

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM

Kỹ THUẬT LẬP TRÌNH

Biên Soạn:

ThS. Nguyễn Thúy Loan

ThS. Văn Thị Thiên Trang

KỸ THUẬT LẬP TRÌNH

Ấn bản 2018

*Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập:
tailieuhoctap@hutech.edu.vn*

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN	IV
BÀI 1: CÁC KHÁI NIỆM CƠ BẢN VỀ LẬP TRÌNH.....	1
1.1 TỪ BÁI TOÁN ĐẾN CHƯƠNG TRÌNH.....	1
1.2 GIẢI THUẬT	2
1.2.1 Khái niệm giải thuật	2
1.2.2 Các đặc trưng của giải thuật.....	3
1.2.3 Ngôn ngữ biểu diễn giải thuật.....	4
1.2.4 Các cấu trúc suy luận cơ bản của giải thuật	5
1.3 NGÔN NGỮ LẬP TRÌNH.....	11
1.3.1 Khái niệm ngôn ngữ lập trình.....	11
1.3.2 Chương trình dịch.....	12
1.3.3 Ngôn ngữ lập trình C	12
1.4 MÔI TRƯỜNG DEV-C	14
CÂU HỎI ÔN TẬP	16
BÀI 2: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C	18
2.1 CÁC KHÁI NIỆM CƠ BẢN	18
2.1.1 Các ký tự dùng trong ngôn ngữ C.....	18
2.1.2 Các từ khoá trong C.....	19
2.1.3 Cách đặt tên trong C	20
2.1.4 Cặp dấu ghi chú thích	20
2.2 CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH C	21
2.2.1 Tiền xử lý và biên dịch	21
2.2.2 Cấu trúc một chương trình C.....	22
2.2.3 Các tập tin thư viện thông dụng	23
2.2.4 Cú pháp khai báo các phần bên trong một chương trình C	24
2.2.5 Cấu trúc một chương trình C đơn giản.....	25
2.3 KIỂU DỮ LIỆU CƠ BẢN, BIẾN, HẲNG	26
2.3.1 Các kiểu dữ liệu cơ bản	26
2.3.2 Hằng (Constant)	27
2.3.3 Biến	30
2.4 BIỂU THỨC, PHÉP TOÁN, CHUYỂN ĐỔI KIỂU DỮ LIỆU	32
2.4.1 Khái niệm biểu thức.....	32
2.4.2 Các phép toán số học	33
2.4.3 Phép gán (lệnh gán)	34
2.4.4 Các phép toán quan hệ và lý luận.....	35
2.4.5 Chuyển đổi kiểu giá trị	38
2.4.6 Phép toán tăng giảm	39

2.4.7 Biểu thức điều kiện.....	40
2.4.8 Các phép toán thao tác trên bit.....	41
2.4.9 Toán tử con trỏ & và *.....	42
2.4.10 Toán tử dấu phẩy (,).....	43
2.5 NHẬP, XUẤT DỮ LIỆU.....	44
2.5.1 Lệnh nhập giá trị từ bàn phím cho biến (hàm scanf)	44
2.5.2 Lệnh xuất giá trị của biểu thức lên màn hình (hàm printf)	45
CÂU HỎI ÔN TẬP.....	49
BÀI 3: CẤU TRÚC ĐIỀU KHIỂN.....	51
3.1 KHÁI NIỆM CÂU LỆNH VÀ KHỐI LỆNH	51
3.1.1 Khái niệm câu lệnh.....	51
3.1.2 Phân loại.....	51
3.2 CÁC LỆNH CÓ CẤU TRÚC.....	53
3.2.1 Cấu trúc rẽ nhánh	53
3.2.2 Cấu trúc lựa chọn switch	57
3.2.3 Cấu trúc lặp for.....	58
3.2.4 Cấu trúc lặp while	62
3.2.5 Vòng lặp do... while	64
3.3 CÁC CÂU LỆNH ĐẶC BIỆT	65
CÂU HỎI ÔN TẬP.....	67
BÀI 4: CHƯƠNG TRÌNH CON.....	68
4.1 KHÁI NIỆM HÀM TRONG C.....	68
4.2 HÀM THƯ VIỆN.....	69
4.3 HÀM NGƯỜI DÙNG	70
4.3.1 Xây dựng một hàm.....	70
4.3.2 Sử dụng hàm.....	71
4.3.3 Nguyên tắc hoạt động của hàm.....	72
4.3.4 Truyền tham số cho hàm	72
CÂU HỎI ÔN TẬP.....	75
BÀI 5: MẢNG	76
5.1 KHÁI NIỆM.....	76
5.2 MẢNG MỘT CHIỀU	77
5.2.1 Khai báo	77
5.2.2 Truy cập vào các phần tử của mảng	78
5.2.3 Nhập dữ liệu cho mảng một chiều	79
5.2.4 Xuất dữ liệu cho mảng một chiều	80
5.2.5 Một vài thuật toán trên mảng một chiều	80
5.3 CHUỖI KÝ TỰ (MẢNG MỘT CHIỀU CÁC KÝ TỰ).....	82
5.3.1 Cách khai báo chuỗi	82
5.3.2 Các thao tác trên chuỗi ký tự.....	83
5.3.3 Một số hàm xử lý chuỗi (trong <string.h>).....	84

5.4 MẢNG HAI CHIỀU	90
5.4.1 Khai báo	90
5.4.2 Truy cập vào các phần tử của mảng	91
5.4.3 Nhập dữ liệu cho mảng hai chiều.....	92
5.4.4 Xuất dữ liệu cho mảng hai chiều	93
CÂU HỎI ÔN TẬP	94
BÀI 6: KIỂU DỮ LIỆU CÓ CẤU TRÚC.....	97
6.1 KHÁI NIỆM.....	97
6.2 CÁCH KHAI BÁO KIỂU CẤU TRÚC.....	98
6.3 TRUY CẬP VÀO TỪNG PHẦN TỬ CỦA CẤU TRÚC	99
6.4 NHẬP, XUẤT DỮ LIỆU CHO KIỂU DỮ LIỆU CÓ CẤU TRÚC	100
6.4.1 Nhập dữ liệu	100
6.4.2 Xuất dữ liệu.....	100
6.5 MẢNG CẤU TRÚC	101
CÂU HỎI ÔN TẬP	102
BÀI 7: ĐỆ QUY.....	103
7.1 KHÁI NIỆM.....	103
7.2 KỸ THUẬT GIẢI BÀI TOÁN BẰNG ĐỆ QUY	104
7.3 MỘT SỐ NHẬN XÉT VỀ HÀM ĐỆ QUY	105
7.4 SO SÁNH CẤU TRÚC LẬP VÀ ĐỆ QUY	105
CÂU HỎI ÔN TẬP	106
BÀI 8: TẬP TIN (FILE).....	107
8.1 KHÁI NIỆM.....	107
8.2 CÁC THAO TÁC TRÊN TẬP TIN	108
8.2.1 Khai báo biến tập tin	108
8.2.2 Mở tập tin	109
8.2.3 Đóng tập tin	110
8.2.4 Kiểm tra đến cuối tập tin hay chưa?.....	110
8.2.5 Di chuyển con trỏ tập tin về đầu tập tin - Hàm rewind().....	111
8.3 TRUY CẬP TẬP TIN VĂN BẢN.....	111
8.3.1 Ghi dữ liệu lên tập tin văn bản	111
8.3.2 Đọc dữ liệu từ tập tin văn bản.....	113
8.4 TRUY CẬP TẬP TIN NHỊ PHÂN	114
8.4.1 Ghi dữ liệu lên tập tin nhị phân - Hàm fwrite()	114
8.4.2 Đọc dữ liệu từ tập tin nhị phân - Hàm fread()	115
CÂU HỎI ÔN TẬP	119
TÀI LIỆU THAM KHẢO	120

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Môn Kỹ thuật Lập trình cung cấp cho sinh viên những kiến thức cơ bản về lập trình thông qua ngôn ngữ lập trình C. Môn học này là nền tảng để tiếp thu hầu hết các môn học khác trong chương trình đào tạo. Mặt khác, nắm vững môn này là cơ sở để phát triển tư duy và kỹ năng lập trình để giải các bài toán và các ứng dụng trong thực tế.

Học xong môn này, sinh viên phải nắm được các vấn đề sau:

- Khái niệm về ngôn ngữ lập trình.
- Ngôn ngữ sơ đồ (lưu đồ), sử dụng lưu đồ để biểu diễn các giải thuật.
- Tổng quan về Ngôn ngữ lập trình C.
- Các kiểu dữ liệu trong C.
- Các lệnh có cấu trúc.
- Cách thiết kế và sử dụng các hàm trong C.
- Xử lý các bài toán trên mảng một chiều.
- Xử lý các bài toán trên mảng hai chiều.
- Biết kỹ thuật viết đệ quy.
- Biết xây dựng và xử lý các bài toán trên dữ liệu có cấu trúc do người dùng định nghĩa.
- Cách lưu trữ và xử lý các file trong C.

NỘI DUNG MÔN HỌC

- Bài 1 CÁC KHÁI NIỆM CƠ BẢN VỀ LẬP TRÌNH.
- Bài 2 CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C/C++.
- Bài 3 CẤU TRÚC ĐIỀU KHIỂN
- Bài 4 CHƯƠNG TRÌNH CON
- Bài 5 MẢNG

- Bài 6 KIỂU DỮ LIỆU CÓ CẤU TRÚC
- Bài 7 ĐỆ QUY
- Bài 8 TẬP TIN (FILE)

YÊU CẦU MÔN HỌC

Có tư duy tốt về logic và kiến thức toán học cơ bản.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Kỹ thuật lập trình là môn học đầu tiên giúp sinh viên làm quen với phương pháp lập trình trên máy tính, giúp sinh viên có khái niệm cơ bản về cách tiếp cận và giải quyết các bài toán tin học, giúp sinh viên có khả năng tiếp cận với cách tư duy của người lập trình, và là tiền đề để tiếp cận với các học phần quan trọng còn lại của ngành Công nghệ Thông tin. Vì vậy, yêu cầu người học phải dự học đầy đủ các buổi lên lớp, làm bài tập đầy đủ ở nhà, nghiên cứu tài liệu trước khi đến lớp và gạch chân những vấn đề không hiểu khi đọc tài liệu để đến lớp trao đổi.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Để học tốt môn này, người học cần ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, người học đọc trước mục tiêu bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người đọc trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, người đọc làm các bài tập.

Điểm đánh giá: gồm 2 phần

1. Điểm quá trình (chuyên cần + bài tập)
2. Điểm cuối kỳ (bài thi trên giấy - tự luận).

BÀI 1: CÁC KHÁI NIỆM CƠ BẢN VỀ LẬP TRÌNH

Sau khi học xong bài này, sinh viên có thể nắm được về cơ bản:

- Khái niệm về ngôn ngữ lập trình.
- Khái niệm về kiểu dữ liệu
- Khái niệm về giải thuật
- Ngôn ngữ biểu diễn giải thuật.
- Ngôn ngữ sơ đồ (lưu đồ), sử dụng lưu đồ để biểu diễn các giải thuật.
- Tổng quan về ngôn ngữ lập trình C.
- Môi trường làm việc và cách sử dụng DevC/ C-Free.
- Cài đặt một số bài toán đơn giản bằng ngôn ngữ C.

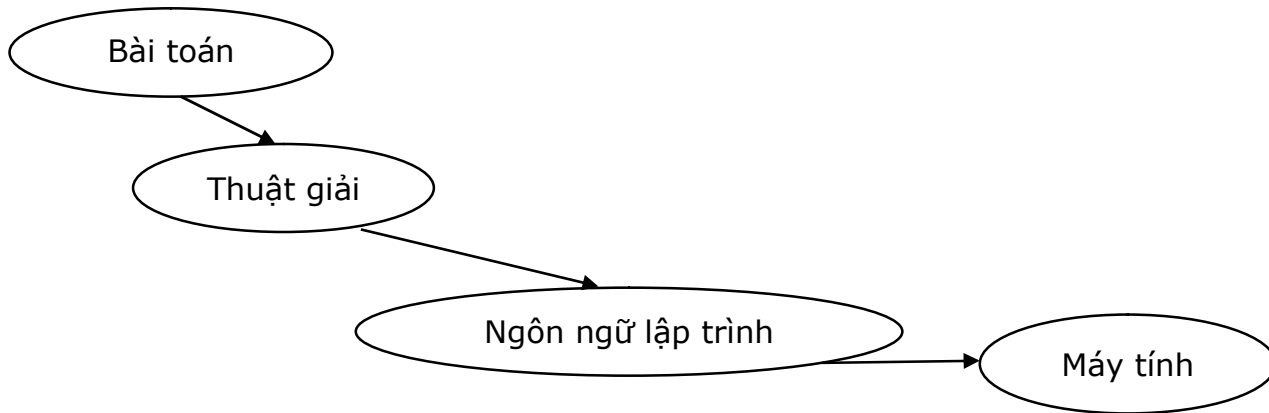
1.1 TỪ BÀI TOÁN ĐẾN CHƯƠNG TRÌNH

Giả sử chúng ta cần viết một chương trình để giải phương trình bậc 2 có dạng $ax^2 + bx + c = 0$, hay viết một chương trình kiểm tra xem một số tự nhiên nào đó có phải là số nguyên tố hay không? Công việc đầu tiên là chúng ta phải hiểu và biết cách giải bài toán bằng lời giải thông thường của người làm toán. Để giải được bài toán trên bằng máy tính (lập trình cho máy tính giải) thì chúng ta cần phải thực hiện qua các bước như sau:

1. Mô tả các bước giải bài toán.
2. Vẽ sơ đồ xử lý dựa trên các bước.
3. Dựa trên sơ đồ xử lý để viết chương trình xử lý bằng ngôn ngữ giả (ngôn ngữ bình thường của chúng ta).

4. Chọn ngôn ngữ lập trình và chuyển chương trình từ ngôn ngữ giả sang ngôn ngữ lập trình để tạo thành một chương trình hoàn chỉnh.
5. Thực hiện chương trình: nhập vào các tham số, nhận kết quả.

Trong nhiều trường hợp, từ bài toán thực tế chúng ta phải xây dựng mô hình toán học rồi mới xác định được các bước để giải. Vấn đề này sẽ được trình bày chi tiết trong môn Cấu Trúc Dữ Liệu.



1.2 GIẢI THUẬT ---

1.2.1 Khái niệm giải thuật

Giải thuật là một hệ thống chặt chẽ và rõ ràng các quy tắc nhằm xác định một dãy các thao tác trên những dữ liệu vào sao cho sau một số hữu hạn bước thực hiện các thao tác đó ta thu được kết quả của bài toán.

Ví dụ 1: Giả sử có hai bình A và B đựng hai loại chất lỏng khác nhau, chẳng hạn bình A đựng rượu, bình B đựng nước mắt. Giải thuật để hoán đổi (swap) chất lỏng đựng trong hai bình đó là:

❖ **Yêu cầu phải có thêm một bình thứ ba gọi là bình C.**

- Bước 1: Đổ rượu từ bình A sang bình C.
- Bước 2: Đổ nước mắt từ bình B sang bình A.
- Bước 3: Đổ rượu từ bình C sang bình B.

Ví dụ 2: Một trong những giải thuật tìm ước chung lớn nhất của hai số a và b là:

- Bước 1: Nhập vào hai số a và b.

- Bước 2: So sánh 2 số a, b chọn số nhỏ nhất gán cho UCLN.
- Bước 3: Nếu một trong hai số a hoặc b không chia hết cho UCLN thì thực hiện bước 4, ngược lại (cả a và b đều chia hết cho UCLN) thì thực hiện bước 5.
- Bước 4: Giảm UCLN một đơn vị và quay lại bước 3
- Bước 5: In UCLN - Kết thúc.

1.2.2 Các đặc trưng của giải thuật

Khác với các bài toán thuần túy toán học chỉ cần xác định rõ giả thiết và kết luận chứ không cần xác định yêu cầu về lời giải.

Tuy nhiên đối với những bài toán tin học ứng dụng trong thực tế chúng ta chỉ cần tìm lời giải tốt tới mức nào đó, thậm chí tồi ở mức chấp nhận được nếu lời giải tốt đòi hỏi thời gian và chi phí.

Ví dụ 3: khi cài đặt các hàm số phức tạp trên máy tính. Nếu tính bằng cách khai triển chuỗi vô hạn thì độ chính xác cao hơn nhưng tốc độ chậm hơn rất nhiều so với phương pháp xấp xỉ

Việc xác định đúng yêu cầu bài toán là rất quan trọng bởi nó ảnh hưởng tới cách thức giải quyết và chất lượng của lời giải. Một bài toán thực tế thường cho bởi những thông tin khá mơ hồ và hình thức, ta phải phát biểu lại một cách chính xác và chặt chẽ để hiểu đúng bài toán.

Ví dụ 4: Lập trình tự động cho bài toán giặt máy.

Việc xác định độ dơ của quần áo cần giặt thì khá mơ hồ.

Trên thực tế, ta nên xét một vài trường hợp cụ thể để thông qua đó hiểu được bài toán rõ hơn và thấy được các bước cần thiết phải tiến hành.

Khái niệm giải thuật hay thuật giải mà nhiều khi còn được gọi là thuật toán dùng để chỉ phương pháp hay cách thức (method) để giải quyết vấn đề.

Một thuật giải tốt là thuật giải phải bảo đảm:

1. **Tính kết thúc:** Giải thuật phải dừng sau một số hữu hạn bước.
2. **Tính xác định:** Các thao tác máy tính phải thực hiện được và các máy tính khác nhau thực hiện cùng một bước của cùng một giải thuật phải cho cùng một kết quả.

3. **Tính phổ dụng:** Giải thuật phải "vét" hết các trường hợp và áp dụng cho một loạt bài toán cùng loại.
4. **Tính hiệu quả:** Một giải thuật được đánh giá là tốt nếu nó đạt hai tiêu chuẩn sau:
- Thực hiện nhanh, tốn ít thời gian.
 - Tiêu phí ít tài nguyên của máy, chẳng hạn tốn ít bộ nhớ.

Giải thuật tìm UCLN nêu trên đạt tính kết thúc bởi vì qua mỗi lần thực hiện bước 4 thì UCLN sẽ giảm đi một đơn vị cho nên trong trường hợp xấu nhất thì $UCLN=1$, giải thuật phải dừng. Các thao tác trình bày trong các bước, máy tính đều có thể thực hiện được nên nó có tính xác định. Giải thuật này cũng đạt tính phổ dụng vì nó được dùng để tìm UCLN cho hai số nguyên dương a và b bất kỳ. Tuy nhiên tính hiệu quả của giải thuật có thể chưa cao; cụ thể là thời gian chạy máy có thể còn tốn nhiều hơn một số giải thuật khác mà chúng ta sẽ có dịp trở lại trong phần lập trình C.

1.2.3 Ngôn ngữ biểu diễn giải thuật

Để biểu diễn giải thuật, cần phải có một tập hợp các ký hiệu dùng để biểu diễn, mỗi ký hiệu biểu diễn cho một hành động nào đó. Tập hợp các ký hiệu đó lại tạo thành ngôn ngữ biểu diễn giải thuật.

1. Ngôn ngữ tự nhiên (Natural language)

Là thứ ngôn ngữ mà chúng ta đang sử dụng, chúng ta có thể sử dụng ngôn ngữ tự nhiên để mô tả giải thuật giống như các ví dụ ở trên. nhưng có thể sự mô tả không rõ ràng trong ngôn ngữ dẫn đến hiểu sai giải thuật.

Ví dụ 5: Ta có giải thuật giải phương trình bậc 1 $ax + b = 0$ như sau:

- Bước 1: Nhận giá trị của các tham số a, b .
- Bước 2: Xét giá trị của a xem có bằng 0 hay không? Nếu $a=0$ thì làm bước 3, Nếu a khác không thì làm bước 4.
- Bước 3: (a bằng 0) Nếu b bằng 0 thì ta kết luận phương trình vô số nghiệm, nếu b khác 0 thì ta kết luận phương trình vô nghiệm.
- Bước 4: (a khác 0) Ta kết luận phương trình có nghiệm $x=-b/a$

2. Mã giả (pseudocode)



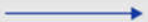


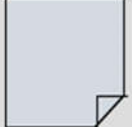



Chúng ta cũng có thể dùng một ngôn ngữ giả lập ngôn ngữ lập trình gọi là **mã giả (pseudocode)** để biểu diễn giải thuật cho một bài toán

3. Ngôn ngữ sơ đồ (Lưu đồ - flow chart)

Lưu đồ thuật toán là công cụ dùng để **biểu diễn thuật toán dưới dạng sơ đồ**, việc mô tả dữ liệu **nhập** (input), dữ liệu **xuất** (output) và luồng xử lý thông qua các **ký hiệu hình học**.

Một dạng quy ước chuẩn cho phép mô tả cách xử lý của chương trình một cách trực quan.

Các ký hiệu thường sử dụng để vẽ lưu đồ

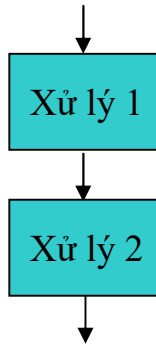
STT	KÝ HIỆU	DIỄN GIẢI
1		Bắt đầu chương trình
2		Kết thúc chương trình
3		Luồng xử lý
4		Điều khiển lựa chọn
5		Nhập
6		Xuất
7		Xử lý, tính toán hoặc gán
8		Trả về giá trị (return)
9		Điểm nối liên kết tiếp theo (Sử dụng khi lưu đồ vượt quá trang)

1.2.4 Các cấu trúc suy luận cơ bản của giải thuật

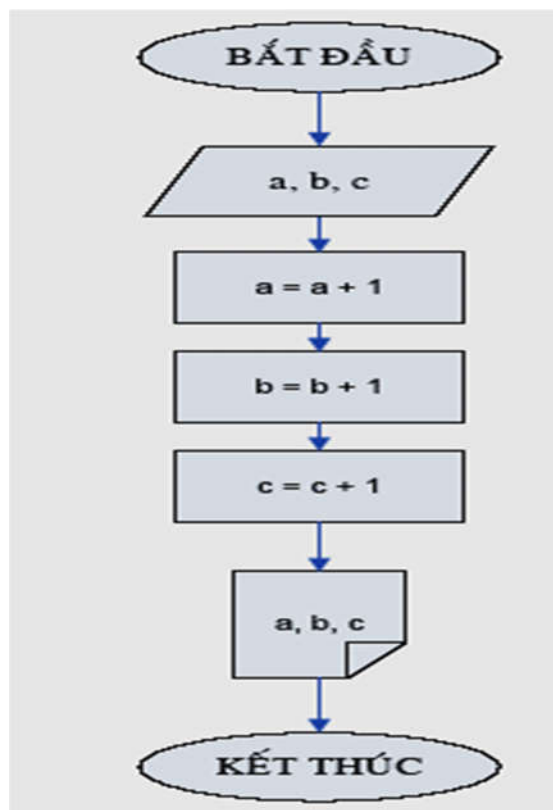
Giải thuật được thiết kế theo ba cấu trúc suy luận cơ bản sau đây:

1.2.4.1 Cấu trúc Tuần tự (Sequential)

Các công việc được thực hiện một cách tuần tự từ trên xuống, công việc này nối tiếp công việc kia.



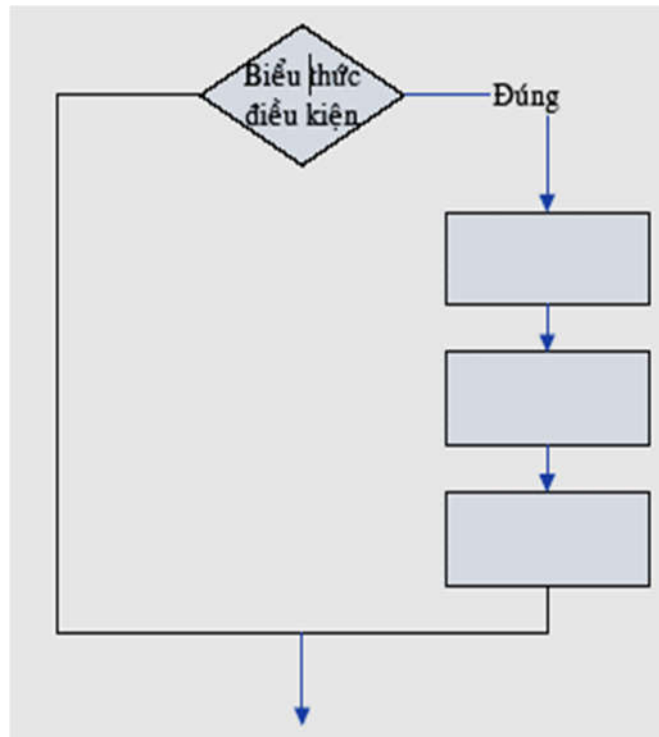
Bài toán 1: Nhập vào 3 số nguyên **a, b, c** và xuất ra màn hình với giá trị của mỗi số tăng lên 1.



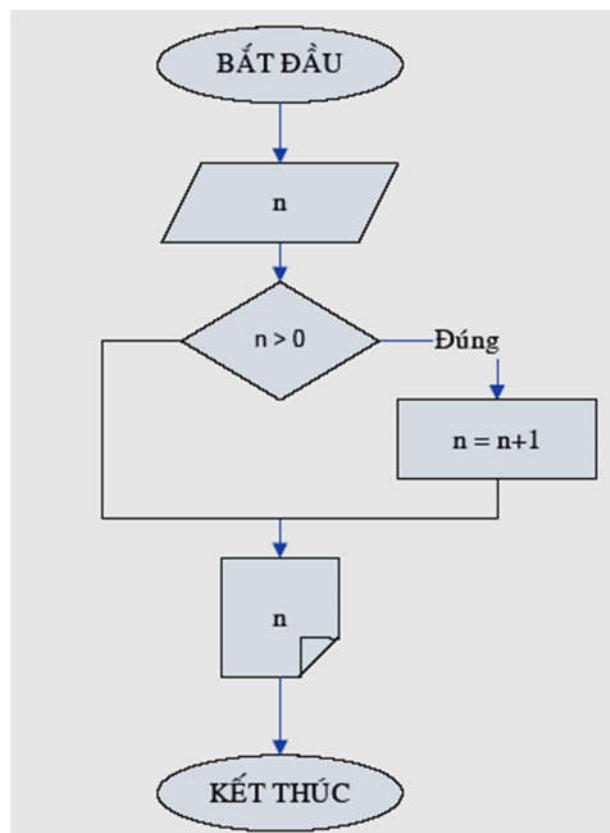
1.2.4.2 Cấu trúc lựa chọn (Selection)

Lựa chọn một công việc để thực hiện căn cứ vào một điều kiện nào đó. Có một số dạng như sau:

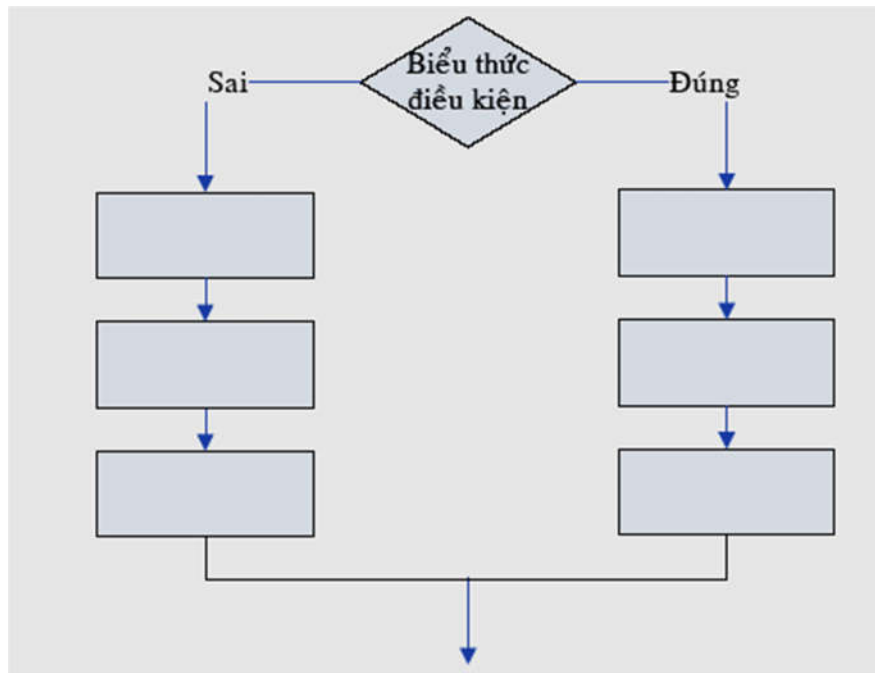
❖ **Cấu trúc 1:** Nếu <điều kiện> (đúng) thì thực hiện <công việc>.



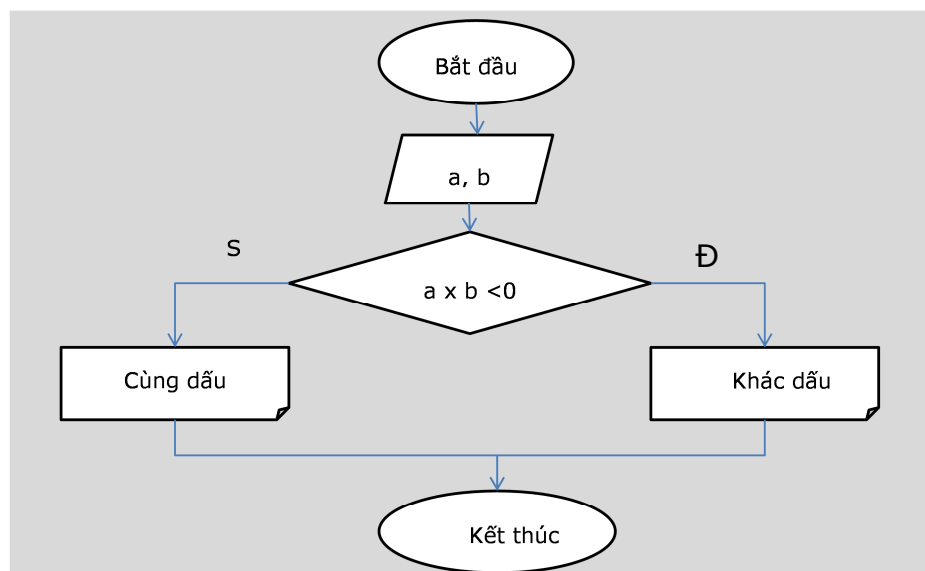
Bài toán 2: Nhập 1 số n , nếu số đó > 0 thì tăng 1. Sau đó hiển thị



❖ **Cấu trúc 2:** Nếu <điều kiện> (đúng) thì thực hiện <công việc 1>, ngược lại (điều kiện sai) thì thực hiện <công việc 2>

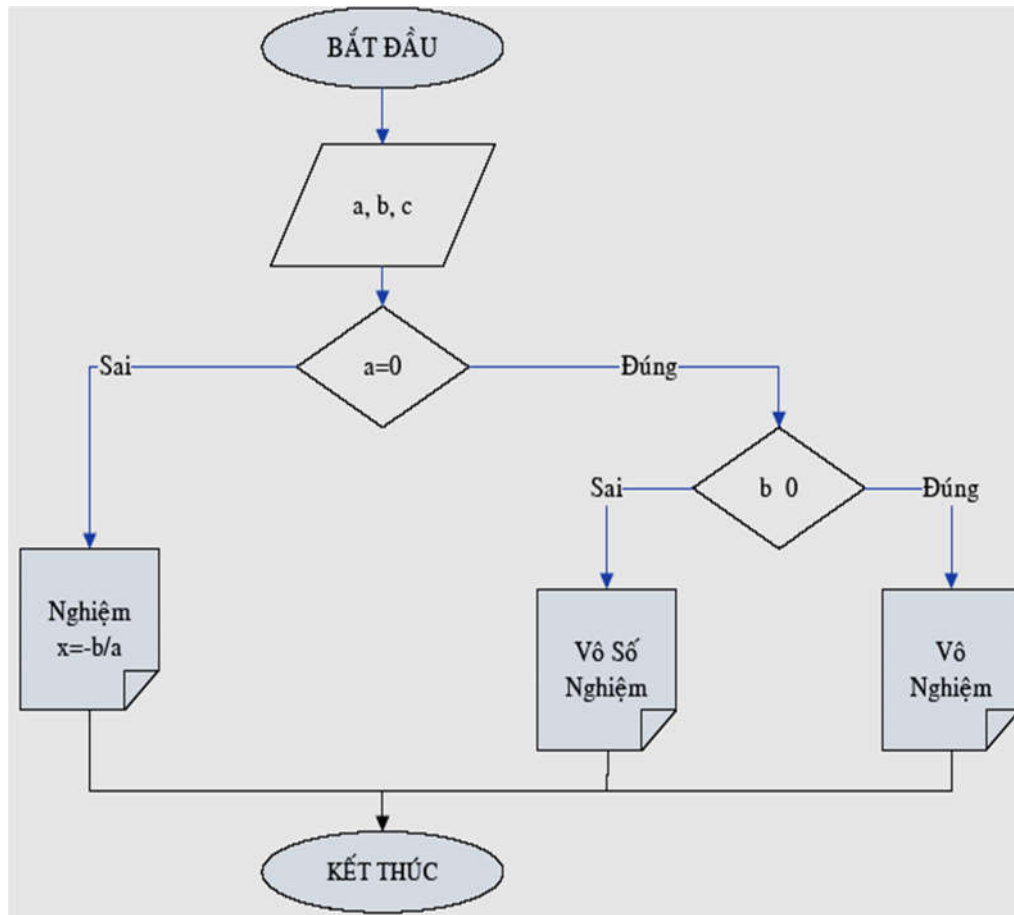


Bài toán 3: Viết chương trình nhập vào 2 số thực, cho biết 2 số đó cùng dấu hay khác dấu



❖ **Cấu trúc 3:** Trường hợp <i> thực hiện <công việc i>

Bài toán 4: Viết chương trình giải và biện luận phương trình bậc nhất $ax+b=0$

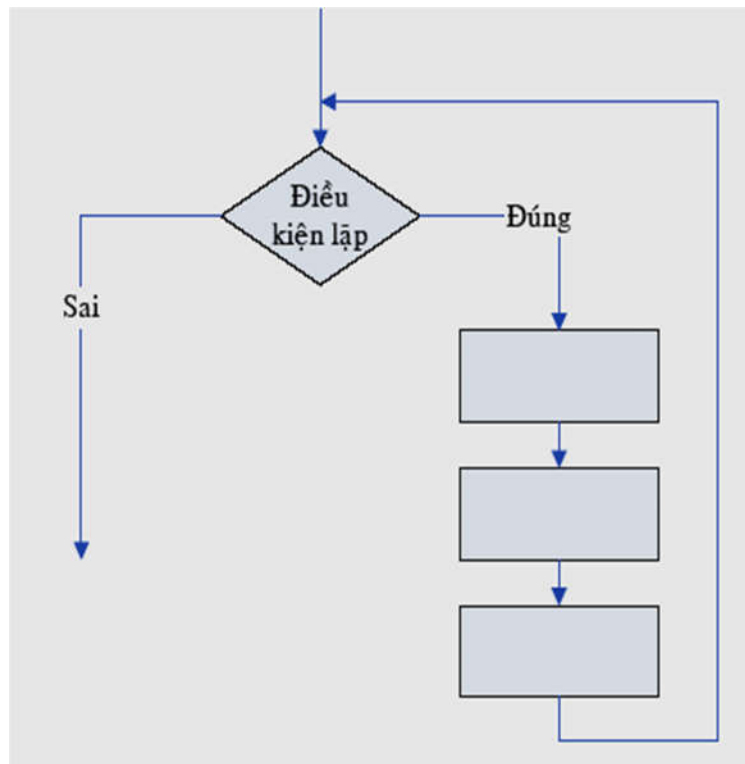


1.2.4.3 Cấu trúc lặp (Repeating)

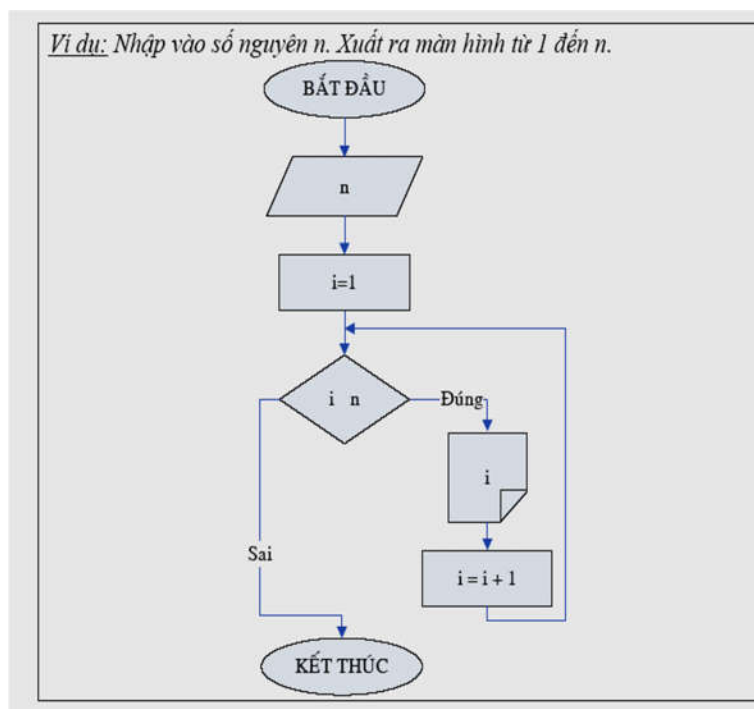
Thực hiện lặp lại một công việc không hoặc nhiều lần căn cứ vào một điều kiện nào đó. Có hai dạng như sau:

- **Lặp xác định:** là loại lặp mà khi viết chương trình, người lập trình đã xác định được công việc sẽ lặp bao nhiêu lần.
- **Lặp không xác định:** là loại lặp mà khi viết chương trình người lập trình chưa xác định được công việc sẽ lặp bao nhiêu lần. Số lần lặp sẽ được xác định khi chương trình thực thi.

Trong một số trường hợp người ta cũng có thể dùng các cấu trúc này để diễn tả một giải thuật.



Bài toán 5: Nhập vào một số nguyên n , xuất ra màn hình các số từ 1 đến n



Bước 1: nhập số n

Bước 2: xuất ra màn hình các số từ 1 đến n thì dừng.

1.3 NGÔN NGỮ LẬP TRÌNH

1.3.1 Khái niệm ngôn ngữ lập trình

Ngôn ngữ lập trình là một ngôn ngữ dùng để viết chương trình cho máy tính. Ta có thể chia ngôn ngữ lập trình thành các loại sau: ngôn ngữ máy, hợp ngữ và ngôn ngữ cấp cao.

Ngôn ngữ máy (machine language): Là các chỉ thị dưới dạng nhị phân, can thiệp trực tiếp vào trong các mạch điện tử. Chương trình được viết bằng ngôn ngữ máy thì có thể được thực hiện ngay không cần qua bước trung gian nào. Tuy nhiên chương trình viết bằng ngôn ngữ máy dễ sai sót, cồng kềnh và khó đọc, khó hiểu vì toàn những con số 0 và 1.

Hợp ngữ (assembly language): Bao gồm tên các câu lệnh và quy tắc viết các câu lệnh đó. Tên các câu lệnh bao gồm hai phần: phần mã lệnh (viết tựa tiếng Anh) chỉ phép toán cần thực hiện và địa chỉ chứa toán hạng của phép toán đó.

Ví dụ:

INPUT a ; Nhập giá trị cho a từ bàn phím
LOAD a ; Đọc giá trị a vào thanh ghi tổng
A PRINT a; Hiển thị giá trị của a ra màn hình.
INPUT b

ADD b; Cộng giá trị của thanh ghi tổng A với giá trị b

Trong các lệnh trên thì INPUT, LOAD, PRINT, ADD là các mã lệnh còn a, b là địa chỉ. Để máy thực hiện được một chương trình viết bằng hợp ngữ thì chương trình đó phải được dịch sang ngôn ngữ máy. Công cụ thực hiện việc dịch đó được gọi là Assembler.

Ngôn ngữ cấp cao (High level language): Ra đời và phát triển nhằm phản ánh cách thức người lập trình nghĩ và làm. Rất gần với ngôn ngữ con người (Anh ngữ) nhưng chính xác như ngôn ngữ toán học. Cùng với sự phát triển của các thế hệ máy tính, ngôn ngữ lập trình cấp cao cũng được phát triển rất đa dạng và phong phú, việc lập trình cho máy tính vì thế mà cũng có nhiều khuynh hướng khác nhau: lập trình cấu trúc, lập trình hướng đối tượng, lập trình logic, lập trình hàm... Một chương trình viết bằng ngôn ngữ cấp cao được gọi là chương trình nguồn (source programs). Để máy tính "hiểu" và thực hiện được các lệnh trong chương trình nguồn thì phải có một

chương trình dịch để dịch chương trình nguồn (viết bằng ngôn ngữ cấp cao) thành dạng chương trình có khả năng thực thi.

1.3.2 Chương trình dịch

Như trên đã trình bày, muốn chuyển từ chương trình nguồn sang chương trình đích phải có chương trình dịch. Thông thường mỗi một ngôn ngữ cấp cao đều có một chương trình dịch riêng nhưng chung quy lại thì có hai cách dịch: Thông dịch và biên dịch

Thông dịch (interpreter): Là cách dịch từng lệnh một, dịch tới đâu thực hiện tới đó. Chẳng hạn ngôn ngữ LISP sử dụng trình thông dịch.

Biên dịch (compiler): Dịch toàn bộ chương trình nguồn thành chương trình đích rồi sau đó mới thực hiện. Các ngôn ngữ sử dụng trình biên dịch như Pascal, C...

Giữa thông dịch và biên dịch có khác nhau ở chỗ: Do thông dịch là vừa dịch vừa thực thi chương trình còn biên dịch là dịch xong toàn bộ chương trình rồi mới thực thi nên chương trình viết bằng ngôn ngữ biên dịch thực hiện nhanh hơn chương trình viết bằng ngôn ngữ thông dịch.

Một số ngôn ngữ sử dụng kết hợp giữa thông dịch và biên dịch chẳng hạn như Java. Chương trình nguồn của Java được biên dịch tạo thành một chương trình đối tượng (một dạng mã trung gian) và khi thực hiện thì từng lệnh trong chương trình đối tượng được thông dịch thành mã máy.

1.3.3 Ngôn ngữ lập trình C

C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống cùng với Assembler và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennis Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennis Ritchie và B.W Kernighan đã cho xuất bản quyển "Ngôn ngữ lập trình C" và được phổ biến rộng rãi đến nay.

Lúc ban đầu, C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các công việc lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của công việc lập trình, C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình để rồi các công ty lập trình sử dụng một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ C và chuẩn ANSI C cũng được khai sinh từ đó.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất “mềm dẻo”, có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng khác. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp “rắc rối” là “hơi khó hiểu” do sự “mềm dẻo” của C. Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

Những đặc điểm cơ bản của ngôn ngữ C

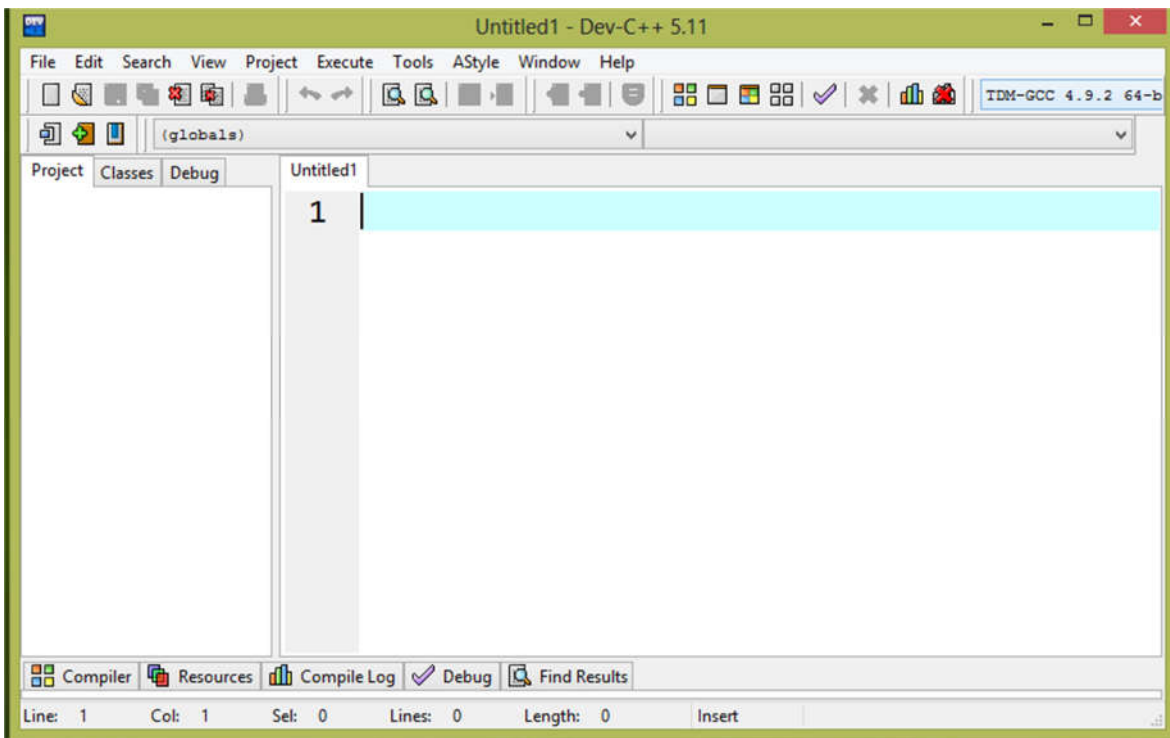
- 1. Tính cô đọng (compact):** C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.
- 2. Tính cấu trúc (structured):** C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp... Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.
- 3. Tính tương thích (compatible):** C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.
- 4. Tính linh động (flexible):** C là một ngôn ngữ rất uyển chuyển và cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
- 5. Biên dịch (compile):** C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

1.4 MÔI TRƯỜNG DEV-C

Dev-C là phần mềm cơ bản để học lập trình C. Ngoài ra còn phần mềm khác có thể sử dụng là Turbo C, Borland C, mặc dù tương đối “thân thuộc” với những người đã từng học Pascal (vì cùng họ “Turbo”); tuy nhiên, chương trình này trên giao diện Console không thân thiện cho lắm và đồng thời sẽ gặp trục trặc với những hệ điều hành 64-bit. Chương trình Dev-C++ hỗ trợ đầy đủ các chức năng như: soạn thảo chương trình, dịch, thực thi chương trình... Phiên bản được sử dụng ở đây là Dev-C 5.11.

❖ Mở Dev-C

Chạy Dev-C cũng giống như chạy các chương trình khác trong môi trường Windows, màn hình sẽ xuất hiện cửa sổ Dev-C như sau:



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.

❖ Soạn thảo chương trình mới

Muốn soạn thảo một chương trình mới ta chọn **File** → **New** → **Source File**. 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:

❖ Quy tắc đặt tên tập tin của ngôn ngữ C:

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: .CPP** phần này dài tối đa 3 ký tự.

Ví dụ: bai_tap.cpp

❖ Thực hiện chương trình:

Sau khi chúng ta viết hoàn thiện một chương trình, nếu muốn chạy thử và xem kết quả của chương trình hãy nhấn tổ hợp phím **F11**.

CÂU HỎI ÔN TẬP

Mục đích yêu cầu

Làm quen và nắm vững các cách mô tả giải thuật; từ đó đứng trước một bài toán cụ thể, sinh viên có thể mô tả thật chi tiết các bước để giải quyết vấn đề.

Lưu ý: Sinh viên cần xác định:

Input (dữ liệu nhập).

Output (dữ liệu xuất).

Tìm thuật toán phù hợp.

Bằng ngôn ngữ tự nhiên và lưu đồ, anh (chị) hãy mô tả giải thuật cho các bài toán sau:

Câu 1: Giải và biện luận phương trình bậc nhất $ax + b = 0$.

Câu 2: Giải và biện luận phương trình bậc 2 $ax^2 + bx + c = 0$.

Câu 3: Tính tổng của n số tự nhiên đầu tiên $S = 1 + 2 + 3 + \dots + n$

Câu 4: Tính tổng của n số chẵn tự nhiên đầu tiên $S = 2 + 4 + \dots + 2n$

Câu 5: Liệt kê tất cả các ước số của số nguyên dương N.

Ví dụ: $N=12$

Kết quả: Các ước số của 12 là: 1 2 3 4 6 12

Câu 1: Đếm các ước của một số nguyên dương N.

Ví dụ: $N=12$

Kết quả: Số ước số của 12 là: 6

Câu 2: Vẽ lưu đồ tính tổng các ước số của số nguyên dương N.

Ví dụ: $N=30$

Kết quả: Tổng các ước số của 30 là: 72.

Câu 3: Cho số tự nhiên n (n khai báo kiểu unsigned long)

a. Số tự nhiên n có bao nhiêu chữ số.

- b. Hãy tìm chữ số cuối cùng của n .
- c. Hãy tìm chữ số đầu tiên của n .
- d. Tính tổng các chữ số của n .
- e. Hãy tìm số đảo ngược của số n .

Ví dụ: $n=64376$

64376 có 5 chữ số

Chữ số cuối cùng là 6

Chữ số đầu tiên là 6

Tổng các chữ số là 26

Số đảo ngược là 67346

Câu 4: Viết chương trình kiểm tra một số có là số nguyên tố hay không.

Câu 5: Vẽ lưu đồ thuật toán tính: $S = 1^2 + 2^2 + \dots + n^2$

Câu 6: Vẽ lưu đồ thuật toán tính: $S = 1 + \frac{1}{2} + \dots + \frac{1}{n}$

Câu 7: Vẽ lưu đồ thuật toán tính: $S = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \dots + \frac{n}{n+1}$

Câu 8: Vẽ lưu đồ thuật toán tính: $T = 1 \times 2 \times 3 \times \dots \times n$

Câu 9: Vẽ lưu đồ thuật toán tính: $S = 1! + 2! + \dots + n!$

BÀI 2: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C

Học xong bài này, sinh viên sẽ nắm rõ các vấn đề sau:

- Các ký tự và từ khóa dùng trong C.
- Cách đặt tên trong C
- Khai báo và sử dụng các thư viện trong C
- Các kiểu dữ liệu chuẩn (*int, long, char, float...*).
- Các phép toán và các hàm chuẩn của ngôn ngữ lập trình C.
- Câu lệnh là gì? Cách sử dụng câu lệnh gán giá trị của một biểu thức cho một biến.
- Cách sử dụng lệnh *scanf* để nhập giá trị cho biến.
- Cách sử dụng lệnh *printf* để xuất giá trị của biểu thức lên màn hình.
- Thực hiện viết các chương trình hoàn chỉnh sử dụng các lệnh đơn giản và các kiểu dữ liệu chuẩn đó.

2.1 CÁC KHÁI NIỆM CƠ BẢN

2.1.1 Các ký tự dùng trong ngôn ngữ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một tập ký tự nào đó. Các ký tự này được ghép với nhau để tạo thành các từ ; Sau đó các từ được liên kết lại theo một quy tắc nào đó để tạo thành một câu lệnh. Một chương trình bao gồm nhiều câu lệnh và được diễn đạt theo một thuật toán nào đó nhằm để giải quyết một bài toán cụ thể nào đó.

Tập ký tự trong ngôn ngữ C bao gồm những ký tự, ký hiệu sau: (*phân biệt chữ in hoa và in thường*):

- 26 chữ cái latin lớn A,B,C...Z
- 26 chữ cái latin nhỏ a,b,c...z.
- 10 chữ số thập phân 0,1,2...9.
- Các ký hiệu toán học: +, -, *, /, =, <, >, (,)
- Các ký hiệu đặc biệt: ., ; " ' _ @ # \$! ^ [] { } ...
- Dấu cách hay khoảng trống.

2.1.2 Các từ khoá trong C

Từ khóa được sử dụng để khai báo các kiểu dữ liệu, viết các toán tử và các câu lệnh.

asm	auto	break	case	cdecl	char
class	const	continue	_cs	default	delete
do	double	_ds	else	enum	_es
extern	_export	far	_fastcall	float	for
friend	goto	huge	if	inline	int
interrupt	_loadds	long	near	new	operator
pascal	private	protected	public	register	return
_saveregs	_seg	short	signed	sizeof	_ss
static	struct	switch	template	this	typedef
union	unsigned	virtual	void	volatile	while

Ví dụ 1: Có thể định nghĩa các kiểu riêng bằng từ khóa **typedef**.

```
#define TRUE 1
```

```
#define FALSE 0
```

```
typedef int boolean;
```

Chú ý:

- Không được sử dụng từ khóa để đặt tên cho các hằng, biến, mảng, hàm, ...
- Các từ khóa phải được viết bằng những ký tự thường.

2.1.3 Cách đặt tên trong C

Tên được dùng để xác định các thành phần khác nhau trong một chương trình. Chúng ta có các tên như: tên hằng, tên biến, tên mảng, tên cấu trúc, ...

Tên được đặt theo qui tắc: là một dãy các ký tự chữ, số và dấu gạch nối dưới, nhưng phải bắt đầu bằng một chữ cái hoặc dấu gạch nối dưới và không được trùng với từ khóa.

Ví dụ 2: Các tên đúng

```
Bien_dem
X
_a1
Delta
pt_bac_2
```

Ví dụ 3: Các tên sai

5Bien	bắt đầu bằng ký số 5
n!	sử dụng ký tự chấm than !
while	trùng với từ khóa
f(x)	sử dụng ký tự ngoặc tròn ()
del ta	sử dụng ký tự trắng

Chú ý:

- Trong ngôn ngữ C tên có phân biệt chữ thường và chữ hoa.

Ví dụ tên **AB** khác với tên **ab**, tên **Delta** khác với tên **delta**.

- Tên do người sử dụng đặt ra nên phải được đặt sao cho chúng có ý nghĩa với đối tượng mà chúng hiển thị.

2.1.4 Cặp dấu ghi chú thích

Khi viết chương trình đôi lúc ta cần phải có vài lời ghi chú về một đoạn chương trình nào đó để dễ nhớ và để điều chỉnh sau này. phần nội dung ghi chú này phải

không thuộc về chương trình (khi biên dịch phần này bị bỏ qua). Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu `/*` và `*/` hoặc `//`.

Ví dụ 4: Viết chương trình cho xuất ra màn hình chuỗi ký tự "Hello, World !".

```
/* Chú thích trên nhiều dòng  
    Đây là chương trình minh họa  
*/  
#include "stdio.h" //chú thích trên 1 dòng  
#include "conio.h"  
int main() // chương trình chính  
{  
    printf("\nHello, World !"); // xuất chuỗi Hello, World! Ra màn hình  
    return 0;  
}
```

2.2 CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH C

2.2.1 Tiền xử lý và biên dịch

Trong C, việc dịch (translation) một tập tin nguồn được tiến hành trên hai bước hoàn toàn độc lập với nhau:

- Tiền xử lý.
- Biên dịch.

Hai bước này trong phần lớn thời gian được nối tiếp với nhau một cách tự động theo cách thức mà ta có ấn tượng rằng nó đã được thực hiện như là một xử lý duy nhất. Nói chung, ta thường nói đến việc tồn tại của một bộ tiền xử lý (preprocessor?) nhằm chỉ rõ chương trình thực hiện việc xử lý trước. Ngược lại, các thuật ngữ trình biên dịch hay sự biên dịch vẫn còn nhập nhằng bởi vì nó chỉ ra khi thì toàn bộ hai giai đoạn, khi thì lại là giai đoạn thứ hai.

Bước tiền xử lý tương ứng với việc cập nhật trong văn bản của chương trình nguồn, chủ yếu dựa trên việc diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor);

Các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu (symbol) #.

Hai chỉ thị quan trọng nhất là:

- Chỉ thị sự gộp vào của các tập tin nguồn khác: #include
- Chỉ thị việc định nghĩa các macros hoặc ký hiệu: #define

Chỉ thị đầu tiên được sử dụng trước hết là nhằm gộp vào nội dung của các tập tin cần có (header file), không thể thiếu trong việc sử dụng một cách tốt nhất các hàm của thư viện chuẩn, phổ biến nhất là:

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

Chỉ thị thứ hai rất hay được sử dụng trong các tập tin thư viện (header file) đã được định nghĩa trước đó và thường được khai thác bởi các lập trình viên trong việc định nghĩa các ký hiệu như là:

```
#define NB_COUPS_MAX 100
```

```
#define SIZE 25
```

2.2.2 Cấu trúc một chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, khai báo biến ngoài, các hàm tự tạo, chương trình chính (hàm main).

Cấu trúc có thể như sau:

Các chỉ thị tiền xử lý (Preprocessor directives)

```
#include <Tên tập tin thư viện>
```

```
#define ....
```

Định nghĩa kiểu dữ liệu (phần này không bắt buộc): dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gọi nhớ hay đặt 1 kiểu dữ liệu cho riêng mình dựa trên các kiểu dữ liệu đã có.

Cú pháp: typedef <Tên kiểu cũ> <Tên kiểu mới>

Ví dụ: typedef int SoNguyen ; // Kiểu SoNguyen là kiểu int

Khai báo các prototype (tên hàm, các tham số, kiểu kết quả trả về,... của các hàm (sẽ cài đặt trong phần sau, phần này không bắt buộc):

Phần khai báo này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm.

Khai báo các biến ngoài (các biến toàn cục) *phần này không bắt buộc*: phần này khai báo các biến toàn cục được sử dụng trong cả chương trình.

Chương trình chính phần này bắt buộc phải có

<Kiểu dữ liệu trả về> main()

{

Các khai báo cục bộ trong hàm main: Các khai báo này chỉ tồn tại trong hàm mà thôi, có thể là khai báo biến hay khai báo kiểu.

Các câu lệnh dùng để định nghĩa hàm main

return <kết quả trả về>; // Hàm phải trả về kết quả

Cài đặt các hàm

<Kiểu dữ liệu trả về> Tên hàm (các tham số)

{

Các khai báo cục bộ trong hàm.

Các câu lệnh dùng để định nghĩa hàm return <kết quả trả về>;

}

Một chương trình C bắt đầu thực thi từ hàm main (thông thường là từ câu lệnh đầu tiên đến câu lệnh cuối cùng).

2.2.3 Các tập tin thư viện thông dụng

Đây là các tập tin chứa các hàm thông dụng khi lập trình C, muốn sử dụng các hàm trong các tập tin header này thì phải khai báo `#include <Tên tập tin>` ở phần đầu của chương trình

1. `stdio.h`: Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output). Gồm các hàm in dữ liệu (`printf()`), nhập giá trị cho biến (`scanf()`), nhận ký tự từ bàn phím (`getc()`), in ký tự ra màn hình (`putc()`), nhận một dãy ký tự từ bàn phím (`gets()`), in chuỗi ký tự ra màn hình (`puts()`), xóa vùng đệm bàn phím (`fflush()`), `fopen()`, `fclose()`, `fread()`, `fwrite()`, `getchar()`, `putchar()`, `getw()`, `putw()`...

2. **conio.h**: Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console).
Gồm các hàm clrscr(), , getch(), getpass(), cgets(), cputs(), putchar(), clrscr(),...
 3. **math.h**: Tập tin định nghĩa các hàm tính toán gồm các hàm abs(), sqrt(), log(), log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),...
 4. **alloc.h**: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(),...
 5. **io.h**: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),...
 6. **graphics.h**: Tập tin định nghĩa các hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), ...
- Còn nhiều tập tin khác nữa.

2.2.4 Cú pháp khai báo các phần bên trong một chương trình C

❖ Chỉ thị #include để sử dụng tập tin thư viện.

Cú pháp:

#include <Tên tập tin> // Tên tập tin được đặt trong dấu <>

hay **#include "Tên đường dẫn"**

Menu Option của Borland C có mục INCLUDE DIRECTORIES, mục này dùng để chỉ định các tập tin thư viện được lưu trữ trong thư mục nào.

Nếu ta dùng #include<Tên tập tin> thì Borland C sẽ tìm tập tin thư viện trong thư mục đã được xác định trong INCLUDE DIRECTORIES.

Ví dụ: include <stdio.h>

Nếu ta dùng #include"Tên đường dẫn" thì ta phải chỉ rõ tên ở đâu, tên thư mục và tập tin thư viện.

Ví dụ: #include"C:\\TC\\math.h"

Trong trường hợp tập tin thư viện nằm trong thư mục hiện hành thì ta chỉ cần đưa tên tập tin thư viện

Ví dụ:

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

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

```
#include "math.h"
```

❖ Chỉ thị **#define** để định nghĩa hằng số

Cú pháp:

```
#define <Tên hằng> <Giá trị>
```

Ví dụ:

```
#define pi 3.14
```

❖ Khai báo các prototype của hàm

Cú pháp:

```
<Kiểu kết quả trả về> Tên hàm (danh sách đối số)
```

Ví dụ:

```
long giaithua (int n); //Hàm tính giai thừa của số nguyên n
```

```
double x_mu_y (float x, float y); /*Hàm tính x mũ y*/
```

2.2.5 Cấu trúc một chương trình C đơn giản

```
//Khai báo thư viện
```

```
#include<tên tập tin thư viện>
```

```
//Định nghĩa hằng (không bắt buộc)
```

```
//Chương trình chính
```

```
int main()
```

```
{
```

```
    Khai báo biến – nếu có (chỉ tồn tại trong hàm);
```

```
    Các câu lệnh dùng để định nghĩa hàm main;
```

```
    return 0;
```

```
}
```

Một vài chương trình C đơn giản

Ví dụ 11: Viết chương trình nhập vào một số thực x , sau đó tính giá trị của biểu thức $(x*x - 5*x + 4)$ và xuất kết quả ra màn hình.

```
#include "stdio.h"
int main()
{
    float x, y;
    printf("\nNhap x = ");
    scanf("%f",&x);
    y= x*x-5*x+4;
    printf("Gia tri bieu thuc: %f", y);
}
```

Ví dụ 12: Viết chương trình nhập vào một chuỗi ký tự, sau đó xuất chuỗi ký tự đó ra màn hình.

```
#include "stdio.h"
int main()
{
    char ChuoiKyTu[50];
    printf("\nNhap chuoi ky tu: ");
    gets(ChuoiKyTu);
    printf("Chuoi da nhap: %s",ChuoiKyTu);
}
```

2.3 KIỂU DỮ LIỆU CƠ BẢN, BIẾN, HẲNG

2.3.1 Các kiểu dữ liệu cơ bản

❖ Kiểu kí tự

Kiểu char chiếm 1 byte và được biểu diễn thông qua bảng mã ASCII.

Có hai kiểu char:

Kiểu dữ liệu	Miền giá trị (Domain)
unsigned char	Từ 0 đến 255 (tương đương 256 ký tự trong bảng mã ASCII)
char	Từ - 128 đến 127

❖ Kiểu số nguyên

Kiểu dữ liệu	Kích thước	Miền giá trị (Domain)
char	1 byte	-128 đến 127 hoặc 0 đến 255
unsigned char	1 byte	0 đến 255
signed char	1 byte	-128 đến 127
int	2 bytes	-32,768 đến 32,767
unsigned int	2 bytes	0 đến 65,535
long	4 bytes	-2,147,483,648 đến 2,147,483,647
unsigned long	4 bytes	0 đến 4,294,967,295

❖ **Kiểu số thực:** Kiểu số thực dùng để lưu các số thực hay các số có dấu chấm thập phân gồm có 3 kiểu sau:

Kiểu dữ liệu	Kích thước (Size)	Miền giá trị (Domain)
float	4 bytes	Từ $3.4 * 10^{-38}$ đến $3.4 * 10^{38}$
double	8 bytes	Từ $1.7 * 10^{-30}$ đến $1.7 * 10^{308}$
long double	10 bytes	Từ $3.4 * 10^{-4932}$ đến $1.1 * 10^{4932}$

Ngoài ra ta còn có kiểu dữ liệu **void**, kiểu này mang ý nghĩa là kiểu rỗng không chứa giá trị gì cả.

2.3.2 Hằng (Constant)

Là đại lượng không đổi trong suốt quá trình thực thi của chương trình.

Hằng có thể là một chuỗi ký tự, một ký tự, một con số xác định. Chúng có thể được biểu diễn hay định dạng (Format) với nhiều dạng thức khác nhau.

❖ Khai báo hằng

Cú pháp: `#define <ten hang> <gia tri hang>`

Ví dụ:

```
#define MAX 253 /* khai báo một hằng MAX có giá trị là 253 */
#define PI 3.141593 /* khai báo một hằng PI có giá trị là 3.141593 */
```

❖ Hằng số thực

Số thực bao gồm các giá trị kiểu float, double, long double được thể hiện theo 2 cách sau:

- **Cách 1:** (dạng thập phân): số thực gồm 2 phần là: phần nguyên và phần dấu chấm thập phân. Ví dụ: 152.25 -3.726 275.0
- **Cách 2:** số thực gồm 2 phần là: phần định trị và phần mũ. Phần định trị là một số nguyên hoặc là số thực dạng thập phân, còn phần mũ là một số nguyên. Hai phần này cách nhau bởi chữ E hoặc chữ e.

Ví dụ:

183.21E-5	có giá trị là 183.21×10^{-5}
0.24e+6	có giá trị là 0.24×10^6
-32.7E-2	có giá trị là -32.7×10^{-2}
21e3	có giá trị là 21×10^3

❖ Hằng số nguyên

Hằng số nguyên (2 byte) hệ thập phân:

Ví dụ: 546 -27 5021

Hằng số nguyên (4 byte) long: là số nguyên được viết thêm ký tự L hoặc l vào tận cùng bên phải

Ví dụ: 564L -27456l

- **Chú ý:** nếu một hằng số nguyên vượt quá miền giá trị của int thì cũng được xem là hằng long.

Ví dụ: 123456L và 123456 là hai hằng long có cùng 1 giá trị.

❖ Hằng số nguyên hệ 8 (bát phân):

Cách viết: 0c1c2c3... Trong đó: ci là các ký tự số từ 0 đến 7.

Ví dụ: 05345 0721 là các hằng nguyên int hệ 8.

❖ **Hằng số nguyên hệ 16 (thập lục phân):**

Cách viết: 0xc1c2c3... hoặc 0Xc1c2c3...

Trong đó: ci là các ký tự từ 0 đến 9, và từ A đến F (hoặc từ a đến f)

Ví dụ: 0x79 0X5A1 0X129 0x6c8

là các hằng số nguyên hệ 16.

❖ **Hằng ký tự**

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn (''). Mỗi một ký tự tương ứng với một giá trị trong bảng mã ASCII. Hằng ký tự cũng được xem như trị số nguyên.

Ví dụ: 'a', 'A', '0', '9'

Chúng ta có thể thực hiện các phép toán số học trên 2 ký tự (thực chất là thực hiện phép toán trên giá trị ASCII của chúng)

❖ **Hằng chuỗi ký tự**

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép ("").

Ví dụ: "Ngon ngu lap trinh C", "Khoa CNTT-DHCT", "NVLinh-DVHieu"

Chú ý:

1. Một chuỗi không có nội dung "" được gọi là chuỗi rỗng.
2. Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL ('\0': mã Ascii là 0).
3. Cần phân biệt 'A' và "A". 'A' là một ký tự và được lưu trữ trong vùng nhớ 1 byte, còn "A" là một chuỗi ký tự và được lưu trữ trong vùng nhớ 2 byte.
4. Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước

Ví dụ: "I'm a Teacher" ta phải viết "I \ 'm a Teacher".

" đây là " **ngôn ngữ** "của" thì phải viết" đây là \ " **ngôn ngữ** \ " của "

2.3.3 Biến

Biến là một đại lượng được người lập trình định nghĩa và được đặt tên thông qua việc khai báo biến. Biến dùng để chứa dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này. Cách đặt tên biến giống như cách đặt tên đã nói trong phần trên.

Mỗi biến thuộc về một kiểu dữ liệu xác định và có giá trị thuộc kiểu đó.

Biến được xem như là một vùng nhớ được đặt tên. Mọi biến trước khi sử dụng cần phải được khai báo.

❖ Khai báo biến

Cú pháp: **<kiểu du lieu> <ten bien>;**

Ví dụ: `int a; /* khai báo biến a có kiểu dữ liệu là int */`
 `float x; /* khai báo biến x có kiểu dữ liệu là float */`
 `double y, z; /* khai báo 2 biến y, z có kiểu dữ liệu là double */`
 `long m, n, i; /* khai báo 3 biến m, n, i có kiểu dữ liệu là long */`

Lưu ý: Để kết thúc 1 lệnh phải có dấu chấm phẩy (;) ở cuối lệnh.

❖ Vị trí khai báo biến trong C

Trong ngôn ngữ lập trình C, ta phải khai báo biến đúng vị trí. Nếu khai báo (đặt các biến) không đúng vị trí sẽ dẫn đến những sai sót ngoài ý muốn mà người lập trình không lường trước (hiệu ứng lè). Chúng ta có 2 cách đặt vị trí của biến như sau:

❖ Khai báo biến ngoài

Các biến này được đặt bên ngoài tất cả các hàm và nó có tác dụng hay ảnh hưởng đến toàn bộ chương trình (còn gọi là biến toàn cục).

Ví dụ:

```
int i;        /*Bien ben ngoai */  
float pi;     /*Bien ben ngoai*/  
int main()  
{ ... }
```

❖ Khai báo biến trong

Các biến được đặt ở bên trong hàm, chương trình chính hay một khối lệnh. Các biến này chỉ có tác dụng hay ảnh hưởng đến hàm, chương trình hay khối lệnh chứa nó. Khi khai báo biến, phải đặt các **biến này ở đầu của khối lệnh**, trước các lệnh gán, ...

Ví dụ 1:

```
#include <stdio.h>
#include<conio.h>
int  bienngoai;      /*khai bao bien ngoai*/
intmain ()
{
    int  j,i ; /*khai bao bien ben trong chuong trinh chinh*/
    clrscr() ; / * lệnh xóa màn hình */
    i=1; j=2;
    bienngoai=3;
    printf ("\n Gia tri cua i la :  %d  ", i);
    printf ("\n Gia tri cua bienngoai la %d ", bienngoai);
    ;
    return 0;
}
```

Ví dụ 2:

```
#include <stdio.h>
#include<conio.h>
int      main ()
{
    int i, j;    /*Bien ben trong*/
    i=4;
    j=5;
    printf("\n Gia tri cua i la %d",i);
    printf("\n Gia tri cua j la %d",j);
}
```

```
    if(j>i)
    {
        int hieu = j-i ;    /* biến bên trong */
        printf (" hiệu số của j trừ i là: %d ", hieu);
    }
    else
    {
        int hieu = i - j ;    /* biến bên trong */
        printf (" hiệu số của i trừ j là: %d ", hieu);
    }
    return 0 ;
}
```

❖ **Khởi gán giá trị ban đầu cho biến:**

Ví dụ: `int a=10; /* khai báo biến a có kiểu int và được khởi gán giá trị ban đầu bằng 10 */`

`float x, y=3.1, z; /* khai báo 3 biến x, y, z có kiểu float và biến y được khởi gán giá trị ban đầu bằng 3.1 */`

❖ **Lấy địa chỉ của biến:** để lấy địa chỉ của biến ta dùng phép toán &.

Ví dụ: `float x ;`

Biểu thức **&x** trả về địa chỉ của biến x.

2.4 BIỂU THỨC, PHÉP TOÁN, CHUYỂN ĐỔI KIỂU DỮ LIỆU

2.4.1 Khái niệm biểu thức

Biểu thức là một sự kết hợp giữa các toán tử (operator) và các toán hạng (operand) theo đúng một trật tự nhất định, dùng để diễn đạt một công thức toán học nào đó.

Phép toán (toán tử) : +, -, *, /, ...

Toán hạng: hằng, biến, ...

Trong biểu thức ta có thể dùng các dấu ngoặc tròn để thể hiện đúng trình tự thực hiện của các phép toán.

Ví dụ:

Biểu thức tính tổng của n số tự nhiên đầu tiên: $n*(n+1)/2$;

Biểu thức tính nửa chu vi của một tam giác: $p=(a+b+c)/2$;

Biểu thức tính diện tích của một tam giác: $s=\text{sqrt}(p*(p-a)*(p-b)*(p-c))$;

- Mỗi biểu thức có một giá trị duy nhất.
- Hằng, biến cũng được xem là một biểu thức.

2.4.2 Các phép toán số học

❖ Phép toán 2 ngôi

Phép toán	Ý nghĩa
+	Phép cộng
-	Phép trừ
*	Phép nhân
/	Phép chia
%	Lấy phần dư

Phép toán % chỉ áp dụng cho số nguyên, nó không sử dụng được cho các số thực.

Ví dụ: biểu thức $25\%3$ có giá trị là 1

❖ Phép toán 1 ngôi

- (số âm).

Ví dụ: -3

❖ Thứ tự thực hiện các phép toán

1. Các phép toán +, - có cùng độ ưu tiên và có độ ưu tiên nhỏ hơn các phép toán *, /, %
2. Các phép toán *, /, % có cùng độ ưu tiên và có độ ưu tiên nhỏ hơn phép toán một ngôi -

3. Các phép toán số học có cùng độ ưu tiên thì được thực hiện theo thứ tự từ trái sang phải.

2.4.3 Phép gán (lệnh gán)

Lệnh gán (assignment statement) dùng để gán giá trị của một biểu thức cho một biến.

Cú pháp: `< tên biến > = < biểu thức >;`

Ví dụ:

```
int main ()
{
    int x,y;
    x = 5 ;    //Gán giá trị 5 cho biến x
    y = 2*x;   //Gán giá trị 2*x = 2*5 =10 cho biến y
    return 0;
}
```

Nguyên tắc khi dùng lệnh gán thì kiểu của biến và kiểu của biểu thức phải giống nhau, gọi là có sự tương thích giữa các kiểu dữ liệu. Chẳng hạn ví dụ sau cho thấy một sự không tương thích về kiểu:

```
int main()
{
    int x, y;
    x = 10; //Gán hằng số 10 cho biến x
    y = " hoa "; //y có kiểu int, còn " hoa " có kiểu chuỗi ký tự
    return 0;
}
```

Khi biên dịch chương trình này, C sẽ báo lỗi *"Cannot convert 'char *' to 'int'"* tức là C không thể tự động chuyển đổi kiểu từ `char *` (chuỗi ký tự) sang `int`.

Chú ý:

- Khi một biểu thức được gán cho một biến thì giá trị của nó sẽ thay thế giá trị cũ mà biến đã lưu giữ trước đó.
- Trong câu lệnh gán, dấu = là một toán tử; do đó nó có thể được sử dụng là một thành phần của biểu thức. Trong trường hợp này giá trị của biểu thức gán chính là giá trị của biến.

Ví dụ:

```
int x, y;  
y = x = 3;           // y lúc này cùng bằng 3
```

Ta có thể gán trị cho biến lúc biến được khai báo theo cách thức sau:

<Tên kiểu> <Tên biến> = <Biểu thức>;

Ví dụ: `int x = 10, y=x;`

Giả sử ta có câu lệnh gán sau: `<biến> = <biến> + <biểu thức>;`

Ta có thể viết gọn lại: `<biến> += <biểu thức>;`

Theo cách đó ta có các toán tử gán số học sau: `+=, -=, *=, /=, %=`

Ví dụ:

```
int i=5;  
i += 3;           // nghĩa là i=i+3, sau khi thực hiện câu lệnh thì i=8  
float x=1.1, y=3.0;  
y *= x+1;  
// nghĩa là y=y*(x+1), sau khi thực hiện câu lệnh thì y=6.3 và x=1.1
```

2.4.4 Các phép toán quan hệ và lý luận

❖ Các phép toán quan hệ:

Ý tưởng chính của phép toán quan hệ và toán tử Logic là đúng hoặc sai. Trong C mọi giá trị khác 0 được gọi là đúng, còn sai là 0. Các biểu thức sử dụng các toán tử quan hệ và Logic trả về 0 nếu sai và trả về 1 nếu đúng.

- Các dữ liệu thuộc kiểu dữ liệu số nguyên, số thực, ký tự có thể được so sánh với nhau bằng các phép toán quan hệ.
- Kết quả của phép toán quan hệ cho ta giá trị đúng (bằng 1) hoặc giá trị sai (bằng 0).
- Các phép toán:

Phép toán	Ý nghĩa
$>$	Lớn hơn
$>=$	Lớn hơn hay bằng
$<$	Nhỏ hơn
$<=$	Nhỏ hơn hay bằng
$==$	Bằng nhau
$!=$	Khác nhau

Ví dụ:

$5 > 7$ có giá trị 0 (sai)

$10 >= 10$ có giá trị 1 (đúng)

$6 < 6$ có giá trị 0 (sai)

$7 <= 10$ có giá trị 1 (đúng)

$12 == 8$ có giá trị 0 (sai)

$25 != 9$ có giá trị 1 (đúng)

$'A' > 'F'$ có giá trị 0 (sai)

- Các phép toán $>$, $>=$, $<$, $<=$ có cùng độ ưu tiên.
- Các phép toán $==$, $!=$ có cùng độ ưu tiên
- Các phép toán ở mục (1) có độ ưu tiên cao hơn các phép toán ở mục (2).
- Các phép toán quan hệ có độ ưu tiên thấp hơn so với các phép toán số học.

❖ Các phép toán luận lý:

Kết quả của phép toán luận lý cho ta giá trị đúng (bằng 1) hoặc giá trị sai (bằng 0).

Phép phủ định: (toán tử một ngôi) toán tử **!**

Phép và: (toán tử 2 ngôi) toán tử **&&**

Phép hoặc: (toán tử 2 ngôi) toán tử **||**

A	B	!A	A && B	A B
Đúng (khác 0)	Đúng (khác 0)	Sai (bằng 0)	Đúng (bằng 1)	Đúng (bằng 1)
Đúng (khác 0)	Sai (bằng 0)	Sai (bằng 0)	Sai (bằng 0)	Đúng (bằng 1)
Sai (bằng 0)	Đúng (khác 0)	Đúng (bằng 1)	Sai (bằng 0)	Đúng (bằng 1)
Sai (bằng 0)	Sai (bằng 0)	Đúng (bằng 1)	Sai (bằng 0)	Sai (bằng 0)

Ví dụ:

5 && 8 có giá trị 1

! 12.5 có giá trị 0

!(9 < 2) có giá trị 1

('A' == 'B') || (7 > 3) có giá trị 1

('A' > 'Y') && (6 >= 1) có giá trị 0

- Các phép toán quan hệ có độ ưu tiên nhỏ hơn phép toán **!** (phủ định), nhưng lớn hơn so với các phép toán **&&** (và), **||** (hoặc).

❖ Tóm tắt: thứ tự ưu tiên của các phép toán

1. **!, -** (số âm)
2. ***, /, %**
3. **+, -**
4. **. >, >=, <, <=**
5. **=, !=**
6. **&&, ||**

2.4.5 Chuyển đổi kiểu giá trị

❖ Cú pháp:

<(type)> <biểu thức>

Ý nghĩa: kiểu giá trị của biểu thức được đổi thành kiểu giá trị **type**

Ví dụ:

float x;

int n=5;

x= (float) n; /* giá trị của biến n có kiểu int được đổi sang kiểu float và gán cho biến x */

Phép ép kiểu có độ ưu tiên như toán tử một ngôi.

Ví dụ:

(int) 1.4*10 có giá trị 10

(int) (1.4*10) có giá trị 14

(float) 21/6 + 8.0 có giá trị 11.5

❖ Chuyển đổi kiểu trong biểu thức:

Khi 2 toán hạng trong một phép toán có kiểu giá trị khác nhau thì kiểu giá trị thấp hơn sẽ được tự động đổi sang kiểu giá trị cao hơn trước khi thực hiện phép toán. Kết quả của phép toán là một giá trị có kiểu giá trị cao nhất.

Ví dụ: Kiểu int và long thì được đổi thành kiểu long.

Kiểu int và kiểu float thì được đổi thành kiểu float.

Kiểu float và kiểu double thì được đổi thành kiểu double.

Ví dụ: Biểu thức

11/2 có giá trị 5

1.5*(11/2) có giá trị 7.5

1.5*11/2 có giá trị 8.25

21/6 + 8.0 có giá trị 11.0

❖ Chuyển đổi kiểu được thực hiện thông qua phép gán:

Kiểu giá trị của biểu thức bên vế phải được đổi sang kiểu giá trị của biến bên vế trái và đó cũng là kết quả của phép gán.

Kiểu float được đổi thành kiểu int bằng cách bỏ đi phần sau dấu chấm thập phân.

Kiểu double được đổi thành kiểu float bằng cách làm tròn.

Ví dụ:

```
int n;
```

```
n = (float) 20/6 + 8.3;           biến n có giá trị 11
```

2.4.6 Phép toán tăng giảm

❖ Phép toán tăng toán hạng lên một đơn vị: ++

- Phép toán tăng sau (a++): a được tăng lên một đơn vị sau khi giá trị của nó được sử dụng.

Ví dụ 1:

```
int n, i=3;
```

```
n = i++;
```

/* giá trị của biến i được gán cho biến n, sau đó biến i được tăng lên một đơn vị; sau khi thực hiện câu lệnh thì n=3 và i=4 */

- Phép toán tăng trước (++a): a được tăng lên một đơn vị trước khi giá trị của nó được sử dụng.

Ví dụ 2:

```
int n, i=3;
```

```
n = ++i;
```

/* giá trị của biến i được tăng lên một đơn vị, sau đó giá trị của biến i được gán cho biến n; sau khi thực hiện câu lệnh thì n=4 và i=4 */

❖ Phép toán giảm toán hạng đi một đơn vị: --

Ví dụ 1:

```
int n, i=2;
```

```
n = i--;
```

/* giá trị của biến i được gán cho biến n, sau đó biến i bị giảm đi một đơn vị; sau khi thực hiện câu lệnh thì n=2 và i=1 */

Ví dụ 2:

```
int n, i=2;
```

```
n = --i;
```

/* giá trị của biến i bị giảm đi một đơn vị, sau đó giá trị của biến i được gán cho biến n; sau khi thực hiện câu lệnh thì n=1 và i=1 */

Chú ý: Các phép toán tăng giảm một đơn vị chỉ được sử dụng với biến, mà không sử dụng được với hằng.

2.4.7 Biểu thức điều kiện

Biểu thức điều kiện có dạng:

<biểu thức 1> ? <biểu thức 2>: <biểu thức 3>

Trong đó:

- <biểu thức 1>, <biểu thức 2>, <biểu thức 3> là các biểu thức.
- Giá trị trả về của biểu thức điều kiện bằng giá trị của <biểu thức 2> nếu giá trị của <biểu thức 1> khác 0 (đúng) và bằng giá trị của <biểu thức 3> nếu giá trị của <biểu thức 1> bằng 0 (sai).
- Kiểu trả về của biểu thức điều kiện là kiểu cao nhất trong các kiểu của <biểu thức 2> và <biểu thức 3>.

Ví dụ:

```
int a;
```

```
float b;
```


$(a > b) ? a : b$

Biểu thức trả về giá trị lớn nhất trong 2 giá trị a và b .

Kiểu giá trị của biểu thức điều kiện là float.

2.4.8 Các phép toán thao tác trên bit

Để thao tác trên từng bit của một số nguyên ta có thể dùng các phép toán sau:

PHÉP TOÁN	Ý NGHĨA
&	Phép và theo bit (AND)
	Phép hoặc theo bit (OR)
^	Phép hoặc loại trừ theo bit (XOR)
<<	Phép dịch trái theo bit
>>	Phép dịch phải theo bit
~	Phép lấy phần bù theo bit

Ví dụ:

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

1 1 = 1
1 0 = 1
0 1 = 1
0 0 = 0

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

$a << n$ /*dịch trái giá trị a đi n bit*/

$a >> n$ /*dịch phải giá trị a đi n bit*/

$\sim 1 = 0$ /*phủ định của 1 là 0*/

$\sim 0 = 1$ /*phủ định của 0 là 1*/

Chú ý:

- Các phép toán thao tác trên bit chỉ dùng cho kiểu số nguyên, chứ không dùng cho kiểu số thực (float, double).
- Các phép dịch chuyển được phân thành hai loại: Phép dịch chuyển số học và phép dịch chuyển logic.
- Phép dịch chuyển số học được thực hiện trên giá trị kiểu int, và bảo toàn bit dấu.
- Phép dịch chuyển logic được thực hiện trên giá trị kiểu unsigned.

Ví dụ:

$0x5b1c \& 0x9a = 0x18$

$0x5b1c | 0x9a = 0x5b9e$

$0x5b1c \wedge 0x9a = 0x5b86$

$\sim 0x5b1c = 0xa4e3$

`int k=0xffe0;`

$k \ll 2 = 0xff80$

`unsigned int m=0x5b1c;`

$k \ll 3 = 0xd8e0$

2.4.9 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. Ta sẽ tìm hiểu kỹ hơn về con trỏ trong chương về con trỏ. Ở đây, chúng ta sẽ đề cập ngắn gọn đến hai toán tử được sử dụng để thao tác với các con trỏ.

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

Ví dụ: `m = &count`

Đặt vào biến `m` địa chỉ bộ nhớ của biến `count`.

Chẳng hạn, biến `count` ở vị trí bộ nhớ 2000, giả sử `count` có giá trị là 100. Sau câu lệnh trên `m` sẽ nhận giá trị 2000.

Toán tử thứ hai là * 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.

Ví dụ: `q = *m`

Sẽ đặt giá trị của `count` vào `q`. Bây giờ `q` sẽ có giá trị là 100 vì 100 được lưu trữ tại địa chỉ 2000.

2.4.10 Toán tử dấu phẩy (,)

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

Ví dụ: $x = (y=3, y+1);$

Trước hết gán 3 cho y 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.

CÁC VÍ DỤ:

Ví dụ 1:

Cho biết giá trị của biểu thức $5.6+2.7+20/6+8.0$

Đáp số: Giá trị của biểu thức là: 19.3

Ví dụ 2:

Cho biết int $x=10, a;$

Hãy cho biết giá trị của các biến x, a sau khi thực hiện câu lệnh:

a. $a=3* x++ + 8;$

b. $a=3* ++x + 8;$

Đáp số:

Câu a: $x=11$ và $a=38$

Câu b: $x=11$ và $a=41$

Ví dụ 3:

Hãy viết một câu lệnh để gán giá trị x cho y chỉ khi x nằm trong khoảng từ 1 đến 20. Không thay đổi y nếu x không thuộc khoảng trên.

Đáp số: Câu lệnh

$$y = (x \geq 1 \ \&\& \ x \leq 20) ? x : y;$$

2.5 NHẬP, XUẤT DỮ LIỆU

2.5.1 Lệnh nhập giá trị từ bàn phím cho biến (hàm scanf)

Là hàm cho phép đọc dữ liệu từ bàn phím và gán cho các biến trong chương trình khi chương trình thực thi. Trong ngôn ngữ C, đó là hàm scanf nằm trong thư viện stdio.h.

Cú pháp: `scanf (" mã định dạng", địa chỉ của các biến);`

Giải thích:

Mã định dạng: dùng để qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân...

Một số định dạng khi nhập kiểu số nguyên, số thực, ký tự.

Định dạng	Ý nghĩa
%d	Nhập số nguyên kiểu 2 byte
%f	Nhập số thực
%c	Nhập một ký tự
%ld	Nhập số nguyên kiểu 4 byte
%s	Nhập chuỗi ký tự

Địa chỉ của các biến: là địa chỉ (&) của các biến mà chúng ta cần nhập giá trị cho nó. Được viết như sau: **&<tên biến>**.

Ví dụ:

```
scanf ("%d",&bien1); /*Doc gia tri cho bien1 co kieu nguyen*/
```

```
scanf ("%f",&bien2); /*Doc gia tri cho bien2 co kieu thuc*/
```

```
scanf ("%d%f",&bien1,&bien2);
```

```
/*Doc gia tri cho bien1 co kieu nguyen, bien2 co kieu thuc*/
```

```
scanf ("%d%f%c",&bien1,&bien2,&bien3);
```

```
/*bien3 co kieu char*/
```

Lưu ý:

- Chuỗi định dạng phải đặt trong cặp dấu nháy kép ("").

- Các biến (địa chỉ biến) phải cách nhau bởi dấu phẩy (,).
- Có bao nhiêu biến thì phải có bấy nhiêu định dạng.
- Thứ tự của các định dạng phải phù hợp với thứ tự của các biến.
- Để nhập giá trị kiểu char được chính xác, nên dùng hàm ***fflush(stdin)*** để loại bỏ các ký tự còn nằm trong vùng đệm bàn phím trước hàm scanf().
- Để nhập vào một chuỗi ký tự (không chứa khoảng trắng hay *kết thúc bằng khoảng trắng*), chúng ta phải khai báo kiểu *mảng ký tự* hay *con trỏ ký tự*, sử dụng định dạng %s và *tên biến thay cho địa chỉ biến*.
- Để đọc vào một chuỗi ký tự có chứa khoảng trắng (*kết thúc bằng phím Enter*) thì phải dùng hàm gets().

Ví dụ:

Khai báo biến:

```
int biennguyen;
float bienthuc;
char bienchar;
char chuoi1[20], *chuoi2;
```

Nhập giá trị cho các biến:

```
scanf ("%d",&biennguyen);
scanf ("%f",&bienthuc);
scanf ("%d%f",&biennguyen, &bienthuc);
scanf ("%d%f%c",&biennguyen, &bienthuc, &bienchar) ;
```

2.5.2 Lệnh xuất giá trị của biểu thức lên màn hình (hàm printf)

Hàm printf (nằm trong thư viện **stdio.h**) dùng để xuất giá trị của các biểu thức lên màn hình.

Cú pháp: **printf("Chuỗi định dạng ", Các biểu thức);**

Giải thích:

- ***Chuỗi định dạng***: dùng để qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân...

Một số định dạng khi đối với số nguyên, số thực, ký tự.

Mã định dạng	Ý nghĩa
%d	Xuất số nguyên
%[.số chữ số thập phân] f	Xuất số thực có <số chữ số thập phân>.
%o	Xuất số nguyên hệ bát phân
%x	Xuất số nguyên hệ thập lục phân
%c	Xuất một ký tự
%s	Xuất chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	Xuất số nguyên dạng khoa học (nhân 10 mũ x)
%p	Xuất địa chỉ của biến con trỏ

Các biểu thức: là các biểu thức mà chúng ta cần xuất giá trị của nó lên màn hình, mỗi biểu thức phân cách nhau bởi dấu phẩy (,).

Ví dụ:

```
# include<stdio.h>

int main()
{
    int bien_nguyen=1234, i=65;

    float    bien_thuc=123.456703;

    printf("Gia tri nguyen cua bien nguyen =%d\n",bien_nguyen);

    printf("Gia tri thuc cua bien thuc =%f\n",bien_thuc);

        printf("Truoc khi lam tron=%f \n Sau khi lam tron=%.2f ",bien_thuc,
        bien_thuc);

    return 0;
}
```

Kết quả in ra màn hình như sau:

```
Gia tri nguyen cua bien nguyen =1234
Gia tri thuc cua bien thuc =123.456703
Truoc khi lam tron=123.456703
Sau khi lam tron=123.46
```

Nếu ta thêm vào dòng sau trong chương trình:

```
printf("\nKy tu co ma ASCII %d la %c",i,i);
```

Kết quả ta nhận được thêm:

```
Ky tu co ma ASCII 65 la A_
```

```
printf(" So nguyen la %d \n So thuc la %f ",i, (float)i);
```

```
So nguyen la 65
So thuc la 65.000000
```

```
printf("\n So thuc la %f \n So nguyen la %d",bien_thuc, (int)bien_thuc);
```

```
So thuc la 123.456703
So nguyen la 123_
```

```
printf("\n Viet binh thuong =%f \n Viet kieu khoa hoc=%e",bien_thuc, bien_thuc);
```

Kết quả in ra màn hình:

```
Viet binh thuong=123.456703
Viet kieu khoa hoc=1.234567e+02
```

Lưu ý: Đối với các ký tự điều khiển, ta không thể sử dụng cách viết thông thường để hiển thị chúng.

Ký tự điều khiển là các ký tự dùng để điều khiển các thao tác xuất, nhập dữ liệu.

Một số ký tự điều khiển được mô tả trong bảng:

Ký tự điều khiển	Giá trị thập lục phân	Ký tự được hiển thị	Ý nghĩa
\a	0x07	BEL	Phát ra tiếng chuông
\b	0x08	BS	Di chuyển con trỏ sang trái 1 ký tự và xóa ký tự bên trái (backspace)
\f	0x0C	FF	Sang trang
\n	0x0A	LF	Xuống dòng
\r	0x0D	CR	Trở về đầu dòng
\t	0x09	HT	Tab theo cột (giống gõ phím Tab)
\\	0x5C	\	Dấu \
\'	0x2C	'	Dấu nháy đơn (')
\"	0x22	"	Dấu nháy kép (")

\?	0x3F	?	Đầu chấm hỏi (?)
\ddd	Ddd	Ký tự có mã ACSII trong hệ bát phân là số ddd	
\xHHH	oxHHH	Ký tự có mã ACSII trong hệ thập lục phân là HHH	

Ví dụ:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    printf("\n Tiếng Beep \a");
    printf("\n Doi con tro sang trai 1 ky tu\b");
    printf("\n Dau Tab \tva dau backslash \\");
    printf("\n Dau nhay don \' va dau nhay kep '\"");
    printf("\n Dau cham hoi \?");
    printf("\n Ky tu co ma bat phan 101 la \101");
    printf("\n Ky tu co ma thap luc phan 41 la \x041");
    printf("\n Dong hien tai, xin go enter");
    getch();
    printf("\rVe dau dong");
    return 0;
}
```

Kết quả trước khi gõ phím Enter:

```
Tieng Beep
Doi con tro sang trai 1 ky tu
Dau Tab      va dau backslash \
Dau nhay don ' va dau nhay kep "
Dau cham hoi ?
Ky tu co ma bat phan 101 la A
Ky tu co ma thap luc phan 41 la A
Dong hien tai, xin go enter
```

Kết quả sau khi gõ phím Enter:

```
Tieng Beep
Doi con tro sang trai 1 ky tu
Dau Tab      va dau backslash \
Dau nhay don ' va dau nhay kep "
Dau cham hoi ?
Ky tu co ma bat phan 101 la A
Ky tu co ma thap luc phan 41 la A
Ve dau dongtai, xin go enter
```


CÂU HỎI ÔN TẬP

Câu 1: Cho biết chỗ sai của chương trình sau:

```
int main()
{
    n=22;
    printf("%d",n);
}
```

Câu 2: Cho biết chỗ sai của chương trình sau:

```
int main()
{
    float x;
    scanf("%f",x);
}
```

Câu 3: Hãy cho biết các biểu thức sau đây đúng hay sai; Nếu đúng hãy cho biết giá trị của biểu thức đó.

$37/(5*2)$

$37/5/2$

$37(5/2)$

$37\%(5\%2)$

$37\%5\%2$

$37-5-2$

$(37-5)2$

$37/(5*2)$

Câu 4: Giả sử $m=5$, $n=2$. Hãy cho biết giá trị của m và n sau khi thực thi mỗi câu lệnh sau:

$m *= n++ ;$

$m += --n;$

Câu 5: Liệt kê tất cả các ước số của số nguyên dương N .

Ví dụ: $N=12$

Kết quả: Các ước số của 12 là: 1 2 3 4 6 12

Câu 6: Đếm các ước của một số nguyên dương N.

Ví dụ: N=12

Kết quả: Số ước số của 12 là: 6

Câu 7: Vẽ lưu đồ tính tổng các ước số của số nguyên dương N.

Ví dụ: N=30

Kết quả: Tổng các ước số của 30 là: 72

Câu 8: Hoàn thiện các bài tập ở các bài trước.

Câu 9: Viết chương trình tính $\log_a x$ với a, x là các số thực nhập vào từ bàn phím, và $x > 0$, $a > 0$, $a \neq 1$ (dùng $\log_a x = \ln x / \ln a$).

Câu 10: Viết chương trình nhập vào tọa độ của hai điểm (x1, y1) và (x2, y2)

- Tính hệ số góc của đường thẳng đi qua hai điểm đó theo công thức: Hệ số góc = $(y_2 - y_1) / (x_2 - x_1)$
- Tính khoảng cách giữa hai điểm.

Câu 11: Viết chương trình nhập vào một ký tự:

- In ra mã Ascii của ký tự đó.
- In ra ký tự kế tiếp của nó.

Câu 12: Viết chương trình nhập vào điểm ba môn Toán, Lý, Hoá của một học sinh. In ra điểm trung bình của học sinh đó với hai số lẻ thập phân.

BÀI 3: CẤU TRÚC ĐIỀU KHIỂN

Sau khi học xong bài này, Sinh viên có thể nắm được:

- Câu lệnh là gì? Phân loại câu lệnh.
- Cấu trúc rẽ nhánh.
- Cấu trúc lựa chọn.
- Cấu trúc vòng lặp.
- Các câu lệnh "đặc biệt".

3.1 KHÁI NIỆM CÂU LỆNH VÀ KHỐI LỆNH

3.1.1 Khái niệm câu lệnh

- Một câu lệnh (statement) xác định một công việc mà chương trình phải thực hiện để xử lý dữ liệu đã được mô tả và khai báo.
- Mỗi câu lệnh có thể được viết trên một hoặc nhiều dòng và được kết thúc bằng dấu chấm phẩy (;).

3.1.2 Phân loại

Có 3 loại lệnh: lệnh đơn, khối lệnh và lệnh có cấu trúc.

- **Lệnh đơn** là một lệnh không chứa các lệnh khác. Các lệnh đơn gồm: lệnh gán (mục 2.4.3), các câu lệnh nhập xuất dữ liệu (mục 2.5), vv...
- **Khối lệnh** là một dãy các câu lệnh được đặt trong các dấu { và }.

Ví dụ: khối lệnh

```
{  
    float x;
```

```
printf("\nNhap x=");  
scanf("%f",&x);  
}
```

Lưu ý

- Không được đặt dấu chấm phẩy sau dấu ngoặc kết thúc một khối.
- Khối lệnh tương đương với một câu lệnh riêng lẻ về mặt cú pháp; Nghĩa là nơi nào đặt được một câu lệnh thì nơi đó có thể viết được bằng một khối lệnh.
- Khi khối lệnh chỉ gồm một câu lệnh thì ta có thể bỏ các dấu ngoặc ở đầu và cuối; Nghĩa là có thể xem câu lệnh là một trường hợp riêng của khối lệnh.

Một khối lệnh có thể chứa bên trong nó nhiều khối lệnh khác gọi là khối lệnh lồng nhau. Sự lồng nhau của các khối lệnh là không hạn chế.

Lưu ý về phạm vi tác động của biến trong khối lệnh lồng nhau: Trong các khối lệnh khác nhau hay các khối lệnh lồng nhau có thể khai báo các biến cùng tên.

Ví dụ 1:

```
{  
    ... lệnh;  
    {  
        int a,b; /*biến a, b trong khối lệnh thứ nhất*/  
        ... lệnh;  
    }  
    ...lệnh;  
    {  
        int a,b; /*biến a,b trong khối lệnh thứ hai*/  
        ... lệnh;  
    }  
}
```

Ví dụ 2:

```
{  
    int a, b;      /*biến a,b trong khối lệnh "bên ngoài"*/  
    ... lệnh;  
    {  
        int a,b; /*biến a,b bên trong khối lệnh con*/  
    }  
}
```

- Nếu một biến được khai báo bên ngoài khối lệnh và không trùng tên với biến bên trong khối lệnh thì nó cũng được sử dụng bên trong khối lệnh.
 - Một khối lệnh con có thể sử dụng các biến bên ngoài, Nhưng các lệnh bên ngoài không thể sử dụng các biến bên trong khối lệnh con.
- **Lệnh có cấu trúc** là lệnh trong đó chứa các lệnh khác. Lệnh có cấu trúc bao gồm: cấu trúc điều kiện rẽ nhánh, cấu trúc điều kiện lựa chọn, cấu trúc lặp và cấu trúc lệnh hợp thành. Lệnh hợp thành (khối lệnh) là một nhóm bao gồm nhiều khai báo biến và các lệnh được gom vào trong cặp dấu {}.

3.2 CÁC LỆNH CÓ CẤU TRÚC

3.2.1 Cấu trúc rẽ nhánh

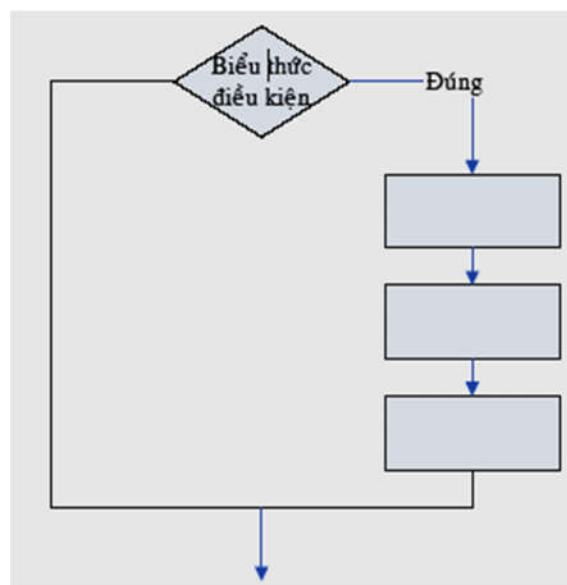
Cấu trúc rẽ nhánh là một cấu trúc được dùng rất phổ biến trong các ngôn ngữ lập trình nói chung. Trong C, có hai dạng: dạng không đầy đủ và dạng đầy đủ.

❖ Dạng không đầy đủ

Cú pháp:

```
if (<Biểu thức điều kiện>)  
{  
    <Công việc>  
}
```

Lưu đồ cú pháp:



Giải thích:

<Công việc> được thể hiện bằng 1 câu lệnh hay 1 khối lệnh.

Kiểm tra Biểu thức điều kiện trước.

Nếu điều kiện đúng ($\neq 0$) thì thực hiện câu lệnh hoặc khối lệnh liền sau điều kiện. Nếu điều kiện sai thì bỏ qua lệnh hoặc khối lệnh liền sau điều kiện (những lệnh và khối lệnh sau đó vẫn được thực hiện bình thường vì nó không phụ thuộc vào điều kiện sau if).

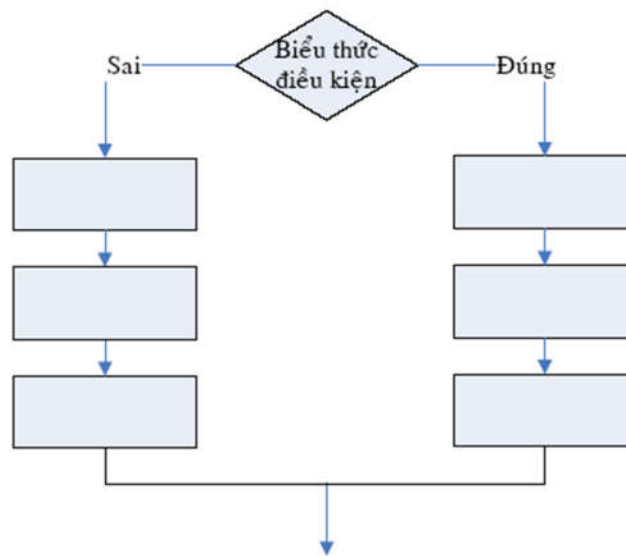
Ví dụ 1: Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi $a \neq 0$

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    float a;
    printf("Nhập a = "); scanf("%f",&a);
    if (a !=0)
        printf("Nghịch đảo của %f là %f ",a, 1/a);
    ;
    return 0;
}
```

❖ Dạng đầy đủ**Cú pháp:**

```
if (<Biểu thức điều kiện>)
    <Công việc 1>
else
    <Công việc 2 >
```

Lưu đồ:



Giải thích:

Công việc 1, công việc 2 được thể hiện là 1 câu lệnh hay 1 khối lệnh.

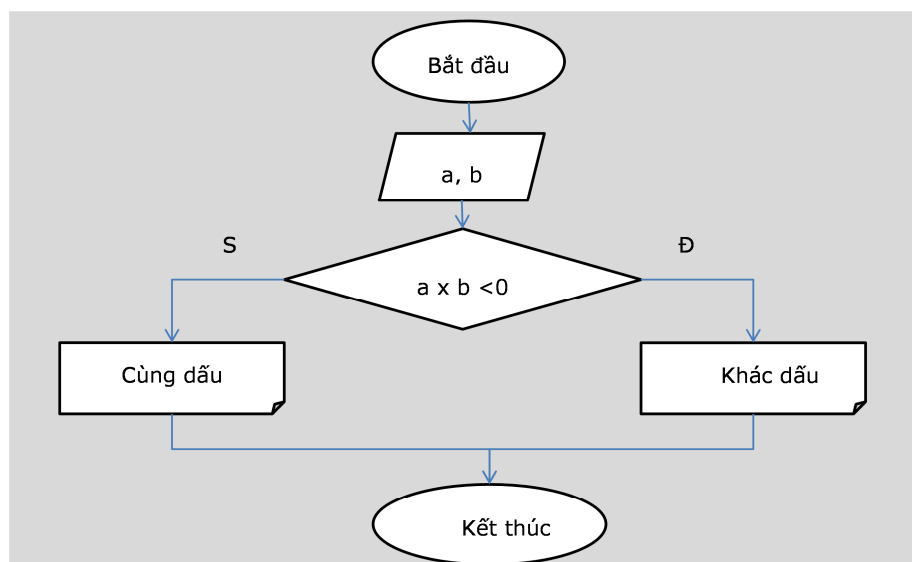
Đầu tiên *Biểu thức điều kiện* được kiểm tra trước.

Nếu điều kiện đúng thì thực hiện công việc 1.

Nếu điều kiện sai thì thực hiện công việc 2.

Các lệnh phía sau công việc 2 không phụ thuộc vào điều kiện.

Ví dụ: Viết chương trình nhập vào 2 số thực, cho biết 2 số đó cùng dấu hay khác dấu.



```
#include< stdio.h>           // thư viện phải khai báo
#include<conio.h>
int main()                   // chương trình chính
{
    float a,b;               // khai báo biến
    printf (" nhập a: "); scanf(" %f ", & a);
    printf (" nhập b: "); scanf(" %f ", & b);
    if (a*b<0)
        printf (" %f va %f khác dau ",a,b);
    else
        printf (" %f va %f cùng dau ", a,b);
    return 0;
}
```

Lưu ý:

- Ngôn ngữ C cho phép sử dụng các toán tử if lồng nhau; Nghĩa là các <khởi lệnh 1> và <khởi lệnh 2> có thể chứa các toán tử if khác.
- Nếu số từ khoá else ít hơn số từ khoá if thì else được gắn với if không có từ khoá else gần nhất trước đó.
- Ví dụ:

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

- Để chương trình rõ ràng, dễ kiểm tra, và tránh nhầm lẫn chúng ta nên viết chương trình theo các quy tắc sau:
 - Các câu lệnh và khối lệnh nằm trong một toán tử điều khiển thì viết tụt vào bên phải.
 - Các câu lệnh, khối lệnh cùng cấp thì viết trên cùng một cột.
 - Điểm đầu và điểm cuối của một khối lệnh cũng phải thẳng cột.

Ví dụ:

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

3.2.2 Cấu trúc lựa chọn switch

Cấu trúc lựa chọn cho phép lựa chọn một trong nhiều trường hợp. Trong C, đó là câu lệnh switch.

Cú pháp:

```
switch (<Biểu thức>)
{
    case giá trị 1:
        Khối lệnh thực hiện công việc 1;
        break;
    ...
    case giá trị n:
        Khối lệnh thực hiện công việc n;
        break;
    default:
        Khối lệnh thực hiện công việc mặc định;
        break;
}
```

Trong đó:

- Giá trị của biểu thức <biểu thức> là một số nguyên.
- Các <giá trị i> là các số nguyên, hằng ký tự, hoặc biểu thức hằng và cần có giá trị khác nhau.
- default là thành phần không bắt buộc.
- Trước hết giá trị của biểu thức <biểu thức> được tính, sau đó được so sánh với các giá trị <giá trị 1>, <giá trị 2>, ..., <giá trị n> và thực hiện khối lệnh tương ứng.
- Khi giá trị <biểu thức> khác tất cả các giá trị <giá trị i> và trong câu lệnh có thành phần default thì khối lệnh <khối lệnh thực hiện công việc mặc định> được thực hiện.

Ví dụ: Nhập vào một số nguyên, chia số nguyên này cho 2 lấy phần dư. Kiểm tra nếu phần dư bằng 0 thì in ra thông báo "số chẵn", nếu số dư bằng 1 thì in thông báo "số lẻ".

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int songuyen, phandu;
    printf("\n Nhap vao so nguyen "); scanf("%d",&songuyen);
    phandu=(songuyen % 2);
    switch(phandu)
    {
        case 0: printf("%d la so chan ",songuyen);
                break;
        case 1: printf("%d la so le ",songuyen);
                break;
    }
    return 0;
}
```

3.2.3 Cấu trúc lặp for

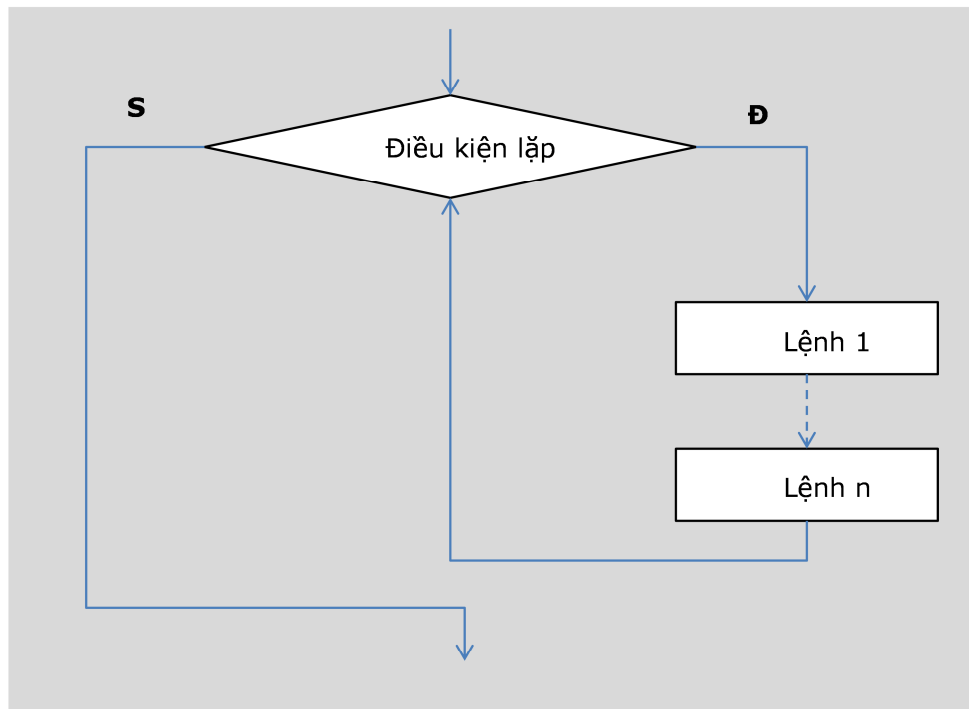
Cấu trúc vòng lặp cho phép lặp lại nhiều lần 1 công việc (được thể hiện bằng 1 câu lệnh hay 1 khối lệnh) nào đó cho đến khi thỏa mãn 1 điều kiện cụ thể.

- Vòng lặp for: cho phép lặp lại công việc cho đến khi điều kiện sai.

Cú pháp:

```
for (Biểu thức 1; biểu thức 2; biểu thức 3)
{
    <Công việc>
}
```

Lưu đồ:



Sự hoạt động của toán tử for:

Bước 1: Xác định <biểu thức 1>

Bước 2: Xác định <biểu thức 2>

Bước 3: Tùy thuộc vào giá trị của <biểu thức 2> ta có lựa chọn sau:

- Nếu <biểu thức 2> có giá trị 0 (sai) thì máy kết thúc lệnh for và thực hiện câu lệnh sau thân for.
- Nếu <biểu thức 2> có giá trị khác 0 (đúng) thì máy thực các câu lệnh trong thân for. Khi gặp dấu ngoặc đóng } cuối cùng của thân for hoặc gặp câu lệnh continue thì máy chuyển sang bước 4.

Bước 4: Tính giá trị của <biểu thức 3>, sau đó quay lại thực hiện bước 2 để bắt đầu một vòng lặp mới.

Chú ý:

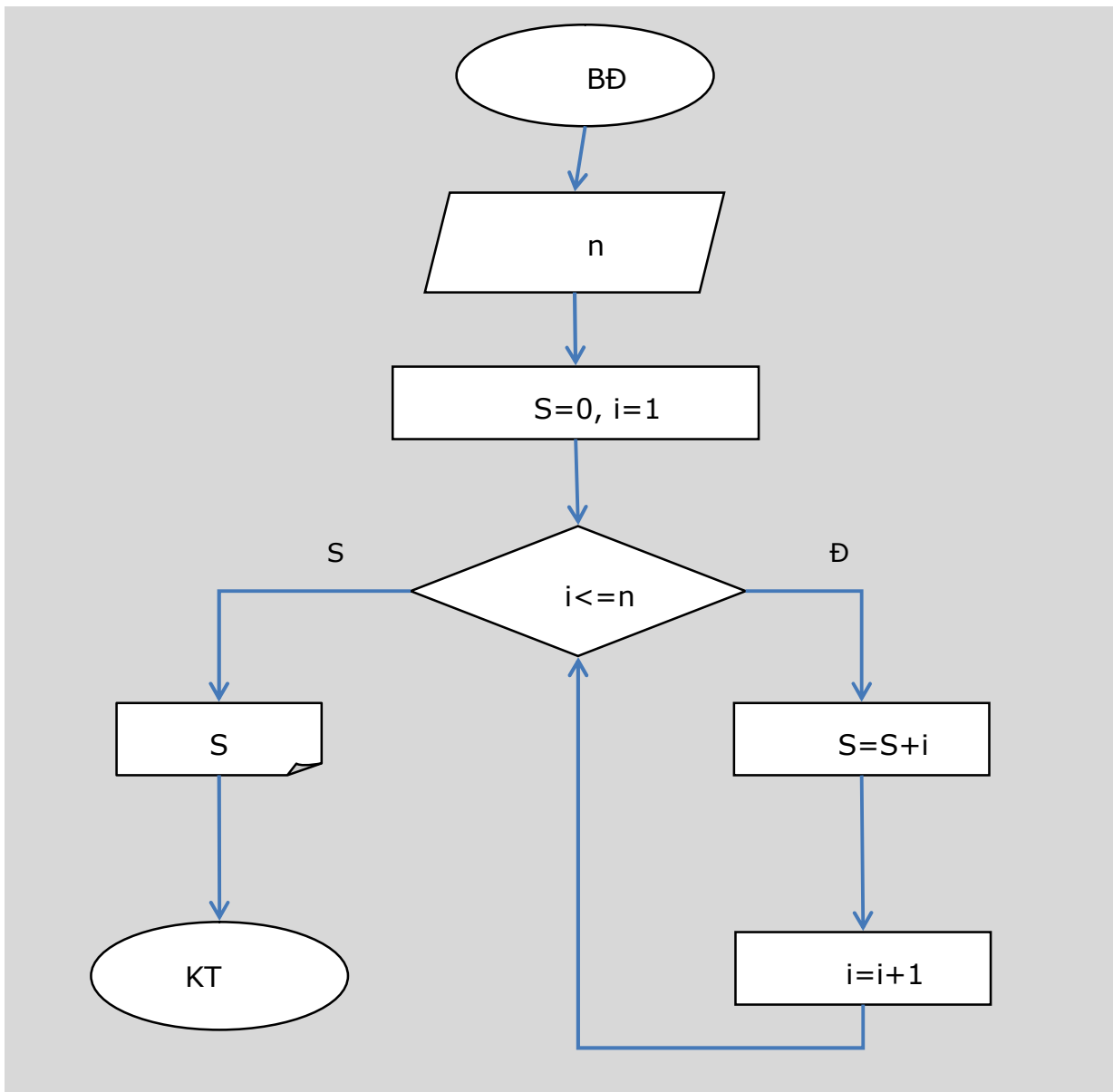
- <biểu thức 1> thông thường là một toán tử gán dùng để khởi tạo giá trị ban đầu cho các biến điều khiển và nó chỉ được tính một lần.

- <biểu thức 2> thông thường là một quan hệ logic và là điều kiện để tiếp tục một chu trình.
- <biểu thức 3> thông thường dùng để thay đổi giá trị của các biến điều khiển.
- Các <biểu thức 1>, <biểu thức 2>, <biểu thức 3> có thể vắng mặt trong câu lệnh for. Khi <biểu thức 2> vắng mặt trong câu lệnh thì nó luôn được xem là đúng; Trong trường hợp này để thoát khỏi vòng lặp for ta dùng các câu lệnh break hoặc return.
- Trong dấu ngoặc sau từ khoá for có 3 thành phần là <biểu thức 1>, <biểu thức 2>, <biểu thức 3> và các phần này ngăn cách với nhau bởi dấu chấm phẩy (;). Trong mỗi phần có thể viết một dãy biểu thức và được phân cách nhau bởi dấu phẩy (,). Khi đó giá trị của <biểu thức 2> được xác định bằng giá trị của biểu thức sau cùng của dãy biểu thức.
- Bên trong thân vòng lặp for có thể sử dụng các vòng lặp for khác.
- Khi gặp câu lệnh break trong thân vòng lặp for thì máy sẽ thoát khỏi vòng lặp for sâu nhất chứa câu lệnh này.
- Trong thân vòng lặp for ta có thể dùng câu lệnh return để kết thúc một hàm nào đó.
- Trong thân vòng lặp for ta có thể dùng câu lệnh continue để chuyển đến đầu vòng lặp mới; Nghĩa là máy sẽ bỏ qua những câu lệnh còn lại của thân vòng lặp và chuyển sang xét <biểu thức 3>.

Ví dụ 1: Viết đoạn chương trình in dãy số nguyên từ 1 đến 10.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int i;
    printf("\n Dãy số từ 1 đến 10:");
    for (i=1; i<=10; i++)
        printf("%d ",i);
    return 0;
}
```

Ví dụ 2: Viết chương trình nhập vào một số nguyên n . Tính tổng của các số nguyên từ 1 đến n



```
#include< stdio.h>
int main()
{
    int n;
    printf(" nhập vào 1 số tự nhiên n:");
    scanf("%d",&n);
    long s=0;
```

```
for (int i=1; i<=n; i++)  
{  
    s= s+i;  
}  
printf(" tổng là s= %ld", s);  
return 0;  
}
```

3.2.4 Cấu trúc lặp while

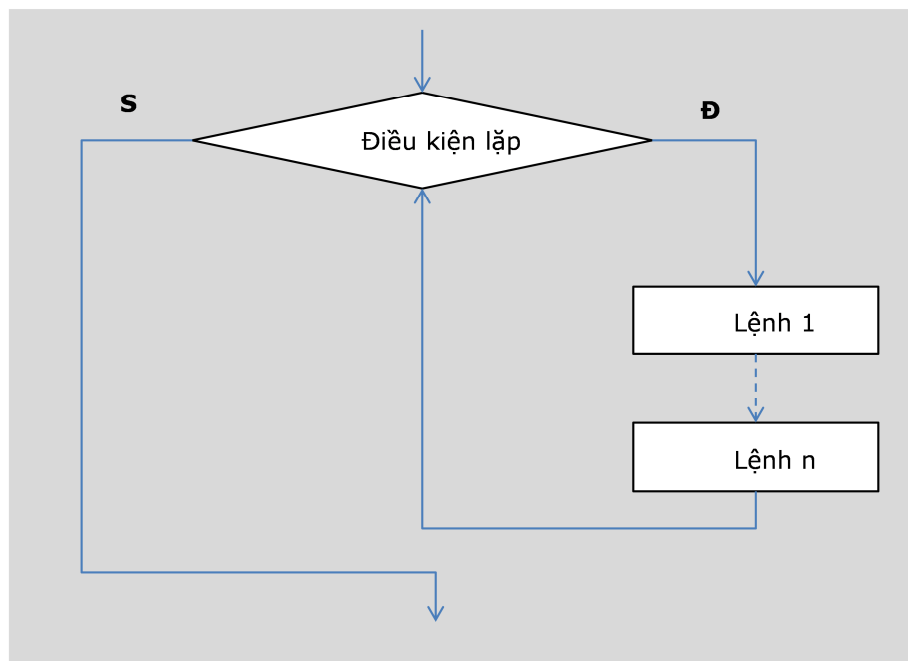
Vòng lặp while giống như vòng lặp for, dùng để lặp lại một công việc nào đó cho đến khi điều kiện sai.

Cú pháp:

while (Biểu thức điều kiện)

```
{  
    <Công việc>  
}
```

Lưu đồ:



Sự hoạt động của toán tử while:

Bước 1: Xác định giá trị của <biểu thức>

Bước 2:

- Nếu giá trị của <biểu thức> là sai (bằng 0) thì máy sẽ thoát khỏi vòng lặp và thực hiện câu lệnh đứng sau lệnh while.
- Nếu giá trị của <biểu thức> là đúng (khác 0) thì máy sẽ thực hiện các câu lệnh trong <khối lệnh>. Khi gặp dấu ngoặc đóng } cuối <khối lệnh> hoặc gặp lệnh continue thì máy trở lại bước 1.

Ví dụ: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

Lưu đồ thuật toán như của vòng for.

```
#include <stdio.h>
int main()
{
    int n;
    printf("nhập n:");
    scanf("%d",&n);
    long s=0;
    int i=1;
    while(i<=n)
    {
        s=s+i;
        i++;
    }
    printf("s= %ld", s);
    return 0;
}
```

Ví dụ 2: Viết chương trình tìm số nguyên dương nhỏ nhất n, sao cho $1+2+3+\dots+n > 10000$. Xuất ra màn hình tổng tìm được và số n.

```
#include "stdio.h"
int main()
{
    unsigned int n=0, s=0;
    while (s <= 10000)
        s += ++n;
    printf("\nSố n = %d \t Tổng: %d", n, s);
}
```

3.2.5 Vòng lặp do... while

Vòng lặp **do ... while** giống như vòng lặp for, while, dùng để lặp lại một công việc nào đó khi điều kiện còn đúng.

Trong các toán tử while và for, việc kiểm tra biểu thức điều kiện kết thúc được đặt ở đầu vòng lặp. Khác với 2 toán tử trên, toán tử **do ... while** thực hiện việc kiểm tra biểu thức điều kiện ở cuối vòng lặp.

Cú pháp:

```
do{
    <Công việc>
}while (<Biểu thức điều kiện>);
```

Sự hoạt động của toán tử do...while:

Bước 1: Thực hiện các câu lệnh trong <khối lệnh công việc>.

Bước 2: Xác định giá trị của <biểu thức điều kiện>. Tùy theo giá trị của <biểu thức điều kiện > mà máy sẽ thực hiện rẽ nhánh:

- Nếu giá trị của <biểu thức đk> là đúng (khác 0) thì máy quay lại bước 1.
- Nếu giá trị của <biểu thức đk > là sai (bằng 0) thì máy kết thúc câu lệnh **do...while**, và thực hiện câu lệnh sau câu lệnh do...while.

Ví dụ: viết chương trình nhập vào 1 số tự nhiên <100. Nếu nhập sai thì thông báo sai và nhập lại.

```
do {
    printf (" Nhập vào số phần tử ");
    scanf ("%d", &n);
    if(n<=0 || n >= 100)
        printf("Nhập sai. Nhập lại");
}while (n<=0 || n >= 100);
```

❖ So sánh các vòng lặp

Vòng lặp for, while:

- Kiểm tra điều kiện trước thực hiện công việc sau nên đoạn lệnh thực hiện công việc có thể không được thực hiện.

- Vòng lặp kết thúc khi nào điều kiện sai.

Vòng lặp do...while:

- Thực hiện công việc trước kiểm tra điều kiện sau nên đoạn lệnh thực hiện công việc được thực hiện ít nhất 1 lần.
- Vòng lặp kết thúc khi nào điều kiện sai.

3.3 CÁC CÂU LỆNH ĐẶC BIỆT

❖ **Lệnh break**

Cú pháp: **break ;**

Dùng để thoát khỏi vòng lặp. Khi gặp câu lệnh này trong vòng lặp, chương trình sẽ thoát ra khỏi vòng lặp và chỉ đến câu lệnh liền sau nó. Nếu nhiều vòng lặp thì break sẽ thoát ra khỏi vòng lặp gần nhất. Ngoài ra, break còn được dùng trong cấu trúc lựa chọn switch.

Ví dụ:

```
#include "stdio.h"
int main()
{
    int i;
    for (i=0; i<10; i++)
    {
        if (i == 5) break;
        printf("%d ",i);
    }
}
```

Chương trình trên xuất ra màn hình các số: 0 1 2 3 4

❖ **Lệnh continue**

Cú pháp: continue

- Khi gặp lệnh này trong các vòng lặp, chương trình sẽ bỏ qua phần còn lại trong vòng lặp và tiếp tục thực hiện lần lặp tiếp theo.
- Đối với lệnh for, *biểu thức 3* sẽ được tính trị và quay lại bước 2.

- Đối với lệnh while, do while; *biểu thức điều kiện* sẽ được tính và xét xem có thể tiếp tục thực hiện <Công việc> nữa hay không? (dựa vào kết quả của *biểu thức điều kiện*).

Ví dụ:

```
#include "stdio.h"
int main()
{
    int i;
    for (i=0 ;i <10; i++)
    {
        if (i%2 == 0) continue;
        printf("%d ",i);
    }
}
```

Chương trình trên xuất ra màn hình các số: 1 3 5 7 9

CÂU HỎI ÔN TẬP

Câu 1: Nhập vào 1 ký tự, nếu là chữ hoa xuất: CHUHOA, nếu là chữ thường xuất CHUTHUONG, nếu là số chẵn xuất SOCHAN và là số lẻ xuất SOLE

Câu 2: Viết chương trình nhập vào một số nguyên dương N. Tính tổng của các ký số có trong N.

Ví dụ: nhập 1234 $\rightarrow 1 + 2 + 3 + 4 = 10 \rightarrow$ in ra 10.

Câu 3: Viết chương trình cho phép nhập 1 số nguyên dương N. Xuất ra số lượng chữ số lẻ của N.

Câu 4: Viết chương trình tính tổng sau:

$$S(n) = 1 + 1.2 + 1.2.3 + + 1.2.3....n, \text{ Với } n \text{ nhập từ bàn phím.}$$

Câu 5: Viết chương trình nhập 1 số nguyên có hai chữ số. Hãy in ra cách đọc của nó.

<i>Ví dụ:</i>	Số	Cách đọc
	95	Chín mươi lăm
	11	Mười một
	31	Ba mươi mốt

Câu 6: Viết chương trình nhập 1 số nguyên có ba chữ số. Hãy in ra cách đọc của nó.

<i>Ví dụ:</i>	Số	Cách đọc
	105	Một trăm lẻ năm
	101	Một trăm lẻ một
	111	Một trăm mười một
	131	Một trăm ba mươi mốt
	145	Một trăm bốn mươi lăm

BÀI 4: CHƯƠNG TRÌNH CON

Sau khi học xong bài này, sinh viên sẽ nắm được các vấn đề sau:

- *Khái niệm về hàm (function) trong C.*
- *Cách xây dựng và cách sử dụng hàm trong C.*

4.1 KHÁI NIỆM HÀM TRONG C

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình; người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một công việc nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu chỉnh trong chương trình chính sẽ thuận lợi hơn.

Trong C, chương trình con được gọi là hàm. Hàm trong C có thể trả về kết quả thông qua tên hàm hay có thể không trả về kết quả.

Hàm có hai loại: hàm chuẩn và hàm tự định nghĩa. Trong chương này, ta chú trọng đến cách định nghĩa hàm và cách sử dụng các hàm đó.

Một hàm khi được định nghĩa thì có thể sử dụng bất cứ đâu trong chương trình. Trong C, một chương trình bắt đầu thực thi bằng hàm main.

Ví dụ 1: Ta có hàm max để tìm số lớn giữa 2 số nguyên a, b như sau:

```
int max(int a, int b)
{
    return (a>b) ? a:b;
}
```

Ví dụ 2: Ta có chương trình chính (hàm main) dùng để nhập vào 2 số nguyên a,b và in ra màn hình số lớn trong 2 số.

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b)
{
    return (a>b) ? a:b;
}

int main()
{
    int a, b, c;
    printf("\n Nhap vao 3 so a, b,c ");
    scanf("%d%d%d",&a,&b,&c);
    printf("\n So lon la %d",max(a, max(b,c)));
    return 0;
}
```

4.2 HÀM THƯ VIỆN

Hàm thư viện là những hàm đã được định nghĩa sẵn trong một thư viện nào đó, muốn sử dụng các hàm thư viện thì phải khai báo thư viện trước khi sử dụng bằng lệnh

```
#include <tên thư viện.h>
```

Ý nghĩa của một số thư viện thường dùng:

- 1. stdio.h :** Thư viện chứa các hàm vào/ ra chuẩn (**s**tandard **i**nput/**o**utput). Gồm các hàm **printf()**, **scanf()**, **getc()**, **putc()**, **gets()**, **puts()**, **fflush()**, **fopen()**, **fclose()**, **fread()**, **fwrite()**, **getchar()**, **putchar()**, **getw()**, **putw()**...
- 2. conio.h :** Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm **clrscr()**, **getche()**, **getpass()**, **cgets()**, **cputs()**, **putch()**, **clreol()**,...
- 3. math.h:** Thư viện chứa các hàm tính toán gồm các hàm **abs()**, **sqrt()**, **log()**, **log10()**, **sin()**, **cos()**, **tan()**, **acos()**, **asin()**, **atan()**, **pow()**, **exp()**,...

4. **alloc.h**: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm **calloc()**, **realloc()**, **malloc()**, **free()**, **farmalloc()**, **farcalloc()**, **farfree()**, ...
5. **io.h**: Thư viện chứa các hàm vào ra cấp thấp. Gồm các hàm **open()**, **_open()**, **read()**, **_read()**, **close()**, **_close()**, **creat()**, **_creat()**, **creatnew()**, **eof()**, **filelength()**, **lock()**,...
6. **graphics.h**: Thư viện chứa các hàm liên quan đến đồ họa. Gồm **initgraph()**, **line()**, **circle()**, **putpixel()**, **getpixel()**, **setcolor()**, ...

Muốn sử dụng các hàm thư viện thì ta phải xem cú pháp của các hàm và sử dụng theo đúng cú pháp (xem trong phần trợ giúp của borland C).

4.3 HÀM NGƯỜI DÙNG

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình.

4.3.1 Xây dựng một hàm

Cấu trúc của một hàm tự thiết kế:

```
<Kiểu kết quả> Tên hàm ([<kiểu t_ số> <tham số>] [,<kiểu t số><tham số>][...])
{
    [Khai báo biến cục bộ và các câu lệnh thực hiện hàm]
    [return [<Biểu thức>];]
}
```

Giải thích:

- **Kiểu kết quả**: là kiểu dữ liệu của kết quả trả về, có thể là: int, byte, char, float, int... 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*.
- **Kiểu t số**: là kiểu dữ liệu của tham số.
- **Tham số**: là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. **Tham số này gọi là tham số hình thức, khi gọi hàm chúng ta phải truyền cho nó các tham số thực tế**. Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm (*phần giới hạn bởi cặp dấu {}*) là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.
- Khi định nghĩa hàm, ta thường sử dụng câu lệnh **return** để trả về kết quả thông qua tên hàm.
- Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

return ; //không trả về giá trị, chỉ thoát khỏi hàm

return <biểu thức>; //Trả về giá trị của biểu thức và thoát

- Nếu hàm có kết quả trả về, ta bắt buộc phải sử dụng câu lệnh return để trả về kết quả cho hàm.

Ví dụ: Viết hàm tìm số lớn giữa 2 số nguyên a và b

```
int max(int a, int b)
{
    return (a>b) ? a:b;
}
```

4.3.2 Sử dụng hàm

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi ta có một lời gọi đến hàm đó.

Cú pháp gọi hàm: <Tên hàm>([Danh sách các tham số])

Ví dụ: Viết chương trình kiểm tra 1 số tự nhiên có phải là nguyên tố hay không.

```
int ktnt(int n)
{
    if (n<2) return 0;
    if (n==2) return 1;
    if(n %2 == 0) return 0; //n là số chẵn thì không nguyên tố
    for(int i =3; i< n/2 ; i=i+2) //nếu là số lẻ thì kiểm tra n có ước số nào không
        if (n%i ==0)
```

```
        return 0;

    return 1;
}

int main()
{
    unsigned int a;
    printf("Nhap a: "); scanf("%d",&a);
    int nt= ktnt (a);
    if (nt== 1)
        printf("%d la so nto", a);
    else
        printf("%d không phải la so nto", a);
    return 0;
}
```

4.3.3 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. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tương ứng*.
- 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 } cuối cùng trong thân 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.

4.3.4 Truyền tham số cho hàm

Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị; nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức

Ví dụ 1: Giả sử ta muốn in ra nhiều dòng, mỗi dòng 50 ký tự nào đó. Để đơn giản ta viết một hàm, nhiệm vụ của hàm này là in ra trên một dòng 50 ký tự nào đó. Hàm này có tên là InKT.


```
#include <stdio.h>
#include <conio.h>
int InKT(char ch)
{
    int i;
    for(i=1;i<=50;i++) printf("%c",ch);
    printf("\n");
}
int main()
{
    char c = 'A';
    InKT('*'); /* In ra 50 dau * */
    InKT('+');
    InKT(c);
    return 0;
}
```

Lưu ý:

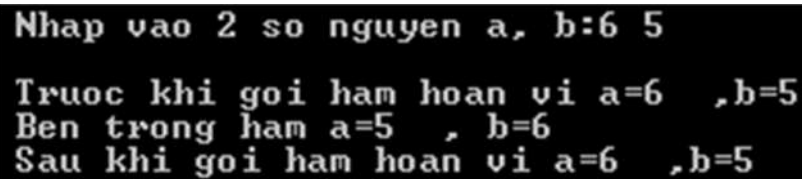
- Trong hàm InKT ở trên, biến ch gọi là tham số hình thức được truyền bằng giá trị (gọi là tham trị của hàm). Các tham trị của hàm coi như là một biến cục bộ trong hàm và chúng được sử dụng như là dữ liệu đầu vào của hàm.
- Khi chương trình con được gọi để thi hành, tham trị được cấp ô nhớ và nhận giá 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, không ảnh hưởng đến chương trình chính, nghĩa là không làm ảnh hưởng đến tham số thực tương ứng.

Ví dụ 2: Ta xét chương trình sau đây:

```
#include <stdio.h>
#include <conio.h>
int    hoanvi (int a, int b)
{
    int t;
    t=a; //Đoạn này hoán vị giá trị của 2 biến a, b
    a=b;
    b=t;
}
```

```
        printf("\Ben trong ham a=%d  , b=%d",a,b);
        return 0;
    }
    int main()
    {
        int a, b;
        printf("\n Nhap vao 2 so nguyen a, b:");
        scanf("%d%d",&a,&b);
        printf("\n Truoc khi goi ham hoan vi a=%d  ,b=%d",a,b);
        hoanvi(a,b);
        printf("\n Sau khi goi ham hoan vi a=%d ,b=%d",a,b);
        return 0;
    }
```

Kết quả thực hiện chương trình:



```
Nhap vao 2 so nguyen a, b:6 5
Truoc khi goi ham hoan vi a=6  ,b=5
Ben trong ham a=5  , b=6
Sau khi goi ham hoan vi a=6  ,b=5
```

❖ Lưu ý

Trong đoạn chương trình trên, nếu ta muốn sau khi kết thúc chương trình con giá trị của a, b thay đổi thì ta phải đặt tham số hình thức là các con trỏ, còn tham số thực tế là địa chỉ của các biến.

Lúc này mọi sự thay đổi trên vùng nhớ được quản lý bởi con trỏ là các tham số hình thức của hàm thì sẽ ảnh hưởng đến *vùng nhớ đang được quản lý bởi tham số thực tế tương ứng* (cần để ý rằng vùng nhớ này chính là các biến ta cần thay đổi giá trị).Người ta thường áp dụng cách này đối với các dữ liệu đầu ra của hàm.

CÂU HỎI ÔN TẬP

Viết dưới dạng hàm. Rồi gọi thực hiện chúng trong hàm main

Câu 1: Viết chương trình nhập 3 số từ bàn phím, tìm số lớn nhất trong 3 số đó, in kết quả lên màn hình.

Câu 2: Viết chương trình tính chu vi, diện tích của tam giác với yêu cầu sau khi nhập 3 số a, b, c phải kiểm tra lại xem a, b, c có tạo thành một tam giác không? Nếu có thì tính chu vi và diện tích. Nếu không thì in ra câu " Không tạo thành tam giác".

Câu 3: Viết chương trình giải phương trình bậc nhất $ax + b = 0$ với a, b nhập từ bàn phím.

Câu 4: Viết chương trình giải phương trình bậc hai $ax^2 + bx + c = 0$ với a, b, c nhập từ bàn phím.

Câu 5: Viết chương trình nhập từ bàn phím 2 số a, b và một ký tự ch .

Nếu: ch là "+" thì thực hiện phép tính $a + b$ và in kết quả lên màn hình.

ch là "-" thì thực hiện phép tính $a - b$ và in kết quả lên màn hình.

ch là "*" thì thực hiện phép tính $a * b$ và in kết quả lên màn hình.

ch là "/" thì thực hiện phép tính a / b và in kết quả lên màn hình.

Câu 6: Viết hàm kiểm tra 1 số nguyên có phải là số nguyên tố hay không. Áp dụng hàm này viết chương trình xuất ra các số nguyên tố nhỏ hơn N (N nhập từ bàn phím)

Câu 7: Làm lại các bài tập cũ dưới dạng hàm.

BÀI 5: MẢNG

Học xong bài này, sinh viên sẽ nắm được các vấn đề sau:

- *Khái niệm về kiểu dữ liệu mảng cũng như ứng dụng của nó.*
- *Cách khai báo biến kiểu mảng và các phép toán trên các phần tử của mảng.*
- *Đi sâu vào các giải thuật trên mảng 1 chiều và 2 chiều.*

5.1 KHÁI NIỆM

Mảng là một tập hợp các phần tử cố định có cùng một kiểu dữ liệu, gọi là kiểu phần tử. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự...; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng (trong trường hợp này ta gọi là mảng của mảng hay mảng nhiều chiều).

Ta có thể chia mảng làm 2 loại: mảng một chiều và mảng nhiều chiều.

Mảng là kiểu dữ liệu được sử dụng rất thường xuyên. Chẳng hạn, người ta cần quản lý một danh sách họ và tên của khoảng 100 sinh viên trong một lớp. Nhận thấy rằng mỗi họ và tên để lưu trữ ta cần 1 biến kiểu chuỗi, như vậy 100 họ và tên thì cần khai báo 100 biến kiểu chuỗi. Nếu khai báo như thế này thì đoạn khai báo cũng như các thao tác trên các họ tên sẽ rất dài dòng và rắc rối. Vì thế, kiểu dữ liệu mảng giúp ích ta trong trường hợp này; chỉ cần khai báo 1 biến, biến này có thể coi như là tương đương với 100 biến kiểu chuỗi ký tự; đó là 1 mảng mà các phần tử của nó là chuỗi ký tự. Hay như để lưu trữ các từ khóa của ngôn ngữ lập trình C, ta cũng dùng đến một mảng để lưu trữ chúng.

Kích thước của mảng là số phần tử của mảng. Kích thước này phải được biết ngay khi khai báo mảng.

Nếu xét dưới góc độ toán học, mảng 1 chiều giống như một vector. Mỗi phần tử của mảng một chiều có giá trị *không phải là một mảng khác*.

5.2 MẢNG MỘT CHIỀU

5.2.1 Khai báo

❖ Khai báo tường minh (số phần tử xác định):

Cú pháp: <kiểu cơ sở> <tên mảng> [<số phần tử >];

Ý nghĩa:

- **<Tên mảng>**: được đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên của biến mảng.
- **[<Số phần tử>]**: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi nó là kích thước của mảng).
- **<Kiểu cơ sở >**: là kiểu dữ liệu của mỗi phần tử của mảng.

Ví dụ:

Khai báo mảng một chiều có tên là **songuyen** gồm 10 phần tử và kiểu cơ sở là int

int songuyen [10];

Khai báo mảng một chiều có tên là **sothuc** gồm 15 phần tử và kiểu cơ sở là float

float sothuc [15];

Khai báo mảng một chiều có tên là **daykytu** gồm 30 phần tử và kiểu cơ sở là char

char daykytu [30];

❖ Khai báo không tường minh (số phần tử không xác định)

Cú pháp: <kiểu cơ sở> <tên mảng> [];

Khi khai báo, không cho biết rõ số phần tử của mảng, kiểu khai báo này thường được áp dụng trong các trường hợp: vừa khai báo vừa gán giá trị, hoặc khai báo mảng là tham số hình thức của hàm.

Cách 1. Vừa khai báo vừa gán giá trị

Cú pháp:

<Kiểu> <Tên mảng> [] = {Các giá trị cách nhau bởi dấu phẩy}

Nếu vừa khai báo vừa gán giá trị thì mặc nhiên C sẽ hiểu số phần tử của mảng là số giá trị mà chúng ta gán cho mảng trong cặp dấu {}.

Ví dụ: `float x[] = {12.1, 7.23, 5.0, 27.6, 87.9, 9.31};`

Chúng ta có thể sử dụng hàm **sizeof()** để biết số phần tử của mảng như sau:

Số phần tử = **sizeof(tên mảng) / sizeof(kiểu)**

Cách 2. Khai báo mảng là tham số hình thức của hàm, trong trường hợp này ta không cần chỉ định số phần tử của mảng là bao nhiêu.

Ví dụ: `void nhapmang (int a[], int n)`

5.2.2 Truy cập vào các phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua **Tên biến mảng** theo sau là **chỉ số** nằm trong cặp **dấu ngoặc vuông []**.

Chẳng hạn `a[0]` là phần tử đầu tiên của mảng `a` được khai báo ở trên.

Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

Với cách truy xuất theo kiểu này, **Tên biến mảng[Chỉ số]** có thể coi như là một biến có kiểu dữ liệu là **kiểu** được chỉ ra trong khai báo biến mảng.

Chỉ số của mảng có thể là một hằng, một biến hay một biểu thức đại số.

Một phần tử của mảng là một biến có kiểu dữ liệu là kiểu cơ sở nên các thao tác trên các biến cũng được áp dụng như trên các phần tử của mảng.

Ví dụ Khai báo mảng số thực có 5 phần tử **`float a [5] ;`**

Khi đó Mảng số thực trên có các phần tử là:

`a [0], a [1], a [2], a [3], a [4]` và là những biến kiểu **`float`**

Và ta có thể thực hiện các phép toán:

`float t=10.0;`

`int i=1;`

`a [0]=4.2;`

`a [2]=t;`

```

a [i ]=(a [0] +a [2])/2;

printf("\nGia tri: %f ", a[1]);

scanf("%f",&a [4]);

t= a [4];

a [2*i+1]= a [2*i]+ a [2*i+2];

```

Ta có thể khởi gán giá trị cho mảng:

```
float x[6] = {12.1, 7.23, 5.0, 27.6, 87.9, 9.31};
```

khi đó: $x[0]=12.1$, $x[1]=7.23$, $x[2]=5.0$, $x[3]=27.6$, $x[4]=87.9$, $x[5]=9.31$

5.2.3 Nhập dữ liệu cho mảng một chiều

Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên: **int a [100]** khi đó máy sẽ cấp phát 200 byte để lưu trữ mảng a

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	n-3	n-2	n-1
Giá trị a[]	7	3	9	4	5	8	12	2
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	a[n-3]	a[n-2]	a[n-1]

Nhập dữ liệu cho các phần tử

```

a [ 0 ] = 7    scanf (" %d", & a[0 ]);

a [ 1 ] = 3    scanf (" %d", & a[1 ]);

a [ 2 ] = 9    scanf (" %d", & a[2 ]);

.....

a [ n-1 ] = 2   scanf (" %d", & a[n-1 ]);

```

// Nhập từng phần tử của mảng

```

for (int i =0 ; i<n ; i++)
{

```

```
printf (" nhập a[%d]: ", i) ;
scanf (" %d ", & a[ i ]);
}
```

5.2.4 Xuất dữ liệu cho mảng một chiều

Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên: `int a [100]`, khi đó máy sẽ cấp phát 200 byte để lưu trữ mảng a.

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	n-3	n-2	n-1
Giá trị a[]	7	3	9	4	5	8	12	2
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	a[n-3]	a[n-2]	a[n-1]

Xuất dữ liệu cho từng phần tử

```
a [0] = 7    printf(" % 4d", a[0]);
a [1] = 3    printf(" % 4d", a[1]);
a [2] = 9    printf(" % 4d", a[2]);
a [3] = 4    printf(" % 4d", a[3]);

.....

a [n-1] = 2   printf(" % 4d", a[n-1]);
```

// Xuất từng phần tử của mảng

```
for (int i =0 ; i<n ; i++)
{
    printf (" % 4d ", a[ i ] ) ;
}
```

5.2.5 Một vài thuật toán trên mảng một chiều

Bài toán 1: Tính tổng các phần tử trong mảng một chiều các số nguyên.

Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên: `int a [100]`

Khi đó máy sẽ cấp phát 200 byte để lưu trữ mảng *a*

Hình ảnh mảng *a* gồm *n* phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	n-3	n-2	n-1
Giá trị <i>a[]</i>	7	3	9	4	5	8	12	2
Tên phần tử	<i>a</i> [0]	<i>a</i> [1]	<i>a</i> [2]	<i>a</i> [3]	<i>a</i> [4]	<i>a</i> [n-3]	<i>a</i> [n-2]	<i>a</i> [n-1]

Tính tổng: khai báo một biến *s* để lưu trữ tổng

$$S = a[0] + a[1] + a[2] + \dots + a[n - 1]$$

Giải thuật:

Đi từ đầu mảng đến cuối mảng

```
for (int i= 0 ; i<n ; i++)
```

Cộng dồn các phần tử *a*[*i*] vào biến *s*

```
S= S + a[i] ;
```

Hàm cài đặt

```
long tinh tong (int a[ ], int n)
```

```
{
```

```
    long s =0 ;
```

```
    for (int i = 0 ; i<n ; i++)
```

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

```
    return s;
```

```
}
```

Bài toán 2: Tìm phần tử âm đầu tiên có trong mảng một chiều các số nguyên.

Hình ảnh mảng *a* gồm *n* phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	n-3	n-2	n-1
<i>a</i> []	7	3	-9	4	5	8	12	2

Giải thuật:

Đi từ đầu mảng đến cuối mảng

```
for (int i= 0 ; i<n ; i++)
```

Kiểm tra phần tử $a[i] < 0$ đầu tiên

```
if (a[i] < 0)
```

Nếu gặp thì xuất giá trị $a[i]$ và dừng chương trình.

```
return a[i] ;
```

Hàm cài đặt

```
int amdau (int a[ ], int n)
```

```
{
```

```
    for (int i = 0 ; i < n ; i++)
```

```
        if (a[i] < 0)
```

```
            return a[i] ;
```

```
    return 1; // Tại sao lại phải thêm lệnh return 1?
```

```
}
```

5.3 CHUỖI KÝ TỰ (MẢNG MỘT CHIỀU CÁC KÝ TỰ)

Chuỗi là một dãy ký tự dùng để lưu trữ và xử lý văn bản như từ, tên, câu. Trong ngôn ngữ C không có kiểu chuỗi và chuỗi được thể hiện bằng mảng các ký tự (có kiểu cơ sở char), được kết thúc bằng ký tự `'\0'` (còn được gọi là ký tự NULL trong bảng mã Ascii).

Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép `" "`.

Chú ý: Chuỗi được khai báo là một mảng các ký tự nên các thao tác trên mảng có thể áp dụng đối với chuỗi ký tự.

5.3.1 Cách khai báo chuỗi

❖ Khai báo chuỗi: **char < tên biến> [chiều dài tối đa chuỗi];**

Ví dụ: char Hoten [20];

Khai báo như trên là khai báo 1 chuỗi chứa tối đa 19 ký tự (còn 1 ký tự cuối của chuỗi chứa NULL)

❖ Vừa khai báo vừa gán giá trị: **char <Biến> [] = <"Hằng chuỗi">;**

```
Ví dụ:      char  chuỗi [ ] = " Kỹ thuật lập trình ";
             char  chuỗi [50]= "CONG HOA XA HOI CHU NGHIA VIET NAM";
             char  name []= {'K','C','N','T','T','\0'};
             char  ten[10]={ 'h','o','a','h','o','n','g','\0'};
```

khi đó:

```
ten[0]= 'h'; ten[1]= 'o'; ten[2]= 'a'; ten[3]= 'h'; ten[4]= 'o';
ten[5]= 'n'; ten[6]= 'g'; ten[7]= '\0';
```

5.3.1.1 Lỗi khi tạo một chuỗi

1. Chú ý: Không sử dụng toán tử gán = để chép nội dung của một chuỗi này sang chuỗi khác.

```
char      a[4]="hi";
char  b[4];
b = a;      //??? Máy báo lỗi
```

2. Không dùng toán tử == để so sánh nội dung hai chuỗi

```
char a[]="hi";
char b[] = "there";
if(a==b) //??? Máy báo lỗi
{
}
```

3. Phép gán trong kiểu dữ liệu chuỗi như thế này là sai

```
char ten[10];
ten = "hoahong"
```

5.3.2 Các thao tác trên chuỗi ký tự

Cách 1: Nhập xuất chuỗi dùng thư viện < stdio.h>

❖ Nhập: scanf

Ví dụ: **scanf** ("%s", & Hoten);

Đối với hàm `scanf` khi gặp phím `space`, `tab`, `new line`, `Enter` thì dừng, cho nên chỉ dùng **hàm `scanf` để nhập chuỗi không có khoảng trắng**.

❖ Xuất:

`printf` (xuất chuỗi xong không xuống dòng)

Ví dụ `printf` ("%s", Hoten);

`printf` (xuất chuỗi xong có xuống dòng)

Ví dụ `printf` ("%s \n ", Hoten);

Cách 2: Nhập xuất chuỗi dùng thư viện `<string.h>`

❖ **Nhập:** `gets (Hoten);`

Tiếp nhận được `space`, `tab`, `new line`.

Gặp `Enter` thì dừng, phải khai báo hàm xóa bộ đệm bàn phím trước khi dùng hàm `gets`: **`fflush (stdin)`** hay **`flushall ()`**.

❖ **Xuất:** **`puts (Hoten);`** (Xuất chuỗi xong tự động xuống dòng)

5.3.3 Một số hàm xử lý chuỗi (trong `<string.h>`)

1. Hàm `kbhit`: kiểm tra bộ đệm bàn phím.

Cú pháp: `int kbhit (void)`

Hàm trả về giá trị khác không nếu bộ đệm bàn phím khác rỗng, trả về giá trị không nếu ngược lại.

2. Hàm `strcat`: dùng để nối hai chuỗi lại với nhau.

Cú pháp: `char * strcat(char* s1, char* s2)`

Hàm có công dụng ghép nối hai chuỗi `s1` và `s2` lại với nhau; kết quả ghép nối được chứa trong `s1`.

Ví dụ:

```
#include "stdio.h"
#include "string.h"
void main()
{
```

```

char s1[50],s2[50];
printf("\nNhap chuoi 1: ");    gets(s1);
printf("Nhap chuoi 2: ");    gets(s2);
strcat(s1,s2);
printf("Xuat chuoi 1: %s",s1);
printf("\nXuat chuoi 2: %s",s2);
}

```

3. Hàm strchr: Tìm lần xuất hiện đầu tiên của ký tự trong chuỗi.

Cú pháp: *char* strchr (char* ch, int kt)*

Hàm có tác dụng tìm lần xuất hiện đầu tiên của ký tự kt trong chuỗi ch. Nếu tìm thấy hàm trả về địa chỉ của ký tự được tìm thấy trong chuỗi ch, trái lại hàm trả về giá trị NULL.

Ví dụ:

```

#include "stdio.h"
#include "string.h"
void main()
{
    char s[50], ch, *p;
    printf("\nNhap chuoi: ");
    gets(s);
    printf("Nhap ky tu: ");
    ch=getche();
    p=strchr(s,ch);
    if (p != NULL) printf("\nchi so cua ky tu: %d",(int)(p-s));
    else printf("\nKhong tim thay!");
}

```

4. Hàm strcmp: so sánh hai chuỗi.

Cú pháp: *int strcmp (char* s1, char* s2)*

Hàm có công dụng so sánh hai chuỗi s1 và s2.

- Nếu hàm trả về giá trị <0 thì chuỗi s1 nhỏ hơn chuỗi s2.
- Nếu hàm trả về giá trị 0 nếu chuỗi s1 bằng chuỗi s2.

- Nếu hàm trả về giá trị >0 thì chuỗi s1 lớn hơn chuỗi s2.

5. So sánh chuỗi - Hàm stricmp()

Hàm này thực hiện việc so sánh trong n ký tự đầu tiên của 2 chuỗi s1 và s2, giữa chữ thường và chữ hoa không phân biệt.

Cú pháp: *int stricmp (const char *s1, const char *s2)*

Kết quả trả về tương tự như kết quả trả về của hàm strcmp()

6. Hàm strcpy: sao chép chuỗi.

Hàm được dùng để sao chép toàn bộ nội dung của chuỗi nguồn vào chuỗi đích.

Cú pháp: *char *strcpy (char *Des, const char *Source)*

Ví dụ:

```
#include "stdio.h"
#include "string.h"
void main()
{
    char s[50];
    strcpy(s,"Truong Dai hoc Ky thuat");
    printf("\nXuat chuoi: %s",s);
}
```

7. Hàm strncpy (): Sao chép một phần chuỗi

Hàm này cho phép chép n ký tự đầu tiên của chuỗi nguồn sang chuỗi đích.

Cú pháp: *char *strncpy(char *Des, const char *Source, size_t n)*

8. Hàm strchr(): Trích một phần chuỗi

Để trích một chuỗi con của một chuỗi ký tự bắt đầu từ một ký tự được chỉ định trong chuỗi cho đến hết chuỗi, ta sử dụng hàm strchr().

Cú pháp: *char *strchr (const char *str, int c)*

Ghi chú:

Nếu ký tự đã chỉ định không có trong chuỗi, kết quả trả về là NULL.

Kết quả trả về của hàm là một con trỏ, con trỏ này chỉ đến ký tự c được tìm thấy đầu tiên trong chuỗi str.

9. Tìm kiếm nội dung chuỗi - hàm strstr()

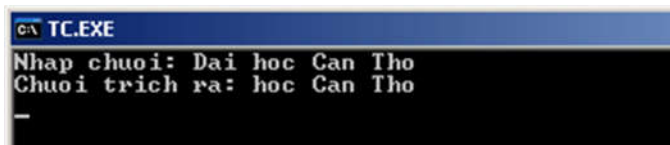
Hàm strstr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1.

Cú pháp: char *strstr(const char *s1, const char *s2)

Kết quả trả về của hàm là một con trỏ chỉ đến phần tử đầu tiên của chuỗi s1 có chứa chuỗi s2 hoặc giá trị NULL nếu chuỗi s2 không có trong chuỗi s1.

Ví dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi "hoc".

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Chuoi[255], *s ;
    printf("Nhap chuoi: ");gets(Chuoi);
    s=strstr(Chuoi,"hoc");
    -printf("Chuoi trich ra: ");puts(s);
    getch();
    return 0;
}
```



10. Lấy chiều dài chuỗi với hàm strlen()

Cú pháp: int strlen (const char *s);

```
#include <stdio.h>
#include <string.h>
void main()
```

```

{
    char string [ ] = "Borland International";
    printf("%d\n", strlen(string));    // kết quả 21
    getch ();
}

```

Ví dụ: Gán một chuỗi vào chuỗi khác

Gán từng ký tự trong chuỗi.

```

#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr() ;
    char newstr [35];
    char str[] = " Trường Đại học Công Nghệ Tp.HCM";
    for(int i = 0; i<strlen(str); i++)
        newstr[i] = str[i];
    newstr[i] = '\0';
    getch () ;
}

```

11.Đổi một ký tự thường thành ký tự hoa - Hàm toupper()

Hàm toupper() (trong ctype.h) được dùng để chuyển đổi một ký tự thường thành ký tự hoa.

Cú pháp: *char toupper (char c)*

12.Đổi chuỗi chữ thường thành chuỗi chữ hoa - Hàm strupr()

Hàm strupr() được dùng để chuyển đổi chuỗi chữ thường thành chuỗi chữ hoa, kết quả trả về của hàm là một con trỏ chỉ đến địa chỉ chuỗi được chuyển đổi.

Cú pháp: *char *strupr(char *s)*

Ví dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm `strupr()` để chuyển đổi chúng thành chuỗi chữ hoa.

```
#include <conio.h>

#include <stdio.h>

#include <string.h>

int main()
{
    char Chuoi[255], *s;
    printf("Nhap chuoi: "); gets(Chuoi);
    s=strupr(Chuoi) ;
    printf("\Chuoi chu hoa: "); puts(s);
    getch();
    return 0;
}
```

13. Đổi chuỗi chữ hoa thành chuỗi chữ thường - Hàm `strlwr()`

Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàm `strlwr()`, các tham số của hàm tương tự như hàm `strupr()`

Cú pháp: **char *strlwr (char *s)**

14. Đổi từ chuỗi ra số, hàm `atoi()`, `atof()`, `atol()` (trong `stdlib.h`)

Để chuyển đổi chuỗi ra số, ta sử dụng các hàm trên.

Cú pháp:

int atoi(const char *s): chuyển chuỗi thành số nguyên

long atol(const char *s): chuyển chuỗi thành số nguyên dài float

atof(const char *s): chuyển chuỗi thành số thực

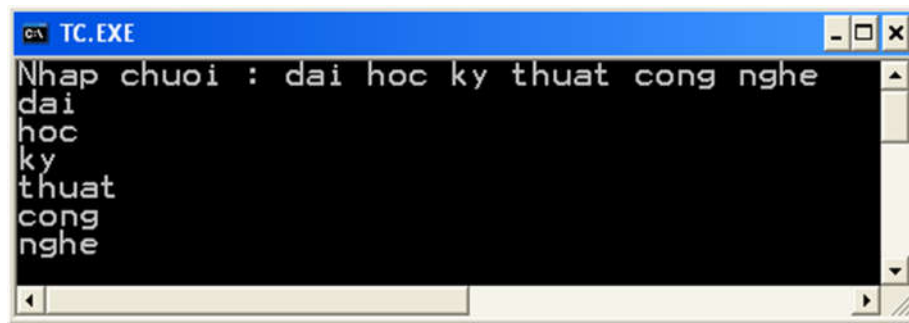
Nếu chuyển đổi không thành công, kết quả trả về của các hàm là 0.

15. **char* strtok (char *s1, const char *s2)**

Xem `s1` là 1 loạt chuỗi con, ngăn cách nhau bởi 1 hay nhiều ký tự có trong `s2`.

Ví dụ:

```
#include <string.h>
void main()
{
    char s[80], *p ;
    gets(s);
    p = strtok(s, " ");
    if (p) printf("%s", p);
    while(p)
    {
        p = strtok(NULL, " ");
        if (p) printf("%s", p);
    }
}
```

**16. Đảo ngược chuỗi: char* strrev(char *s)**

Ngoài ra, thư viện *string.h* còn hỗ trợ các hàm xử lý chuỗi khác, ta có thể đọc thêm trong phần trợ giúp.

5.4 MẢNG HAI CHIỀU

5.4.1 Khai báo

Cú pháp: **<kiểu cơ sở> <tên mảng> [<số dòng >] [<số cột >];**

Ý nghĩa:

- **Tên mảng:** Được đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên biến mảng.
- **Số dòng:** là một hằng số nguyên, cho biết số lượng dòng tối đa trong mảng là bao nhiêu.

- **Số cột:** là một hằng số nguyên, cho biết số lượng cột tối đa trong mảng là bao nhiêu.
- **Số phần tử:** là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi nó là kích thước của mảng). Số phần tử của mảng chính bằng số dòng nhân số cột.

Ví dụ:

Khai báo mảng hai chiều có tên là sn gồm 8 hàng, 14 cột và kiểu cơ sở là int

int sn [8][14];

Khai báo mảng hai chiều có tên là st gồm 10 hàng, 5 cột và kiểu cơ sở là float

float st [10][5];

5.4.2 Truy cập vào các phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua **Tên biến mảng** theo sau là **chỉ số** nằm trong cặp **dấu ngoặc vuông [][]**.

Chẳng hạn a[0][0] là phần tử đầu tiên của mảng a được khai báo ở trên.

Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

Chỉ số của mảng có thể là một hằng, một biến hay một biểu thức đại số.

Một phần tử của mảng là một biến có kiểu dữ liệu là kiểu cơ sở nên các thao tác trên các biến cũng được áp dụng trên các phần tử của mảng.

Ví dụ: Cho mảng a[4][8],

Khi đó, mảng gồm $4 \times 8 = 32$ phần tử, vị trí các phần tử của mảng theo dòng và cột như hình sau:

	0	1	2	3	4	5	6	7
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]

5.4.3 Nhập dữ liệu cho mảng hai chiều

Ví dụ minh họa trên mảng số nguyên, mảng kiểu khác thực hiện tương tự.

Khai báo hằng:

```
#define d 30 // m là số dòng
```

```
#define c 30 // n là số cột
```

Khai báo mảng số nguyên 2 chiều:

```
int a [d ][c]
```

Hình ảnh mảng *a* gồm *m* dòng *n* cột lưu trữ các số nguyên được biểu diễn như sau

	0	1	2	3	4	5	...	c-1
0	1	2	3	4	5	6	...	7
1	10	11	12	13	14	15	...	17
...								
d-1	20	21	22	23	24	25	...	27

- **Nhập dữ liệu cho dòng thứ nhất**

```
a [ 0 ][ 0 ] = 1    scanf (" %d", & a [0 ][0]);
```

```
a [ 0 ][ 1 ] = 2    scanf (" %d", & a [0 ][1]);
```

```
a [ 0 ][ 2 ] = 3    scanf (" %d", & a [0 ][2]);
```

```
.....
```

```
a [ 0 ][ c-1 ] = 2    scanf (" %d", & a[0][c-1 ]);
```

```
for (int j=0 ; j< n ; j++)
```

```
    scanf (" %d ", & a[0][j]) ;
```

- **Nhập dữ liệu cho dòng thứ hai**

```
a [ 1 ][ 0 ] = 1    scanf (" %d", & a [1][0]);
```

```
a [ 1 ][ 1 ] = 2    scanf (" %d", & a [1][1]);
```

```
a [ 1 ][ 2 ] = 3    scanf (" %d", & a [1][2]);
```

```
.....
```

```
a [ 1 ][ c-1 ] = 2    scanf ( " %d", & a[1][c-1 ] );
```

```
for (int j=0 ; j< n ; j++)
```

```
    scanf ( " %d ", & a[1][j]) ;
```

```
.....
```

- Nhập dữ liệu cho dòng thứ d

```
a [ d-1 ][ 0 ] = 1    scanf ( " %d", & a [d-1 ][0]);
```

```
a [ d-1 ][ 1 ] = 2    scanf ( " %d", & a [d-1 ][1]);
```

```
a [ d-1 ][ 2 ] = 3    scanf ( " %d", & a [d-1 ][2]);
```

```
.....
```

```
a [ d-1 ][ c-1 ] = 2    scanf ( " %d", & a[d-1][c-1 ] );
```

```
for (int j=0 ; j< c ; j++)
```

```
    scanf ( " %d ", & a[d-1][j]) ;
```

// Tóm lại, nhập từng phần tử cho ma trận các số nguyên gồm d dòng c cột

```
for (int i =0 ; i<d ; i++)
```

```
    for (int j=0; j< c ; j++)
```

```
    {
```

```
        printf ( " nhap a[%d][%d]: ", i,j) ;
```

```
        scanf ( " %d ", & a[ i ][ j ] );
```

```
    }
```

5.4.4 Xuất dữ liệu cho mảng hai chiều

Tương tự, ta có:

```
for (int i =0 ; i<d ; i++)
```

```
{
```

```
    for (int j=0; j< c ; j++)
```

```
        printf ( " %4d ", a[ i ][ j ] );
```

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

```
}
```

CÂU HỎI ÔN TẬP

Câu 1: Viết hàm nhập vào một mảng một chiều các số nguyên gồm n phần tử ($0 < n < 100$).

Câu 2: Viết hàm nhập vào một mảng một chiều các số thực gồm n phần tử ($0 < n < 100$).

Câu 3: Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên.

Câu 4: Viết hàm xuất mảng số thực n phần tử vừa nhập ở trên.

Câu 5: Tính tổng các phần tử có trong mảng.

Câu 6: Tính tổng các phần tử chẵn có trong mảng.

Câu 7: Tính tổng các phần tử lẻ có trong mảng.

Câu 8: Tính tổng các phần tử nguyên tố có trong mảng.

Câu 9: Tìm phần tử chẵn đầu tiên có trong mảng

Câu 10: Tìm phần tử lẻ đầu tiên có trong mảng

Câu 11: Tìm phần tử nguyên tố đầu tiên có trong mảng

Câu 12: Tìm phần tử chẵn cuối cùng có trong mảng

Câu 13: Tìm phần tử chính phương cuối cùng có trong mảng

Câu 14: Tìm phần tử lớn nhất có trong mảng

Câu 15: Đếm số phần tử chẵn có trong mảng

Câu 16: Đếm số phần tử có giá trị là x có trong mảng

Câu 17: Đếm số phần tử lớn nhất có trong mảng

Câu 18: In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng

Câu 19: In ra vị trí của phần tử có giá trị là x cuối cùng có trong mảng

Câu 20: Thêm một phần tử vào đầu mảng.

Câu 21: Thêm một phần tử vào cuối mảng.

Câu 22: Thêm một phần tử vào vị trí x trong mảng.

Câu 23: Xóa phần tử chẵn đầu tiên.

Câu 24: Xóa tất cả các phần tử lớn nhất trong mảng

Câu 25: Sắp xếp mảng tăng dần

Câu 26: Thêm một phần tử có giá trị là x vào trong mảng sao cho mảng vẫn có thứ tự tăng dần.

Câu 27: Đảo ngược mảng

Câu 28: Nhập chuỗi

Câu 29: Xuất chuỗi

Câu 30: Đếm số ký tự 'a' có trong chuỗi

Câu 31: Cắt khoảng trắng có trong chuỗi

Câu 32: Đếm khoảng trắng trong chuỗi.

Câu 33: Đếm số từ có trong chuỗi

Câu 34: Sắp xếp chuỗi tăng dần

Câu 35: Viết hàm nhập vào một mảng hai chiều các số nguyên gồm m dòng, n cột ($0 < m, n < 100$).

Câu 36: Viết hàm xuất mảng hai chiều các số nguyên dxc phần tử vừa nhập ở trên.

Câu 37: Tính tổng các phần tử có trong mảng.

Câu 38: Tính tổng các phần tử chẵn có trong mảng.

Câu 39: Tính tổng các phần tử nguyên tố có trong mảng.

Câu 40: Tính tổng các phần tử nằm trên đường chéo chính có trong mảng.

Câu 41: Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng.

Câu 42: Tính tổng các phần tử nằm trên đường biên có trong mảng.

Câu 43: Tìm phần tử lẻ đầu tiên có trong mảng

Câu 44: Tìm phần tử nguyên tố đầu tiên có trong mảng

Câu 45: Tìm phần tử chẵn cuối cùng có trong mảng

Câu 46: Tìm phần tử chính phương cuối cùng có trong mảng

Câu 47: Tìm phần tử lớn nhất có trong mảng

Câu 48: Đếm số phần tử chẵn có trong mảng

Câu 49: Đếm số phần tử lớn nhất có trong mảng

Câu 50: In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng

Câu 51: Tính tổng các phần tử nằm trên một dòng .

Câu 52: Tìm dòng có tổng lớn nhất.

BÀI 6: KIỂU DỮ LIỆU CÓ CẤU TRÚC

Sau khi học xong bài này, học viên có thể:

- *Nắm được các khái niệm cơ bản về kiểu dữ liệu có cấu trúc;*
- *Biết nhập, xuất dữ liệu có cấu trúc cho một phần tử;*
- *Biết nhập, xuất dữ liệu có cấu trúc và lưu trên mảng một chiều;*
- *Cách tìm kiếm và sắp xếp dữ liệu có cấu trúc trên mảng một chiều;*
- *Đi sâu vào các giải thuật trên mảng một chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử theo từng thành phần của kiểu dữ liệu có cấu trúc.*

6.1 KHÁI NIỆM

Kiểu cấu trúc (Structure) là một kiểu dữ liệu do người dùng tự định nghĩa, là kiểu dữ liệu bao gồm nhiều thành phần, mỗi thành phần có một kiểu dữ liệu khác nhau, mỗi thành phần được gọi là một trường (field).

Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng có cùng kiểu dữ liệu còn các phần tử của kiểu cấu trúc có thể có các kiểu dữ liệu khác nhau.

Hình ảnh của kiểu cấu trúc được minh họa:

1	2	3	4	5
Field	Field	Field	Field	Field

Đây là kiểu cấu trúc có 5 trường

Còn kiểu mảng có dạng

0	1	2	3	4	5	6
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

6.2 CÁCH KHAI BÁO KIỂU CẤU TRÚC

❖ Cú pháp

```

struct <Tên cấu trúc>
{
    <Kiểu> <Trường 1> ;
    <Kiểu> <Trường 2> ;
    .....
    <Kiểu> <Trường n> ;
};

typedef struct Tên cấu trúc> < tên mới >;

```

Trong đó:

1. **<Tên cấu trúc>**: là một tên được đặt theo quy tắc đặt tên của danh biểu; tên này mang ý nghĩa sẽ là tên kiểu cấu trúc.
2. **<Kiểu> <Trường i >** ($i = 1 \dots n$): mỗi trường trong cấu trúc có dữ liệu thuộc kiểu gì (tên của trường phải là một tên được đặt theo quy tắc đặt tên của danh biểu).

Ví dụ 1: Khai báo một kiểu struct có tên **NgayThang** có các thành phần là: ngày, tháng, năm, và đặt tên mới là Date.

```

struct NgayThang
{
    int ngay;
    int thang;
    int nam;
};

typedef struct NgayThang Date ;

```

Ví dụ 2: Khai báo một kiểu struct có tên **DiemThi** có các thành phần là: Mã số sinh viên, Mã số môn học, Lần thi, điểm thi.

```

struct DiemThi
{
    char masv[8];    /* mã số sinh viên */
    char mamh[5];    /* mã số môn học */
    int lanthi;      /* lần thi */
    float diem;      /* điểm thi */
};

typedef struct DiemThi Diem ;

```

Ví dụ 3: Khai báo một kiểu struct để lưu thông tin của một sinh viên bao gồm: mã số sinh viên, họ lót, tên, ngày sinh, nơi sinh, địa chỉ. Trong đó, ngày sinh là kiểu struct đã được định nghĩa trước đó.

```
typedef struct SinhVien
{
    char masv[8];
    char holot[30];
    char ten[10];
    Date ngaysinh;
    char noisinh[50];
    char diachi[50];
} SV;
```

❖ **Khai báo biến có kiểu cấu trúc**

Ví dụ: Khai báo 1 biến x để lưu thông tin của 1 sinh viên:

SV x;

6.3 TRUY CẬP VÀO TỪNG PHẦN TỬ CỦA CẤU TRÚC

❖ **Cú pháp:** Thông qua dấu chấm để truy cập đến từng thành phần của struct

<tên biến struct>.<tên thành phần của struct>

Ví dụ 1: Để xuất ra màn hình tên của sinh viên x, ta thực hiện câu lệnh sau:

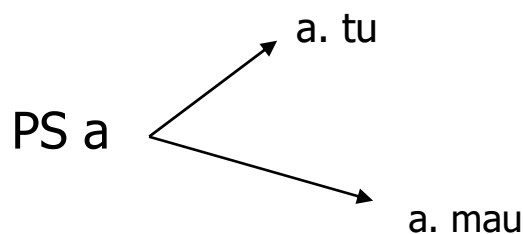
```
printf("%s", x.ten);
```

Ví dụ 2: Định nghĩa cấu trúc dữ liệu cho phân số

```
struct phanso
{
    int tu;
    int mau ;
};

typedef struct phanso PS;
```

Khi đó Nếu ta khai báo một biến `a` có kiểu dữ liệu là phân số thì `a` gồm 2 thành phần: `a.tu`, `a.mau`.



6.4 NHẬP, XUẤT DỮ LIỆU CHO KIỂU DỮ LIỆU CÓ CẤU TRÚC

6.4.1 Nhập dữ liệu

Để nhập dữ liệu cho biến kiểu cấu trúc, ta phải nhập cho từng thành phần của cấu trúc.

Ví dụ: Nhập dữ liệu cho một phân số có cấu trúc khai báo như trên:

```
void nhapphanso (PS &x)
{
    printf("Nhap tu so:");
    scanf("%d",&x.tu);
    do{
        printf("Nhap mau so khac 0:");
        scanf("%d",&x.mau);
    }while (x.mau ==0) ;
}
```

6.4.2 Xuất dữ liệu

Để xuất dữ liệu cho biến cấu trúc, ta xuất dữ liệu trong từng thành phần của cấu trúc.

Ví dụ: Xuất dữ liệu cho một phân số:

```
void xuatphanso (PS x) // VD 3/5
{
    printf ("phan so:%d / %d ", x.tu, x.mau);
}
```

6.5 MẢNG CẤU TRÚC

Bài toán minh họa: Viết chương trình

1. Nhập vào một dãy phân số.
2. Xuất ra dãy phân số vừa nhập.
3. Tính tổng các phân số có trong dãy.
4. Tìm phân số lớn nhất có trong dãy.
5. Đếm số phần tử lớn nhất có trong dãy.

Hình ảnh mảng một chiều các phân số

	0	1	2	3	4	5	6	7	8
a[]	9/1	6/2	1/2	5/3	1/7	1/3	1/2	8/2	3/2

Trong đó

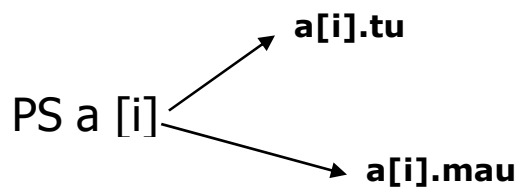
a[0] = 9/1

a[1] = 6/2

a[2] = 1/2

.....

a[8] = 3/2



Nhập mảng phân số

// hàm cài đặt

```
void nhapmang (PS a[], int n)
{
    for (int i=0 ; i<n ; i++)
        nhapphanso(a[i]);
}
```

Xuất mảng phân số

// hàm cài đặt

```
void xuatmang (PS a[], int n)
{
    for (int i=0 ; i<n ; i++)
        xuatphanso(a[i]);
}
```

CÂU HỎI ÔN TẬP

Câu 1: Viết hàm Nhập vào một phân số

Câu 2: Viết hàm Nhập vào một dãy phân số

Câu 3: Viết hàm xuất một phân số

Câu 4: Viết hàm xuất một dãy phân số

Câu 5: Viết hàm tìm phân số lớn nhất trong dãy phân số

Câu 6: Viết hàm tính tổng các phân số có trong dãy

Câu 7: Viết hàm nhập dữ liệu cho một sinh viên, thông tin về một sinh viên gồm có:

1. Mã số sinh viên (chuỗi 10 ký tự).
2. Họ (là chuỗi tối đa 20 ký tự).
3. Tên (là chuỗi tối đa 10 ký tự).
4. Ngày tháng năm sinh (theo kiểu ngày tháng năm).
5. Giới tính (Nam hoặc Nữ).
6. Lớp (chuỗi 7 ký tự trong đó 2 ký tự đầu là năm vào học, 1 ký tự tiếp là bậc học (D: Đại học, C: Cao đẳng), 2 ký tự tiếp là ngành học (TH: Tin Học, KT: Kế Toán, QT: Điện tử, ĐT: Điện tử...)).
7. Điểm trung bình (Kiểu số thực).

Câu 8: Viết hàm xuất dữ liệu một sinh viên với thông tin vừa nhập ở trên.

Câu 9: Viết hàm nhập danh sách sinh viên, lưu trên mảng một chiều.

Câu 10: Viết hàm xuất danh sách sinh viên.

Câu 11: Tìm sinh viên có mã sinh viên là " X ".

Câu 12: Đếm số lượng sinh viên có điểm trung bình >5.

Câu 13: uất danh sách sinh viên thuộc ngành công nghệ thông tin.

Câu 14: Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin.

Câu 15: Sắp xếp danh sách sinh viên theo MSSV.

BÀI 7: ĐỆ QUY

Sau khi học xong bài này, học viên có thể

- *Hiểu khái niệm về đệ quy, các kiểu đệ quy;*
- *Biết ưu điểm và nhược điểm khi cài đặt hàm bằng phương pháp đệ quy;*
- *Biết cách khai báo và viết hàm theo kiểu đệ quy;*
- *Biết xử lý các giải thuật trên mảng 1 chiều bằng phương pháp đệ quy.*

7.1 KHÁI NIỆM

Đệ quy là một thuật toán dùng để đơn giản hóa những bài toán phức tạp bằng cách phân nhỏ phép toán đó thành nhiều phần đồng dạng.

Qua việc giải những bài toán được phân nhỏ này, những lời giải sẽ được kết hợp lại để giải quyết bài toán lớn hơn.

Một hàm được gọi là đệ quy nếu bên trong thân hàm có lệnh gọi đến chính nó.

Ví Dụ: Tính tổng $S(n) = 1 + 2 + 3 + \dots + n$.

Ta có:

$$S(n) = 1 + 2 + 3 + 4 + \dots + (n-1) + n = S(n-1) + n;$$

//Phương pháp thứ nhất là dùng vòng lặp (không đệ quy)

```
long tinhTong (int n)
{
    long s = 0;
    for(int i=1; i <= n; i++)
        s = s + i;
    return s;
}
```

//Phương pháp thứ hai là dùng hàm đệ quy:

```
long tinh tong (int n)
{
    if (n == 0) return 0;
    long s = tinh tong (n - 1);
    return s + n ;
}
```

7.2 KỸ THUẬT GIẢI BÀI TOÁN BẰNG ĐỆ QUY

1. Thông số hóa bài toán.
2. Tìm các điều kiện biên (chặn, dừng), tìm giải thuật cho các tình huống này.
3. Tìm giải thuật tổng quát theo hướng đệ quy lui dần về tình huống bị chặn.

Ví dụ Tính tổng 1 mảng a, n phần tử

- Thông số hóa: int a [], int n
- Điều kiện biên: Mảng 0 phần tử thì tổng bằng 0.
- Giải thuật chung:

$$\text{Sum}(a, n) = a[0] + a[1] + \dots + a[n-3] + a[n-2] + a[n-1]$$

$\underbrace{\hspace{10em}}_{\text{Sum}(a, n-1)}$

$$\text{Sum}(a, n) = \begin{cases} 0 & , n = 0 \\ a[n-1] + \text{Sum}(a, n-1), & n > 0 \end{cases}$$

// hàm cài đặt

```
long tong mang (int a[ ], int n)
{
    if (n==0) return 0;
    return a[n-1] + tong mang (a, n-1) ;
}
```


Lưu ý: Với các thuật toán đệ quy trên mảng, ta nên giảm dần số phần tử của mảng.

7.3 MỘT SỐ NHẬN XÉT VỀ HÀM ĐỆ QUY

1. Hàm đệ quy là hàm mà trong thân hàm lại gọi chính nó.
2. Giải thuật đệ quy đẹp (gọn gàng, dễ chuyển thành chương trình).
3. Tuy nhiên HÀM ĐỆ QUY: Vừa tốn bộ nhớ vừa chạy chậm
4. Nhiều ngôn ngữ không hỗ trợ giải thuật đệ quy (Fortran).
5. Nhiều giải thuật rất dễ mô tả dạng đệ quy nhưng lại rất khó mô tả với giải thuật không-đệ-quy.
6. Hàm đệ quy kém hiệu quả: tốn bộ nhớ và gọi hàm quá nhiều lần. Tuy nhiên viết hàm đệ quy rất ngắn gọn vì vậy tùy từng bài toán cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không (có những trường hợp không dùng đệ quy thì không giải quyết được bài toán).

7.4 SO SÁNH CẤU TRÚC LẶP VÀ ĐỆ QUY

1. Lập trình đệ qui sử dụng cấu trúc lựa chọn
2. Phương pháp lặp sử dụng cấu trúc lặp.
3. Cả 2 phương pháp đều liên quan đến quá trình lặp, tuy nhiên phương pháp lặp sử dụng vòng lặp một cách tường minh, còn phương pháp đệ qui có được quá trình lặp bằng cách sử dụng liên tục lời gọi hàm.
4. Cả 2 phương pháp đều phải kiểm tra khi nào thì kết thúc.
 - a. Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai.
 - b. Phương pháp đệ qui kết thúc khi đến trường cơ sở.
 - c. Phương pháp lặp thay đổi biến đếm trong vòng lặp cho đến khi nó làm cho điều kiện lặp sai.
 - d. Còn đệ qui làm cho các lời gọi hàm đơn giản dần cho đến khi đơn giản đến trường cơ sở.

5. Cả 2 phương pháp đều có thể dẫn đến trường hợp chạy vô hạn mãi.
- a. Lặp sẽ không thoát ra được khi điều kiện lặp không bao giờ sai.
 - b. Còn đệ quy không thoát ra được khi các bước đệ quy không làm cho bài toán đơn giản hơn và cuối cùng hội tụ về trường cơ sở.
 - c. Tuy nhiên đệ quy tồi hơn vì nó liên tục đưa ra lời gọi hàm làm tốn thời gian của bộ vi xử lý và không gian nhớ.

CÂU HỎI ÔN TẬP

Dùng phương pháp đệ quy giải các bài toán sau

Câu 1: Viết hàm nhập vào một mảng một chiều các số nguyên gồm n phần tử ($0 < n < 100$)

Câu 2: Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên

Câu 3: Tính tổng các phần tử có trong mảng

Câu 4: Tính tổng các phần tử chẵn có trong mảng

Câu 5: Tính tổng các phần tử lẻ có trong mảng

Câu 6: Tính tổng các phần tử nguyên tố có trong mảng

Câu 7: Tìm phần tử chẵn đầu tiên có trong mảng

Câu 8: Tìm phần tử lẻ đầu tiên có trong mảng

Câu 9: Tìm phần tử nguyên tố đầu tiên có trong mảng

Câu 10: Tìm phần tử chẵn cuối cùng có trong mảng

Câu 11: Tìm phần tử lớn nhất có trong mảng

Câu 12: Đếm số phần tử chẵn có trong mảng

Câu 13: Đếm số phần tử lớn nhất có trong mảng

BÀI 8: TẬP TIN (FILE)

Sau khi học xong bài này, học viên có thể

- *Hiểu một số khái niệm về tập tin;*
- *Biết Các bước thao tác với tập tin;*
- *Biết sử dụng một số hàm truy xuất đến tập tin văn bản;*
- *Biết sử dụng một số hàm truy xuất đến tập tin nhị phân.*

8.1 KHÁI NIỆM

Đối với các kiểu dữ liệu ta 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 chúng ta 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 cần lưu trữ. 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 chúng ta 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).

Có 3 loại dữ liệu kiểu tập tin:

- 1. Tập 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 (xác định bởi tổ hợp phím Ctrl + Z).

Tập tin văn bản chỉ có thể truy xuất theo kiểu tuần tự.

2. Tập tin định kiểu (Typed File): là loại tập tin bao gồm nhiều phần tử có cùng kiểu: char, int, long, cấu trúc... và được lưu trữ trên đĩa dưới dạng một chuỗi các byte liên tục.

3. Tập tin không định kiểu (Untyped File): là loại tập tin mà dữ liệu của chúng gồm các cấu trúc dữ liệu mà người ta không quan tâm đến nội dung hoặc kiểu của nó, chỉ lưu ý đến các yếu tố vật lý của tập tin như độ lớn và các yếu tố tác động lên tập tin mà thôi.

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 Of File).

8.2 CÁC THAO TÁC TRÊN TẬP TIN

Muốn thao tác trên tập tin, ta phải lần lượt làm theo các bước:

1. Khai báo biến tập tin.
2. Mở tập tin bằng hàm fopen().
3. Thực hiện các thao tác xử lý dữ liệu của tập tin bằng các hàm đọc/ghi dữ liệu.
4. Đóng tập tin bằng hàm fclose().

Ở đây, ta thao tác với tập tin nhờ các hàm được định nghĩa trong thư viện stdio.h.

8.2.1 Khai báo biến tập tin

Cú pháp: **FILE** <Danh sách các biến con trỏ>

Các biến trong danh sách phải là các con trỏ và được phân cách bởi dấu phẩy(,).

Ví dụ: FILE *f1, *f2;

8.2.2 Mở tập tin

Cú pháp: FILE *fopen(char *Path, const char *Mode)

Trong đó:

- **Path:** chuỗi chỉ đường dẫn đến tập tin trên đĩa.
- **Type:** chuỗi xác định cách thức mà tập tin sẽ mở. Các giá trị có thể của *Mode*:

Chế độ	Ý nghĩa
r	Mở tập tin văn bản để đọc
w	Tạo ra tập tin văn bản mới để ghi
a	Nối vào tập tin văn bản
rb	Mở tập tin nhị phân để đọc
wb	Tạo ra tập tin nhị phân để ghi
ab	Nối vào tập tin nhị phân
r+	Mở một tập tin văn bản để đọc/ghi
w+	Tạo ra tập tin văn bản để đọc ghi
a+	Nối vào hay tạo mới tập tin văn bản để đọc/ghi
r+b	Mở ra tập tin nhị phân để đọc/ghi
w+b	Tạo ra tập tin nhị phân để đọc/ghi
a+b	Nối vào hay tạo mới tập tin nhị phân

Hàm fopen trả về một con trỏ tập tin. Chương trình của ta không thể thay đổi giá trị của con trỏ này. Nếu có một lỗi xuất hiện trong khi mở tập tin thì hàm này trả về con trỏ NULL.

Ví dụ: Mở một tập tin tên c:\\ TEST.txt để ghi.

```
FILE *f;
f = fopen(" c:\\TEST.txt", "w");
if (f!=NULL)
{
    /* Các câu lệnh để thao tác với tập tin*/
}
```

```
/* Đóng tập tin*/  
}
```

Trong ví dụ trên, ta có sử dụng câu lệnh kiểm tra điều kiện để xác định mở tập tin có thành công hay không?

Khi mở tập tin để ghi (**chế độ w**), nếu tập tin đã tồn tại rồi thì nội dung của tập tin sẽ bị xóa và một tập tin mới được tạo ra.

Nếu ta muốn ghi nối dữ liệu, ta phải sử dụng chế độ "a".

Khi mở với chế độ đọc, tập tin phải tồn tại rồi, nếu không một lỗi sẽ xuất hiện.

8.2.3 Đóng tập tin

Hàm fclose() được dùng để đóng tập tin được mở bởi hàm fopen().

Hàm này sẽ ghi dữ liệu còn lại trong vùng đệm vào tập tin và đóng lại tập tin.

Cú pháp: **int fclose (FILE *f)**

Trong đó f là con trỏ tập tin được mở bởi hàm fopen().

Giá trị trả về của hàm là 0 báo rằng việc đóng tập tin thành công.

Hàm trả về EOF nếu có xuất hiện lỗi.

Ngoài ra, ta còn có thể sử dụng hàm fcloseall() để đóng tất cả các tập tin lại.

Cú pháp: **int fcloseall()**

Kết quả trả về của hàm là tổng số các tập tin được đóng lại.

Nếu không thành công, kết quả trả về là EOF.

8.2.4 Kiểm tra đến cuối tập tin hay chưa?

Cú pháp: **int feof (FILE *f)**

Ý nghĩa: Kiểm tra xem đã chạm tới cuối tập tin hay chưa và trả về EOF nếu cuối tập tin được chạm tới, ngược lại trả về 0.

8.2.5 Di chuyển con trỏ tập tin về đầu tập tin - Hàm `rewind()`

Khi ta đang thao tác một tập tin đang mở, con trỏ tập tin luôn di chuyển về phía cuối tập tin. Muốn cho con trỏ quay về đầu tập tin như khi mở nó, ta sử dụng hàm `rewind()`.

Cú pháp: **`void rewind (FILE *f)`**

8.3 TRUY CẬP TẬP TIN VĂN BẢN

8.3.1 Ghi dữ liệu lên tập tin văn bản

❖ Hàm `putc()`

Hàm này được dùng để ghi một ký tự lên một tập tin văn bản đang được mở để làm việc.

Cú pháp: **`int putc (int c, FILE *f)`**

Trong đó, tham số `c` chứa mã Ascii của một ký tự nào đó. Mã này được ghi lên tập tin liên kết với con trỏ `f`. Hàm này trả về EOF nếu gặp lỗi.

❖ Hàm `fputs()`

Hàm này dùng để ghi một chuỗi ký tự chứa trong vùng đệm lên tập tin văn bản.

Cú pháp: **`int fputs (const char *buffer, FILE *f)`**

Trong đó, `buffer` là con trỏ có kiểu `char` chỉ đến vị trí đầu tiên của chuỗi ký tự được ghi vào. Hàm này trả về giá trị 0 nếu `buffer` chứa chuỗi rỗng và trả về EOF nếu gặp lỗi.

❖ Hàm `fprintf()`

Hàm này dùng để ghi dữ liệu có định dạng lên tập tin văn bản.

Cú pháp: **`fprintf (FILE *f, const char *format, varexpr)`**

Trong đó:

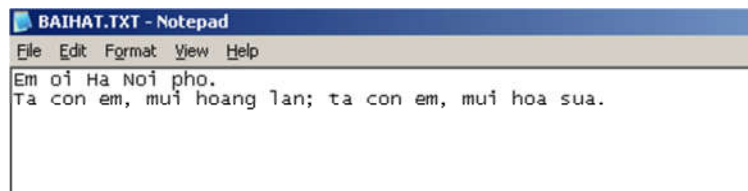
- **format:** chuỗi định dạng (giống với các định dạng của hàm `printf()`).
- **varexpr:** danh sách các biểu thức, mỗi biểu thức cách nhau dấu phẩy (,).

Định dạng	Ý nghĩa
%d	Ghi số nguyên
%[.số chữ số thập phân] f	Ghi số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
%o	Ghi số nguyên hệ bát phân
%x	Ghi số nguyên hệ thập lục phân
%c	Ghi một ký tự
%s	Ghi chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	Ghi số thực dạng khoa học (nhân 10 mũ x)

Ví dụ: Viết chương trình ghi chuỗi ký tự lên tập tin văn bản D:\\Baihat.txt

```
#include<stdio.h>
#include<conio.h>
int main ()
{
    FILE *f;
    f = fopen ("D:\\Baihat.txt","r+");
    if (f!=NULL)
    {
        fputs("Em oi Ha Noi pho.\n",f);
        fputs("Ta con em, mui hoang lan; ta con em, mui hoa sua.",f);
        fclose(f);
    }
    return 0;
}
```

Nội dung tập tin Baihat.txt khi được mở bằng trình soạn thảo văn bản Notepad.



8.3.2 Đọc dữ liệu từ tập tin văn bản

❖ Hàm `getc()`

Hàm này dùng để đọc dữ liệu từ tập tin văn bản đang được mở để làm việc.

Cú pháp: `int getc (FILE *f)`

Hàm này trả về mã Ascii của một ký tự nào đó (kể cả EOF) trong tập tin liên kết với con trỏ `f`.

❖ Hàm `fgets()`

Cú pháp: `char *fgets (char *buffer, int n, FILE *f)`

Hàm này được dùng để đọc một chuỗi ký tự từ tập tin văn bản đang được mở ra và liên kết với con trỏ `f` cho đến khi đọc đủ `n` ký tự hoặc gặp ký tự xuống dòng `'\n'` (ký tự này cũng được đưa vào chuỗi kết quả) hay gặp ký tự kết thúc EOF (ký tự này không được đưa vào chuỗi kết quả).

Trong đó:

- **buffer (vùng đệm):** con trỏ có kiểu `char` chỉ đến vùng nhớ đủ lớn chứa các ký tự nhận được.
- **n:** giá trị nguyên chỉ độ dài lớn nhất của chuỗi ký tự nhận được.
- **f:** con trỏ liên kết với một tập tin nào đó.
- **Ký tự NULL (`'\0'`)** tự động được thêm vào cuối chuỗi kết quả lưu trong vùng đệm.

Hàm trả về địa chỉ đầu tiên của vùng đệm khi không gặp lỗi và chưa gặp ký tự kết thúc EOF. Ngược lại, hàm trả về giá trị `NULL`.

❖ Hàm `fscanf()`

Hàm này dùng để đọc dữ liệu từ tập tin văn bản vào danh sách các biến theo định dạng.

Cú pháp: `fscanf (FILE *f, const char *format, varlist)`

Trong đó:

format: chuỗi định dạng (giống hàm `scanf()`);

varlist: danh sách các biến mỗi biến cách nhau dấu phẩy (,).

Ví dụ: Viết chương trình chép tập tin D:\Baihat.txt ở trên sang tập tin D:\Baica.txt.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *f1,*f2 ;
    f1=fopen("D:\\Baihat.txt", "rt");
    f2=fopen("D:\\Baica.txt", "wt");
    if (f1!=NULL && f2!=NULL)
    {
        int ch=fgetc (f1);
        while (! feof (f1))
        {
            fputc(ch,f2);
            ch=fgetc(f1);
        }
        fcloseall();
    }
    return 0;
}
```

8.4 TRUY CẬP TẬP TIN NHỊ PHÂN

8.4.1 Ghi dữ liệu lên tập tin nhị phân - Hàm fwrite()

Cú pháp: `size_t fwrite(const void *ptr, size_t size, size_t n, FILE*f)`

Trong đó:

- **ptr:** con trỏ chỉ đến vùng nhớ chứa thông tin cần ghi lên tập tin.
- **n:** số phần tử sẽ ghi lên tập tin.
- **size:** kích thước của mỗi phần tử.

- **f**: con trỏ tập tin đã được mở.

Giá trị trả về của hàm này là số phần tử được ghi lên tập tin. Giá trị này bằng n trừ khi xuất hiện lỗi.

8.4.2 Đọc dữ liệu từ tập tin nhị phân - Hàm fread()

Cú pháp: `size_t fread (const void *ptr, size_t size, size_t n, FILE *f)`

Trong đó:

- **ptr**: con trỏ chỉ đến vùng nhớ sẽ nhận dữ liệu từ tập tin.
- **n**: số phần tử được đọc từ tập tin.
- **size**: kích thước của mỗi phần tử.
- **f**: con trỏ tập tin đã được mở.

Giá trị trả về của hàm này là số phần tử đã đọc được từ tập tin. Giá trị này bằng n hay nhỏ hơn n nếu đã chạm đến cuối tập tin hoặc có lỗi xuất hiện.

Ví dụ 1: Viết chương trình 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;
    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);
    }
    return 0;
}

```

Ví dụ 2: Mỗi sinh viên cần quản lý ít nhất 2 thông tin: mã sinh viên và họ tên. Viết chương trình cho phép lựa chọn các chức năng: nhập danh sách sinh viên từ bàn phím rồi ghi lên tập tin SinhVien.dat, đọc dữ liệu từ tập tin SinhVien.dat rồi hiển thị danh sách lên màn hình, tìm kiếm họ tên của một sinh viên nào đó dựa vào mã sinh viên nhập từ bàn phím.

Ta nhận thấy rằng mỗi phần tử của tập tin SinhVien.Dat là một cấu trúc có 2 trường: mã và họ tên. Do đó, ta cần khai báo cấu trúc này và sử dụng các hàm đọc/ghi tập tin nhị phân với kích thước mỗi phần tử của tập tin là chính kích thước cấu trúc đó.

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
typedef struct
{
    char Ma[10];
    char HoTen[40];
} SinhVien;
void WriteFile (char *FileName)
{
    FILE *f;
    int n,i;
    SinhVien sv;
    f=fopen(FileName,"ab");
    printf("Nhap bao nhieu sinh vien? ");
    scanf("%d",&n);
    fflush(stdin);
    for(i=1;i<=n;i++)

```

```
{
    printf("Sinh vien thu %i\n",i);
    printf(" - MSSV: ");
    gets(sv.Ma);
    printf(" - Ho ten: ");
    gets(sv.HoTen);
    fwrite(&sv,sizeof(sv),1,f);
    fflush(stdin);
}
fclose(f);
printf("Bam phim bat ky de tiep tuc");
getch();
}
void ReadFile(char *FileName)
{
    FILE *f;
    SinhVien sv;
    f=fopen(FileName,"rb");
    printf("    MSSV    |    Ho va ten\n");
    fread (&sv,sizeof(sv),1,f);
    while (!feof(f))
    {
        printf(" %s    |    %s\n",sv.Ma,sv.HoTen);
        fread(&sv,sizeof(sv),1,f);
    }
    fclose(f);
    printf("Bam phim bat ky de tiep tuc!!!");
    getch();
}
void Search(char *FileName)
{
    char MSSV[10] ;
    FILE *f ;
    int Found=0;
    SinhVien sv;
    fflush(stdin);
    printf("Ma so sinh vien can tim: ");
    gets(MSSV);
    f=fopen(FileName,"rb");
    while (!feof(f) && Found==0)
    {
        fread(&sv,sizeof(sv),1,f);
```

```
        if (strcmp(sv.Ma,MSSV)==0)
            Found=1;
    }
    fclose(f);
    if (Found == 1)
        printf("Tim thay SV co ma %s. Ho ten la: %s",sv.Ma,sv.HoTen);
    else
        printf("Tim khong thay sinh vien co ma %s",MSSV);
    printf("\nBam phim bat ky de tiep tuc!!!");
    getch();
}

int main()
{
    int c;
    for (;;)
    {
        printf("1. Nhap DSSV\n");
        printf("2. In DSSV\n");
        printf("3. Tim kiem\n");
        printf("4. Thoat\n");
        printf("Ban chon 1, 2, 3, 4: ");
        scanf("%d",&c);
        if(c==1)
            WriteFile("d:\\SinhVien.Dat");
        else if (c==2)
            ReadFile("d:\\SinhVien.Dat");
        else if (c==3) Search("d:\\SinhVien.Dat");
        else break;
    }
    return 0 ;
}
```

CÂU HỎI ÔN TẬP

Câu 1: Viết chương trình quản lý một tập tin văn bản theo các yêu cầu:

- Nhập từ bàn phím nội dung một văn bản sau đó ghi vào đĩa.
- Đọc từ đĩa nội dung văn bản vừa nhập và in lên màn hình.
- Đọc từ đĩa nội dung văn bản vừa nhập, in nội dung đó lên màn hình và cho phép nối thêm thông tin vào cuối tập tin đó.

Câu 2: Cho file TXT có cấu trúc như sau:

Dòng đầu lưu giá trị của 1 số nguyên dương n . Dòng còn lại lưu giá trị của 1 dãy gồm n số nguyên

- Viết chương trình đọc file trên, lưu vào mảng 1 chiều.
- Xuất dãy đọc được ra màn hình.
- Ghi các số nguyên tố có trong mảng vào file đã cho (ghi tiếp theo, không xoá nội dung cũ).

Câu 3: Cho file TXT có cấu trúc như sau:

Dòng đầu lưu giá trị của 2 số nguyên dương d, c . Các dòng còn lại lưu giá trị của 1 ma trận d dòng và c cột là các số nguyên.

- Viết chương trình đọc file ma trận trên.
- Xuất ma trận đó ra màn hình
- Nếu ma trận vuông thì hãy ghi các phần tử trên đường chéo chính của ma trận vào file đã cho (ghi tiếp theo, không xoá nội dung cũ).

TÀI LIỆU THAM KHẢO

1. Phạm Văn Ất, Kỹ thuật Lập trình C- cơ bản và nâng cao, NXB KH & KT - 2003.
2. Quách Tuấn Ngọc, Tin học căn bản, Nhà xuất bản giáo dục - 1997.
3. Hoàng Kiếm, Nguyễn Đức Thắng, Đinh Nguyễn Anh Dũng, Giáo trình Tin học Đại cương, Nhà xuất bản giáo dục - 1999.
4. Nguyễn Tấn Trần Minh Khang, Bài giảng Kỹ Thuật Lập trình, Khoa Công Nghệ Thông Tin, Đại học Khoa học Tự nhiên
5. Nguyễn Thanh Thủy (chủ biên), Nhập môn lập trình ngôn ngữ C, Nhà xuất bản Khoa học kỹ thuật – 2000.
6. Trần Minh Thái, Bài giảng và bài tập Lập trình căn bản; Khoa Công Nghệ Thông Tin, Đại học Khoa học Tự nhiên
7. Mai Ngọc Thu, Giáo trình C. Khoa Công Nghệ Thông Tin, Đại học Kỹ Thuật Công Nghệ.
8. Brain W. Kernighan & Dennis Ritchie, The C Programming Language, Prentice Hall Publisher, 1988.