TRƯỜNG CAO ĐẲNG NGHỀ CÔNG NGHIỆP HÀ NỘI

Lê Thị Hồng Phượng (chủ biên)

GIÁO TRÌNH

LẬP TRÌNH CĂN BẢN

(Lưu hành nội bộ)

Hà Nội năm 2013

MỤC LỤC

1. Lịch sử phát triển ngôn ngữ lập trình C

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

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

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

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

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất "mềm dẻo", có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng khác. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp "rắc rối" là "hơi khó hiểu" do sự "mềm dẻo" của C.

Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

Ngôn ngữ C có những đặc điểm cơ bản sau:

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

2. Một số khái niệm dùng trong ngôn ngữ lập trình C

2. 1. Tập ký tự dùng trong ngôn ngữ C.

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để 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 in 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, 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.

2.2. Tên, từ khóa

2.2.1. 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 từ khoá. Độ dài cực đại của tên mặc định là 32 và có thể được đặt lại nằm 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. T*rong 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.*

2.2.2. Từ khóa

Từ khóa 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 khóa của TURBO C:

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

BÀI 1: TỔNG QUAN VỀ NGÔN NGỮ C

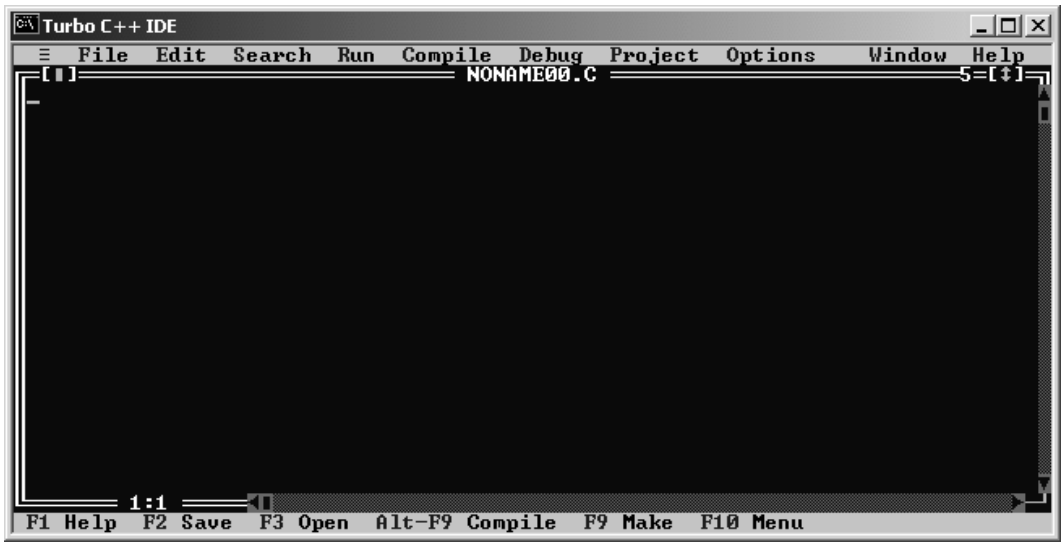
1. Các thao tác cơ bản

1.1. Khởi động và thoát khỏi môi trường C

1.1.1. Khởi động

Chạy Turbo C cũng giống như chạy các chương trình khác trong môi trường DOS hay Windows. Tùy vào máy tính của bạn cài C for Dos hay C for Windows.

Để khởi động C for Dos, chọn thư mục chứa Turbo C đã cài. (Ví dụ: C:\Turbo C\Bin\TP C) màn hình sẽ xuất hiện menu của Turbo C có dạng như sau:



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

Dòng dưới cùng ghi chức năng của một số phím đặc biệt. Chẳng hạn khi gõ phím F1 thì ta có được một hệ thống trợ giúp mà ta có thể tham khảo nhiều thông tin bổ ích, gõ phím F2 thực hiện chức năng ghi tệp, ...

1.1.2. Thoát khỏi

- Vào thực đơn File chọn Quit hoặc nhấn Alt + X.

1.2. Tạo mới, mở và lưu file

1.2.1.Tạo mới một tệp

- Vào thực đơn File/New.

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

- *Các phím xem thông tin trợ giúp*:

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

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

- *Các phím di chuyển con trỏ trong vùng soạn thảo chương trình*:

Phím	Ý nghĩa	Phím tắt
Enter	Đưa con trỏ xuống dòng	
Mũi tên đi lên	Đưa con trỏ lên hàng trước	Ctrl-E
Mũi tên đi xuống	Đưa con trỏ xuống hàng sau	Ctrl-X
Mũi tên sang trái	Đưa con trỏ sang trái một ký tư	Ctrl-S
Mũi tên sang phải	Đưa con trỏ sang phải một ký tư	Ctrl-D
End	Đưa con trỏ đến cuối dòng	
Home	Đưa con trỏ đến đầu dòng	
PgUp	Đưa con trỏ lên trang trước	Ctrl-R
PgDn	Đưa con trỏ xuống trang sau	Ctrl-C
	Đưa con trỏ sang từ bên trái	Ctrl-A
	Đưa con trỏ sang từ bên phải	Ctrl-F

- *Các phím xoá ký tự/dòng*:

Phím	Ý nghĩa	Phím tắt
Delete	Xoá ký tự tại ví trí con trỏ	Ctrl-G
BackSpace	Di chuyển sang trái đồng thời xoá ký tự đứng trước	Ctrl-H
	Xoá một dòng chứa con trỏ	Ctrl-Y
	Xóa từ vị trí con trỏ đến cuối dòng	Ctrl-Q-
	Xóa ký tự bên phải con trỏ	Ctrl-T

- *Các phím chèn ký tự/dòng*:

Insert	Thay đổi viết xen hay viết chồng lên
Ctrl-N	Xen một dòng trống vào trước vị trí con trỏ

- *Sử dụng khối* :

Khối là một đoạn văn bản chương trình hình chữ nhật được xác định bởi đầu khối là góc trên bên trái và cuối khối là góc dưới bên phải của hình chữ nhật. Khi một khối đã được xác định (trên màn hình khối có màu sắc khác chỗ bình thường) thì ta có thể chép khối, di chuyển khối, xoá khối... Sử dụng khối cho phép chúng ta soạn thảo chương trình một cách nhanh chóng. Sau đây là các thao tác trên khối:

Phím tắt	Ý nghĩa
Ctrl-K-B	Đánh dấu đầu khối
Ctrl-K-K	Đánh dấu cuối khối
Ctrl-K-C	Chép khối vào sau vị trí con trỏ
Ctrl-K-V	Chuyển khối tới sau vị trí con trỏ
Ctrl-K-Y	Xoá khối
Ctrl-K-W	Ghi khối vào đĩa như một tập tin

Ctrl-K-R	Đọc khối (tập tin) từ đĩa vào sau vị trí con trỏ
Ctrl-K-H	Tắt/mở khối
Ctrl-K-T	Đánh dấu từ chứa chon trỏ
Ctrl-K-P	In một khối

- *Các phím, phím tắt thực hiện các thao tác khác*:

Phím	Ý nghĩa	Phím tắt
F10	Kích hoạt menu chính	Ctrl-K-, Ctrl-K-Q
F2	Lưu chương trình đang soạn vào đĩa	Ctrl-K-S
F3	Tạo tập tin mới	
Tab	Di chuyển con trỏ một khoảng đồng thời đẩy dòng văn bản	Ctrl-I
ESC	Hủy bỏ thao tác lệnh	Ctrl-U
	Đóng tập tin hiện tại	Alt-F3
	Hiện hộp thoại tìm kiếm	Ctrl-Q-F
	Hiện hộp thoại tìm kiếm và thay thế	Ctrl-Q-A
	Tìm kiếm tiếp tục	Ctrl-L

Ví dụ: Bạn hãy gõ đoạn chương trình sau:

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

```
char ten[50];
printf("Xin cho biet ten cua ban !");
scanf("%s",ten);
printf("Xin chao ban %s",ten);

getch();
return 0;
```

1.2.2. Ghi một tệp lên đĩa

- Vào thực đơn File/Save hoặc nhấn F2.
- Có hai trường hợp xảy ra:

+ Nếu chương trình chưa được ghi lần nào thì một hộp hội thoại sẽ xuất hiện cho phép bạn xác định tên tập tin (File Name). Tên tập tin phải tuân thủ quy cách đặt tên của DOS và không cần có phần mở rộng (sẽ tự động có phần mở rộng là .C hoặc .CPP sẽ nói thêm trong phần Option). Sau đó gõ phím Enter.

+ Nếu chương trình đã được ghi một lần rồi thì nó sẽ ghi những thay đổi bổ sung lên tập tin chương trình cũ.

Chú ý: Đường dẫn mặc định để lưu tệp là Ổ đĩa:\TP C\Bin. Có thể lưu ở đường dẫn khác. Chọn File/Change dir để thay đổi đường dẫn.

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

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

- Phần mở rộng: phần này dài tối đa 3 ký tự.

1.2.3. Mở một tệp có sẵn

Với một chương trình đã có trên đĩa, ta có thể mở nó ra để thực hiện hoặc sửa chữa bổ sung. Để mở một chương trình ta dùng File/Open hoặc gõ phím F3.

Sau đó gõ tên tập tin vào hộp File Name hoặc lựa chọn tập tin trong danh sách các tập tin rồi gõ Enter.

Ví dụ: Mở tập tin CHAO.C sau đó bổ sung để có chương trình mới như sau:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    char ten[50];
    printf("Xin cho biet ten cua ban !");
    scanf("%s",ten);

    printf("Xin chao ban %s\n ",ten);
    printf("Chao mung ban den voi Ngon ngu lap trinh
        C");
    getch();

    return 0;
}
```

Ghi lại chương trình này (F2) và cho thực hiện (Ctrl-F9). Hãy so sánh xem có gì khác trước?

1.3. Dịch và chạy chương trình

Để thực hiện việc dịch chương trình nhấn phím F9.

Để thực hiện chương trình dùng Ctrl-F9 (giữ phím Ctrl và gõ phím F9).

Ví dụ: Thực hiện chương trình vừa soạn thảo xong và quan sát trên màn hình để thấy kết quả của việc thực thi chương trình sau đó gõ phím bất kỳ để trở lại với Turbo.

1.4. Sử dụng menu Help

Trên menu Help bao gồm các lệnh gọi trợ giúp khi người lập trình cần giúp đỡ một số vấn đề nào đó như: Cú pháp câu lệnh, cách sử dụng các hàm có sẵn…

- Lệnh Contents: Hiển thị toàn bộ nội dung của phần help.

- Lệnh Index : Hiển thị bảng tìm kiếm theo chỉ mục.

- Các lệnh còn lại, sẽ tìm hiểu khi thực hành trên máy.

2. Cấu trúc một chương trình C

2.1. Tiền xử lý và biên dịch

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

- Tiền xử lý.

- Biên dịch.

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

Bước tiền xử lý tương ứng với việc cập nhật trong văn bản của chương trình nguồn, chủ yếu dựa trên việc diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor); các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu #.

Hai chỉ thị quan trọng nhất là:

- Chỉ thị sự gộp vào của các tập tin nguồn khác: #include

- Chỉ thị việc định nghĩa các macros hoặc ký hiệu: #define

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

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

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

```
#define SIZE 25
```

2.2. Cấu trúc một chương trình C

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

Cấu trúc có thể như sau:

Các chỉ thị tiền xử lý (Preprocessor directives)

#include <Tên tập tin thư viện>

#define ….

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

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

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

Khai báo các prototype (tên hàm, các tham số, kiểu kết quả trả về của các hàm *sẽ cài đặt trong phần sau, phần này không bắt buộc*): phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm.

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

Chương trình chính phần này bắt buộc phải có.

<Kiểu dữ liệu trả về> main()

{

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

Các câu lệnh dùng để định nghĩa hàm main

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

}

Cài đặt các hàm

<Kiểu dữ liệu trả về> function1(các tham số)

{

Các khai báo cục bộ trong hàm.

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

}

…

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

2.3. Các thư viện thông dụng

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

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

2) conio.h: Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…

3) math.h: Tập tin định nghĩa các hàm tính toán gồm các hàm abs(), sqrt(), log(), log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…

4) alloc.h: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

5) io.h: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…

6) graphics.h: Tập tin định nghĩacác hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …

7) string.h: Chứa các hàm xử lý chuỗi ký tự.

Còn nhiều tập tin khác nữa.

3. Câu lệnh nhập, xuất dữ liệu

3.1. Xuất dữ liệu ra màn hình

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

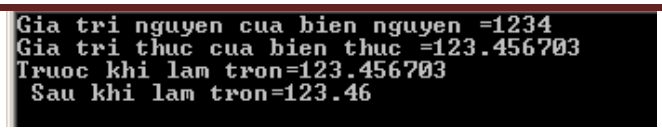
%d	In ra số nguyên
%4d	In số nguyên tối đa 4 ký số;
%f	In số thực
%6f	In số thực tối đa 6 ký số (tính luôn dấu chấm)
%.3f	In số thực có 3 số lẻ, nếu số cần in có nhiều hơn 3 số lẻ thì làm tròn.

- *Các biểu thức*: là các biểu thức mà chúng ta cần xuất giá trị của nó lên màn hình, mỗi biểu thức 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:



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:

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


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

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

Kết quả in ra màn hình:

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	?	Dấu chấm hỏi (?)

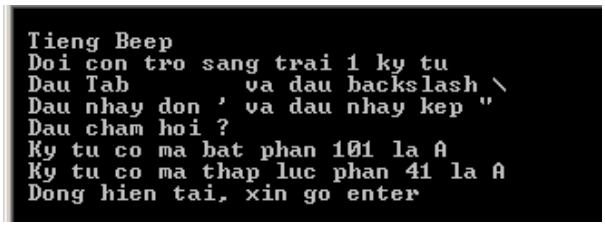
\ddd	ddd	Ký tự có mã ACSII trong hệ bát phân là số ddd
\xHHH	oxHHH	Ký tự có mã ACSII trong hệ thập lục phân là HHH

Ví dụ:

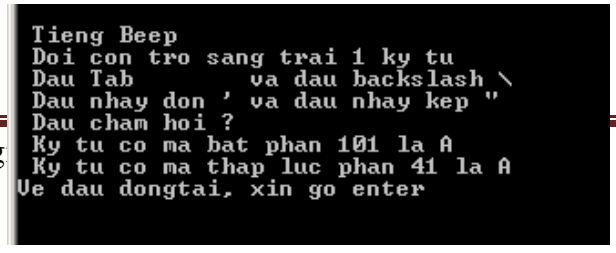
```
#include <stdio.h>
```

```
#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:



Kết quả sau khi gõ phím Enter:



3.2. Đưa dữ liệu vào từ bàn phím

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 cả 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 thuc*/
scanf("%d%f",&bien1,&bien2);/*Doc gia tri cho bien1 co kieu nguyen, bien2 co kieu thuc*/
scanf("%d%f%c",&bien1,&bien2,&bien3);/*bien3 co kieu char*/
```

Lưu ý:

- Chuỗi định dạng phải đặt trong cặp dấu nháy kép (" ").
- Các biến (địa chỉ biến) phải cách nhau bởi dấu phẩy (,).
- Có bao nhiêu biến thì phải có bấy nhiêu định dạng.
- Thứ tự của các định dạng phải phù hợp với thứ tự của các biến.
- Để nhập giá trị kiểu char được chính xác, nên dùng hàm *fflush(stdin)* để loại bỏ các ký tự còn nằm trong vùng đệm bàn phím trước hàm scanf().

- Để nhập vào một chuỗi ký tự (không chứa khoảng trắng hay *kết thúc bằng khoảng trắng*), chúng ta phải khai báo kiểu *mảng ký tự* hay *con trỏ* ký tự, sử dụng định dạng %s và *tên biến thay cho địa chỉ biến*.

- Để đọc vào một chuỗi ký tự có chứa khoảng trắng (*kết thúc bằng phím Enter)*thì *phải dùng hàm gets*().

Ví dụ:

```
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. Hai ký số đầu tiên (12) sẽ được đọc vào cho biennguyen. Hai 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: Hai ký số đầu tiên (12) sẽ được đọc vào cho biennguyen. Ba 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: Hai ký số đầu tiên (12) sẽ được đọc vào cho biennguyen. Năm 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)
```

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

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

Nếu ta nhập chuỗi như sau: *Duong Van Hieu* thì giá 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*

4. Một vài chương trình đơn giản

5. Thực hành

BÀI 2: HẰNG, BIẾN VÀ MẢNG

1. Kiểu dữ liệu

1.1. 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
int	Từ -32768 đến 32767	2 byte
unsigned int	Từ 0 đến 65535	2 byte
long	Từ -2147483648 đến 2147483647	4 byte
unsigned long	Từ 0 đến 4294967295	4 byte

*** *Chú ý:*** Kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên.

1.2. Kiểu dấu phẩy động

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

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

* ***Giải thích:*** Máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ $3.4*10^{-38}$ đến $3.4*10^{38}$. Các số có giá trị tuyệt đối nhỏ hơn $3.4*10^{-38}$ được xem bằng 0. phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

1.3. Kiểu ký tự (char)

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**		**Kích thước**
char (signed char)	Từ -128 đến 127	256	1 byte
unsigned char	Từ 0 đến 255	256	1 byte

Ví dụ sau minh họa 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).

1.4. Định nghĩa kiểu TYPEDEF

1.4.1. Công dụng

Từ khoá typedef dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ được dùng để khai báo dữ liệu sau này. Nên chọn tên kiểu ngắn gọn để dễ nhớ. Chỉ cần thêm từ khoá typedef vào trước một khai báo ta sẽ nhận được một tên kiểu dữ liệu và có thể dùng tên này để khai báo các biến, mảng, cấu trúc, vv...

1.4.2. Cách viết

Viết từ khoá typedef, sau đó kiểu dữ liệu (một trong các kiểu trên), rồi đến tên của kiểu.

Ví dụ câu lệnh:

```
typedef int nguyen;
```

Sẽ đặt tên một kiểu int là nguyên. Sau này ta có thể dùng kiểu nguyên để khai báo các biến, các mảng int như ví dụ sau:

```
nguyen x,y,a[10],b[20][30];
```

Tương tự cho các câu lệnh:

```
typedef float mt50[50];
```

Đặt tên một kiểu mảng thực một chiều có 50 phần tử tên là mt50.

typedef int m_20_30[20][30];

Đặt tên một kiểu mảng thực hai chiều có 20x30 phần tử tên là m_20_30.

Sau này ta sẽ dùng các kiểu trên khai báo:

mt50 a,b;

m_20_30 x,y;

2. Hằng

2.1. Khái niệm

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

2.2. 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 pi 3.141593

Đặt tên cho một hằng float là pi có giá trị là 3.141593.

2.3. Các loại hằng:

2.3.1. Hằng int

Hằng int là số nguyên có giá trị trong khoảng từ -32768 đến 32767.

Ví dụ:

#define number1 -50	Định nghĩa hằng int number1 có giá trị là -50
#define sodem 2732	Định nghĩa hằng int sodem có giá trị là 2732

* *Chú ý:* Cần phân biệt hai hằng 5056 và 5056.0: ở đây 5056 là số nguyên còn 5056.0 là hằng thực.

2.3.2. Hằng long

Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647.

Hằng long được viết theo cách:

1234*l* 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 nghĩa hằng long sl có giá trị là 8865056
#define sl 8865056	Định nghĩa hằng long sl có giá trị là 8865056

2.3.3. Hằng int hệ 8

Hằng int hệ 8 được viết theo cách $0c_1c_2c_3$....ở đây c_i 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 nghĩa hằng int hệ 8 có giá trị là $3*8^2+4*8+5=229$

2.3.4. 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 $0xc_1c_2c_3$... hoặc $0Xc_1c_2c_3$... ở đây c_i là một số trong hệ 16.

Ví dụ:

```
#define h16 0xa5

#define h16 0xA5

#define h16 0Xa5

#define h16 0XA5
```

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

10*16+5=165.

2.3.5. 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 1: '9'-'0'=57-48=9

Ví dụ 2:

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

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

' \$\backslash c_1c_2c_3$'

Trong đó $c_1c_2c_3$ là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

Ví dụ 3: 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'	Xuống 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 ký tự \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.

2.3.6. Hằng xâu ký tự

Hằng xâu ký tự là một dãy ký tự bất kỳ đặt trong cặp 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. Biến

3.1. Định nghĩa biến

Biến là một đại lượng được người lập trình định nghĩa và được đặt tên thông qua việc khai báo biến. Biến dùng để chứa dữ liệu trong quá trình

thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này.

Cách đặt tên biến giống như cách đặt tên đã nói trong phần trên. Mỗi biến thuộc về một kiểu dữ liệu xác định và có giá trị thuộc kiểu đó.

3.2. Cú pháp khai báo biến

<Kiểu dữ liệu> Danh sách các tên biến cách nhau bởi dấu phẩy;

Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng.

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
double canh1, canh2;	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***.

3.3. Vị trí của khai báo biến

Vị trí của khai báo biến trong C:

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

<u>*Khai báo biến ngoài:*</u> Các biến này được đặt bên ngoài tất cả các hàm và nó có tác dụng hay ảnh hưởng đến toàn bộ chương trình (còn gọi là biến toàn cục).

Ví dụ:

```
int i; /* Bien ben ngoai */

float pi;/* Bien ben ngoai */

int main()

{ .....

}
```

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

Ví dụ:

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

int bienngoai;          /*khai bao bien ngoai*/
int   main ()
{ int  j,i;  /*khai  bao  bien  ben  trong  chuong  trinh
  chinh*/

  clrscr(); i=1; j=2; bienngoai=3;
  printf("\n Gia tri cua i la %d",i);
       /*%d là số  nguyên, sẽ  biết sau */

  printf("\n Gia tri cua j la %d",j);

  printf("\n   Gia   tri   cua   bienngoai   la   %d",
    bienngoai);
```

```
        getch();
        return 0;

    }

        getch();
        return 0;
}
```

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;

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

4. Câu lệnh và khối lệnh

4.1. Khái niệm

4.1.1. Khái niệm câu lệnh

Một câu lệnh (statement) xác định một công việc mà chương trình phải thực hiện để xử lý dữ liệu đã được mô tả và khai báo. Các câu lệnh được ngăn cách với nhau bởi dấu chấm phẩy (;).

4.1.2. Khái niệm khối lệnh

Một dãy các câu lệnh được bao bởi 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);
}
```

4.2. Phân loại

Có hai loại lệnh: lệnh đơn và lệnh có cấu trúc.

Lệnh đơn là một lệnh không chứa các lệnh khác. Các lệnh đơn gồm: lệnh gán, các câu lệnh nhập xuất dữ liệu…

Lệnh có cấu trúc là lệnh trong đó chứa các lệnh khác. Lệnh có cấu trúc bao gồm: cấu trúc điều kiện rẽ nhánh, cấu trúc điều kiện lựa chọn, cấu trúc lặp và cấu trúc lệnh hợp thành. Lệnh hợp thành (khối lệnh) là một nhóm bao gồm nhiều khai báo biến và các lệnh được gom vào trong cặp dấu {}.

5. Mảng

6. Các loại biến và mảng

6.1. Biến, mảng tự động

6.2. Biến, mảng ngoài

CÂU HỎI ÔN TẬP

1. Trình bày lịch sử phát triển của turbo C?
2. Trình bày sự cần thiết của ngôn ngữ C?
3. Trình bày thao tác khởi động Turbo C?
4. Trình bày thao tác tạo mới, ghi tệp trong turbo C?
5. Trình bày thao tác thoát khỏi chương trình turbo C?

BÀI 3: BIỂU THỨC

1. Biểu thức

1.1. Khái niệm

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.

Khi viết biểu thức có thể và nên dùng dấu ngoặc tròn ( ) để thể hiện đúng trình tự toán học trong biểu thức. Mỗi biểu thức sẽ có một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là một biểu thức. Trong C đưa ra nhiều quan niệm mới về biểu thức như biểu thức gán, biểu thức điều kiện.

Biểu thức được phân theo kiểu giá trị: nguyên và thực. Trong các mệnh đề logic, biểu thức gồm True và False.

Biểu thức thường được dùng trong:

- Vế phải của câu lệnh gán.

- Tham số thực sự của hàm (như hàm printf)

- Làm chỉ số.

- Trong các toán tử if, for, while, do while, switch.

Ví dụ: Để tính diện tích tam giác, biểu thức tính diện tích thông qua nửa chu vi:

```
p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
```

Thành phần của biểu thức: toán hạng và phép toán. Toán hạng gồm: hằng, biến, phần tử mảng và hàm. Hằng, biến đã được trình bày ở các mục trước. Còn mảng và hàm được trình bày ở phần sau.

1.2. Các phép toán

1.2.1. Phép toán số học

Các phép toán số học hai ngôi gồm:

Phép toán	**Ý nghĩa**	**Ví dụ**
+	Cộng	a+b
-	Trừ	a-b
*	Nhân	a*b
/	Chia	a/b

%	Lấy phần dư	a%b

Phép toán số học một ngôi: -, ví dụ -(a+b).

- Phép % không được áp dụng cho các giá trị kiểu float và double.

Thứ tự ưu tiên giảm dần: Phép trừ một ngôi, *, /, %, + và -. Các phép toán số học được thực hiện từ trái sang phải.

1.2.2. Phép toán thao tác với bit

Các phép toán này không dùng cho kiểu float và double.

Phép toán	**Ý nghĩa**	**Ví dụ**
&	Phép VÀ (AND) theo bit	a&b
|	Phép HOẶC (OR) theo bit	a|b
^	Phép hoặc loại trừ (XOR) theo bit	a^b
<<	Dịch trái	a<<4
>>	Dịch phải	a>>2
~	Lấy phần bù theo bit	~a

Ví dụ: 1&1=1, 1&0=0, 1|1=1, 1|0=1, 0|0=0, 1^1=0, 1^0=1, a<<n=a*2^n, a>>n=a/2^n.

1.2.3.Phép toán so sánh và logic

a. Các phép toán so sánh cho trong bảng sau:

Phép toán	**Ý nghĩa**	**Ví dụ**
>	Có lớn hơn không?	3>7 trả về giá trị 0
>=	Có lơn hơn hay bằng không?	8>=8 có giá trị 1
<	Có nhỏ hơn không?	3<7 có giá trị 1
<=	Có nhỏ hơn hay bằng	3<=2 có giá trị 0
==	Có bằng nhau không?	4==5 có giá trị 0
!=	Có khác nhau không?	3!=4 có giá trị 1

Bốn phép toán đầu có cùng số ưu tiên, hai phép toán sau có thứ tự ưu tiên thấp hơn. Các phép toán so sánh có mức độ ưu tiên thấp hơn phép toán số học. Ví dụ i<n-1 thì được hiểu là i<(n-1)

b. Phép toán logic

- Phép phủ định một ngôi !.

- Phép VÀ (AND) &&

- Phép HOẶC (OR) ||

Chú ý: có thể thực hiện trên các số nguyên, thực.

Các phép toán số sánh có mức ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và ||. Nên biểu thức (a<b) && (c>d) thì có thể được viết gọn thành: a<b && c>d.

1.2.4. Phép toán tăng giảm

Ngôn ngữ C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực).

- Toán tử ++ sẽ tăng (cộng) 1 vào toán hạng của nó.

- Toán tử -- sẽ giảm (trừ) 1 vào toán hạng của nó.

Chẳng hạn nếu n đang có giá trị = 5 thì sau phép tính ++n thì n có giá trị =6, sau phép tính --n thì n có giá trị =4.

Dấu ++ và -- có thể đứng trước hoặc sau toán hạng.

* Sự khác nhau giữa ++n và n++: ở phép toán n++ thì n tăng sau khi giá trị của nó đã được sử dụng, còn ++n thì n được tăng trước khi nó được sử dụng.

Sự khác nhau giữa --n và n-- cũng vậy.

Ví dụ: nếu n=5 thì câu lệnh x=n++ sẽ gán 5 cho x rồi tăng lên một đơn vị nên x=6. Còn câu lệnh x=++n thì n sẽ tăng lên 6 rồi gán 6 cho x.

Các phép toán ++ và -- được sử dụng trong các toán tử for, while, ... để tăng hay giảm giá trị cho các biến điều khiển. Tuy nhiên không nên sử dụng tùy tiện.

1.2.5. Chuyển đổi kiểu giá trị

a. Chuyển đổi kiểu trong biểu thức

Khi hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán. Kết quả thu được là một giá trị có kiểu cao hơn.

Chẳng hạn:

+ Giữa int và long thì int sẽ chuyển thành long.

+ Giữa int và float thì int sẽ chuyển thành float.

+ Giữa float và double thì float sẽ chuyển thành double.

Ví dụ:

1. 5 * (11/3) = 4.5

1.5 * 11/3 = 5.5

(11/3) * 1.5 = 4.5

b. Các phép chuyển đổi kiểu

Các phép chuyển đổi kiểu cũng được thực hiện thông qua phép gán. Giá trị của vế phải được chuyển sang kiểu của vế trái đó là kiểu của kết quả. Kiểu int có thể được chuyển thành kiểu float. Kiểu float có thể chuyển thành int do bị mất phần thập phân. Kiểu double có thể chuyển thành kiểu float bằng cách làm tròn. Kiểu long được chuyển thành kiểu int bằng cách cắt bỏ vài chữ số.

Ví dụ: nếu n là biến nguyên, thì sau khi thực hiện câu lệnh n=15.6 nó sẽ có giá trị là 15.

c. Ép kiểu

Cú pháp: (type) (biểu thức)

Ý nghĩa: diễn ra sự biến đổi kiểu. Kiểu của biểu thức được đổi thành kiểu type theo các nguyên tắc nêu trên.

Ví dụ:

Phép toán: (int) a cho một giá trị kiểu int. Nếu a là biến kiểu float thì ở đây có sự chuyển đổi từ float sang int. Chú ý rằng bản thân của a vẫn không bị thay đổi. Hay a vần có kiểu float nhưng (int) a có kiểu int.

2. Câu lệnh gán và biểu thức

2.1. Khái niệm lệnh gán

Lệnh gán dùng để gán giá trị của một biểu thức cho một biến.

Cú pháp: **<Tên biến> = <biểu thức>**

Ví dụ 1:

```
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 mảng ký
        tự*/
    return 0;
}
```

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

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

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;`

2.2. Câu lệnh gán và biểu thức gán.

2.2.1. Câu lệnh gán

Toán tử gán trong C: biến=biểu thức;

Ví dụ 1: i=i+2; có thể được viết gọn hơn là i+=2;

Toán tử gán dạng: v+=e; trong đó e là biểu thức, v là một biến hoặc phần tử mảng. Có thể được áp dụng cho phép toán hai ngôi.

Ví dụ 2: x=x*(y+3); có thể được viết ngắn gọn là x*=y+3

2.2.2. Biểu thức gán

Là biểu thức có dạng: v=e

Trong đó v là biến (hay phần tử mảng), e là một biểu thức. Giá trị của biểu thức gán là e, kiểu của nó là kiểu của v. Nếu đặt thêm dấu ; vào sau biểu thức gán thì được toán tử gán.

v=e;

Ví dụ: a=b=c=d=8 thì sẽ gán 8 cho cả a, b, c và d.

z=(y=2) * (x=6); thì x=6, y=2 và z=12

3. Biểu thức điều kiện

Biểu thức điều kiện là biểu thức có dạng:

e1?e2:e3

Trong đó e1, e2, e3 là các biểu thức. Giá trị của các biểu thức điều kiện có giá trị bằng giá trị của e2 nếu e1 khác không (e1 đúng) và bằng giá trị của e3 nếu e1 bằng không (e1 sai). Kiểu của biểu thức điều kiện là kiểu cao nhất trong các kiểu của e2 và e3.

Cần chú ý rằng biểu thức điều kiện thực sự là một biểu thức và ta có thể dụng nó như bất kỳ biểu thức nào khác.

Ví dụ: Biểu thức

(5>3) ? 1:0

Biểu thức trên sẽ trả về giá trị là 1 vì biểu thức (5>3) trả về giá trị đúng (!=0)

4. Một số ví dụ

5. Thực hành

CÂU HỎI ÔN TẬP

1. Phân biệt từ khóa và ký hiệu?
2. Trình bày khái niệm tên, quy tắc đặt tên?
3. Trình bày các kiểu dữ liệu sơ cấp?
4. Phân biệt biến và hằng? Các loại biến, hằng?
5. Trình bày vị trí khai báo biến trong chương trình?
6. Trình bày khái niệm biểu thức. Cách sử dụng biểu thức?
7. Trình bày cấu trúc chung của một chương trình?
8. Trình bày một số thư viện thông dụng?
9. Trình bày khái niệm câu lệnh? Phân loại câu lệnh?
10. Trình bày cú pháp và giải thích các tham số của các lệnh nhập và xuất dữ liệu?
11. Thao tác thực thi một chương trình trong Turbo C?

BÀI 4: CÁC LỆNH ĐIỀU KHIỂN

1. Cấu trúc rẽ nhánh

1.1. Câu lệnh if

1.1.1. Câu lệnh if đầy đủ

a. Cú pháp:

if (<*Biểu thức điều kiện*>)

<Khối lệnh 1>

else

<Khối lệnh 2>

b. Ý nghĩa: Đầu tiên *Biểu thức điều kiện* được kiểm tra trước. Nếu điều kiện đúng thì thực hiện Khối lệnh 1. Nếu điều kiện sai thì thực hiện Khối lệnh 2. Các lệnh phía sau Khối lệnh 2 không phụ thuộc vào điều kiện.

Ví dụ 1: Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi a ≠ 0, khi a =0 in ra thông báo "Khong the tim duoc nghich dao cua a"

```
#include <stdio.h>
#include <conio.h>
int main ()
{
float a;
printf("Nhap a = "); scanf("%f",&a);
if (a !=0 )
```

```
        printf("Nghich dao cua %f la %f",a,1/a);

    else
        printf(“Khong the tim duoc nghich dao cua a”);
    getch();
    return 0;
    }
```

Giải thích:

- Nếu chúng ta nhập vào a ≠ 0 thì câu lệnh `printf("Nghich dao cua %f la %f",a,1/a)` được thực hiện, ngược lại câu lệnh `printf(“Khong the tim duoc nghich dao cua a”)` được thực hiện.

- Lệnh getch() luôn luôn được thực hiện.

Ví dụ 2: Yêu cầu người chạy chương trình nhập vào số nguyên a và b, nếu a lớn hơn b thì in ra thông báo “Giá trị của a lớn hơn giá trị của b, giá trị của 2 số là:”, ngược lại thì in ra màn hình câu thông báo “Giá trị của a nhỏ hơn hoặc bằng giá trị của b, giá trị của 2 số là:”.

```
#include <stdio.h>
#include<conio.h>
int main ()
{
int a, b;
printf("Nhap vao gia tri cua 2 so a va b !");
scanf("%d%d",&a,&b);
```

```
if (a>b)
      printf("\n a lon hon b”);
      printf("\n a=%d b=%d ",a,b);
else
      printf("\n a nho hon hoac bang b");
      printf("\n a=%d b=%d",a,b);printf("\n   Thuc   hien
      xong lenh if");
getch();
return 0;
}
```

Giải thích:

- Nếu chúng ta nhập vào 40 30 thì kết quả hiển ra trên màn hình là

a lon hon b

a=40 b=30

Thuc hien xong lenh if

- Còn nếu chúng ta nhập 40 50 thì kết quả hiển ra trên màn hình là

a nho hon hoac bang b

a=40 b=50

Thuc hien xong lenh if

Ví dụ 3: Yêu cầu người thực hiện chương trình nhập vào một số nguyên dương là tháng trong năm và in ra số ngày của tháng đó.

- Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12

- Tháng có 30 ngày: 4, 6, 9, 11

- Tháng có 28 hoặc 29 ngày: 2

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

int thg;
printf("Nhap vao thang trong nam!");
scanf("%d",&thg);
if(thg==1||thg==3||thg==5||thg==7||thg==8||thg==10||
thg==12)

   printf("\n Thang %d co 31 ngay ",thg);
else if (thg==4||thg==6||thg==9||thg==11)
  printf("\n Thang %d co 30 ngay",thg);
     else if (thg==2)
             printf("\n Thang %d co 28 hoac 29 ngay",thg);
          else printf("Khong co thang %d",thg);
printf("\n Thuc hien xong lenh if");

getch();
return  0;
}
```

Giải thích:

- Nếu chúng ta nhập vào một trong các số 1, 3, 5, 7, 8, 10, 12 thì kết quả xuất hiện trên màn hình sẽ là:

Thang <sè> co 31 ngay

Thuc hien xong lenh if

- Nếu chúng ta nhập vào một trong các số 4, 6, 9, 11 thì kết quả xuất hiện trên màn hình sẽ là:

Thang <sè> co 30 ngay

Thuc hien xong lenh if

- Nếu chúng ta nhập vào số 2 thì kết quả xuất hiện trên màn hình sẽ là:

Thang 2 co 28 hoac 29 ngay

Thuc hien xong lenh if

- Nếu chúng ta nhập vào số nhỏ hơn 0 hoặc lớn hơn 12 thì kết quả xuất hiện trên màn hình sẽ là:

Khong co thang <sè>

Thuc hien xong lenh if

Trong đó <số> là con số mà chúng ta đã nhập vào.

Lưu ý:

- Ta có thể sử dụng các câu lệnh if…else lồng nhau. Trong trường hợp if… else lồng nhau thì *else sẽ kết hợp với if gần nhất chưa có else*.

- Trong trường hợp câu lệnh if "bên trong" không có else thì phải viết nó trong cặp dấu {} (coi như là khối lệnh) để tránh sự kết hợp else if sai.

Ví dụ 1:

```
if (so1>0)
```

```
if (so2 > so3)
        a=so2;
else                /*else của if (so2>so3) */
        a=so3;
```

Ví dụ 2:

```
if (so1>0)
{
if (so2>so3)              /*lệnh if này không có else*/
a=so2;
}
else          /*else của if (so1>0)*/
a=so3;
```

1.1.2. Dạng không đầy đủ

a. Cú pháp:

if (<*Biểu thức điều kiện*>)

<Khối lệnh>

b. Ý nghĩa: Kiểm tra *Biểu thức điều kiện* trước. Nếu điều kiện đúng (!= 0) thì thực hiện câu lệnh hoặc khối lệnh liền sau điều kiện. Nếu điều kiện sai thì bỏ qua lệnh hoặc khối lệnh liền sau điều kiện (những lệnh và khối lệnh sau đó vẫn được thực hiện bình thường vì nó không phụ thuộc vào điều kiện sau if).

Ví dụ 1:

Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi a≠0.

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

```
#include <conio.h>
int main ()
{
float a;
printf("Nhap a = "); scanf("%f",&a);
if (a !=0 )
printf("Nghich dao cua %f la %f",a,1/a);
getch();
return 0;
}
```

Giải thích:

- Nếu chúng ta nhập vào a ≠ 0 thì câu lệnh `printf("Nghich dao cua %f la %f",a,1/a)` được thực hiện, ngược lại câu lệnh này không được thực hiện.

- Lệnh getch() luôn luôn được thực hiện vì nó không phải là "lệnh liền sau"

điều kiện if.

Ví dụ 2: Yêu cầu người chạy chương trình nhập vào giá trị của 2 số a và b, nếu a lớn hơn b thì in ra thông báo "Gia trị của a lớn hơn giá trị của b", sau đó hiển thị giá trị cụ thể của 2 số lên màn hình.

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

```
#include<conio.h>
int main ()
{
int a,b;
printf("Nhap vao gia tri cua 2 so a, b!");
scanf("%d%d",&a,&b);
if (a>b)
{
printf("\n Gia tri cua a lon hon gia tri cua b");
printf("\n a=%d, b=%d",a,b);
}
getch();
return 0;
}
```

Giải thích:

Nếu chúng ta nhập vào giá trị của a lớn hơn giá trị của b thì khối lệnh:

```
 {
printf("\n Gia tri cua a lon hon gia tri cua b");
printf("\n a=%d, b=%d",a,b);
}
```

sẽ được thực hiện, ngược lại khối lệnh này không được thực hiện

1.2. Cấu trúc lựa chọn switch

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

1.2.1. Cú pháp:

switch (<*Biểu thức*>)

{

case giá trị 1:

Khối lệnh 1;

break;

case giá trị 2:

Khối lệnh 2;

Break;

.....

case giá trị n:

Khối lệnh n;

break;

[default:

Khối lệnh mặc định;]

}

1.2.2.Ý nghĩa

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

Lưu ý:

Biểu thức trong switch() phải có kết quả là giá trị kiểu số nguyên (int, char, long, …).

Các giá trị sau case cũng phải là kiểu số nguyên. Không bắt buộc phải có default.

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

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

{ int songuyen, phandu;
clrscr();
printf("\n Nhap vao so nguyen ");

scanf("%d",&songuyen); phandu=(songuyen % 2);
switch(phandu)
{

    case 0: printf("%d la so chan ",songuyen);
```

```
        break;

    case 1: printf("%d la so le ",songuyen);
        break;
}

getch();
return 0;
}
```

Ví dụ 2: Nhập vào 2 số nguyên và 1 phép toán. Nếu phép toán là '+', '-', '*' thì in ra kết qua là tổng, hiệu, tích của 2 số. Nếu phép toán là '/' thì kiểm tra xem số thứ 2 có khác không hay không? Nếu khác không thì in ra thương của chúng, ngược lại thì in ra thông báo "khong chia cho 0".

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

{
   int so1,so2; float thuong;char pheptoan;
   clrscr();
   printf("\n Nhap vao 2 so nguyen "); scanf("%d
   %d",&so1,&so2); fflush(stdin);/*Xóa ký tự enter trong
   vùng đệm trước khi nhập phép toán */

   printf("\n Nhap vao phep toan ");
   scanf("%c",&pheptoan);
```

```
switch(pheptoan)
{
   case '+':
       printf("\n %d + %d =%d",so1, so2, so1+so2);
       break;
   case '-':
       printf("\n %d - %d =%d",so1, so2, so1-so2);
       break;
   case '*':
       printf("\n %d * %d =%d",so1, so2, so1*so2);
       break;
   case '/':
        if (so2!=0)
        {   thuong=float(so1)/float(so2);
        printf("\n %d / %d =%f", so1, so2, thuong);
        }
        else printf("Khong chia duoc cho 0");
       break;
       default :
       print(“\n Chua ho tro phep toan %c”, pheptoan);
       break;
```

```
}

}
getch();
return 0;
```

Ví dụ 3: Yêu cầu người thực hiện chương trình nhập vào một số nguyên dương

là tháng trong năm và in ra số ngày của tháng đó.

Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12

Tháng có 30 ngày: 4, 6, 9, 10

Tháng có 28 hoặc 29 ngày : 2

Nếu nhập vào số <1 hoặc >12 thì in ra câu thông báo “không có tháng này “.

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

{ int thang;
clrscr();
printf("\n Nhap vao thangs trong nam ");

scanf("%d",&thang);
switch(thang)
{
```

```
case 1: case 3: case 5: case 7: case 8: case 10: case
12:
    printf("\n Thang %d co 31 ngay ",thang);
    break;

case 4: case 6: case 9: case 11:
    printf("\n Thang %d co 30 ngay ",thang);
    break;
default:
    printf(“n Khong co thang nay”);
    break;
}
getch();
return 0;
```

2. Câu lệnh lặp

2.1. Câu lệnh for

Lệnh for cho phép lặp lại khối lệnh cho đến khi điều kiện sai.

2.1.1. Cú pháp

for (Biểu thức 1; điều kiện; biểu thức 2)

<Khối lệnh>

2.1.2.Ý nghĩa

Thứ tự thực hiện của câu lệnh for như sau:

Bước 1: Tính giá trị của biểu thức 1.

Bước 2: Tính giá trị của điều kiện.

- Nếu giá trị của điều kiện là sai (==0): *thoát khỏi câu lệnh for*.

- Nếu giá trị của điều kiện là đúng (!=0): <Khối lệnh> được thực hiện.

Bước 3: Tính giá trị của biểu thức 2 và quay lại Bước 2.

Một số lưu ý khi sử dụng câu lệnh for:

- Khi điều kiện vắng mặt thì nó được coi là luôn luôn đúng

- Biểu thức 1: thông thường là một phép gán để khởi tạo giá trị ban đầu cho biến điều kiện.

- Điều kiện: là một biểu thức kiểm tra điều kiện đúng sai để dừng vòng lặp.

- Biểu thức2: thông thường là một phép gán để thay đổi giá trị của biến điều kiện.

- Trong mỗi biểu thức có thể có nhiều biểu thức con. Các biểu thức con được phân biệt bởi dấu phẩy.

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

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

{ int i;
clrscr();
printf("\n Day so tu 1 den 10 :");

for (i=1; i<=10; i++)
printf("%d ",i);
```

```
getch();

return 0;

}
```

Kết quả chương trình như sau:

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

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

{ unsigned int n,i,tong;
clrscr();
printf("\n   Nhap   vao   so   nguyen   duong   n:");

scanf("%d",&n);

tong=0;
for (i=1; i<=n; i++)
     tong+=i;

printf("\n Tong tu 1 den %d =%d ",n,tong);
getch();
return 0;

}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:

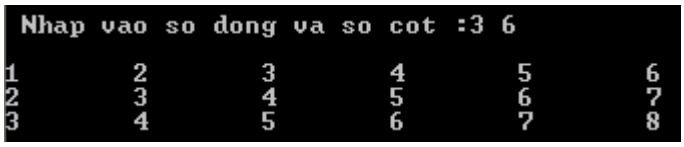
Ví dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau:

```
1   2     3     4     5       6   7

2   3     4     5     6     7    8

3   4     5     6     7     8    9
```

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

int main ()
{ unsigned int dong, cot, n, m;
clrscr();

printf("\n Nhap vao so dong va so cot :");
scanf("%d%d",&n,&m);
for (dong=0;dong<n;dong++)

{

printf("\n");
for (cot=1;cot<=m;cot++)

printf("%d\t",dong+cot);
```

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

Kết quả khi nhập 3 dòng 6 cột như sau



2.2. Câu lệnh while

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

a. Cú pháp:

while (*Biểu thức điều kiện*)

<Khối lệnh>

b. Ý nghĩa:

- Kiểm tra *Biểu thức điều kiện* trước. Nếu điều kiện sai (=0) thì thoát khỏi lệnh while. Nếu điều kiện đúng (!=0) thì thực hiện công việc rồi quay lại kiểm tra điều kiện tiếp.

***Lưu ý*:**

- Lệnh while gồm có biểu thức điều kiện và thân vòng lặp (khối lệnh thực hiện công việc)

- Vòng lặp dừng lại khi nào điều kiện sai.

- Khối lệnh thực hiện công việc có thể rỗng, có thể làm thay đổi điều kiện.

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

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

{ int i;
clrscr();
printf("\n Day so tu 1 den 10 :");

i=1;
while (i<=10)
printf("%d ",i++);

getch();
return 0;

}
```

Kết quả chương trình như sau:

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

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

int main ()
{ unsigned int n,i,tong;
```

```
clrscr();
printf("\n Nhap vao so nguyen duong n:");

scanf("%d",&n);
tong=0;
i=1;

while (i<=n)
    {

       tong+=i;
       i++;
    }

printf("\n Tong tu 1 den %d =%d ",n,tong);
getch();
return 0;
}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:

Ví dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau:

1 2 3 4 5 6 7

2 3 4 5 6 7 8

3 4 5 6 7 8 9

```
#include <stdio.h>
#include <conio.h>
int main()
{
    unsigned int dong, cot, n, m;
clrscr();
printf("\n Nhap vao so dong va so cot:");
scanf("%d%d,&n,&m);
dong=0;
while (dong<n)
{
    printf("\n");
    cot=1;
    while (cot<=m)
    {
        printf("%d\t",dong+cot);
        cot++;
    }
    dong++;
}
getch();
return 0;
}
```

Kết quả khi nhập 3 dòng 6 cột như sau:

```
Nhap vao so dong va so cot :3 6

1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

2.3. Cấu trúc lặp do …while

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

2.3.1. Cú pháp:

do

<Khối lệnh>

while (<*Biểu thức điều kiện*>)

2.3.2. Ý nghĩa

- Trước tiên khối lệnh được thực hiện trước, sau đó mới kiểm tra *Biểu thức điều kiện*. Nếu điều kiện sai thì thoát khỏi lệnh do …while. Nếu điều kiện còn đúng thì thực hiện khối lệnh rồi quay lại kiểm tra điều kiện tiếp.

**** Lưu ý*:**

- Lệnh do…while thực hiện khối lệnh ít nhất 1 lần.

- Vòng lặp dừng lại khi điều kiện sai.

- Khối lệnh có thể rỗng, có thể làm thay đổi điều kiện.

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

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

{ int i;
clrscr();
printf("\n Day so tu 1 den 10 :");

i=1;
do

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

```
while (i<=10);
getch();
return 0;

}
```

Kết quả chương trình như sau:

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

```
#include <stdio.h>
#include<conio.h>
int main ()
{ unsigned int n,i,tong;
clrscr();

printf("\n Nhap vao so nguyen duong n:");
scanf("%d",&n);
tong=0;

i=1;
do
{

    tong+=i;
    i++;
```

```
} while (i<=n);

printf("\n Tong tu 1 den %d =%d ",n,tong);
getch();
return 0;

}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:

Ví dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau (n, m>=1):

```
1    2    3    4    5    6    7
2    3    4    5    6    7    8
3    4    5    6    7    8    9
…
#include <stdio.h>
#include<conio.h>

int main ()
{ unsigned int dong, cot, n, m;
clrscr();

printf("\n Nhap vao so dong va so cot :");
scanf("%d%d",&n,&m);
dong=0;
do
```

```
{

    printf("\n");
    cot=1;

    do
      {

          printf("%d\t",dong+cot);
          cot++;
        } while (cot<=m);
    dong++;

}
while (dong<n);
getch();
return 0;

}
```

Kết quả khi nhập 3 dòng 6 cột như sau

```
Nhap vao so dong va so cot :3 6

1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

2.4. So sánh các vòng lặp

2.4.1. Vòng lặp for và while

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

- Vòng lặp kết thúc khi nào điều kiện sai.

2.4.2. Vòng lặp do…while

- Thực hiện công việc trước kiểm tra điều kiện sau nên đoạn lệnh thực hiện công việc được thực hiện ít nhất 1 lần.

- Vòng lặp kết thúc khi nào điều kiện sai.

3. Các lệnh dừng vòng lặp

3.1.Lệnh Break

Cú pháp: **break**

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

3.2.Lệnh Continue

Cú pháp: **continue**

- Khi gặp lệnh này trong các vòng lặp, chương trình sẽ bỏ qua phần còn lại trong vòng lặp và tiếp tục thực hiện lần lặp tiếp theo.

- Đối với lệnh for, *biểu thức 2* sẽ được tính giá trị và quay lại bước 2.

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

3.3. Câu lệnh go to

3.4. Ví dụ

CÂU HỎI ÔN TẬP

1. Trình bày cú pháp, ý nghĩa của lệnh có cấu trúc rẽ nhánh dạng đầy đủ và không đầy đủ?
2. Trình bày cú pháp, ý nghĩa của lệnh có cấu trúc lựa chọn?
3. Trình bày cú pháp, ý nghĩa của lệnh có cấu trúc lặp?
4. So sánh cấu trúc lặp for, while và do …while?
5. Trình bày cú pháp và ý nghĩa của các lệnh đặc biệt?

BÀI 5: HÀM

1. Khái niệm hàm

1.1. Khái niệm và phân loại

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

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

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

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

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

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

```
int max(int a, int b)
{

    return (a>b) ? a:b;
}
```

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

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b)
{
      return (a>b) ? a:b;
}
int main()
{
    int a, b, c;
    printf("\n Nhap vao 3 so a, b,c  ");
    scanf("%d%d%d",&a,&b,&c);
    printf("\n So lon la %d",max(a, max(b,c)));
    getch();
    return 0;
}
```

1.2. Quy tắc hoạt động của hàm

+ Lời gọi hàm có dạng: Tên_hàm ([danh sách các tham số thực]);

- Số tham số thực phải bằng số tham số hình thức (đối) và mỗi tham số thực phải có cùng kiểu giá trị như kiểu giá trị của đối tương ứng với nó.

- Khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện. Nói cách khác khi máy gặp một lời gọi hàm ở một chỗ nào đố của chương trình thì máy sẽ tạm rời chỗ đó và chuyển đến hàm tương ứng. quá trình đố sẽ diễn ra theo trình tự các bước sau:

+ Cấp phát bộ nhớ cho các đối số và các biến cục bộ.

+ Gán giá trị của các tham số thực cho các đối số 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 xóa các đối số, các biến cục bộ và thoát 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.

2. Xây dựng hàm

2.1. Định nghĩa hàm

2.1.1. Cấu trúc của một hàm tự thiết kế

***<kiểu kết quả>*Tên hàm([*<kiểu tham số> <tham số>*][,*<kiểu tham số><tham số>*][…])**

{

[Khai báo biến cục bộ và các câu lệnh thực hiện hàm] [return [<Biểu thức>];]

}

2.1.2. Giải thích

- *Kiểu kết quả:* là kiểu dữ liệu của kết quả trả về, có thể là : int, byte, char, float, void… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp *hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void*.

- *Kiểu tham số*: là kiểu dữ liệu của tham số.

- *Tham số:* là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. *Tham số này gọi là tham số hình thức*, *khi gọi hàm chúng*

ta phải truyền cho nó các tham số thực tế. Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm (phần giới hạn bởi cặp dấu {}) là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

- Khi định nghĩa hàm, ta thường sử dụng câu lệnh return để trả về kết quả thông qua tên hàm.

Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

return ; /*không trả về giá trị*/

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

return (<biểu thức>); /*Trả về giá trị của biểu thức*/

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

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

```
int max(int a, int b)
{

return (a>b) ? a:b;
}
```

2.2. Sử dụng hàm

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

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

Chú ý: Số tham số thực phải bằng số đối. Kiểu của tham số thực phải phù hợp với kiểu của đối tương ứng

Ví dụ: Viết chương trình cho phép tìm ước số chung lớn nhất của hai số tự nhiên.

```
#include<stdio.h>
unsigned int      ucln(unsigned int a, unsigned  int b)
{

unsigned         int u;
if (a<b)
u=a;

else
u=b;

while ((a%u !=0) || (b%u!=0))
u--;

return u;
}

int main()
{

unsigned int a, b, UC;
printf(“Nhap a,b: ”);scanf(“%d%d”,&a,&b);
UC = ucln(a,b);
```

```
printf("Uoc chung lon nhat la: ", UC);
return 0;
}
```

Lưu ý: Việc gọi hàm là một phép toán, không phải là một phát biểu.

3. Các tham số của hàm

3.1. Phân biệt các loại tham số

- Biến toàn cục: Không thuộc khối nào, có tác dụng ở cả chương trình kể từ khi khai báo.

- Biến cục bộ: Khai báo trong một khối, chỉ có tác dụng trong khối này.

3.2. Cách truyền tham số

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

Khi chương trình con được gọi để thi hành, tham trị được cấp ô nhớ và nhận giá trị là bản sao giá trị của tham số thực. Do đó, mặc dù tham trị cũng là biến, nhưng việc thay đổi giá trị của chúng không có ý nghĩa gì đối với bên ngoài hàm, không ảnh hưởng đến chương trình chính, nghĩa là không làm ảnh hưởng đến tham số thực tương ứng.

C hỗ trợ hai cách truyền tham số:

+ Truyền tham số bởi giá trị.

+ Truyền tham số bởi địa chỉ.

CÂU HỎI ÔN TẬP

1. Trình bày khái niệm và phân loại hàm?
2. Trình bày quy tắc hoạt động của hàm?

3. Trình bày các bước xây dựng hàm?
4. Phân biệt các loại tham số trong hàm?
5. Trình bày cách truyền tham số cho hàm?

CHƯƠNG 5: MẢNG

1. Khái niệm mảng

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

Ta có thể chia mảng làm 2 loại: *mảng 1 chiều và mảng nhiều chiều*.

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

2.Khai báo mảng

2.1. Khai báo mảng một chiều

2.1.1. Khai báo mảng với số phần tử xác định (khai báo tường minh)

Cú pháp: **<Kiểu> <Tên mảng ><[số phần tử]>**

Ý nghĩa:

- Tên mảng: đây là một cái tên đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên biến mảng.

- Số phần tử: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi kích thước của mảng là gì).

- Kiểu: mỗi phần tử của mảng có dữ liệu thuộc kiểu gì.

Ở đây, ta khai báo một biến mảng gồm có *số phần tử* phần tử, phần tử thứ nhất là *tên mảng* [*0*], phần tử cuối cùng là *tên mảng*[*số phần tử -1*]

Ví dụ: int a[10]; /* Khai báo biến mảng tên a, phần tử thứ nhất là a[0], phần tử

cuối cùng là a[9].*/

Ta có thể coi mảng a là một dãy liên tiếp các phần tử trong bộ nhớ như sau:

Vị trí	0	1	2	3	4	5	6	7	8	9
Tên phần tử:	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

2.1.2. Khai báo mảng với số phần tử không xác định (khai báo không tường minh)

Cú pháp: **<Kiểu> <Tên mảng> <[]>**

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

a. Vừa khai báo vừa gán giá trị

Cú pháp:

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

Nếu vừa khai báo vừa gán giá trị thì mặc nhiên C sẽ hiểu số phần tử của mảng là số giá trị mà chúng ta gán cho mảng trong cặp dấu {}. Chúng ta có thể sử dụng hàm sizeof() để lấy số phần tử của mảng như sau:

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

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

2.2. Khai báo mảng hai chiều

2.2.1. Khai báo mảng 2 chiều tường minh

***Cú pháp*: <Kiểu> <Tên mảng><[Số phần tử chiều 1]><[Số phần tử chiều 2]>**

Ví dụ: Người ta cần lưu trữ thông tin của một ma trận gồm các số thực. Lúc này ta có thể khai báo một mảng 2 chiều như sau:

float m[8][9]; /* Khai báo mảng 2 chiều có 8*9 phần tử là số thực*/ Trong trường hợp này, ta đã khai báo cho một ma trận có tối đa là 8 dòng, mỗi dòng có tối đa là 9 cột. Hình ảnh của ma trận này được cho trong sau:

Dòng\Cột	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0									
1									
2									
3									
4									
5									
6									
7									

Ma trận được mô tả là 1 mảng 2 chiều

2.2.2. Khai báo mảng 2 chiều không tường minh

Để khai báo mảng 2 chiều không tường minh, ta vẫn phải chỉ ra số phần tử của chiều thứ hai (chiều cuối cùng).

Cú pháp: <Kiểu> <Tên mảng> <[]><[Số phần tử chiều 2]>

Cách khai báo này cũng được áp dụng trong trường hợp vừa khai báo, vừa gán trị hay đặt mảng 2 chiều là tham số hình thức của hàm.

3. Truy xuất mảng

3.1. Truy xuất mảng một chiều

Mỗi phần tử của mảng được truy xuất thông qua Tên biến mảng theo sau là chỉ số nằm trong cặp dấu ngoặc vuông [ ]. Chẳng hạn a[0] là phần tử đầu tiên của mảng a được khai báo ở trên. Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

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

Ví dụ 1: Đổi một số nguyên dương thập phân thành số nhị phân. Việc chuyển đổi này được thực hiện bằng cách lấy số đó chia liên tiếp cho 2 cho tới khi bằng 0 và lấy các số dư theo chiều ngược lại để tạo thành số nhị phân. Ta sẽ dùng mảng một chiều để lưu lại các số dư đó. Chương trình cụ thể như sau:

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

unsigned int N;
unsigned int Du;
unsigned int NhiPhan[20],K=0,i;
printf("Nhap vao so nguyen N= ");scanf("%d",&N);

do
{
```

```
Du=N % 2;
NhiPhan[K]=Du; /* Lưu số dư vào mảng ở vị trí K*/

K++; /* Tăng K lên để lần kế lưu vào vị trí kế*/

N = N/2;
} while(N>0);

printf("Dang nhi phan la: ");
for(i=K-1;i>=0;i--)

printf("%d",NhiPhan[i]);
getch();
return 0;

}
```

Ví dụ 2: Nhập vào một dãy n số và sắp xếp các số theo thứ tự tăng. Đây là một bài toán có ứng dụng rộng rãi trong nhiều lĩnh vực. Có rất nhiều giải thuật sắp xếp. Một trong số đó được mô tả như sau:

Đầu tiên đưa phần tử thứ nhất so sánh với các phần tử còn lại, nếu nó lớn hơn một phần tử đang so sánh thì đổi chỗ hai phần tử cho nhau. Sau đó tiếp tục so sánh phần tử thứ hai với các phần tử từ thứ ba trở đi ... cứ tiếp tục như vậy cho đến phần tử thứ n-1.

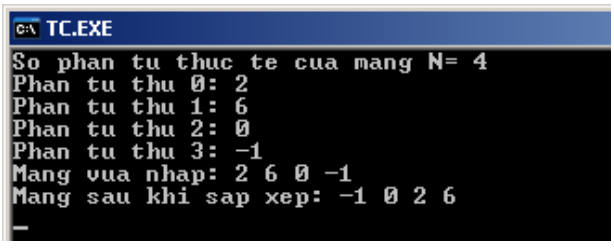
Chương trình sẽ được chia thành các hàm Nhap (Nhập các số), SapXep (Sắp xếp) và InMang (In các số); các tham số hình thức của các hàm này là 1 mảng không chỉ định rõ số phần tử tối đa, nhưng ta cần có *thêm số phần tử thực tế được sử dụng của mảng là bao nhiêu*, đây là một giá trị nguyên.

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

```
#include<stdio.h>
void Nhap(int a[],int N)
{

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

      printf("Phan tu thu %d: ",i);scanf("%d",&a[i]);
    }
}
void InMang(int a[], int N)
{

int i;
for (i=0; i<N;i++)
    printf("%d ",a[i]);
}

void SapXep(int a[], int N)
{

int t,i,j;
for(i=0;i<N;i++)
    for(j=i+1;j<N;j++)
        if (a[i]>a[j])

            {
```

```
            t=a[i];
            a[i]=a[j];
            a[j]=t;
          }
}

int main()
{

int a[20], N;
printf("So phan tu thuc te cua mang N= ");
scanf("%d",&N);

Nhap(a,N);
printf("Mang  vua  nhap:  ");
InMang(a,N);
SapXep(a,N); /* Gọi hàm sắp xếp*/

printf("Mang sau khi sap xep: ");
InMang(a,N);
getch();
return 0;
}
```

Kết quả chạy chương trình có thể là:



3.2. Truy xuất mảng hai chiều

Ta có thể truy xuất một phần tử của mảng hai chiều bằng cách viết ra **tên mảng**

theo sau là hai chỉ số đặt trong hai cặp dấu ngoặc vuông. Chẳng hạn ta viết m[2][3].

Với cách truy xuất theo cách này, **Tên mảng[Chỉ số 1][Chỉ số 2]** có thể coi là 1 biến có kiểu được chỉ ra trong khai báo biến mảng.

Ví dụ: Viết chương trình cho phép nhập 2 ma trận a, b có m dòng n cột, thực hiện phép toán cộng hai ma trận a,b và in ma trận kết quả lên màn hình.

Trong ví dụ này, ta sẽ sử dụng hàm để làm ngắn gọn hơn chương trình của ta. Viết các hàm: nhập 1 ma trận từ bàn phím, hiển thị ma trận lên màn hình, cộng 2 ma trận.

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

void Nhap(int a[][10],int M,int N)
{
 int i,j;

 for(i=0;i<M;i++)

         for(j=0; j<N; j++)

         {

              printf("Phan tu o dong %d cot %d: ",i,j);
              scanf("%d",&a[i][j]);

         }
}

void InMaTran(int a[][10], int M, int N)
{
     int i,j;
```

```
    for(i=0;i<M;i++)
    {
        for(j=0; j< N; j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
}

/* Cong 2 ma tran A & B ket qua la ma tran C*/
void CongMaTran(int a[][10],int b[][10],int M,int N,int c[]
  [10])
{
 int i,j;
 for(i=0;i<M;i++)
    for(j=0; j<N; j++)
        c[i][j]=a[i][j]+b[i][j];
}

int main()
{
 int a[10][10], b[10][10], M, N; int c[10][10];/* Ma tran
 tong*/
 printf("So dong M= "); scanf("%d",&M);
 printf("So cot M= "); scanf("%d",&N);
 printf("Nhap ma tran A\n"); Nhap(a,M,N);
 printf("Nhap ma tran B\n"); Nhap(b,M,N);
 printf("Ma tran A: \n");InMaTran(a,M,N);
 printf("Ma tran B: \n");InMaTran(b,M,N);
 CongMaTran(a,b,M,N,c);
 printf("Ma tran tong C:\n"); InMaTran(c,M,N);
```

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

CÂU HỎI ÔN TẬP

1. Trình bày khái niệm mảng?
2. Trình bày cách khai báo mảng một chiều và mảng hai chiều?
3. Trình bày cách truy xuất mảng một chiều và mảng hai chiều?

TÀI LIỆU THAM KHẢO

1. GS. Phạm Văn Ất, Ngôn ngữ lập trình C lý thuyết và thực hành, Bộ năng lượng Hà Nội, 1991.
2. Võ Văn Thành, Turbo C, Samis, 2001.
3. Nguyễn Minh San, Cẩm nang lập trình, NXB Giáo dục, 1993.
4. Ngôn ngữ lập trình C – Ebook, www.ebook.edu.vn
5. Lập trình C – www.congdongcviet.com
6. Michael Tischer, System Programing.