BỘ LAO ĐỘNG -THƯƠNG BINH VÀ XÃ HỘI

TỔNG CỤC DẠY NGHỀ

-----□□ □ □□-----

GIÁO TRÌNH
LẬP TRÌNH CĂN BẢN
NGHỀ: KỸ THUẬT SỬA CHỮA, LẮP RÁP MÁY TÍNH

TRÌNH ĐỘ: CAO ĐẲNG

(Ban hành theo Quyết định số: 120/QĐ-TCDN ngày 25 tháng 02 năm 2013 của Tổng cục trưởng Tổng cục dạy nghề)

TUYÊN BỐ BẢN QUYỀN:

LỜI NÓI ĐẦU

Tin học là một ngành khoa học mũi nhọn phát triển hết sức nhanh chóng trong vài chục năm lại đây và ngày càng mở rộng lĩnh vực nghiên cứu, ứng dụng trong mọi mặt của đời sống xã hội.

Ngôn ngữ lập trình là một loại công cụ giúp con người thể hiện các vấn đề của thực tế lên máy tính một cách hữu hiệu. Với sự phát triển của tin học, các ngôn ngữ lập trình cũng dần cải tiến để đáp ứng các thách thức mới của thực tế.

Khoảng cuối những năm 1960 đầu 1970 xuất hiện nhu cầu cần có các ngôn ngữ bậc cao để hỗ trợ cho những nhà tin học trong việc xây dựng các phần mềm hệ thống, hệ điều hành. Ngôn ngữ C ra đời từ đó, nó đã được phát triển tại phòng thí nghiệm Bell. Đến năm 1978, giáo trình " Ngôn ngữ lập trình C " do chính các tác giả của ngôn ngữ là Dennish Ritchie và B.W. Kernighan viết, đã được xuất bản và phổ biến rộng rãi.

C là ngôn ngữ lập trình vạn năng. Ngoài việc C được dùng để viết hệ điều hành UNIX, người ta nhanh chóng nhận ra sức mạnh của C trong việc xử lý cho các vấn đề hiện đại của tin học. C không gắn với bất kỳ một hệ điều hành hay máy nào, và mặc dầu nó đã được gọi là " ngôn ngữ lập trình hệ thống" vì nó được dùng cho việc viết hệ điều hành, nó cũng tiện lợi cho cả việc viết các chương trình xử lý số, xử lý văn bản và cơ sở dữ liệu.

Toàn bộ giáo trình gồm sáu chương chứa đựng tương đối đầy đủ các vấn đề cơ bản nhất của ngôn ngữ lập trình C, các loại ví dụ và bài tập chọn lọc cùng một số vấn đề liên quan, giúp người học có khả năng sử dụng thành thạo ngôn ngữ này trong việc giải quyết một số lớp bài toán thông dụng trong thực tế.

Giáo trình được biên soạn cho đối tượng chính là học sinh THCN, kỹ thuật viên tin học, tuy nhiên nó cũng có thể là tài liệu tham khảo bổ ích cho bậc đại học và những người quan tâm.

mặc dù đã cố gắng nhiều trong quá trình biên soạn giáo trình này, nhưng chắc chắn không tránh khỏi có những thiếu sót. Rất mong nhận được ý kiến đóng góp của độc giả và các đồng nghiệp để giáo trình ngày càng hoàn thiện hơn.

Và bây giờ chúng ta đi tìm hiểu thế giới của ngôn ngữ C từ những khái niệm ban đầu cơ bản nhất.

Hà Nội, 2013
Tham gia biên soạn
Khoa Công Nghệ Thông Tin
Trường Cao Đẳng Nghề Kỹ Thuật Công Nghệ
Địa Chỉ: Tổ 59 Thị trấn Đông Anh – Hà Nội
Tel: 04. 38821300
Chủ biên: Trần Thị Vinh

Mọi góp ý liên hệ: Phùng Sỹ Tiến – Trưởng Khoa Công Nghệ Thông Tin
Mobible: 0983393834
Email: tienphungktcn@gmail.com – tienphungktcn@yahoo.com

MỤC LỤC

TRANG

MÔ ĐUN: LẬP TRÌNH CĂN BẢN

Mã mô đun: MĐ11

Vị trí, ý nghĩa, vai trò của mô đun:

- Vị trí:

+ Mô đun được bố trí sau khi học sinh học xong các môn học chung, trước các môn học/ mô đun đào tạo chuyên môn nghề.

- Tính chất:

+ Là mô đun cơ sở.

- Ý nghĩa và vai trò của mô đun:

+ Là mô đun không thể thiếu của nghề SCLR máy tính

+ Là mô đun tư duy logic về lập trình

Mục tiêu của mô đun:

- Hiểu được công dụng của ngôn ngữ lập trình, hiểu cú pháp, công dụng của các câu lệnh dùng trong ngôn ngữ lập trình.
- Phân tích được chương trình: xác định nhiệm vụ chương trình (phải làm gì).
- Vận dụng điều kiện, trợ giúp môi trường của ngôn ngữ lập trình, chẳng hạn: các thao tác biên tập chương trình, các công cụ, điều khiển, thực đơn lệnh trợ giúp, gỡ rối, bẫy lỗi, v. v.
- Viết chương trình và thực hiện chương trình trong máy tính.
- Tự tin khi tiếp cận các mã (code) chương trình.
- Loại bỏ tâm lý lo sợ khi gặp những công việc được lập trình hóa.

Nội dung của mô đun:

Mã bài	Tên các bài trong mô đun	Thời lượng			
		Tổng số	Lý thuyết	Thực hành	Kiểm tra
MĐ11-01	Bài 1: Tổng quan về ngôn ngữ lập trình:	02	02		
MĐ11-02	Bài 2: Các thành phần cơ bản	08	02	06	
MĐ11-03	Bài 3: Các lệnh cấu trúc	16	04	10	02
MĐ11-04	Bài 4: Hàm	20	08	10	02
MĐ11-05	Bài 5: Mảng	13	04	08	01
MĐ11-06	Bài 6: Chuỗi ký tự	13	04	08	01
MĐ11-07	Bài 7: Mảng và biến con trỏ	18	04	12	02

BÀI 1
TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C

Mã bài: MĐ 11-01

Giới thiệu:

Bài này nhằm cung cấp cho người học các kiến thức về lịch sử phát triển của ngôn ngữ, ứng dụng thực tế của ngôn ngữ, các cách khởi động và thoát khỏi chương trình.

Mục tiêu

- Biết được ngôn ngữ này có những ứng dụng thực tế như thế nào;

- Biết sử dụng được hệ thống trợ giúp của ngôn ngữ lập trình;

1.Giới thiệu lịch sử phát triển của ngôn ngữ lập trình C

Mục tiêu:

- Biết được lịch sử phát triển của ngôn ngữ lập trình;

Vào đầu những năm 70 tại phòng thí nghiệm Bell, Dennis Ritchie đã phát triển ngôn ngữ C. C được sử dụng lần đầu trên một hệ thống cài đặt hệ điều hành UNIX. C có nguồn gốc từ ngôn ngữ BCPL do Martin Richards phát triển. BCPL sau đó đã được Ken Thompson phát triển thành ngôn ngữ B, đây là người khởi thủy ra C.

Trong khi BCPL và B không hỗ trợ kiểu dữ liệu, thì C đã có nhiều kiểu dữ liệu khác nhau. Những kiểu dữ liệu chính gồm : kiểu ký tự (character), kiểu số nguyên (interger) và kiểu số thực (float).

C liên kết chặt chẽ với hệ thống UNIX nhưng không bị trói buộc vào bất cứ một máy tính hay hệ điều hành nào. C rất hiệu quả để viết các chương trình thuộc nhiều những lĩnh vực khác nhau.

C cũng được dùng để lập trình hệ thống. Một chương trình hệ thống có ý nghĩa liên quan đến hệ điều hành của máy tính hay những tiện ích hỗ trợ nó. Hệ điều hành (OS), trình thông dịch (Interpreters), trình soạn thảo (Editors), chương trình Hợp Ngữ (Assembly) là các chương trình hệ thống. Hệ điều hành UNIX được phát triển dựa vào C. C đang được sử dụng rộng rãi bởi vì tính hiệu quả và linh hoạt. Trình biên dịch (compiler) C có sẵn cho hầu hết các máy tính. Mã lệnh viết bằng C trên máy này có thể được biên dịch và chạy trên máy khác chỉ cần thay đổi rất ít hoặc không thay đổi gì cả. Trình biên dịch C dịch nhanh và cho ra mã đối tượng không lỗi.

C khi thực thi cũng rất nhanh như hợp ngữ (Assembly). Lập trình viên có thể tạo ra và bảo trì thư viện hàm mà chúng sẽ được tái sử dụng cho chương trình khác. Do đó, những dự án lớn có thể được quản lý dễ dàng mà tốn rất ít công sức.

2. Cách khởi động và thoát chương trình

Mục tiêu:

- Biết cách khởi động được và thoát khỏi chương trình C;

2.1. Khởi Động Chương Trình C

Nhập lệnh tại dấu nhắc DOS: gõ BC ↵ (Enter) (nếu đường dẫn đã được cài đặt bằng lệnh path trong đó có chứa đường dẫn đến thư mục chứa tập tin BC.EXE). Nếu đường dẫn chưa được cài đặt ta tìm xem thư mục BORLANDC nằm ở ổ đĩa nào. Sau đó ta gõ lệnh sau: <ổ đĩa>:\BORLANDC\BIN\BC ↵ (Enter)

Nếu bạn muốn vừa khởi động BC vừa soạn thảo chương trình với một tập tin có tên do chúng ta đặt, thì gõ lệnh: BC [đường dẫn]<tên file cần soạn thảo>, nếu tên file cần soạn thảo đã có thì được nạp lên, nếu chưa có sẽ được tạo mới.

Khởi động tại Windows: Bạn vào menu Start, chọn Run, bạn gõ vào hộp Open 1 trong

các dòng lệnh như nhập tại DOS. Hoặc bạn vào Window Explorer, chọn ổ đĩa chứa thư mục BORLANDC, vào thư mục BORLANDC, vào thư mục BIN, khởi động tập tin BC.EXE.

Ví dụ: Bạn gõ D:\BORLANDC\BIN\BC E:\BAITAP_BC\VIDU1.CPP

Câu lệnh trên có nghĩa khởi động BC và nạp tập tin VIDU1.CPP chứa trong thư mục BAITAP_BC trong ổ đĩa E. Nếu tập tin này không có sẽ được tạo mới.

2.2. Thoát khỏi chương trình C

Ấn phím F10 (kích hoạt Menu), chọn menu File, chọn Quit;

Hoặc ấn tổ hợp phím Alt – X.

3.Hệ thống thông tin giúp đỡ

Mục tiêu:

- *Trình bày được hệ thống trợ giúp của C*
- Ấn phím F1 để kích hoạt màn hình Help chính.
- Muốn xem Help của hàm trong soạn thảo, di chuyển con trỏ đến vị trí hàm đó ấn tổ hợp

phím Ctrl - F1

- Ấn tổ hợp phím Shift - F1 để xem danh sách các mục Help
- Ấn tổ hợp phím Alt - F1 để quay về màn hình Help trước đó.

BÀI 2
CÁC THÀNH PHẦN CƠ BẢN

Mã bài: MĐ 11-02

Giới thiệu

Cũng như những ngôn ngữ lập trình khác, ngôn ngữ lập trình c cũng có hệ thống ký hiệu và từ khóa, các kiểu dữ liệu. Bài học này sẽ cho người học các kiến thức về các loại hằng, biến, các lệnh, khối lệnh và thực hiện việc chạy chương trình.

Mục tiêu:

- Hiểu và vận dụng được các lệnh cấu trúc: cấu trúc lựa chọn, cấu trúc lặp xác định và lặp vô định.
- Hiểu và vận dụng được các lệnh bẻ vòng lặp
- Rèn luyện thói quen suy luận logic.

1. Hệ thống ký hiệu và từ khóa

Mục tiêu:

- Trình bày được tập kí tự dùng trong ngôn ngữ C

- Trình bày được các quy tắc cần nhớ khi viết chương trình

- Tập ký tự dùng trong ngôn ngữ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Các từ lại được liên kết với nhau theo một qui tắc nào đó để tạo nên các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và thể hiện một thuật toán để giải một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau :

26 chữ cái hoa : A B C .. Z
26 chữ cái thường : a b c .. z
10 chữ số : 0 1 2 .. 9
Các ký hiệu toán học : + - * / = ( )
Ký tự gạch nối : _
Các ký tự khác : . , : ; [ ] { } ! \ & % # $ ...

Dấu cách (space) dùng để tách các từ. Ví dụ chữ VIET NAM có 8 ký tự, còn VIETNAM chỉ có 7 ký tự.

Chú ý :

Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên.

Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2+bx+c=0$, ta cần tính biệt thức Delta = $b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự , vì vậy ta phải dùng ký hiệu khác để thay thế.

- Từ khóa

Từ khoá là những từ được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Bảng dưới đây liệt kê các từ khoá của TURBO C :

1.	asm	break	case	cdecl
2.	char	const	continue	default
3.	do	double	else	enum
4.	extern	far	float	for
5.	goto	huge	if	int
6.	interrupt	long	near	pascal
7.	register	return	short	signed
8.	sizeof	static	struct	switch
9.	tipedef	union	unsigned	void

10.	volatile	while		

Ý nghĩa và cách sử dụng của mỗi từ khoá sẽ được đề cập sau này, ở đây ta cần chú ý :

- Không được dùng các từ khoá để đặt tên cho các hằng, biến, mảng, hàm ...

- Từ khoá phải được viết bằng chữ thường, ví dụ : viết từ khoá khai báo kiểu nguyên là int chứ không phải là INT.

- Tên

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn,...

Tên được đặt theo qui tắc sau :

Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng : Option-Compiler-Source-Identifier length khi dùng TURBO C.

Ví dụ :

Các tên đúng :

a_1 delta x1 _step GAMA

Các tên sai :

3MN	Ký tự đầu tiên là số
m#2	Sử dụng ký tự #
f(x)	Sử dụng các dấu ( )
do	Trùng với từ khoá
te ta	Sử dụng dấu trắng
Y-3	Sử dụng dấu -

Chú ý :

Trong TURBO C, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. trong C, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

- Một số quy tắc cần nhớ khi viết chương trình

Qui tắc đầu tiên cần nhớ là :

Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu;

Qui tắc thứ hai là :

Các lời giải thích cần được đặt giữa các dấu / và */ và có thể được viết*

Trên một dòng

Trên nhiều dòng

Trên phần còn lại của dòng

Qui tắc thứ ba là :

Trong chương trình, khi ta sử dụng các hàm chuẩn, ví dụ như printf(), getch(),... mà các hàm này lại chứa trong file stdio.h trong thư mục của C, vì vậy ở đầu chương trình ta phải khai báo sử dụng ;

#include "stdio.h "

Qui tắc thứ tư là :

Một chương trình có thể chỉ có một hàm chính (hàm main()) hoặc có thể có thêm vài hàm khác.

2. Các kiểu dữ liệu: kiểu số, chuỗi, ký tự

Mục tiêu:

- Trình bày được các kiểu dữ liệu.

- Kiểu ký tự

Một giá trị kiểu char chiếm 1 byte (8 bit) và biểu diễn được một ký tự thông qua bảng mã ASCII. Ví dụ:

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
B	066
A	097
B	098

Có hai kiểu dữ liệu char : kiểu signed char và unsigned char.

	Kiểu	Phạm vi biểu diễn	Số ký tự	Kích thước	
	Char (Signed char)	-128 đến 127	256	1 byte	
	Unsigned char	0 đến 255	256	1 byte	

Ví dụ sau minh hoạ sự khác nhau giữa hai kiểu dữ liệu trên : Xét đoạn chương trình sau :

```
char  ch1;
unsigned char ch2;
......
ch1=200; ch2=200;
```

Khi đó thực chất :

```
ch1=-56;
ch2=200;
```

Nhưng cả ch1 và ch2 đều biểu diễn cùng một ký tự có mã 200.

Phân loại ký tự :

Có thể chia 256 ký tự làm ba nhóm :

Nhóm 1: Nhóm các ký tự điều khiển có mã từ 0 đến 31. Chẳng hạn ký tự mã 13 dùng để chuyển con trỏ về đầu dòng, ký tự 10 chuyển con trỏ

xuống dòng dưới (trên cùng một cột). Các ký tự nhóm này nói chung không hiển thị ra màn hình.

Nhóm 2 : Nhóm các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể được đưa ra màn hình hoặc máy in.

Nhóm 3 : Nhóm các ký tự đồ hoạ có mã số từ 127 đến 255. Các ký tự này có thể đưa ra màn hình nhưng không in ra được (bằng các lệnh DOS).

- Kiểu số nguyên

Trong C cho phép sử dụng số nguyên kiểu int, số nguyên dài kiểu long và số nguyên không dấu kiểu unsigned. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây :

	Kiểu	Phạm vi biểu diễn	Kích thước
1	int	-32768 đến 32767	2 byte
2	unsigned int	0 đến 65535	2 byte
3	long	-2147483648 đến 2147483647	4 byte
4	unsigned long	0 đến 4294967295	4 byte

Chú ý :

Kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên.

- Kiểu số phẩy động độ chính xác đơn (float), kép (double)

Trong C cho phép sử dụng ba loại dữ liệu dấu phảy động, đó là float, double và long double. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây :

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
Float	3.4E-38 đến 3.4E+38	7 đến 8	4 byte
Double	1.7E-308 đến 1.7E+308	15 đến 16	8 byte
long double	3.4E-4932 đến 1.1E4932	17 đến 18	10 byte

Giải thích :

Máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ 3.4E-38 đến 3.4E+38. Các số có giá trị tuyệt đối nhỏ hơn3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

3. Biến, hằng, biểu thức

3.1. Hằng

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán.

- Tên hằng :

Nguyên tắc đặt tên hằng ta đã xem xét trong mục 1.3.
Để đặt tên một hằng, ta dùng dòng lệnh sau :
#define tên hằng giá trị
Ví dụ :

#define MAX 1000

Lúc này, tất cả các tên MAX trong chương trình xuất hiện sau này đều được thay bằng 1000. Vì vậy, ta thường gọi MAX là tên hằng, nó biểu diễn số 1000.
Một ví dụ khác : #define pi 3.141593 Đặt tên cho một hằng float là pi có giá trị là 3.141593.
- Các loại hằng:
+ Hằng int:
Hằng int là số nguyên có giá trị trong khoảng từ -32768 đến 32767.
Chú ý :
Cần phân biệt hai hằng 5056 và 5056.0 : ở đây 5056 là số nguyên còn 5056.0 là hằng thực.
+ Hằng long :
Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647.
Hằng long được viết theo cách :
1234L hoặc 1234l
(thêm L hoặc l vào đuôi)
Một số nguyên vượt ra ngoài miền xác định của int cũng được xem là long.
Ví dụ :

#define sl 8865056L	Định nghiã hằng long sl có giá trị là 8865056
#define sl 8865056	Định nghiã hằng long sl có giá trị là 8865056

+ Hằng int hệ 8 :
Hằng int hệ 8 được viết theo cách 0c1c2c3....Ở đây ci là một số nguyên dương trong khoảng từ 1 đến 7. Hằng int hệ 8 luôn luôn nhận giá trị dương.

Ví dụ :

#define h8 0345	Định nghiã hằng int hệ 8 có giá trị là 3*8*8+4*8+5=229

+ Hằng int hệ 16 :
Trong hệ này ta sử dụng 16 ký tự : 0,1..,9,A,B,C,D,E,F.

Cách viết	Giá trị
a hoặc A	10

b hoặc B	11
c hoặc C	12
d hoặc D	13
e hoặc E	14
f hoặc F	15

Hằng số hệ 16 có dạng 0xc1c2c3... hoặc 0Xc1c2c3... Ở đây ci là một số trong hệ 16.

Ví dụ :

```
#define  h16 0xa5
#define  h16 0xA5
#define  h16 0Xa5
 #define  h16 0XA5
```

Cho ta các hắng số h16 trong hệ 16 có giá trị như nhau. Giá trị của chúng trong hệ 10 là :

10*16+5=165.

+ Hằng ký tự :

Hằng ký tự là một ký tự riêng biệt được viết trong hai dấu nháy đơn, ví dụ 'a'.

Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác. Ví dụ :

'9'-'0'=57-48=9

Ví dụ :

```
#define  kt 'a'          Định nghiã hằng ký tự kt có giá trị là 97
```

Hằng ký tự còn có thể được viết theo cách sau :

' \c1c2c3'

Trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

Ví dụ : chữ a có mã hệ 10 là 97, đổi ra hệ 8 là 0141. Vậy hằng ký tự 'a' có thể viết dưới dạng '\141'. Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu\ ):

Cách viết	Ký tự
'\''	'
'\"'	"
'\\'	\
'\n'	\n (chuyển dòng)
'\0'	\0 (null)
'\t'	Tab
'\b'	Backspace
'\r'	CR (về đầu

dòng)

'\f' LF (sang trang)

Chú ý :

Cần phân biệt hằng ký tự '0' và '\0'. Hằng '0' ứng với chữ số 0 có mã ASCII là 48, còn hằng '\0' ứng với kýtự \0 (thường gọi là ký tự null) có mã ASCII là 0.

Hằng ký tự thực sự là một số nguyên, vì vậy có thể dùng các số nguyên hệ 10 để biểu diễn các ký tự, ví dụ lệnh printf("%c%c",65,66) sẽ in ra AB.

+ Hằng xâu ký tự :

Hằng xâu ký tự là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép.

Ví dụ :

```
#define xau1 "Ha noi"
        #define xau2 "My name is Giang"
```

Xâu ký tự được lưu trữ trong máy dưới dạng một bảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự null \0 vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

Chú ý :

Cần phân biệt hai hằng 'a' và "a". 'a' là hằng ký tự được lưu trữ trong 1 byte, còn "a" là hằng xâu ký tự được lưu trữ trong 1 mảng hai phần tử : phần tử thứ nhất chứa chữ a còn phần tử thứ hai chứa \0.

3.2. Biến

Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng. Việc khai báo biến được thực hiện theo mẫu sau :

Kiểu dữ liệu của biến tên biến ;

Ví dụ :

int a,b,c;	Khai báo ba biến int là a,b,c
long dai,mn;	Khai báo hai biến long là dai và mn
char kt1,kt2;	Khai báo hai biến ký tự là kt1 và kt2
float x,y	Khai báo hai biến float là x và y
doublecanh1,canh 2;	Khai báo hai biến double là canh1 và canh2

Biến kiểu int chỉ nhận được các giá trị kiểu int. Các biến khác cũng có ý nghĩa tương tự. Các biến kiểu char chỉ chứa được một ký tự. Để lưu trữ được một xâu ký tự cần sử dụng một mảng kiểu char.

Vị trí của khai báo biến :

Các khai báo cần phải được đặt ngay sau dấu { đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác. Sau đây là một ví dụ về khai báo biến sai :

```
( Khái niệm về hàm và cấu trúc chương trình sẽ nghiên cứu sau này)
main()
```

```
{
        int a,b,c;
        a=2;
        int d; /* Vị trí của khai báo sai */
        .....
}
```

Khởi đầu cho biến :

Nếu trong khai báo ngay sau tên biến ta đặt dấu = và một giá trị nào đó thì đây chính là cách vừa khai báo vừa khởi đầu cho biến.

Ví dụ :

```
        int a,b=20,c,d=40;
        float e=-55.2,x=27.23,y,z,t=18.98;
```

Việc khởi đầu và việc khai báo biến rồi gán giá trị cho nó sau này là hoàn toàn tương đương.

Lấy địa chỉ của biến :

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ của biến sẽ được sử dụng trong một số hàm ta sẽ nghiên cứu sau này (ví dụ như hàm scanf).

Để lấy địa chỉ của một biến ta sử dụng phép toán :

& tên biến

3.3. Biểu thức :

Biểu thức là một sự kết hợp giữa các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó. Mỗi biểu thức có sẽ có một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức.

Trong C, ta có hai khái niệm về biểu thức :

Biểu thức gán.

Biểu thức điều kiện .

Biểu thức được phân loại theo kiểu giá trị : nguyên và thực. Trong các mệnh đề logic, biểu thức được phân thành đúng (giá trị khác 0) và sai (giá trị bằng 0).

Biểu thức thường được dùng trong :

Vế phải của câu lệnh gán.

Làm tham số thực sự của hàm.

Làm chỉ số.

Trong các toán tử của các cấu trúc điều khiển.

Tới đây, ta đã có hai khái niệm chính tạo nên biểu thức đó là toán hạng và phép toán. Toán hạng gồm: hằng, biến, phần tử mảng và hàm trước đây ta đã xét. Dưới đây ta sẽ nói đến các phép toán.

4. Lệnh và khối lệnh

Mục tiêu:

- Phân biệt được lệnh và khối lệnh;

- Trình bày được phạm vi hoạt động của biến và mảng;

4.1. Lệnh:

Một biểu thức kiểu như x=0 hoặc ++i hoặc scanf(...) trở thành câu lệnh khi có đi kèm theo dấu ;

Ví dụ :

```
x=0;
++i;
scanf(...);
```

Trong chương trình C, dấu ; là dấu hiệu kết thúc câu lệnh.

4.2. Khối lệnh :

Một dãy các câu lệnh được bao bởi các dấu { } gọi là một khối lệnh. Ví dụ :

```
{
    a=2;
    b=3;
    printf("\n%6d%6d",a,b);
}
```

TURBO C xem khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác, chỗ nào viết được một câu lệnh thì ở đó cũng có quyền đặt một khối lệnh.

Khai báo ở đầu khối lệnh :

Các khai báo biến và mảng chẳng những có thể đặt ở đầu của một hàm mà còn có thể viết ở đầu khối lệnh :

```
{
    int a,b,c[50];
    float x,y,z,t[20][30];
    a==b==3;
    x=5.5; y=a*x;
    z=b*x;
    printf("\n y= %8.2f\n z=%8.2f",y,z);
}
```

Sự lồng nhau của các khối lệnh và phạm vi hoạt động của các biến và mảng:

Bên trong một khối lệnh lại có thể viết lồng khối lệnh khác. Sự lồng nhau theo cách như vậy là không hạn chế.

Khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại trong thời gian máy làm việc bên trong khối lệnh và chúng lập tức biến mất ngay sau khi máy ra khỏi khối lệnh. Vậy :

Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó.

ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh

Nếu bên trong một khối ta dùng một biến hay một mảng có tên là a thì điều này không làm thay đổi giá trị của một biến khác cũng có tên là a (nếu có) được dùng ở đâu đó bên ngoài khối lệnh này.

Nếu có một biến đã được khai báo ở ngoài một khối lệnh và không trùng tên với các biến khai báo bên trong khối lệnh này thì biến đó cũng có thể sử dụng cả bên trong cũng như bên ngoài khối lệnh.

Ví dụ :

Xét đoạn chương trình sau :

```
 {
        int a=5,b=2;
          {
                int a=4;
                b=a+b;
                printf("\n a trong =%3d  b=%3d",a,b);
           }
        printf("\n a ngoai =%3d  b=%3d",a,b);
 }
```

Khi đó đoạn chương trình sẽ in kết quả như sau :

a trong =4 b=6

a ngoài =5 b=6

Do tính chất biến a trong và ngoài khối lệnh.

5. Lệnh gán, lệnh xuất nhập, lệnh gán kết hợp

Mục tiêu:

- Trình bày được cú pháp của các lệnh gán, lệnh xuất nhập;

5.1. 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 =10; /*Gán hằng số 10 cho biến x*/
            y = 2*x; /*Gán giá trị 2*x=2*10=20 cho x*/
            return 0;
          }
```

Nguyên tắc khi dùng lệnh gán là 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 = "Xin chao";
            /*y có kiểu int, còn "Xin chao" có kiểu char* */
```

```
    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.

Tuy nhiên trong đa số trường hợp sự tự động biến đổi kiểu để sự tương thích về kiểu sẽ được thực hiện.

Ví dụ:

```
int main() {
    int x,y;
    float r;
    char ch;
    r = 9000;
    x = 10; /* Gán hằng số 10 cho biến x */
    y = 'd'; /* y có kiểu int, còn ‘d’ có kiểu char*/
    r = 'e'; /* r có kiểu float, ‘e’ có kiểu char*/
    ch = 65.7; /* ch có kiểu char, còn 65.7 có kiểu float*/
    return 0;
}
```

Trong nhiều trường hợp để tạo ra sự tương thích về kiểu, ta phải sử dụng đến cách thức chuyển đổi kiểu một cách tường minh. Cú pháp của phép toán này như sau:

(Tên kiểu) <Biểu thức>

Chuyển đổi kiểu của <Biểu thức> thành kiểu mới <Tên kiểu>. Chẳng hạn như:

```
float f;
f = (float) 10 / 4; /* f lúc này là 2.5*/
```

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;

2.5.2. Lệnh xuất,nhập

a. Lệnh nhập

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("Chuỗi định dạng", địa chỉ của các biến);

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 nhập kiểu số nguyên, số thực, ký tự.

Định dạng	Ý nghĩa
%[số ký số]d	Nhập số nguyên có tối đa <số ký số>
%[số ký số] f	Nhập số thực có tối đa <số ký số> tính cả dấu chấm
%c	Nhập một ký tự
Ví dụ:	
%d	Nhập số nguyên
%4d	Nhập số nguyên tối đa 4 ký số, nếu nhập nhiều hơn 4 ký số thì chỉ nhận được 4 ký số đầu tiên
%f	Nhập số thực
%6f	Nhập số thực tối đa 6 ký số (tính luôn dấu chấm), nếu nhập nhiều hơn 6 ký số thì chỉ nhận được 6 ký số đầu tiên (hoặc 5 ký số với dấu chấm)

- 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 thực*/
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:

int biennguyen;
float bienthuc;
char bienchar;
char chuoi1[20], *chuoi2;
Nhập giá trị cho các biến:
scanf("%3d",&biennguyen);

Nếu ta nhập 1234455 thì giá trị của biennguyen là 3 ký số đầu tiên (123). Các ký số còn lại sẽ còn nằm lại trong vùng đệm.

scanf("%5f",&bienthuc);

Nếu ta nhập 123.446 thì giá trị của bienthuc là 123.4, các ký số còn lại sẽ còn nằm trong vùng đệm.

scanf("%2d%5f",&biennguyen, &bienthuc);

Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 1223 3.142325

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
- 2 ký số tiếp theo trước khoảng trắng (23) sẽ được đọc vào cho bienthuc.

scanf("%2d%5f%c",&biennguyen, &bienthuc,&bienchar)

Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 12345 3.142325:

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
- 3 ký số tiếp theo trước khoảng trắng (345) sẽ được đọc vào cho bienthuc.
- Khoảng trắng sẽ được đọc cho bienchar.

Nếu ta chỉ nhập 1 số gồm nhiều ký số như sau: 123456789:

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
- 5 ký số tiếp theo (34567) sẽ được đọc vào cho bienthuc.
- bienchar sẽ có giá trị là ký số tiếp theo '8'.

scanf("%s",chuoi1); hoặc scanf("%s",chuoi2)

thì Nếu ta nhập chuỗi như sau: Nguyen Van Linh giá trị của biến chuoi1 hay chuoi2 chỉ là Nguyen .

scanf("%s%s",chuoi1, chuoi2);

thì giá Nếu ta nhập chuỗi như sau: Duong Van Hieu trị của biến chuoi1 là Duong và giá trị của biến chuoi2 là Van.

Vì sao như vậy? C sẽ đọc từ đầu đến khi gặp khoảng trắng và gán giá trị cho biến đầu tiên, phần còn lại sau khoảng trắng là giá trị của các biến tiếp theo.

gets(chuoi1);

Nếu nhập chuỗi : Nguyen Van Linh thì giá trị của biến chuoi1 là Nguyen Van Linh

b. Lệnh xuất:

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.

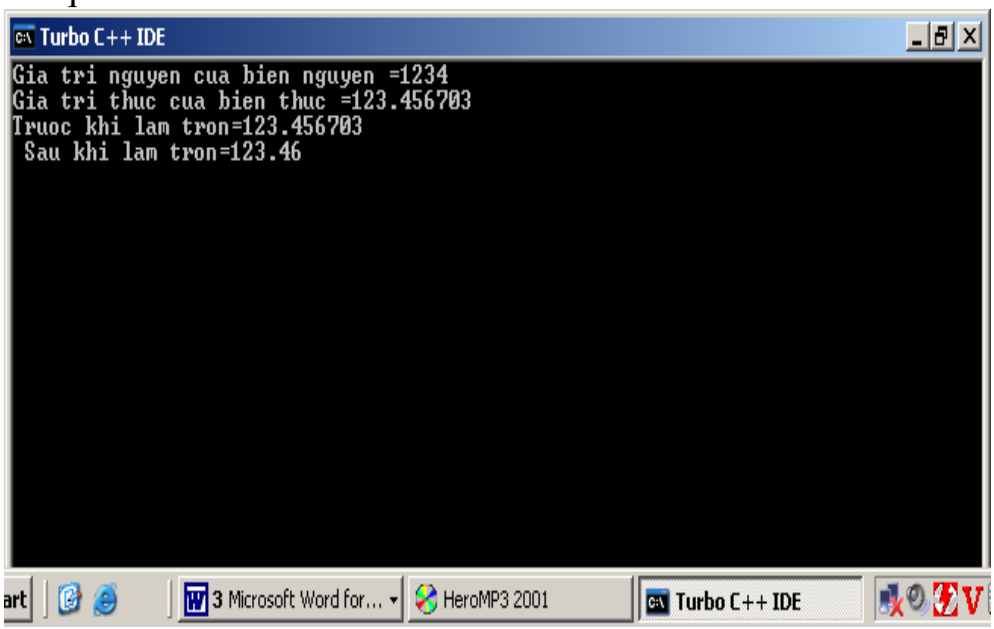
Định dạng	Ý nghĩa
%d	Xuất số nguyên
%[.số chữ số thập phân] f	Xuất số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
%o	Xuất số nguyên hệ bát phân
%x	Xuất số nguyên hệ thập lục phân
%c	Xuất một ký tự
%s	Xuất chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	Xuất số nguyên dạng khoa học (nhân 10 mũ x)
Ví dụ	
%d	In ra số nguyên
%4d	In số nguyên tối đa 4 ký số, nếu số cần in nhiều hơn 4 ký số thì in hết
%f	In số thực
%6f	In số thực tối đa 6 ký số (tính luôn dấu chấm), nếu số cần in nhiều hơn 6 ký số thì in hết
%.3f	In số thực có 3 số lẻ, nếu số cần in có nhiều hơn 3 số lẻ thì làm tròn.

- 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:



Hình 2-1

Nếu ta thêm vào dòng sau trong chương trình:

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

Kết quả ta nhận được thêm:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

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

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

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

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

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

Kết quả in ra màn hình:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

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.

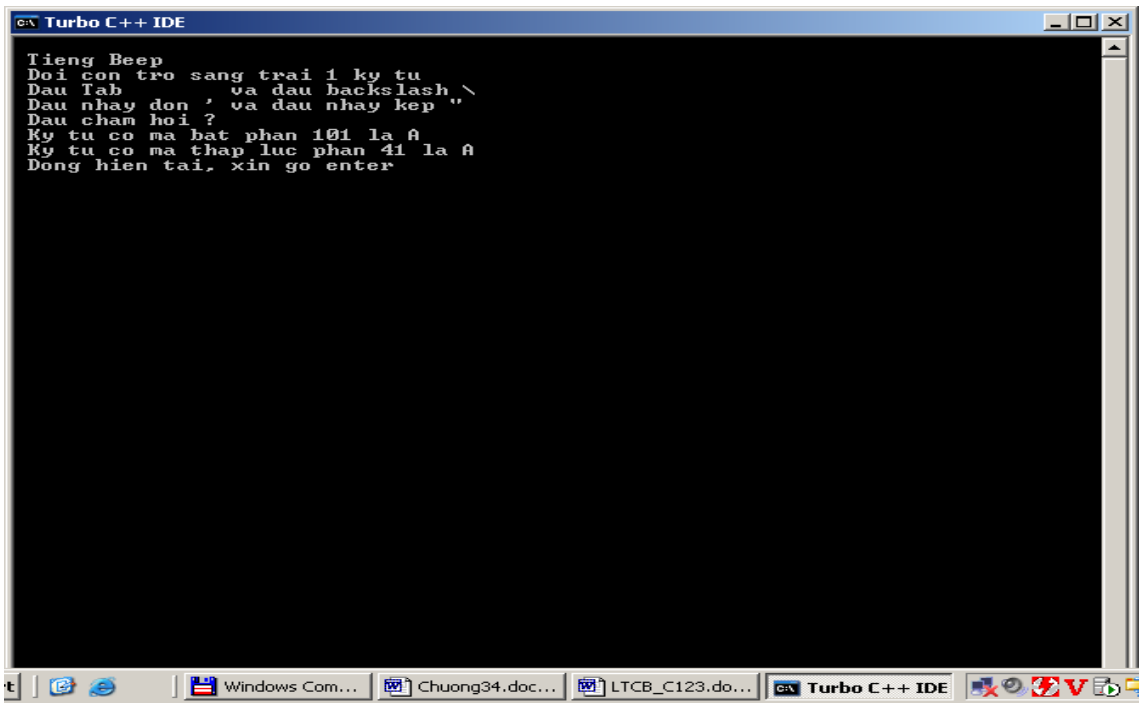
Một số ký tự điều khiển được mô tả trong bảng:

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	?	Đấu 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:

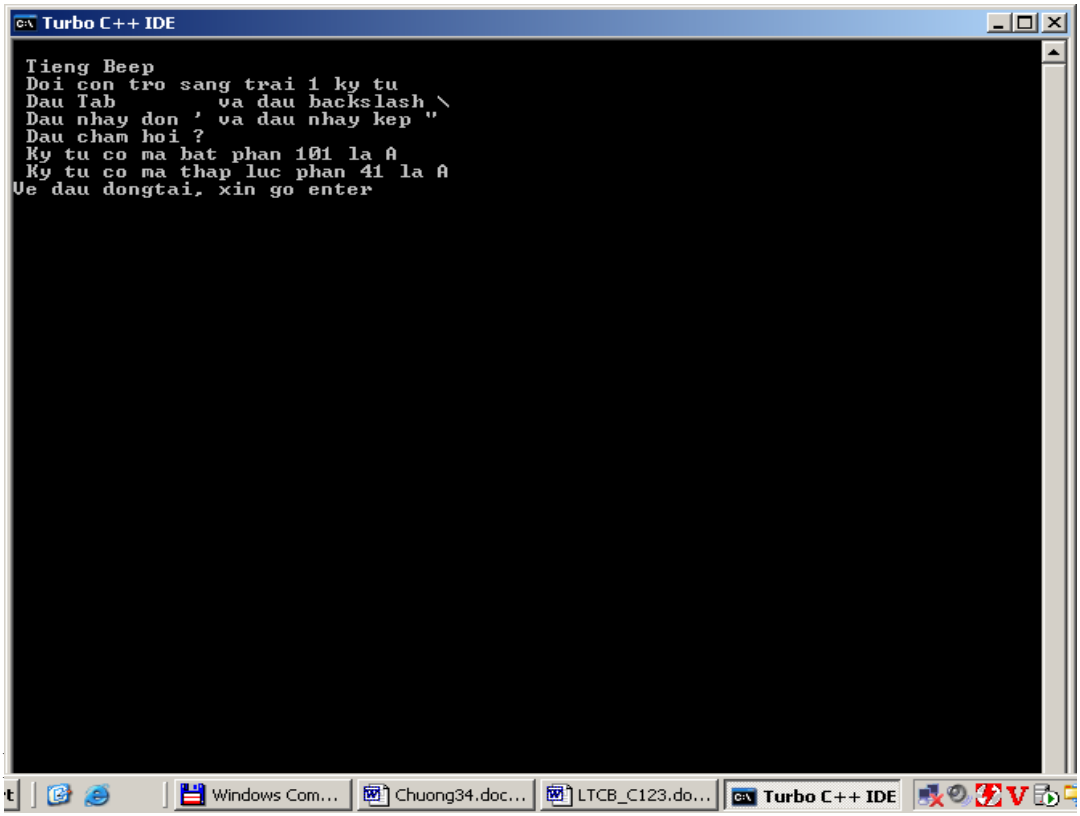
```
#include <stdio.h>
#include<conio.h>
int main ()
{ clrscr();
        printf("\n Tieng Beep \a");
        printf("\n Doi con tro sang trai 1 ky tu\b");
        printf("\n Dau Tab \tva dau backslash \\");
        printf("\n Dau nhay don \' va dau nhay kep \"");
        printf("\n Dau cham hoi \?");
        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");
        getch();
        return 0;
}
```

Kết quả trước khi gõ phím Enter:



Hình 2-2

Kết quả sau khi gõ phím Enter:



6. Các phép toán

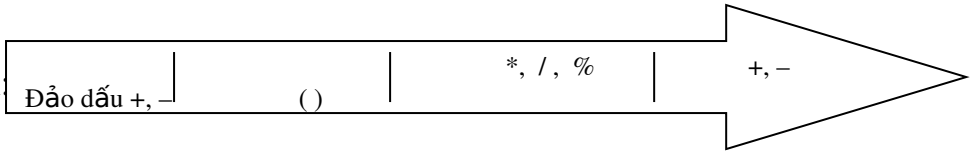
Mục tiêu:

- Trình bày được các phép toán số học, các phép toán quan hệ;

Phép toán số học

+ : cộng – : trừ * : nhân / : chia	áp dụng trên tất cả các toán hạng có kiểu dữ liệu char, int float, double (kể cả long, short, unsigned)
% : lấy phần dư	áp dụng trên các toán hạng có kiểu dữ liệu char, int, long

Thứ tự ưu tiên:



Ví dụ :

10%4 = 2 (10 chia 4 dư 2); 9%3 = 0 (9 chia 3 dư 0)

3 * 5 + 4 = 19

6 + 2 / 2 – 3 = 4

–7 + 2 * ((4 + 3) * 4 + 8) = 65

) chỉ sử dụng cặp ngoặc () trong biểu thức, cặp ngoặc đơn được thực hiện theo thứ tự ưu tiên từ trong ra ngoài.

- **Phép quan hệ**

> : lớn hơn

>= : lớn hơn hoặc bằng

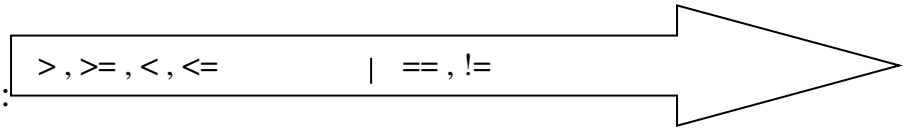
< : nhỏ hơn

<= : nhỏ hơn hoặc bằng

== : bằng

!= : khác

* Thứ tự ưu tiên:



Kết quả của phép toán quan hệ là số nguyên kiểu int, bằng 1 nếu đúng, bằng 0 nếu sai. Phép toán quan hệ ngoài toán hạng được sử dụng là số còn được sử dụng với kiểu dữ liệu char.

* Thứ tự ưu tiên giữa toán tử số học và toán tử quan hệ

Toán tử số học Toán tử quan hệ

Ví dụ :

4 > 10 → có giá trị 0 (sai)

4 >= 4 → có giá trị 1 (đúng)
3 == 5 → có giá trị 0 (sai)
2 <= 1 → có giá trị 0 (sai)
6 != 4 → có giá trị 1 (đúng)
6 – 3 < 4 → có giá trị 1 (đúng), tương đương (6 – 3) < 4
–2 * –4 < 3 + 2 → có giá trị 0 (sai), tương đương (–2 * –4) < (3 + 2)

7. Cách chạy chương trình

Mục tiêu:

- Dịch và chạy được chương trình C

Ctrl – F9: Dịch và chạy chương trình**. Alt – F5:** Xem màn hình kết quả.

BÀI 3
CÁC LỆNH CÓ CẤU TRÚC

Giới thiệu:

Bài này sẽ cung cấp cho người học những kiến thức về

- ✓ Cú pháp, chức năng của các lệnh cấu trúc: cấu trúc lựa chọn, cấu trúc lặp xác định và lặp vô định.
- ✓ Các lệnh cấu trúc.
- ✓ Các lệnh bẻ vòng lặp

Mục tiêu:

- Hiểu và vận dụng được các lệnh cấu trúc**:** cấu trúc lựa chọn, cấu trúc lặp xác định và lặp vô định.
- Hiểu và vận dụng được các lệnh bẻ vòng lặp
- Rèn luyện thói quen suy luận logic.

1. Lệnh rẽ nhánh có điều kiện if

Mục tiêu:

- Nêu được ý nghĩa của lệnh rẽ nhánh if;
- Vận dụng được lệnh rẽ nhánh if vào bài tập lập trình;

1.1.Ý nghĩa

Một câu lệnh **if** cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào một điều kiện được viết trong câu lệnh là đúng hay sai. Nói cách khác câu lệnh **if** cho phép chương trình rẽ nhánh (chỉ thực hiện 1 trong 2 nhánh).

1.2.Cú pháp

- **if (điều kiện) { khối lệnh 1; } else { khối lệnh 2; }**
- **if (điều kiện) { khối lệnh 1; }**

Trong cú pháp trên câu lệnh if có hai dạng: có else và không có else. điều kiện là một biểu thức lôgic tức nó có giá trị đúng (khác 0) hoặc sai (bằng 0).

Khi chương trình thực hiện câu lệnh if nó sẽ tính biểu thức điều kiện. Nếu điều kiện đúng chương trình sẽ tiếp tục thực hiện các lệnh trong khối lệnh 1, ngược lại nếu điều kiện sai chương trình sẽ thực hiện khối lệnh 2 (nếu có else) hoặc không làm gì (nếu không có else).

1.3.Đặc điểm

Đặc điểm chung của các câu lệnh có cấu trúc là bản thân nó chứa các câu lệnh khác. Điều này cho phép các câu lệnh if có thể lồng nhau.

Nếu nhiều câu lệnh if (có else và không else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Qui tắc là else sẽ đi với if gần nó nhất mà chưa được ghép cặp với else khác. Ví dụ câu lệnh

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

là tương đương với

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

1.4.Ví dụ minh hoạ

Ví dụ 1 : Bằng phép toán gán có điều kiện có thể tìm số lớn nhất max trong 2 số a, b như sau: max = (a > b) ? a: b ;

hoặc max được tìm bởi dùng câu lệnh if:

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

Ví dụ 2 : Tính năm nhuận. Năm thứ n là nhuận nếu nó chia hết cho 4, nhưng không chia hết cho 100 hoặc chia hết 400. Chú ý: một số nguyên a là chia hết cho b nếu phần dư của phép chia bằng 0, tức a%b == 0.

```
#include <iostream.h>
void main()
{
   int nam;
   cout << "Nam = " ; cin >> nam ;
   if (nam%4 == 0 && year%100 !=0 || nam%400 == 0)
         cout << nam << "la nam nhuan" ;
      else
         cout << nam << "la nam khong nhuan" ;
}
```

Ví dụ 3 : Giải phương trình bậc 2. Cho phương trình $ax^2 + bx + c = 0$ (a ,, 0),

tìm x.

```
#include <iostream.h>                    // tệp chứa các phương thức vào/ra
#include <math.h>                        // tệp chứa các hàm toán học
void main()
{
    float a, b, c;                       // khai báo các hệ số
    float delta;
  float x1, x2;                          // 2 nghiem
  cout << "Nhap a, b, c:\n" ; cin >> a >> b >> c ;      // qui ước nhập a ,, 0
  delta = b*b - 4*a*c ;
  if (delta < 0) cout << "ph. trình vô nghiệm\n" ;
  else if (delta==0) cout<<"ph. trình có nghiệm kép:" << -b/(2*a) << '\n';
  else
  {
        x1 = (-b+sqrt(delta))/(2*a);
        x2 = (-b-sqrt(delta))/(2*a);
        cout << "nghiem 1 = " << x1 << " và nghiem 2 = " << x2 ;
  }
}
```

Chú ý: do C++ quan niệm "đúng" là một giá trị khác 0 bất kỳ và "sai" là giá trị 0 nên thay vì viết if (x != 0) hoặc if (x == 0) ta có thể viết gọn thành if (x) hoặc if (!x) vì nếu (x != 0) đúng thì ta có x ,, 0 và vì x ,, 0 nên (x) cũng đúng. Ngược lại nếu (x) đúng thì x ,, 0, từ đó (x != 0) cũng đúng. Tương tự ta dễ dàng thấy được (x == 0) là tương đương với (!x).

2. Lệnh rẽ nhánh có điều kiện switch..case

Mục tiêu:

- Sử dụng được lệnh rẽ nhánh switch ..case vào chương trình;

- Cấu trúc switch…case (switch thiếu): Chọn thực hiện 1 trong n lệnh cho trước.

Cú pháp lệnh

```
switch (biểu thức)
{
case giá trị 1 : lệnh 1;
                 break;
case giá trị 2 : lệnh 2;
                 break;
…
case giá trị n : lệnh n;
                [break;]
```

từ khóa **switch, case, break**
phải viết bằng chữ thường
biểu thức phải là có kết quả là
giá trị hằng nguyên (char, int, long,…)
Lệnh 1, 2…n có thể gồm nhiều lệnh, nhưng không cần đặt trong cặp dấu { }

Khi giá trị của biểu thức bằng giá trị i thì lệnh i sẽ được thực hiện. Nếu sau lệnh i không có lệnh break thì sẽ tiếp tục thực hiện lệnh i + 1…Ngược lại thoát khỏi cấu trúc switch.

3. Cấu trúc switch…case…default (switch đủ):

Mục tiêu:

- *Trình bày được cú pháp của cấu trúc switch …case…default;*
- *Sử dụng được các câu lệnh break, continue, goto;*

Chọn thực hiện 1 trong n + 1 lệnh cho trước.

```
switch (biểu thức)
{
case giá trị 1 : lệnh 1;
                 break;
case giá trị 2 : lệnh 2;
                 break;
…
case giá trị n : lệnh n;
                 break;
default        : lệnh;
                [break;]
```

từ khóa **switch, case, break, default** phải viết bằng chữ thường
biểu thức phải là có kết quả là **giá trị nguyên (char, int, long,…)**
Lệnh 1, 2…n có thể gồm nhiều lệnh, nhưng không cần đặt trong cặp dấu { }

Khi giá trị của biểu thức bằng giá trị i thì lệnh i sẽ được thực hiện. Nếu sau lệnh i không có lệnh break thì sẽ tiếp tục thực hiện lệnh i +1… Ngược lại thoát khỏi cấu trúc switch. Nếu giá trị biểu thức không trùng với bất kỳ giá trị i nào thì lệnh tương ứng với từ khóa default sẽ được thực hiện.

Cấu trúc switch lồng

Quyết định sẽ thực hiện 1 trong n khối lệnh cho trước.

• ***Cú pháp lệnh***

Cú pháp là một trong 2 dạng trên, nhưng trong 1 hoặc nhiều lệnh bên trong phải chứa ít nhất một trong 2 dạng trên gọi là cấu trúc switch lồng nhau. Thường cấu trúc switch lồng nhau càng nhiều cấp độ phức tạp càng cao, chương trình chạy càng chậm và trong lúc lập trình dễ bị nhầm lẫn.

3.1. Các lệnh break, continue, goto

3.1.1. Câu lệnh break

Câu lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình bên trong nhất chứa nó không cần điều kiện gì. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ví dụ :

Biết số nguyên dương n sẽ là số nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn bậc hai của n. Viết đoạn chương trình đọc vào số nguyên dương n, xem n có là số nguyên tố.

```
# include "stdio.h"
# include "math.h"
unsigned int n;
main()
        {
                int i,nt=1;
                printf("\n cho n=");
                scanf("%d",&n);
                for (i=2;i<=sqrt(n);++i)
                if ((n % i)==0)
                {
                        nt=0;
                        break;
                }
                if (nt)
                        printf("\n %d la so nguyen to",n);
                else
                        printf("\n %d khong la so nguyen to",n);
        }
```

3.1.2. Câu lệnh continue

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra, còn trong for điều khiển được chuyển về bước khởi đầu lại (tức là bước: tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình).

Chú ý :

Lệnh continue chỉ áp dụng cho chu trình chứ không áp dụng cho switch.

Ví dụ :

Viết chương trình để từ một nhập một ma trận a sau đó :

Tính tổng các phần tử dương của a.

Xác định số phần tử dương của a.

Tìm cực đại trong các phần tử dương của a.

```
#include "stdio.h"
float a[3[4];
main()
        {
                int i,j,soptd=0;
                float tongduong=0,cucdai=0,phu;
```

```
        for (i=0;i<3;++i)
        for (j=0;i<4;++j)
        {
                printf("\n a[%d][%d]=",i,j  );
                scanf("%f",&phu);
                a[i][j]=phu;
                if (a[i][j]<=0) continue;
                tongduong+=a[i][j];
                if (cucdai<a[i][j]) cucdai=a[i][j];
                ++soptd;
        }
        printf("\n So phan tu duong la : %d",soptd);
        printf("\n Tong cac phan tu duong la : %8.2f",tongduong);
        printf("\n Cuc dai phan tu duong la : %8.2f",cucdai);
}
```

3.1.3 Lệnh nhảy không điều kiện - toán tử goto:

Nhãn có cùng dạng như tên biến và có dấu: đứng ở phía sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình.

Ví dụ:

ts : s=s++;

thì ở đây **ts** là nhãn của câu lệnh gán s=s++.

Toán tử goto có dạng :

goto nhãn;

Khi gặp toán tử này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khoá goto.

Khi dùng toán tử goto cần chú ý :

Câu lệnh goto và nhãn cần nằm trong một hàm, có nghĩa là toán tử goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân một hàm và không thể dùng để nhảy từ một hàm này sang một hàm khác.

Không cho phép dùng toán tử goto để nhảy từ ngoài vào trong một khối lệnh. Tuy nhiên việc nhảy từ trong một khối lệnh ra ngoài là hoàn toàn hợp lệ. Ví dụ như đoạn chương trình sau là sai.

```
goto n1;
.......
{       .....
        n1: printf("\n Gia tri cua N la: ");
        .....
}
```

Ví dụ :

Tính tổng s=1+2+3+....+10

```
#include "stdio.h"
```

```
main()
        {
                int s,i;
                i=s=0;
                tong:
                ++i;
                s=s+i;
                if (i<10) goto tong;
                printf("\n tong s=%d",s);
        }
```

4. Cấu trúc vòng lặp For

Mục tiêu:

- *Trình bày được cú pháp của vòng lặp for;*

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau :

for (biểu thức 1; biểu thức 2; biểu thức 3)

Lệnh hoặc khối lệnh ;

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ; .

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for :

Toán tử for hoạt động theo các bước sau :

Xác định biểu thức 1

Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh

Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for.

Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Chú ý:

Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu;. Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu

thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy ra sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một ví trí mong muốn bất kỳ.

Ví dụ 1:

Nhập một dãy số rồi đảo ngược thứ tự của nó.

Cách 1:

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
        int i,j;
        float c;
        for (i=0,j=n-1;i<j;++i,--j)
        {
                c=x[i];x[i]=x[j];x[j]=c;
        }
        fprintf(stdprn,“\n Day so dao la \n\n”);
        for (i=0;i<n;++i)
        fprintf(stdprn,“%8.2f”,x[i]);
}
```

Cách 2 :

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
        {
                int i,j;
                float c;
                for (i=0,j=n-1;i<j;c=x[i],x[i]=x[j],x[j]=c,++i,--j)
                fprintf(stdprn,“\n Day so dao la \n\n”);
                for (i=0;++i<n;)
                fprintf(stdprn,“%8.2f”,x[i]);
        }
```

Cách 3 :

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
```

```
main()
{
        int i=0,j=n-1;
        float c;
        for ( ; ; )
        {
                c=x[i];x[i]=x[j];x[j]=c;
                if (++i>--j) break;
        }
        fprintf(stdprn,“\n Day so dao la \n\n”);
        for (i=-1;i++<n-1; fprintf(stdprn,“%8.2f”,x[i]));
}
```

Ví dụ 2:

Tính tích hai ma trận mxn và nxp.

```
#include "stdio.h"
float x[3][2],y[2][4],z[3][4],c;
main()
{
        int i,j;
        printf("\n nhap gia tri cho ma tran X ");
        for (i=0;i<=2;++i)
        for (j=0;j<=1;++j)
        {
                printf("\n x[%d][%d]=",i,j);
                scanf("%f",&c);
                x[i][j]=c;
        }
        printf("\n nhap gia tri cho ma tran Y ");
        for (i=0;i<=1;++i)
        for (j=0;j<=3;++j)
        {
                printf("\n y[%d][%d]=",i,j);
                scanf("%f",&c);
                y[i][j]=c;
        }
for (i=0;i<=3;++i)
for (j=0;j<=4;++j)
z[i][j]
}
```

5. Cấu trúc vòng lặp while

Mục tiêu:

- *Trình bày được cú pháp của vòng lặp while*

- *Vận dụng được vòng lặp while để làm bài tập*

Toán tử while dùng để xây dựng chu trình lặp dạng :

```
while ( biểu thức )
        Lệnh hoặc khối lệnh;
```

Như vậy toán tử while gồm một biểu thức và thân chu trình. Thân chu trình có thể là một lệnh hoặc một khối lệnh.

Hoạt động của chu trình như sau :

Máy xác định giá trị của biểu thức, tuỳ thuộc giá trị của nó máy sẽ chọn cách thực hiện như sau :

Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi chu trình và chuyển tới thực hiện câu lệnh tiếp sau chu trình trong chương trình.

Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh hoặc khối lệnh trong thân của while. Khi máy thực hiện xong khối lệnh này nó lại thực hiện xác định lại giá trị biểu thức rồi làm tiếp các bước như trên.

Chú ý :

Trong các dấu ngoặc ( ) sau while chẳng những có thể đặt một biểu thức mà còn có thể đặt một dãy biểu thức phân cách nhau bởi dấu phảy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

Bên trong thân của một toán tử while lại có thể sử dụng các toán tử while khác. bằng cách đó ta đi xây dựng được các chu trình lồng nhau.

Khi gặp câu lệnh break trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

Trong thân while có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân while để ra khỏi một hàm nào đó.

Ví dụ :

Chương trình tính tích vô hướng của hai véc tơ x và y :

Cách 1 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
    {
        float s=0;
        int i=-1;
        while (++i<4)
        s+=x[i]*y[i];
        printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
    }
```

Cách 2 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
	{
		float s=0;
		int i=0;
		while (1)
		{
			s+=x[i]*y[i];
			if (++i>=4) goto kt;
		}
		kt:printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
	}
```

Cách 3 :

```
#include "stdio.h"
float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
main()
	{
		float s=0;
		int i=0;
		while (	s+=x[i]*y[i], ++i<=3 );
		printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
	}
```

6.Cấu trúc vòng lặp do..while

Mục tiêu:

- *Nêu được cú pháp của vòng lặp do...whlie;*
- *Phân biệt được vòng lặp while và do …while;*
- *Vận dụng được vòng lặp do... while vào bài tập;*

Khác với các toán tử while và for, việc kiểm tra điều kiện kết thúc đặt ở đầu chu trình, trong chu trình do while việc kiểm tra điều kiện kết thúc đặt cuối chu trình. Như vậy thân của chu trình bao giờ cũng được thực hiện ít nhất một lần.

Chu trình do while có dạng sau :

```
do
	Lệnh hoặc khối lệnh;
while ( biểu thức );
```

Lệnh hoặc khối lệnh là thân của chu trình có thể là một lệnh riêng lẻ hoặc là một khối lệnh.

Hoạt động của chu trình như sau :

Máy thực hiện các lệnh trong thân chu trình.

Khi thực hiện xong tất cả các lệnh trong thân của chu trình, máy sẽ xác định giá trị của biểu thức sau từ khoá while rồi quyết định thực hiện như sau :

Nếu biểu thức đúng (khác 0) máy sẽ thực hiện lặp lại khối lệnh của chu trình lần thứ hai rồi thực hiện kiểm tra lại biểu thức như trên.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc chu trình và chuyển tới thực hiện lệnh đứng sau toán tử while.

Chú ý :

Những điều lưu ý với toán tử while ở trên hoàn toàn đúng với do while.

Ví dụ :

Đoạn chương trình xác định phần tử âm đầu tiên trong các phần tử của mảng x.

```
#include "stdio.h"
float x[5],c;
main()
	{
		int i=0;
		printf("\n nhap gia tri cho ma tran x ");
		for (i=0;i<=4;++i)
		{
			printf("\n x[%d]=",i);
			scanf("%f",&c);
			y[i]=c;
		}
		do
			++i;
		while (x[i]>=0 && i<=4);
		if (i<=4)
			printf("\n Phan tu am dau tien = x[%d]=%8.2f",i,x[i]);
		else
			printf("\n Mang khong có phan tu am ");
	}
```

B. PHẦN BÀI TẬP

Bài 1: Viết hàm đổi một ký tự hoa sang ký tự thường

Bài 2: Viết chương trình giải phương trình bậc nhất.

Bài 3 :Viết chương trình giải phương trình bậc hai.

Bài 4: Viết chương trình trả về giá trị nhỏ nhất của 4 số nguyên.

Bài 5. Viết chương trình kiểm tra một số được nhập vào từ bàn phím có phải là số chính phương không.

Bài6. nhập số nguyên n kiểm tra Có phải là số nguyên tố hay không.

Bài 7. Tổng các chữ số lẻ từ n chữ số nhập từ bàn phím.

PHẦN HƯỚNG DẪN LÀM BÀI TẬP

Bài 1: Viết hàm đổi một ký tự hoa sang ký tự thường

```
//Chuong trinh chuyen tu ki tu hoa song ki tu thuong
#include<stdio.h>
#include<conio.h>
int main()
{
char c;
printf("Nhap mot chu cai bat ki: %\n");
scanf("%c",&c);
if ((65<=c) and (c<=90)){c=c+32;printf("Ki tu da chuyen doi la %c\n",c);}
else printf("Ban vua nhap moi ki tu thuong \n");
getch();
}
```

//Trong bo ma ASCII thì kí tu hoa co ma tu 65 den 90, con ki tu thuong bang ki tu hoa cong them 32

Bài 2: Viết chương trình giải phương trình bậc nhất.

PHP Code:

```
//Chuong trinh giai phuong trinh bac nhat

#include<stdio.h>
#include<conio.h>
int main()
{
float a,b;
float x;
printf("nhap a vao: \n"); scanf("%f",&a);
printf("nhap b vao: \n"); scanf("%f",&b);
if (a!=0) {x=-b/a; printf("phuong trinh co nghiem %3.2f\n",x);}
else if (b==0) printf("Phuong trinh vo so nghiem \n");
else printf("phuong trinh vo nghiem");getch();
}
```

Bài 3 :Viết chương trình giải phương trình bậc hai

```
//Chuong trinh giai phuong trinh bac 2

#include<stdio.h>
```

```
#include<conio.h>
#include<math.h>
int main()
{
float a,b,c,d,x,x1,x2,k;
printf("nhap a vao: \n"); scanf("%f",&a);
printf("nhap b vao: \n"); scanf("%f",&b);
printf("nhap c vao: \n"); scanf("%f",&c);
if (a==0)
if (b==0)
if (c==0) printf("Phuong trinh vo so nghiem");
else printf("phuong trinh vo nghiem");
else {x=-c/b; printf("phuong trinh co nghiem la: %3.2f\n",x);}
else {
d=b*b-4*a*c;
if (d==0) printf("phuong trinh co nghiem kep: %f",x=-b/2/a);
else if (d<0) printf("phuong trinh vo nghiem");
else
{
k=sqrt(d);
x1=(-b-k)/2/a;
x2=(-b+k)/2/a;
printf("puong trinh co hai nghiem phan biet:\n x1=%f\n x2=%f\n",x1,x2);
}
}

getch();
}
```

Bài 4: Viết chương trình trả về giá trị nhỏ nhất của 4 số nguyên

```
//Chuong trinh tim gia tri nho nhat cua bon so nguyen

#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c,d,min,x,y;
printf("nhap a: \n"); scanf("%d",&a);
printf("nhap b: \n"); scanf("%d",&b);
if (a<b) x=a; else x=b;
printf("nhap c: \n"); scanf("%d",&c);
printf("nhap d: \n"); scanf("%d",&d);
```

```
if (c<d) y=c; else y=d;
if (x<y) min=x; else min=y;
printf("Gia tri nho nhat la %3d",min);
getch();
}
```

Viết thủ tục sắp xếp 4 số nguyên tăng dần.

PHP Code:

```
//Chuong trinh nhap 4 so nguyen va dua ra man hinh thu tu tang dan ;

#include<stdio.h>
#include<conio.h>
main()
{
int a[10],i,tg,j=0;
for(i=0;i<4;i++)
{
printf("\nNhap so thu: %d\n",i+1); scanf("%d",&a[i]);
}
for (i=0;i<4;i++) //Duyet qua tat ca phan tu cua mang
{
for (j=i+1;j<4;j++) //Duyet tat ca phan tu cua mang dung sau i
{
if (a[i]>a[j])
{
tg=a[i]; //Vi a[i] duoc duyet truoc a[j] nen gan tg=a[i];
a[i]=a[j]; /*Ba phep gan nay giong nhau ke ca voi xap xep tang*/
a[j]=tg;
}
}
}
printf("\nDay so sau khi xap xem la: ");
for (i=0;i<4;i++) printf("%d ",a[i]);
getch();
}
```

Bài 5. Viết chương trình kiểm tra một số được nhập vào từ bàn phím có phải là số chính phương không.
Kiểm tra số chính phương

Yêu cầu:Thế nào là số chính phương: Những số nguyên dương mà có căn bậc hai là số nguyên dương là số chính phương;

Ý tưởng: Nhập vào số nguyên dương n bất kì; bây giờ mình kiểm tra các số từ (1-n) nếu mà trong các bình phương vừa kiểm tra có số nào bằng n thì n là số chính phương
ví dụ như số 9 bây giờ mình kiểm tra từ 1-9thì thấy có số 3 sau khi bình phương bằng 9;
giả sử nhập n=5 thì kiểm tra từ 1-5 không thấy số nào bình phương bằng 5 cả

//Chuong trinh nhap mot so nguyen va in ket qua co phai la so chinh phuong hay khong

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
int i,n,s;
float c;
printf("Ban hay nhap mot so nguyen: \n"); scanf("%d",&n);
if (n<=0) printf("So ",n," khong phai so chinh phuong \n"); else
{
for (i=1;i<=n;i++)
{
if(i*i==n) s=1;
}
if(s==1) printf("so ban nhap la so chinh phuong");
else printf("so ban nhap khong la so chinh phuong");
}
getch();
}
```

Bài6. nhập số nguyên n kiểm tra Có phải là số nguyên tố hay không

Thứ 1: Thế nào là số nguyên tố: là số nguyên dương chỉ có 2 ước là 1 và chính nó;
Thứ 2: Vận dụng số ước của số nguyên tố để giải quyết bài toán này bằng cách sử dụng biến s đếm số ước của n khi lấy n chia cho các số từ 1-n:
+ Nếu có 2 ước thì kết luận số nguyên tố; còn nếu khác 2 thì kết luận ngược lại
+ Sử dụng vòng lặp for để giải quyết bài toán:

//Kiem tra so nguyen to

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,n,s=0;
lap:
printf("nhap n: \n"); scanf("%d",&n);
s=0;
for (i=1;i<=n;i++)
{
if (n%i==0) s=s+1;}
if (s==2) printf("so nguyen so \n"); else printf("khong nguyen to \n");
printf("Nhan phim bat ki de tiep tuc \n");
getch();
goto lap;
getch();
}
```

Bài 7. Tổng các chữ số lẻ từ n chữ số nhập từ bàn phím. (Tổnng số chẵn tương tự)

```
//Chuong trinh nhap mot so nguyen n va tinh tong cac so le

#include<stdio.h>
#include<conio.h>
main()
{
unsigned int i,n,tl;
printf("Nhap n vao: \n"); scanf("%x",&n);
tl=0;
for (i=1;i<=n;i++)
{
if(i%2!=0) tl=tl+i;
}
printf("Tong le cua can tinh la: %5x\n",tl);
getch();
}
```

BÀI 4 HÀM

Mã bài: MĐ 11- 04

Giới thiệu:

Bài này sẽ cung cấp cho người học những kiến thức sau:

- ✓ Khái niệm hàm.
- ✓ Quy tắc xây dựng hàm.
- ✓ Tham số, tham trị và so sánh tham số và tham trị.
- ✓ Truyền tham số cho hàm
- ✓ Các lệnh kết thúc và lấy giá trị trả về cho hàm

Mục tiêu:

- Trình bày được qui tắc xây dựng hàm và vận dụng được khi thiết kế xây dựng chương trình
- Hiểu được nguyên tắc xây dựng hàm, thế nào là tham số, tham trị
- Biết cách truyền tham số đúng cho hàm
- Sử dụng được các lệnh kết thúc và lấy giá trị trả về của hàm.
- Rèn luyện tính cách tận dụng tài nguyên có sẵn.

1. Khái niệm hàm

Mục tiêu:

- *Trình bày được khái niệm hàm;*

Hàm là một chương trình con thực hiện một khối công việc được lặp đi lặp lại nhiều lần trong khi chạy chương trình hoặc dùng tách một khối công việc cụ thể để chương trình đỡ phức tạp.

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó có một hàm chính (hàm main()). Hàm chia các bài toán lớn thành các công việc nhỏ hơn, giúp thực hiện những công việc lặp lại nào đó một cách nhanh chóng mà không phải viết lại đoạn chương trình. Thứ tự các hàm trong chương trình là bất kỳ, song chương trình bao giờ cũng đi thực hiện từ hàm main().

Ví dụ 1:

Dòng	File Edit Search Run Compile Debug Project Option Window Help
1	#include <stdio.h>
2	#include <conio.h>
3	
4	// khai bao prototype
5	void line();
6	
7	// ham in 1 dong dau
8	void line()
9	{
10	int i;
11	for(i = 0; i < 19; i++)
12	printf("*");
13	printf("\n");
14	}
15	
16	void main(void)
17	{
18	line();
19	printf("* Minh hoa ve ham *");
20	line();
21	getch();
22	}
	F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt - F9 Compile F9 Make F10 Menu

Kết quả in ra màn hình

```
******************
* Minh hoa ve ham *
******************

_
```

Giải thích chương trình

- Dòng 8 đến dòng 14: định nghĩa hàm line, hàm này không trả về giá trị, thực hiện công việc in ra 19 dấu sao.

- Dòng 5: khai báo prototype, sau tên hàm phải có dấu chầm phẩy

Trong hàm line có sử dụng biến i, biến i là biến cục bộ chỉ sử dụng được trong phạm vi hàm line.

- Dòng 18 và 20: gọi thực hiện hàm line.

**** Trình tự thực hiện chương trình:***

```
void main(void)
{
 line();
 printf("* Minh hoa ve ham
*");
 line();
 getch();
}
```

```
void line()
 {
  int i;
    for(i = 0; i < 19; i+
+)
   printf("*");
  printf("\n");
 }
```

- Gọi thực hiện hàm line
- Quay về chương trình chính
- Thực hiện lệnh kế tiếp

-> Không có dấu chấm phẩy sau tên hàm, phải có cặp dấu ngoặc ( ) sau tên hàm nếu hàm không có tham số truyền vào. Phải có dấu chấm phẩy sau tên hàm khai báo prototype. Nên khai báo prototype cho dù hàm được gọi nằm trước hay sau câu lệnh gọi nó.

2. Quy tắc xây dựng một hàm

Mục tiêu:

- *Nêu được quy tắc xây dựng hàm;*

Hàm có thể xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không có phép xây dựng một hàm bên trong các hàm khác.

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau :

```
type tên hàm ( khai báo các đối )
{
        Khai báo các biến cục bộ
        Các câu lệnh
        [return[biểu thức];]
}
```

Dòng tiêu đề :

Trong dòng đầu tiên của hàm chứa các thông tin về : kiểu hàm, tên hàm, kiểu và tên mỗi đối.

Ví dụ :

```
        float max3s(float a, float b, float c)
```

khai báo các đối có dạng :

Kiểu đối 1 tên đối 1, kiểu đối 2 tên đối 2,..., kiểu đối n tên đối n

Thân hàm :

Sau dòng tiêu đề là thân hàm. Thân hàm là nội dung chính của hàm bắt đầu và kết thúc bằng các dấu { }.

Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đã đề ra cho hàm.

Thân hàm có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở các chỗ khác nhau, và cũng có thể không sử dụng câu lệnh này.

Dạng tổng quát của nó là :

return [biểu thức];

Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

Ví dụ :

Xét bài toán : Tìm giá trị lớn nhất của ba số mà giá trị mà giá trị của chúng được đưa vào bàn phím.

Xây dựng chương trình và tổ chức thành hai hàm : Hàm main() và hàm max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số đọc vào, giả sử là a,b,c. Nhiệm vụ của hàm main() là đọc ba giá trị vào từ bàn phím, rồi dùng hàm max3s để tính như trên, rồi đưa kết quả ra màn hình.

Chương trình được viết như sau :

```
#include "stdio.h"
float max3s(float a,float b,float c ); /* Nguyên mẫu hàm*/
main()
        {
                float x,y,z;
                printf("\n Vao ba so x,y,z:");
                scanf("%f%f%f",&x&y&z);
                printf("\n Max cua ba so x=%8.2f y=%8.2f z=%8.2f la:
%8.2f",
                x,y,z,max3s(x,y,z));
        }       /* Kết thúc hàm main*/
float max3s(float a,float b,float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
                if (max<c) max=c;
                return(max);
        } /* Kết thúc hàm max3s*/
```

3. Sử dụng hàm

Mục tiêu:

- *Áp dụng được hàm vào trong các chương trình lập trình;*

Nói chung, các hàm được sử dụng trong C để thực thi một chuỗi các lệnh liên tiếp. Tuy nhiên, cách sử dụng các hàm thì không giống với các vòng lặp. Các vòng lặp có thể lặp lại một chuỗi các chỉ thị với các lần lặp liên tiếp nhau. Nhưng việc gọi một hàm sẽ sinh ra một chuỗi các chỉ thị được thực thi tại vị trí bất kỳ trong chương trình. Các hàm có thể được gọi nhiều lần khi có yêu cầu. Giả sử một phần của mã lệnh trong một chương trình dùng để tính tỉ lệ phần trăm cho một vài con số. Nếu sau đó, trong cùng chương trình, việc tính toán như vậy cần phải thực hiện trên những con số khác, thay vì phải viết lại các chỉ thị giống như trên, một hàm có thể được viết ra để tính tỉ lệ phần trăm của bất kỳ các con số. Sau đó chương trình có thể nhảy đến hàm đó, để thực hiện việc tính toán (trong hàm) và trở về nơi nó đã được gọi. Điều này sẽ được giải thích rõ ràng hơn khi thảo luận về cách hoạt động của các hàm.

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

4. Nguyên tắc hoạt động của hàm

Mục tiêu:

- *Trình bày được các tham số thực, các đối và biến cục bộ;*

Một cách tổng quát lời gọi hàm có dạng sau :

tên hàm ([Danh sách các tham số thực])

Số các tham số thực tế thay vào trong danh sách các đối phải bằng số tham số hình thức và lần lượt chúng có kiểu tương ứng với nhau.

Khi gặp một lời gọi hàm thì nó sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau :

Cấp phát bộ nhớ cho các biến cục bộ.

Gán giá trị của các tham số thực cho các đối tương ứng.

Thực hiện các câu lệnh trong thân hàm.

Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối, biến cục bộ và ra khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

Các tham số thực, các đối và biến cục bộ :

Do đối và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối và biến cục bộ cần có tên khác nhau.

Đối và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được xét đến và bị xoá khi ra khỏi hàm nên ta không thể mang giá trị của đối ra khỏi hàm.

Đối và biến cục bộ có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gán cho các đối (trong ví dụ trên hàm max3s, các tham số thực là x,y,z, các đối tương ứng là a,b,c). Như vậy các đối chính là các bản sao của các tham số thực. Hàm chỉ làm việc trên các đối.

Các đối có thể bị biến đổi trong thân hàm, còn các tham số thực thì không bị thay đổi.

Chú ý :

Khi hàm khai báo không có kiểu ở trước nó thì nó được mặc định là kiểu int.

Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nói chung nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm hay tự động việc chuyển dạng.

Nguyên mẫu của hàm thực chất là dòng đầu tiên của hàm thêm vào dấu ;. Tuy nhiên trong nguyên mẫu có thể bỏ tên các đối.

Hàm thường có một vài đối. Ví dụ như hàm max3s có ba đối là a,b,c. cả ba đối này đều có giá trị float. Tuy nhiên, cũng có hàm không đối như hàm main.

Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị các đối.

5. Cách truyền tham số

Mục tiêu:

- Truyền được tham số cho các chương trình đơn giản;

- Trình bày được hàm đệ quy;

- Ví dụ minh họa

Sử dụng các tham số trong hàm

Các tham số được sử dụng để truyền thông tin đến hàm. Các chuỗi định dạng và danh sách các biến được đặt bên trong cặp dấu ngoặc () của hàm là các tham số.

Một hàm được định nghĩa với một tên hàm theo sau là dấu ngoặc mở (sau đó là các tham số và cuối cùng là dấu ngoặc đóng). Bên trong hàm, có thể có một hoặc nhiều câu lệnh. Ví dụ,

```
calculatesum (int x, int y, int z)
    {
        statement 1;
```

```
        statement 2;
        statement 3;
}
```

Xem chương trình hoàn thiện sau.

1. **Tạo một tập tin mới.**

2. **Nhập vào mã lệnh sau:**

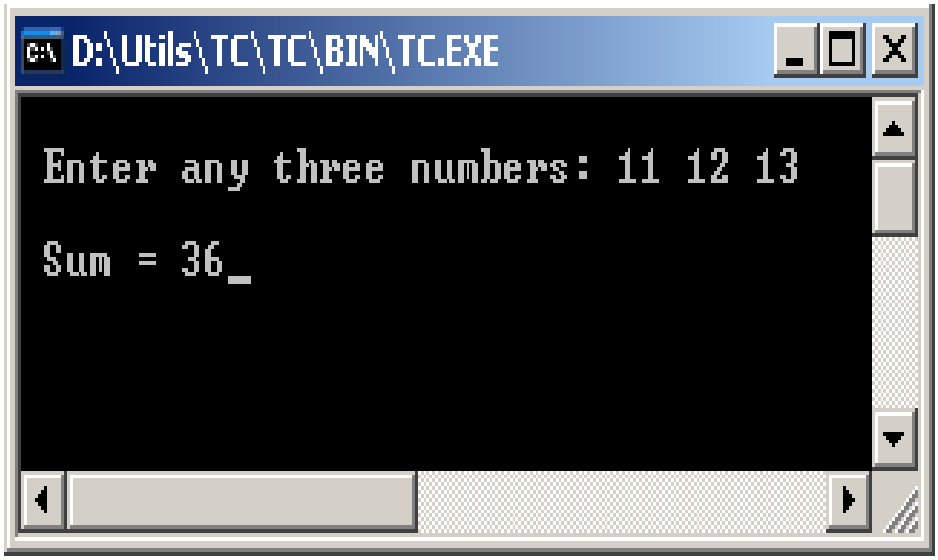
```
#include<stdio.h>

void main()
{
        int a, b, c, sum;

        printf("\nEnter    any    three
   numbers: ");
        scanf("%d %d %d", &a, &b, &c);

        sum = calculatesum(a, b, c);

        printf("\nSum = %d", sum);
}

calculatesum(int x, int y, int z)
{
     int d;

     d = x + y + z;
     return (d);
}
```

3. **Lưu tập tin với tên functionII.C.**

4. **Biên dịch tập tin, functionII.C.**

5. **Thực thi chương trình, functionII.C.**

Kết quả của chương trình trên được minh họa như hình dưới:



- Truyền tham số cho hàm

Khi các đối số được truyền bằng giá trị, các giá trị của đối số của hàm đang gọi không bị thay đổi. Tuy nhiên, có thể có trường hợp, ở đó giá trị của các đối số phải được thay đổi. Trong những trường hợp như vậy, **truyền bằng tham chiếu** được dùng. **Truyền bằng tham chiếu**, hàm được phép truy xuất đến vùng bộ nhớ thực của các đối số và vì vậy có thể thay đổi giá trị của các đối số của hàm gọi.

Ví dụ, xét một hàm, hàm này nhận hai đối số, hoán vị giá trị của chúng và trả về các giá trị của chúng. Nếu một chương trình giống như chương trình dưới đây được viết để giải quyết mục đích này, thì sẽ không bao giờ thực hiện được.

```
#include <stdio.h>
main()
    {
        int x, y;
        x = 15; y = 20;
        printf("x = %d, y = %d\n", x, y);
        swap(x, y);
        printf("\nAfter interchanging x = %d, y = %d\n", x, y);
    }
    swap(int u, int v)
    {
        int temp;
        temp = u;
```

```
        u = v;
        v = temp;
        return;
    }
```

Kết quả của chương trình trên như sau:

```
x = 15, y = 20
After interchanging x = 15, y = 20
```

Hàm **swap**() hoán vị các giá trị của u và v, nhưng các giá trị này không được truyền trở về hàm **main**(). Điều này là bởi vì các biến **u** và **v** trong **swap()** là khác với các biến **u** và **v** được dùng trong **main**(). Truyền bằng tham chiếu có thể được sử dụng trong trường hợp này để đạt được kết quả mong muốn, bởi vì nó sẽ thay đổi các giá trị của các đối số thực. Các con trỏ được dùng khi thực hiện truyền bằng tham chiếu.

Các con trỏ được truyền đến một hàm như là các đối số để cho phép hàm được gọi của chương trình truy xuất các biến mà phạm vi của nó không vượt ra khỏi hàm gọi. Khi một con trỏ được truyền đến một hàm, địa chỉ của dữ liệu được truyền đến hàm nên hàm có thể tự do truy xuất nội dung của địa chỉ đó. Các hàm gọi nhận ra bất kỳ thay đổi trong nội dung của địa chỉ. Theo cách này, đối số hàm cho phép dữ liệu được thay đổi trong hàm gọi, cho phép truyền dữ liệu hai chiều giữa hàm gọi và hàm được gọi. Khi các đối số của hàm là các con trỏ hoặc mảng, truyền bằng tham chiếu được tạo ra đối nghịch với cách truyền bằng giá trị.

Các đối số hình thức của một hàm là các con trỏ thì phải có một dấu * phía trước, giống như sự khai báo biến con trỏ, để xác định chúng là các con trỏ. Các đối số thực kiểu con trỏ trong lời gọi hàm có thể được khai báo là một biến con trỏ hoặc một biến được tham chiếu đến (&var).

Ví dụ, định nghĩa hàm

```
getstr(char *ptr_str, int *ptr_int)
```

đối số **ptr_str** trỏ đến kiểu **char** và **ptr_int** trỏ đến kiểu **int**. Hàm có thể được gọi bằng câu lệnh,

```
getstr(pstr, &var)
```

ở đó **pstr** được khai báo là một con trỏ và địa chỉ của biến **var** được truyền. Gán giá trị thông qua,

```
*ptr_int = var;
```

Hàm bây giờ có thể gán các giá trị đến biến **var** trong hàm gọi, cho phép truyền theo hai chiều đến và từ hàm.

```
char *pstr;
```

Quan sát ví dụ sau của hàm **swap**(). Bài toán này sẽ giải quyết được khi con trỏ được truyền thay vì dùng biến. Mã lệnh tương tự như sau:

```
#include <stdio.h>
    void main()
    {
        int x, y, *px, *py;

        /* Storing address of x in px */
        px = &x;

        /* Storing address of y in py */
        py = &y;
        x = 15; y = 20;
        printf("x = %d, y = %d \n", x, y);
        swap (px, py);

        /* Passing addresses of x and y */
        printf("\n After interchanging x = %d, y = %d\n", x, y);
    }
    swap(int *u, int *v)
    /*  Accept the values of px and py into u and v */
    {
        int temp;
        temp = *u;
        *u = *v;
        *v = temp;
        return;
    }
```

Kết quả của chương trình trên như sau:

```
x = 15, y = 20
After interchanging x = 20, y = 15
```

Hai biến kiểu con trỏ **px** và **py** được khai báo, và địa chỉ của biến **x** và **y** được gán đến chúng. Sau đó các biến con trỏ được truyền đến hàm **swap()**, hàm này hoán vị các giá trị lưu trong x và y thông qua các con trỏ.

- Hàm đệ quy

Mở đầu :

C không những cho phép từ hàm này gọi tới hàm khác, mà nó còn cho phép từ một điểm trong thân của một hàm gọi tới chính hàm đó. Hàm như vậy gọi là hàm đệ qui.

Khi hàm gọi đệ qui đến chính nó, thì mỗi lần gọi máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với tập các biến cục bộ đã được tạo ra trong các lần gọi trước.

Để minh hoạ chi tiết những điều trên, ta xét một ví dụ về tính giai thừa của số nguyên dương n. Khi không dùng phương pháp đệ qui hàm có thể được viết như sau :

```
long int gt(int n) /* Tính n! với n>=0*/
 {
        long int gtphu=1;
        int i;
        for (i=1;i<=n;++i)
        gtphu*=i;
        return s;
 }
```

Ta nhận thấy rằng n! có thể tính theo công thức truy hồi sau :

```
n!=1                nếu n=0
n!=n*(n-1)!         nếu n>0
```

Hàm tính n! theo phương pháp đệ qui có thể được viết như sau :

```
long int gtdq(int n)
  {
        if (n==0 || n==1)
          return 1;
        else
          return(n*gtdq(n-1));
  }
```

Ta đi giải thích hoạt động của hàm đệ qui khi sử dụng trong hàm main dưới đây :

```
#include "stdio.h"
main()
        {
                printf("\n 3!=%d",gtdq(3));
        }
```

Lần gọi đầu tiên tới hàm gtdq được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến tự động của hàm gtdq. Tập này chỉ gồm các đối n. Ta gọi đối n được tạo ra lần thứ nhất là n thứ nhất. Giá trị của tham số thực (số 3) được gán cho n thứ nhất. Lúc này biến n trong thân hàm được xem là n thứ nhất. Do n thứ nhất có giá trị bằng 3 nên điều kiện trong toán tử if là sai và do

đó máy sẽ lựa chọn câu lệnh else. Theo câu lệnh này, máy sẽ tính giá trị biểu thức :

n*gtdq(n-1) (*)

Để tính biểu thức trên, máy cần gọi chính hàm gtdq vì thế lần gọi thứ hai sẽ thực hiện. Máy sẽ tạo ra đối n mới, ta gọi đó là n thứ hai. Giá trị của n-1 ở đây lại là đối của hàm , được truyền cho hàm và hiểu là n thứ hai, do vậy n thứ hai có giá trị là 2. Bây giờ, do n thứ hai vẫn chưa thoả mãn điều kiện if nên máy lại tiếp tục tính biểu thức :

n*gtdq(n-1) (**)

Biểu thức trên lại gọi hàm gtdq lần thứ ba. Máy lại tạo ra đối n lần thứ ba và ở đây n thứ ba có giá trị bằng 1. Đối n=1 thứ ba lại được truyền cho hàm, lúc này điều kiện trong lệnh if được thoả mãn, máy đi thực hiện câu lệnh :

return 1=gtdq(1) (***)

Bắt đầu từ đây, máy sẽ thực hiện ba lần ra khỏi hàm gtdq. Lần ra khỏi hàm thứ nhất ứng với lần vào thứ ba. Kết quả là đối n thứ ba được giải phóng, hàm gtdq(1) cho giá trị là 1 và máy trở về xét giá trị biểu thức

n*gtdq(1) đây là kết quả của (**)

ở đây, n là n thứ hai và có giá trị bằng 2. Theo câu lệnh return, máy sẽ thực hiện lần ra khỏi hàm lần thứ hai, đối n thứ hai sẽ được giải phóng, kết quả là biểu thức trong (**) có giá trị là 2.1. Sau đó máy trở về biểu thức (*) lúc này là :

n*gtdq(2)=n*2*1

n lại hiểu là thứ nhất, nó có giá trị bằng 3, do vậy giá trị của biểu thức trong (*) là 3.2.1=6. Chính giá trị này được sử dụng trong câu lệnh printf của hàm main() nên kết quả in ra trên màn hình là :

3!=6

Chú ý :

Hàm đệ qui so với hàm có thể dùng vòng lặp thì đơn giản hơn, tuy nhiên với máy tính khi dùng hàm đệ qui sẽ dùng nhiều bộ nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy khi gặp một bài toán mà có thể có cách giải lặp (không dùng đệ qui) thì ta nên dùng cách lặp này. Song vẫn tồn tại những bài toán chỉ có thể giải bằng đệ qui.

- Các bài toán có thể dùng đệ qui :

Phương pháp đệ qui thường áp dụng cho các bài toán phụ thuộc tham số có hai đặc điểm sau :

Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Người ta thường gọi là trường hợp suy biến.

Trong trường hợp tổng quát, bài toán có thể qui về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Sau một số hữu hạn bước biến đổi đệ qui nó sẽ dẫn tới trường hợp suy biến.

Bài toán tính n giai thừa nêu trên thể hiện rõ nét đặc điểu này.

- Cách xây dựng hàm đệ qui :

Hàm đệ qui thường được xây dựng theo thuật toán sau :

```
if ( trường hợp suy biến)
        {
                Trình bày cách giải bài toán khi suy biến
        }
  else /* Trường hợp tổng quát */
        {
                Gọi đệ qui tới hàm ( đang viết ) với các giá
                trị khác của tham số
        }
```

- Các ví dụ về dùng hàm đệ qui :

Ví dụ 1 :

Bài toán dùng đệ qui tìm USCLN của hai số nguyên dương a và b.

Trong trường hợp suy biến, khi a=b thì USCLN của a và b chính là giá trị của chúng.

Trong trường hợp chung :

uscln(a,b)=uscln(a-b,b) nếu a>b

uscln(a,b)=uscln(a,b-a) nếu a<b

Ta có thể viết chương trình như sau :

```
#include "stdio.h"
int uscln(int a,int b ); /* Nguyên mẫu hàm*/
main()
        {       int m,n;
                printf("\n Nhap cac gia tri cua a va b :");
                scanf("%d%d",&m,&n);
                printf("\n USCLN cua a=%d va b=%d la :
%d",m,m,uscln(m,n))
        }
int uscln(int a,int b)
        {
                if (a==b)
                        return a;
                else
                  if (a>b)
                        return uscln(a-b,b);

                  else
```

```
            return uscln(a,b-a);
        }
```

Ví dụ 2 :

Chương trình đọc vào một số rồi in nó ra dưới dạng các ký tự liên tiếp.

```
# include "stdio.h"
# include "conio.h"
void prind(int n);
main()
{
  int a;
  clrscr();
  printf("n=");
  scanf("%d",&a);
  prind(a);
  getch();
}
void prind(int n)
 {
  int i;
  if (n<0)
  { putchar('-');
    n=-n;
  }
  if ((i=n/10)!=0)
  prind(i);
  putchar(n%10+'0');
 }
```

6. Câu lệnh return và exit

Mục tiêu:

- Nêu được mục đích của hàm return và exit;

- Vận dụng được hàm return và exit vào chương trình;

6.1. Câu lệnh return

Lệnh **return** có hai mục đích:

- Ngay lập tức trả điều khiển từ hàm về chương trình gọi

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

Trong hàm **squarer(),** một biến **j** kiểu int được định nghĩa để lưu giá trị bình phương của đối số truyền vào. Giá trị của biến này được trả về cho hàm gọi thông qua lệnh **return**. Một hàm có thể thực hiện một tác vụ xác định và trả quyền điều khiển về cho thủ tục gọi nó mà không cần trả về bất kỳ giá trị nào. Trong trường hợp như vậy, lệnh **return** có thể được viết dạng

return(0) hoặc **return**. Chú ý rằng, nếu một hàm cung cấp một giá trị trả về và nó không làm điều đó thì nó sẽ trả về giá trị không thích hợp.
Trong chương trình tính bình phương của các số, chương trình truyền dữ liệu tới hàm **squarer** thông qua các đối số. Có thể có các hàm được gọi mà không cần bất kỳ đối số nào. Ở đây, hàm thực hiện một chuỗi các lệnh và trả về giá trị, nếu được yêu cầu
Chú ý rằng, hàm **squarer()** cũng có thể được viết như sau

```
squarer(int x)
{
    return(x*x);
}
```

Ở đây một biểu thức hợp lệ được xem như một đối số trong câu lệnh return. Trong thực tế, lệnh return có thể được sử dụng theo một trong các cách sau đây:

```
return;
return(hằng);
return(biến);
return(biểu thức);
return(câu lệnh đánh giá); ví dụ: return(a>b?a:b);
```

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

Kiểu của một hàm

type-specifier được sử dụng để xác định kiểu dữ liệu trả về của một hàm. Trong ví dụ trên, **type-specifier** không được viết bên cạnh hàm **squarer**(), vì **squarer**() trả về một giá trị kiểu **int**. type-specifier là không bắt buộc nếu một giá trị kiểu số nguyên được trả về hoặc nếu không có giá trị nào được trả về. . Tuy nhiên, tốt hơn nên chỉ ra kiểu dữ liệu trả về là int nếu một giá trị số nguyên được trả về và tương tự dùng void nếu hàm không trả về giá trị nào.

6.2. Câu lệnh exit

Mục đích của exit là kết thúc chương trình và trả về một mã xác định. Dạng thức của nó như sau:

void exit (int exit code);

exit code được dùng bởi một số hệ điều hành hoặc có thể được dùng bởi các chương trình gọi.

Theo quy ước, mã trả về 0 có nghĩa là chương trình kết thúc bình thường còn các giá trị khác 0 có nghĩa là có lỗi.

các lệnh trên mình chủ yếu chỉ dùng lệnh break để thoát khỏi vòng lặp . Các lệnh khác thường rất ít dược sử dụng

PHÂN BÀI TẬP

1. Viết hàm tính n!
2. Viết hàm tính tổng S = 1+2+….+n.
4. Viết hàm tính số hạng thứ n trong dãy Fibonacci.
5. Viết hàm tìm số lớn nhất trong 2 số. Áp dụng tìm số lớn nhất trong ba số a, b, c với a, b, c nhập từ bàn phím.
6. Viết một chương trình C để tính diện tích và chu vi hình tròn.
7. Viết một chương trình in ra giai thừa của một số nguyên.
8. Viết hàm tìm UCLN của hai số nguyên a và b.
9. Viết hàm in n ký tự c trên một dòng. Viết chương trình cho nhập 5 số nguyên cho biết số lượng hàng bán được của mặt hàng A ở 5 cửa hàng khác nhau. Dùng hàm trên vẽ biểu đồ so sánh 5 giá trị đó, mỗi trị dùng một ký tự riêng.
10. Viết chương trình nhập vào một số nguyên. Viết một hàm tính tổng các chữ số của một số nguyên.
11. Viết chương trình tính tổng S3=1+1/2+1/3+1/4+...1/n với n nhập từ bàn phím.
12. Viết chương trình tính tổng S=1!+2!+...+n! với n nhập vào từ bàn phím.

PHẦN HƯỚNG DẪN LÀM BÀI TẬP

8.Hàm trả về USCLN của 2 số nguyên.

Cách làm:

thế nào là ước chung lớn nhất của hai số a và b: Là số nguyên dương lớn nhất thỏa mãn cả hai số đề cả a và b đều chia hết cho nó

Cụ thể mình đưa vào vòng lặp với biến chạy kiểm tra từ 1 tới (a hoặc b vì ưcln luôn không quá a hoặc không quá b) nếu thỏa mãn điều kiện cả a và b đều chia hết thì tăng ưcln lên với bội là i

```
//Chuong trinh nhap 2 so nguyen va tim uoc chung lon nhat;

#include<stdio.h>
#include<conio.h>
main()
{
int a,b,i,ucln;
lap:
printf("\nNhap so thu nhat: "); scanf("%d",&a);
printf("\nNhap so thu hai : "); scanf("%d",&b);
```

```
if((a<=0)||(b<=0)) { printf("\nMoi nhap lai");goto lap;}
else
{

ucln=1;
for (i=1;i<=a;i++)
{
if((a%i==0)&&(b%i==0)) ucln=ucln*i;
}
printf("\nUCLN= %d",ucln);
printf("\nNhap phim bat ki de tiep tuc: ");
getch();
goto lap;
}
getch();
}
```

10.Tổng các chữ số nguyên tố từ 1 đến n

Yêu cầu: thứ nhất là kiểm tra xem các số từ 1-n có số nào là sô nguyên tố hay không
thứ hai: nếu là số nguyên tố thì tính tổng của nó;
Cách làm: Sử dụng 2 vòng for lồng nhau; vòng thứ nhất duyệt qua các phần tử từ 1 tới n dùng biến chạy i
vòng 2 duyệt các phần tử từ 1 đến i (dùng biến đếm j)mục đích kiểm tra xem i là số nguyên tố hay không nếu có thì tăng biến tổng s thêm giá trị i
Bài này nên xem lại bài kiểm tra số nguyên tố để rõ hơn
//Chuong trinh nhap so nguyen n va tinh tong so nguyen to

```
#include<stdio.h>
#include<conio.h>
main()
{
int n,i,j,s,t;
printf("Nhap n vao: "); scanf("%d",&n);
t=0;
for (i=1;i<=n;i++)
{ s=0;
for (j=1;j<=i;j++)
{ if(i%j==0) s=s+1; } //Kiem tra xem co phai so nguyen to hay khong
if (s==2) t=t+i;} // neu la so nguyen to thi cong cho bien S gia tri i
printf("tong cua so nguyen to la: %d",t);
```

```
    getch();
    }
```

11. Viết chương trình tính tổng S3=1+1/2+1/3+1/4+...1/n với n nhập từ bàn phím.

```
        //Chuong trinh tinh tong S3=1+1/2+1/3+1/4+...1/n voi n nhap tu ban phim


        #include<stdio.h>
        #include<conio.h>
        main()
        {
        unsigned int i,n;
        float s;
        lap:
        printf("\nNhap n vao: "); scanf("%d",&n);
        s=0;
        for (i=1;i<=n;i++)
        {
        s=s+(float)1/i;
        }
        printf("\nTong la: %3.3f",s);
        printf("\nNhan Enter de nhap tiep");
        getch();
        goto lap;
        getch();
        }
        S4 = 1 * 2 * … * n
```

12. Viết chương trình tính tổng S=1!+2!+...+n! với n nhập vào từ bàn phím.

```
//Chuong trinh tinh tong S=1!+2!+...+n! voi n nhap tu ban phim

#include<stdio.h>
#include<conio.h>
main()
{
int i,gt,n,s;
lap:
printf("\nNhap n vao: "); scanf("%d",&n);
s=0;gt=1;
for(i=1;i<=n;i++)
{
gt=gt*i;s=s+gt;
}
```

```
printf("\nTong day S5= %d",s);
printf("\nNhan Enter de tiep tuc: ");
getch();
goto lap;
}
```

BÀI 5
KIỂU MẢNG

Mã bài: MĐ 11-05

Giới thiệu:

Bài học này sẽ cung cấp cho người học các kiến thức sau:

- ✓ Khái niệm mảng.
- ✓ Cách khai báo mảng một chiều, mảng hai chiều, mảng nhiều chiều.
- ✓ Gán giá trị cho mảng trực tiếp, gián tiếp
- ✓ Mảng một chiều, mảng hai chiều, mảng nhiều chiều.
- ✓ Sắp xếp mảng theo thứ tự tăng dần hoặc giảm dần.

Mục tiêu:

- Hiểu khái niệm mảng
- Khai báo được mảng một chiều, mảng hai chiều, mảng nhiều chiều
- Biết cách gán giá trị cho mảng trực tiếp, gián tiếp.
- Vận dụng được mảng làm tham số cho hàm.
- Sắp xếp được mảng theo thứ tự tăng dần hoặc giảm dần
- Rèn luyện tính gọn gàng, ngăn nắp trong công việc.

1. Khai báo mảng

Mục tiêu:

- *Khai báo được biến mảng;*
- *Sử dụng được biến mảng trong chương trình đơn giản;*

Ví dụ 2 : int ia[10]; với int là kiểu mảng, ia là tên mảng, 10 số phần tử mảng

Ý nghĩa: Khai báo một mảng số nguyên gồm 10 phần tử, mỗi phần tử có kiểu int. Mỗi phần tử trong mảng có kiểu int

Mỗi biến chỉ có thể biểu diễn một giá trị. Để biểu diễn một dãy số hay một bảng số ta có thể dùng nhiều biến nhưng cách này không thuận lợi. Trong trường hợp này ta có khái niệm về mảng. Khái niệm về mảng trong ngôn ngữ C cũng giống như khái niệm về ma trận trong đại số tuyến tính.

Mảng có thể được hiểu là một tập hợp nhiều phần tử có cùng một kiểu giá trị và chung một tên. Mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu biến thì có bấy nhiêu kiểu mảng. Mảng cần được khai báo để định rõ:

Loại mảng : int, float, double...

Tên mảng.

Số chiều và kích thước mỗi chiều.

Khái niệm về kiểu mảng và tên mảng cũng giống như khái niệm về kiểu biến và tên biến. Ta sẽ giải thích khái niệm về số chiều và kích thước mỗi chiều thông qua các ví dụ cụ thể dưới đây.

Các khai báo :

```
int a[10],b[4][2];
float x[5],y[3][3];
```

sẽ xác định 4 mảng và ý nghĩa của chúng như sau :

Thứ tự	Tên mảng	Kiểu mảng	Số chiều	Kích thước	Các phần tử
1	A	Int	1	10	a[0],a[1],a[2]...a[9]
2	B	Int	2	4x2	b[0][0], b[0][1] b[1][0], b[1][1] b[2][0], b[2][1] b[3][0], b[3][1]
3	X	Float	1	5	x[0],x[1],x[2]...x[4]
4	Y	Float	2	3x3	y[0][0], y[0][1], y[0][2] y[1][0], y[1][1], y[1][2] y[2][0], y[2][1], y[1][2]

Chú ý :

Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác, các phần tử của mảng có địa chỉ liên tiếp nhau.

Trong bộ nhớ, các phần tử của mảng hai chiều được sắp xếp theo hàng.

Chỉ số mảng :

Một phần tử cụ thể của mảng được xác định nhờ các chỉ số của nó. Chỉ số của mảng phải có giá trị int không vượt quá kích thước tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử z,b,x,y đã được khai báo như trên, và giả sử i,j là các biến nguyên trong đó i=2, j=1. Khi đó :

```
a[j+i-1]    là    a[2]
b[j+i][2-i] là    b[3][0]
y[i][j]     là    y[2][1]
```

Chú ý :

Mảng có bao nhiêu chiều thì ta phải viết nó có bấy nhiêu chỉ số. Vì thế nếu ta viết như sau sẽ là sai : y[i] (Vì y là mảng 2 chiều) vv..

Biểu thức dùng làm chỉ số có thể thực. Khi đó phần nguyên của biểu thức thực sẽ là chỉ số mảng.

Ví dụ :

a[2.5] là a[2]
b[1.9] là a[1]

* Khi chỉ số vượt ra ngoài kích thước mảng, máy sẽ vẫn không báo lỗi, nhưng nó sẽ truy cập đến một vùng nhớ bên ngoài mảng và có thể làm rối loạn chương trình.

Lấy địa chỉ một phần tử của mảng :

Có một vài hạn chế trên các mảng hai chiều. Chẳng hạn có thể lấy địa chỉ của các phần tử của mảng một chiều, nhưng nói chung không cho phép lấy địa chỉ của phần tử của mảng hai chiều. Như vậy máy sẽ chấp nhận phép tính : &a[i] nhưng không chấp nhận phép tính &y[i][j].

Địa chỉ đầu của một mảng :

Tên mảng biểu thị địa chỉ đầu của mảng. Như vậy ta có thể dùng a thay cho &a[0].

Khởi đầu cho biến mảng :

Các biến mảng khai báo bên trong thân của một hàm (kể cả hàm main()) gọi là biến mảng cục bộ.

Muốn khởi đầu cho một mảng cục bộ ta sử dụng toán tử gán trong thân hàm.

Các biến mảng khai báo bên ngoài thân của một hàm gọi là biến mảng ngoài.

Để khởi đầu cho biến mảng ngoài ta áp dụng các qui tắc sau :

Các biến mảng ngoài có thể khởi đầu (một lần) vào lúc dịch chương trình bằng cách sử dụng các biểu thức hằng. Nếu không được khởi đầu máy sẽ gán cho chúng giá trị 0.

Ví dụ :

```
....
float y[6]={3.2,0,5.1,23,0,42};
int z[3][2]={
                {25,31},
                {12,13},
                {45,15}
            {
....
main()
{
        ....
}
```

Khi khởi đầu mảng ngoài có thể không cần chỉ ra kích thước (số phần tử) của nó. Khi đó, máy sẽ dành cho mảng một khoảng nhớ đủ để thu nhận danh sách giá trị khởi đầu.

Ví dụ :

```
....
float a[]={0,5.1,23,0,42};
int m[][3]={
                {25,31,4},
                {12,13,89},
                {45,15,22}
            };
```

Khi chỉ ra kích thước của mảng, thì kích thước này cần không nhỏ hơn kích thước của bộ khởi đầu.

Ví dụ :

```
....
float m[6]={0,5.1,23,0};
int z[6][3]={
                {25,31,3},
                {12,13,22},
                {45,15,11}
            };
....
```

Đối với mảng hai chiều, có thể khởi đầu với số giá trị khởi đầu của mỗi hàng có thể khác nhau :

Ví dụ :

```
....
float z[][3]={
                {31.5},
                {12,13},
                {-45.76}
             };
int z[13][2]={
                {31.11},
                {12},
                {45.14,15.09}
              };
```

Khởi đầu của một mảng char có thể là

Một danh sách các hằng ký tự.

Một hằng xâu ký tự.

Ví dụ :

```
char ten[]={'h','a','g'}
char ho[]='tran'
char dem[10]        ="van"
```

Khai báo : < kiểu phần tử > < tên mãng > [< chỉ số hàng >] [< chỉ số cột >]

*Ví dụ 1 : int a [3] [2] ; float b [3] [4] ; char c [5] [6] ;

=> a [0] [0] a [0] [1]

a [1] [0] a [1] [1]
a [2] [0] a [2] [1]
Ví dụ 2 : #define Hang 5
define Cot 6
int a [Hang] [Cot] ;
⇨ ta có các biến chạy i (chỉ số chạy từ 0 đến (Dong – 1)).
ta có các biến chạy j (chỉ số chạy từ 0 đến (Cot – 1)) .
a [0] [0] a [0][1] …… a [0][Cot - 1]
a [1] [0] a [1][1] …… a [a][Cot - 1]
. . . . . .
a[Dong-1][0]…… . . . . . a[Dong-1][Cot-1]

*Ví dụ : Viết chương trình tính tổng, tích các số trong mãng số thực a[3][2] ;

```
#include < stdio.h>
#define N 3
#define N 2
main ( )
        {
                int i , j ; float a [M][N] ; float tong, tich, tam ;
                /* nhập số liệu */
                for ( i = 0 ; i < M ; i ++ )
                for ( j = 0 ; j < N ; j ++ )
                        { printf ( " nhập a [ %d][%d] = " , i , j );
                          scanf ( " %f " , & tam ) ; a [i][j] = tam ;}
                          /* tính tổng */
                          Tong = 0 ; Tich = 1;
                          for ( i = 0 ; i < M ; i ++ )
                          for ( j = 0 ); j < N ; j ++ )
                        {
                Tong = Tong + a [ i ][j] ; Tich = Tich * a [i][j] ; }
                /* in kết quả */
                printf ( " Tổng là tổng = %f, TONG );
                printf ( " tích là TICH = %F, TICH );
                getch ( ) ;
        }
```

Khởi tạo mảng :
a [5] = { 1,2,3,5,4 }a[0]=1 a[2]=2 .. a[4]=4
Mảng ký tự
- là chuỗi ký tự kết thúc bằng ký tự NULL có mã ASCII là 0 .
- Ví dụ : char S [3] = { ‘L’, ‘0′, ‘P’] : chuỗi này không đúng do thiếu chỗ cho ký tự kết thúc là NULL.
- Ta có thể gán :
char S [4] = " Lop “; Ngôn ngữ C sẽ tự động ghi ký tự kết thúc là NULL, tức là ‘ ‘.

char S[ ] = ” Lop ” ; Không cần khai báo số phần tử mãng.

Ví dụ 1 : Nhập vào một mảng số nguyên sau đó sắp xếp theo thứ tự tăng dần :

```
#include < stdio.h>
#define n 5
main ( )
    {
        int a [ n ] ; int i , j, t ;
        for ( i = 0 ; i > n ; i ++ );
            {
              printf ( ” nhập a [ % d] = ” , i ); scanf ( ” %d”, & a [i ]);
            }
             /* Sắp xếp tăng dần */
                for ( i = 0 ; i < n – 1 ; i ++)
                for ( j = i + 1 ; j < n ; j ++ )
                if ( a [ i ] < a [j ] )
        {
            t = a [ i ] ; a [ i ] = a [ j ]; a [j ] = t ;
        }
           /* in kết quả */
              for ( i = 0 ; i < n ; i ++ )
              printf ( ” % 5d ” , a [ i ] );
        getch ( );
                                                                    }
```

Ví dụ 2: Làm lại ví dụ 1 nhưng viết riêng hàm sắp xếp và truyền tham số cho mảng

1 chiều

```
#include <stdio.h>
#include <conio.h>
#define N 5
void sapxep ( int a [ ] , int n );
void main ( )
{
int a [ N ] ; int i ;
/* nhập 1 số liệu cho mãng */
for ( i = 0 ; i < N , i ++ )
{
printf ( ” A [ %d ] = “, i ); scanf ( ” %d “, & a [ i ] ); }
/* gọi hàm sắp xếp để sắp tăng dần */
sapxep ( a, N );
/* in kết quả */
for ( i = 0 ; i < N ; i ++ )
printf ( ” %5d “, a [ i ] );
```

```
getch ( );
}
/* hàm sắp xếp tăng dần */
void sapxep ( int a [ ], int n )
{
int i, j, t ;
for ( i = 0 ; i > n – 1 ; i ++)
for ( j = i + 1 ; j < n ; j ++ )
if ( a [ i ] > a [ j ]
{
t = a [ i ] ; a [ i ] = a [ j ] ; a [j ] = t ;
}
```

2. Mảng và tham số của hàm

Mục tiêu:

- Trình bày được tham số của hàm;

Truyền tham số mảng nhiều chiều cho hàm

– giả sử a là mãng 2 chiều : float a[M][N]

+ Chương trình gọi :

```
{ float a [M][N]
Tong ( a ) ; ( truyền địa chỉ của mãng cho hàm )
}
```

+ Chương trình bị gọi (chương trình con) :

```
float tong ( float a[ ][N] ) /* khai báo đối để nhận địa chỉ của
mãng */
{
}
```

Note : hàm tong chỉ dùng được đối với các mãng hai chiều có N cột và số hàng không quan trọng, không
khai báo) :

Ví dụ : Viết chương trình tính tổng của 2 ma trận cấp m x n theo công thức :

C[i][j] = a[i][j] + b [i][j]

```
#include <stdio.h>
#define m 3
#define n 4
/* các prototype ( khai báo hàm )*/
void nhap ( int a[ ][N] , int M, int N );
void TongMT ( int a[ ][N], int b[ ][N] , int c [ ][N], int M , int N );
void TongMT ( int a[ ][N], int b[ ][N] , int c [ ][N], int M , int N );
/* chương trình chính */
{ int a [M][N], b[M][N], c[M][N] ;
/* gọi các hàm */
```

```
Nhap ( a, M ,N ) ; nhap ( b, M,N);
TONGMT ( a, b, c , M, N );
InMT ( c, M, N );
Getch ( ) ;
}
/* Hàm nhập số liệu cho mãng 2 chiều m x n phần tử */
void Nhap ( int a [ ][N] , int M , int N )
{
int i , j ;
for ( i= 0 ; i < M ; i ++ )
for ( j = 0 ; j < N ; j++ )
{
printf ( " a[%d][5d] = " , i , j ) ; scanf ( " %d " , &a [i][j]) ; }
return ;
}
Void TongMT ( int a [ ][N], int b [ ][N], int c [ ][N], int M , int N )
{
int i, j ;
for ( i = 0 ; i < M ; i ++ )
for ( j = 0 ; j < N ; j ++ )
c [i][j] = a [i][j] + b [i][j] ;
return ;
}
/* in kết quả */
void inMT ( int c[ ][N], int M, int N )
{
int i, j ;
for ( i = o ; i < M ; i ++ )
{ for ( j = 0 ; j < N ; j ++ )
printf ( " % 3d", a[i][j] );
printf ( " \n " ) ; /* xuống dòng */
}
return ;
}
```

Ví dụ

Nhập vào 2 ma trận vuông cấp n số thập phân. Cộng 2 ma trận này lưu vào ma
trận thứ 3 và tìm số lớn nhất trên ma trận thứ 3.

Dòng	**File Edit Search Run Compile Debug Project Option Window Help**
1 2	/* cong ma tran */

```
3   #include <stdio.h>
4   #include <conio.h>
5
6   #define MAX  20
7
8   //Khai bao prototype
9   void input(float);
10  void output(float);
11  void add(float, float, float);
12  float max(float);
13
14  //khai bao bien toan cuc
15  int in;
16
17  //ham tim so lon nhat trong mang 2 chieu
18  float max(float fa[][MAX])
19  {
20   float fmax;
21    fmax = fa[0][0];          //cho phan tu dau tien la max
22    for (int i = 0; i < in; i++)
23      for (int ij = 0; ij < in; ij++)
24        if (fmax < fa[i][ij])      //neu so dang xet > max
25      fmax = fa[i][ij];     //gan so nay cho max
26   return fmax;        //tra ve ket qua so lon nhat
27  }
28
29  //ham nhap lieu mang 2 chieu
30  void input(float fa[][MAX])
31  {
32    for (int i = 0; i < in; i++)
33      for (int ij = 0; ij < in; ij++)
34    {
35        printf("Nhap vao ptu[%d][%d]: ", i, ij);
36     scanf("%f", &fa[i, j]);
37    }
38  }
39
40  //ham in mang 2 chieu ra man hinh
41  void output(float fa[][MAX])
42  {
43    for (int i = 0; i < in; i++)
44  {
45      for (int ij = 0; ij < n; ij++)
```

```
46      printf("%5.2f", fa[i][ij]);
47     printf("\n");
48     }
49    }
50
51    //ham cong 2 mang 2 chieu
52    void add(float fa[][MAX], float fb[][MAX], float fc[][MAX])
53    {
54     for (int i = 0; i < in; i++)
55       for (int ij = 0; ij < in; ij++)
56         fc[i, ij] = fa[i, ij] + fb[i, ij];
57    }
58
59    void main(void)
60    {
61     float fa[MAX][MAX], fb[MAX][MAX], fc[MAX][MAX];
62      printf("Nhap vao cap ma tran: ");
63     scanf("%d", &in);
64      printf("Nhap lieu ma tran a: \n");
65     input(fa);
66      printf("Nhap lieu ma tran b: \n");
67     input(fb);
68      printf("Nhap lieu ma tran c: \n");
69     input(fc);
70      add(fa, fb, fc);
71      printf("Ma tran a: \n");
72     output(fa);
73      printf("Ma tran b: \n");
74     output(fb);
75      printf("Ma tran c: \n");
76     output(fc);
77      printf("So lon nhat cua ma tran c la: %5.2f.\n", max(fc));
78     getch();
79    }
```

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt - F9 Compile F9 Make F10 Menu

kết quả in ra màn hình

```
Nhap vao cap ma tran : 2
Nhap lieu ma tran a:
Nhap vao ptu[0][0] : 5.2
Nhap vao ptu[0][1] : 4
Nhap vao ptu[1][0] : 7.1
```

```
Ma tran a:
  5.20   4.00
  7.10   9.00
Ma tran b:
 12.00   3.40
```

Nhap vao ptu[1][1] : 9 Nhap lieu ma tran b: Nhap vao ptu[0][0] : 12 Nhap vao ptu[0][1] : 3.4 Nhap vao ptu[1][0] : 9.6 Nhap vao ptu[1][1] : 11	9.60 11.00 Ma tran c: 17.20 7.40 16.70 20.00 So lon nhat cua ma tran c la: 20.00 _
Chạy lại chương trình và thử lại với số liệu khác. Viết thêm hàm tìm số nhỏ nhất.	

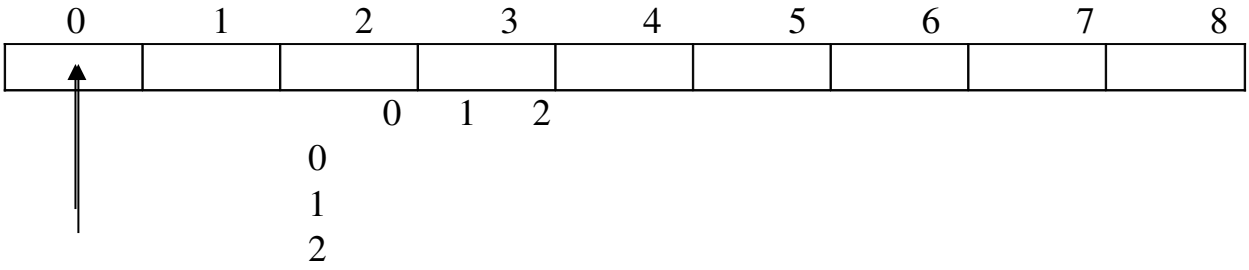
Giải thích chương trình

Trong chương trình khai báo biến in toàn cục do biến này sử dụng trong suốt quá trình chạy chương trình. Tham số truyền vào hàm là mảng hai chiều dưới dạng a[][MAX] vì hàm không dành chỗ cho mảng, hàm chỉ cần biết số cột để tham khảo đến các phần tử.

Trong bài này: Mảng 2 chiều được khai báo int ia[3][3]
Truyền tham số vào hàm: ia[][3]
để tham khảo đến ptử[2][1],
hàm tính như sau:

2 * 3 + 1 = 7 (chỉ số hàng * số cột + chỉ số cột)

ia[3][3] gồm 9 phần tử được lưu trữ trong bộ nhớ như sau:



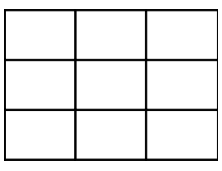
=> Giống như mảng 1 chiều khi truyền mảng 2 chiều sang hàm cũng không tạo bản
sao mới.

3. Sắp xếp mảng

Mục tiêu:

- *Viết được chương trình sắp xếp mảng theo thứ tự tăng dần và giảm dần;*

Trước khi sắp xếp mảng, tốt hơn là nên giữ lại mảng gốc. Vì vậy một mảng khác được khai báo và các phần tử của mảng thứ nhất có thể được sao chép vào mảng mới này. Các dòng mã lệnh sau được sử dụng để thực hiện điều này:



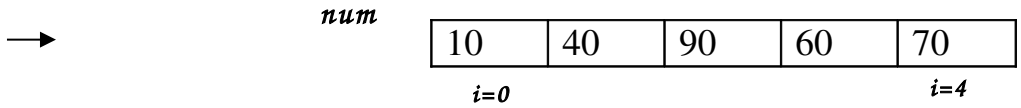
```
int desnum[100], k;
for(k = 0; k < n; k++)
```

desnum[k] = num[k];

- Sắp xếp mảng theo thứ tự giảm dần.

Để sắp xếp một mảng, các phần tử trong mảng cần phải được so sánh với những phần tử còn lại. Cách tốt nhất để sắp xếp một mảng, theo thứ tự giảm dần, là chọn ra giá trị lớn nhất trong mảng và hoán vị nó với phần tử đầu tiên. Một khi điều này được thực hiện xong, giá trị lớn thứ hai trong mảng có thể được hoán vị với phần tử thứ hai của mảng, phần tử đầu tiên của mảng được bỏ qua vì nó đã là phần tử lớn nhất. Tương tự, các phần tử của mảng được loại ra tuần tự đến khi phần tử lớn thứ n được tìm thấy. Trong trường hợp mảng cần sắp xếp theo thứ tự tăng dần giá trị lớn nhất sẽ được hoán vị với phần tử cuối cùng của mảng.

Quan sát ví dụ một dãy số để hiểu được giải thuật. Hình 12.1 trình bày một mảng số nguyên cần được sắp xếp.

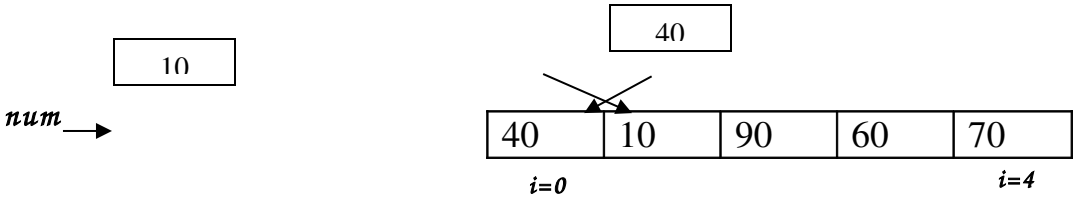


Hình 5-1: Mảng num với chỉ số i (5 phần tử)

Để sắp xếp mảng này theo thứ tự giảm dần,

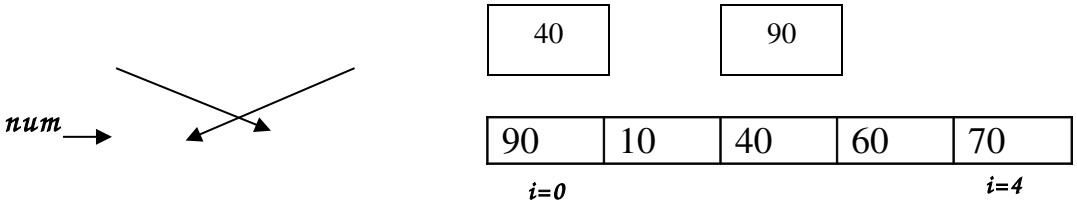
- Chúng ta cần tìm phần tử lớn nhất và hoán vị nó vào vị trí phần tử đầu tiên. Xem như đây là lần thực hiện thứ nhất. Để đưa giá trị lớn nhất về vị trí đầu tiên, chúng ta cần so sánh phần tử thứ nhất với các phần tử còn lại. Khi phần tử đang được so sánh lớn hơn phần tử đầu tiên thì hai phần tử này cần phải được hoán vị.

Khởi đầu, ở lần thực hiện đầu tiên, phần tử ở ví trí thứ nhất được so sánh với phần tử ở vị trí thứ hai. Hình 12.2 biểu diễn sự hoán vị tại vị trí thứ nhất.



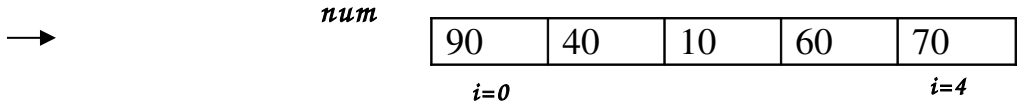
Hình 5-2: Đảo vị trí phần tử thứ nhất với phần tử thứ hai

Tiếp đó, phần tử thứ nhất được so sánh với phần tử thứ ba. Hình 12.3 biểu diễn sự hoán vị giữa phần tử thứ nhất và phần tử thứ ba.



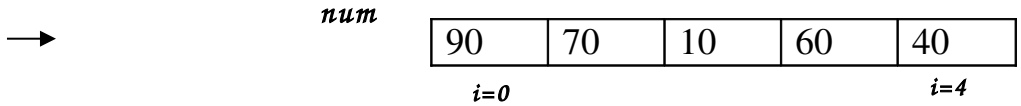
Hình 5-3 Đảo vị trí phần tử thứ nhất với phần tử thứ ba

Quá trình này được lặp lại cho đến khi phần tử thứ nhất được so sánh với phần tử cuối cùng của mảng. Mảng kết quả sau lần thực hiện đầu tiên được trình bày trong hình 12.4 bên dưới.



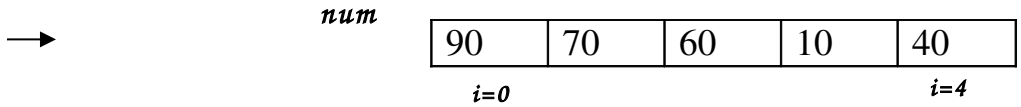
Hình 5-4: Mảng sau lần thực hiện đầu tiên

- Bỏ qua phần tử đầu tiên, chúng ta cần tìm phần tử lớn thứ hai và hoán vị nó với phần tử thứ hai của mảng. Hình 12.5 biểu diễn mảng sau khi được thực hiện lần hai.



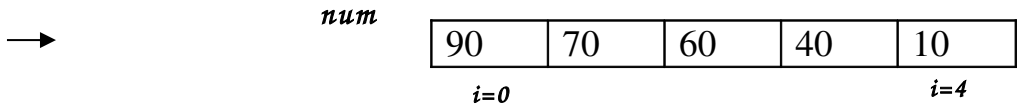
Hình 5-5: Mảng sau lần thực hiện thứ hai

- Phần tử thứ ba phải được hoán vị với phần tử lớn thứ ba của mảng. Hình 12.6 biểu diễn mảng sau khi hoán vị phần tử lớn thứ ba.



Hình 5-6: Mảng sau lần thực hiện thứ ba

- Phần tử thứ tư phải được hoán vị với phần tử lớn thứ tư của mảng. Hình 12.7 biểu diễn mảng sau khi hoán vị phần tử lớn thứ tư.



Hình 5-7: Mảng sau lần thực hiện thứ tư

Để lập trình cho bài toán này, chúng ta cần hai vòng lặp, một để tìm phần tử lớn nhất trong mảng và một vòng lặp kia để lặp quá trình thực hiện n lần. Thực chất quá trình phải lặp n-1 lần cho một phần tử của mảng bởi vì phần tử cuối cùng sẽ không còn phần tử nào để so sánh với nó. Vì vậy, chúng ta khai báo hai biến i và j để thao tác với hai vòng lặp **for**. Vòng lặp **for** với chỉ số i được dùng để lặp lại quá trình xác định phần tử lớn nhất trong phần còn lại của mảng. Vòng lặp **for** với chỉ số j được dùng để tìm phần tử lớn thứ i của mảng trong các phần tử từ phần tử thứ i+1 đến phần tử cuối cùng của mảng. Theo cách đó, phần tử lớn thứ nhất thứ i trong phần còn lại của mảng sẽ được đưa vào vị trí thứ i.

Đoạn mã lệnh khai báo chỉ số và vòng lặp thực hiện n - 1 lần với i như là chỉ số:

```
int i,j;
for(i = 0; i < n - 1; i++)
{
```

Đoạn mã lệnh cho vòng lặp từ phần tử thứ i + 1 đến phần tử thứ n của mảng:

```
for(j = i + 1; j < n; j++)
{
```

Để hoán vị hai phần tử trong mảng chúng ta cần sử dụng một biến tạm. Bởi vì đây là thời điểm một phần tử của mảng được sao chép thành một phần tử khác, giá trị trong phần tử thứ hai sẽ bị mất. Để tránh mất giá trị của phần tử thứ hai, giá trị cần phải được lưu lại trong một biến tạm. Đoạn mã lệnh để hoán vị phần tử thứ i với phần tử lớn nhất trong phần còn lại của mảng là:

```
            if(desnum[i] < desnum[j])
            {
                temp = desnum[i];
                desnum[i] = desnum[j];
                desnum[j] = temp;
            }
        }
    }
```

Các vòng lặp for cần được đóng lại và vì vậy hai dấu ngoặc đóng xuất hiện trong đoạn mã lệnh trên.

- Hiển thị mảng đã được sắp xếp.

Chỉ số i có thể được dùng để hiển thị các giá trị của mảng như các câu lệnh trình bày bên dưới:

```
for(i = 0; i < n; i++)
printf("\n Number at [%d] is %d", i, desnum[i]);
```

Theo cách đó các phần tử của một mảng được sắp xếp. Hãy xem chương trình hoàn thiện dưới đây.

Phép toán này chỉ áp dụng cho các phần tử của mảng một chiều. Giả sử ta có khai báo :

```
double b[20];
```

Khi đó phép toán :

&b[9]

sẽ cho địa chỉ của phần tử b[9].

- Tên mảng là một hằng địa chỉ :

Khi khai báo :

float a[10];

máy sẽ bố trí bố trí cho mảng a mười khoảng nhớ liên tiếp, mỗi khoảng nhớ là 4 byte. Như vậy, nếu biết địa chỉ của một phần tử nào đó của mảng a, thì ta có thể dễ dàng suy ra địa chỉ của các phần tử khác của mảng.

Với C ta có :

a tương đương với &a[0]

a+i tương đương với &a[i]

*(a+i) tương đương với a[i]

- Các phần tử của mảng một chiều :

Khi con trỏ pa trỏ tới phần tử a[k] thì :

pa+i trỏ tới phần tử thứ i sau a[k], có nghĩa là nó trỏ tới a[k+i].

pa-i trỏ tới phần tử thứ i trước a[k], có nghĩa là nó trỏ tới a[k-i].

*(pa+i) tương đương với pa[i].

Như vậy, sau hai câu lệnh :

float a[20],*p;

p=a;

thì bốn cách viết sau có tác dụng như nhau :

a[i] *(a+i) p[i] *(p+i)

Ví dụ :

Vào số liệu của các phần tử của một mảng và tính tổng của chúng :

Cách 1:

```
#include "stdio.h"
main()
	{
		float a[4],tong;
		int i;
		for (i=0;i<4;++i)
		 {
			printf("\n a[%d]=",i);
			scanf("%f",a+i);
		 }
		tong=0;
		for (i=0;i<4;++i)
			tong+=a[i];
		printf("\n Tong cac phan tu mang la :%8.2f ",tong);
	}
```

Cách 2 :

```
#include "stdio.h"
main()
	{
		float a[4],tong, *troa;
		int i;
		troa=a;
		for (i=0;i<4;++i)
		 {
			printf("\n a[%d]=",i);
			scanf("%f",&troa[i]);
		 }
		tong=0;
		for (i=0;i<4;++i)
			tong+=troa[i];
		printf("\n Tong cac phan tu mang la :%8.2f ",tong);
	}
```

Cách 3 :

```
#include "stdio.h"
main()
	{
		float a[4],tong,*troa;
		int i;
		troa=a;
		for (i=0;i<4;++i)
		 {
			printf("\n a[%d]=",i);
			scanf("%f",troa+i);
		 }
		tong=0;
		for (i=0;i<4;++i)
			tong+=*(troa+i);
		printf("\n Tong cac phan tu mang la :%8.2f ",tong);
	}
```

Chú ý :

Mảng một chiều và con trỏ tương ứng phải cùng kiểu.

- Mảng, con trỏ và xâu ký tự :

Như ta đã biết trước đây, xâu ký tự là một dãy ký tự đặt trong hai dấu nháy kép, ví dụ như :

"Viet nam"

Khi gặp một xâu ký tự, máy sẽ cấp phát một khoảng nhớ cho một mảng kiểu char đủ lớn để chứa các ký tự của xâu và chứa thêm ký tự '\0' là ký

tự dùng làm ký tự kết thúc của một xâu ký tự. Mỗi ký tự của xâu được chứa trong một phần tử của mảng.

Cũng giống như tên mảng, xâu ký tự là một hàng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo biến **xau** như một con trỏ kiểu char :

```
char *xau;
```

thì phép gán :

```
xau="Ha noi"
```

là hoàn toàn có nghĩa. Sau khi thực hiện câu lệnh này trong con trỏ **xau** sẽ có địa chỉ đầu của mảng (kiểu char) đang chứa xâu ký tự bên phải. Khi đó các câu lệnh :

```
puts("Ha noi");
puts(xau);
```

sẽ có cùng một tác dụng là cho hiện lên màn hình dòng chữ **Ha noi.**

Mảng kiểu char thường dùng để chứa một dãy ký tự đọc vào bộ nhớ. Ví dụ, để nạp từ bàn phím tên của một người ta dùng một mảng kiểu char với độ dài 25, ta sử dụng các câu lệnh sau :

```
char ten[25];
printf("\n Ho ten :");
gets(ten);
```

Bây giờ ta xem giữa mảng kiểu char và con trỏ kiểu char có những gì giống và khác nhau. Để thấy được sự khác nhau của chúng, ta đưa ra sự so sánh sau :

```
char *xau, ten[15];
ten="Ha noi"
gets(xau);
```

Các câu lệnh trên là không hợp lệ. Câu lệnh thứ hai sai ở chỗ : ten là một hằng địa chỉ và ta không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác. Câu lệnh thứ ba không thực hiện được, mục đích của câu lệnh là đọc từ bàn phím một dãy ký tự và lưu vào một vùng nhớ mà con trỏ **xau** trỏ tới. Song nội dung của con trỏ **xau** còn chưa xác định. Nếu trỏ **xau** đã trỏ tới một vùng nhớ nào đó thì câu lệnh này hoàn toàn có ý nghĩa. Chẳng hạn như sau khi thực hiện câu lệnh:

```
xau=ten;
```

thì cách viết :

gets(ten) ; và gets(xau);

đều có tác dụng như nhau.

4. Gán giá trị cho mảng

Mục tiêu:

- *Gán được giá trị cho mảng;*

```
for (i = 0; i < 10; i++)     //vòng for có giá trị i chạy từ 0 đến 9
{
```

```
   printf("Nhap vao phan tu thu %d: ", i + 1);
   scanf("%d",  &ia[i]);
```

+ Mảng số nguyên :

Ví dụ : Nhập vào mảng số nguyên 5 phần tử

```
#include < stdio.h>
#include < conio.h>
#define n 5
main ()
{
int a [ n ] ; int i ;
for ( i = 0 ; i < n ; i ++ )
{
printf ( ” a [ %d ] = ” , i ); scanf ( ” % d” , & a [ i ]);
}
/* Xuất số liệu mảng ra màn hình */
for ( i = 0 ; i < n ; ++ i)
printf ( ” \ n a [ % d ] = % d “, i , a [ i ]);
getch ();
}
```

+ Mảng số thực float :

```
#include <stdio.h>
#include < conio.h>
#define n 5 ;
main ()
{
float a [ n ] , tam ;
…..scanf ( ” % f ” , &tam) ; /*nhập qua biến trung gian tạm */
a [ i ] = tam ;
```

PHẦN BÀI TẬP

1. Nhập mảng 1 chiều các số nguyên và xuất mảng một chiều các số nguyên
2. Viết hàm xóa một phần tử mảng.
3. Cho mảng số nguyên độ dài n.
4. Viết chương trình nhập vào mảng 1 chiều có n phần tử (có thể dùng hàm randomize cho nhanh) sau đó xuất ra phần tử nào xuất hiện trong mảng nhiều nhất và xuất hiện bao nhiêu lần.
5. Viết hàm in ra các số lẻ theo thứ tự trị tuyệt đối tăng dần, các số chẵn theo thứ tự trị tuyệt đối giảm dần.

PHẦN HƯỚNG DẪN LÀM BÀI TẬP

1. Nhập và xuất mảng 1 chiều các số nguyên

```
void NhapMang(int a[], int &n)
```

```
{
	printf("\nNhap so luong phan tu: ");
	scanf("%d", &n);
	for(int i = 0; i < n ; i++ )
	{
		printf("\nNhap vao phan tu a[%d]: ", i);
		scanf("%d", &a[i]);
	}
}
void XuatMang(int a[], int n)
{
	printf("\nXuat cac phan tu trong mang: ");
	for(int i = 0; i < n; i++)
		printf("%d  ", a[i]);
}
```

2. Xóa một phần tử mảng

```
void XoaPhanTuDau(int a[], int &n)
{
	for(int i = 0; i < n ; i++)
		a[i] = a[i+1];
	n--;
}

void XoaPhanTuCuoi(int a[], int &n)
{
	a[n-1] = NULL;
	n--;
}

int ViTriDau(int a[], int n, int pt)
{
	for(int i = 0; i < n; i++)
		if(a[i] == pt)
			return i;
	return -1;
}

void XoaPhanTuBatKi(int a[], int &n, int pt)
{
	int vt = ViTriDau(a,n,pt);
	if(vt == -1)
		return;
```

```
	while(a[0] == pt)
	{
		XoaPhanTuDau(a,n);
	}

	while(a[n-1] == pt)
	{
		XoaPhanTuCuoi(a,n);
	}

	do
	{
	for(int i = 0; i < n ; i++)
		if(a[i] == pt)
		{
			a[i] = NULL;
			for(int j = i; j < n ; j++)
				a[j] = a[j+1];
			n--;
		}
	}while(ViTriDau(a,n,pt) != -1);

}

void XoaPhanTuViTriBatKi(int a[], int &n)
{
	int vt;
	printf("Nhap vi tri can xoa: ");
	scanf("%d", &vt);

	if(vt == 0)
	{
		XoaPhanTuDau(a,n);
		return;
	}
	if(vt == n-1)
	{
		XoaPhanTuCuoi(a,n);
		return;
	}

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

```
            {
                        a[i] = NULL;
                break;
            }

        for(int i = vt; i < n; i++)
                a[i] = a[i+1];

        n--;
}
```

3. Cho mảng số nguyên độ dài n.
   - In ra mảng con các phần tử dương dài nhất
   - In ra mảng con có tổng lớn nhất.

   VD: cho mảng -1 30 2 -2 3 1 5 6 -5 4 8
   - In ra mảng con là: 3 1 5 6
   - In ra: 30 2

```
void Mang_Con(int a[],int n)
{
int *b;
int i,j,vt;
int dem=0;
int max;
b=new int[n];
for(i=0;i<n;i++)
{
b[i]=0;
}
for(i=0;i<n;i++)
{
if(a[i]>0)
{
dem++;
}
else if(dem>0)
{
b[i-dem]=dem;
dem=0;
}
}
if(dem>0)
{
b[i-dem]=dem;
```

```
}
max=b[0];
vt=0;
for(i=0;i<n;i++)
if(max<b[i])
{
max=b[i];
vt=i;
}
j=vt+max;
while(vt<j)
{

printf("%3d",a[vt]);
vt++;
}
}
void Mang_ConTong(int a[],int n)
{
int *b;
int i,j,vt;
int dem=0;
int tong=0;
int max;
b=new int[n];
for(i=0;i<n;i++)
{
b[i]=0;
}
for(i=0;i<n;i++)
{
if(a[i]>0)
{
dem++;
tong+=a[i];
}
else if(dem>0)
{
b[i-dem]=tong;
dem=0;
tong=0;
}
}
```

```
if(dem>0)
{
b[i-dem]=tong;
}
max=b[0];
vt=0;
for(i=0;i<n;i++)
if(max<b[i])
{
max=b[i];
vt=i;
}
while((vt<n) &&( a[vt]>0))
{
printf("%3d",a[vt]);
vt++;
}
}
```

BÀI 6
CHUỖI

Giới thiệu

Bài học này sẽ cung cấp cho người học những kiến thức sau:

- ✓ Khái niệm về chuỗi.
- ✓ Khai báo biến chuỗi.
- ✓ Nhập vào một chuỗi ký tự cho chương trình trước và sau khi runtime.
- ✓ Các phép toán trên chuỗi.
- ✓ Các hàm xử lý chuỗi để xử lý.

Mục tiêu:

- Khai báo được biến chuỗi
- Biết cách nhập vào một chuỗi kí tự cho chương trình trước và sau khi runtime.
- Hiểu và áp dụng được các phép toán trên chuỗi.
- Vận dụng được các hàm xử lý chuỗi để xử lý.
- Rèn luyện tính gọn gàng, ngăn nắp trong công việc.

1. Khái niệm

Mục tiêu:

- *Hiểu được thế nào là chuỗi kí tự*

Chuỗi được xem như là một mảng 1 chiều gồm các phần tử có kiểu char như mẫu tự, con số và bất cứ ký tự đặc biệt như +, -, *, /, $, #…
Theo quy ước, một chuỗi sẽ được kết thúc bởi ký tự null ('\0' : kí tự rỗng).
Ví dụ: chuỗi "Infoworld" được lưu trữ như sau:

I	n	f	o	w	o	r	l	d	\0

Kí tự kết thúc chuỗi

Các chuỗi trong C được cài đặt như là các mảng ký tự kết thúc bởi ký tự NULL (‘\0’). Bài này sẽ thảo luận về công dụng và thao tác trên chuỗi

2. Khai báo biến chuỗi

Mục tiêu:

- Khai báo được biến chuỗi;

- Ký tự (character) :

– Ví dụ : char ch , ch1 ;

```
ch = ‘a’ ; /* Đúng : ký tự chữ */
ch1 = ‘1′ /* đúng : ký tự số */
```

- Ví dụ 2 : scanf (“%c”, &ch) ; /* gõ A và Enter */

```
printf (“%c”, ch) ; /* In ra chữ A */
printf(“%d”, ch) ; /* In ra 65 là mã ASCII của A */
```

* Hàm dùng cho kiểu ký tự :

char ch ;

ch = getchar ( ) ; (Nhập 1 ký tự từ bàn phímm sau khi ấn Enter và ký tự nhập vào không hiện lên màn hinh).

putchar (ch) ; in ký tự nằm trong biến ch ra màn hình.

putch (“\n”) ; đưa dấu nháy về đầu dòng.ch = getche ( ) ; Nhập 1 ksy tự từ bàn phím và ký tự nhập vào sẽ hiển thị trên màn hình.

- Chuỗi ký tự : Ngôn ngữ C quan niệm 1 chuỗi ký tự là một mãng ký tự kết thúc bằng ký tự NULL (”) mã ASCII là 0.

-Ví dụ: char s[10] L E V A N A ”

s[0] s[1] s[3] s[4] s[5] s[7] s[8]

-Muốn nhập chuỗi ta thường dùng hàm gets(s)

- Muốn in chuỗi ta thường dùng hàm puts(s) : in xong xuống dòng.

- Một số hàm trên chuỗi : các hàm cơ bản trong thư viện string.h

a/ gets(s1) : nhập dữ liệu vào chuỗi s1.

b/ n = strlen(s1) : cho biết độ dài của chuỗi s1.

c/ n= strcmp (s1,s2) : so sánh 2 chuỗi s1,s2 (so theo mã ASCII từng ký tự).

+ nếu n>0 : s1> s2

n = 0 : s1=s2

n < 0 : s1<s2.

d/ strcpy (đích , nguồn) ; chép chuỗi nguồn vào chuỗi đích, gán chuỗi.

- Ví dụ : char [30] ;

Ten = “Nguyễn Văn Đông “; (sai).

strcpy (ten , “Nguyễn Văn Đông “);

gets (ten) : Nhập vào từ bàn phím.

e/ strcat (s1,s2) : nối s1 và s2 .

- Ví dụ : giá trị cảu s1 : ” ABC” ; s2 : ” ABE” => strcat(s1,s2) ; => ” ABCABE”;

f/ m = strncmp (s1, s2, n) ; so sánh n ký tự đầu tiên của chuỗi s1 với s2.

- Ví dụ : m = strncmp (s1, s2, 2) ; thì m = 0 do 2 ký tự đầu của chuỗi là :

+ s1 : “ABC” và s2 : ” ABE” là giống nhau.

g/ strnpy (s1, s2, n) ; chép n phần tử đầu tiên của chuỗi s2 vào chuỗi s1.

- Ví dụ : strnpy (s1, “xyz”, 2) ;

Puts (s1); -ă ” xyC”.

h/ strncat (s1,s2, n) ; nối n phần tử đầu tiên của s2 vào đuôi s1.

- Ví dụ : strncat (s1 , “xyz”, 2);

Puts(s1) ; => “ABCxy”.

* Chú ý : + char s1[10], s2[4]

+ strcpy (s1,”ABCDE”);

+ strcpy(s2,”ABCDE”); => “ABCD” (do s[4] = “”).

i/ Hàm strstr :

- char *p ;
  p = strstr (s1,s2);

- Tìm xem chuỗi s2 có trong s1 hay không. Nếu có thì in ra cuỗi s1 tại vị trí đầu tiên mà nó thấy. Nếu không có thì in ra giá trị NULL.

- Ví dụ : s1: “abc abc ac”
  s2 : “bc”, s3 = “cd”
  p= strstr (s1,s2);
  puts (p) ; => ” bc abc ac “
  p = strstr (s1, s3)
  Đoán thử puts(p) ; => p[(NULL)] .

k/ d= atoi (chuỗi số) ; chuyển chuỗi số thành int.
  f = atof (chuỗi số) ; chuyển chuỗi số thành số thực(float).
  l = atol(chuỗi số); chuyển chuỗi số thành long (nguyên 4 byte).

- Ví dụ : char s[20] ;
  Gets (s) ; nhập vào s từ bàn phím chuỗi ” 123.45″
  d=atoi(s) ; thì d = 123.
  F = atof(s); thì f = 123.45

l/ toupper (ch) ; làm thay đổi ký tự ch thành chữ Hoa.
  tolower(ch); làm thay đổi ký tự ch thành chữ thường.

* Chú ý :Muốn dùng các hàm về chuỗi phải khai báo đầu chương TRÌNH
#INCLUDE
<STRING.H>

3. Nhập chuỗi ký tự

Mục tiêu:

- *Nhập được chuỗi kí tự;*

Các thao tác nhập/xuất (I/O) chuỗi trong C được thực hiện bằng cách gọi các hàm. Các hàm này là một phần của thư viện nhập/xuất chuẩn tên **stdio.h**. Một chương trình muốn sử dụng các hàm nhập/xuất chuỗi phải có câu lệnh khai báo sau ở đầu chương trình:

#include <stdio.h>;

Khi chương trình có chứa câu lệnh này được biên dịch, thì nội dung của tập tin **stdio.h** sẽ trở thành một phần của chương trình

Sử dụng hàm **gets()** là cách đơn giản nhất để nhập một chuỗi thông qua thiết bị nhập chuẩn. Các ký tự sẽ được nhập vào cho đến khi nhấn phím Enter. Hàm **gets()** thay thế ký tự kết thúc trở về đầu dòng ‘\n’ bằng ký tự ‘\0’. Cú pháp hàm này như sau:

gets(str);

Trong đó **str** là một mảng ký tự đã được khai báo. Có thể sử đụng các hàm **scanf()** và **printf()** để nhập và hiển thị các giá trị chuỗi. Các hàm này được dùng để nhập và hiển thị các kiểu dữ liệu hỗn hợp trong một câu lệnh duy nhất. Cú pháp để nhập một chuỗi như sau:

scanf(“%s”, str);

Trong đó ký hiệu định dạng **%s** cho biết rằng một giá trị chuỗi sẽ được nhập vào. **str** là một mảng ký tự đã được khai báo.

4. Các phép toán chuỗi ký tự

Mục tiêu

- Trình bày được phép toán chuỗi ký tự;

-Nhập xuất chuỗi

Nhập chuỗi từ bàn phím

Để nhập một chuỗi ký tự từ bàn phím, ta sử dụng hàm gets()

Cú pháp:gets(<Biến chuỗi>)

Ví dụ:char Ten[20];

gets(Ten);

Ta cũng có thể sử dụng hàm scanf() để nhập dữ liệu cho biến chuỗi, tuy nhiên lúc này ta chỉ có thể nhập được một chuỗi không có dấu khoảng trắng.

Ngoài ra, hàm cgets() (trong conio.h) cũng được sử dụng để nhập chuỗi.

- Xuất chuỗi lên màn hình

Để xuất một chuỗi (biểu thức chuỗi) lên màn hình, ta sử dụng hàm puts().

Cú pháp:puts(<Biểu thức chuỗi>)

Ví dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
        {

                char Ten[12];
                printf("Nhap chuoi: ");gets(Ten);
                printf("Chuoi vua nhap: ");puts(Ten);
                getch();
                return 0;
        }
```

Ngoài ra, ta có thể sử dụng hàm printf(), cputs() (trong conio.h) để hiển thị chuỗi lên màn hình.

Một số hàm xử lý chuỗi (trong string.h)

Cộng chuỗi - Hàm strcat()

Cú pháp:char *strcat(char *des, const char *source)

Hàm này có tác dụng ghép chuỗi nguồn vào chuỗi đích.

Ví dụ: Nhập vào họ lót và tên của một người, sau đó in cả họ và tên của họ lên màn hình.

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

```
#include<string.h>
 int main()
  {
        char HoLot[30], Ten[12];
        printf("Nhap Ho Lot: ");gets(HoLot);
        printf("Nhap Ten: ");gets(Ten);
        strcat(HoLot,Ten);
        printf("Ho ten la: ");puts(HoLot);
        getch();
        return 0;
  }
```

- Xác định độ dài chuỗi - Hàm strlen()

Cú pháp: int strlen(const char* s)

Ví dụ: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main(){
char Chuoi[255];
int Dodai;
printf("Nhap chuoi: ");gets(Chuoi);
Dodai = strlen(Chuoi)
printf("Chuoi vua nhap: ");puts(Chuoi);
printf("Co do dai %d",Dodai);
getch();
return 0;
}
```

- 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)

- Đổi chuỗi chữ thường thành chuỗi chữ hoa, hàm strupr()

Hàm struppr() 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;
                                                                    }
```

- 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)

Sao chép chuỗi, hàm strcpy()

Hàm này 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: Viết chương trình cho phép chép toàn bộ chuỗi nguồn vào chuỗi đích.

```
        #include<conio.h>
        #include<stdio.h>
        #include<string.h>
        int main()
        {
                char Chuoi[255],s[255];
                printf("Nhap chuoi: ");gets(Chuoi);
                strcpy(s,Chuoi);
                printf(“Chuoi dich: ”);puts(s);
                getch();
                return 0;
        }
```

- Sao chép một phần chuỗi, hàm strncpy()

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)

- Trích một phần chuỗi, hàm strchr()

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.
- 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;
        Hình 1
                                                                    }
```

- So sánh chuỗi, hàm strcmp()

Để so sánh hai chuỗi theo từng ký tự trong bảng mã Ascii, ta có thể sử dụng hàm strcmp().

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

Hai chuỗi s1 và s2 được so sánh với nhau, kết quả trả về là một số nguyên (số này có được bằng cách lấy ký tự của s1 trừ ký tự của s2 tại vị trí đầu tiên xảy ra sự khác nhau).

- Nếu kết quả là số âm, chuỗi s1 nhỏ hơn chuỗi s2.
- Nếu kết quả là 0, hai chuỗi bằng nhau.
- Nếu kết quả là số dương, chuỗi s1 lớn hơn chuỗi s2.

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()

Khởi tạo chuỗi, hàm memset()

Hàm này được sử dụng để đặt n ký tự đầu tiên của chuỗi là ký tự c.

Cú pháp:memset(char *Des, int c, size_t n)

- 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.
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. Các thao tác trên chuỗi ký tự

Mục tiêu:

- *Trình bày được thao tác trên chuỗi ký tự;*

- strlen
  - strlen(s) : trả về chiều dài của chuỗi s. (không tính ký tự null)
- strcpy và strncpy
  - strcpy(s1,s2): copy chuỗi s2 vào chuỗi s1. (strcpy1.cpp)
  - strncpy(s1,s2,n): copy n ký tự đầu tiên của chuỗi s2 vào chuỗi s1. s1 phải có đội lớn đủ để chứa n ký tự.
    - n>strlen(s2): giống strcpy
    - n<=strlen(s1): an toàn vì '\0' của s1 vẫn còn
    - strlen(s1)<n<=strlen(s2): không an toàn vì '\0' của s1 không còn =>phải thêm s1[n] = '\0';
- strcmp và strncmp
  - strcmp(s1,s2): so sánh 2 chuỗi s1 và s2 (dựa vào ASCII)
    -1: s1<s2
    0: s1=s2
    1: s1>s2
  - stricmp(s1,s2): so sánh không phân biệt hoa thường
  - strncmp(s1,s2, n)
  - strcmp1.cpp

- strcat và strncat
  - strcat(s1,s2): nối chuỗi s2 vào sau chuỗi s1. s1 phải đủ lớn để chứa cả 2 chuỗi
  - strncat(s1,s2, n)
  - Strcat1.cpp
- strstr
  - strstr(s1,s2): tìm vị trí xuất hiện của chuỗi s2 trong s1. Trả về con trỏ tại vị trí xuất hiện nếu tìm thấy, trả về NULL nếu không tìm thấy.

(strstr1.cpp)

- strchr

- strchr(s1,ch): tìm vị trí xuất hiện của ký tự ch trong s1. Trả về con trỏ tại vị trí xuất hiện nếu tìm thấy, trả về NULL nếu không tìm thấy.

(strchr1.cpp)

PHẦN BÀI TẬP

Mục đích yêu cầu

Đi sâu vào kiểu dữ liệu chuỗi và các phép toán trên chuỗi.

Nội dung

1. Nhập vào 1 chuỗi và xuất chuỗi đó ra theo chiều ngược lại.
2. Nhập vào họ và tên tách ra họ, tên.
3. Nhập vào họ và tên xuất ra họ, tên đệm, tên mỗi từ 1 dòng.
4. Nhập vào 1 dãy số và đọc dãy số đó.

VD: 123 đọc là một trăm hai mươi ba

5. Nhập vào 1 chuỗi sau đó nhập váo 1 từ và kiểm tra xem từ đó có xuất hiện trong chuỗi trên hay không, nếu có thì xuất hiện bao nhiêu lần.

PHẦN HƯỚNG DẪN LÀM BÀI TẬP

1. Nhập vào 1 chuỗi và xuất chuỗi đó ra theo chiều ngược lại.

```
#include <string.h> //thư viện chuỗi

int main()
{
char xau[30];
printf("Nhap vao 1 chuoi: ");
gets(xau);
for(int i=strlen(xau)-1;i>=0;i--) //strlen trả về độ dài của chuỗi
{
printf("%c",xau[i]);
}
getch();
}
```

2. Nhập vào họ và tên tách ra họ, tên.

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

int main()
{
char xau[30];
printf("Nhap vao mot chuoi: ");
gets(xau);
for(int i=0;i<strlen(xau);i++)
```

```
{
if(xau[i]!=32)
{
printf("%c",xau[i]);
}
else
{
for(int j=strlen(xau)-1;j>=i;j--)
{
if(xau[j]==32)
{
for(int k=j;k<=strlen(xau)-1;k++)
printf("%c",xau[k]);
break;
}
}
break;
}
}
getch();
}
```

3. Nhập vào họ và tên xuất ra họ, tên đệm, tên mỗi từ 1 dòng.

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

int main()
{
char xau[30];
printf("Nhap vao mot chuoi: ");
gets(xau);
for(int i=0;i<=strlen(xau)-1;i++)
{
if(xau[i]!=32)
{
printf("%c",xau[i]);
}
else
{
printf("\n");
}
}
```

```
getch();
}
```

4. Nhập vào 1 dãy số và đọc dãy số đó.

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

char doc_so[50];
char *docso(int n)
{
char doc[10]
[5]={"","Mot","Hai","Ba","Bon","Nam","Sau","Bay","Tam","Chin"};
doc_so[0]=0;
int donvi=n%10;
n=n/10;
int chuc=n%10;
int tram=n/10;
if(tram>0)
{
strcat(doc_so,doc[tram]);
strcat(doc_so," Tram ");
}
if(chuc>0)
{
if(chuc==1)
strcat(doc_so," Muoi ");
else
{
strcat(doc_so,doc[chuc]);
strcat(doc_so," Muoi ");
}
}
if(donvi>0)
strcat(doc_so,doc[donvi]);
return doc_so;
}
int main()
{
int n;
printf("Nhap vao mot day so: ");
scanf("%d",&n);
if(n==0)
{
```

```
printf("Khong");
}
else
{
int tram=n%1000;
n=n/1000;
int ngan=n%1000;
n=n/1000;
int trieu=n%1000;
int ty=n/1000;
if(ty>0)
{
printf("%s Ty",docso(ty));
}
if(trieu>0)
{
printf(" %s Trieu ",docso(trieu));
}
if(ngan>0)
{
printf(" %s Ngan ",docso(ngan));
}
if(tram>0)
{
printf(" %s ",docso(tram));
}
}
getch();
}
```

5. Nhập vào 1 chuỗi sau đó nhập vào 1 từ và kiểm tra xem từ đó có xuất hiện trong chuỗi trên hay không, nếu có thì xuất hiện bao nhiêu lần.

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

int main()
{
char xau[50];
char kitukiemtra;
int dem;
printf("Nhap vao mot chuoi: ");
gets(xau);
printf("Nhap vao ki tu muon kiem tra: ");
```

```
scanf("%c",&kitukiemtra);
for(int i=0;i<strlen(xau)-1;i++)
{
if(xau[i]==kitukiemtra)
dem++;
}
if(dem==0)
printf("Ki tu %c khong co trong chuoi",kitukiemtra);
else
printf("Ki tu %c xuat hien %d lan trong chuoi",kitukiemtra,dem);
getch();
}
```

BÀI 7
BIẾN CON TRỎ

Mã bài: MĐ 11-07

Giới thiệu

Bài học này cung cấp cho người học những kiến thức sau:

- ✓ Khái niệm về biến con trỏ.
- ✓ Cơ chế làm việc của biến con trỏ với cấu trúc dữ liệu kiểu mảng.
- ✓ Viết chương trình sử dụng biến con trỏ với cấu trúc dữ liệu kiểu mảng.

Mục tiêu:

- Hiểu được biến con trỏ.
- Biết được cách làm việc của biến con trỏ với cấu trúc dữ liệu kiểm mảng.
- Viết được chương trình sử dụng biến con trỏ với cấu trúc dữ liệu kiểu mảng.
- Tính cách suy luận. tư duy logic

1. Biến con trỏ

- Tác dụng của biến con trỏ;
- Khai báo được biến con trỏ;

Một **con trỏ** là một biến, nó chứa địa chỉ vùng nhớ của một biến khác, chứ không lưu trữ giá trị của biến đó. Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là **con trỏ** đến biến thứ hai kia. Một con trỏ cung cấp phương thức gián tiếp để truy xuất giá trị của các phần tử dữ liệu. Xét hai biến var1 và var2, var1 có giá trị 500 và được lưu tại địa chỉ 1000 trong bộ nhớ. Nếu var2 được khai báo như là một con trỏ tới biến var1, sự biểu diễn sẽ như sau:

Vị trí Bộ nhớ	**Giá trị lưu trữ**	**Tên biến**
1000	500	var1
1001		
1002		
.		
.		
1108	1000	var2

Ở đây, var2 chứa giá trị 1000, đó là địa chỉ của biến var1.

Các con trỏ có thể trỏ đến các biến của các kiểu dữ liệu cơ sở như **int**, **char**, hay **double** hoặc dữ liệu có cấu trúc như **mảng**.

Nếu một biến được sử dụng như một con trỏ, nó phải được khai báo trước. Câu lệnh khai báo con trỏ bao gồm một kiểu dữ liệu cơ bản, một dấu *, và một tên biến. Cú pháp tổng quát để khai báo một biến con trỏ như sau:

type *name;

Ở đó **type** là một kiểu dữ liệu hợp lệ bất kỳ, và **name** là tên của biến con trỏ. Câu lệnh khai báo trên nói với trình biên dịch là **name** được sử dụng để lưu địa chỉ của một biến có kiểu dữ liệu **type**. Trong câu lệnh khai báo, * xác định rằng một biến con trỏ đang được khai báo.

Trong ví dụ của **var1** và **var2** ở trên, vì **var2** là một con trỏ giữ địa chỉ của biến **var1** có kiểu **int**, nó sẽ được khai báo như sau:

int *var2;

Bây giờ, **var2** có thể được sử dụng trong một chương trình để trực tiếp truy xuất giá trị của **var1**. Nhớ rằng, **var2** không phải có kiểu dữ liệu **int** nhưng nó là một con trỏ trỏ đến một biến có kiểu dữ liệu int.

Kiểu dữ liệu cơ sở của con trỏ xác định kiểu của biến mà con trỏ trỏ đến. Về mặt kỹ thuật, một con trỏ có kiểu bất kỳ có thể trỏ đến bất kỳ vị trí nào trong bộ nhớ. Tuy nhiên, tất cả các phép toán số học trên con trỏ đều có liên quan đến kiểu cơ sở của nó, vì vậy khai báo kiểu dữ liệu của con trỏ một cách rõ ràng là điều rất quan trọng.

Con trỏ và mảng một chiều

Tên của một mảng thật ra là một con trỏ trỏ đến phần tử đầu tiên của mảng đó. Vì vậy, nếu **ary** là một mảng một chiều, thì địa chỉ của phần tử đầu tiên trong mảng có thể được biểu diễn là **&ary[0]** hoặc đơn giản chỉ là **ary**. Tương tự, địa chỉ của phần tử mảng thứ hai có thể được viết như **&ary[1]** hoặc **ary+1**,... Tổng quát, địa chỉ của phần tử mảng thứ (i + 1) có thể được biểu diễn là **&ary[i]** hay **(ary+i)**. Như vậy, địa chỉ của một phần tử mảng bất kỳ có thể được biểu diễn theo hai cách:

- Sử dụng ký hiệu & trước một phần tử mảng
- Sử dụng một biểu thức trong đó chỉ số được cộng vào tên của mảng.

Ghi nhớ rằng trong biểu thức **(ary + i), ary** tượng trưng cho một địa chỉ, trong khi **i** biểu diễn số nguyên. Hơn thế nữa, **ary** là tên của một mảng mà các phần tử có thể là cả kiểu số nguyên, ký tự, số thập phân,… (dĩ nhiên, tất cả các phần tử của mảng phải có cùng kiểu dữ liệu). Vì vậy, biểu thức ở trên không chỉ là một phép cộng; nó thật ra là xác định một địa chỉ, một số xác

định của các ô nhớ . Biểu thức **(ary + i)** là một sự trình bày cho một địa chỉ chứ không phải là một biểu thức toán học.
Như đã nói ở trước, số lượng ô nhớ được kết hợp với một mảng sẽ tùy thuộc vào kiểu dữ liệu của mảng cũng như là kiến trúc của máy tính. Tuy nhiên, người lập trình chỉ có thể xác định địa chỉ của phần tử mảng đầu tiên, đó là tên của mảng (trong trường hợp này là **ary**) và số các phần tử tiếp sau phần tử đầu tiên, đó là, một giá trị chỉ số. Giá trị của **i** đôi khi được xem như là một **độ dời** khi được dùng theo cách này.

Các biểu thức **&ary[i]** và **(ary+i)** biểu diễn địa chỉ phần tử thứ **i** của **ary**, và như vậy một cách logic là cả **ary[i]** và ***(ary + i)** đều biểu diễn nội dung của địa chỉ đó, nghĩa là, giá trị của phần tử thứ **i** trong mảng **ary**. Cả hai cách có thể thay thế cho nhau và được sử dụng trong bất kỳ ứng dụng nào khi người lập trình mong muốn.

Chương trình sau đây biểu diễn mối quan hệ giữa các phần tử mảng và địa chỉ của chúng.

```
#include<stdio.h>
void main()
{
      static int ary[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
      int i;
      for (i = 0; i < 10; i ++)
      {
            printf("\n i = %d , ary[i] = %d , *(ary+i)= %d ", i,
         ary[i], *(ary + i));
            printf("&ary[i] = %X , ary + i = %X", &ary[i], ary + i);
            /* %X gives unsigned hexadecimal */
      }
}
```

Chương trình trên định nghĩa mảng một chiều **ary**, có 10 phần tử kiểu số nguyên, các phần tử mảng được gán giá trị tương ứng là 1, 2, ..10. Vòng lặp **for** được dùng để hiển thị giá trị và địa chỉ tương ứng của mỗi phần tử mảng. Chú ý rằng, giá trị của mỗi phần tử được xác định theo hai cách khác nhau, ary[i] và *(ary + i), nhằm minh họa sự tương đương của chúng. Tương tự, địa chỉ của mỗi phần tử mảng cũng được hiển thị theo hai cách. Kết quả của chương trình trên như sau:

```
i=0   ary[i]=1      *(ary+i)=1      &ary[i]=194   ary+i = 194
i=1   ary[i]=2      *(ary+i)=2      &ary[i]=196   ary+i = 196
i=2   ary[i]=3      *(ary+i)=3      &ary[i]=198   ary+i = 198
i=3   ary[i]=4      *(ary+i)=4      &ary[i]=19A   ary+i = 19A
i=4   ary[i]=5      *(ary+i)=5      &ary[i]=19C   ary+i = 19C
```

```
i=5   ary[i]=6      *(ary+i)=6      &ary[i]=19E  ary+i = 19E
i=6   ary[i]=7      *(ary+i)=7      &ary[i]=1A0  ary+i = 1A0
i=7   ary[i]=8      *(ary+i)=8      &ary[i]=1A2  ary+i = 1A2
i=8   ary[i]=9      *(ary+i)=9      &ary[i]=1A4  ary+i = 1A4
i=9   ary[i]=10     *(ary+i)=10     &ary[i]=1A6  ary+i = 1A6
```

Kết quả này trình bày rõ ràng sự khác nhau giữa **ary[i]** - biểu diễn giá trị của phần tử thứ **i** trong mảng, và **&ary[i]** - biểu diễn địa chỉ của nó.

Khi gán một giá trị cho một phần tử mảng như **ary[i]**, vế trái của lệnh gán có thể được viết là **ary[i]** hoặc ***(ary + i)**. Vì vậy, một giá trị có thể được gán trực tiếp đến một phần tử mảng hoặc nó có thể được gán đến vùng nhớ mà địa chỉ của nó là phần tử mảng. Đôi khi cần thiết phải gán một địa chỉ đến một định danh. Trong những trường hợp như vậy, một con trỏ phải xuất hiện trong vế trái của câu lệnh gán. Không thể gán một địa chỉ tùy ý cho một tên mảng hoặc một phần tử của mảng. Vì vậy, các biểu thức như **ary**, **(ary + i)** và **&ary[i]** không thể xuất hiện trong vế trái của một câu lệnh gán. Hơn thế nữa, địa chỉ của một mảng không thể thay đổi một cách tùy ý, vì thế các biểu thức như **ary++** là không được phép. Lý do là vì: **ary** là địa chỉ của mảng **ary.** Khi mảng được khai báo, bộ liên kết đã quyết định mảng được bắt đầu ở đâu, ví dụ, bắt đầu ở địa chỉ 1002. Một khi địa chỉ này được đưa ra, mảng sẽ ở đó. Việc cố gắng tăng địa chỉ này lên là điều vô nghĩa, giống như khi nói

```
x = 5++;
```

Bởi vì hằng không thể được tăng trị, trình biên dịch sẽ đưa ra thông báo lỗi.

Trong trường hợp mảng ary, **ary** cũng được xem như là một **hằng con trỏ**. Nhớ rằng, (ary + 1) không di chuyển mảng **ary** đến vị trí (ary + 1), nó chỉ trỏ đến vị trí đó, trong khi ary++ cố găng dời **ary** sang 1 vị trí.

Địa chỉ của một phần tử không thể được gán cho một phần tử mảng khác, mặc dù giá trị của một phần tử mảng có thể được gán cho một phần tử khác thông qua con trỏ.

```
&ary[2] =     &ary[3];     /* không cho phép*/
ary[2]  =     ary[3];      /* cho phép*/
```

Nhớ lại rằng trong hàm scanf(), tên các tham biến kiểu dữ liệu cơ bản phải đặt sau dấu (&), trong khi tên tham biến mảng là ngoại lệ. Điều này cũng dễ hiểu. Vì scanf() đòi hỏi địa chỉ bộ nhớ của từng biến dữ liệu trong danh sách tham số, trong khi toán tử & trả về địa chỉ bộ nhớ của biến, do đó trước tên biến phải có dấu &. Tuy nhiên dấu & không được yêu cầu đối với

tên mảng, bởi vì tên mảng tự biểu diễn địa chỉ của nó.Tuy nhiên, nếu một phần tử trong mảng được đọc, dấu & cần phải sử dụng.

```
scanf("%d", *ary)  /*      đối với phần tử đầu tiên */
scanf("%d", &ary[2])       /*      đối với phần tử bất kỳ */
```

Con trỏ và mảng nhiều chiều

Một mảng nhiều chiều cũng có thể được biểu diễn dưới dạng con trỏ của mảng một chiều (tên của mảng) và một độ dời (chỉ số). Thực hiện được điều này là bởi vì một mảng nhiều chiều là một tập hợp của các mảng một chiều.Ví dụ, một mảng hai chiều có thể được định nghĩa như là một con trỏ đến một nhóm các mảng một chiều kế tiếp nhau. Cú pháp báo mảng hai chiều có thể viết như sau:

```
data_type (*ptr_var)[expr 2];
```

thay vì

```
data_type array[expr 1][expr 2];
```

Khái niệm này có thể được tổng quát hóa cho các mảng nhiều chiều, đó là,

```
data_type (*ptr_var)[exp 2] .... [exp N];
```

thay vì

```
data_type array[exp 1][exp 2] ... [exp N];
```

Trong các khai báo trên, data_type là kiểu dữ liệu của mảng, ptr_var là tên của biến con trỏ, array là tên mảng, và exp 1, exp 2, exp 3, ... exp N là các giá trị nguyên dương xác định số lượng tối đa các phần tử mảng được kết hợp với mỗi chỉ số.

Chú ý dấu ngoặc () bao quanh tên mảng và dấu * phía trước tên mảng trong cách khai báo theo dạng con trỏ. Cặp dấu ngoặc () là không thể thiếu, ngược lại cú pháp khai báo sẽ khai báo một mảng của các con trỏ chứ không phải một con trỏ của một nhóm các mảng.

Ví dụ, nếu **ary** là một mảng hai chiều có 10 dòng và 20 cột, nó có thể được khai báo như sau:

```
int (*ary)[20];
```

thay vì

```
int ary[10][20];
```

Trong sự khai báo thứ nhất, **ary** được định nghĩa là một con trỏ trỏ tới một nhóm các mảng một chiều liên tiếp nhau, mỗi mảng có 20 phần tử kiểu số nguyên. Vì vậy, **ary** trỏ đến phần tử đầu tiên của mảng, đó là dòng đầu tiên (dòng 0) của mảng hai chiều. Tương tự, (ary + 1) trỏ đến dòng thứ hai của mảng hai chiều, ...

Một mảng thập phân ba chiều **fl_ary** có thể được khai báo như:

```
float (*fl_ary)[20][30];
```

thay vì

```
float fl_ary[10][20][30];
```

Trong khai báo đầu, fl_ary được định nghĩa như là một nhóm các mảng thập phân hai chiều có kích thước 20 x 30 liên tiếp nhau. Vì vậy, fl_ary trỏ đến mảng 20 x 30 đầu tiên, (fl_ary + 1) trỏ đến mảng 20 x 30 thứ hai,...

Trong mảng hai chiều **ary**, phần tử tại dòng 4 và cột 9 có thể được truy xuất sử dụng câu lệnh:

```
ary[3][8];
```

hoặc

```
*(*(ary + 3) + 8);
```

Cách thứ nhất là cách thường được dùng. Trong cách thứ hai, (ary + 3) là một con trỏ trỏ đến dòng thứ 4. Vì vậy, đối tượng của con trỏ này, *(ary + 3), tham chiếu đến toàn bộ dòng. Vì dòng 3 là một mảng một chiều, *(ary + 3) là một con trỏ trỏ đến phần tử đầu tiên trong dòng 3, sau đó 8 được cộng vào con trỏ. Vì vậy, *(*(ary + 3) + 8) là một con trỏ trỏ đến phần tử 8 (phần tử thứ 9) trong dòng thứ 4. Vì vậy đối tượng của con trỏ này, *(*(ary + 3) + 8), tham chiếu đến tham chiếu đến phần tử trong cột thứ 9 của dòng thứ 4, đó là ary [3][8].

Có nhiều cách thức để định nghĩa mảng, và có nhiều cách để xử lý các phần tử mảng. Lựa chọn cách thức nào tùy thuộc vào người dùng. Tuy nhiên, trong các ứng dụng có các mảng dạng số, định nghĩa mảng theo cách thông thường sẽ dễ dàng hơn.

BÀI TẬP

1.Nhập một chuỗi kí tự từ bàn phím sau đó hiển thị từ và số lượng nguyên âm.

2. Sử dụng kiểu con trỏ làm lại tất cả các bài tập của chương 5 (mảng)

PHẦN HƯỚNG DẪN LÀM BÀI TẬP

1. Hiển thị từ và số lượng nguyên âm

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char *ptr;
cha word[10];
int I, vowcnt=0;
printf("\Enter a word:");
scanf("%s", word);
ptr = &word[0];
for(i=0; i<strlen(word); i++)
 {
if((*ptr=='a')‖ ((*ptr=='e')‖ ((*ptr=='i')‖ ((*ptr=='o')‖ ((*ptr=='u')‖
((*ptr=='A')‖ ((*ptr=='E')‖ ((*ptr=='I')‖ ((*ptr=='O')‖ ((*ptr=='U'))
vowcnt++;
ptr++;
}
printf("\n The word is:  %s  \n The number of vowels in the word is:  %d",
word,  vowcnt);
getch();
}
```

TÀI LIỆU THAM KHẢO

1. Các tài liệu tiếng Việt :

1.1. Ngô Trung Việt - *Ngôn ngữ lập trình C và C++ - Bài giảng- Bài tập - Lời giải mẫu* - NXB giao thông vận tải 1995

1.2. Viện tin học - *Ngôn ngữ lập trình C*

1.3. Lê Văn Doanh - *101 thuật toán và chương trình bằng ngôn ngữ C*

2. Các tài liệu tiếng Anh :

2.1. B. Kernighan and D. Ritchie - *The C programming language*
Prentice Hall 1989

2.2. *Programmer's guide Borland C++ Version 4.0*
Borland International, Inc 1993

2.3. Bile - Nabaiyoti - *TURBO C++*