từ khóa thông dụng sẽ được giới thiệu kỹ lần lượt trong các phần sau.

Cần phải lưu ý rằng khi ta đặt tên biến, tên hằng, tên nhãn … thì không được đặt trùng với từ khóa. Quy tắc đặt tên sẽ được trình bày rõ trong phần sau.

2.3.2. Quy tắc đặt tên

Tên là một khái niệm rất quan trọng trong bất kỳ ngôn ngữ lập trình nào, nó dùng để xác định các đối tượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn,…v.v. Một số quy tắc đặt tên sau sẽ giúp các bạn tránh nhầm lẫn và sai sót khi lập trình:

- Tên là một dãy kí tự: chữ, số và dấu gạch nối. Ký tự đầu tiên phải là chữ hoặc dấu gạch nối.
- Tên không được trùng với từ khóa.
- Độ dài cực đại của tên mặc định là 32.
- Trong C phân biệt chữ hoa và chữ thường. Chữ hoa thường dùng để đặt tên cho các hằng, chữ thường thường được dùng để đặt tên cho hầu hết các đối tượng như biến, mảng, hàm, cấu trúc (tuy nhiên điều này không bắt buộc).

- Trong cùng một phạm vi và thời gian tồn tại, không được đặt hai tên trùng nhau. Có nghĩa là trong cùng khối lệnh, không được khai báo hai tên trùng nhau.
- Tên nên đặt gợi nhớ, dễ hiểu, phản ánh được tính chất của đối tượng được đặt tên.

Một số ví dụ về cách đặt tên như sau:

Tên hợp lệ:

Dien_tro1,

_tanso,

r1, r2.

Tên không hợp lệ:

3r: kí tự đầu tiên là số

while: trùng với từ khóa

dien-tro: Có sử dụng ký tự dấu trừ '-'

Tong tro: Có sử dụng khoảng cách.

2.4. Các kiểu dữ liệu

2.4.1. Kiểu dữ liệu cơ bản

Trong C định nghĩa bốn kiểu dữ liệu cơ bản là: int, float, char và double. Các kiểu dữ liệu này được mô tả chi tiết trong bảng dưới đây:

Bảng 2.1 Các kiểu dữ liệu cơ bản

Kiểu dữ liệu	**Mô tả ý nghĩa**	**Bộ nhớ (byte)**	**Phạm vi**
int	Kiểu số nguyên có dấu	2	-32768 đến 32767
float	Kiểu số thực dấu phảy động đơn	4	3.4E-38 đến 3.4E38
double	Kiểu số thực dấu phảy động kép	8	1.7E-308 đến 1.7E308
char	Kiểu ký tự đơn	1	-128 đến 127

+) Một số chú ý:

- Kiểu số thực **float** có độ chính xác là 6 chữ số sau dấu thập phân, còn kiểu số thực **double** có độ chính xác là 15 chữ số sau dấu thập phân.
- Khi biểu diễn số thập phân, thì phần nguyên hoặc phần thập phân có thể vắng mặt, nhưng không thể thiếu dấu '.' phân cách. Ví dụ như: 3.05, 15., .15
- Kiểu **char** chiếm 1 byte bộ nhớ để biểu diễn một ký tự mã ASCII. Ví dụ:

Ký tự	Mã ASCII
0	048
1	049
A	065
B	066
A	097

Thực chất kiểu **char** cũng là một số nguyên không dấu nằm trong khoảng từ 0 đến 255. Điều này sẽ được làm rõ trong các phần sau.

2.4.2. Kiểu mở rộng

Mỗi kiểu dữ liệu cơ bản kể trên lại có thể kết hợp với một hoặc nhiều tiền tố như : **long, short** và **unsigned** để thay đổi phạm vi biểu diễn của mỗi kiểu dữ liệu đó. Bảng sau mô tả một số kiểu mở rộng thông dụng và phạm vi biểu diễn của chúng:

Bảng 2.2 Các kiểu dữ liệu mở rộng

Kiểu dữ liệu	**Bộ nhớ (byte)**	**Phạm vi biểu diễn**
long int hoặc long	4	-2.147.483.648 đến 2.147.483.647
unsigned int hoặc unsigned	2	0 đến 65535
short int	2	-32768 đến 32767
unsigned long int hoặc unsigned long	4	0 đến 4294967295
unsigned char	1	0 đến 255
long double	10	3.4E-4932 đến 1.1E+4932

2.4.3. Địnhnghĩa kiểu dữ liệu bằng typedef

Từ khóa **typedef** dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ dùng để khai báo dữ liệu sau này. Nên chọn một tên vừa ngắn gọn vừa dễ nhớ.

Cú pháp định nghĩa kiểu bằng **typedef** như sau:

typedef kiểu_dữ_liệu Tên_biến;

Sau đó, tên biến sẽ có ý nghĩa như kiểu dữ liệu và có thể dùng tên biến vừa đặt để khai báo cho các biến mới với kiểu dữ liệu được sử dụng ở trên.

Thí dụ: Câu lệnh sau:

typedef int mang_dientro;

tức là sẽ đặt tên kiểu **int** là mang_dientro. Sau đó có thể dùng mang_dientro để khai báo cho các biến, mảng nguyên như sau:

mang_dien_tro a[10];

// khai báo một mảng kiểu nguyên a (mảng các điện trở) gồm 10 phần tử.

2.5. Biến

2.5.1. Khái niệm biến

Biến là vùng nhớ được cấp phát dùng để lưu trữ giá trị cho một kiểu dữ liệu nào đó tại một thời điểm nhất định. Biến là đại lượng thay đổi khi thực hiện chương trình và nó được truy xuất thông qua tên đã được khai báo cho nó.

2.5.2. Khai báo biến

Cấu trúc khai báo của biến như sau:

Kiểu_dữ_liệu danh_sách_tên_biến;

Trong đó, **Kiểu_dữ_liệu** xác định kiểu của dữ liệu mà biến lưu trữ. Tên biến là một định danh được gán cho vùng nhớ chứa biến và dùng để truy xuất giá trị của biến. Tên biến phải tuân theo các quy tắc đặt tên đã đưa trong phần trên.

Trong C, một khai báo cho phép khai báo nhiều biến thuộc cùng một kiểu dữ liệu và các biến phân cách nhau bởi dấu phẩy (,). Cuối cùng, kết thúc khai báo bởi dấu chấm phẩy (;).

+) ***Lưu ý:*** Trong câu lệnh khai báo biến, ta cũng có thể khởi tạo giá trị ban đầu cho biến sử dụng toán tử gán (=).

Ví dụ: Một số khai báo sau:

char ky_tu; //khai báo biến kiểu char có tên là ky_tu.

long van_toc_as = 300000000;

//Vừa khai báo vừa khởi tạo giá trị cho biến kiểu long.

float f, pi = 3.1416 ; /* khai báo 2 biến kiểu float trong đó có một biến được khởi tạo giá trị */

2.5.3. Vị trí của khai báo biến

Vị trí khai báo biến là một đặc trưng rất quan trọng của biến vì nó xác định phạm vi sử dụng và thời gian tồn tại của biến trong chương trình. Do đó, bạn cần tuân theo những quy tắc về vị trí đặt biến sau:

2.5.3.1. Khai báo biến ngoài

Biến được khai báo ở bên ngoài tất cả các hàm. Khi đó biến còn được gọi là *biến ngoài* hay *biến toàn cục*. Các biến này sẽ tác động tới toàn bộ chương trình, tức là biến có thể được truy xuất bởi bất cứ hàm nào trong chương trình và thời gian tồn tại (biến được cấp phát bộ nhớ) là suốt thời gian chương trình làm việc.

Ví dụ:

```
    int f;          // khai báo biến toàn cục
    float T;   // khai báo biến toàn cục
main( )
    {
        T = 1/f;
        …
    }
```

2.5.3.2. Khai báo biến trong

Biến được khai báo bên trong một khối lệnh (có nghĩa là bên trong hàm) thì biến đó được gọi là *biến trong* hay biến *cục bộ.*

Phạm vi hoạt động của biến là bên trong khối lệnh mà nó được khai báo. Còn thời gian tồn tại của biến là từ khi máy bắt đầu làm việc với khối lệnh cho tới khi ra khỏi khối lệnh đó.

Do đó, sau khi máy kết thúc làm việc với khối lệnh mà biến được khai báo trong đó, vùng nhớ cấp phát cho biến sẽ mất đi nên các giá trị của biến cũng mất đi.

Ví dụ đoạn lệnh sau có một số khai báo biến trong:

```
    float dien_ap, dong_dien, tro_khang; //khai báo biến ngoài
main()
    {
        float  R, C, L, ; //khai báo biến trong
        char bang_mau[15]; //khai báo biến trong
        …
    }
```

+) *Lưu ý:*

- Biến luôn phải được khai báo đầu tiên trong khối lệnh, ngay sau dấu { và trước tất cả các câu lệnh khác.
- Nếu hai biến được khai báo với cùng một phạm vi hoạt động thì không được phép trùng tên.
- Do biến cục bộ sẽ bị xóa sau khi khối lệnh mà có khai báo biến đó kết thúc nên nó không ảnh hưởng tới toàn bộ chương trình. Vì vậy, biến toàn cục và biến cục bộ hoặc hai biến cục bộ của hai khối lệnh khác nhau có thể đặt trùng tên.

2.5.4. Lấy địa chỉ của biến

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

&tên_biến

Ví dụ: &a: lấy địa chỉ biến a

2.6. Hằng

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình thực hiện chương trình. Trong Turbo C, mỗi hằng đều có một giá trị và kiểu dữ liệu của nó như hằng số nguyên, hằng số thực, hằng ký tự…. Sau đây, chúng ta sẽ xét lần lượt các hằng trong C.

2.6.1. Hằng kiểu số nguyên

Hằng nguyên (hằng **int**) là các số nguyên có giá trị trong khoảng từ -32768 đến 32767. Có thể biểu diễn hằng nguyên theo ba dạng:

- Dạng thập phân (cơ số 10): số nguyên thập phân là dạng đơn giản và thông dụng nhất,ví dụ 222, -123…
- Dạng bát phân (cơ số 8): một số nguyên dạng bát phân luôn bắt đầu bằng một số 0, ví dụ: 023, 0956, 012…
- Dạng thập lục phân (cơ số 16): một số nguyên dạng thập lục phân (hệ hexa) luôn bắt đầu bằng 0x, ví dụ: 0x23, 0x3a4, 0xa02

2.6.2. Hằng kiểu long

Hằng **long** cũng giống như hằng **int** nhưng khác là có thêm chữ *L* hoặc *l* ở cuối giá trị hằng để biểu thị hằng đó là hằng kiểu **long.**

Ví dụ: 345*l,* -2006L.

Một hằng **int** nếu vượt ra ngoài phạm vi cho phép thì được ngầm hiểu là hằng **long.**

2.6.3. Hằng dấu phẩy động

Hằng dấu phẩy động gồm các số thuộc kiểu **float** và **double**, và được biểu diễn theo hai cách:

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

214.35 -456.48 244.0

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

.33 105.

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

123.456E-4 (biểu diễn giá trị 0.0123456)

0.548E3 (biễu diễn giá trị 548.0)

-49.5e2 (biểu diễn giá trị -4950.0)

2.6.4. Hằng ký tự

Hằng ký tự là một ký tự riêng biệt được viết trong 2 dấu nháy đơn ‘ ’.

Ví dụ: ‘d’: giá trị của ‘d’ chính là mã ASCII của chữ cái a. Theo bảng mã ASCII thì giá trị của ‘d’ là 100. Vì vậy, hằng ký tự có thể tham gia vào các phép toán như các số nguyên khác.

+) ***Lưu ý:*** ký tự ‘0’ và ‘\0’ là khác nhau: Ký tự ‘0’ là ký số 0, có giá trị mã ASCII là 48. Còn ký tự ‘\0’ là ký tự NULL, có giá trị mã ASCII bằng 0.

2.6.5. Hằng chuỗi

Hằng chuỗi ký tự là một dãy ký tự bất kỳ đặt trong cặp dấu nháy kép “ ”.

Ví dụ:

“lap trinh c”

“Dien tu vien thong”

“ ” /* xâu rỗng*/

Chuỗi kí tự được lưu trữ trong máy dưới dạng một mảng các phần tử là các ký tự riêng biệt. Tất cả các chuỗi đều có ký tự kết thúc ‘\0’ ở cuối mỗi chuỗi.

+) ***Lưu ý:*** Cần phân biệt ‘a’ và “a”: ‘a’ là hằng kí tự có giá trị nguyên là mã ASCII của chữ cái a, còn “a” là một hằng xâu kí tự được lưu trữ trong một mảng hai phần tử: phần tử thứ nhất là chữ a còn phần tử thứ 2 chứa ký tự kết thúc ‘\0’.

2.6.6. Khai báo hằng

+) Ta có thể khai báo hằng theo cú pháp sau:

const tên_hằng = biểu_thức;

Ví dụ: **const** pi = 3.1416;

const tong_tro = r1 + r1;

+) Hoặc ta có thể khai báo hằng bằng cách định nghĩa một macro sau:

#define tên_hằng = giá_trị

Ví dụ: #define pi = 3.1416

#define micro = 1e-6

Khi định nghĩa macro **#define** cần phải khai báo ngoài hàm và lưu ý là không có dấu chấm phẩy (;) ở cuối khai báo.

2.7. Các phép toán

Trong C chia thành bốn nhóm phép toán cơ bản sau:

2.7.1. Phép toán số học

Các phép toán số học được sử dụng để thực hiện những thao tác mang tính số học. Chúng được chia ra thành hai lớp : phép toán số học một ngôi và phép toán số học hai ngôi.

2.7.1.1. Phép toán số học hai ngôi

Phép toán số học hai ngôi bao gồm các phép toán: + - * / và %.

Cú pháp của phép toán số học hai ngôi (hai toán hạng) như sau:

Toán hạng *phép toán* **toán hạng**

Bảng sau sẽ mô tả rõ hơn cách sử dụng các phép toán trên:

Bảng 2.3 Các phép toán số học hai ngôi

Các toán tử hai ngôi	**Chức năng**
+	Cộng
-	Trừ
*	Nhân
%	Phép chia lấy dư
/	Chia

Thí dụ: a + b, 1/f, 10 % 3

+) ***Lưu ý:***

- Phép cộng, trừ, nhân và chia được thực hiện đối với tất cả các toán hạng có các kiểu dữ liệu: **char, int, long, float, double .**
- Phép chia lấy dư là một phép chia số nguyên, vì vậy phép toán này chỉ áp dụng đối với các kiểu dữ liệu **char, int, long.**

2.7.1.2. Phép toán số học một ngôi

Các phép toán số học một ngôi được biểu diễn trong bảng sau:

Bảng 2.4 Các phép toán số học một ngôi

Phép toán	Ý nghĩa
-	Trừ một ngôi
++	Tăng một đơn vị
--	Giảm một đơn vị

Lưu ý: Các phép toán một ngôi (++ và --) mang ý nghĩa khác nhau khi đứng trước hoặc sau biến.

Thí dụ:

i++ ⇔ i = i +1: Tăng giá trị của i sau khi đã sử dụng giá trị của i.

++i ⇔ i = i +1: Tăng giá trị của i trước khi sử dụng giá trị của i.

+) Quyền ưu tiên: được thể hiện trong bảng sau với mức độ ưu tiên giảm dần từ trên xuống dưới

Bảng 2.5 Bảng thể hiện quyền ưu tiên của các phép toán

Loại toán tử	**Toán tử**
Một ngôi	- , ++, --
Hai ngôi	*, /, %
	+, -
	=

2.7.2. Phép toán so sánh

Phép toán so sánh, hay còn gọi là phép toán quan hệ, được dùng để kiểm tra mối quan hệ giữa hai biến, hay giữa một biến và một hằng và trả về kết quả là đúng (TRUE) hoặc sai (FALSE). Trong C quy ước giá trị 0 là sai và giá trị khác 0 (thường là 1) là đúng.

Bảng sau mô tả ý nghĩa của các phép toán quan hệ.

Bảng 2.6 Các phép toán so sánh

Phép toán so sánh	**Mô tả**
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Khác

Thí dụ:

3 >= 3 → có giá trị 1 (đúng)

5 == 6 → có giá trị 0 (sai)

4 <= 2 → có giá trị 0 (sai)

8 != 4 → có giá trị 1 (đúng)

+) *Lưu ý:*

- Cần phân biệt hai phép toán: phép gán (=) và phép toán so sánh bằng (==)
- Kết quả của phép toán so sánh là một số nguyên kiểu int (bằng 1 nếu đúng, bằng 0 nếu sai), nên hoàn toàn có thể tham gia vào các phép toán số học khác.

+) Quyền ưu tiên:

- Trong phép toán so sánh đã chỉ ra ở bảng trên, bốn phép toán đầu (>, >=, <, và <=) có độ ưu tiên như nhau nhưng cao hơn hai phép toán sau cũng có độ ưu tiên như nhau (== và !=).
- Các phép toán so sánh thì có độ ưu tiên thấp hơn các phép toán số học.

Thí dụ: nếu thực hiện phép toán:

10 >=5+7 sẽ tương đương với 10>=(5+7) và kết quả trả về là fasle (0)

2.7.3. Phép toán logic

Các phép toán logic là các ký hiệu dùng để kết hợp hoặc phủ định biểu thức có chứa các toán tử quan hệ. Các phép toán logic cho kết quả trả về là TRUE (1) hoặc FASLE (0). Bảng sau mô tả các toán tử logic trong C:

Bảng 2.7 Các phép toán logic và mức độ ưu tiên

Phép toán logic	Ý nghĩa	Độ ưu tiên
!	NOT (phủ định)	1
&&	AND (phép giao)	2
||	OR (phép hoặc)	3

Thí dụ: Đoạn lệnh sau đây sử dụng các phép toán so sánh và logic:

```
scanf("%d", &a);
if(a>0 && a<255)
// sử dụng hai phép toán so sánh và một phép toán logic
 printf("nhập đúng");
```

Kết quả trả về của các phép toán logic được cho trong bảng chân lý sau:

A	B	!A	A&&B	A||B
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

***Lưu ý*:** Độ ưu tiên của các phép toán logic đã chỉ ra ở trên. Tuy nhiên cần lưu ý hai điều sau:

- Phép toán NOT (!) có độ ưu tiên cao hơn các phép toán quan hệ; còn các phép toán quan hệ lại có độ ưu tiên cao hơn phép toán AND (&&) và OR (||).
- Các phép toán logic đều có độ ưu tiên thấp hơn các phép toán số học.

2.7.4. Phép toán xử lý bit

Thông thường, trong các ngôn ngữ bậc cao như Basic, Fortran. Pascal… đơn vị được xử lý tính bằng **byte.** Các phép toán xử lý trên bit thường chỉ thấy trong ngôn ngữ Assembly. Tuy nhiên, trong C cũng có khả năng xử lý dữ liệu đến từng bit thông qua các phép toán sau:

Bảng 2.8 Các phép toán xử lý bit

Phép toán xử lý bit	Ý nghĩa
&	Phép Và (AND) theo bit
|	Phép Hoặc (OR) theo bit
^	Phép Hoặc loại trừ (XOR)
<<	Dịch trái
>>	Dịch phải
~	Lấy bù theo bít (NOT)

Các phép toán này chỉ thực hiện trên các toán hạng có kiểu dữ liệu là số nguyên như **char, int, long, signed** và **unsigned.** Ý nghĩa, cách thực hiện của các phép toán xử lý bít sẽ được trình bày rõ trong phần sau.

+) ***Lưu ý:***

Cần phân biệt phép toán logic và phép toán xử lý bit. Chúng khác nhau cơ bản về ý nghĩa. Ví dụ giữa phép toán AND logic (&&) và phép AND bit (&) như sau:

- Phép AND logic thực hiện khảo sát trên cả hai toán hạng. Nó thực chất là phép lấy giao của hai toán hạng. Kiểm tra nếu cả hai toán hạng đều đúng (1) thì nó cho kết quả phép AND logic là đúng (1). Ngược lại, nó trả về kết quả là sai (0).
- Phép AND bit thực hiện xử lý trên từng bit tương ứng của hai toán hạng, nếu hai bít tương ứng đều là bit 1 thì bít kết quả là 1

2.7.5. Một số phép toán khác

*2.7.5.1. Toán tử con trỏ & và **

Một con trỏ là địa chỉ trong bộ nhớ của một biến. Một biến con trỏ là một biến được khai báo riêng để chứa một con trỏ đến một đối tượng của kiểu đã chỉ ra nó. Con trỏ sẽ được tìm hiểu kỹ hơn trong chương sau con trỏ. Trong C có hai toán tử được sử dụng để thao tác với các con trỏ:

Toán tử &: là một toán tử quy ước trả về địa chỉ bộ nhớ của hệ số của nó.

Thí dụ: *dia_chi = &mang_dien_tro*

Câu lệnh trên nghĩa là đặt vào biến *dia_chi* địa chỉ bộ nhớ của biến *mang_dien_tro*. Chẳng hạn, biến *mang_dien_tro* ở vị trí bộ nhớ 250, giả sử biến *mang_dien_tro* có giá trị là 50. Sau câu lệnh trên *dia_chi* sẽ nhận giá trị 250.

Toán tử *: là một bổ sung cho &; đây là một toán tử quy ước trả về giá trị của biến được cấp phát tại địa chỉ theo sau đó. Thí dụ:

*R1 = *dia_chi*

Sẽ lấy giá trị của biến có địa chỉ chứa trong biến *dia*_chi và gán cho biến mới *R1*, tức là đặt giá trị của *mang_dien_tro* vào *R1*. Bây giờ *R1* sẽ có giá trị là 50 vì giá trị 50 được lưu trữ tại địa chỉ 250.

2.7.5.2. Toán tử dấu phẩy(,)

Toán tử dấu phẩy (,) được sử dụng để kết hợp các biểu thức lại với nhau. Bên trái của toán tử dấu "," luôn được xem là kiểu void. Điều đó có nghĩa là biểu thức bên phải trở thành giá trị của tổng các biểu thức được phân cách bởi dấu phẩy. Thí dụ:

x = (a=3,a+1);

Trước hết gán 3 cho a rồi gán 4 cho x. Cặp dấu ngoặc đơn là cần thiết vì toán tử dấu , có độ ưu tiên thấp hơn toán tử gán.

2.7.5.3. Toán tử (), [ ]

Trong C, cặp dấu ngoặc đơn ( )là toán tử để thay đổi độ ưu tiên của các phép toán. Biểu thức nào được đặt trong dấu ngoặc đơn sẽ có độ ưu tiên cao hơn. Nếu có nhiều dấu ngoặc đơn lồng nhau, thì biểu thức nằm trong cặp dấu ngoặc trong cùng sẽ có độ ưu tiên cao nhất.

Thí dụ:

A= 5 + (3 * (1&&(3>=4)));

Câu lệnh gán trên, đầu tiên phép toán so sánh 3>=4 được thực hiện, trả về kết quả 0, sau đó phép toán logic AND thực hiện 1&&0 và trả về kết quả 0. Tiếp theo là phép nhân 3 * 0 cho kết quả 0, và cuối cùng là tổng 5 + 0. Vậy A=5.

Các cặp dấu ngoặc vuông [ ] thực hiện thao tác truy xuất phần tử trong mảng.

2.8. Phép toán xử lý bit

2.8.1. Phép toán AND

Kí hiệu: &

Ý nghĩa: Nhân logic trên các bit. Phép toán này thực hiện trên từng cặp bit tương ứng của các toán hạng theo quy tắc trong bảng sau:

Bảng 2.9 Bảng chân lý phép toán AND trên bit

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Thí dụ: Thực hiện phép toán:

12 & 7 ⇔ 1100 & 0111 = 0100 = 4.

2.8.2. Phép toán OR

Kí hiệu: **|**

Ý nghĩa: Cộng logic trên các bit. Phép toán này thực hiện trên từng cặp bit tương ứng của các toán hạng theo quy tắc trong bảng sau:

Bảng 2.10 Bảng chân lý phép toán OR trên bit

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Thí dụ: thực hiện phép toán:

12 | 7 = 1100 | 0111 = 1111 =15.

2.8.3. Phép toán XOR

Kí hiệu: ^

Ý nghĩa: Phép lật bít . Thực hiện trên từng cặp bit tương ứng của các toán hạng theo quy tắc trong bảng sau.

Bảng 2.11 Bảng chân lý phép toán XOR trên bit

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Thí dụ: Thực hiện phép toán:

12 ^ 7 = 1100 ^ 0111= 1011 = 11 (giá trị thập phân).

2.8.4. Phép toán NOT

Kí hiệu: ~

Ý nghĩa: phép đảo bit, đổi các giá trị trong mỗi bit của toán hạng x từ 0->1, 1->0.

A	~A
0	1
1	0

Thí dụ:

~12 = ~ 1100 = 0011 = 3.

2.8.5. Phép toán dịch trái/phải

2.8.5.1. Phép dịch phải: x >> i

Cho giá trị có được từ số nguyên **x** sau khi dịch sang phải **i** bit; các số 0 sẽ lấp đầy các kết quả bên trái nếu là số nguyên dương; nếu là số nguyên âm thì số 1 sẽ lấp đầy các kết quả bên trái.

Thí dụ:

8 >> 2 = 2 (1000 >> 2 = 0010)

2.8.5.2. Phép dịch trái: x << i

Phép dịch trái, cho giá trị có được từ số nguyên x sau khi dịch sang trái i bit; các số 0 sẽ lấp đầy các kết quả ở bên phải.

8 << 2 = 32 (1000 << 2 = 100000)

2.9. Biểu thức

2.9.1. Khái niệm biểu thức

Trong ngôn ngữ C, biểu thức được hiểu là sự kết hợp của các phép toán và các toán hạng để diễn đạt một công thức, một quy trình toán học nào đó.

- Toán hạng có thể là các biến, hằng, các hàm hoặc giá trị của các hàm
- Phép toán bao gồm các phép toán đã trình bày ở trên.

Biểu thức thường được dùng trong các trường hợp:

- Vế phải của câu lệnh gán

- Làm tham số thực của hàm (như hàm printf)
- Làm chỉ số
- Trong các toán tử if, switch, for, while, do while.

Thí dụ:

Tong_tro = R1*R2*/(R1+R2);

Cam_khang = 2.0* 3.14*f*L;

2.9.2. Phép gán

Phép gán là một phép toán cơ bản của bất kỳ ngôn ngữ lập trình nào, dùng để gán giá trị của một biểu cho một biến.

Cú pháp**:** Tên_biến = biểu_thức;

Thí dụ:

Tong_tro = R1*R2*/(R1+R2);

Cam_khang = 2.0* 3.14*f*L;

Đối với một số phép gán đặc biệt khi mà biến được gán có mặt ở cả hai vế của biểu thức gán thì có thể được viết ngắn gọn lại như sau:

Thí dụ:

x = x +3; ⇔ x += 3;

a = a * (b+5); ⇔ a *= b+5;

Các phép toán có thể được áp dụng cách viết ngắn gọn trên là các phép toán số học và phép toán thao tác bit hai ngôi (+, -, * , / , %, <<, >>, &, | và ^).

2.9.3. Chuyển đổi kiểu dữ liệu

Trong C có hai cách chuyển đổi kiểu dữ liệu là: chuyển kiểu tự động và chuyển kiểu bắt buộc.

2.9.3.1. Chuyển kiểu tự động:

Trong một biểu thức có các toán hạng của một toán tử thuộc về nhiều kiểu dữ liệu khác nhau. Những toán hạng này thông thường được chuyển về cùng kiểu với toán hạng có kiểu dữ liệu lớn nhất. Điều này được gọi là tăng cấp kiểu. Sự phát triển về kiểu dữ liệu theo thứ tự sau : **char < int <long <float <double**

Quy tắc đổi kiểu được trình bày dưới đây giúp các bạn xác định được giá trị của biểu thức một cách chính xác:

- **char** và **short** được chuyển thành **int** và **float** được chuyển thành **double.**
- Nếu có một toán hạng là **double**, toán hạng còn lại sẽ được chuyển thành **double**, và kết quả là **double**.
- Nếu có một toán hạng là **long**, toán hạng còn lại sẽ được chuyển thành **long**, và kết quả là **long.**
- Nếu có một toán hạng là **unsigned**, toán hạng còn lại sẽ được chuyển thành **unsigned** và kết quả cũng là **unsigned**.

- Nếu tất cả toán hạng kiểu **int,** kết quả là **int.**
- Nếu một toán hạng là **long** và một toán hạng là **unsigned** thì sẽ chuyển thành **unsigned long.**

Thí dụ:

khai báo biến

```
char ch;
        int i;
        float f;
        double d;
    // biểu thức:
          bt =  (ch/i) + (f*d) – (f+i);
                  ↓        ↓       ↓
                 int    double   float
                  └───┬───┘        │
                      ↓            │
                   double          │
                      └─────┬──────┘
                            ↓
                         double
```

2.9.3.2. Chuyển kiểu ép buộc:

Khi ta muốn điều khiển quá trình chuyển kiểu theo ý muốn của mình thì có thể dùng phép ép kiểu:

Cú pháp: **(kiểu_dữ_liệu)** biểu thức;

Thí dụ: float R1, R2;

int

(int) R1;

(int) (R1 +R2);

Khi đó, giá trị của R1 và tổng R1+R2 sẽ là kiểu **int.**

Tuy nhiên, cần lưu ý rằng bản thân các biến vẫn giữ nguyên kiểu khai báo.Tức là, ở đây biến R1 vẫn có kiểu là **float** nhưng (int) R1 thì có giá trị kiểu **int.**

2.10. Cấu trúc cơ bản của chương trình C

Trước khi trình bày về cấu trúc cơ bản của một chương trình C, chúng tôi đưa ra một số khái niệm có liên quan đến chương trình C để bạn đọc dễ theo dõi:

+) Câu lệnh: Là đơn vị nhỏ nhất trong mỗi chương trình nhằm thực hiện một công việc nào đó.

Câu lệnh được chia làm hai loại: câu lệnh đơn và câu lệnh có cấu trúc. Câu lệnh đơn chỉ gồm một lệnh duy nhất như: lệnh gán, lệnh nhập hay xuất dữ liệu, lệnh gọi

hàm, lệnh nhảy… Câu lệnh có cấu trúc có thể là lệnh ghép gồm một khối lệnh đặt giữa cặp dâu {}, lệnh lựa chọn (if, case) hay các lệnh lặp (for, while, do...while).

+) Khối lệnh: Là một tập các câu lệnh nằm trong cặp dấu { và }. Trong Turbo C, một khối lệnh cũng được xem như một câu lênh riêng lẻ. Do vậy, ở đâu có thể viết một câu lệnh thì ở đó cũng có thể đặt một khối lệnh.

Các khối lệnh có thể lồng nhau, có nghĩa là bên trong một khối lệnh lại có thể viết các khối lệnh khác.

+) Lời chú thích: Trong quá trình lập trình, người lập trình thường sử dụng những chú thích để giải thích về các khai báo, các lệnh, các hàm chức năng, thuật toán, … giúp cho chương trình dễ đọc hơn. Có hai cách chú thích:

- Chú thích trên một dòng được đặt sau cặp dấu: // lời chú thích
- Chú thích trên nhiều dòng được đặt trong cặp đấu: /* lời chú thích*/

Khi chạy chương trình máy tính sẽ bỏ qua các lời chú thích trong chương trình

2.10.1. Cấu trúc chương trình

Cấu trúc cơ bản của một chương trình C như sau:

```
+ khai báo tiền xử lý:
    #include “tệp tiêu đề”
    #define tên_hằng  giá_trị
    …
+ khai báo biến toàn cục
+ hàm main( )
    {    // điểm bắt đầu chương trình
         // thân hàm
         Các khai báo (biến, hằng, mảng, xâu…)
         Các câu lệnh, khối lệnh …
    }    // điểm kết thúc chương trình
```

Thí dụ: **Chương trình 2.1.**

```
/* Tính tổng trở của mạch gồm 2 điện trở mắc song song                nối  tiếp  với
một cuộn cảm*/
/* ++++++ khai báo tệp tiêu đề +++++++*/
    #include “stdio.h”
    #include “conio.h”
    #include “math.h”
/* ++++++ khai báo hằng +++++++*/
    #define PI 3.1416
    # define mili 1e-3
    Void main( )
```

```
{
    float r1, r2, L,ZL, tong_tro;
    int f = 500;
    puts("nhap gia tri cua bien:");
    printf("\nr1="); scanf("%f",&r1);
    printf("\nr2="); scanf("%f",&r2);
    printf("\nL ="); scanf("%f",&L);
    tong_tro = (r1*r2) / (r1+r2);
    ZL = 2.0*PI*f*L*mili;//tinh cam khang.
    tong_tro += ZL;  // tinh tong tro cua mach
    printf("\ntong tro cua mach la: %f", tong_tro);            getch();
}
```

Giả sử ta nhập các giá trị r1= 25.5, r2 = 100.0, và L = 1.2

Kết quả hiển thị trên màn hình là:

```
nhap gia tri cua bien;
r1 = 25.500000
r2 = 100.000000
L = 1.200000
tong tro cua mach la: 24.088645
```

2.10.2. Một số chú ý khi viết chương trình

Để tránh sai sót trong lập trình với ngôn ngữ C, sau đây chúng tôi đưa ra một số chú ý quan trọng cần lưu ý khi viết một chương trình:

- Mỗi câu lệnh có thể viết trên một dòng hay nhiều dòng và phải được kết thúc bởi dấu (;). Ngược lại, nhiều câu lệnh có thể viết trên cùng một dòng.

Nếu viết một xâu hằng kí tự trên nhiều dòng thì phải đặt thêm kí tự '/' trước khi xuống dòng để báo cho TURBO C biết một chuỗi kí tự vẫn còn tiếp tục ở dòng dưới.

Thí dụ:

```
{
printf("\nNgon ngu lap trinh C/
\n trong dien tu va ky thuat so") ;
}
```

- Các lời chú thích được viết trong cặp dấu /* */.

- Khi sử dụng một hàm nào đó cần phải biết hàm đó nằm trong thư viện có tệp tiêu đề nào của chương trình và phải khai báo tệp tiêu đề đó bằng lệnh:

#include <Tệp tiêu đề>

ở phần đầu của chương trình.

- Ở cuối các khai báo #include không có dấu chấm phẩy (;).

Thí dụ, lệnh **printf** để xuất dữ liệu ra màn hình, hàm này nằm trong tệp tiêu đề stdio.h trong thư mục của C. Do đó phải khai báo:

#include <stdio.h>

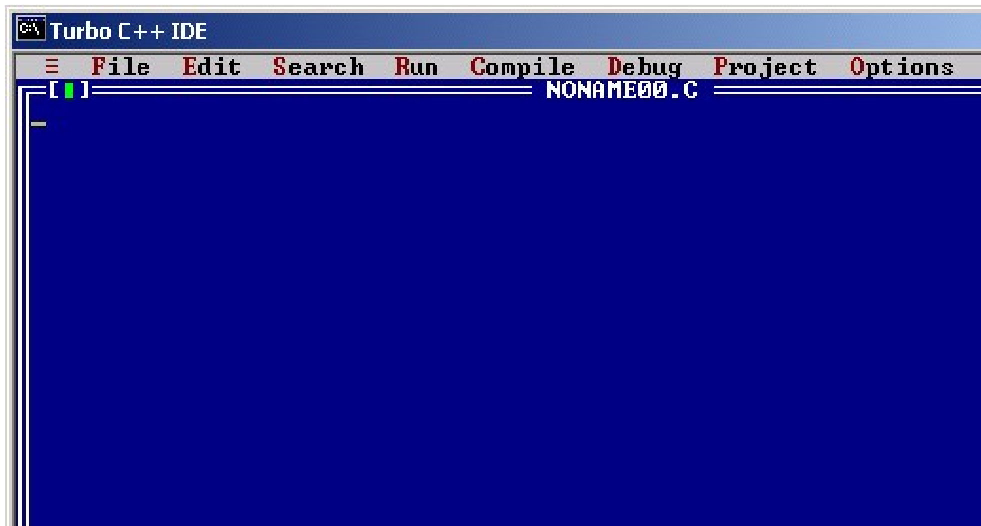
- Một chương trình chỉ có thể chỉ có duy nhất một hàm chính (main) và có thể chứa thêm nhiều hàm khác nữa.

2.11. Làm quen với môi trường soạn thảo Turbo C

Turbo C là môi trường hỗ trợ lập trình C do hãng Borland cung cấp. Môi trường này cung cấp các chức năng như: soạn thảo chương trình, dịch, thực thi chương trình…Sau đây chúng ta sẽ làm quen với một số thao tác với môi trường Turbo C.

2.11.1. Mở Turbo C

Để khởi động môi trường của Turbo C, từ dấu nhắc của DOS ta nhập dòng lệnh TC.EXE và nhấn phím Enter hoặc có thể mở trực tiếp file chương trình TC.EXE đã được đặt trong một thư mục nào đó. Kết quả sẽ mở ra màn hình giao diện của TC như sau:



Hình 2.4 Màn hình giao diện của Turbo C

Dòng trên cùng gọi là thanh menu (menu bar). Mỗi mục trên thanh menu lại có thể có nhiều mục con nằm trong một menu kéo xuống.

Dòng dưới cùng ghi chức năng của một số phím đặc biệt. Ví dụ khi gõ phím F1 thì ta có được một hệ thống trợ giúp mà ta có thể tham khảo nhiều thông tin bổ ích.

2.11.2. Soạn thảo chương trình mới

Muốn soạn thảo một chương trình mới ta chọn mục New trong menu File (File ->New).

Trên màn hình sẽ xuất hiện một vùng trống để cho ta soạn thảo nội dung của chương trình. Trong quá trình soạn thảo chương trình ta có thể sử dụng các phím sau:

2.11.2.1. Các phím xem thông tin trợ giúp:

- F1: Xem toàn bộ thông tin trong phần trợ giúp.

- Ctrl-F1: Trợ giúp theo ngữ cảnh (tức là khi con trỏ đang ở trong một từ nào đo, chẳng hạn int mà bạn gõ phím Ctrl-F1 thì bạn sẽ có được các thông tin về kiểu dữ liệu **int**).

2.11.2.2. Các phím di chuyển con trỏ trong vùng soạn thảo

Các phím xóa dòng/ ký tự

Các phím chèn ký tự/dòng

2.11.3. Lưu chương trình vào bộ nhớ

Sử dụng File/Save hoặc gõ phím F2. Có hai trường hợp xảy ra:

- Nếu chương trình chưa được ghi lần nào thì một hội thoại sẽ xuất hiện cho phép bạn xác định tên tập tin (FileName). Tên tập tin phải tuân thủ quy cách đặt tên của DOS và không cần có phần mở rộng (sẽ tự động có phần mở rộng là .C hoặc .CPP sẽ nói thêm trong phần Option). Sau đó gõ phím Enter.
- Nếu chương trình đã được ghi một lần rồi thì nó sẽ ghi những thay đổi bổ sung lên tập tin chương trình cũ.

Quy tắc đặt tên tập tin của DOS: Tên của tập tin gồm 2 phần: Phần tên và phần mở rộng.

- Phần tên của tập tin phải bắt đầu là 1 ký tự từ a..z (không phân biệt hoa thường), theo sau có thể là các ký tự từ a..z, các ký số từ 0..9 hay dấu gạch dưới (_), phần này dài tối đa là 8 ký tự.
- Phần mở rộng: phần này dài tối đa 3 ký tự. Ví dụ .txt, .c, .cpp…

2.11.4. Mở chương trình từ bộ nhớ

Với một chương trình đã có trên đĩa, ta có thể mở nó ra để thực hiện hoặc sửa chữa bổ sung. Để mở một chương trình ta dùng File/Open hoặc gõ phím F3. Sau đó gõ tên tập tin vào hộp File Name hoặc lựa chọn tập tin trong danh sách các tập tin rồi gõ Enter.

2.11.5. Dịch và thực hiện chương trình

- Để dịch chương trình, nhấn tổ hợp phím Alt– F9.
- Để thực hiện chương trình hãy dùng tổ hợp phím Ctrl-F9.

2.11.6. Thoát khỏi môi trường C

Vào thanh menu File và kéo xuống mục Exit hoặc Alt-X.

Chương 3 CÁC LỆNH XUẤT NHẬP DỮ LIỆU

3.1. Giới thiệu chung

Chúng ra sau khi xử lý có thể được thực hiện theo hai cách:

- Thông qua phương tiện nhập/xuất chuẩn (I / O – Input/ Output).
- Thông qua những tập tin.

Trong ngôn ngữ C, thư viện chuẩn được liên kết trong chương trình thực hiện cung cấp những thủ tục cho việc nhập và xuất. Thư viện chuẩn có những hàm quản lý các thao tác nhập/xuất cũng như các thao tác trên ký tự và chuỗi.

Trong chương này sẽ trình bày chi tiết các hàm nhập dùng để đọc dữ liệu vào từ thiết bị nhập chuẩn (standard input) và tất cả những hàm xuất dùng để viết kết quả ra thiết bị xuất chuẩn (Standard output). Thiết bị nhập chuẩn thông thường là bàn phím. Thiết bị xuất chuẩn thông thường là màn hình. Dữ liệu đầu ra cũng có thể được gửi đến máy in.

Với một ngôn ngữ lập trình bất kỳ, việc nhập giá trị cho các biến và in

3.2. Các lệnh xuất dữ liệu cơ bản

3.2.1. Chuỗi điều khiển

3.2.1.1. Các ký tự điều khiển con trỏ màn hình

Các ký tự điều khiển con trỏ màn hình gồm có các ký tự như trong bảng 3.1 dưới đây:

Bảng 3.12 Các ký tự điều khiển

Ký tự	**Ý nghĩa**
\n	Sang dòng mới
\t	Dấu trống tab
\r	Trở về đầu dòng
\f	Sang trang mới
\b	Xóa trái một ký tự

3.2.1.2. Các ký tự hiển thị đặc biệt

Trong C, ngoài các ký tự hiển thị thông thường còn bao gồm một số ký tự hiển thị đặc biệt và cách hiển thị các ký tự đặc biệt này lên màn hình được cho dưới đây:

Bảng 3.13 Các ký tự hiển thị đặc biệt

Các ký tự hiển thị đặc biệt	**Ý nghĩa**
\”	In ra dấu ”
\\	In ra dấu \
\nnn	In ra ký tự ASCII trong mã bát phân. Ví dụ:

	\041 cho kết quả là '!'
\0xnn	In ra ký tự ASCII trong mã Hexa Ví dụ: \0x41 cho kết quả là 'A'
\0	Ký tự NULL

3.2.1.3. Các đặc tả chuyển dạng và tạo khuôn

a) Cấu trúc tổng quát của đặc tả chuyển dạng và tạo khuôn

Cấu trúc tổng quát của đặc tả:

%[-][fw][.pp]ký tự chuyển dạng.

Trong các thành phần trên, thành phần dấu % và ký tự chuyển dạng là bắt buộc phải có, còn các thành phần khác có thể vắng mặt.

- **Dấu '-':**

- Khi không có mặt dấu trừ (-) thì kết quả ra (trường ra) được dồn về bên phải nếu độ dài thực tế của trường ra nhỏ hơn độ rộng tối thiểu của fw và các vị trí dư thừa được lấp đầy bằng khoảng trống hoặc số 0 (chỉ với trường hợp dãy số fw bắt đầu bằng một số 0)

- Khi có dấu trừ (-) thì kết quả ra được dồn về phía bên trái và các vị trí dư thừa sẽ được chèn thêm các khoảng trống.

Thí dụ:

Trường ra	**Dấu (-)**	**fw**	**Kết quả hiển thị**
1234	Không	6	: 1234 :
1234	Không	06	:001234:
1234	Có	6	:1234 :
"hello"	Có	6	:hello :
"hello"	Không	6	: hello:

- **fw: Là dãy số nguyên xác định độ rộng tối thiểu dành cho trường ra**

- Khi **fw** lớn hơn độ dài thực tế của trường ra thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số 0 và phần nội dung của trường được đẩy về phía bên phải hoặc bên trái như đã được giải thích ở phần trên.

- Khi vắng mặt thành phần fw hay fw nhỏ hơn độ dài thực tế của trường ra thì độ dài của kết quả hiển thị chính là độ dài thực tế của trường.

- **.pp là dãy số nguyên chỉ sử dụng khi đối số tương ứng là xâu ký tự hoặc kiểu float hay double.**

- Trường hợp đối số là xâu kí tự: Nếu **pp** nhỏ hơn độ dài của xâu thì chỉ có **pp** ký tự đầu tiên được in ra. Nếu vắng mặt **pp** hoặc **pp** lớn hơn độ dài của xâu thì cả xâu ký tự được đưa ra.

- Trường hợp đối số là kiểu float hay double: **pp** được gọi là độ chính xác của trường ra.Tức là giá trị in ra sẽ có **pp** chữ số sau dấu chấm thập phân. Nếu vắng mặt **pp** thì độ chính xác mặc định là 6.

- **Ký tự định dạng:** là một hoặc một dãy ký hiệu dùng để xác định quy tắc chuyển dạng và dạng xuất lên màn hình của đối số tương ứng. Bảng dưới đây giải thích về các ký tự chuyển dạng.

Bảng 3.14 Các ký tự chuyển dạng trong xuất dữ liệu

Định dạng	**Ký tự chuyển dạng**
Ký tự đơn (Single Character)	c
Chuỗi (String)	s
Số nguyên có dấu (Signed decimal integer)	d, i
Số thập phân có dấu chấm động (Floating point)	F hoặc f
Số thập phân dấu chấm động - Biểu diễn dạng số mũ	E hoặc e
Số thập phân có dấu chấm động (%f hoặc %e, con số nào ít hơn)	G hoặc g
Số nguyên không dấu (Unsigned decimal integer)	u
Số thập lục phân không dấu (unsigned hexadecimal integer)	x
Số bát phân không dấu (Unsigned octal integer)	o
Số nguyên có dấu kiểu "long"	ld, li
Số thập phân dấu chấm động kiểu "long"	lf
Số bát phân không dấu kiểu "long"	lo
Số thập lục phân không dấu kiểu "long"	lx

Một số trường hợp ví dụ:

%d, %f: Chỉ gồm dấu % và ký tự chuyển dạng

%3f: Gồm dấu %, fw và ký tự chuyển dạng

%-3.2f: Dạng đầy đủ

b) Hai chú ý quan trọng khi viết đặc tả chuyển dạng

- **Chú ý 1**: Đặc tả chuyển dạng bắt đầu bởi dấu % và kết thúc bởi một ký tự chuyển dạng. Và tất cả các ký tự không bắt đầu bằng dấu % hoặc không kết thúc bằng một ký tự chuyển dạng đều được xem là ký tự chuyển dạng
- **Chú ý 2**: Tại vị trí của [fw] có thể thay thế bởi dấu *, khi đó việc thay đổi độ rộng của trường ra hết sức linh động.

3.2.2 Số nguyên hệ 10 có dấu và không dấu

Nội dung này liên quan đến kết quả hiển thị trên màn hình của số nguyên với các

ký tự chuyển dạng d, ld, u và lu. Như đã biết, một số nguyên hệ 10 có dấu (kiểu int) được biểu diễn trên 16 bit, trong đó bit đầu tiên (bên trái) biểu diễn dấu của số và giá trị của số được biểu diễn trên 15 bit sau. Dưới đây chúng ta xét đến cách biểu diễn số nguyên hệ 10 có dấu trên máy tính.

▪ Biểu diễn số nguyên dương: bít xác định dấu (bit đầu tiên bên trái) bằng 0. Giá trị của số được biểu diễn trên 15 bit sau:

$0 <= x <= 2^{15} – 1 = 32767$

Tức là trên máy tính sẽ biểu diễn số nguyên dương kiểu int có giá trị từ 0 đến 32767.

▪ Biểu diễn số nguyên âm: Bit xác định dấu bằng 1. Giá trị của số được biểu diễn trên 15 bit sau. Nếu gọi giá trị tuyệt đối của số là y, thì trong 15 bit cuối sẽ không biểu diễn y mà sẽ biểu diễn phần bù của y như sau:

$x = 2^{15} – y = 32768 - y$

Vì 0 <= x <= 32767 nên 1 <= y <= 32768.

Như vậy máy tính có thể biểu diễn các số nguyên âm trong khoảng giá trị từ -32768 đến -1.

Khác với số nguyên có dấu, số nguyên không dấu không có bit xác định dấu nên giá trị của nó được chứa trong cả 16 bit, tức là các số nguyên không dấu sẽ mang giá trị trong khoảng từ 0 đến $2^{16} – 1= 65535$.

Ví dụ: xét biến z = -1.

Nếu ta dùng lệnh: prinf("%d",z);

Thì kết quả hiển thị trên màn hình là -1.

Nếu ta dùng lệnh: prinf("%u",z);

Thì kết quả hiển thị lại là 65535.

Có sự khác biệt trên là do sử dụng các ký tự chuyển dạng khác nhau: trong trường hợp đầu, ký tự chuyển dạng d xác định đối số z là số nguyên có dấu. Ở trường hợp sau, ký tự chuyển dạng u xác định đối z là số nguyên không dấu, mà theo lý giải ban đầu về biểu diễn số âm trên máy tính thì trên 16 bit biểu diễn số -1 đều có giá trị là 1. Như vậy, giá trị số nguyên không dấu sẽ được hiển thị là 65535.

Từ nguyên lý trên, ta có những quy tắc sau để xác định kết quả hiển thị đúng đối với các ký tự chuyển dạng *d, ld, u, lu*:

- Quy tắc thứ nhất: Nếu dùng ký tự chuyển dạng u cho một đối kiểu **int** có giá trị là -x (1<= x <= 32768) thì kết quả hiển thị là số nguyên có giá trị: 65536 – x.
- Quy tắc thứ hai: Nếu dùng ký tự chuyển dạng lu cho một đối kiểu **long** có giá trị là –x (1<= x <= 2147483648) thì giá trị hiển thị là 4294967296 – x.

Chương trình 3.1

```
//chuong trinh 3.1
#include "stdio.h"
#include "conio.h"
void main()
{
char c = 'b';
int z= -3;
printf("\n");
printf(" in ky tu \'%c\' len man hinh",c);
/*in ky tu dac biet (\') va ky tu 'b' theo dinh dang.*/
printf("\n in so nguyen co dau: %d",z);
/* %d la dinh dang cua so nguyen co dau */
printf("\n in so nguyen khong dau: %u",z);
/* %u la dinh dang cua so nguyen khong dau */
getch();
}
```

Kết quả trên màn hình hiển thị:

3.2.3. Các lệnh xuất dữ liệu lên màn hình

Thông thường màn hình là thiết bị ra chuẩn (hay còn gọi là dòng ra chuẩn **stdout( )** để xuất dữ liệu ra từ chương trình. Để xuất dữ liệu ra màn hình ta dùng các hàm **printf( )**, **puts( )**, **putchar( )**. Các hàm này đều nằm trong thư viện chuẩn **stdio.h** của C.

3.2.3.1 Hàm printf( )

- Dạng hàm:

printf ("chuỗi điều khiển", arg1, arg2…,argN);

- Ý nghĩa: Xuất ra một xâu văn bản đã định dạng lên màn hình hiển thị dưới dạng đã được xác định bởi "chuỗi điều khiển".

Các đối số được định nghĩa là arg1, …argN. Danh sách đối số (argument list) bao gồm các hằng, biến, biểu thức hay hàm và được phân cách bởi dấu phẩy. Và cần phải có các lệnh định dạng nằm trong chuỗi điều khiển cho các đối số trong danh sách. Những lệnh định dạng phải tương ứng với danh sách các tham số về số lượng, kiểu dữ liệu và thứ tự. Chuỗi điều khiển phải luôn được đặt bên trong cặp dấu nháy kép **" "**, đây là dấu phân cách.

- Thí dụ: Đoạn lệnh sau:

```
int a, b;
a=5;
b=8;
```

printf ("\n tổng của a và b là: %d +%d = %d", a, b, a+b);

/* hiển thị trên màn hình tổng của a và b.*/

3.2.3.2. Hàm puts( )

- Dạng hàm: **puts** ("xâu");
- Ý nghĩa: Xuất một xâu văn bản lên dòng ra chuẩn và kết thúc là ký tự '\n', tức là sau khi xuất dữ liệu ra màn hình, con trỏ sẽ nhảy về vị trí đầu dòng mới. Nếu có lỗi hàm trả về EOF( ).
- Thí dụ: Câu lệnh:

puts("\nTurbo C")

// in lên màn hình chuỗi ký tự 'TurboC'

3.2.3.3. Hàm putchar( )

- Dạng hàm: **putchar**(ch);
- Ý nghĩa: Đưa ký tự đơn *ch* lên màn hình. Nếu thành công thì hàm trả về ký tự được xuất 'ch'. Khi có lỗi, hàm trả về EOF.
- Thí dụ: Câu lệnh

putchar('A');

//hiện ký tự A lên màn hình tại vị trí hiện tại của con trỏ

chương trình 3.2/* biểu diễn các ký tự chuyển dạng khác nhau*/

```
//chuong trinh 3.2
#include "stdio.h"
#include "conio.h"
void main()
{
int a = 65, b = -65;
printf("\nin ket qua cua a\n voi cac ky tu chuyen dang khac nhau:");
printf("%d\t%u\t%o\t%x\t%c",a,a,a,a,a);
puts("\n in ket qua cua b \n voi cac ky tu chuyen dang khac nhau:");
printf("%d\t%u\t%o\t%x\t%c",b,b,b,b,b);
getch();
}
```

Trong các câu lệnh xuất **printf( )** có sử dụng các ký tự định dạng khác nhau để chuyển sang các kiểu khác nhau của đối. Cụ thể là khi in giá trị của số nguyên a với các ký tự chuyển dạng **%d** sẽ cho kết quả là một số nguyên hệ 10 có dấu, **%u** sẽ cho kết quả là số nguyên không dấu, **%o** cho kết quả là một số bát phân, **%x** cho kết quả là một số hệ thập lục phân và **%c** cho kết quả là một ký tự mã ASCII.

Như vậy, kết quả hiển thị lên màn hình của chương trình là:

```
in ket qua cua a
 voi cac ky tu chuyen dang khac nhau:65 65      101      41       A
 in ket qua cua b
 voi cac ky tu chuyen dang khac nhau:
-65      65471    177677   ffbf     ┐
```

Hình 3.5 Kết quả chương trình 3.2

Từ kết quả trên ta có thể rút ra nhận xét sau:

Với cùng một đối số, nếu sử dụng các ký tự chuyển dạng khác nhau thì sẽ cho kết quả hiển thị khác nhau.

Ngoài ra ta kiểm tra lại quy tắc chuyển dạng số nguyên hệ 10 có dấu (kiểu int) sang các dạng khác sử dụng các ký tự chuyển dạng khác nhau.

Trong câu lệnh **printf( )**, để chuyển sang một dòng mới thì ở cuối chuỗi điều khiển phải sử dụng ký tự điều khiển \n, còn trong câu lệnh **puts( )** thì không cần vì sau khi xâu ký tự được xuất ra, con trỏ sẽ nhảy xuống đầu dòng mới.

Chương trình 3.3/* Các phép tính OR, AND, NOR,NAND, XOR với hai giá trị thập lục phân*/

```
#include “stdio.h”
#include “conio.h”
Void main( )
{
int x1, x2;
printf(“\nNhap vao hai gia tri hex:”);
scanf( “%x\t%x”, &x1, &x2);
printf(“\nGia tri cua phep OR la: %x”,x1|x2);
printf(“\nGia tri cua phep AND la: %x”,x1&x2);
printf(“\nGia tri cua phep NOR la: %x”,~(x1|x2));
printf(“\nGia tri cua phep NAND la: %x\n”,~(x1&x2));
printf(“\nGia tri cua phep XOR la: %x\n”,x1^x2);
getch( );
}
```

Kết quả chạy chương trình: Trên màn hình hiển thị:

```
Nhap vao hai gia tri hex:03e5 54ac

Gia tri cua phep OR la: 57ed
Gia tri cua phep AND la: a4
Gia tri cua phep NOR la: a812
Gia tri cua phep NAND la: ff5b
Gia tri cua phep XOR la: 5749
```

Hình 3.6 Kết quả chương trình 3.3

3.2.4 Xuất dữ liệu ra máy in

Như đã trình bày, ngoài việc xuất dữ liệu lên dòng ra chuẩn (màn hình), dữ liệu còn có thể được xuất ra máy in. Dạng hàm chuẩn xuất ra máy in có dạng như sau:

fprintf(stdprn, "chuỗi điề u khiển",arg1, ...argN);

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

+ Tham số chuỗi điều khiển, và danh sách các đối đã được trình bày ở trên. Điểm khác nhau duy nhất giữa hàm fprintf ( ) và prinf( ) là thiết bị ra.

3.3. Các lệnh nhập dữ liệu

3.3.1. Chuỗi điều khiển

Chuỗi điều khiển sử dụng trong các lệnh nhập dữ liệu gồm các ký tự đặc tả chuyển dạng, mỗi đặc tả thường có một đối tương ứng.

3.3.1.1. Đặc tả chuyển dạng

Một đặc tả chuyển dạng có thể viết tổng quát như sau:

%[*][d...d]ký tự chuyển dạng

- Dấu % là thành phần bắt buộc của một đặc tả chuyển dạng.
- Dấu * có thể xuất hiện hoặc vắng mặt. Việc có mặt của dấu * nói lên rằng trường vào vẫn được dò đọc bình thường nhưng giá trị của nó bị bỏ qua (không được lưu vào bộ nhớ). Vì vậy mà đặc tả chứa dấu * không có đối tương ứng.
- Ký hiệu d...d là một dãy số xác định chiều dài cực đại của trường vào, ý nghĩa của nó sẽ được giải thích rõ hơn trong mục sau.

3.3.1.2 Dòng vào và trường vào

a. Định nghĩa

- Dòng vào là một dãy ký tự liên tiếp nhau, kể cả các khoảng trắng (dấu cách, dấu tab \t, ký hiệu xuống dòng \n), trên thiết bị vào.
- Trường vào là một dãy ký tự khác khoảng trắng.

Ví dụ: ta xét dòng vào:

Abcd\t123.45 90

Trong dòng vào trên gồm 3 trường vào:

Abcd – độ dài của trường là 4

123.45 – độ dài của trường là 6

90 – độ dài của trường là 2

b. Giải thích ý nghĩa của tham số 'd...d'

Để hiểu ý nghĩa của tham số d...d ta xét trong hai trường hợp sau:

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

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

Thí dụ: Xét chương trình:

Chương trình 3.4

```
#include “stdio.h”
#include “conio.h”
void main()
{
        int i;
        float x;
        char c;
        printf(“\n nhap cac gia tri:\n”);
        scanf(“%3d %4f %c”, &i, &x, &c);
        printf(“in cac gia tri:%3d %4f %c”,i,x,c);
        getch();
}
```

Kết quả chương trình:

Nếu ta nhập vào: 119   34.7892   X

Thì kết quả hiển thị trên màn hình là:

????????????????????

---

c. Các ký tự chuyển dạng

Trong các lệnh nhập dữ liệu, ký tự chuyển dạng xác định cách dò đọc các ký tự trên dòng vào cũng như phương pháp chuyển dịch thông tin đọc được trước khi gán nó cho các địa chỉ tương ứng. Có hai cách dò đọc: Đọc theo trường vào, khi đó các khoảng trắng được coi là dấu phân cách giữa các trường và bị bỏ qua. Cách này áp dụng với hầu hết các loại ký tự chuyển dạng. Cách thứ hai đó là cách đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét như các ký tự khác. Cách này chỉ xảy ra với các chuyển dạng sau: %c, %[…], %[^…].

- Xét ký tự chuyển dạng […], trong hai dấu [] là một dãy ký tự. Các ký tự trên dòng vào sẽ được đọc lần lượt cho đến khi gặp một ký tự không thuộc dãy ký tự trong hai dấu [].

- Xét ký tự chuyển dạng [^…], sau dấu ^ là một dãy ký tự trong hai dấu []. Các ký tự trên dòng vào sẽ được đọc lần lượt cho đến khi gặp một ký tự thuộc dãy ký tự trong hai dấu []. Trong cả hai trường hợp này, khoảng trắng cũng được xét như các ký tự khác.

Dưới đây là bảng các ký tự chuyển dạng sử dụng trong các hàm nhập dữ liệu. Khác với các ký tự chuyển dạng trong các hàm xuất dữ liệu, ở đây không sử dụng đặc tả %i, %li, %g.

Bảng 3.15 Các ký tự chuyển dạng trong xuất dữ liệu

Định dạng	**Ký tự chuyển dạng**
Ký tự đơn (Single Character), có xét các khoảng trắng	c
Chuỗi (String)	s
Số nguyên có dấu (Signed decimal integer)	d
Số nguyên có dấu kiểu long	ld
Số thập phân có dấu chấm động (Floating point)	f hoặc e
Số thập phân có dấu chấm động kiểu long	lf hoặc le
Số nguyên không dấu (Unsigned decimal integer)	u
Số nguyên không dấu kiểu long	lu
Số thập lục phân không dấu (Unsigned hexadecimal integer)	x
Số thập lục phân không dấu kiểu long	lx
Số bát phân không dấu (Unsigned octal integer)	o
Số bát phân không dấu kiểu long	lo

+) Lưu ý: Để việc nhập dữ liệu được chính xác, thì số đối số, số đặc tả và số trường vào phải bằng nhau và phải phù hợp như đã nêu trên.

3.3.2. Dòng vào chuẩn (Stdin)

3.3.2.1. Khái niệm

Như phần trên ta đã có định nghĩa về dòng vào, thì ở đây ta xét đến dòng vào chuẩn (Stdin – Standard input), là nơi lưu trữ dữ liệu để cung cấp cho các hàm nhập dữ liệu **scanf( ), gets( ) và getchar( )**.

3.3.2.2. Một số chú ý

Chú ý 1: Xét câu lệnh:

int ch; **char** c[15];

ch= **getchar**( );

gets(c);

- Giả sử ta nhập ký tự X rồi bấm phím Enter, thì ch = ‘A’, ký tự ‘\n’ vẫn còn lại trong Stdin và nó sẽ làm trôi hàm gets( ), (hoặc getchar( )) sau đó.

- Nếu chỉ bấm phím Enter thì ch = ‘\n’ và ‘\n’ bị loại khỏi Stdin.

Chú ý 2: Hàm **scanf( )** cũng để lại ký tự ‘\n’ trong stdin nên nó sẽ làm trôi hàm **gets( )** và **getchar( )** sau đó. Do vậy cần phải loại bỏ ký tự này ra khỏi Stdin thì các hàm mới có thể hoạt động đúng được. Đối với hàm **scanf( )** ta có thể thêm đặc tả %*c vào cuối chuỗi điều khiển (đặc tả này không có đối tương ứng). Hoặc cách thông dụng cho cả hai trường hợp trên đó là sử dụng hàm **fflush** để làm sạch s**tdin**.

Thí dụ: ***Chương trình 3.5 /*Nhập dữ liệu từ bàn phím và hiển thị kết quả lên màn hình*/***

```
#include “stdio.h”
#include “conio.h”
void main( )
{
char HT[25],NS[20];
int t;
printf(“\nnhap ho ten”);
gets(HT);
printf(“\n nhap tuoi:”);
scanf(“%d%*c”,&t);
printf(“\nnhap noi sinh:”);
gets(NS);
clrscr();
printf(“\ninketqua:\nhoten:%s\ntuoi:%d\nNoisinh:%s”,&HT,&t,&NS);
getch();
}
```

- Giả sử ta nhập dữ liệu cho các lệnh nhập là: Tran Minh   22   Ha Noi

Thì kết quả hiển thị lên màn hình là:

```
============================================================
in ket qua:
ho ten: Tran Minh
tuoi:22
Noi sinh: Ha Noi
============================================================
```

- Nếu ta bỏ đặc tả %*c trong câu lệnh **scanf( )** đi, thì lệnh **gets(NS)** bị trôi và ta không thể nhập dữ liệu cho lệnh này. Do đó, kết quả hiển thị sẽ là:

```
============================================================
in ket qua:
ho ten: Tran Minh
```

tuoi:22

Noi sinh:

- Hoặc bây giờ, ta bỏ đặc tả %*c trong câu lệnh scanf đi và thêm lệnh:

fflush(stdin);

vào trước câu lệnh **gets(NS)**; thì ký tự '\n' sẽ bị khử và kết quả ra đúng.

3.3.3. Các hàm nhập dữ liệu.

3.3.3.1. Hàm scanf( )

- Dạng hàm: Khuôn dạng chung của hàm scanf( ) như sau:

scanf("Chuỗi điều khiển",arg1, arg2, …argN);

- Ý nghĩa: Hàm **scanf( )** được sử dụng để nhập dữ liệu. Cấu trúc hàm scanf**()** giống như hàm **printf()**. Tuy nhiên, chức năng của hai hàm này là trái ngược nhau. Dưới đây ta xét một số điểm khác biệt giữa hai hàm này:

Hàm **printf()** dùng các tên biến, hằng số, hằng chuỗi và các biểu thức, nhưng scanf() sử dụng những con trỏ tới các biến. Một con trỏ tới một biến là một mục dữ liệu chứa đựng địa chỉ của nơi mà biến được cất giữ trong bộ nhớ. Khi sử dụng scanf() cần tuân theo những quy tắc cho danh sách tham số:

- Nếu ta muốn nhập giá trị cho một biến có kiểu dữ liệu cơ bản, gõ vào tên biến cùng với ký hiệu **&** trước nó.
- Khi nhập giá trị cho một biến thuộc kiểu dữ liệu dẫn xuất (không phải thuộc bốn kiểu cơ bản char, int, float, double), không sử dụng **&** trước tên biến.

Thí dụ:***Chương trình 3.6/* Chương trình chuyển từ mã ASCII thành ký tự*/***

```
#include "stdio.h"
#include "conio.h"
void main()
{
int gt;
printf("\n nhap ma ASCII (lon hon 40):");
scanf("%3d",&gt);
printf("\n Ma ASCII: %3d\n Ky tu tuong ung: %c",gt,gt);
getch();
}
```

Kết quả chương trình:

Giả sử ta nhập vào mã: 65

Thì kết quả hiển thị là:

```
===========================================================
Ma ASCII: 65
Ky tu tuong ung: A
```

==

3.3.3.2. Hàm gets( )

- Dạng hàm: **gets**(xâu);
- Ý nghĩa: Nhập một xâu ký tự từ bàn phím cho đến khi phím Enter được nhấn. Một số lưu ý khi sử dụng hàm **gets( )**:
- Đối với hàm **gets( ),** dữ liệu nhập vào sẽ được gán cho một biến con trỏ hoặc biến mảng thuộc kiểu char.

Ví dụ: khai báo biến:

char *s; /*biến con trỏ*/

char s[25]; /*biến mảng*/;

- Xâu nhập vào được bổ sung ký tự kết thúc '\0' và đặt vào vùng nhớ do biến con trỏ trỏ tới.
- Hàm **gets( )** cho phép nhập xâu ký tự có dấu cách còn trong **scanf( )** thì không. Ví dụ: Hàm **gets( )** cho phép nhập xâu: Nguyen Hai Anh, còn trong hàm **scanf( )** chỉ cho phép nhập: NguyenHaiAnh hoặc Nguyen_Hai_Anh.

3.3.3.3. Hàm getchar( )

- Dạng hàm: **getchar(ch)**;
- Ý nghĩa: Hàm **getchar( )** được dùng để nhập một ký tự đơn vào tại một thời điểm từ bàn phím.Trong hầu hết việc thực thi của C, khi dùng **getchar(),** các ký tự nằm trong vùng đệm cho đến khi người dùng nhấn phím xuống dòng. Vì vậy nó sẽ đợi cho đến khi phím Enter được gõ. Hàm **getchar( )** không có tham số. Hàm này trả về một giá trị kiểu ký tự.

Thí dụ: Câu lệnh c = getchar( );

Nhập ký tự O từ bàn phím, thì c = O.

Lưu ý: Như đã nhắc đến ở trên, khi sử dụng hàm nhập **scanf( )** hoặc **getchar( )** mà sau đó còn hàm **gets( )** hoặc **getchar( )**, thì phải thêm câu lệnh **fflush(stdin)** vào trước các hàm **gets( )** hoặc **getchar( )** này.

Chương trình 3.7 /*Hiển thị số nguyên hệ thập phân dưới dạng hexa và hệ bát phân*/

```
#include "stdio.h"
# include "conio.h"
void main( )
{
int  so_nguyen;
puts("Nhap vao mot so nguyen kieu int: ");
scanf("%d",&so_nguyen);
printf("\nDecimal=%d\nOctal=%o",so_nguyen,so_nguyen);
```

```
printf("\nHex=%x\n",so_nguyen);
getch( );
}
```

Kết quả hiển thị trên màn hình của chương trình là:

```
Nhap vao mot so nguyen kieu int: 234
Decimal = 234
Octal = 352
Hex = ea
```

==

3.4. Một số hàm xuất/nhập bổ sung

Trong các phần trên, chúng ta đã xét các hàm vào/ra chuẩn cơ bản. Tất cả các hàm đó đều nằm trong tệp **"stdio.h"**. Trong mục này sẽ xét các hàm vào/ra bổ sung, các hàm này được khai báo trong tệp **"conio.h"**.

3.4.1. Các hàm xuất dữ liệu bổ sung

Các hàm xuất dữ liệu bổ sung được trình bày dưới đây như là hàm putch, cputs, và hàm cprintf có chức năng như các hàm xuất dữ liệu chuẩn đã trình bày ở trên. Nhưng điểm khác biệt giữa chúng sẽ được trình bày dưới đây:

3.4.1.1. Hàm cprintf( )

+) Dạng hàm: **cprintf**("chuỗi điều khiển",arg1,arg2…,argN);

+) Ý nghĩa: Xuất ra một xâu văn bản đã định dạng lên màn hình hiển thị dưới dạng đã được xác định bởi "chuỗi điều khiển". Các đối số được định nghĩa là arg1, …argN. Danh sách đối số (argument list) bao gồm các hằng, biến, biểu thức hay hàm và được phân cách bởi dấu phẩy.

Ta thấy cấu trúc câu lệnh cũng như ý nghĩa của câu lệnh **cprintf( )** hoàn toàn giống với lệnh **printf( )**. Điểm khác nhau duy nhất giữa hai hàm này đó là hàm **cprintf( )** cho phép hiển thị các ký tự màu, còn hàm **printf( )** thì không thể.

3.4.1.2. Hàm cputs( )

+) Dạng hàm: **cputs**("xâu");

+) Ý nghĩa: Hàm **cputs( )** in ra xâu ký tự lên màn hình và cũng có khả năng hiển thị màu với đặc tính màu được định nghĩa thông qua hàm **textcolor( )**. Đây cũng là điểm khác biệt giữa hàm **cputs( )** với hàm **puts( )**.

+) Thí dụ: Câu lệnh:

```
textcolor (RED);
cputs("Lap trinh C cho ĐTVT");
```

Câu lệnh trên sẽ in lên màn hình xâu ký tự "Lap trinh C cho ĐTVT". Màu của các ký tự là màu đỏ, đã được xác định trong lệnh **textcolor**(RED)

3.4.1.3. Hàm putch( ):

+) Dạng hàm: **putch**(int ch);

+) Ý nghĩa: Xuất một ký tự lên cửa sổ văn bản trên màn hình.Tham số của hàm là một giá trị số nguyên biểu thị số thứ tự của mã ASCII. Kết quả của hàm là một ký tự ứng với số thứ tự của mã ASCII.

Hàm **putch( )** cho phép hiển thị ký tự màu với định nghĩa màu thông qua hàm **textcolor( )**. Đây cũng là điểm khác biệt giữa hàm **putch( )** với hàm **putchar( )**.

+) Thí dụ: Câu lệnh: ch = **putch** (65);

Kết quả trả về của hàm là gán ký tự tương ứng với số thứ tự của mã ASCII cho biến kiểu char ch. Tức là ch = 'A'

Sau đây ta sẽ xét một chương trình hiển thị số nguyên dưới dạng số thập lục phân và bát phân. Ta đặt trước mỗi lệnh xuất một lệnh tạo màu **textcolor( )** để tạo màu cho các ký tự và so sánh giữa các hàm **printf( ), cprintf( ), puts( ), cputs( ).**

Chương trình 3.8/* Chương trình hiển thị một số nguyên dưới dạng số bát phân, thập lục phân và có hiển thị màu cho các ký tự.*/

```
include "stdio.h"
include "conio.h"
void main()
{
  int a; char ch;
  textcolor(RED); /* = textcolor(4);:mau do*/
  printf("nhap vao mot so nguyen:");
  scanf("%d",&a);
  textcolor(1);
/* = textcolor(BLUE);:mau xanh da troi*/
cprintf("\n\rket qua:\n\rhe 10:%d\n\rhe 8:%o\n\rhe 16:%x",a,a,a);
  textcolor(RED); /* = textcolor(4);:mau do*/
  puts("\nnhap mot ky tu:");
  scanf("%c",&ch);
  textcolor(GREEN);
/* = textcolor(2);:mau xanh la cay*/
  cputs("in ky tu:");
  textcolor(MAGENTA); /* = textcolor(5);:mau tim*/
  putch(ch);
  cputs("\n\rBam mot phim bat ky de thoat:");
  getch();
}
```

Kết quả chương trình:

Giả sử ta nhập vào một số nguyên là 40 và ký tự là 'D'

Kết quả hiển thị trên màn hình sẽ là:

```
==========================================================
Nhap vao mot so nguyen: 40
Ket qua:
He 10: 40
He 8:
He 16:
Nhap mot ky tu:
In ky tu:D
Bam mot phim bat ky de thoat:
==========================================================
```

3.4.2. Các hàm nhập dữ liệu bổ sung

Các hàm nhập dữ liệu bổ sung như hàm cscanf, hàm getch, hàm getche sẽ được trình bày rõ trong mục này.

3.4.2.1. Hàm cscanf( )

+) Dạng hàm: **cscanf**("chuỗi điều khiển",arg1…argN);

+) Ý nghĩa: Nhập dữ liệu từ bàn phím. Cấu trúc lệnh và chức năng của hàm **cscanf( )** giống như hàm **scanf( )**. Giữa hai hàm này chỉ có điểm khác biệt sau:

- Nếu sử dụng hàm **scanf( )**, thì khi nhập dữ liệu , bạn phải bấm phím Enter để kết thúc. Vì vậy, nếu có sai sót có thể sử dụng các phím mũi tên phải, trái di chuyển con nháy đến chỗ sai để sửa. Còn nếu dùng hàm **cscanf( )** thì tùy theo khuôn dạng của dữ liệu nhập vào, máy sẽ tự động thực hiện đến câu lệnh tiếp theo mà không cần phải bấm phím Enter khi kết thúc. Do đó, nếu có sai sót trong khi nhập thì cũng không sửa được.

+) Thí dụ: Câu lệnh:

```
int a;
cscanf("%2d",&a);
```

3.4.2.2. Hàm getch( ):

+) Dạng hàm: **getch( );**

+) Ý nghĩa: Nhận một ký tự từ bộ đệm bàn phím nhưng không cho hiển thị lên màn hình.

Nếu trong bộ đệm bàn phím có sẵn ký tự thì hàm nhận một ký tự trong đó. Còn nếu bộ đệm rỗng thì máy sẽ chờ nhập vào một ký tự và hàm nhận ngay ký tự mới nhập đó.

Thực chất, hàm **getch( )** được sử dụng ở cuối một chương trình, có tác dụng dừng màn hình để xem kết quả. Do đó nó được sử dụng hầu hết trong mọi chương trình dù đơn giản hay phức tạp.

3.4.2.3. Hàm getche( )

+) Dạng hàm: **getche();**

+) Ý nghĩa: Nhập một ký tự từ bàn phím và cho hiển thị lên màn hình. Hàm này thực chất cũng giống như hàm **getch( )**, chỉ khác là có hiển thị ký tự nhập vào lên màn hình.

3.4.3. Hàm kbhit( )

+) Dạng hàm: **kbhit( );**

+) Ý nghĩa: Kiểm tra bộ đệm bàn phím. Hàm có giá trị khác 0 nếu bộ đệm bàn phím khác rỗng, và có giá trị bằng 0 nếu bộ đệm bàn phím rỗng.

Chú ý: Một vấn đề đặt ra là cần phân biệt xem một ký tự khi được nhập, thì khi nào nó được gửi vào stdin? Khi nào được gửi vào bộ đệm?

Cách phân biệt như sau:

- Nếu gõ phím khi máy dừng chờ trong các hàm scanf( ), gets( ) và getchar( ) thì ký tự được gửi vào stdin.
- Nếu gõ phím trong các trường hợp khác thì ký tự gửi vào bộ đệm.

+) Thí dụ: Xét chương trình báo thức, chuông reo cho tới khi bấm một phím bất kỳ.

Chương trình 3.9

```
#include “conio.h”
main()
{
  reochuong:        /* khai bao nhan*/
  putch(7);         /*Tao chuong reo*/
  if(!kbhit()) goto reochuong;
/* kiem tra khi chua co phim nao duoc bam thi tro lai cau lenh co nhan.*/
}
```

3.4.4. Xóa màn hình và di chuyển con trỏ

Ngoài các lệnh thao tác nhập và xuất dữ liệu, ở đây chúng ta sẽ xét đến hai lệnh cũng rất phổ biến trong các chương trình.

- Lệnh xóa màn hình: **clrscr( );**
- Lệnh di chuyển con trỏ màn hình đến vị trí có tọa độ (x,y):

gotoxy(x,y**);**

Trong đó, x là số hiệu cột nhận giá trị từ 1 đến 80, y là số hiệu dòng và nhận giá trị từ 1 điến 25.

Ví dụ: Câu lệnh **gotoxy**(40,1); Sẽ đưa con trỏ màn hình đến vị trí giữa của dòng 1 (40,1).

Bài tập chương

Bài 1. Viết chương trình tính các giá trị của phép OR, NOR, XOR và N-XOR của hai số nguyên thập phân và chuyển các giá trị kết quả sang hệ bát phân và hệ Hexa.

Bài 2. Sửa lỗi trong chương trình sau:

```
#include “stdio.h”
#include “conio.h”
void main( )
{
char lop[25],truong[40];
int si_so;
printf(“\nnhap ten lop:”);
gets(lop);
printf(“\n nhap si so lop:”);
scanf(“%d”,&si_so);
printf(“\nnhap ten truong:”);
gets(NS);
clrscr();
printf(“\nin ket qua:\nlop:%s\nsi so:%d\nTruong:%s”,
&lop,&si_so,&truong);
getch();
}
```

Bài 3. Viết chương trình tạo chuông reo cho đến khi bấm phím T.

Bài 4. Viết chương trình hiển thị các dòng chữ màu sau:

KY THUAT LAP TRINH C

TRONG DIEN TU VIEN THONG

Chương 4   CÁC LỆNH ĐIỀU KHIỂN

4.1. Giới thiệu chung

Trong lập trình có cấu trúc, mỗi chương trình bao gồm nhiều câu lệnh được thực hiện một cách tuần tự theo thứ tự mà chúng được viết (Cấu trúc tuần tự). Tuy nhiên trong hầu hết các trường hợp ta cần điều khiển thứ tự thực hiện các câu lệnh theo một trật tự nào đó để có thể giải quyết được các vấn đề mà bài toán đặt ra. Công cụ giúp cho lập trình viên có thể làm được điều này chính là các lệnh điều khiển. Các lệnh điều khiển trong lập trình C dùng để tổ chức ra các loại cấu trúc trong chương trình (*cấu trúc lặp, rẽ nhánh*). Về mặt công dụng có thể chia các lệnh điều khiển thành 3 nhóm chính:

- Nhóm các lệnh rẽ nhánh (**if, if…else, switch**)
- Nhóm các lệnh dùng để tổ chức chu trình (**for, while, do…while**)
- Nhóm các lệnh điều khiển khác (**break, continue, return, goto**)

4.2. Câu lệnh if

Cấu trúc rẽ nhánh là một cấu trúc được dùng rất phổ biến trong các ngôn ngữ lập trình nói chung. Câu lệnh **if** cho phép lựa chọn một trong hai nhánh tùy thuộc vào giá trị của biểu thức điều kiện là đúng (TRUE) hay sai (FALSE) hoặc khác không hay bằng không.

4.2.1. Cú pháp câu lệnh if

Câu lệnh **if** quyết định một trong hai lựa chọn dựa trên giá trị đúng (TRUE = 1) hay sai (FALSE = 0) của biểu thức. Cú pháp của câu lệnh **if** như sau:

if ***(****<bieu_thuc_dieu_kien>****)***

*<cong_viec>****;***

Chú ý:

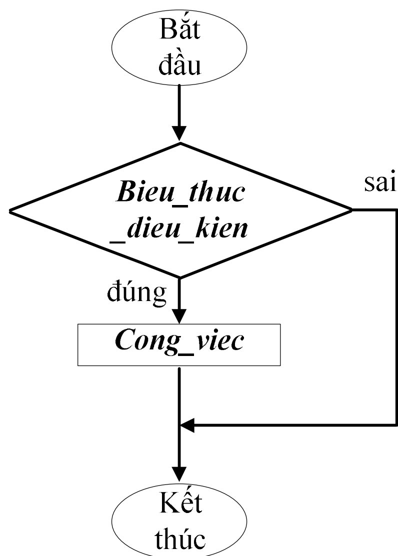
- *bieu_thuc_dieu_kien*: có thể là biểu thức bất kỳ (biểu thức nguyên, biểu thức thực, quan hệ hay logic…) miễn sao cứ trả về giá trị bằng 0 (ứng với trường hợp sai) hoặc khác không (ứng với trường hợp đúng).

- *cong_viec*: nếu có nhiều câu lệnh để thực hiện công việc ta phải đưa chúng vào trong khối lệnh “**{}**”.

- *bieu_thuc_dieu_kien* bao giờ cũng phải được đặt trong hai dấu “**()**”

4.2.2. Lưu đồ

Lưu đồ của câu lệnh **if** như sau:



Hình 4.7: Lưu đồ cấu trúc câu lệnh if

4.2.3. Giải thích lưu đồ

Như hình 4.1, sau khi kiểm tra tính chính xác của *bieu_thuc_dieu_kien* thì câu lệnh **if** thực hiện một trong 2 hướng, nếu đúng thực hiện tiếp *cong_viec* còn nếu sai tự động thoát ra.

Thí dụ

Xét thí dụ với hình 4.2 ta có thể giải thích ngắn gọn như sau:

Với biểu thức điều kiện if((x==4)&&(y>2)||(ch!='x')) ta phân tích theo từng bước như sau:

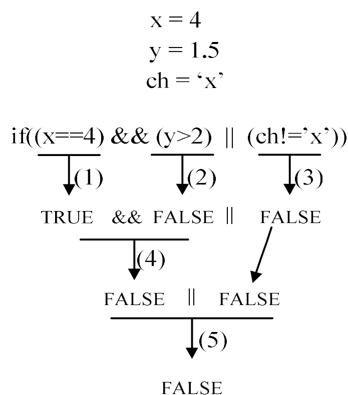
(1): kiểm tra điều kiện x==4 là đúng hay sai, với giả thiết đã cho x = 4, do vậy kết quả trả về của bước này là đúng (TRUE).

(2): kiểm tra điều kiện y>2, với giả thiết cho y = 1.5, do vậy kết quả trả về của bước này là sai (FALSE).

(3): kiểm tra điều kiện ch!='x', sẽ cho kết quả sai (FALSE).

(4): kiểm tra điều kiện đúng (TRUE) && sai (FALSE), kết quả trả về là sai (FALSE).

(5): kiểm tra điều kiện sai(FALSE) || sai(FALSE), kết quả trả về là sai (FALSE).



Hình 4.8. Ví dụ của lệnh if

Thí dụ :Viết chương trình biến đổi thập phân sang nhị phân

Chương trình trong ví dụ này biến đổi một giá trị số nguyên thập phân sang dạng nhị phân 8 bit không dấu. ta có thể giải thích một cách ngắn gọn thông qua sơ đồ hình vẽ 4.3 dưới đây.

Thí dụ ta có chuỗi nhị phân 1110 1010 biểu diễn nhị phân này có giá trị thập phân là 234. Vậy để có thể hiển thị từng *bit* nhị phân lên màn hình ta dùng thuật toán che *bit* như sau:

(1): thực hiện phép "&" với 0x80 để xác định *bit* có trọng số 2^7.

(2): thực hiện phép "&" với 0x40 để xác định *bit* có trọng số 2^6.

(3): thực hiện phép "&" với 0x20 để xac định bit có trọng số 2^5.

(4): thực hiện phép "&" với 0x10 để xác định bit có trọng số 2^4.

(5): thực hiện phép "&" với 0x08 để xác định *bit* có trọng số 2^3.

(6): thực hiện phép "&" với 0x04 để xác định *bit* có trọng sô 2^2.

(7): thực hiện phép "&" với 0x02 để xác định *bit* có trọng số 2^1.

(8): thực hiện phép "&" với 0x01 để xác định được bit LSB (b*it* có trọng số thấp nhất.

Bảng 4.16: Các bước thực hiện của thuật toán che bit

Trọng số	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Số cần hiển thị	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**
(1): & mặt nạ bit "0x80"	**1**	0	0	0	0	0	0	0
(2): & mặt nạ bit "0x40"	0	**1**	0	0	0	0	0	0
(3): & mặt nạ bit "0x20"	0	0	**1**	0	0	0	0	0
(4): & mặt nạ bit "0x10"	0	0	0	**1**	0	0	0	0
(5): & mặt nạ bit "0x08"	0	0	0	0	**1**	0	0	0
(6): & mặt nạ bit "0x04"	0	0	0	0	0	**1**	0	0
(7): & mặt nạ bit "0x02"	0	0	0	0	0	0	**1**	0
(8): & mặt nạ bit "0x01"	0	0	0	0	0	0	0	**1**

Kết quả của câu lệnh & với từng mặt nạ *bit* sẽ hiển thị “0” nếu là sai (FALSE) và “1” nếu là đúng (TRUE). Kết quả hiển thị sẽ là số 01011001. Xét trong chương trình dưới đây.

Chuong_trinh_ 4.1

```
/* chuong trinh dung ky thuat che bit de chuyen doi tu thap phan sang nhi phan */
#include "stdio.h"
#include "conio.h"
#include "string.h"
void main()
{
  int i;
  char ch;
LapLai:
  clrscr();
  printf("nhap gia tri so thap phan can chuyen doi (0-255):");
  scanf("%d%c",&i);
  if(i>=0 && i<=255)
  {
       printf("so nhi phan la: ");
/*(1)*/           if (i & 0x80) printf("1"); else printf("0");
/*(2)*/           if (i & 0x40) printf("1"); else printf("0");
/*(3)*/           if (i & 0x20) printf("1"); else printf("0");
(4)          if (i & 0x10) printf("1"); else printf("0");
(5)          if (i & 0x08) printf("1"); else printf("0");
(6)          if (i & 0x04) printf("1"); else printf("0");
(7)          if (i & 0x02) printf("1"); else printf("0");
(8)          if (i & 0x01) printf("1"); else printf("0");
  }

     else     printf("\nso nhap khong hop le");
     printf("\nban co muon thuc hien tiep khong (c/k?):");
     scanf("%c",&ch);
     if(ch =='C'|| ch=='c')
     goto LapLai;
getch();
}
```

Kết quả chương trình 4-1:

```
TC
nhap gia tri so thap phan can chuyen doi (0-255):234
so nhi phan la: 11101010
ban co muon thuc hien tiep khong (c/k?):k
```

4.3. Câu lệnh if.. else

4.3.1. Cú pháp câu lệnh if … else (dạng đầy đủ)

if (<bieu_thuc_dieu_kien>)

<cong_viec_1>;

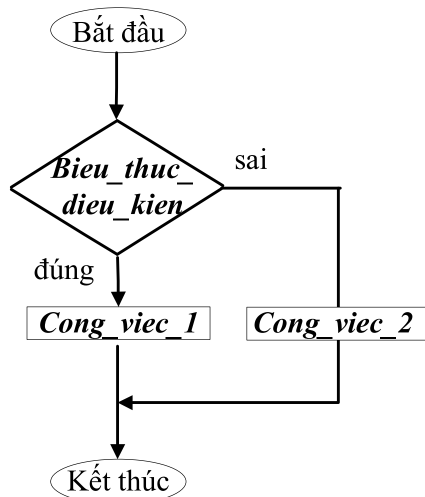
else

<cong_viec_2>;

Chú ý: ***bieu_thuc_dieu_kien*** và ***cong_viec*** cũng tương tự như trong câu lệnh ***if***, ở đây, khác ở chỗ nếu kết quả kiểm tra ***bieu_thuc_dieu_kien*** mà sai thì sẽ không thoát khỏi ngay mà nhảy tới thực hiện ***cong_viec_2***, sau đó mới thoát. Một chú ý nữa là trước else luôn phải có dấu "**;**"

4.3.2. Lưu đồ

Lưu đồ của câu lệnh ***if…else*** như sau:



Hình 4.9. Lưu đồ của câu lệnh if...else

4.3.3. Giải thích lưu đồ

Dữ liệu vào sẽ được kiểm tra so với ***bieu_thuc_dieu_kien***, nếu kết quả trả về là true, chương trình tiếp tục thực hiện ***cong_viec_1*****,** ngược lại, nếu cho kết quả false, chương trình sẽ thực hiện ***cong_viec_2*** sau đó thoát ra.

Một số chú ý

- cong***_viec_1*** và ***cong_viec_2*** có thể là một câu lệnh hay một khối lệnh.

- Các lệnh phía sau ***cong_viec_2*** không phụ thuộc vào điều kiện.

- Ngoài hai dạng câu lệnh rẽ nhánh trên ta xét thêm trường hợp câu lệnh rẽ nhánh ***if*** mở rộng cho n trường hợp có cấu trúc và lưu đồ như sau:

Cấu trúc:

if <(bieu_thuc_dieu_kien_1)>

cong_viec_1;

else if <(bieu_thuc_dieu_kien_2)>

cong_viec_2;

else if <(bieu_thuc_dieu_kien_3)>

cong_viec_3;

.

.

.

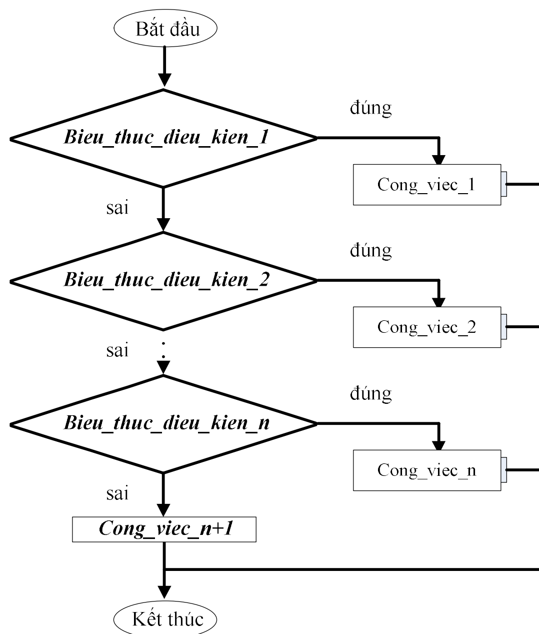
else if <(bieu_thuc_dieu_kien_n)>

cong_viec_n;

else

cong_viec_ n+1;

lược đồ:



Hình 4.10. Lược đồ câu lệnh *if* mở rộng cho n trường hợp

4.4. Biểu thức điều kiện:

Turbo C còn có toán tử điều kiện là (**?:**). Phép toán điều kiện gồm các thành phần sau đây gọi là biểu thức điều kiện:

e1?e2:e3 hay <dieu_kien> ? <cong_viec_1> : <cong_viec_2>;

trong đó: <dieu_kien> là TRUE → sẽ thực hiện cong_viec_1;

<dieu_kien> là FALSE → sẽ thực hiện cong_viec_2;

Ví dụ: (i>=5)? 1: -1

Nếu i >= 5 thì kết quả trả về là 1

Nếu i < 5 thì kết quả trả về là -1

Người ta có thể dùng toán tử điều kiện thay thế cho một khối câu lệnh điều kiện

Ví dụ: Thay vì phải viết một khối lệnh điều kiện như sau:

```
…
if (dieu_kien)
  Z = cong_viec_1;
else
  Z = cong_viec_2;
…
```

Thì ta chỉ cần viết ngắn gọn lại như sau:

```
Z = dieu_kien ? cong_viec_1:cong_viec_2
```

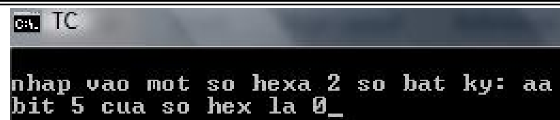
4.5. Một số ví dụ

Ví dụ 4-2**:** Ví dụ muốn kiểm tra xem bit thứ 5 của một số hexa nhập vào từ bàn phím làm bit "0" hay bit "1" ta có chương trình như sau:

Chương trình 4-2.

```
/* Vd 4-2 */
/* chuong trinh kiem tra xem bit 5 cua so hexa la bit "0" hay "1" */
#include "conio.h"
#include "stdio.h"
void main()
{
  unsigned char ch,kq;
  printf("\n nhap vao mot so hexa 2 so bat ky: ");
  scanf("%x", &ch);
  kq=ch & 0x10;
  if (kq==0x10) printf("\nbit 5 cua so hex la 1");
  else printf("\nbit 5 cua so hex la 0");
getch();
}
```

Kết quả chương trình 4-2:



4.6. Câu lệnh switch

Cấu trúc lựa chọn cho phép lựa chọn một trong nhiều trường hợp, câu lệnh ***switch*** cho phép ta làm được điều đó.

Câu lệnh ***switch*** làm việc giống như câu lệnh ***if*** mở rộng cho n trường hợp ở trên, chỉ khác một điều là ***switch*** luôn làm việc với biểu thức nguyên.

4.6.1. Cú pháp câu lệnh switch

```
switch(<bieu_thuc_nguyen>)
{
  case n1:
<cong_viec_1>;
break;
  case n2:
<cong_viec_2>;
break;
…
  case nk:
 <cong_viec_k>;
 break;
  [default
 <cong_viec_n+1>;
break;]
}
```

Trong đó ***ni*** là số nguyên, *hằng kí tự* hoặc *biểu thức hằng*. Các ***ni*** phải có giá trị khác nhau. Các câu lệnh nằm trong hai dấu "{}" được gọi là thân của ***switch***. Ta có thể mô tả hoạt động của ***switch*** như sau:

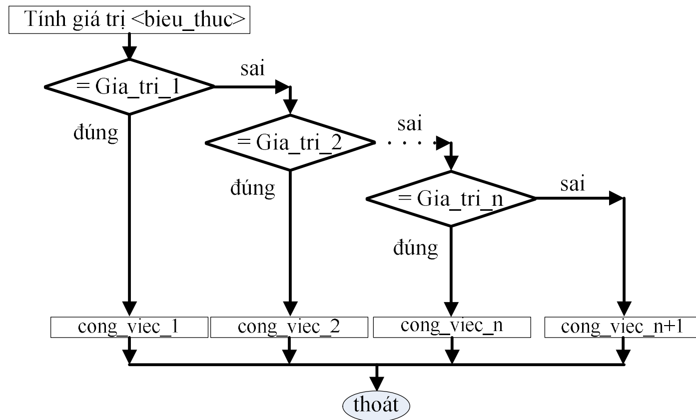
- Trước tiên máy sẽ xét giá trị ***bieu_thuc_nguyen***, tùy theo giá trị của biểu thức này mà có thể quyết định nhảy tới đâu.

- Nếu giá trị này bằng ***ni***, máy sẽ nhảy tới câu lệnh nằm sau nhãn ***case ni*** và bắt đầu thực hiện lệnh từ đó cho đến khi gặp một câu lệnh ***break, goto, return*** hoặc dấu kết thúc "***}***".

- Khi giá trị ***bieu_thuc_nguyen*** khác tất cả các ***ni***, i = 1÷ k thì sự hoạt động của ***switch*** lúc này phụ thuộc vào sự có mặt hay vắng mặt của ***default***. Nếu có mặt ***default*** thì các câu lệnh nằm sau ***default*** sẽ được thực hiện, ngược lại máy sẽ thoát khỏi ***switch*** và bắt đầu thực hiện lệnh nằm sau dấu "***}***" của thân ***switch***.

4.6.2. Lưu đồ thuật toán câu lệnh switch

Lưu đồ thuật toán của câu lệnh ***switch*** như sau:



Hình 4.11. Lưu đồ thuật toán của câu lệnh switch

4.6.3. Giải thích lưu đồ

- Tính giá trị của biểu thức trước.
- Nếu giá trị của biểu thức bằng giá trị 1 thì thực hiện công việc 1 rồi thoát.
- Nếu giá trị của biểu thức khác giá trị 1 thì so sánh với giá trị 2, nếu bằng giá trị 2 thì thực hiện công việc 2 rồi thoát.
- Tiếp tục so sánh tới giá trị n.
- Nếu tất cả các phép so sánh trên đều sai thì thực hiện công việc mặc định của trường hợp default.

4.6.4. Một số chú ý

- Máy sẽ thoát khỏi câu lệnh ***switch*** khi nó gặp một câu lệnh ***break*** hoặc dấu ngoặc đóng "***}***" chỉ sự kết thúc câu lệnh ***switch***. Do đó các câu lệnh ***break*** là không thể vắng mặt mỗi khi một nhánh nào đó đã được lựa chọn.
- Trong thân của lệnh ***switch*** ta cũng có thể sử dụng lệnh ***goto*** để nhảy tới một câu lệnh bất kỳ bên ngoài ***switch***.

Khi ***switch*** nằm trong thân của một hàm nào đó, ta cũng có thể sử dụng một lệnh ***return*** trong thân của ***switch*** để thoát khỏi hàm đó.

- Biểu thức trong switch() phải có kết quả là giá trị kiểu số nguyên (int, char, long, …).
- Các giá trị sau case cũng phải là kiểu số nguyên.
- Không bắt buộc phải có default.

4.6.5. Một số ví dụ

Ví dụ 4-3: Mã màu 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 trong bảng 4.1. dưới đây:

Bảng 4.17. Hệ thống mã màu điện trở

Digit	Màu	Hệ số nhân	Digit	Màu	Hệ số nhân
	Bạc nhũ	0.01	4	Vàng	10k

	Vàng nhũ	0.1	5	Xanh	100k
0	Đen	1	6	Lam	1000k = 1M
1	Nâu	10	7	Tím	10M
2	Đỏ	100	8	Xám	
3	Da cam	1000 = 1k	9	Trắng	

Chương trình 4-3 sẽ sử dụng 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 4-3:

```
//chuong trinh 4-3
//doc mau dien tro thong qua cac gia tri nhap vao tu ban phim
//***************
#include "stdio.h"
#include "conio.h"
int main(void)
{
  unsigned int colour;
  printf("nhap gia tri mau tu 0 toi 9: ");
  scanf("%u", &colour);
  printf("mau cua dien tro la: ");
  switch(colour)
  {
  case 0 : printf("den"); break;
  case 1 : printf("nau"); break;
  case 2 : printf("do"); break;
  case 3 : printf("cam"); break;
  case 4 : printf("vang"); break;
  case 5 : printf("xan"); break;
  case 6 : printf("lam"); break;
  case 7 : printf("tim"); break;
  case 8 : printf("xam"); break;
  case 9 : printf("trang"); break;
  }
getch();
}
```

Kết quả chương trình 4-3:

```
TC
nhap gia tri mau tu 0 toi 9: 6
mau cua dien tro la: lam_
```

4.7. Câu lệnh for

Trong nhiều trường hợp để giải quyết được vấn đề đưa ra của một bài toán yêu cầu lặp lại nhiều lần một công việc trong cùng một chương trình, vòng lặp ***for*** cho phép chấp hành một khối mã dành cho một chức năng điều khiển đã đặt trước.

4.7.1. Cú pháp câu lệnh for

cú pháp tổng quát của câu lệnh ***for*** như sau:

for (starting condition; test condition; operation)

{

Statement block

}

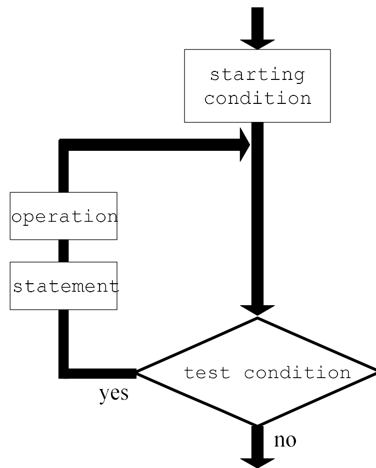
Trong đó:

starting condition	Khởi tạo bộ đếm; là giá trị bắt đầu của vòng lặp, nó là một lệnh gán để thiết lập biến điều khiển của vòng lặp trước khi bắt đầu vòng lặp, câu lệnh này chỉ được thực thi một lần.
test condition	Điều kiện kiểm tra; là một biểu thức quan hệ xác định khi nào vòng lặp sẽ kết thúc; Nếu điều kiện kiểm tra này là đúng (TRUE) thì vòng lặp sẽ được tiếp tục chấp hành
operation	Định lại giá trị tham số; Định nghĩa cách thay đổi giá trị của biến điều khiển vòng lặp (thông thường, biến này sẽ tăng hoặc giảm khi giá trị thiết lập tại thời điểm bắt đầu) mỗi khi vòng lặp được lặp lại

Lưu ý: ba phần này của vòng lặp ***for*** được phân cách nhau bởi dấu '**;**'. Các câu lệnh, phần thân của vòng lặp có thể là một lệnh đơn hoặc nhiều lệnh ghép (nhiều câu lệnh).

4.7.2. Lưu đồ thuật toán câu lệnh for

Lưu đồ của câu lệnh ***for***:



Hình 4.12. Lưu đồ biểu diễn câu lệnh for

4.7.3. Giải thích lưu đồ

Điều kiện ban đầu sẽ được kiểm tra bằng biểu thức kiểm tra điều kiện. Nếu kết quả kiểm tra điều kiện là TRUE thì thực hiện khối lệnh và các phép toán sau đó lại quay lại kiểm tra tiếp cho đến khi nào kết quả của phép kiểm tra điều kiện là FALSE thì thoát khỏi vòng lặp.

4.7.4. Một số chú ý

- starting condition: biểu thức này thông thường là một phép gán để khởi tạo giá trị ban đầu cho biến điều kiện.

- test condition: là một biểu thức kiểm tra điều kiện đúng, sai để dừng vòng lặp (biểu thức logic).

- operation: thông thường là một phép gán để thay đổi giá trị của biến điều kiện.

- Trong mỗi biểu thức có thể có nhiều biểu thức con, các biểu thức con được phân biệt bằng dấu phẩy.

4.7.5. Một số ví 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 một giá trị (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 += 500)

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, …, 400, 450.

c. for (; ;)

diễn tả một vòng lặp vô tận.

d. for (i = 2; i<=128; i 1=2)

bắt đầu với i = 2, mỗi lần chạy qua một vòng i lại được tăng lên thêm 2. Quá 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, 6…, 126, 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 thứ 3 của for ().

Ví dụ 4-4: cũng với bài toán chuyển đổi từ dạng thập phân sang nhị phân và hiển thị ra màn hình, nhưng tư sẽ sử dụng vòng lặp ***for*** để thực hiện bài toán.

Chương trình 4-4 là chương trình hoàn thiện của chương trình 4-1. Trước hết, chương trình che *bit* có giá trị cao nhất (0x80 hay 10000000b) và xác định nếu giá trị là TRUE (nghĩa 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" sẽ được hiển thị. Sau đó mặt nạ *bit* đổi chỗ về phía dưới một vị trí 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 được tiếp tục cho đến khi vòng lặp đạt tới *bit* có giá trị thấp nhất (cụ thể là 0x01 hay 00000001b). Sau đó vòng lặp sẽ kết thúc và chương trình sẽ dừng lại.

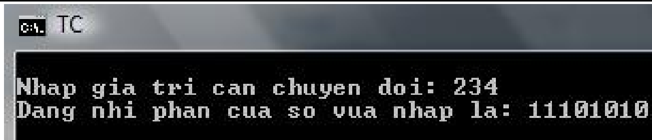
Vòng lặp for() sử dụng phép toán *bit >>=1* để di chuyển mặt nạ *bit* dịch đi một vị trí sang bên trái. Đa là một tương đương shorthand của bit = bit >> 1.

Chương trình 4-4.

```
/* chuong trinh 4-4*/
#include "stdio.h"
#include "conio.h"
int main(void)
{
int val,bit;
  printf("\nNhap gia tri can chuyen doi: ");
  scanf("%d",&val);
  if(val>=0 && val <=255)
  {
  printf("Dang nhi phan cua so vua nhap la: ");
        for (bit = 0x80; bit > 0; bit >>= 1)
            {
                 if(bit & val)  printf("1");
                 else       printf("0");
            }
  }
```

```
else printf("\n gia tri nhap khong thoa man");
getch();
}
```

Kết quả chương trình 4-4



4.8. Câu 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 (thỏa mãn) thì khối được thực hiện, nếu không nó sẽ thoát khỏi vòng.

4.8.1. Cú pháp câu lệnh while

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).

Ví dụ:

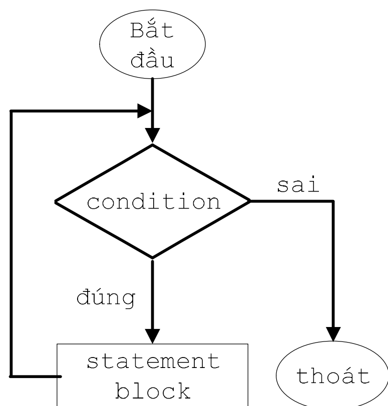
(a) while(i>10) :sẽ lặp lại khối lệnh chừng nào i còn lớn hơn 10.

(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.

4.8.2. Lưu đồ thuật toán

Lưu đồ thuật toán:



Hình 4.13. Lưu đồ thuật toán câu lệnh while()

4.8.3. Giải thích lưu đồ

Trong khi điều kiện (condition) vẫn có giá trị đúng (TRUE) thì vòng lặp vẫn được thực hiện.

statement block: được hiểu là một câu lệnh hay một khối lệnh để thực hiện một công việc nào đó.

condition : là điều kiện mà sẽ được máy tính kiểm tra trước.

4.8.4. Một số chú ý

Trong thân vòng lặp phải có ít nhất một câu lệnh làm thay đổi giá trị của biểu thức điều kiện để chương trình thoát khỏi vòng lặp sau một số lần lặp hữu hạn.

4.8.5. Một số ví dụ

Ví dụ 4-5. Viết đoạn chương trình in ra dãy số nguyên bất kỳ từ 1 tới n. Với n là số nguyên bất kỳ nhập từ bàn phím

Chương trình 4-5:

```
//chuong trinh 4-5
//nhap so nguyen n va in ra tu 1 toi n
/* *************** */
#include "conio.h"
#include "stdio.h"
int main()
{       int i,n;
  printf("\nNhap vao gi tri nguyen n: ");scanf("%d",&n);
  i = 1;
  printf("\Day can in tu 1 den %d la: ",n);
  while(i<=n)
  {


        printf("\%d  ",i);
        i = i++;
```

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

Kết quả chương trình 4-5:

```
TC
Nhap vao gi tri nguyen n: 10
Day can in tu 1 den 10 la: 1  2  3  4  5  6  7  8  9  10
```

4.9. Câu lệnh do … while

Lệnh do…while() tác động giống với lệnh while(), 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.

4.9.1. Cú pháp câu lệnh do … while

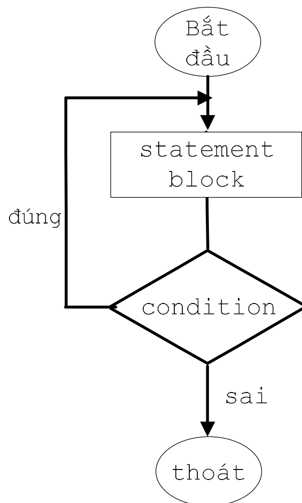
Cú pháp của câu lệnh như sau:

do <cong_viec>

while (<bieu_thuc_dieu_kien>)

4.9.2. Lưu đồ thuật toán

Lưu đồ thuật toán của câu lệnh do…while như sau:



Hình 4.14. Lưu đồ thuật toán của câu lệnh do...while

4.9.3. Giải thích lưu đồ

- statement block: một hay một khối lệnh thực hiện một công việc nào đó.
- Trước tiên công việc được thực hiện trước, sau đó mới kiểm tra biểu thức điều kiện.
- Nếu điều kiện sai thì thoát khỏi lệnh do…while
- Nếu điều kiện đúng thì thực hiện công việc rồi kiểm tra tiếp.

4.9.4. Ví dụ

Viết chương trình nhập vào mật khẩu và tên đăng nhập (Pass, Usename). Nếu mật khẩu là 'ĐTVT' và tên đăng nhập là K1 thì thoát khỏi chương trình.

Chương trình 4-6.

```
//CT 4-6 chuong trinh mat khau
#include "stdio.h"
#include "conio.h"
#include "string.h"
void main()
{
  char p[10],u[10];
  do
      {
       printf("pass:");
       scanf("%s",p);
       printf("\nNhap Usename:");
       scanf("%s",u);
      }
  while   ((strcmp(p,"DTVT")!=0)||(strcmp(u,"K1")!=0));

getch();
}
```

4.10. Nhóm các lệnh điều khiển khác

4.10.1. Câu lệnh break

Lệnh ngừng ***break*** được sử dụng để thoát khỏi một vòng lặp. Nó có thể được sử dụng với các vòng ***for(), while()*** và ***do…while()***.

Khi có nhiều vòng lặp (chu trình) lồng nhau, lệnh ***break*** có tác dụng ra khỏi vòng lặp (hoặc lệnh ***switch***) bên trong nhất chứa nó mà không cần kiểm tra điều kiện kết thúc vòng lặp. Nói một cách khác máy sẽ bỏ qua các câu lệnh còn lại để thoát khỏi vòng lặp.

Mọi câu lệnh ***break*** đều có thể thay thế bằng lệnh ***goto*** kèm theo nhãn để chuyển tới.

Xét ví dụ dưới đây: Câu lệnh break được dùng trong cấu trúc switch()

```
//chuong trinh 4-6
//doc mau dien tro
//thong qua cac gia tri nhap vao tu ban phim
//***************
```

```
#include "stdio.h"
#include "conio.h"
int main(void)
{
  unsigned int colour;
  printf("nhap gia tri mau tu 0 toi 9: ");
  scanf("%u", &colour);
  printf("mau cua dien tro la: ");
  switch(colour)
  {
  case 0 : printf("den"); break;
  case 1 : printf("nau"); break;
  case 2 : printf("do"); break;
  case 3 : printf("cam"); break;
  case 4 : printf("vang"); break;
  case 5 : printf("xan"); break;
  case 6 : printf("lam"); break;
  case 7 : printf("tim"); break;
  case 8 : printf("xam"); break;
  case 9 : printf("trang"); break;
  }
getch();
}
```

4.10.2. Câu lệnh 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()***. Ví dụ sau đây sẽ xuất ra các giá trị của i từ 0 đến 9 nhưng sẽ không in ra 5

```
for (i=0; i<10; i++)
{
        if (i ==5) continue;
        printf("gia tri cua i la: %d",i);
}
```

4.10.3. Câu lệnh return

Cấu trúc: return [<expression>]

Ngay lập tức thoát khỏi tại dòng lệnh thực thi và trả về giá trị của biểu thức expression.

Vì trong bất kỳ hàm nào cũng phải trả về một giá trị nên buộc phải có câu lệnh return

4.10.4. Câu lệnh nhảy không điều kiện goto

Câu lệnh nhảy không điều kiện ***goto*** là câu lệnh dùng để bẻ gãy tính tuần tự của một chương trình C.

a. Cú pháp: ***goto Nhan***;

b. Giải thích:

Nhan là một tên có dấu ':' đứng sau dùng để gán cho bất kỳ một câu lệnh nào trong chương trình. Ví dụ: ***Tiep: ++i***; thì ***Tiep*** là nhãn của câu lệnh ++***i***.

khi gặp câu lệnh này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khóa ***goto*** bỏ qua những câu lệnh đứng trước hoặc đứng sau câu lệnh nhảy này.

Chú ý:

- Câu lệnh ***goto*** và nhãn ***Nhan*** phải nằm trong cùng một hàm (không thể dùng để nhảy từ hàm này sang hàm khác).

- Không thể dùng lệnh ***goto*** để nhảy từ ngoài vào trong một khối lệnh. Nhưng việc nhảy từ trong ra ngoài một khối lệnh là hoàn toàn hợp lệ.

- Trong chương trình hạn chế dùng toán tử ***goto*** vì nó phá vỡ tính cấu trúc của chương trình.

Bài tập chương 4

Bài tập 1. Viết chương trình chuyển đổi số thập phân sang số nhị phân 4 bit

Bài tập 2. Viết chương trình so sánh hai số nhị phân 4 bit

Bài tập 3. Viết chương trình so sánh hai số nhị phân 8 bit

Bài tập 4. Viết chương trình nhập vào giá trị điện trở, hiển thị ra màn hình màu tương ứng

Bài tập 5. Viết chương trình nhập vào mầu của điện trở, hiển thị ra màn hình giá trị tương ứng.

Bài tập 6. Viết chương trình dùng câu lệnh if để phân loại sóng điện từ theo bước sóng như trong hình sau:

10 pm 1 nm 400 nm 700 nm 1 mm 100 mm

Tia gamma	Tia X	Tia tử ngoại	Ánh sáng thường	Tia hồng ngoại	Vi ba	Sóng vô tuyến

Bài tập 7. Nhập vào tần số của sóng và xác định bước sóng theo công thức:

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

Bài tập 8. Viết chương trình tính dòng qua một điốt với nguồn điện áp đặt vào theo công thức sau:

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

Với nhiệt độ là nhiệt độ phòng (ngầm hiểu là 27^0C hoặc 300K. Giá trị điện áp vào (V) và dòng bão hòa ngược (I_0) được nhập từ bàn phím.

Bài tập 9. Viết chương trình nhập vào thời gian kết thúc và số các khoảng thời gian cần có để xác định điện áp sụt trên điện trở ở những khoảng thời gian cho trước. (gợi ý: 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.Công thức tính điện áp sụt trên điện trở là:

Bài tập 10. Hãy viết chương trình xác định sụt áp trên một cuộn cảm và một điện trở ở trên một mạch RL nối tiếp với một bước nhảy điện áp ở đầu vào được đặt tại thời điểm t = 0. Nhập vào các giá trị của các 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 bản dưới đây:

Thông số	Cực tiểu	Cực đại
R	0 Ω	1 MΩ
L	0 H	100 mH
Thời gian kết thúc	1 µs	1s
Số khoảng thời gian	10	100

Chương 5 HÀM

5.1. Giới thiệu chung

Trong chương này chúng ta sẽ đi nghiên cứu xem khi nào thì nên dùng hàm để giải quyết một bài toán lập trình C, và ý nghĩa của việc dùng hàm để giải quyết vấn đề.

Nói chung các hàm sử dụng trong C để thực thi một chuỗi các lệnh liên tiếp. tuy nhiên cách sử dụng của các hàm thì không giống với các vòng lặp. Các vòng lặp có thể lặp lại một chuỗi các chỉ thị với các lần lặp liên tiếp nhau. Nhưng việc gọi một hàm sẽ sinh ra một chuỗi các chỉ thị được thực thi tại vị trí bất kỳ trong chương trình. Các hàm có thể được gọi nhiều lần khi có yêu cầu. Giả sử một phần của mã lệnh trong một chương trình dùng để tính bảng chân lý của phép toán AND hai phần tử. Nếu sau đó, trong cùng chương trình, việc tính toán như vậy cần phải thực hiện trên những phần tử khác, thay vì phải viết lại các chỉ thị giống như trên, một hàm có thể được viết ra để tính kết quả của bất kỳ hai phần tử nào. Sau đó, chương trình có thể nhảy tới hàm đó, để thực hiện việc tính toán (trong hàm) và trở về nơi nó đã được gọi.

Một điểm quan trọng nữa là các hàm thì dễ viết và dễ hiểu. Các hàm đơn giản có thể được viết để thực hiện các tác vụ xác định. Việc gỡ rối chương trình cũng dễ dàng hơn khi cấu trúc chương trình dễ đọc, nhờ vào sự đơn giản hóa hình thức của nó. Mỗi hàm có thể được kiểm tra một cách độc lập với các dữ liệu đầu vào, với dữ liệu hợp lệ cũng như không hợp lệ. Các chương trình chứa các hàm cũng dễ bảo trì hơn, bởi vì những sửa đổi, nếu yêu cầu có thể được giới hạn trong các hàm của chương trình. Một hàm không chỉ được gọi từ các vị trí bên trong chương trình, mà các hàm còn có thể đặt vào một thư viện và được sử dụng bởi nhiều chương trình khác, vì vậy tiết kiệm được thời gian viết chương trình. Đó cũng là ý nghĩa của việc dùng hàm là viết một lần, dùng nhiều lần.

5.2. Định nghĩa và lời gọi hàm

Các hàm là các đoạn mã có thể nhận biết bằng một giáo diện đã được định nghĩa. Các hàm được gọi từ bất kỳ chỗ nào của chương trình và cho phép các chương trình lớn được chia ra thành các công việc dễ quản lý hơn, mỗi một trong chúng có thể được kiểm tra một các độc lập. Hàm có hai loại: hàm chuẩn và hàm do người dùng định nghĩa

+) Hàm chuẩn: Là các hàm có sẵn trong thư viện của C, khi cần chúng ta chỉ việc gọi tên hàm ra trong chương trình. Ví dụ như hàm printf, scanf, puts, clrscr…

+) Hàm người dùng:là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình. Và trong chương này, chúng ta chỉ xét đến hàm do người dùng định nghĩa.

Để sử dụng hàm trước hết người dùng cần phải định nghĩa hàm, sau đây chúng ta sẽ xem xét cấu trúc để định nghĩa một hàm mong muốn.

5.2.1. Định nghĩa

Một hàm được định nghĩa là một đoạn chương trình thực hiện một tác vụ được định nghĩa cụ thể. Chúng thực chất là những đoạn chương trình nhỏ giúp ta có thể giải quyết được vấn đề lớn .

Hàm được định nghĩa theo cấu trúc như sau:

Kiểu_trả_về *Tên_hàm*(Danh sách tham số hình thức)

```
{
    [Khai báo biến cục bộ]
    <Khối lệnh>
    [return [<Biểu thức>];]
}
```

Trong đó:

- **Kiểu_trả_về**: là kiểu dữ liệu của kết quả trả về, có thể là : **int, char, float, void**… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void. Còn nếu không khai báo **kiểu_trả_về** thì mặc định là hàm đó trả về một số nguyên.

- Danh sách tham số: là tham số truyền dữ liệu vào cho hàm và được gọi là tham số hình thức. Một hàm có thể có hoặc không có tham số. Nếu có nhiều tham số, các tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm có thể khai báo các biến cụ bộ, các biến này chỉ tồn tại bên trong hàm khi hàm đang được thực thi và sẽ bị xóa sau khi ra khỏi hàm.

- Khối lệnh: Có thể là một hoặc nhiều câu lệnh nhằm thực hiện nhiệm vụ của hàm.

- Lệnh **return**: dùng để trả về kết quả của hàm tại nơi gọi nó. Lệnh này có thể vắng mặt khi hàm không có giá trị trả về (kiểu trả về là **void**). Câu lệnh **return** sẽ được nói rõ hơn trong phần sau.

5.2.2. Lời gọi hàm

Sau khi đã định nghĩa một hàm, để sử dụng hàm trong chương trình chính thì ta phải có lời gọi hàm bên trong chương trình chính (sau hàm main( )), theo cấu trúc như sau:

Tên_hàm(Danh sách tham số thực);

Như vậy, từ hai cú pháp định nghĩa và gọi hàm ta có thể thấy rằng trong hàm có sử dụng hai loại tham số là tham số hình thức và tham số thực.

- Tham số hình thức là tham số được khai báo trong định nghĩa hàm.
- Tham số thực là tham số khai báo trong lời gọi hàm. Có bao nhiêu tham số hình thức trong định nghĩa thì có bấy nhiêu tham số thực trong lời gọi hàm. Hơn nữa, kiểu của tham số thực phải cùng kiểu với tham số hình thức tương ứng.

+)Một số chú ý :

• Một dấu chấm phẩy được dùng ở cuối câu lệnh khi một hàm được gọi, nhưng nó không được dùng sau một sự định nghĩa hàm.

• Cặp dấu ngoặc ( ) là bắt buộc theo sau tên hàm, cho dù hàm có đối số hay không.

• Các hàm được định nghĩa là ngang cấp nhau, không thể định nghĩa một hàm bên trong một định nghĩa hàm khác. Nhưng có thể gọi một hàm bên trong một lời gọi hàm khác.

• Một hàm nên được khai báo trước hàm main( ) trước khi nó được định nghĩa hoặc sử dụng. Điều này phải được thực hiện trong trường hợp hàm được gọi trước khi nó được định nghĩa.

• Hàm gọi đến một hàm khác được gọi là ***hàm gọi*** hay ***thủ tục gọi***. Và hàm được gọi đến còn được gọi là ***hàm được gọi*** hay ***thủ tục được gọi***.

• Chỉ một giá trị có thể được trả về bởi một hàm.

• Một chương trình có thể có một hoặc nhiều hàm.

5.2.3. Thí dụ về cách gọi hàm.

Bây giờ ta có thể xét một chương trình hoàn thiện như sau:

Chương trình 5.1/*In ra bảng chân lý của phép AND hai phần tử bất kỳ*/

```
#include "stdio.h"
#include "conio.h"
#define FALSE 0
#define TRUE 1
int AND(int,int);//khai bao ham truoc khi su dung
void main()
{
  int a,b,c;
  clrscr();
  puts("\tA\tB\tA&B");
  puts("\t********************");
  for(a=FALSE;a<=TRUE;a++)
        for(b=FALSE;b<=TRUE;b++)
              {
              c=AND(a,b);   //tham so thuc
              printf("%9d%8d%10d\n",a,b,c);
              }
getch();
}
//ham AND
int AND(int x, int y)  //tham so hinh thuc
```

```
{
  if(x && y)
return(TRUE);
  else
return(FALSE);
}
```

Kết quả thực hiện chương trình:

```
Turbo C++ IDE
        A          B          A & B
        *********************
        0          0          0
        0          1          0
        1          0          0
        1          1          1
```

5.3. Nguyên tắc hoạt động của hàm

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi.

Khi một hàm được gọi, quyền điều khiển sẽ được chuyển đến cho nó, ở đó các đối số hình thức được thay thế bởi các đối số thực.

Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.

Khi gặp lệnh return hoặc dấu kết thúc "}" ở cuối hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

5.4. Cách sử dụng các tham số trong hàm:

Như đã nói ở trên, hàm sử dụng đến hai kiểu tham số là tham số hình thức và tham số thực. Trong phần này, chúng ta sẽ xét kỹ hơn về cách khai báo và sử dụng các tham số đó.

Tham số hình thức: là tham số được khai báo và sử dụng trong định nghĩa hàm. Khi thoát khỏi hàm thì các tham số hình thức cũng bị xóa.

Có thể khai báo tham số hình thức theo hai cách:

- Khai báo trong cặp dấu ngoặc đơn ( ) khi định nghĩa tên hàm. Khi đó, mỗi đối số phải được định nghĩa riêng lẻ, cho dù chúng có cùng kiểu dữ liệu. Thí dụ, nếu x và y là hai đối số của một hàm AND( ) thì:

  + **int** AND(char x, char y) là một khai báo đúng

  + **int** AND(char x, y) là khai báo sai

- Khai báo ngay sau dấu ngoặc nhọn bắt đầu thân hàm ({). Cách khai báo này thật tiện lợi khi có nhiều tham số có cùng kiểu dữ liệu được truyền.

Trong trường hợp như vậy, chỉ phải chỉ rõ kiểu đề một lần duy nhất tại điểm bắt đầu. Thí dụ:

```
int AND( )
{
    int a,b;   //khai báo tham số hình thức
// khối lệnh
}
```

Tham số thực: là tham số được sử dụng trong lời gọi hàm. Tham số thực của hàm được khai báo trong chương trình chính sau hàm main( ) cùng với các biến, mảng …khác của chương trình. Thí dụ, ta có khai báo sau:

```
//các chỉ thị tiền xử lý
int AND(int a, int b)
void main( )
{
    int x,y,z;   //khai báo tham số thực
      z = AND(x,y); //lời gọi hàm
          …
} //kết thúc chương trình chính
```

5.5. Chuyển giao tham số của hàm

Một cách tổng quát, các hàm giao tiếp với nhau bằng cách truyền tham số. Các tham số được truyền theo một trong hai cách sau: truyền bằng giá trị hoặc truyền bằng tham chiếu.

5.5.1. Truyền bằng giá trị

Khi đó, các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức.

- Khi chương trình con được gọi để thi hành, các tham trị được cấp ô nhớ và nhận giá trị là bản sao giá trị của tham số thực. Do đó, mặc dù tham trị cũng là biến, nhưng việc thay đổi giá trị của chúng không có ý nghĩa gì đối với bên ngoài hàm, nên không làm ảnh hưởng đến tham số thực tương ứng.
- Nếu muốn sau khi kết thúc chương trình con giá trị của các tham số truyền vào thay đổi thì phải đặt tham số hình thức là các con trỏ, còn tham số thực tế là địa chỉ của các biến.

5.5.2. Truyền bằng tham chiếu

Khi các đối số được truyền bằng giá trị, các giá trị của đối số của hàm đang gọi không bị thay đổi. Tuy nhiên, có thể có trường hợp, ở đó giá trị của các đối số phải

được thay đổi. Trong những trường hợp như vậy, truyền bằng tham chiếu (hay còn gọi là truyền theo địa chỉ) được dùng. Khi truyền bằng tham chiếu, hàm được phép truy xuất đến vùng bộ nhớ thực của các đối số và vì vậy có thể thay đổi giá trị của các đối số của hàm gọi. Các con trỏ được dùng khi thực hiện truyền bằng tham chiếu.

Khi một con trỏ được truyền đến một hàm, địa chỉ của dữ liệu được truyền đến hàm nên hàm có thể tự do truy xuất nội dung của địa chỉ đó. Các hàm gọi nhận ra bất kỳ thay đổi trong nội dung của địa chỉ. Theo cách này, đối số hàm cho phép dữ liệu được thay đổi trong hàm gọi, cho phép truyền dữ liệu hai chiều giữa hàm gọi và hàm được gọi.

5.6. Sự trả về từ hàm

Lệnh **return** được sử dụng trong hàm có hai mục đích:

- Ngay lập tức trả điều khiển từ hàm về chương trình gọi
- Bất kỳ cái gì bên trong cặp dấu ngoặc () theo sau **return** được trả về như là một giá trị cho chương trình gọi.

Trong thực tế, lệnh return có thể được sử dụng theo một trong các cách sau đây:

- **return;** hoặc **return(0);** //không trả về giá trị
- **return(**hằng**);** // trả về giá trị hằng
- **return(**biến**);** // trả về giá trị của biến
- **return(**biểu thức**);** // trả về giá trị của biểu thức
- **return(**câu lệnh đánh giá); // Thí dụ: **return**(a>b?a:b);

Tuy nhiên, giới hạn của lệnh **return** là nó chỉ có thể trả về một giá trị duy nhất

5.7. Hàm đệ quy

+***) Định nghĩa:*** Một hàm được định nghĩa là hàm đệ quy nếu bên trong thân hàm có chứa lời gọi đến chính nó.

+***) Đặc điểm của hàm đệ quy:***

• Hàm đệ quy phải gồm có hai phần:

- Phần dừng hay phải có trường hợp nguyên tố: dùng để kết thúc thuật toán.
- Phần đệ quy: là phần có gọi lại hàm đang được định nghĩa.

• Sử dụng hàm đệ quy sẽ giúp chương trình dễ hiểu hơn nhưng sẽ làm tốn bộ nhớ nhiều hơn và tốc độ thực hiện chương trình sẽ chậm hơn khi không dùng đệ quy.

+***) Lưu ý:*** Khi sử dụng hàm đệ quy, có bao nhiêu lời gọi hàm được thực hiện thì có bấy nhiêu lần hàm được kết thúc và trả về giá trị cho nơi gọi nó. Vì vậy, khi viết chương trình hàm đệ quy cần phải hết sức chú ý đến việc truyền tham số và giá trị trả về của hàm. Nếu không, có thể dẫn đến treo máy hoặc cho kết quả sai.

Thí dụ: Chương trình sau sẽ mô tả thuật toán tính giai thừa của một số nguyên khi sử dụng dạng đệ quy và khi không sử dụng đệ quy.

/*Chương trình 5.2. Tính giai thừa của số nguyên n.*/

```
#include "stdio.h"
#include "conio.h"
/*Ham tinh n! bang de quy*/
unsigned int giaithua_dequy (int n)
{
  if (n==0)
        return 1;
  else
        return n*giaithua_dequy(n-1);
}
/*Ham tinh n! khong de quy*/
unsigned int giaithua_khongdequy(int n)
{
  unsigned int kq,i;
  kq = 1;
  for (i=2;i<=n;i++)
        kq = kq*i;
  return kq;
}
int main()
{
  int n;
  clrscr();
  printf("\nNhap so n can tinh giai thua ");
  scanf("%d",&n);
  printf("\nGoi ham de quy: %d != %u",n,giaithua_dequy(n));
  printf("\nGoi ham khong de quy: %d != %u",n,giaithua_khongdequy(n));
  getch();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
TC.EXE

Nhap so n can tinh giai thua 5

Goi ham de quy: 5 != 120
Goi ham khong de quy: 5 != 120
```

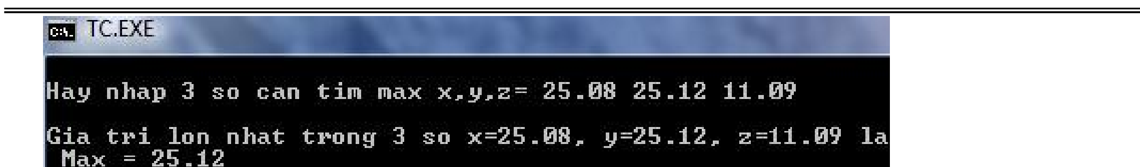
5.8. Một số thí dụ

Sau đây, chúng ta sẽ xét thêm một số thí dụ về cách sử dụng hàm trong C.

Chương trình 5.3 Chương trình tìm số max trong 3 số nhập vào từ bàn phím:

```
/* ********************************* */
#include "stdio.h"
#include "conio.h"
float TimMax(float,float,float);/*Nguyen mau cua ham*/
/* ********************************* */
int main()
{
    float x,y,z; /* Cac tham so thuc cua ham */
    clrscr();
    printf("\nHay nhap 3 so can tim max x,y,z= ");
    scanf("%f%f%f",&x,&y,&z);
    printf("\nGia tri lon nhat trong 3 so x=%10.2f, y=%10.2f, z=%10.2f la\n Max = %10.2f",x,y,z, TimMax(x,y,z));
    getch();
}/* ket thuc ham main()*/
/* *********************************** */
/* Dinh nghia ham TimMax */
float TimMax(float i,float j, float k)/* Cac tham so hinh thuc */
{
/* Bat dau than ham TimMax*/
    float Max; /* Bien cuc bo cua ham */
    Max = i>j?i:j; /* Tim so lon  nhat trong hai so i,j*/
    return(Max>k?Max:k); /* so sanh voi so con lai */
} /* ket thuc ham TimMax */
```

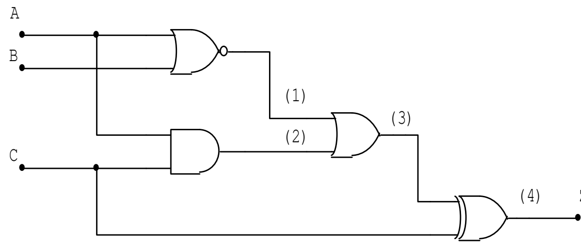
Kết quả thực hiện chương trình



Chương trình 5.4. Chương trình xác định bảng chân lý của một phương trình đại số Boole.

Z = $\left(\overline{A+B}+A.C\right)$ (1)

Hình 5.1 giới thiệu một sơ đồ hàm Boole này:



Hình 5.15. Mô tả hàm Boole (1)

Bốn điểm đánh số trên sơ đồ

(1) $\overline{A}$

(2) A.C

(3) $\overline{A+B}+$

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

Bảng sau giới thiệu giá trị chân lý thông qua các mức logic 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.

A	B	C	$\overline{A+B}$	A.C	$\overline{A+B}+A.$	$(\overline{A+B}+A.C)\oplus C$
0	0	0	1	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	1	0	0	0	1
1	0	0	0	0	0	0
1	0	1	0	1	1	0
1	1	1	0	1	1	0

Sự hoán vị các biến lối vào bảng chân lý (ví dụ: 000, 001, 010…) được tạo ra bằng cách sử dụng 3 vòng lặp for lồng vào nhau. Vòng lặp bên trong thay đổi C từ 0 sang 1, vòng tiếp theo thay đổi B và vòng ngoài thay đổi A. Các hàm Boole sử dụng các toán tử logic && và ||. Chương trình như sau:

// bắt đầu chương trình

```
#include "stdio.h"
#include "conio.h"
void main()
{
int A,B,C,Z;/*bien toan cuc*/
clrscr();
printf("\tA\tB\tC\tZ");
printf ("\n\t************************\n");
```

```
for (A=0;A<=1;A++)
  for (B=0;B<=1;B++)
        for (C=0;C<=1;C++)
              {
                    Z=XOR(OR(NOR(A,B),AND(A,C)),C);
                    printf(" %8d%8d%8d%8d\n",A,B,C,Z);
              }
getch();
}/* Ket thuc chuong trinh chinh*/
/* bat dau thuc hien cac ham con*/
int XOR(int x, int y)/*tham so hinh thuc*/
{
  if(x^y) return(1);
  else return(0);
}
int AND(int x, int y)
{
  if(x&&y) return(1);
  else return(0);
}
int NOR(int x, int y)
{
  if(x||y) return(0);
  else return(1);
}
int OR(int x, int y)
{
  if(x||y) return(1);
  else return(0);
}
/*ket thuc chuong trinh*/
```

Kết quả:

```
Turbo C++ IDE
    A        B        C        Z
**************************************
    0        0        0        1
    0        0        1        0
    0        1        0        0
    0        1        1        1
    1        0        0        0
    1        0        1        0
    1        1        0        0
    1        1        1        0
```

Chương trình 5.4 Tính trở kháng của mạch RL nối tiếp

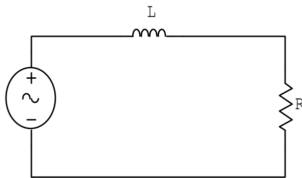
Độ lớn trở kháng của mạch RL nối tiếp được tính bởi công thức:

$|Z| = \sqrt{R^2 + X_L^2}$

Và góc pha của trở kháng xác định bởi công thức:

$$\varphi_Z = tan^{-1}\frac{X_L}{R}$$

Sơ đồ mạch RL nối tiếp như sau:



Trong 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=2π.fL để xác định điện kháng cuộn cảm. Và để tính được giá trị arctang(X_L/R) ta cần khai báo thư viện Math.h để tính toán.

```
//chuong trinh tinh tro khang mach RL noi tiep
#include "stdio.h"
#include "conio.h"
#include "math.h"
#define PI 3.14
void main()
{
  float tong_tro(float x,float y); //bien toan cuc
  float goc_pha(float x,float y);
  float r,L,freq,Xl;
  clrscr();
  printf("\n Nhap vao gia tri dien tro R = ");
scanf("%f",&r);
  printf("\n Nhap vao gia tri dien cam L = ");
scanf("%f",&L);
```

```
  printf("\n Nhap vao gia tri cua tan so f = ");
scanf("%f",&freq);
  Xl=2*PI*freq*L;
  printf("\n Tro khang cua cuon day la %3f",tong_tro(r,Xl));
  printf("\n Goc pha cua tro khang la %4f",goc_pha(Xl,r));
getch();
}
float tong_tro(float x,float y)
{
  float z;
  z = sqrt(x*x+y*y);
  return(z);
}
float goc_pha(float x, float y)
{
  float g;
  g = atan(x/y);//g la gia tri do
  return(g*180/PI);//chuyen doi gia tri sang radian
}
```

Bài tập chương 5

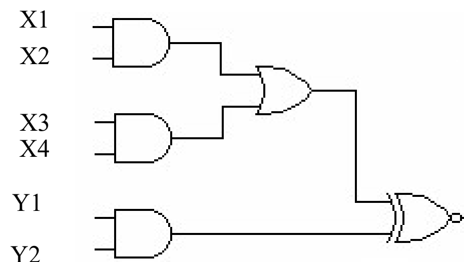
Bài 1: Viết chương trình tính giá trị biểu thức logic Z: *(viết dưới dạng hàm)*

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

trong đó, A,B,C,D là các số 0,1 nhập vào từ bàn phím

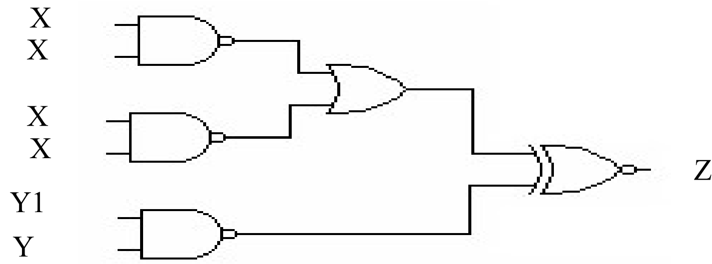
Bài 2: Viết chương trình (dùng hàm) hiển thị bảng chân lý của các hàm cơ bản AND, NAND, OR, XOR, NOR

Bài 3: Cho mạch logic như dưới đây, hãy lập trình hiển thị bản chân lý của mạch



Trong đó: X1, X2, X3, X4, Y1, Y2 được nhập vào từ bàn phím

Bài 4: Cho mạch logic như dưới đây, hãy lập trình hiển thị bảng chân lý của mạch



Trong đó: X1, X2, X3, X4, Y1, Y2 được nhập vào từ bàn phím

Bài 5: Cho hệ số khuếch đại của mạch lọc tần số thấp được tính

$$K = \frac{1}{2\pi fRC}\Omega$$

Hãy viết chương trình có thể in ra một bảng tần số và hệ số khuếch đại tương ứng với dải tần từ 1Hz–1KHz theo từng bước hơn kém nhau 20, với R=1KΩ, C=0.1μF.

Bài 6: Viết chương trình hiển thị bảng chân lý của hàm NAND, hàm XOR và hàm OR.

Bài 7: Viết chương trình hiển thị bảng chân lý của hàm AND, hàm NOR, và hàm NXOR

Bài 8: Viết chương trình hiển thị bảng chân lý của mạch hiệu toàn phần (Full Subtractor – FS)

D = $A \oplus B \oplus B_i$

$B_0 = \overline{A}B + \overline{A}B_i + BB_i$

Trong đó: A,B,B_i là số nhị phân được nhập vào từ bàn phím

Bài 9: Viết chương trình hiển thị bảng chân lý sơ đồ biến đổi mã nhị phân sang mã Gray: Trong đó A_0, A_1, A_3 là đầu vào, G_0, G_1, G_3 là đầu ra

$G_0 = A_0 \oplus A_1 \quad G_1 = A_1 \oplus A_2 \quad G_2 = A_2 \oplus A_3 \quad G_3 = A_3$

Bài 10: Viết chương trình hiển thị bảng chân lý của bộ phân kênh 1 đường vào và 4 đường ra: trong đó D là hằng số có giá trị bằng 1; A, B là tín hiệu vào, Y_0...Y_3 là tín hiệu ra.

$Y_0 = \overline{A}.\overline{B}.D \qquad Y_1 = A.\overline{B}.D \qquad Y_2 = \overline{A}.BD \qquad Y_3 = A.B.D$

Chương 6 CON TRỎ

6.1. Giới thiệu

Các biến đã khai báo và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là *biến tĩnh*. Khi khai báo và sử dụng biến tĩnh, ta có thể gặp phải một số hạn chế sau:

- Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
- Cấp phát ô nhớ thiếu dẫn đến chương trình thực thi bị lỗi.

Để khắc phục hạn chế trên, trong C cung cấp cho ta kiểu *biến động* với các đặc điểm sau:

- Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
- Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
- Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên các *biến động* không có địa chỉ nhất định nên không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp một loại biến đặc biệt nữa để khắc phục tình trạng này, đó là *biến con trỏ* (pointer). Con trỏ có thể được sử dụng trong một số trường hợp sau:

- Để trả về nhiều hơn một giá trị từ một hàm
- Để cấp phát bộ nhớ động (dynamic memory allocation) và truy xuất vào vùng nhớ được cấp phát này
- Thuận tiện hơn trong việc truyền các mảng và chuỗi từ một hàm đến một hàm khác
- Sử dụng con trỏ để làm việc với các phần tử của mảng thay vì truy xuất trực tiếp vào các phần tử này.

6.2. Biến con trỏ

6.2.1. Khái niệm con trỏ.

Một con trỏ thực chất là một biến, nó chứa địa chỉ vùng nhớ của một biến khác, chứ không lưu trữ giá trị của biến đó. Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là con trỏ chỉ đến biến thứ hai kia. Một con trỏ cung cấp phương thức gián tiếp để truy xuất giá trị của các phần tử dữ liệu.

6.2.2. Khai báo biến con trỏ

Vì con trỏ cũng là một biến nên trước khi sử dụng, bạn phải khai báo biến con trỏ theo cú pháp sau:

Kiểu_dữ_liệu *Tên_biến_con_trỏ;

Ý nghĩa: Khai báo biến con trỏ có kiểu là **kiểu_dữ_liệu** và có tên là tên_biến_con_trỏ.

- Kiểu dữ liệu: là kiểu dữ liệu của nội dung chứa trong địa chỉ mà con trỏ chỉ tới. Có bao nhiêu kiểu dữ liệu thì có bấy nhiêu kiểu con trỏ. Ví dụ, kiểu con trỏ là **int** thì nó lưu trữ địa chỉ của một biến kiểu **int**, kiểu con trỏ là **char** thì nó chứa địa chỉ của một biến kiểu **char**.

Địa chỉ của biến là một vị trí xác định trong bộ nhớ. Tùy thuộc vào kiểu dữ liệu mà biến đó được cấp cho một vùng nhớ xác định khác nhau. Thí dụ, nếu khai báo kiểu ***int*** thì biến sẽ được cấp một vùng nhớ là 2 byte, kiểu **char** thì sử dụng vùng nhớ là 1 byte, kiểu **float** sử dụng vùng nhớ là 4 byte… Bộ nhớ chia thành nhiều ngăn nhớ, mỗi ngăn nhớ có độ rộng là 1 byte (8 bit) và có một giá trị địa chỉ xác định. Thông thường, địa chỉ này được chỉ rõ bằng một giá trị thập lục phân vì nó được biểu diễn ngắn gọn và có thể dễ dàng chuyển sang địa chỉ nhị phân.

Thí dụ: Một số khai báo biến con trỏ như sau:

int *ptr;

float *dientro_ptr;

char *ch_ptr;

Ghi chú:

- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu (kiểu dữ liệu chỉ xác định kích thước cho biến tĩnh) và luôn có kích thước là 2 byte.
- Nếu chưa muốn khai báo kiểu dữ liệu mà con trỏ đang chỉ đến, ta sử dụng toán tử **void** để khai báo:

void * tên_biến_con_trỏ;

Sau đó, ta muốn con trỏ chỉ đến kiểu dữ liệu nào cũng được.

6.2.3. Gán giá trị cho biến con trỏ

Việc gán giá trị cho biến con trỏ thực chất là việc quy định vùng con trỏ chỉ tới và truy xuất giá trị của biến có địa chỉ mà con trỏ đó chứa.

6.2.3.1. Quy định vùng con trỏ chỉ tới

Toán tử *&* dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

Cú pháp:

Tên_biến_con_trỏ = &Tên_biến;

Ý nghĩa: Gán địa chỉ của biến Tên_biến cho con trỏ Tên_biến_con_trỏ.

Thí dụ: **char** ch, *ptr_ch;

float r1, *ptr_r;

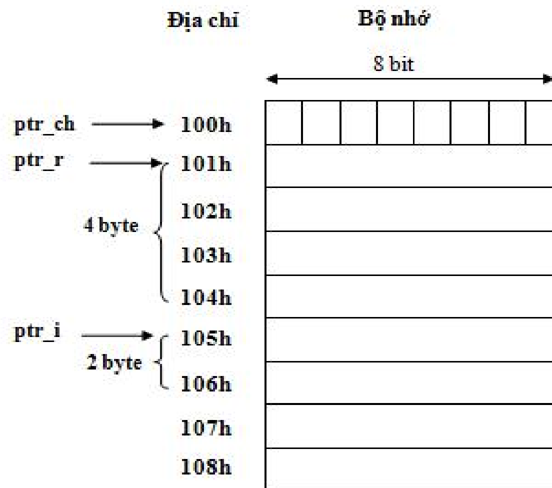
int i, *ptr_i;

ptr_ch = &ch; // gán địa chỉ của biến ch cho con trỏ ptr_ch

ptr_r = &r1; // gán địa chỉ của biến r1 cho con trỏ ptr_r

ptr_i = &i; // gán địa chỉ của biến i cho con trỏ ptr_i

Ta có thể mô tả phân bố biến trong bộ nhớ như sau: Biến ch sẽ được cấp cho 1 byte vùng nhớ (kiểu **char**), biến thực r1 sẽ được cấp cho vùng nhớ 4 byte và biến nguyên i sẽ được cấp một vùng nhớ 2 byte.



Hình 6.16 Mô tả tổ chức bộ nhớ các biến

Lưu ý: Khi gán địa chỉ của một biến tĩnh cho con trỏ cần phải lưu ý đến kiểu dữ liệu của biến. Nếu kiểu dữ liệu của biến và kiểu khai báo cho con trỏ không tương thích sẽ gây ra lỗi.

Thí dụ: khai báo:

```
int *ptr;
float tro_khang;
ptr = &tro_khang;
```

Phép gán trên sẽ làm lỗi chương trình vì kiểu khai báo cho con trỏ *ptr* (kiểu **int**) không tương thích với kiểu của biến tĩnh *tro_khang* (kiểu **float**).

6.2.3.2. Nội dung của ô nhớ mà con trỏ chỉ tới

Để truy xuất nội dung của ô nhớ mà con trỏ chỉ tới ta sử dụng toán tử * theo cú pháp:

```
* biến_con_trỏ;
```

Với cú pháp trên thì *biến_con_trỏ có thể coi là một biến có kiểu dữ liệu được mô tả trong phần khai báo con trỏ.

Thí dụ:

```
    float r1, r2, r_td, *ptr1, *ptr2;
x = 50;
ptr1 = &r1; // con trỏ ptr1 chỉ đến địa chỉ của phần tử r1.
ptr2 = &r2; // con trỏ ptr2 chỉ đến địa chỉ của phần tử r2.
```

Với khai báo trên ta có các câu lệnh sau đây có tác dụng như sau:

1. r_td = r1*r2/(r1+r2);

2. r_td = (*ptr1)*(*ptr2)/((*ptr1) + (*ptr2));

6.3. Cấp pháp bộ nhớ

6.3.1. Cấp phát bộ nhớ để lưu dữ liệu

Trước khi sử dụng biến con trỏ phải cấp phát vùng nhớ cho biến con trỏ này quản lý. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện **alloc.h.**

Cú pháp:

(1) Biến_con_trỏ **= malloc(sizeof(kiểu_dữ liệu));**

Cấp phát vùng nhớ có kích thước là kích thước của **kiểu_dữ_liệu**.

(2) Biến_con_trỏ **= calloc(n, sizeof(kiểu_dữ_liệu));**

Cấp phát vùng nhớ có kích thước bằng n lần (n là số nguyên) kích thước của **kiểu_dữ_liệu.**

Thí dụ: Khai báo:

int a, *pa, *pb;

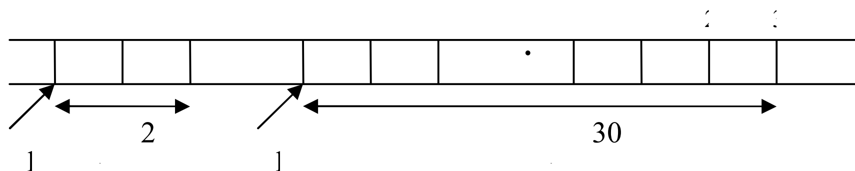
pa = (**int*)malloc(sizeof(int)**);

Cấp phát vùng nhớ để lưu dữ liệu mà con trỏ pa chỉ đến có kích thước bằng với kích thước của một số nguyên kiểu **int,** tức vùng nhớ có kích thước 2 byte.

pb= (**int*)calloc(15, sizeof(int)**);

Cấp phát vùng nhớ chứa dữ liệu mà con trỏ pb chỉ đến có thể chứa được 15 số nguyên (tương ứng kích thước vùng nhớ là 15*2 = 30 byte).

Có thể biểu diễn vùng nhớ mà con trỏ pa và pb chỉ tới như sau:



Hình 6.17 Mô tả vùng nhớ được cấp phát cho hai biến pa và pb

Việc thay đổi kích thước vùng nhớ đã được cấp phát để lưu dữ liệu trước đó có thể được thực hiện bằng hàm **realloc( )** được định nghĩa trong thư viện **alloc.h.**

Cú pháp: **void *realloc(**biến_con_trỏ**, size_t** size**);**

Ý nghĩa:

- Cấp phát lại 1 vùng nhớ mới cho con trỏ có tên biến_con_trỏ đã được cấp bộ nhớ trước đó bởi hàm ***malloc*** hay ***calloc***, vùng nhớ này có kích thước mới là size; khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại.
- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

Thí dụ: Trong ví dụ trên, có thể cấp phát lại vùng nhớ do con trỏ pa quản lý như sau:

int a, *pa;

pa = **(int*)malloc(sizeof(int))**; /*Cấp phát vùng nhớ có kích thước 2 byte*/

pa = **realloc**(pa, 6); /* Cấp phát lại vùng nhớ có kích thước 6 byte*/

6.3.3 Giải phóng vùng nhớ động

Một vùng nhớ đã cấp phát cho biến con trỏ, khi không còn sử dụng nữa phải thu hồi lại vùng nhớ này nhờ hàm **free( )**.

Cú pháp: **void free(**biến_con_trỏ**);**

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Thí dụ: Ở ví dụ trên, sau khi thực hiện xong giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

free(pa);

free(pb);

6.4. Các phép toán trên biến con trỏ

6.4.1. Phép cộng/trừ biến con trỏ với số nguyên

Con trỏ chỉ có thể thực hiện phép cộng (+) và phép trừ (-) với một số nguyên N và cho kết quả trả về là một con trỏ. Các con trỏ tăng giảm giá trị phụ thuộc vào kiểu dữ liệu mà chúng trỏ tới.

Thí dụ: **int** n, *ptr1, *ptr2;

float R, *ptr_R;

ptr1 = &n; // con trỏ ptr1 trỏ đến địa chỉ của n.

ptr2 = ptr1 + 3; /* con trỏ ptr2 trỏ đến địa chỉ của phần tử cách n 3 phần tử. */

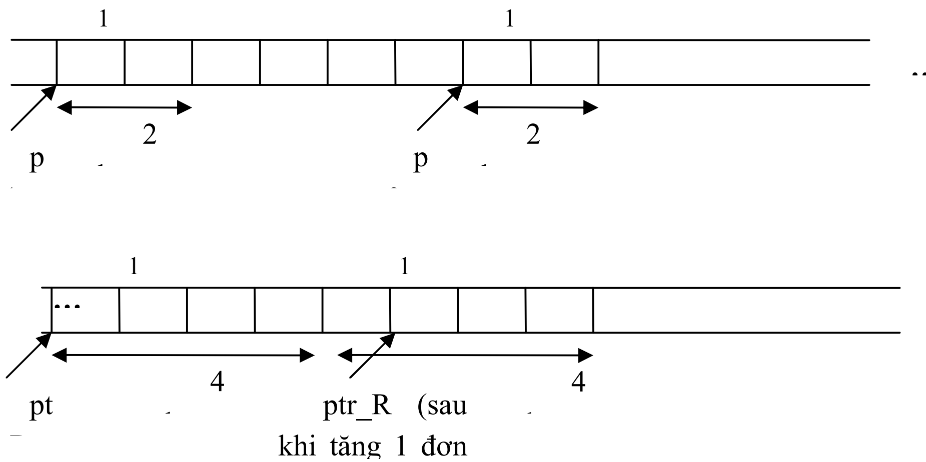
ptr_R =&R; // con trỏ ptr_R trỏ đến địa chỉ của R

ptr_R++; // con trỏ ptr_R trỏ đến phần tử liền kề sau của R

Giải thích:

- Ta thấy, biến n là biến kiểu **int** nên có vùng nhớ là 2 byte. Giả sử giá trị của n được lưu và địa chỉ 100h, khi đó con trỏ ptr1 trỏ đến địa chỉ 100h, còn con trỏ ptr2 sẽ trỏ đến địa chỉ 106h vì đó là địa chỉ bắt đầu lưu giá trị phần tử cách n là 3.

- Còn biến R có kiểu dữ liệu số thực nên nó được cấp một vùng nhớ 4 byte để lưu giá trị. Ban đầu, con trỏ ptr_R trỏ đến địa chỉ của R, giả sử là 1000h. Sau khi thực hiện phép tăng giá trị của con trỏ lên 1 đơn vị, con trỏ này sẽ chỉ đến phần tử dữ liệu kiểu **float** kế tiếp ở địa chỉ 1004h.



Có thể tổng kết phép toán số học trên con trỏ ptr trỏ đến biến n trong bảng sau:

Bảng 6.18 Phép toán số học trên con trỏ

++ptr hoặc ptr++	Trỏ đến phần tử dữ liệu kế tiếp đứng sau n
--ptr hoặc ptr--	Trỏ đến phần tử dữ liệu liền kề trước n
ptr + i	Trỏ đến phần tử dữ liệu thứ i đứng sau kể từ địa chỉ của n.
ptr – i	Trỏ đến phần tử dữ liệu thứ i đứng trước kể từ địa chỉ của n.
++*ptr hoặc (*ptr)++	Sẽ tăng giá trị biến n lên 1.
*ptr++	Sẽ tác động đến giá trị của phần tử dữ liệu kế tiếp sau biến n.

Chú ý:

- không thể thực hiện phép cộng hai con trỏ với nhau
- Phép trừ hai con trỏ cùng kiểu sẽ trả về một số nguyên. Đây chính là khoảng cách (số phần tử) giữa hai con trỏ đó

6.4.2 Phép gán và phép so sánh

+) Phép gán

Ta có thể thực hiện phép gán địa chỉ của một biến cho một con trỏ cùng kiểu hoặc ta cũng có thể gán giá trị của một con trỏ cho một con trỏ cùng kiểu khác.

Thí dụ: **int** n, *ptr1, *ptr2;
ptr1 = &n;
ptr2 = ptr1;

Sau khi gán giá trị của con trỏ ptr1 cho con trỏ ptr2 (cùng kiểu **int**), thì cả hai con trỏ ptr1 và ptr2 cùng trỏ đến một địa chỉ của biến n. Để có thể gán giá trị của các con

trỏ khác kiểu cho nhau, ta phải dùng toán tử ép kiểu. Phép ép kiểu sẽ được trình bày trong phần sau.

+) Phép so sánh

Hai con trỏ có thể được so sánh trong một biểu thức quan hệ. Tuy nhiên, điều này chỉ có thể nếu cả hai biến này đều trỏ đến các biến có cùng kiểu dữ liệu. ptr_a và ptr_b là hai biến con trỏ trỏ đến các phần tử dữ liệu có cùng kiểu a và b. Trong trường hợp này, các phép so sánh sau đây là có thể thực hiện:

Bảng 6.19 Các phép so sánh thực hiện với con trỏ

ptr_a < ptr_b	Trả về giá trị **true** nếu a được lưu trữ ở vị trí trước b
ptr_a > ptr_b	Trả về giá trị **true** nếu a được lưu trữ ở vị trí sau b
ptr_a <= ptr_b	Trả về giá trị **true** nếu a được lưu trữ ở vị trí trước b hoặc ptr_a và ptr_b trỏ đến cùng một vị trí
ptr_a >= ptr_b	Trả về giá trị **true** nếu a được lưu trữ ở vị trí sau b hoặc ptr_a và ptr_b trỏ đến cùng một vị trí
ptr_a == ptr_b	Trả về giá trị **true** nếu cả hai con trỏ ptr_a và ptr_b trỏ đến cùng một phần tử dữ liệu.
ptr_a != ptr_b	Trả về giá trị **true** nếu cả ptr_a và ptr_b trỏ đến các phần tử dữ liệu khác nhau nhưng có cùng kiểu dữ liệu.
ptr_a == NULL	Trả về giá trị **true** nếu ptr_a được gán giá trị NULL (0)

Lưu ý:

- Phép so sánh con trỏ với giá trị NULL sẽ xác định cho ta biết con trỏ đã chỉ đến vùng nhớ nào hay chưa. Nếu kết quả trả về **true** có nghĩa là con trỏ chưa trỏ đến vùng nhớ nào.
- Khi một con trỏ đã được khai báo nhưng chưa được gán địa chỉ của một vùng nhớ nào thì vẫn chưa thể sử dụng được.

6.4.3. Sự chuyển kiểu

Việc chuyển kiểu cũng được thực hiện trên con trỏ theo cú pháp:

(Kiểu_kết_quả*)biến_con_trỏ

Thí dụ:

```
int a, *pa;
float *pf;
pa = &a;
pf = (float*)pa;
```

Ép kiểu con trỏ pa từ kiểu nguyên sang kiểu số thực **float.**

6.4.4 Con trỏ hằng và con trỏ đến đối tượng hằng

- Để khai báo con trỏ hằng ta sử dụng cú pháp sau:

Kiểu_dữ_liệu *const tên_con_trỏ = giá trị;

- Để con trỏ chỉ đến một đối tượng hằng ta sử dụng cú pháp:

Kiểu_dữ_liệu const *tên_con_trỏ = giá trị hằng;

Hoặc

const kiểu_dữ_liệu *tên_con_trỏ = giá trị hằng;

Lưu ý: Cần phân biệt con trỏ hằng và con trỏ chỉ đến đối tượng là hằng:

- Con trỏ hằng không bị thay đổi khi thực hiện chương trình, không áp dụng được các phép toán số học với nó.

- Còn con trỏ chỉ tới đối tượng hằng là giá trị địa chỉ vùng nhớ là một hằng số xác định. Trong trường hợp này, con trỏ vẫn tham gia vào các phép toán số học đã trình bày ở trên.

Thí dụ: Xét khai báo sau:

char *const ptr1 = “dien tu”;

const char *ptr2 = “dien tu”;

ptr1++; //sai

ptr2++; //đúng: khi đó *ptr2= ‘i’ và ptr2 = “ien tu”.

6.4.5. Con trỏ NULL

Con trỏ NULL là con trỏ không chứa địa chỉ nào cả. Ta có thể gán giá trị NULL cho 1 con trỏ có kiểu bất kỳ.

6.5. Con trỏ và hàm

Con trỏ và hàm có mối quan hệ khá mật thiết với nhau. Bởi vì, một con trỏ có thể được dùng như một tham số hình thức của hàm, hoặc là một hàm cũng có thể cho kết quả trả về là một con trỏ.

6.5.1. Khái niệm và cách khai báo con trỏ hàm

Trong khi xây dựng chương trình, có thể ta cần thiết kế các hàm mà tham số thực trong lời gọi đến nó lại là tên của một hàm khác. Để thực hiện được điều đó, ta cần sử dụng đến con trỏ tới hàm (hay còn gọi là con trỏ hàm). Con trỏ hàm là một con trỏ đặc biệt dùng để chứa địa chỉ của một hàm và được khai báo theo cú pháp sau:

Kiểu_dữ_liệu (*tên_hàm)(danh sách kiểu tham số);

- Ý nghĩa: khai báo một con trỏ hàm có kiểu trả về là **kiểu_dữ_liệu.** Thí dụ, để khai báo một con trỏ hàm có tên là ptr có kiểu dữ liệu trả về là **float** và với tham số hình thức có kiểu là **int,** ta khai báo như sau:

float (* ptr)(int)

Lưu ý: một con trỏ hàm trước khi sử dụng phải được gán cho một tên hàm xác định, và kiểu của hàm và kiểu của con trỏ phải tương thích nhau. Phép gán có thể được thực hiện ngay trong lúc khai báo hoặc sau khai báo. Sau phép gán, ta có thể dùng tên con trỏ thay cho tên hàm. Thí dụ sau sẽ làm sáng tỏ điều này:

float tong_tro (float r1, float r2)

```
{  float R_td;
     R_td = r1*r2/(r1+r2);
}
float (*ptr_R) (float, float); // khai báo con trỏ hàm
void main()
{
    ptr_R = tong_tro;    //gán tên hàm tong_tro cho con trỏ hàm ptr_R.
    printf("tong tro cua mach: %5.3f ", ptr_R(35.0, 10.45));
}
```

Hoặc có thể gán tên hàm cho con trỏ hàm ngay trong câu lệnh khai báo con trỏ hàm như sau:

```
float (*ptr_R) (float, float) = tong_tro;
 // khai báo và gán tên hàm cho con trỏ hàm
```

6.5.2. Đối con trỏ hàm

Như đã nhắc đến trong chương trước, khi tham số thực là tên của một hàm nào đó trong lời gọi hàm thì tham số hình thức cho hàm đó phải là *con trỏ hàm*.

Khi đó, trong thân hàm ta có thể sử dụng tham số con trỏ hàm theo một trong ba cách sau:

(1). Tên_con_trỏ_hàm(danh sách tham số thực của hàm được trỏ tới);

(2). (Tên_con_trỏ_hàm)(danh sách tham số thực của hàm được trỏ tới);

(3). (*Tên_con_trỏ_hàm)(danh sách tham số thực của hàm được trỏ tới);

6.5.3. Thí dụ

Chương trình 6.1 *Tính điện áp lối ra của bộ cộng đảo.*

```
#include<conio.h>
#include<stdio.h>
void nhap(float *a,int n)
{
  int i;
  for(i=0;i<n;i++)
    {
   printf("a[%d]=",i);scanf("%f",&a[i]);
    }
}
void main()
{
 clrscr();
 float *r,*Uv,Ur=0;
```

```
  int i,n;
  printf("ban hay nhap so dau vao cua bo cong dao n: ");
  scanf("%d",&n);
  printf("ban hay nhap gia tri cua cac dien tro\n");
  nhap(r,n);
  printf("nhap vao cac dien ap dau vao\n");
  nhap(Uv,n);
  for(i=0;i<n;i++)
    Ur+=Uv[i]/r[i];
  Ur=Ur*rht;
  printf("gia tri dau ra cua bo cong dao la Ur= %f",Ur);
  getch();
}
```

Chương trình 6.2 *Chương trình tính điện trở tương đương của hai điện trở mắc song song.*

```
#include <stdio.h>
#include <conio.h>
void  nhap_giatri(float *r1,float *r2);
void  tinh_Rtd(float r1,float r2,float *r_e);
void  in_ketqua(float r1,float r2,float r_e);
void main()
{ float     R1,R2,R_equ;
  nhap_giatri(&R1,&R2);
  tinh_Rtd(R1,R2,&R_equ);
  in_ketqua(R1,R2,R_equ);
  getch();
}
void  nhap_giatri(float *r1,float *r2)
{
  do
  {
      printf("Nhap R1 >>");
      scanf("%f",r1);
      if (*r1<0) puts("Gia tri khong hop le: nhap lai");
  } while (*r1<0);
  do
  {
```

```
        printf("Nhap R2 >>");
        scanf("%f",r2);
        if (*r2<0) puts("Gia tri khong hop le: Nhap lai");
  } while (*r2<0);
}
void  tinh_Rtd(float r1,float r2,float *r_e)
{
  *r_e=1/(1/r1+1/r2);
}

void  in_ketqua(float r1,float r2,float r_e)
{
  printf("Cac gia tri dien tro %8.3f va %8.3f ohm\n",r1,r2);
  printf("Dien tro tuong duong cua mach la %8.3f ohm\n",r_e);
}
```

Chương 7 MẢNG VÀ CHUỖI

7.1. Giới thiệu chung

Trong chương này chúng ta sẽ đi nghiên cứu về các vấn đề liên quan đến **mảng** và **chuỗi** ký tự trong C. Các mảng là một trong những kiểu dữ liệu thường hay gặp trong thực tế: một dãy số, một dòng văn bản, bản danh sách các điện trở cũng thuộc mảng dữ liệu. ví dụ như ta có một mạch gồm 5 phần tử điện trở, thay vì khai báo:

float R1, R2, R3, R4, R5;

Ta chỉ cần khai báo:

float R[5];

Khai báo như trên nghĩa là sẽ có một mảng điện trở (R[5]): mảng một chiều R, gồm có 5 phần tử thuộc kiểu số thực và được đánh số như sau: R[0], R[1], R[2], R[3], R[4]. Như vậy thay vì phải khai báo từng điện trở trong một biến có tên khác nhau, ta dùng một biến chung cho chúng. Một **mảng** là tập hợp các phần tử dữ liệu có cùng kiểu, mỗi phần tử được lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ chính. Những phần tử này được gọi là **phần tử mảng**.

Trong Turbo C, chuỗi ký tự được cấu tạo từ kiểu **char**. Có thể coi chuỗi ký tự là một mảng, trong đó các thành phần là các ký tự và tổng số ký tự là kích thước mảng.

7.2. Phần tử mảng và các chỉ số mảng

Mỗi **phần tử của mảng** được định danh bằng một **chỉ mục** hoặc **chỉ số** gán cho nó. **Chiều** của mảng được xác định bằng số chỉ số cần thiết để định danh duy nhất mỗi phần tử. Một chỉ số là một số nguyên dương được bao bằng dấu ngoặc vuông **[ ]** đặt ngay sau tên mảng, không có khoảng trắng ở giữa. Một **chỉ số** chứa các giá trị nguyên bắt đầu bằng 0. Vì vậy, một mảng **dien_tro** với 5 phần tử được biểu diễn như sau:

dien_tro[0],…,dien_tro[4].

Như đã thấy, phần tử mảng bắt đầu với dien_tro[0], và vì vậy phần tử cuối cùng là dien_tro[4] không phải là dien_tro[5]. Điều này là do bởi trong C, chỉ số mảng bắt đầu từ 0; do đó trong mảng **N** phần tử, phần tử cuối cùng có chỉ số là **N-1**. Phạm vi cho phép của các giá trị chỉ số được gọi là **miền giới hạn** của chỉ số mảng, giới hạn **dưới** và giới hạn **trên**. Một chỉ số mảng hợp lệ phải có một giá trị nguyên nằm trong niềm giới hạn. Thuật ngữ **hợp lệ** được sử dụng cho một nguyên nhân rất đặc trưng. Trong C, nếu người dùng cố gắng truy xuất một phần tử nằm ngoài dãy chỉ số hợp lệ (như dien_tro[5] trong ví dụ trên của mảng), trình biên dịch C sẽ không phát sinh ra lỗi. Tuy nhiên, có thể nó truy xuất một giá trị nào đó dẫn đến kết quả không đoán được. Cũng có nguy cơ viết chồng lên dữ liệu hoặc mã lệnh chương trình. Vì vậy, người lập trình phải đảm bảo rằng tất cả các chỉ số là nằm trong miền giới hạn hợp lệ.

7.3. Cách khai báo một mảng

7.3.1. Khái niệm

Mảng là một tập hợp nhiều phần tử có cùng một kiểu giá trị và có chung một tên. Mỗi phần tử mảng có thể biểu diễn được một giá trị. Có bao nhiêu kiểu dữ liệu (kiểu giá trị) thì cũng có bấy nhiêu kiểu mảng.

Thí dụ có các mảng sau:

int a[15], b[4][6]

float R[5], Z[3][3]

Trong đó:

- Mảng a[15]: mảng một chiều a, có 15 phần tử thuộc kiểu số nguyên (int) được đánh số như sau: a[0], a[1],…,a[13], a[14].

Như vậy mảng a có thể biểu diễn một dãy 15 phần tử, mỗi phần tử a[i] chứa một giá trị nguyên

- Mảng b[4][6]: mảng hai chiều b, có 24 phần tử thuộc kiểu nguyên (int) gồm 4 dòng và 6 cột.

- Mảng R[5]: mảng một chiều R, có 5 phần tử thuộc kiểu số thực (float) được đánh số: R[0],…,R[4]

Mỗi phần tử R[i] biểu diễn một giá trị kiểu số thực (float). Mảng R biểu diễn được 5 số thực

- Mảng Z[3][3]: mảng hai chiều Z gồm 9 phần tử, 3 dòng và 3 cột, mỗi phần tử Z[i][j] chứa một giá trị kiểu số thực được sắp xếp như sau:

Z[0][0]   Z[0][1]   Z[0][2]
Z[1][0]   Z[1][1]   Z[1][2]
Z[2][0]   Z[2][1]   Z[2][2]

Như vậy để xác định một mảng cần phải định rõ:

- Kiểu dữ liệu của mảng: như **int, float, double…**
- Tên mảng: Là các ký tự từ a đến z, viết thường hoặc viết hoa
- Số chiều và kích thước của mỗi chiều được đặt trong dấu ngoặc vuông []

7.3.2. Khai báo mảng một chiều

Trước khi sử dụng mảng ta cần phải khai báo mảng, có một trong hai cách khai báo mảng như dưới đây:

7.3.2.1. Khai báo tường minh

Khai báo tường minh tức là khai báo rõ ràng số phần tử của mảng hay khai báo rõ ràng kích thước mảng có cú pháp như sau:

<kiểu><tên mảng><[số phần tử mảng]>;

Thí dụ: int R[5];

Khai báo một mảng kiểu int, tên mảng là R, là mảng một chiều có 5 phần tử: R[0], R[1], R[2], R[3], R[4].

7.3.2.2. Khai báo không tường minh

Khai báo không tường minh là khai báo không chỉ định rõ ràng kích thước của mảng như cú pháp sau:

<kiểu> <tên mảng> <[]>;

Trong dấu ngoặc vuông không chỉ rõ kích thước mảng, cách khai báo này chỉ được sử dụng trong các trường hợp sau:

- *Trường hợp vừa khai báo mảng vừa gán giá trị*

Cú pháp vừa khai báo mảng, vừa gán giá trị cho mảng như dưới đây:

<kiểu> <tên mảng> [] = { Dữ liệu gán };

Chú ý rằng dữ được gán phải nằm trong cặp dấu ngoặc móc{}, các phần tử được phân cách nhau bởi dấu (,). Kết thúc dòng khai báo mảng phải có dấu chấm phẩy (;)

Ngoài ra cần chú ý nếu phần tử mảng có dữ liệu kiểu chuỗi thì chuỗi đó phải nằm trong cặp dấu ngoặc kép (“”), thí dụ:

```
int a[] = {25,37,44,28};
char ten[] = {“Nguyen Van A”};
```

để rõ hơn ta sẽ xét thí dụ dưới đây:

Thí dụ: **Chương trình 7.1**

```
#include “stdio.h”
#include “conio.h”
main()
{
  int i, A[] = {25,37,44,28};
  clrscr();
  for(i=0; i<=3; i++)
        printf(“A[%d]=%d\t”,i,A[i]);
  getch();
}
```

Kết quả chương trình 7.1

- *Trường hợp sử dụng khai báo extern*

Khi muốn một mảng nào đó sẽ được sử dụng ở nhiều vị trí khác nhau (nhiều hàm khác nhau) trong chương trình thì nên sử dụng khai báo extern như cú pháp sau đây:

<extern> <kiểu> <tên mảng> [];

Sau khai báo này cũng phải gán giá trị cho từng phần tử mảng như thí dụ sau:

Chương trình 7.2

```
#include “stdio.h”
```

```
#include "conio.h"
extern int A[];
main()
{
int i;
clrscr();
for(i=0;i<=4;i++)
  printf("A[%d]=%d",i,A[i]);
getch();
}
int A[]={20,34,45,37,68}
```

Kết quả chương trình 7.2

```
Turbo C++ IDE
A[0]=20 A[1]=34 A[2]=45 A[3]=37 A[4]=68
```

7.3.3. Mảng nhiều chiều

Mảng có từ hai chiều trở lên người ta gọi là mảng nhiều chiều. Sau đây ta sẽ xét tới một số mảng nhiều chiều cơ bản:

7.3.3.1. Mảng hai chiều:

Ta đã biết mảng một chiều là các mảng chỉ có một chỉ số. Mảng hai chiều sẽ có hai chỉ số và được đặt trong hai dấu ngoặc vuông []. Một mảng hai chiều có thể được xem như là một mảng của hai mảng một chiều. Một mảng hai chiều đặc trưng như bảng tra cứu giá trị điện trở thông qua mầu tương ứng, để xác định thông tin ta sẽ chỉ định dòng và cột cần thiết, thông tin được đọc ra từ vị trí dòng và cột được tìm thấy. Tương tự như vậy, mảng hai chiều là một khung lưới chứa các dòng và cột trong đó mỗi phần tử được xác định duy nhất bằng tọa độ dòng và cột của nó. Một mảng hai chiều **led** có kiểu int với 2 dòng và 3 cột có thể được khai báo như sau:

int led[2][3];

Mảng này sẽ chứa 2x3 (6) phần tử và chúng có thể được biểu diễn như sau:

Dòng / Cột	0	1	2
0	e1	e2	e3
1	e4	e5	e6

Trong đó e1 ÷ e6 biểu diễn cho các phần tử của mảng. Cả dòng và cột được đánh số từ 0. Phần tử e6 được xác định bằng dòng 1 cột 2 truy xuất đến phần tử này như sau: led[1][2].

7.3.3.2. Mảng đa chiều

Khai báo mảng đa chiều có thể kết hợp với việc gán các giá trị khởi tạo. Cần lưu ý đến thứ tự các giá trị khởi tạo được gán cho các phần tử của mảng (chỉ có mảng **external** và **static** có thể được khởi tạo). Các phần tử trong dòng đầu tiên của mảng hai chiều sẽ được gán giá trị trước, sau đó đến các phần tử của dòng thứ hai, xem thí dụ sau:

int arr[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};

kết quả của khai báo trên sẽ là như sau:

arr[0][0]=1; arr[0][1]=2; arr[0][2]=3;arr[0][3]=4;

arr[1][0]=5; arr[1][1]=6; arr[1][2]=7;arr[1][3]=8;

arr[2][0]=9; arr[2][1]=10; arr[2][2]=11;arr[2][3]=12;

Chú ý rằng chỉ số thứ nhất chạy từ 0÷2 và chỉ số thứ hai chạy từ 0÷3. Một điểm cần nhớ là các phần tử của mảng sẽ được lưu trữ ở những vị trí kế tiếp nhau trong bộ nhớ. Mảng arr ở trên có thể xem như là một mảng 3 phần tử, mỗi phần tử là một mảng của 4 số nguyên và sẽ xuất hiện như sau:

Dòng 0				Dòng 1				Dòng 2			
									10	11	12

Thứ tự tự nhiên mà các giá trị khởi tạo được gán có thể thay đổi bằng hình thức nhóm các giá trị khởi tạo lại trong các dấu ngoặc nhọn {} như trong thí dụ sau:

```
int arr[3][4]=
{
        {1,2,3},
        {4,5,6},
        {7,8,9}
}
```

Mảng sẽ được khởi tạo như sau:

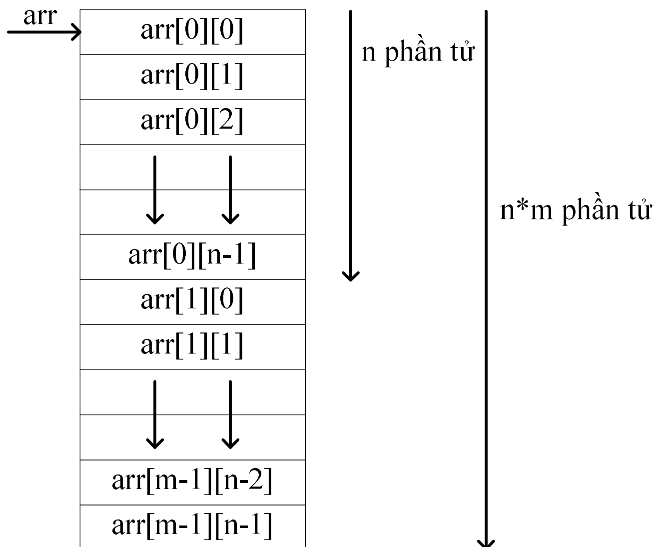
arr[0][0]=1; arr[0][1]=2; arr[0][2]=3;arr[0][3]=0;

arr[1][0]=5; arr[1][1]=6; arr[1][2]=7;arr[1][3]=0;

arr[2][0]=9; arr[2][1]=10; arr[2][2]=11;arr[2][3]=0;

Một phần tử của mảng đa chiều có thể được sử dụng như một biến trong C bằng cách dùng các chỉ số để xác định phần tử của mảng.

Hình 7.3 dưới đây sẽ cho biết sự phân bố trong bộ nhớ của một mảng hai chiều.



Hình 7.18: Sự phân bố trong bộ nhớ của một mảng hai chiều

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;

Thí dụ

Thí dụ dưới đây hiển thị sự phân bố của mảng 3*4. Kết quả chỉ ra rằng địa chỉ cơ sở của mảng là FFC6h. 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 7.5 phác thảo sự phân bố bộ nhớ dùng cho kết quả chương trình thí dụ này.

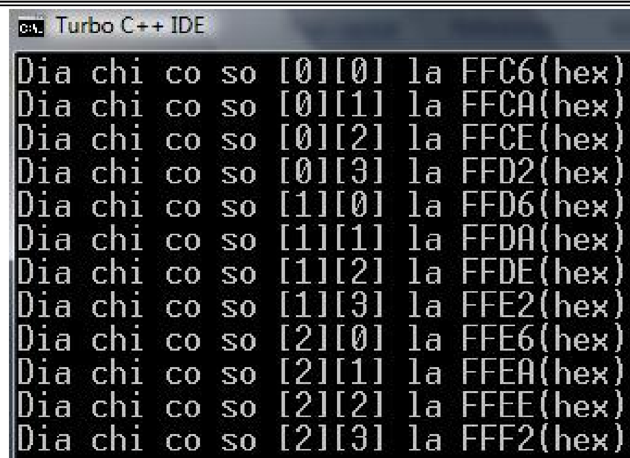
Chương trình 7.3

```
/* program 7_3.C */
#include "stdio.h"
#include "conio.h"
int main(void)
{
  float arr[3][4];
  int hang,cot;
  clrscr();
  for (hang = 0; hang < 3; hang++)
       for(cot = 0; cot <4; cot++)
```

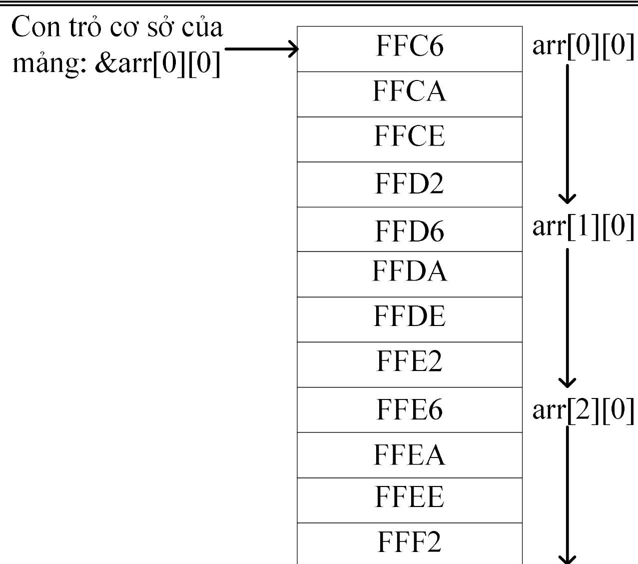
```
        printf("Dia chi co so [%d][%d]\
la %p(hex)\n",hang, cot, &arr[hang][cot]);

getch();
}
```

Kết quả chương trình 7.3



Hình 7.19: Kết quả chương trình 7.3



Hình 7.20: phân bố bộ nhớ cho chương trình

7.4. Sử dụng mảng một chiều và mảng hai chiều

7.4.1. Sử dụng mảng một chiều

Mảng một chiều có thể được sử dụng để lưu trữ một tập các giá trị có cùng kiểu dữ liệu. Xét một tập giá trị các điện trở trong một phòng thực hành. Chúng ta sẽ sắp xếp các giá trị này theo thứ tự giảm dần giá trị.

Các bước sắp xếp mảng một chiều điện trở theo thứ tự giảm dần của các giá trị như sau:

Bước 1: Nhập vào số lượng các điện trở

Để làm được điều này một biến phải được khai báo và giá trị của biến phải được nhập. Mã lệnh như sau:

```
int n;
printf("\nnhap vao so luong cac dien tro :");
scanf("%d",&n);
```

Bước 2: Nhập vào tập các phần tử của mảng điện trở

Để nhập vào tập các giá trị cho một mảng, mảng phải được khai báo. Mã lệnh như sau:

```
int R[100];
```

Số phần tử của mảng được xác định bằng giá trị đã nhập vào biến n. n phần tử của mảng phải được khởi tạo giá trị. Để nhập n giá trị dùng vòng lặp for. Một biến nguyên cần được khai báo để sử dụng như là chỉ số của mảng. Biến này giúp truy xuất từng phần tử của mảng. Sau đó giá trị của các phần tử mảng được khởi tạo bằng cách nhận các giá trị nhập vào từ người dùng. Mã lệnh như sau:

```
int l;
for (l=0; l<100; l++)
{
printf("nhap vao phan tu thu %d la: ",l+1);
scanf("%d",&R[l]);
}
```

Vì các chỉ số của mảng luôn bắt đầu từ 0 nên chúng ta cùng khởi tạo biến l là 0. Mỗi khi vòng lặp được thực thi, một giá trị nguyên được gán đến một phần tử của mảng.

Bước 3: Tạo một bản sao của mảng điện trở

Trước khi sắp xếp mảng, tốt hơn là nên giữ lại mảng gốc. Vì vậy một mảng khác được khai báo và các phần tử của mảng thứ nhất có thể được sao chép vào mảng mới này. Các dòng mã lệnh sau được sử dụng để thực hiện điều này:

```
int desR[100], k;
for(k = 0; k < n; k++)
desR[k] = R[k];
```

Bước 4: Sắp xếp mảng theo thứ tự giảm dần

Để sắp xếp một mảng, các phần tử trong mảng cần phải được so sánh với những phần tử còn lại. Cách tốt nhất để sắp xếp một mảng, theo thứ tự giảm dần, là chọn ra giá trị lớn nhất trong mảng và hoán vị nó với phần tử đầu tiên. Một khi điều này được thực hiện xong, giá trị lớn thứ hai trong mảng có thể được hoán vị với phần tử thứ hai của mảng, phần tử đầu tiên của mảng được bỏ qua vì nó đã là phần tử lớn nhất. Tương tự, các phần tử của mảng được loại ra tuần tự đến khi phần tử lớn thứ n được tìm thấy.

Trong trường hợp mảng cần sắp xếp theo thứ tự tăng dần giá trị lớn nhất sẽ được hoán vị với phần tử cuối cùng của mảng.

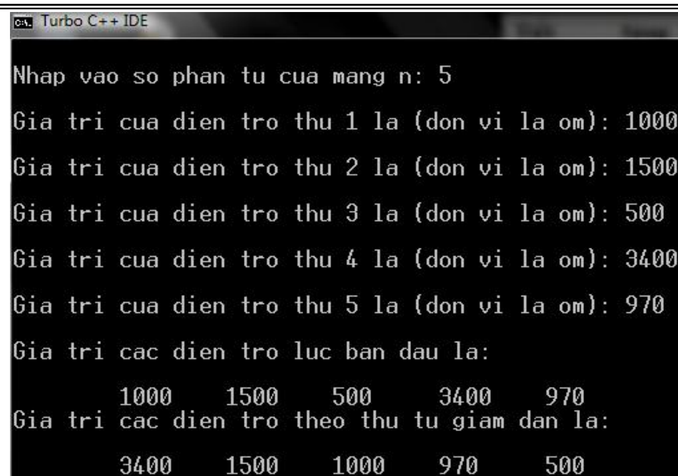
Thí dụ: Nhập vào một mảng điện trở gồm n điện trở, n được nhập vào từ bàn phím, với n giá trị tương ứng; sau đó sắp xếp mảng điện trở theo thứ tự giảm dần, hiển thị lên màn hình giá trị của mảng trước và sau khi sắp xếp

Chương trình 7.4:

```
//Chuong trinh thi du 7_4.C
//Chuong trinh nhap mang n dien tro va n gia tri tuong ung
//Sap xep lai mang dien tro theo thu tu giam dan cua gia tri
#include "stdio.h"
#include "conio.h"
void main()
{
  int R[100];
  int desR[100];
  int k,i,n,l,m,tg;
  clrscr();
//1. nhap vao so luong cac dien tro
  printf("\nNhap vao so phan tu cua mang n: ");scanf("%d",&n);
//2. nhap vao tap cac phan tu cua mang dien tro
  for (i=0;i<n; i++)
  {
        printf("\nGia tri cua dien tro thu %d la (don vi la om): ",i+1);
        scanf("%d",&R[i]);
  }
//3. Tao mot ban sao cua mang dien tro
  for(k=0; k<n; k++)
  desR[k] = R[k];
//4. Sap xep mang theo thu tu giam dan
  for(l=0;l<n-1;l++)
  {
        for (m=l+1;m<n;m++)
        {
              if(desR[l]<desR[m])
              {
                    tg = desR[l];
                    desR[l] = desR[m];
```

```
                    desR[m] = tg;
                }
            }
    }
printf("\nGia tri cac dien tro luc ban dau la:\n\n");
for (i=0;i<n;i++)
printf("\t%d",R[i]);
printf("\nGia tri cac dien tro theo thu tu giam dan la:\n\n");
for (i=0;i<n;i++)
printf("\t%d",desR[i]);
getch();
}
```

Kết quả chương trình 7.4



7.4.2. Sử dụng mảng hai chiều để cộng ma trận

Các mảng có thể có nhiều chiều. Một ví dụ tiêu biểu của mảng hai chiều là ma trận. Một ma trận được tạo bởi các dòng và các cột. Giao điểm của mỗi dòng và cột có một giá trị. Hình 7.4 biểu diễn một ma trận.

D	Cột		
òng	1	2	3
1	a[1][1]=2	a[2][1]=1	a[3][1]=8
2	a[1][2]=3	a[2][2]=6	a[3][2]=7
3	a[1][3]=12	a[2][3]=9	a[3][3]=10

Hình 7.21 Biểu diễ một ma trận 2 chiều

Số dòng và cột trong ma trận khi được biểu diễn dạng (số dòng) x (số cột) được gọi là kích thước của ma trận. Kích thước của ma trận trong hình 7.4 là 3x3.

Chúng ta hãy xem ví dụ cộng ma trận để hiểu cách sử dụng của mảng hai chiều. Quan sát hai ma trận A và B trong dưới đây:

A	1	2	3
1	2	3	5
2	1	2	3
3	4	6	7

B	1	2	3
1	7	5	2
2	1	2	1
3	8	3	1

Tổng của hai ma trận này là một ma trận khác. Nó được tạo từ việc cộng các giá trị tại mỗi dòng và cột tương ứng. Ví dụ. phần tử đầu tiên C(1,1) trong ma trận C sẽ là tổng của A(1,1) và B(1,1). Phần tử thứ hai C(1,2) sẽ là tổng của A(1,2) và B(1,2) ... Một qui luật quan trọng trong việc cộng các ma trận là kích thước của các ma trận phải giống nhau. Nghĩa là, một ma trận 2x3 chỉ có thể được cộng với một ma trận 2x3. Hình sau Biểu diễn ma trận A, B và C

A	1	2	3
1	2	3	5
2	1	2	3
3	4	6	7

B	1	2	3
1	7	5	2
2	1	2	1
3	8	3	1

C	1	2	3
2	9	8	7
2	2	4	4
3	7	9	8

Để giải quyết bài toán này ta cần làm thông qua các bước sau:

Bước 1: khai báo 3 mảng, mã lệnh như sau:

```
int A[10][10], B[10][10], C[10][10];
```

Bước 2: Nhập vào kích thức của các ma trận. Mã lệnh là:

```
int hang, cot;
printf("\nNhap vao kich thuoc ma tran");
scanf("%d  %d",&hang,&cot);
```

Bước 3: Nhập các giá trị cho hai ma trận A và B

Các giá trị của ma trận được nhập theo dòng. Trước tiên các giá trị của dòng thứ nhất được nhập vào. Kế đến các giá trị của dòng thứ hai được nhập, ... Bên trong một dòng, các giá trị của cột được nhập tuần tự. Vì vậy cần hai vòng lặp để nhập các giá trị của một ma trận. Vòng lặp thứ nhất đi qua từng dòng một, trong khi vòng lặp bên trong sẽ đi qua từng cột.

Đoạn mã lệnh như sau:

```
printf("\n Nhap vao gia tri cua ma tran A & B : \n");
int i, j;
for(i = 0; i < hang; i++)
  for(j = 0; j < cot; j++)
{
print("A[%d,%d], B[%d,%d]:", i,j,i,j);
       scanf("%d %d", &A[i][j], &B[i][j]);
}
```

Bước 4: Cộng hai ma trận.

Hai ma trận có thể được cộng bằng cách sử dụng đoạn mã lệnh sau:

C[i][j] = A[i][j] + B[i][j];

Chú ý, dòng lệnh này cần đặt ở vòng lặp bên trong của đoạn lệnh đã nói ở trên. Một cách khác, hai vòng lặp có thể được viết lại để cộng ma trận.

Bước 5: Hiển thị 3 ma trận.

Mã lệnh hiển thị 3 ma trận sẽ như sau:

```
for(i = 0; i < row; i++)
for(j = 0;j < col; j++)
{
printf("\nA[%d,%d]=%d, B[%d,%d] = %d, C[%d,%d]=%d \n",
i, j, A[i][j], i, j, B[i][j], i, j, C[i][j]);
}
```

Chương trình 7.5

```
//Chuong trinh tinh tong hai ma tran hai chieu
#include "stdio.h"
#include "conio.h"
//chuong trinh chinh
void main()
{
//khai bao
  int A[10][10],B[10][10],C[10][10],i,j,hang,cot;
//xoa man hinh
  clrscr();
//Nhap so hang va so cot
  printf("Nhap vao so hang va so cot cua hai ma tran:");
  printf("\nSo hang: "); scanf("%d",&hang);
  printf("\nSo cot:  "); scanf("%d",&cot);
//Nhap vao cac phan tu cua ma tran A
  printf("\nNhap vao ma tran A");
  for(i=0;i<hang;i++)
       {
       for(j=0;j<cot;j++)
            {
            printf("\nGia tri cua phan tu A[%d][%d] = ",i+1,j+1);
            scanf("%d",&A[i][j]);
            }
```

```
        }
//Nhap vao cac phan tu cua ma tran B
  printf("\nNhap vao ma tran B:");
  for(i=0;i<hang;i++)
        {
        for(j=0;j<cot;j++)
              {
              printf("\nGia tri cua phan tu B[%d][%d] = ",i+1,j+1);
              scanf("%d",&B[i][j]);
              }
        }
//Tinh tong cua hai ma tran C=A+B
  for(i=0;i<hang;i++)
        for(j=0;j<cot;j++)
              C[i][j]=A[i][j]+B[i][j];
//Hien thi cac ma tran
printf("\nma tran A\tMa tran B\tMa tran C la:");
  for(i=0;i<hang;i++)
        for(j=0;j<cot;j++)

  printf("\nA[%d][%d]=%d\tB[%d][%d]=%d\tC[%d][%d]=%d",i,j,A[i][j],i,j,B[i][j],
i,j,C[i][j]);

getch();
}
```

Kết quả chương trình 7.5

```
Turbo C++ IDE
Nhap vao so hang va so cot cua hai ma tran:
So hang: 2
So cot:  2

Nhap vao ma tran A:
Gia tri cua phan tu A[1][1] = 1
Gia tri cua phan tu A[1][2] = 2
Gia tri cua phan tu A[2][1] = 3
Gia tri cua phan tu A[2][2] = 4

Nhap vao ma tran B:
Gia tri cua phan tu B[1][1] = 5
Gia tri cua phan tu B[1][2] = 6
Gia tri cua phan tu B[2][1] = 7
Gia tri cua phan tu B[2][2] = 8

ma tran A       Ma tran B       Ma tran C la:
A[0][0]=1       B[0][0]=5       C[0][0]=6
A[0][1]=2       B[0][1]=6       C[0][1]=8
A[1][0]=3       B[1][0]=7       C[1][0]=10
A[1][1]=4       B[1][1]=8       C[1][1]=12
```

7.5. Con trỏ và mảng

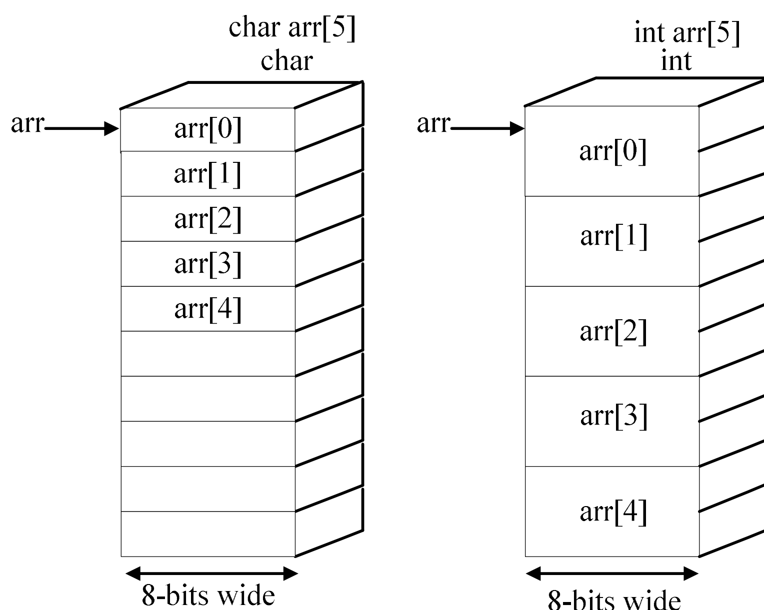
7.5.1. Mối quan hệ giữa con trỏ và mảng

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 trao đổi lẫn nhau như thế nào.

Hình 7.7 Mô tả hai loại khai báo mảng đối với mảng arr. Mỗi loại có 5 phần tử: thứ nhất là arr[0] và cuối cùng là arr[4]. Số byte được phân bố 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 arr 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 7.20: Mối quan hệ giữa mảng và con trỏ

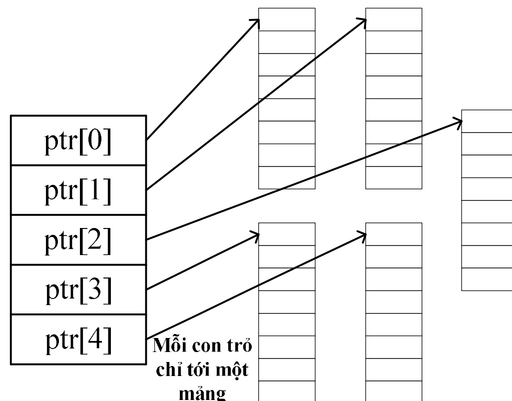
Mảng được dung	Con trỏ được dùng
float arr[10]	float *arr
	arr = (float *) malloc(10*sizeof(float));
arr[0]	*(arr)
arr[1]	*(arr+1)
arr[2]	*(arr+2)
arr[9]	*(arr+9)



Hình 7.22. Các phần tử mảng

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 sử chạy sử dụng một hàm phân bố bộ nhớ, như malloc(). Hình 7.8 chỉ ra một mảng của 5 con trỏ; mỗi con trỏ trong số này chỉ vào một mảng của 8 dấu phảy động

```
float *ptr[5]
for (i=0; i<5; i++)
  ptr[i] = (float *) malloc (8*sizeof(float));
```



Hình 7.23: Mảng của những con trỏ

7.5.2. Con trỏ và mảng một chiều

Tên của một mảng thật ra là một con trỏ trỏ đến phần tử đầu tiên của mảng đó. Vì vậy, nếu **ary** là một mảng một chiều, thì địa chỉ của phần tử đầu tiên trong mảng có thể được biểu diễn là **&ary[0]** hoặc đơn giản chỉ là **ary**. Tương tự, địa chỉ của phần tử mảng thứ hai có thể được viết như **&ary[1]** hoặc **ary+1**,... Tổng quát, địa chỉ của phần tử mảng thứ (i + 1) có thể được biểu diễn là **&ary[i]** hay **(ary+i)**. Như vậy, địa chỉ của một phần tử mảng bất kỳ có thể được biểu diễn theo hai cách:

- sử dụng ký hiệu & trước một phần tử mảng
- sử dụng một biểu thức trong đó chỉ số được cộng vào tên của mảng

Ghi nhớ rằng trong biểu thức **(ary + i), ary** tượng trưng cho một địa chỉ, trong khi **i** biểu diễn số nguyên. Hơn thế nữa, **ary** là tên của một mảng mà các phần tử có thể là kiểu số nguyên, ký tự, số thập phân,… (dĩ nhiên, tất cả các phần tử của mảng phải có cùng kiểu dữ liệu). Vì vậy, biểu thức ở trên không chỉ là một phép cộng; nó thật ra là xác định một địa chỉ, một số xác định của các ô nhớ . Biểu thức **(ary + i)** là một sự trình bày cho một địa chỉ chứ không phải là một biểu thức toán học.

Như đã nói ở trước, số lượng ô nhớ được kết hợp với một mảng sẽ tùy thuộc vào kiểu dữ liệu của mảng cũng như là kiến trúc của máy tính. Tuy nhiên, người lập trình chỉ có thể xác định địa chỉ của phần tử mảng đầu tiên, đó là tên của mảng (trong

trường hợp này là **ary**) và số các phần tử tiếp sau phần tử đầu tiên, đó là, một giá trị chỉ số. Giá trị của **i** đôi khi được xem như là một **độ dời** khi được dùng theo cách này.

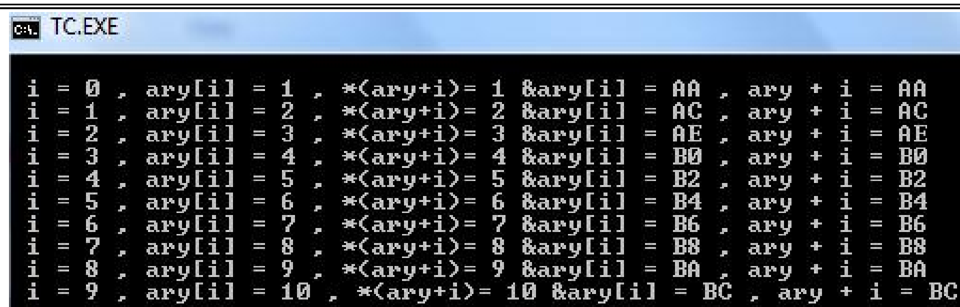
Các biểu thức **&ary[i]** và **(ary+i)** biểu diễn địa chỉ phần tử thứ **i** của **ary**, và như vậy một cách logic là cả **ary[i]** và ***(ary + i)** đều biểu diễn nội dung của địa chỉ đó, nghĩa là, giá trị của phần tử thứ **i** trong mảng **ary**. Cả hai cách có thể thay thế cho nhau và được sử dụng trong bất kỳ ứng dụng nào khi người lập trình mong muốn.

Chương trình sau đây biểu diễn mối quan hệ giữa các phần tử mảng và địa chỉ của chúng.

Chương trình 7.6

```
#include<stdio.h>
void main()
{
static int ary[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int i;
for (i = 0; i < 10; i ++)
{
printf("\n i = %d , ary[i] = %d , *(ary+i)= %d ", i,
                                        ary[i], *(ary + i));
printf("&ary[i] = %X , ary + i = %X", &ary[i], ary + i);
/* %X gives unsigned hexadecimal */
}
}
```

Chương trình trên định nghĩa mảng một chiều **ary**, có 10 phần tử kiểu số nguyên, các phần tử mảng được gán giá trị tương ứng là 1, 2, ..10. Vòng lặp **for** được dùng để hiển thị giá trị và địa chỉ tương ứng của mỗi phần tử mảng. Chú ý rằng, giá trị của mỗi phần tử được xác định theo hai cách khác nhau, ary[i] và *(ary + i), nhằm minh họa sự tương đương của chúng. Tương tự, địa chỉ của mỗi phần tử mảng cũng được hiển thị theo hai cách. Kết quả của chương trình như sau:



Kết quả này trình bày rõ ràng sự khác nhau giữa ary[i] và &ary[i].

ary[i]: đưa ra giá trị của phần tử thứ i của mảng ary

&ary[i]: đưa ra giá trị của địa chỉ của phầnt tử thứ i của mảng ary.

7.5.3. Con trỏ và mảng nhiều chiều

Một mảng nhiều chiều cũng có thể được biểu diễn dưới dạng con trỏ của mảng một chiều (tên của mảng) và một độ dời (chỉ số). Thực hiện được điều này là bởi vì một mảng nhiều chiều là một tập hợp của các mảng một chiều.Ví dụ, một mảng hai chiều có thể được định nghĩa như là một con trỏ đến một nhóm các mảng một chiều kế tiếp nhau. Cú pháp báo mảng hai chiều có thể viết như sau:

data_type (*ptr_var)[expr 2];

thay vì

data_type array[expr 1][expr 2];

Khái niệm này có thể được tổng quát hóa cho các mảng nhiều chiều, đó là,

data_type (*ptr_var)[exp 2] .... [exp N];

thay vì

data_type array[exp 1][exp 2] ... [exp N];

Trong các khai báo trên, data_type là kiểu dữ liệu của mảng, ptr_var là tên của biến con trỏ, array là tên mảng, và exp 1, exp 2, exp 3, ... exp N là các giá trị nguyên dương xác định số lượng tối đa các phần tử mảng được kết hợp với mỗi chỉ số.

Chú ý dấu ngoặc () bao quanh tên mảng và dấu * phía trước tên mảng trong cách khai báo theo dạng con trỏ. Cặp dấu ngoặc () là không thể thiếu, ngược lại cú pháp khai báo sẽ khai báo một mảng của các con trỏ chứ không phải một con trỏ của một nhóm các mảng.

Ví dụ, nếu **ary** là một mảng hai chiều có 10 dòng và 20 cột, nó có thể được khai báo như sau:

int (*ary)[20];

thay vì

int ary[10][20];

Trong sự khai báo thứ nhất, **ary** được định nghĩa là một con trỏ trỏ tới một nhóm các mảng một chiều liên tiếp nhau, mỗi mảng có 20 phần tử kiểu số nguyên. Vì vậy, **ary** trỏ đến phần tử đầu tiên của mảng, đó là dòng đầu tiên (dòng 0) của mảng hai chiều. Tương tự, (ary + 1) trỏ đến dòng thứ hai của mảng hai chiều, ...

Một mảng thập phân ba chiều **fl_ary** có thể được khai báo như:

float (*fl_ary)[20][30];

thay vì

float fl_ary[10][20][30];

Trong khai báo đầu, fl_ary được định nghĩa như là một nhóm các mảng thập phân hai chiều có kích thước 20 x 30 liên tiếp nhau. Vì vậy, fl_ary trỏ đến mảng 20 x 30 đầu tiên, (fl_ary + 1) trỏ đến mảng 20 x 30 thứ hai,...

Trong mảng hai chiều **ary**, phần tử tại dòng 4 và cột 9 có thể được truy xuất sử dụng câu lệnh:

ary[3][8];

hoặc

((ary + 3) + 8);

Cách thứ nhất là cách thường được dùng. Trong cách thứ hai, (ary + 3) là một con trỏ trỏ đến dòng thứ 4. Vì vậy, đối tượng của con trỏ này, *(ary + 3), tham chiếu đến toàn bộ dòng. Vì dòng 3 là một mảng một chiều, *(ary + 3) là một con trỏ trỏ đến phần tử đầu tiên trong dòng 3, sau đó 8 được cộng vào con trỏ. Vì vậy, *(*(ary + 3) + 8) là một con trỏ trỏ đến phần tử 8 (phần tử thứ 9) trong dòng thứ 4. Vì vậy đối tượng của con trỏ này, *(*(ary + 3) + 8), tham chiếu đến tham chiếu đến phần tử trong cột thứ 9 của dòng thứ 4, đó là ary [3][8].

Có nhiều cách thức để định nghĩa mảng, và có nhiều cách để xử lý các phần tử mảng. Lựa chọn cách thức nào tùy thuộc vào người dùng. Tuy nhiên, trong các ứng dụng có các mảng dạng số, định nghĩa mảng theo cách thông thường sẽ dễ dàng hơn.

7.5.4. Một số thí dụ

7.5.4.1. Thí dụ về trở mạch điện Boole

Thí dụ sau 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 7.10 chỉ ra mảng trang_thai_vao[] 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.

```
int trang_thai_vao[3][6]=
{
{1,1,0,0,1,1},
{0,1,0,1,0,1},
{1,0,0,0,0,0}
};
```

	cột 0					cột 5
Hàng 0	1	1	0	0	1	1
	0	1	0	1	0	1
Hàng 2	1	0	0	0	0	0

trang_thai_vao[2][0]   trang_thai_vao[1][5]

Hình 7.24: Các phần tử của mảng

Chương trình 7.7

//chuong trinh 7_7.C/

```
#include "stdio.h"
#include "conio.h"
void main()
{
  int in[3][6]=
  {{1,1,0,0,1,1},{0,1,0,1,0,1},{1,0,0,0,0,0}};
  int i,out[1];
  clrscr();
  for(i=0;i<6;i++)
  out[i]=(in[0][i]&&in[1][i])||in[2][i];
  printf("kq loi ra tuong ung voi bieu thuc Out=(in1.in2)+in3 la: \n");
  puts("\tin1\tin2\tin3\tOut");
  for (i=0;i<6;i++)
  printf("\t %d\t %d\t %d\t %d\n",in[0][i],in[1][i],in[2][i],out[i]);
getch();
}
```

Kết quả chương trình:

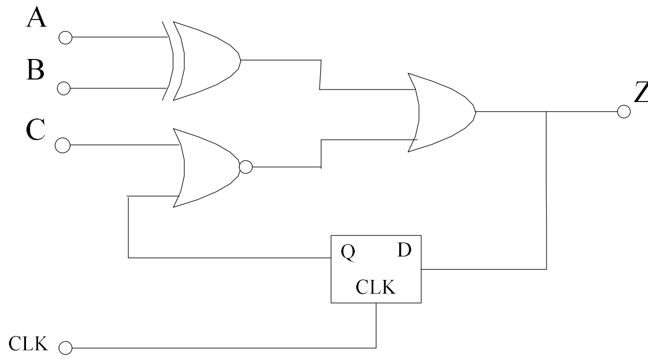
```
Turbo C++ IDE
kq loi ra tuong ung voi bieu thuc Out=(in1.in2)+in3 la:
        in1     in2     in3     Out
         1       0       1       1
         1       1       0       1
         0       0       0       0
         0       1       0       0
         1       0       0       0
         1       1       0       1
```

7.5.4.2. Mô phỏng logic

thí dụ này mô phỏng một mạch logic 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 theo phương trình đại số Boole:

$$Z_n = A \oplus B + \overline{(C + Z_{n-1})}$$

ở đây Z_n là trạng thái đầu ra hiện tại và Z_{n-1} là trạng thái đầu ra trước đấy. Hình 7.6 mô tả phương trình bằng một sơ đồ điện



Hình 7.25: Biểu diễn hàm boole bằng sơ đồ

Chương trình 7.8

```
#include "stdio.h"
#include "conio.h"
#define so_cac_trang_thai 7
void in_bang_chan_li(int a[],int b[],int c[],int q[]);
void xu_li_cac_trang_thai(int a[],int b[],int c[],int q[]);
int main(void)
{
int A[so_cac_trang_thai]={1,1,1,0,0,1,0};
int B[so_cac_trang_thai]={1,0,1,1,1,0,0};
int C[so_cac_trang_thai]={0,1,0,1,0,1,0};
int Q[so_cac_trang_thai];
clrscr();
xu_li_cac_trang_thai(A,B,C,Q);
in_bang_chan_li(A,B,C,Q);
getch();
return(0);
}
void xu_li_cac_trang_thai(int a[],int b[],int c[],int q[])
{
int i;
q[0]=(a[0]^b[0]||!(c[0]||q[0]));
for (i=1;i<so_cac_trang_thai;i++)
q[i]=(a[i]^b[i]|| !(c[i]||q[i-1]));
}
void in_bang_chan_li(int a[],int b[],int c[],int q[])
{
int i;
```

```
printf("\n A\tB\tC\tQ\n");
for (i=0;i<so_cac_trang_thai;i++)
printf(" %d\t%d\t%d\t%d\n",a[i],b[i],c[i],q[i]);
}
```

Kết quả chương trình 7.8

```
TC.EXE

A       B       C       Q
1       1       0       0
1       0       1       1
1       1       0       0
0       1       1       1
0       1       0       1
1       0       1       1
0       0       0       0
```

7.6. Các biến và hằng kiểu chuỗi

Các biến chuỗi được sử dụng để lưu trữ một chuỗi các ký tự. Như các biến khác, các biến này phải được khai báo trước khi sử dụng.

Thí dụ khai báo một biến chuỗi

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

str là một mảng các ký tự, có thể lưu tối đa 10 ký tự. Giả sử được gán một hằng chỗi: “DIEN TRO”.

Một **hằng chuỗi** là một dãy các ký tự nằm trong dấu nháy kép. Mỗi ký tự trong một chuỗi được lưu trữ như là phần tử của mảng. Trong bộ nhớ, chuỗi được lưu trữ như sau:

‘D’	‘I’	‘E’	‘N’	‘ ’	‘T’	‘R’	‘O’	‘\0’	

Ký tự ‘\0’ (null) được tự động thêm vào trong cách biểu diễn bên trong của chuỗi để đánh dấu điểm kết thúc chuỗi. Vì vậy khi khai báo một chuỗi, phải tăng kích thước của nó lên một phần tử để chứa ký hiệu kết thúc null.

7.7. Các thao tác nhập/xuất chuỗi

Các thao tác nhập/xuất (I/O) chuỗi trong C được thực hiện bằng cách gọi các hàm. Các hàm này là một phần của thư viện nhập/xuất chuẩn tên **stdio.h**. Một chương trình muốn sử dụng các hàm nhập/xuất chuỗi phải có câu lệnh khai báo sau ở đầu chương trình:

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

Khi chương trình có chứa câu lệnh này được biên dịch, thì nội dung của tập tin **stdio.h** sẽ trở thành một phần của chương trình.

7.7.1. Thao tác nhập/xuất chuỗi đơn giản

Sử dụng hàm **gets()** là cách đơn giản nhất để nhập một chuỗi thông qua thiết bị nhập chuẩn. Các ký tự sẽ được nhập vào cho đến khi nhấn phím Enter. Hàm **gets()** thay thế ký tự kết thúc trở về đầu dòng ‘\n’ bằng ký tự ‘\0’. Cú pháp hàm này như sau:

gets(str);

Trong đó **str** là một mảng ký tự đã được khai báo.

Tương tự, hàm **puts()** được sử dụng để hiển thị một chuỗi ra thiết bị xuất chuẩn. Ký tự xuống dòng sẽ kết thúc việc xuất chuỗi. Cú pháp hàm như sau:

puts(str);

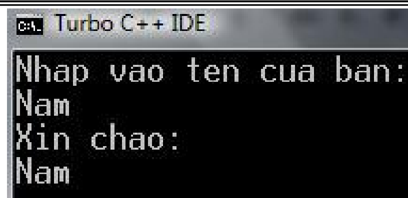
Trong đó **str** là một mảng ký tự đã được khai báo và khởi tạo. Chương trình sau đây nhận vào một tên và hiển thị một thông báo.

Chương trình 7.9:

```
#include “stdio.h”
#include “conio.h”
void main()
{
  char name[20];
  clrscr();                /* lenh xoa man hinh */
puts("Nhap vao ten cua ban:");
gets(name);
puts("Xin chao: ");
puts(name);                /* Hien thi ten vua nhap */
getch();
}
```

Nếu tên Nam được nhập vào, chương trình trên cho ra kết quả:

Kết quả chương trình 7.9



7.7.2. Thao tác nhập/xuất chuỗi có định dạng

Có thể sử đụng các hàm **scanf()** và **printf()** để nhập và hiển thị các giá trị chuỗi. Các hàm này được dùng để nhập và hiển thị các kiểu dữ liệu hỗn hợp trong một câu lệnh duy nhất. Cú pháp để nhập một chuỗi như sau:

scanf(“%s”, str);

Trong đó ký hiệu định dạng **%s** cho biết rằng một giá trị chuỗi sẽ được nhập vào. **str** là một mảng ký tự đã được khai báo. Tương tự, để hiển thị chuỗi, cú pháp sẽ là:

printf(“%s”, str);

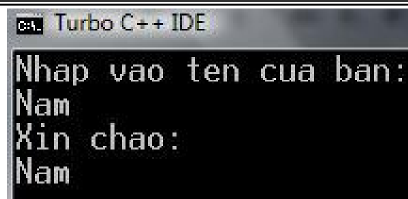
Trong đó ký hiệu định dạng **%s** cho biết rằng một giá trị chuỗi sẽ được hiển thị và **str** là một mảng ký tự đã được khai báo và khởi tạo. Hàm printf() có thể dùng để hiển thị ra các thông báo mà không cần kí tự định dạng.

Có thể sửa đổi chương trình bên trên để nhập vào và hiển thị một tên, sử dụng hàm scanf() và printf().

Chương trình 7.10

```
#include “stdio.h”
#include “conio.h”
void main()
{
  char name[20];
  clrscr();                    /* xoa man hinh */
printf("Nhap ten: ");          /*Nhap ten*/
scanf(“%s”, name);                  /* cho phep nhap */
printf("Xin chao: %s", name);  /*Hien thi ten*/
getch();
}
```

Nếu nhập vào tên Nam , chương trình trên cho ra kết quả:



7.8. Sử dụng các hàm về chuỗi

Trước kh đi tìm hiểu về cách sử dụng các hàm về chuỗi, ta đi về từng hàm được dùng để xử lý chuỗi

7.8.1. Hàm strcat()

Hàm strcat() được dùng để nối hai chuỗi thành một chuỗi. Cú pháp là:

strcat(str1,str2);

trong đó: str1, str2 là hai chuỗi đã được khai báo và khởi tạo. Hàm này sẽ thực hiện nối chuỗi str2 và sau chuỗi str1.

Thí dụ viết chương trình nhập họ và nhập tên, dùng hàm strcat() để nối chúng vào thành họ tên đầy đủ.

Chương trình 7.11

```
#include "stdio.h"
#include "conio.h"
#include "string.h"
void main()
```

```
{
  char ho[30],dem[10],ten[10];
  printf("\nNhap vao ten ho cua ban: ");gets(ho);
  printf("\nNhap vao ten dem cua ban: ");gets(dem);
  printf("\nNhap vao ten goi cua ban: ");gets(ten);
  strcat(ho,dem);
  strcat(ho,ten);
  printf("Ho ten day du cua ban la:%s ",ho);
getch();
}
```

Kết quả chương trình7.11

```
Turbo C++ IDE

Nhap vao ten ho cua ban:  Dinh

Nhap vao ten dem cua ban:  Thai

Nhap vao ten goi cua ban:  An

Ho ten day du cua ban la: Dinh Thai An
```

7.8.2. Hàm strcmp()

Việc so sánh hai số (bằng nhau hay không bằng nhau) có thể thực hiện bằng cách sử dụng các toán tử quan hệ. Tuy nhiên để so sánh hai chuỗi kí tự, ta phải dùng tới hàm strcmp(). Cú pháp của hàm này như sau:

strcmp(str1,str2);

trong đó str1, str2 là hai chuỗi đã được khai báo và khởi tạo. Hàm trả về giá trị:

- ✓ Nhỏ hơn 0 nếu str1 < str2
- ✓ Bằng 0 nếu str1 = str2
- ✓ Lớn hơn 0 nếu str1 > str2

7.8.3. Hàm strchr()

Hàm strchr() xác định vị trí xuất hiện của một ký tự trong một chuỗi. Cú pháp là:

strchr(str,chr);

trong đó str là một mảng kí tự hay chuỗi, chr là một biến kí tự chứa giá trị cần tìm. Hàm trả về con trỏ, trỏ đến giá trị tìm được đầu tiên trong chuỗi, hoặc NULL nếu không tìm thấy.

7.8.4. Hàm strcpy()

Trong trường hợp ta cần gán một chuỗi này cho một chuỗi khác thì ta cần dùng tới hàm strcpy(). Cú pháp của hàm này là:

strcpy(str1,str2);

trong đó str1, str2 là hai mảng kí tự đã được khai báo và khởi tạo. Hàm sao chép giá trị str2 vào str1 và trả về chuỗi str1. Chú ý là nội dung của str1 sẽ hoàn toàn được thay thế bởi str2 và str2 không thay đổi

7.8.5. Hàm strlen()

Hàm strlen() sẽ trả về chiều dài của chuỗi. Chiều dài của chuỗi rất hay được sử dụng trong các vòng lặp truy cập từng kí tự của chuỗi. Cú pháp của hàm là:

strlen(str);

trong đó str là mảng kí tự đã được khai báo và khởi tạo. Hàm trả về chiều dài của chuỗi str.

7.8.6. Sắp xếp chuỗi sử dụng các hàm trong thư viện

Các hàm về chuỗi được dùng để thao tác trên các mảng ký tự. Chẳng hạn như chiều dài của một chuỗi có thể được xác định bằng hàm strlen(). Chúng ta thử viết một chương trình C để sắp xếp 5 chuỗi theo độ dài giảm dần. Các bước thực hiện được liệt kê như sau:

Bước 1: Để sử dụng các hàm về chuỗi, trước tiên khai báo phải có hai thư viện chuẩn: stdio.h và string.h với câu lệnh là:

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

Bước 2: Khai báo một mảng ký tự để lưu 5 chuỗi. Câu lệnh sẽ là:

```
char str_arr[5][20];
```

Bước 3: Nhập vào 5 chỗi trong vòng lặp for. Câu lệnh sẽ là:

```
for(i=0; i<5; i++)
{
  printf("\n Nhap vao chuoi %d: ",i+1);
  scanf("%s",str_arr[i]);
}
```

Bước 4: So sánh chiều dài của mỗi chuỗi với các chuỗi khác, nếu chiều dài của chuỗi này nhỏ hơn chiều dài của một chuỗi đứng ở vị trí sau nó trong mảng, ta sẽ thực hiện đổi chỗ hai chuỗi đó cho nhau. Câu lệnh sẽ như sau:

```
for (i=0; i<4; i++)
  for (j=i+1; j<5; j++)
  {
      if(strlen(str_arr[i]) < strlen(str_arr[j]))
      {
      strcpy(str, str_arr[i]);
      strcpy(str_arr[i],str_arr[j]);
      strcpy(str_arr[j],str);
```

```
}
}
```

Bước 5: Hiển thị các chuỗi theo thứ tự đã sắp xếp. Câu lệnh là:

```
printf("\nThu tu sap xep cua mang la: ")
for( i=0; i<5; i++)
printf("\n%s",str_arr[i]);
```

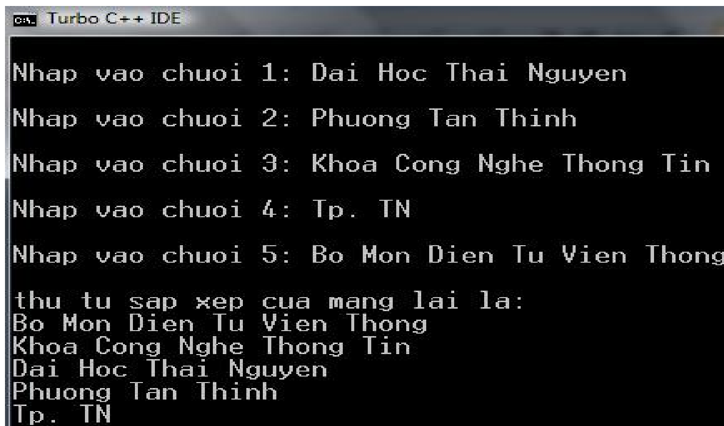
Chương trình 7.12

```
#include "stdio.h"
#include "conio.h"
#include "string.h"
void main()
{
//Khai bao bien
  int i,j;
  char str[50],str_arr[5][50];
  clrscr();//Lenh xoa man hinh
//Nhap vao cac chuoi can sap xep
  for(i=0;i<5;i++)
        {
              printf("\nNhap vao chuoi %d: ",i+1);
              gets(str_arr[i]);
        }
//Thuc hien so sanh va sap xep chuoi
  for(i=0;i<4;i++)
        for(j=i+1;j<5;j++)
        {
              if(strlen(str_arr[i])<strlen(str_arr[j]))
              {
                    strcpy(str,str_arr[i]);
                    strcpy(str_arr[i],str_arr[j]);
                    strcpy(str_arr[j],str);
              }
        }
//hien thi chuoi sap xep
printf("\nthu tu sap xep cua mang lai la: ");
for (i=0;i<5;i++)
printf("\n%s",str_arr[i]);
```

getch();

}

Kết quả chương trình 7.12


```
Nhap vao chuoi 1: Dai Hoc Thai Nguyen

Nhap vao chuoi 2: Phuong Tan Thinh

Nhap vao chuoi 3: Khoa Cong Nghe Thong Tin

Nhap vao chuoi 4: Tp. TN

Nhap vao chuoi 5: Bo Mon Dien Tu Vien Thong

thu tu sap xep cua mang lai la:
Bo Mon Dien Tu Vien Thong
Khoa Cong Nghe Thong Tin
Dai Hoc Thai Nguyen
Phuong Tan Thinh
Tp. TN
```

7.8.7. Sử dụng hàm để chuyển một mảng ký tự về chữ hoa

Các chuỗi có thể được truyền vào hàm để thao tác. Khi chuỗi hay mảng ký tự được truyền vào hàm, thực ra là truyền địa chỉ của nó. Sau đây ta xét một chương trình C để chuyển các chuỗi về dạng chữ hoa. Việc chuyển đổi về kiểu chữ in hoa sẽ được thực hiện bằng một hàm.

Các bước thực hiện như sau:

Bước 1: Đưa vào các thư viện cần thiết. Câu lệnh sẽ là:

#include “stdio.h”

#include “conio.h”

#include “string.h”

Bước 2: Khai báo một mảng để lưu 5 chuỗi. Câu lệnh sẽ là:

char names[5][20];

Bước 3: Khai báo một hàm nhận vào một chuỗi như là một đối số. Câu lệnh sẽ là:

void chuoihoa (char name_arr[]);

Bước 4: Nhập 5 chuỗi đưa vào mảng. Câu lệnh sẽ là:

for (i=1;i<5;i++)

{

printf(“\nNhap vao chuoi %d: ”,i+1);

gets(chuoi);

}

Bước 5: Truyền mỗi chuỗi vào hàm để chuyển thành in hoa. Sau khi chuyển đổi, hiển thị chuỗi đã thay đổi. Câu lệnh sẽ là:

for(i = 0; i < 5; i++)

{

chuoihoa(chuoi[i]);

```
        printf("\nChuoi moi %d: %s", i + 1, chuoi[i]);
}
```

Bước 6: Định nghĩa hàm, câu lệnh sẽ là:

```
void chuoihoa(char name_arr[])
{
int x;
for(x = 0; name_arr[x] != '\0'; x++)
{
if(name_arr[x] >= 97 && name_arr[x] <= 122)
        name_arr[x] = name_arr[x] - 32;
}
  }
```

Chương trình 7.13

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void  main()
{
int i;
char arr[5][20];
void chuoihoa(char name_arr[]);
  clrscr();
  for(i = 0; i < 5; i++)
  {
  printf("\nNhap chuoi %d: ", i + 1);
        gets(arr[i]);
  }
  for(i = 0; i < 5; i++)
  {
        chuoihoa(arr[i]);
        printf("\nChuoi moi %d: %s", i + 1, arr[i]);
  }
  getch();
}
void chuoihoa(char name_arr[])
{
  int x;
```

```
  for(x = 0; name_arr[x] != '\0'; x++)
  {     if(name_arr[x] >= 97 && name_arr[x] <= 122)
              name_arr[x] = name_arr[x] - 32;
  }
}
```

Kết quả chương trình 7.13

```
TC.EXE

Nhap chuoi 1: khoa cong nghe thong  tin

Nhap chuoi 2: Dai hoc thai nguyen

Nhap chuoi 3: tan thinh

Nhap chuoi 4: thinh dan

Nhap chuoi 5: Tp TN

Chuoi moi 1: KHOA CONG NGHE THONGDAI HOC THAI NGUYEN
Chuoi moi 2: DAI HOC THAI NGUYEN
Chuoi moi 3: TAN THINH
Chuoi moi 4: THINH DAN
Chuoi moi 5: TP TN
```

7.9. Bài tập

Chương 8 CẤU TRÚC VÀ DANH SÁCH LIÊN KẾT

8.1. Giới thiệu chung

Từ các chương trước, chúng ta đã tìm hiểu về biến và mảng là hai đơn vị để lưu trữ và xử lý dữ liệu. Tuy nhiên, mỗi biến chỉ có thể chứa một giá trị có kiểu dữ liệu xác định, còn mảng được coi là một tập hợp các biến có cùng kiểu dữ liệu và được biểu thị bằng một tên chung. Trong khi việc lưu trữ và xử lý thông tin trong nhiều ứng dụng thực tế không chỉ đơn thuần là các thao tác trên một giá trị hay một tập giá trị cùng kiểu, mà đòi hỏi có sự tổng hợp của nhiều kiểu dữ liệu trong cùng một đối tượng xử lý. Thí dụ, khi quản lý với đối tượng là *sinh viên,* chúng ta cần quản lý về *họ tên, ngày sinh, lớp, mã số sinh viên, điểm học tập.*

Do đó, trong C cho phép chúng ta tạo ra các kiểu dữ liệu tùy chọn. Một trong số các kiểu dữ liệu tự tạo, cấu trúc được xem như sự mở rộng của biến và mảng, đáp ứng được các yêu cầu xử lý phức tạp hơn vì nó là một tập các biến, các mảng được biểu diễn bởi một tên chung. Trong chương này, chúng ta sẽ đi sâu vào định nghĩa, cách khai báo và sử dụng kiểu cấu trúc cũng như một số kiểu dữ liệu tự tạo khác.

8.2. Kiểu cấu trúc

8.2.1. Khái niệm cấu trúc

Cấu trúc (**struct**) là một kiểu dữ liệu do người dùng tự định nghĩa bao gồm nhiều thành phần, mỗi thành phần có thể là một biến, mảng, hay kiểu con trỏ đã được định nghĩa hoặc một cấu trúc nào đó. Mỗi thành phần được gọi là một trường (field).

8.2.2. Định nghĩa kiểu cấu trúc

Một định nghĩa kiểu cấu trúc hình thành một khuôn mẫu để tạo ra các biến cấu trúc. Các biến trong cấu trúc được gọi là các phần tử hay thành phần của cấu trúc. Việc định nghĩa một kiểu cấu trúc cần phải chỉ ra: tên kiểu cấu trúc và các thành phần của nó theo một trong hai cách sau:

Cú pháp 1:

```
struct  tên_cấu _trúc
              {
                Kiểu_dữ_liệu trường_1;
                Kiểu_dữ_liệu trường_2;
                ….
                Kiểu_dữ_liệu trường_n;
              };
```

Cú pháp 2: Sử dụng từ khóa **typedef** để định nghĩa:

```
            typedef struct
              {
                Kiểu_dữ_liệu trường_1;
                Kiểu_dữ_liệu trường_2;
                ….
                Kiểu_dữ_liệu trường_n;
              }tên_cấu_trúc;
```

Trong đó:

- **struct:** là từ khóa dùng để định nghĩa kiểu cấu trúc. Ngoài ra có thể sử dụng từ khóa **typedef** để định nghĩa kiểu cấu trúc như trong cú pháp 2.
- Tên_cấu_trúc**:** là một tên bất kỳ tuân theo quy tắc đặt tên trong C.
- **Kiểu_dữ_liệu** trường i (i = 1…n):Khai báo tên các trường trong cấu trúc có kiểu được xác định là **kiểu_dữ_liệu.**

Thí dụ: khai báo một cấu trúc có tên là **dien_tro**

Cách 1: **struct** dien_tro

```
      {
        char ten_DT[5];
        float giatri;
      };
```

Cách 2: **typedef struct**

```
      {
        char ten_DT[5];
        float giatri;
```

}dien_tro;

8.2.3. Khai báo theo kiểu cấu trúc đã định nghĩa

Sau khi đã định nghĩa một cấu trúc, ta có thể dùng nó để khai báo các biến có kiểu cấu trúc tương tự như khai báo biến thuộc kiểu dữ liệu chuẩn. Một biến, mảng hay con trỏ cấu trúc sau khai báo sẽ có đầy đủ tính chất như các biến mảng hay con trỏ thông thường. Có thể khai báo biến cấu trúc theo hai cách: khai báo trực tiếp khi định nghĩa cấu trúc và khai báo sau khi đã định nghĩa.

Để khai báo trực tiếp, chỉ cần đặt danh sách tên các biến vào sau dấu }của cú pháp 1, tức là cú pháp khai báo trực tiếp như sau:

struct tên_cấu_trúc

{

Kiểu trường_1;

Kiểu trường_2;

….

Kiểu trường_n;

}*danh sách các biến cấu trúc;*

Trong đó, các biến cấu trúc được phân cách nhau bởi dấu phẩy và kết thúc của khai báo biến là dấu chấm phẩy (;).

Nếu khai báo sau khi định nghĩa, thì tùy vào cách định nghĩa ta sẽ có cách khai báo riêng. Tức là:

- Đối với cấu trúc được định nghĩa theo cú pháp 1, ta có cú pháp khai báo biến cấu trúc như sau:

struct Tên_cấu_trúc *danh_sách_biến*;

- Đối với các cấu trúc được định nghĩa theo cú pháp 2, ta khai báo biến cấu trúc như sau:

Tên_cấu_trúc *danh_sách_biến*;

Trong đó, tên biến cấu trúc được đặt theo quy tắc danh định trong C, nếu khai báo nhiều biến thì các biến phân cách nhau bởi dấu phẩy. Lưu ý là, nếu định nghĩa cấu trúc theo cú pháp 2 thì khi khai báo biến không cần từ khóa **struct.**

Thí dụ: ta khai báo biến cho cấu trúc có tên dien_tro đã định nghĩa ở trên:

Cách 1: **struct** dien_tro R;

Cách 2: dien_tro R, *ptr;

+) Khởi tạo cấu trúc khi khai báo:

Việc khởi tạo cấu trúc có thể được thực hiện trong lúc khai báo biến cấu trúc. Các trường của cấu trúc được khởi tạo được đạt giữa 2 dấu { và }, chúng được phân cách nhau bởi dấu phẩy (,).

Ví dụ: Khởi tạo giá trị cho biến cấu trúc R:

struct dien_tro R ={“R1”, 50};

8.2.4. Truy xuất đến các thành phần của cấu trúc

Cú pháp:

Biến_cấu_trúc.Tên_trường

Khi sử dụng cách truy xuất theo kiểu này, các thao tác trên các trường của biến cấu trúc giống như các thao tác trên các biến của kiểu dữ liệu của trường.

Thí dụ:

Bước 1: Với định nghĩa cấu trúc có tên **dien_tro** và khai báo biến R, ta có thể truy xuất đến các thành phần **ten_DT** và **giatri** của cấu trúc như sau:

R.ten_DT

R.giatri

Bước 2: Viết chương trình cho phép đọc dữ liệu từ bàn phím cho biến mẩu tin linh kiện và in biến mẩu tin đó lên màn hình:

Chương trình 8.1

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
typedef struct
{
unsigned char Ngay;
unsigned char Thang;
unsigned int Nam;
} NgayThang;
typedef struct
{
    char ma_lk[10];
    char ten_lk[40];
        NgayThang NgaySX; //cấu trúc lồng nhau
        char DiaChi_NSX[40];
}LinhKien;
/* Hàm in lên màn hình 1 mẩu tin linh kiện*/
void InLK(LinhKien s)
{
 printf("Ma linh kien|Ten linh kien|Ngay san xuat|Dia chi NSX\n");
 printf("%s         |%s         |%d-%d-%d         |%s\n",s.ma_lk,s.ten_lk,
s.NgaySX.Ngay,s.NgaySX.Thang,s.NgaySX.Nam,s.DiaChi_NSX);
}
```

```
void main()
{
        SinhVien LK;
    printf("Nhap ma linh kien: ");gets(LK.ma_lk);
    printf("Nhap ten linh kien: ");gets(LK.ten_lk);
    printf("Ngay san xuat: ");
    printf("\nNgay:");scanf("%d",&LK.NgaySX.Ngay);
    printf("Thang: ");scanf("%d",&LK.NgaySX.Thang);
        printf("Nam: ");scanf("%d",&LK.NgaySX.Nam);
    fflush(stdin);
    printf("Dia chi NSX: ");gets(LK.DiaChi_NSX);
    printf("Thong tin ve linh kien:");
    InLK(LK);
    getch();
}
```

Sau khi chạy chương trình trên màn hình sẽ hiển thị kết quả như sau:

```
Nhap ma linh kien: R01
Nhap ten linh kien: dien tro
Ngay san xuat:
Ngay: 20
Thang:05
Nam: 2007
Dia chi NSX: Ha Noi
Thong tin ve linh kien:
Ma linh kien|Ten linh kien| Ngay san xuat | Dia chi NSX
  R01          dien tro        20-05-2007       Ha Noi
```

8.2.5. Cấu trúc lồng nhau

8.2.5.1. Định nghĩa cấu trúc lồng nhau

Khi một cấu trúc là một thành phần của một cấu trúc khác được gọi là cấu trúc lồng nhau.

Thí dụ, ta có thể định nghĩa một cấu trúc lồng nhau về việc quản lý mượn sách ở thư viện như sau:

```
struct ngaythang
{  int ngay;
   int thang;
   int nam;
};
```

```
struct  quanlysach
{   char tenSV[35];
      char lop[10];
          char tensach[50];
          char masach[10];
      struct ngaythang  ngaymuon;   //cấu trúc lồng nhau
};
```

Với định nghĩa trên thì cấu trúc "ngaythang" cũng là một trường trong cấu trúc "quanlysach".

8.2.5.2. Truy xuất đến các thành phần của cấu trúc lồng nhau

Việc truy xuất đến các thành phần của một cấu trúc lồng nhau được thực hiện tuần tự từ ngoài vào trong.

Thí dụ, việc truy xuất đến thành phần "ngay" của cấu trúc "quanlysach", với biến cấu trúc là "sach1" như sau:

sach1.ngaymuon.ngay

8.3. Mảng cấu trúc

8.3.1. Khai báo biến mảng cấu trúc

Tương tự như khai báo một biến mảng thông thường sử dụng các kiểu dữ liệu chuẩn, chúng ta hoàn toàn có thể khai báo một biến mảng cấu trúc sử dụng kiểu cấu trúc đã định nghĩa.

Cú pháp khai báo mảng cấu trúc như sau:

struct tên_cấu_trúc   tên_mảng[kích thước];

Thí dụ: ta khai báo mảng cấu trúc có tên R của cấu trúc dien_tro như sau:

```
struct   dien_tro  R[50];  // khai báo mảng R gồm 50 phần tử.
```

8.3.2. Truy xuất các phần tử mảng cấu trúc

Việc truy xuất tới từng phần tử của mảng cấu trúc cũng giống như truy xuất tới các phần tử mảng thông thường.

Cú pháp truy xuất đến thành phần thứ i của mảngnhư sau:

tên_mảng[i] **.**trường

8.3.3. Khởi tạo giá trị cho các phần tử của mảng cấu trúc

8.4. Con trỏ cấu trúc

Một biến cấu trúc cũng là một biến trong bộ nhớ nên nó có địa chỉ bộ nhớ. Bạn có thể lấy địa chỉ của biến bằng phép toán lấy địa chỉ với toán tử **&.** Khi đó, bạn sẽ nhận được địa chỉ là vị trí bắt đầu của một cấu trúc

8.4.1. Khai báo con trỏ cấu trúc

Việc khai báo một biến con trỏ kiểu cấu trúc cũng tương tự như khi khai báo một biến con trỏ khác, nghĩa là đặt thêm dấu * vào phía trước tên biến.

Cú pháp**:** **struct** Tên_cấu_trúc * Biến_con_trỏ;

Thí dụ: Ta có thể khai báo một con trỏ cấu trúc kiểu NgayThang như sau:

struct NgayThang *p;

/* hoặc là: NgayThang *p; Nếu có định nghĩa theo cách 2 (dùng từ khóa **typedef**)*/

8.4.2. Sử dụng con trỏ cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa có địa chỉ cụ thể. Lúc này nó chỉ mới được cấp phát 2 byte để lưu giữ địa chỉ và được ghi nhận là con trỏ chỉ đến 1 cấu trúc, nhưng chưa chỉ đến 1 đối tượng cụ thể. Muốn thao tác trên con trỏ cấu trúc hợp lệ, cũng tương tự như các con trỏ khác người lập trình phải:

- Cấp phát một vùng nhớ cho nó (sử dụng hàm malloc() hay calloc)
- Hoặc cho nó quản lý địa chỉ của một biến cấu trúc nào đó.

Thí dụ, sau khi khai báo:

struct NgayThang NgaySX, *p;

p=&NgaySX; //Cho con trỏ p quản lý địa chỉ của biến NgaySX.

8.4.2.1. Truy xuất các thành phần cấu trúc thông qua con trỏ

Để truy cập đến nội dung của từng trường của 1 cấu trúc thông qua con trỏ của nó, người lập trình phải sử dụng toán tử dấu mũi tên (->: dấu - và dấu >). Thực ra, phép toán mũi tên là dạng viết gọn của sự kết hợp của phép toán lấy nội dung (*) và sau đó lấy thành phần bằng toán tử dấu chấm (**.**) của đối tượng. Có nghĩa là ta có thể viết:

(*p).tên_trường

Hoặc: **p->**tên_trường

Thí dụ: Để hiểu hơn về cách sử dụng con trỏ cấu trúc chúng ta cùng xét chương trình hiển thị lên màn hình ngày tháng năm dưới đây:

Chương trình 8.2

```
#include<conio.h>
#include<stdio.h>
 typedef struct
{
  unsigned char Ngay;
  unsigned char Thang;
  unsigned int Nam;
} NgayThang;
void main()
{
  NgayThang Ng={17,11,2009};
  NgayThang *p;
```

```
  clrscr();
  p=&Ng;
  printf("Truy cap thong thuong %d-%d-%d\n",
Ng.Ngay,Ng.Thang,Ng.Nam);
  printf("Truy cap qua con tro %d-%d-%d\n",
p->Ngay,p->Thang,p->Nam);
  printf("Truy cap qua vung nho con tro %d-%d-%d\n",
(*p).Ngay,(*p).Thang,(*p).Nam);
  getch();
}
```

Kết quả hiển thị lên màn hình sau khi thực hiện chương trình là:

```
Truy cap thong thuong 17- 11- 2009
Truy cap qua con tro 17- 11- 2009
Truy cap qua vung nho con tro 17- 11-2009
```

8.4.2.2. Phép toán số học với con trỏ cấu trúc

Đối với con trỏ cấu trúc, ta có thể thực hiện phép toán số học như cộng, trừ địa chỉ. Sau đó, con trỏ sẽ trỏ tới các thành phần bất kỳ khác.

Thí dụ, con trỏ ptr đang trỏ tới đầu mảng Dien_tro[], phép cộng sau sẽ làm con trỏ ptr trỏ đến thành phần khác của mảng cấu trúc:

ptr = ptr+5 ; //con trỏ ptr trỏ tới địa chỉ của Dien_tro[5].

ptr = ptr -3;// con trỏ ptr sẽ trỏ tới địa chỉ của Dien_tro[2].

Lưu ý:

- Các phép toán để truy xuất đến thành phần của cấu trúc (“.” và “->”) là có độ ưu tiên cao nhất. Do đó cần phải lưu ý đến các biểu thức hỗn hợp có chứa các phép toán này và phép toán số học. Thí dụ:

Biểu thức: ++ptr->dien_tro[0] ⇔ ++(ptr->dien_tro[0]). Có nghĩa là, trong biểu thức trên không phải tăng giá trị của con trỏ ptr mà là tăng giá trị của thành phần dien_tro[0] được xác định bởi ptr.

- Phép lấy địa chỉ chỉ thực hiện tốt với các thành phần là kiểu nguyên, còn với các kiểu không nguyên thì có thể làm treo máy. Vì vậy, trước tiên ta nên thao tác với một biến trung gian, sau đó mới gán giá trị cho thành phần của cấu trúc.

8.4.3. Con trỏ và mảng cấu trúc

Tương tự như quan hệ giữa con trỏ và mảng thông thường, trong kiểu cấu trúc nếu con trỏ ptr trỏ tới địa chỉ bắt đầu của mảng cấu trúc Dien_tro[] thì các cách viết sau có ý nghĩa tương đương nhau:

Dien_tro[i] , ptr[i] và *(ptr+i)

Khi đó ta có thể truy cập tới các thành phần theo các cách sau:

Dien_tro[i].tên_trường

ptr[i].tên_trường

(ptr+i)-> tên_trường

Thí dụ: để truy cập tới thành phần gia_tri của cấu trúc có biến mảng là Dien_tro, ta có thể dùng các cách sau:

Dien_tro[i].gia_tri

ptr[i].gia_tri

(ptr+i)->gia_tri

8.5. Danh sách liên kết

8.5.1. Khái niệm cấu trúc tự trỏ

Cấu trúc tự trỏ là dạng cấu trúc đặc biệt có chứa một thành phần là con trỏ trỏ đến chính nó. Một cấu trúc tự trỏ có thể được định nghĩa theo cú pháp sau:

Cú pháp:

```
struct  tên_cấu_trúc
{
     Các thành phần của cấu trúc;
     struct  tên_cấu_trúc *tên_con_trỏ;
}danh_sách_các_biến;
```

Khi đó, con trỏ ptr được dùng để trỏ đến các biến cấu trúc khác có cùng kiểu.

Thí dụ, ta có định nghĩa của một cấu trúc tự trỏ về linh kiện điện tử như sau:

```
struct pointer
{
     char ten_linhkien[20];
     int ma;
     float gia;
     struct pointer *next;
}linhkien;
```

8.5.2. Khái niệm danh sách liên kết

Định nghĩa: Danh sách liên kết là loại danh sách mà các phần tử nó được lưu trữ khắp nơi trong bộ nhớ và được liên kết với nhau theo một trật tự nhất định nhờ vào các vùng dữ liệu đặc biệt gọi là vùng liên kết.

Mỗi phần tử của danh sách liên kết là một biến kiểu cấu trúc tự trỏ hoặc là kiểu cấu trúc có một thành phần dữ liệu là một con trỏ trỏ đến một cấu trúc khác. Có hai loại liên kết là liên kết theo chiều thuận và liên kết theo chiều ngược.

Danh sách liên kết theo chiều thuận:

- Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó.

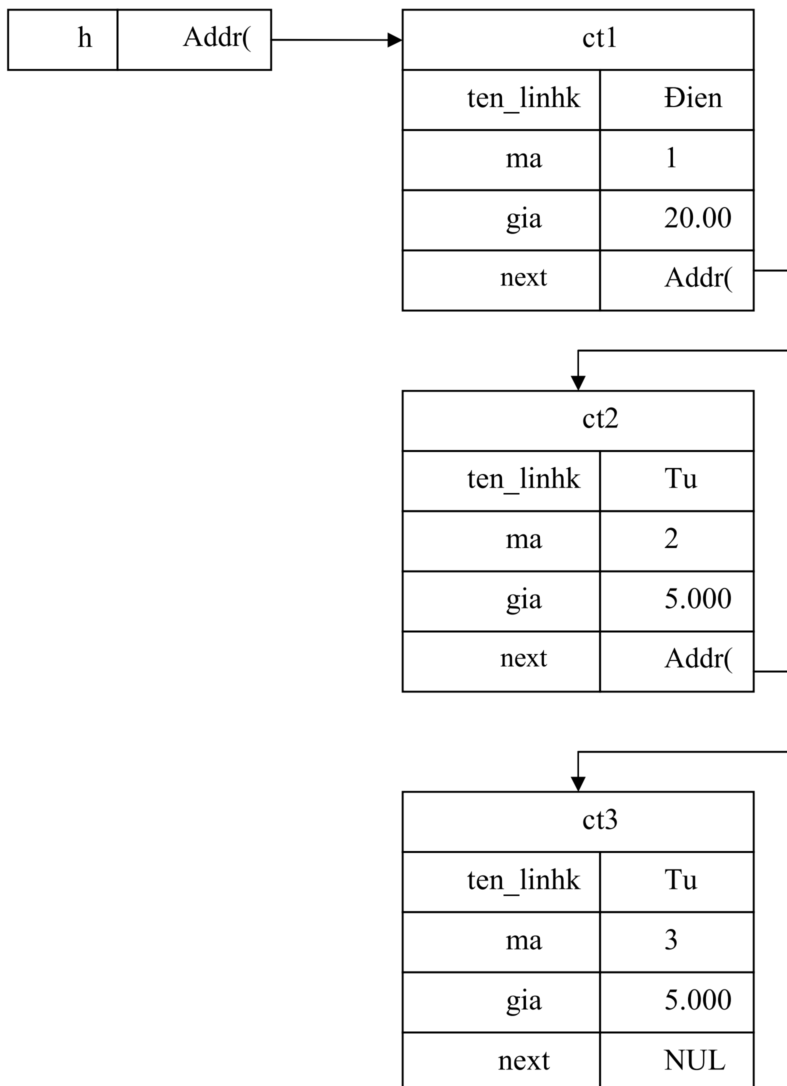
- Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp theo trong danh sách.
- Cấu trúc cuối chứa hằng Null.

Danh sách liên kết theo chiều ngược:

- Biết cấu trúc cuối.
- Trong mỗi cấu trúc, chứa địa chỉ của cấu trúc đứng trước.
- Cấu trúc đầu chứa hằng NULL.

Danh sách liên kết theo chiều thuận cho phép truy xuất từ cấu trúc đầu đến cấu trúc cuối và ngược lại, nếu liên kết theo chiều ngược sẽ cho phép truy xuất từ cấu trúc cuối đến cấu trúc đầu.

Có thể mô tả một danh sách liên kết theo sơ đồ thí dụ sau, trong đó



8.5.3. Các phép toán trên danh sách liên kết

8.5.3.1. Tạo danh sách

Để tạo danh sách liên kết mới cần thực hiện các khâu sau:

- Cấu phát bộ nhớ cho một cấu trúc
- Nhập các trường thông tin vào vùng nhớ vừa cấp
- Gán địa chỉ cấu sau cho thành phần con trỏ của cấu trúc đứng trước.

8.5.4.2. Duyệt danh sách

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

- Đầu tiên p = head
- Để chuyển đến cấu trúc tiếp theo dùng phép toán gán

p = p->next

- Dấu hiệu để biết đang xét đến cấu trúc cuối cùng là p -> next =NULL

8.5.4.3. Xóa một cấu trúc khỏi danh sách

Để xóa một phần tử của danh sách, bạn cần thực hiện các bước sau:

- Lưu trữ địa chỉ của cấu trúc cần loại vào một con trỏ (dùng để giải phóng bộ nhớ của phần tử đó)
- Gán địa chỉ bộ nhớ của cấu trúc cần loại cho cấu trúc đứng trước nó.
- Giải phóng bộ nhớ của cấu trúc cần loại.

8.5.4.5. Bổ sung một phần tử vào danh sách

Để bổ sung một phần tử vào danh sách ta cần thực hiện:

- Cấp phát bộ nhớ và nhập bổ sung phần tử
- Sửa thành phần con trỏ trong các cấu trúc có liên quan để đảm bảo mỗi phần tử chứa địa chỉ của phần tử tiếp theo.

Thí dụ: Chương trình 8.3 /*Nhập vào danh sách các điện trở, mỗi điện trở gồm các thông tin: Tên điện trở, ngày sản xuất, giá trị. Thực hiện tìm kiếm các điện trở theo tên với 2 cách khác nhau truy nhập thông thường và truy nhập theo con trỏ.*/

```
#include"stdio.h"
#include"conio.h"
#include"string.h"
 struct date
    {
 int ngay,thang,nam;
    };
 typedef struct
    {
  char ten[25];
  struct date nsx;
  float giatri;
    } DienTro;
```

```
DienTro *ptim(char *ten,DienTro ds[],int n);
DienTro tim(char *ten,DienTro ds[],int n);
void DoiCho(DienTro *p1,DienTro *p2);
void SapXep(DienTro *ps,int n);
void Nhap(DienTro *p);
void in(DienTro p);
DienTro tim(char *ten,DienTro ds[],int n)
   {
 int i;DienTro ps;
 ps.nsx.ngay=ps.nsx.thang=ps.nsx.nam=0;  ps.giatri=0.0;
 ps.ten[0]=0;
 for(i=1;i<=n;i++)
   if (strcmp(ten,ds[i].ten)==0) return(ds[i]);
 return(ps);
   }
DienTro *ptim(char *ten,DienTro ds[],int n)
   {
 int i;
 for(i=1;i<=n;i++)
    if(strcmp(ten,ds[i].ten)==0) return(&ds[i]);
 return(NULL);
   }
void DoiCho(DienTro *p1,DienTro *p2)
   {
DienTro t;
  t=*p1;
  *p1=*p2;
  *p2=t;
    }
void Nhap(DienTro *p)
   {
   DienTro q; float gt;
   printf("\nNhap vao ten dien tro :");gets(q.ten);
   printf("\nNhap ngay ngay san xuat :");
   scanf("%d%d%d",&q.nsx.ngay,&q.nsx.thang,&q.nsx.nam);
   printf("\Nhap gia tri dien tro :");
   scanf("%f%*c",&gt);q.giatri=gt;
```

```
     *p=q;
     }
   void in(DienTro p)
     {
     printf("\n\n%s%12d-%d-%d%12.1f",p.ten,p.nsx.ngay,
p.nsx.thang,p.nsx.nam,p.giatri);
     }
   void SapXep(DienTro *p,int n)
     {
       int i,j;
       for(i=1;i<=n;++i)
         for(j=i+1;j<=n;++j)
           if(p[i].nsx.nam>p[j].nsx.nam)
        DoiCho(&p[i],&p[j]);
     }
  void  main()
    {
     DienTro *p,ds[100];int n,i,j;char ten[40];
     /*vao so lieu */
     clrscr();
     printf("\n So dien tro n=");scanf("%d%*c",&n);
     for(i=1;i<=n;i++)
     Nhap(&ds[i]);
   //sap xep theo chieu tang cua ngay san xuat
     SapXep(ds,n);    //in danh sach sau khi sap xep
     printf("\n\nTen dien tro|Ngay san xuat|Gia tri");
     for(i=1;i<=n;++i)
    in(ds[i]);     //tim kiem theo ho ten
     while(1)
    {
 printf("\nNhap ten dien tro can tim\(Bam Enter de ket thuc)");
     gets(ten);
     if(ten[0]==0) break;
     if((p=ptim(ten,ds,n))==NULL)
     printf("\n Khong tim thay");
     else in(*p);
```

```
}
 //tim kiem theo ho ten dung ham tim
  while(1)
{
 printf("\nNhap ten dien tro can tim\(bam Enter de ket thuc)");
 gets(ten);
 if(ten[0]==0) break;
 if(tim(ten,ds,n).ten[0]==0)
   printf("\nKhong tim thay");
 else in(tim(ten,ds,n));
}                  //ket thuc
 }
```

8.6. Một số kiểu dữ liệu tự tạo khác (kiểu liệt kê (enum) và kiểu hợp (union))

8.6.1. Kiểu liệt kê (enum)

8.6.1.1. Khái niệm

Enum là kiểu dữ liệu mà một biến của nó có thể nhận được một giá trị nào đó trong các giá trị được liệt kê.

8.6.1.2. Định nghĩa kiểu enum

Cú pháp định nghĩa kiểu enum như sau:

enum tên

{

  bảng liệt kê

}*;*

Thí dụ, định nghĩa các kiểu liệt kê các ngày trong tuần như sau:

enum ngay {thu2, thu3, thu4, thu5, thu6, thu7, chu_nhat};

8.6.1.3. Khai báo biến

Để khai báo biến của kiểu **enum** đã định nghĩa, ta sử dụng cú pháp:

enum tên *tên_biến;*

Thí dụ, khai báo biến sau:

enum ngay thu;

Lưu ý:

- Kiểu **enum** là trường hợp đặc biệt của kiểu **int** vì trong tính toán thì mỗi biến **enum** đều được đổi thành kiểu **int.**
- Các giá trị liệt kê trong kiểu **enum** là một hằng số chứ không phải chuỗi ký tự.

8.6.2. Kiểu hợp (union)

8.6.2.1. Khái niệm

Kiểu hợp là một cấu trúc đặc biệt được định nghĩa bằng từ khóa **union** có các thành phần chỉ dùng chung một vùng nhớ và kích thước của hợp sẽ là kích thước của thành phần lớn nhất. Nghĩa là, với kiểu hợp ta có thể khai báo ra các biến có khả năng chứa được nhiều kiểu dữ liệu khác nhau. Hơn nữa, ở mỗi thời điểm, toàn bộ union chỉ được sử dụng như một trong các thành phần của nó mà thôi.

8.6.2.2. Định nghĩa kiểu union

Cú pháp để định nghĩa một **union** cũng tương tự như định nghĩa một cấu trúc như sau:

```
union   tên_union
{
        Kiểu trường_1;
        Kiểu trường_2;
         ….
        Kiểu trường_n;
};
```

8.6.2.3. Khai báo biến

Khai báo biến kiểu hợp cũng tương tự như khai báo biến **struct** theo cấu trúc như sau:

union tên_union tên_biến;

Thí dụ, ta có định nghĩa union sau:

```
union   dientro
{
char  tendt[5];
        float  giatri;
};
union   dientro   R;// khai báo biến R.
```

8.6.2.4. Truy xuất kiểu union

Khi đó, để truy xuất đến các thành phần của **union**, ta cũng thực hiện giống như kiểu cấu trúc:

Tên_biến_union.tên_trường

Thí dụ: trong thí dụ về union dientro, ta truy cập vào các trường theo cú pháp:

R.tendt và R.giatri

Bài tập chương

Bài 1 Viết chương trình nhập vào một danh sách các điện trở, mỗi điện trở gồm các thông tin: Ten (tên), GiaTri (giá trị) sau đó in thông tin của mảng .

Bài 2 Viết chương trình nhập vào một mảng các linh kiện, mỗi linh kiên là một cấu trúc LinhKien gồm các thông tin: Ten(Tên điện trở), loaiLK, NamSX (năm sản xuất) rồi in ra màn hình thông tin về các linh kiện thuộc loại "sensor".

Bài 3 Viết chương trình nhập vào một mảng các linh kiện, mỗi linh kiên là một cấu trúc LinhKien gồm các thông tin: Ten(Tên điện trở), loaiLK, giaLK, NamSX (năm sản xuất) sắp xếp các linh kiện trong danh sách theo vần a,b,c, của tên rồi in danh sách linh kiện ra màn hình.

Bài 4 Viết chương trình nhập vào một danh sách liên kết theo chiều thuận các điện trở có cấu trúc như dưới đây. Sau đó thực hiện chức năng thêm điện trở và hiển thị danh sách điện trở đã nhập

```
Struct Dien_Tro
{
  char TenDT[5];
  float Gia_tri;
};
```

Chương 9 TỆP TIN

9.1. Giới thiệu chung

Đối với các kiểu dữ liệu đã biết như kiểu số, kiểu mảng, kiểu cấu trúc thì dữ liệu được tổ chức trong bộ nhớ trong (RAM) của máy tính nên khi kết thúc việc thực hiện chương trình thì dữ liệu cũng bị mất; khi cần người lập trình bắt buộc phải nhập lại từ bàn phím. Điều đó vừa mất thời gian vừa không giải quyết được các bài toán với số liệu lớn. Để giải quyết vấn đề, người ta đưa ra kiểu tập tin (file) cho phép lưu trữ dữ liệu ở bộ nhớ ngoài (đĩa). Khi kết thúc chương trình thì dữ liệu vẫn còn, do đó có thể sử dụng nhiều lần. Một đặc điểm khác của kiểu tập tin là kích thước lớn với số lượng các phần tử không hạn chế (chỉ bị hạn chế bởi dung lượng của bộ nhớ ngoài).

Trong chương này chúng ta sẽ tìm hiểu về một số loại tệp tin và các thao tác xử lý trên tệp tin trong môi trường Turbo C.

9.2. Một số khái niệm về tệp tin

Trong chương này chúng ta sẽ xét đến hai loại tệp tin cơ bản liên quan đến tính chất đã trình bày ở trên, đó là tệp tin văn bản và tệp tin nhị phân.

Tệp tin văn bản (Text File): là loại tập tin dùng để ghi các ký tự lên đĩa, các ký tự này được lưu trữ dưới dạng mã ASCII. Điểm đặc biệt là dữ liệu của tập tin được lưu trữ thành các dòng, mỗi dòng được kết thúc bằng ký tự xuống dòng (new line), ký hiệu '\n'; ký tự này là sự kết hợp của 2 ký tự CR (Carriage Return - Về đầu dòng, mã ASCII là 13) và LF (Line Feed - Xuống dòng, mã ASCII là 10). Mỗi tập tin được kết

thúc bởi ký tự EOF (End Of File) có mã ASCII là 26. Tập tin văn bản chỉ có thể truy xuất theo kiểu tuần tự.

Tệp tin nhị phân (Binary File): là tệp tin dùng để ghi lên đĩa các cấu trúc dưới dạng nhị phân (dạng mã máy), do đó bạn không thể đọc được nội dung của nó. Khi làm việc với các con số thì ta nên làm việc ở chế độ nhị phân vì nếu ở chế độ văn bản, trong tệp có thể đã tồn tại sẵn mã 26 và mã 10 nên có thể gây ra sai lệch dữ liệu.

Biến tập tin: là một biến thuộc kiểu dữ liệu tập tin dùng để đại diện cho một tập tin. Dữ liệu chứa trong một tập tin được truy xuất qua các thao tác với thông số là biến tập tin đại diện cho tập tin đó.

Con trỏ tập tin: Khi một tập tin được mở ra để làm việc, tại mỗi thời điểm, sẽ có một vị trí của tập tin mà tại đó việc đọc/ghi thông tin sẽ xảy ra. Người ta hình dung có một con trỏ đang chỉ đến vị trí đó và đặt tên nó là con trỏ tập tin.

Sau khi đọc/ghi xong dữ liệu, con trỏ sẽ chuyển dịch thêm một phần tử về phía cuối tập tin. Sau phần tử dữ liệu cuối cùng của tập tin là dấu kết thúc tập tin EOF (End f File).

9.3. Thao tác trên tệp tin văn bản

Để làm việc với tập tin văn bản, bạn cần phải thực hiện lần lượt các bước sau:

- Khai báo tệp tin
- Mở tệp tin
- Truy xuất tệp tin: ghi hoặc đọc nội dung dữ liệu từ một tệp tin đã mở, hay các thao tác khác như đánh dấu tập tin, xóa tập tin…
- Đóng tập tin.

Dưới đây là các hàm trong Turbo C để thực hiện các thao tác trên.

9.3.1. Hàm đóng/mở file

9.3.1.1. Khai báo tệp tin

Khi làm việc với tệp tin thì trước hết bạn cần khai báo biến tệp tin bằng từ khóa **FILE** được định nghĩa trong tệp tiêu đề "stdio.h". Cú pháp khai báo biến tệp tin như sau:

FILE ***danh sánh biến con trỏ***;

Nếu có nhiều biến, thì các biến được phân cách nhau bởi dấu phẩy.

Ví dụ: Một số khai báo tệp tin:

FILE doc;

FILE *p, *t;

9.3.1.2. Hàm fopen( )

+) Dạng hàm:

fopen("đường dẫn", "kiểu tùy chọn");

+) Ý nghĩa:

- Mở một tệp tin theo tên đường dẫn. Nếu trong "đường dẫn" chỉ có tên tệp tin thì sẽ truy xuất trong thư mục hiện hành. Tên tệp tin phải khai báo theo quy định của hệ điều hành (Phần tên (tối đa là 8 ký tự)**.**phần mở rộng (gồm 3 ký tự liền nhau)).Và tên tệp tin phải đặt trong cặp dấu " ". Ví dụ: một tệp tin có tên là: "vidu.txt".
- Kiểu tùy chọn: là tổ hợp các ký tự mô tả các tùy chọn sau:

Chế độ	**Ý nghĩa**
r	Mở một tập tin văn bản chỉ để đọc vào bộ nhớ
w	Mở một tập tin văn bản chỉ để ghi vào đĩa
a	Nối vào một tập tin văn bản
t	Mở một tệp tin kiểu văn bản (text)
r+	Mở một tập tin văn bản để đọc/ghi
w+	Mở một tập tin văn bản để đọc/ghi
a+	Nối hoặc tạo một tập tin văn bản để đọc/ghi

+) Thí dụ: Câu lệnh:

FILE doc;

doc = **fopen**("baitap.txt", "r");

Đầu tiên ta khai báo biến tệp tin "doc" với từ khóa **FILE.**

Sau đó, mở file có tên là "baitap.txt" để đọc (tùy chọn "r") và gán cho biến tệp tin "doc".

9.3.1.3. Hàm fclose( )

Sau khi không còn làm việc với tệp tin, bạn nhất thiết phải đóng tệp tin để đảm bảo an toàn dữ liệu.

+) Dạng hàm:

fclose(biến tệp tin);

fcloseall(); //đóng tất cả các tập tin.

+) Ý nghĩa:

- Đóng tệp tin đã được mở bằng lệnh *fopen( )* và gán vào biến tệp tin.
- Hàm *fclose( )* trả về 0 nếu đóng thành công. Bất kỳ giá trị trả về nào khác 0 đều cho thấy có lỗi xảy ra. Hàm *fclose( )* sẽ thất bại nếu đĩa đã sớm được gỡ ra khỏi ổ đĩa hoặc đĩa bị đầy.

+) Ví dụ:

FILE f;

f = **fopen**("baitapc.txt", "rt");

fclose(f);

9.3.1.4. Hàm feof( )

+) Dạng hàm:

foef (biến tệp tin);

+) Ý nghĩa: Trả về TRUE nếu đã đến cuối tệp tin (End Of File), nếu không nó trả về FALSE. Hàm này chỉ sử dụng khi đọc tệp tin nhị phân.

9.3.2. Một số hàm có chức năng điều khiển

9.3.2.1. Làm sạch vùng đệm

+) Dạng hàm:

Dạng 1: **int fflush**(FILE *fp);

Dạng 2: **int fflushall** (void);

+) Ý nghĩa:

- Hàm **fflush( )** dùng để làm sạch vùng đệm của tệp fp (fp là con trỏ tệp). Nếu thành công hàm cho giá trị 0, ngược lại hàm trả về EOF.
- Hàm **fflushall( )** dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công, hàm cho giá trị nguyên bằng số tệp đang mở, ngược lại, hàm cho EOF.

9.3.2.2. Kiểm tra lỗi

+) Hàm kiểm tra lỗi thao tác:

- Dạng hàm:

int ferror (FILE *fp);

- Ý nghĩa: Hàm dùng để kiểm tra lỗi thao tác trên tệp fp. Hàm trả về giá trị 0 nếu không có lỗi, ngược lại, nếu có lỗi hàm cho kết quả khác 0.

+) Hàm thông báo lỗi hệ thống:

- Dạng hàm:

void perror (“chuỗi thông báo”);

- Ý nghĩa: Hàm hiển thị chuỗi thông báo lỗi lên màn hình.

9.3.2.3. Xóa tệp tin

+) Dạng hàm:

remove (“tên tệp tin”);

+) Ý nghĩa: Xóa tệp tin có tên đặt trong cặp dấu “ ” trên đĩa.

9.3.2.4. Đổi tên tệp tin

+) Dạng hàm:

rename (“tên tệp tin cũ”, “tên tệp mới”);

+) Ý nghĩa: Đổi tên của tệp cũ thành tên của tệp mới, trong đó:

- Tên tệp cũ là tên của tệp đang có trong thư mục và ổ đĩa hiện hành. Bạn phải chỉ đúng đường dẫn nến như tệp tin cần đổi tên nằm ở thư mục khác
- Tên tệp mới là tên mới được đặt cho tệp cũ. Với tên tệp mới này, bạn không cần chỉ định đường dẫn vì nó được mặc định là nằm trong cùng thư mục của tệp cũ.

+) Thí dụ: Chương trình 9.1 /* Xóa một tệp tin trên đĩa, nếu không có tệp đó trên đĩa thì hiển thị thông báo lỗi.*/

```
#include "stdio.h"
#include "conio.h"
Void main()
{
  int xoa;
  xoa = remove ("tep.txt");
  if (xoa==0)
        printf ("\nDa xoa tep thanh cong");
  else
    {
        printf("\nkhong xoa duoc tep");
        perror ("\nVi:");
     }
  getch();
}
```

9.3.3. Các hàm ghi dữ liệu

9.3.3.1. Hàm putc( ) và fputc( )

+) Dạng hàm:

putc (int c, biến tệp tin);

fputc (int c, biến tệp tin);

+) Ý nghĩa: Ghi lên tệp tin ứng với 'biến tệp tin' một ký tự có mã 'c % 256' (phép chia lấy phần nguyên). Với c là số nguyên không dấu.

Nếu thành công hàn cho mã ký tự được ghi, nếu thất bại, hàm cho EOF.

Hai hàm trên có ý nghĩa như nhau.

+) Ví dụ: Câu lệnh:

putc(25100,f1);

thực hiện ghi lên tệp được xác định bởi biến tệp tin là f1 một ký tự ASCII có mã là (25100 % 256) = 98. Tức sẽ ghi ký tự 'b' lên tệp.

Một ví dụ khác, ta xét câu lệnh sau:

putc(-1, f1);

Câu lệnh này sẽ ghi lên tệp xác định bởi biến f1 một ký tự ASCII có mã là (65535 % 256) = 255 (vì dạng không dấu của -1 là 65535).

9.3.3.2. Hàm fprintf ( )

+) Dạng hàm:

fprintf(biến tệp tin, "chuỗi điều khiển", danh sách đối số);

+) Ý nghĩa: Hàm thực hiện ghi dữ liệu theo định dạng lên tệp tin. Biến tệp tin dùng để xác định file để ghi dữ liệu . "Chuỗi điều khiển" và danh sách đối số có cùng ý nghĩa như trong hàm **printf( ).**

+) Ví dụ: Xét chương trình sau:

Chương trình 9.2/* Tạo một tệp tin và ghi dữ liệu lên tệp đó*/

```
#include "stdio.h"
void main()
{
  FILE *f;
  f = fopen("file", "wt");
  fprintf(f, "nhap du lieu cho file:");
  fprintf(f, "in ma ASCII : %c", 49);
  fclose(f);
}
```

Kết quả chương trình là tạo ra một tệp tin có tên là "file" có dữ liệu là:

```
Nhap du lieu cho file:
In ma ASCII: !
```

9.3.3.3. Hàm fputs( )

+) Dạng hàm:

fputs("xâu", biến tệp tin);

+) Ý nghĩa: Ghi một xâu ký tự lên tệp có tên 'biến tệp tin'(không ghi ký tự '\0' lên tệp). Nếu có lỗi, hàm trả về EOF.

9.3.4. Các hàm đọc dữ liệu từ file

9.3.4.1. Hàm fscanf( )

+) Dạng hàm:

fscanf (biến tệp tin, "chuỗi điều khiển", danh sách đối);

+) Ý nghĩa: Đọc dữ liệu có định dạng từ tệp tin được xác định theo biến tệp tin. Chuỗi điều khiển và danh sách đối số được sử dụng như hàm **scanf( ).**

+) Ví dụ: Câu lệnh:

fscanf(tep, "%3d %s",&a, xau);

Sẽ đọc từ tệp tin được định nghĩa theo biến tệp tin "tep" giá trị của số nguyên a và xâu ký tự tương ứng với đối số là "xau".

9.3.4.2. Hàm fgets( )

+) Dạng hàm:

fgets (char *s, int n, biến tệp tin);

+) Ý nghĩa: Đọc một chuỗi ký tự từ tệp được xác định bởi biến tệp tin. Với n là độ dài cực đại của chuỗi cần đọc và s là biến con trỏ kiểu char trỏ tới một vùng nhớ đủ lớn để lưu chuỗi ký tự vừa đọc từ tệp đó.

Chú ý: Việc đọc dãy ký tự kết thúc khi:

- Hoặc đã đọc n-1 ký tự.
- Hoặc gặp dấu xuống dòng (cặp mã 13 10), khi đó mã 10 được đưa vào xâu kết quả.
- Hoặc khi kết thúc tệp.

Xâu kết quả sẽ được bổ sung thêm ký tự kết thúc '\0'.

9.3.4.3. Hàm getc( )

+) Dạng hàm:

getc (biến tệp tin);

+) Ý nghĩa:

Hàm **getc( )** thực hiện đọc một ký tự từ tệp được xác định bởi biến tệp tin. Nếu thành công hàm trở về mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi thì hàm cho EOF.

+) Ví dụ: Xét chương trình sau:

Chương trình 9.3/*Chương trình sao chép tệp*/

```
#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
void main()
{
        int c;
        char tep1[15], tep2[15];
        FILE *f1,*f2;
printf("\n tep nguon:");
        gets(tep1);
        printf("\n Tep dich:");
        gets(tep2);
        f1 = fopen(tep1,"rb");      /*mo tep1 de doc*/
        if(f1 == NULL)
        {
            printf("\nTep %s khong ton tai",ten1);
            getch();
            exit(1);
        }
```

```
        f2 = fopen(tep2,"wb"); /*mo tep 2 de ghi*/
/* Thuc hien sao chep tep*/
  while((c=getc(f1))!= EOF)  putc(c,f2);
        fclose(f1);
fclose(f2);
}
```

Chương trình thực hiện sao tệp theo thuật toán là:

- Mở tệp thứ nhất (tệp nguồn) để đọc. Nếu tệp không tồn tại thì thoát khỏi chương trình. Ngược lại, thực hiện bước tiếp theo.
- Mở tệp thứ hai (tệp đích) để ghi.
- Đọc một ký tự từ tệp nguồn, kết quả đặt vào biến c.
- Nếu c = EOF thì kết thúc, nếu c khác EOF thì ghi c vào tệp đích và tiếp tục đọc ký tự tiếp theo.

9.4. Thao tác trên tệp tin nhị phân

Đối với tệp tin nhị phân, các thao tác về khai báo, đóng/ mở tệp, thay đổi tên, làm sạch vùng đệm… hoàn toàn giống với tệp tin văn bản. Tuy nhiên, các lệnh ghi và đọc tệp tin nhị phân thì khác với tệp văn bản. Do đó, trong phần này chúng ta sẽ chỉ xét đến các lệnh ghi đọc với tệp nhị phân.

9.4.1. Các hàm ghi dữ liệu

9.4.1.1. Hàm putw( )

+) Dạng hàm:

int putw (int n, FILE *f);

+) Ý nghĩa: Ghi giá trị n dưới dạng 2 byte lên tệp f. Nếu thành công, hàm trả về số nguyên được ghi; ngược lại, khi có lỗi hàm trả về EOF.

9.4.1.2. Hàm fwrite( )

+) Dạng hàm:

int fwrite (void *ptr, int size, int n, FILE *fp)

Trong đó:

- ptr: là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi
- size: là kích thước của mẫu tin theo byte
- n: là số mẫu tin cần ghi
- fp: là con trỏ tệp.

+) Ý nghĩa: Ghi n mẫu tin có kích thước là size byte từ vùng nhớ được trỏ bởi con trỏ ptr lên tệp fp. Hàm trả về một giá trị bằng số mẫu tin thực sự được ghi.

9.4.2. Các hàm đọc dữ liệu

9.4.2.1. Hàm getw( )

+) Dạng hàm:

*int getw**(FILE *fp);***

+***) Ý nghĩa:*** Đọc một số nguyên 2 byte từ tệp fp. Nếu thành công, hàm trả về số nguyên đọc được, nếu có lỗi hoặc gặp cuối tệp hàm trả về EOF.

9.4.2.2. Hàm fread( )

+***) Dạng hàm:***

int fread (void *ptr, int size, int n, FILE *fp);

Trong đó:

- ptr: là con trỏ trỏ tới vùng nhớ để chứa dữ liệu được đọc.
- size: là kích thước của mẫu tin theo byte
- n: là số mẫu tin cần đọc
- fp: là con trỏ tệp.

+***) Ý nghĩa:*** Đọc n mẫu tin kích thược là size byte từ tệp fp và chứa vào vùng nhớ được trỏ bởi ptr.

9.4.3. Di chuyển con trỏ tập tin - Hàm fseek()

Việc ghi hay đọc dữ liệu từ tập tin sẽ làm cho con trỏ tập tin dịch chuyển một số byte, đây chính là kích thước của kiểu dữ liệu của mỗi phần tử của tập tin. Khi đóng tập tin rồi mở lại nó, con trỏ luôn ở vị trí ngay đầu tập tin. Nếu sử dụng kiểu mở tập tin là “a” để ghi nối dữ liệu, con trỏ tập tin sẽ di chuyển đến vị trí cuối cùng của tập tin này.

Người lập trình cũng có thể điều khiển việc di chuyển con trỏ tập tin đến vị trí chỉ định bằng hàm fseek().

Cú pháp:

int fseek (FILE *f, long offset, int whence)

Trong đó:

- f: con trỏ tập tin đang thao tác.
- offset: số byte cần dịch chuyển con trỏ tập tin kể từ vị trí trước đó. Phần tử đầu tiên là vị trí 0.
- whence: vị trí bắt đầu để tính offset, ta có thể chọn điểm xuất phát là:

0	SEEK_SET	Vị trí đầu của tệp tin
1	SEEK_CUR	Vị trí hiện tại của tệp tin
2	SEEK_END	Vị trí cuối của tệp tin

- Kết quả trả về của hàm là 0 nếu việc di chuyển thành công. Nếu không thành công, 1 giá trị khác 0 (đó là 1 mã lỗi) được trả về.

Chương trình 9.4/* Nhập vào một danh sách các diện trở rồi lưu danh sách đó vào một tệp “DT.DAT” (tệp nhị nhân), rồi đọc danh sách đó từ tệp ra màn hình.*/

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

```
#include<conio.h>
#include<string.h>
#include<io.h>      //chua ham eof
typedef struct {
    char   name[15];
    float value;
    } DienTro;
void NhapDS(DienTro *DT,int n){
 float t;
 DienTro p;
 // phai nhap qua cau truc trung gian;
   //ko nhap truc tiep duoc
 for(int i=0;i<n;i++)
  {
    printf("Nhap ten dien tro: ");gets(p.name);
    printf("Nhap gia tri DT: ");
    scanf("%f%*c",&t); //Nhap xong xoa bo dem ban phim
    p.value=t;
    (*(DT+i))=p;// dau cham co do uu tien cao hon toan tu * nen fai viet (*(DT+i))
  }
}
void InDS(DienTro DT[50],int n){
 float t;
 for(int i=0;i<n;i++)
  {
    printf("Ten dien tro: %15s ",DT[i].name);
    printf("   Gia tri DT: %5.2f\n",DT[i].value);
  }
}
void GhiTep(DienTro *DT,int n){
   FILE *f;  DienTro p;
   f=fopen("DT.DAT","w+");
   if (f!=NULL) {//neu mo tep thanh cong
      for(int i=0;i<n;i++){
    fwrite(DT+i,sizeof(DienTro),1,f);
      }
   rewind(f); //dua con tro ve dau tep de doc tep
```

```
  //Doc du lieu tu tep
  printf("\n\nDoc danh sach tu tep\n");
  while(fread(&p,sizeof(DienTro),1,f)>0){ //hoac cach khac while(!feof(f){})
    //doc tung phan tu tu tep
   printf("Ten dien tro: %15s ",p.name);
   printf("   Gia tri DT: %5.2f\n",p.value);
  }
 }
 fclose(f);//dong tep
}
void main(){
  DienTro DT[100];int n;
  clrscr();
  printf("Nhap vao so phan tu cua mang: ");
  scanf("%d%*c",&n);// phai xoa bo dem trc khi su dung ham gets()
  NhapDS(DT,n);
  InDS(DT,n) ;
  GhiTep(DT,n);
getch();
}
```

Chương trình 9.5 /*Ghi lên tập tin CacSo.Dat 3 giá trị số (thực, nguyên, nguyên dài). Sau đó đọc các số từ tập tin vừa ghi và hiển thị lên màn hình.*/

```
#include<stdio.h>
#include<conio.h>
 int main()
{
 FILE *f;
 clrscr();
 f=fopen("D:\\CacSo.txt","wb");
 if (f!=NULL)
  {
        double d=3.14;
     int i=101;
        long l=54321;
        fwrite(&d,sizeof(double),1,f);
        fwrite(&i,sizeof(int),1,f);
        fwrite(&l,sizeof(long),1,f);
```

```
/* Doc tu tap tin*/
            rewind(f);
            fread(&d,sizeof(double),1,f);
            fread(&i,sizeof(int),1,f);
            fread(&l,sizeof(long),1,f);
            printf("Cac ket qua la: %f  %d  %ld",d,i,l);
            fclose(f);
}
 getch();
  return 0;
}
```

Bài tập chương

Bài 1. Viết chương trình nhập vào một mảng các số nguyên rồi ghi các thông tin của mảng vừa nhập vào một tệp có tên "Mang.txt" theo cách:

- Dòng đầu tiên của tệp là giá trị của n
- Các dòng tiếp theo là các phần tử của mảng, mỗi phần tử trên một dòng
- Dòng cuối cùng là tổng các phần tử của mảng.

Bài 2. Viết chương trình nhập vào một danh sách các điện trở, mỗi điện trở gồm các thông tin: Ten (tên), GiaTri (giá trị), và ghi các thông tin của mảng các điện trở vào một tệp có tên "DienTro.DAT". Sau đó đọc thông tin của mảng các điện trở từ tệp và in kết quả ra màn hình.