Chương 1: **TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH**

Mục tiêu:

- Hiểu được lịch sử phát triển của ngôn ngữ
- Biết được ngữ này có những ứng dụng thực tế như thế nào - Biết cách khởi động được và thoát khỏi chương trình.
- Sử dụng được hệ thống trợ giúp từ help file

I. Giới thiệu

Tiền thân của ngôn ngữ lập trình C là ngôn ngữ BCPL (Basic Conbined Programming Language) do Martin Richards nghiên cứu. Ảnh hưởng của ngôn ngữ BCPL lên ngôn ngữ lập trình C gián tiếp thông qua ngôn ngữ B, do Ken Thompson viết năm 1970 cho hệ điều hành UNIX chạy trên họ máy tính PDP – 7.

Nhu cầu cải tiến và phát triển cho UNIX đã thúc đẩy Dennis Ritchie và Brian Kernighan sáng tạo ra ngôn ngữ lập trình C ngay tại phòng thí nghiệm BELL (Hoa Kỳ) vào đầu những năm 70 nhằm mục đích ban đầu là phát triển một ngôn ngữ hệ thống mềm dẻo thay thế cho ngôn ngữ Assembly vốn nặng nề và "cứng nhắc" với các thiết bị phần cứng.

Ngôn ngữ lập trình C đặc biệt khác với ngôn ngữ BCPL và ngôn ngữ B ở chỗ : ngôn ngữ BCPL và ngôn ngữ B chỉ có duy nhất một kiểu dữ liệu là *từ máy,* trong khi đó ngôn ngữ lập trình C đã có các đối tượng dữ liệu cơ bản như *kí tự, các kiểu số nguyên và các kiểu số thực*. Đặc biệt *con trỏ* trong ngôn ngữ lập trình C tạo ra thêm được rất nhiều tính ưu việt.

Sau khi ra đời, đặc biệt thành công với hệ điều hành UNIX, ngôn ngữ lập trình C bắt đầu được phổ biến rộng rãi và người ta đã nhận thấy sức mạnh của nó. C là ngôn ngữ lập trình tương đối vạn năng, có mức độ thích nghi cao, mềm dẻo. Khác với ngôn ngữ Pascal, là ngôn ngữ lập trình có cấu trúc rất chặt chẽ và thường được dùng để giảng dạy về lập trình đặc biệt trong các trường đại học, thì ngôn ngữ lập

trình C lại được sử dụng rộng rãi trong các lĩnh vực chuyên nghiệp vì tính hiệu quả và mềm dẻo của nó.

Vào những năm 80 do nhu cầu trong việ xử lý dữ liệu ngày một cao, các chương trình viết ra ngày một phức tạp, việc bảo dưỡng chương trình ngày một khó khăn dẫn đến một phong cách lập trình mới – lập trình hướng đối tượng (OOP – Object Oriented Programming) xuất hiện và ngôn ngữ lập trình C bắt đầu được trang bị thêm khả năng lập trình hướng đối tượng, ngôn ngữ lập trình C^{++} ra đời từ đó và ngày càng chiếm ưu thế.

Hiện nay có rất nhiều bộ chương trình biên dịch (Compiler) và liên kết (Link) cho ngôn ngữ lập trình C của nhiều hãng khác nhau và mỗi bộ chương trình đều có những ưu, nhược điểm riêng. Xếp ở vị trí hàng đầu có thể kể đến Turbo C của hãng Borland, MS C của hãng Microsoft, ZORTECH C của hãng SYMANTEC. Phần mềm Turbo C được sử dụng khá rộng rãi vì nó cung cấp cho người dùng một thư viện khá đầy đủ các hàm vào ra, truy cập đồ họa. Tuy nhiên khả năng tối ưu mã của nó không bằng MS C. Trong giáo trình này chúng ta sử dụng Turbo C vì tính tiện lợi và phổ dụng của nó.

II. Khởi động và thoát khỏi C ❁

Khởi động:

- *Khởi động tại cửa sổ MS DOS* : 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)

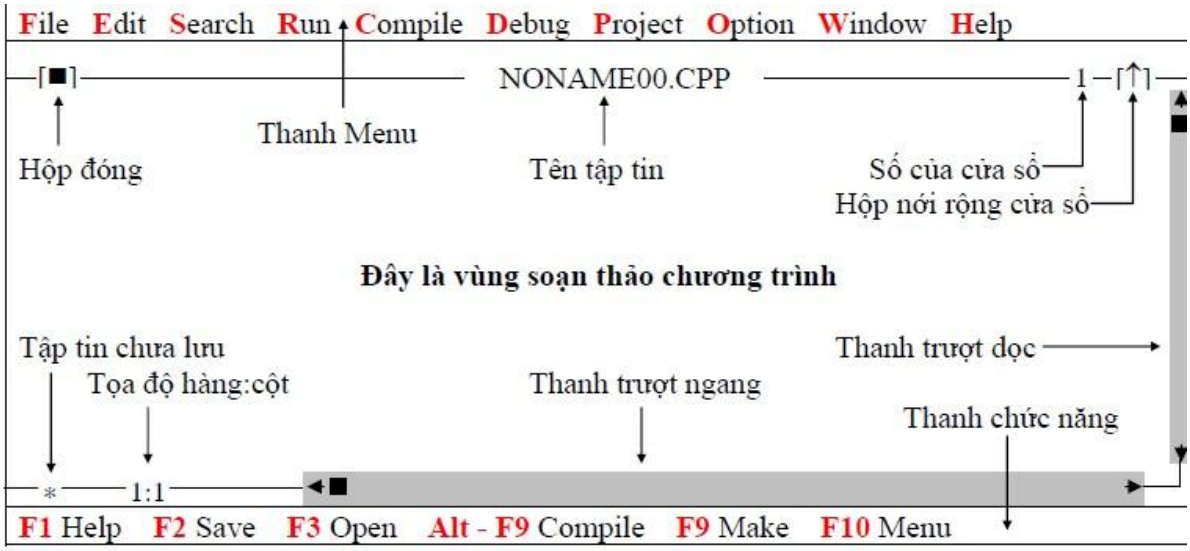
Nếu bạn muốn vừa khởi động BC vừa soạn thảo chương trình với một tập tin có tên do chúng ta đặt, thì gõ lệnh: **BC [đường dẫn]<tên file cần soạn thảo>**, nếu tên file cần soạn thảo đã có thì được nạp lên, nếu chưa có sẽ được tạo mới. ○ *Khởi động tại Windows*: Bạn vào menu Start, chọn Run, bạn gõ vào hộp Open 1 trong các dòng lệnh như nhập tại DOS. Hoặc bạn vào Window Explorer, chọn ổ đĩa chứa thư mục

BORLAND C, vào thư mục BORLAND C, 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.

Màn hình sau khi khởi động thành công



❁ ***Thoát***

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

Hoặc ấn tổ hợp phím **Alt – X**.

III. Hệ thống thông tin giúp đỡ

- 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.

Chương 2: CÁC THÀNH PHẦN CƠ BẢN

Mục tiêu:

- Hiểu và sử dụng được hệ thống kí hiệu và từ khóa
- Hiểu được các kiểu dữ liệu

- Hiểu được và vận dụng được các loại biến, hằng biểu thức cho từng chương trình cụ thể.
- Biết, hiểu và so sánh được các lệnh, khối lệnh - Thực hiện được việc chạy chương trình

1. Hệ thống ký hiệu và từ khóa *Thời gian: 1h*

2. Các kiểu dữ liệu *Thời gian: 1h*

3. Biến, Hằng,biểu thức *Thời gian: 2h*

4. Các phép toán *Thời gian: 2h*

5. Lệnh, khối lệnh *Thời gian: 1h*

6. Lệnh gán, lệnh xuất nhập, lệnh gán kết hợp *Thời gian: 2h*

7. Cách chạy chương trình *Thời gian: 1h*

I. Hệ thống ký hiệu và từ khóa

❁ *Hệ thống ký hiệu*

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 ax2 +bx+c=0 , ta cần tính biệt thức Delta $\Delta = b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự Δ, 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 :

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. ❁ 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.

II. Các kiểu dữ liệu

Trong C sử dụng các các kiểu dữ liệu sau :

1. 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	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).

2. Kiểu 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	-32.768 đến 32.767	2 byte
unsigned int	0 đến 65.535	2 byte
long	-2.147.483.648 đến 2.147.483.647	4 byte
unsigned long	0 đến 4.294.967.295	4 byte

Chú ý :

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

3. Kiểu dấu phẩy động :

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 hơn 3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

III. Biến, Hằng,biểu thức

1. 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, n;

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.

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 :

```
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***

2. Hằng

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

2.1. Tên hằng :

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.

2.2. Các loại hằng :

2.2.1 Hằng int :

Hằng int là số nguyên có giá trị trong khoảng từ -32.768 đến 32.767

Ví dụ :

#define number1 -50→ Định nghiã hằng int number1 có giá trị là -50

#define sodem 2732 → Định nghiã 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.

2.2.2 Hằng long :

Hằng long là số nguyên có giá trị trong khoảng từ -2.147.483.648 đến 2.147.483.647. 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

2.2.3 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

2.2.4 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 0xc1c2c3... hặc 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 hằng số h16 trong hệ 16 có giá trị như nhau. Giá trị của chúng trong hệ 10 là : 10*16+5=165.

2.2.5 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 : '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 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.

2.2.6. 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. 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. Biểu thức gán là biểu thức có dạng : v=e

Trong đó v là một biến (hay phần tử mảng), e là một biểu thức. Giá trị của biểu thức gán là giá trị của e, kiểu của nó là kiểu của v. Nếu đặt dấu ; vào sau biểu thức gán ta sẽ thu được phép toán gán có dạng :v=e;

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác.

Ví dụ như khi ta viết a=b=5; thì điều đó có nghĩa là gán giá trị của biểu thức b=5 cho biến a. Kết quả là b=5 và a=5.

Hoàn toàn tương tự như : a=b=c=d=6; gán 6 cho cả a, b, c và d

Ví dụ : z=(y=2)*(x=6); { ở đây * là phép toán nhân } gán 2
cho y, 6 cho x và nhân hai biểu thức lại cho ta z=12.

IV. Các phép toán, lệnh, khối lệnh

1. Phép toán số học

Các phép toán hai ngôi số học là

Phép toán	Ý nghiã	Ví dụ
+	Phép cộng	a+b
-	Phép trừ	a-b
*	Phép nhân	a*b
/	Phép chia	a/b (Chia số nguyên sẽ chặt phần thập phân)
%	Phép lấy phần dư	a%b (Cho phần dư của phép chia a cho b)

Có phép toán một ngôi - ví du -(a+b) sẽ đảo giá trị của phép cộng (a+b).

Ví dụ :

11/3=3

11%3=2

-(2+6)=-8

Các phép toán + và - có cùng thứ tự ưu tiên, có thứ tự ưu tiên nhỏ hơn các phép * , / , % và cả ba phép này lại có thứ tự ưu tiên nhỏ hơn phép trừ một ngôi.

Các phép toán số học được thực hiện từ trái sang phải. Số ưu tiên và khả năng kết hợp của phép toán được chỉ ra trong một mục sau này

2. Phép toán quan hệ và logic

Phép toán quan hệ và logic cho ta giá trị đúng (1) hoặc giá trị sai (0). Nói cách khác, khi các điều kiện nêu ra là đúng thì ta nhận được giá trị 1, trái lại ta nhận giá trị 0.

Các phép toán quan hệ là:

Phép toán	Ý nghĩa	Ví dụ
>	So sánh lớn hơn	a>b 4>5 có giá trị 0
>=	So sánh lớn hơn hoặc bằng	a>=b 6>=2 có giá trị 1
<	So sánh nhỏ hơn	a<b 6<7 có giá trị 1
<=	So sánh nhỏ hơn hoặc bằng	a<=b 8<=5 có giá trị 0
==	So sánh bằng nhau	a==b 6==6 có giá trị 1
!=	So sánh khác nhau	a!=b 9!=9 có giá trị 0

Bốn phép toán đầu có cùng số ưu tiên, hai phép sau có cùng số thứ tự ưu tiên nhưng thấp hơn số thứ tự của bốn phép đầu.

Các phép toán quan hệ có số thứ tự ưu tiên thấp hơn so với các phép toán số học, cho nên biểu thức : i<n-1 được hiểu là i<(n-1). **Các phép toán logic :**

Trong C sử dụng ba phép toán logic :

- Phép phủ định một ngôi ! ❁ Phép và (AND) &&
- Phép hoặc (OR) ||

Các phép quan hệ có số ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và ||, vì vậy biểu thức như :(a<b)&&(c>d) có thể viết lại thành : a<b && c>d **Chú ý :** Cả a và b có thể là nguyên hoặc thực.

3. Chuyển đổi kiểu giá trị

Việc chuyển đổi kiểu giá trị thường diễn ra một cách tự động trong hai trường hợp sau :

- Khi gán biểu thức gồm các toán hạng khác kiểu.
- Khi gán một giá trị kiểu này cho một biến (hoặc phần tử mảng) kiểu khác. Điều này xảy ra trong toán tử gán, trong việc truyền giá trị các tham số thực sự cho các đối. Ngoài ra, ta có thể chuyển từ một kiểu giá trị sang một kiểu bất kỳ mà ta muốn bằng phép chuyển sau :

(type) biểu thức

Ví dụ : (float) (a+b)

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 kiểu cao hơn. Chẳng hạn :

Giữa int và long thì int chuyển thành long.

Giữa int và float thì int chuyển thành float.

Giữa float và double thì float chuyển thành double.

Ví dụ :

1.5*(11/3)=4.51.5*11/3=5.5

(11/3)*1.5=4.5

Chuyển đổi kiểu thông qua phép gán :

Giá trị của vế phải được chuyển sang kiểu vế trái đó là kiểu của kết quả. Kiểu int có thể được được chuyển thành float. Kiểu float có thể chuyển thành int do chặt đi phần thập phân.

Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int bằng cách cắt bỏ một vài chữ số.

Ví dụ : int

n;

n=15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức :

Theo cách này, kiểu của biểu thức được đổi thành kiểu type theo nguyên tắc trên.

Ví dụ :

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

Đối với hàm toán học của thư viện chuẩn, thì giá trị của đối và giá trị của hàm đều có kiểu double, vì vậy để tính căn bậc hai của một biến nguyên n ta phải dùng phép ép kiểu để chuyển kiểu int sang double như sau : sqrt((double)n)

Phép ép kiểu có cùng số ưu tiên như các toán tử một ngôi. **Chú ý** :

Muốn có giá trị chính xác trong phép chia hai số nguyên cần dùng phép ép kiểu : ((float)a)/b

Để đổi giá trị thực r sang nguyên, ta dùng :

(int)(r+0.5)

Chú ý thứ tự ưu tiên : (int)1.4*10=1*10=10

(int)(1.4*10)=(int)14.0=14

4. Phép toán tăng giảm

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng là ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm -- thì sẽ trừ toán hạng đi 1.

Ví dụ : n=5

++n Cho ta n=6

--n Cho ta n=4

Ta có thể viết phép toán ++ và -- trước hoặc sau toán hạng như sau : ++n, n++, --n, n--. Sự khác nhau của ++n và n++ ở chỗ : trong phép n++ thì tăng sau khi giá trị của

nó đã được sử dụng, còn trong phép ++n thì n được tăng trước khi sử dụng. Sự khác nhau giữa n-- và --n cũng như vậy.

Ví dụ : n=5 x=++n Cho

ta x=6 và n=6 x=n++ Cho

ta x=5 và n=6

5. Lệnh

Là một tác vụ, biểu thức, hàm, cấu trúc điều khiển…

Ví dụ 1: x = x + 2;

printf("Day la mot lenh\n");

6. Khối lệnh

Là một dãy các câu lệnh được bọc bởi cặp dấu { }, các lệnh trong khối lệnh phải viết thụt vô 1 tab so với cặp dấu { }

Ví dụ 2:

```
{ //dau khoi a
        = 5;
        b = 6;                  //viết thụt vô 1 tab so với cặp { } printf("Tong
        %d + %d = %d", a, b, a+b);
} //cuoi khoi
```

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

1. Lệnh gán:

- Lệnh gán có dạng : Biến = Biểu thức

Ví dụ : a = b + c

Các toán hạng bên trái bao giờ cũng phải là một biến

- Lệnh gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác.

Ví dụ: a=b=5; → nghĩa là gán giá trị của biểu thức b=5 cho biến a. Kết quả là b=5 và a=5.

a=b=c=d=6; → gán 6 cho cả a, b, c và d

2. Lệnh gán kết hợp

Có dạng : Biến pt = Biểu thức

Trong đó pt có thể là một trong các phép toán hai ngôi: +, -, *, /, %, <, >, &, |, ^

Ví dụ : x *= y + 3; → nghĩa là x = x * (y + 3);

a +=b; → nghĩa là a = a + b;

Người ta thường sử dụng lệnh gán kết hợp khi toán hạng bên phải là một biểu thức khá dài.

3. Lệnh xuất nhập

Xuất dữ liệu lên màn hình

Kết xuất dữ liệu được định dạng. ***Cú pháp* printf** ("chuỗi định dạng"[, đối mục 1, đối mục 2,…]); *Khi sử dụng hàm phải khai báo tiền xử lý* ***#include <stdio.h>*** - printf: tên hàm, ***phải viết bằng chữ thường.***

- đối mục 1,…: là các mục dữ kiện cần in ra màn hình. Các đối mục này có thể là biến, hằng hoặc biểu thức phải được định trị trước khi in ra.

- chuỗi định dạng: được đặt trong cặp nháy kép (" "), gồm 3 loại:

+ Đối với chuỗi kí tự ghi như thế nào in ra giống như vậy.

+ Đối với những kí tự chuyển đổi dạng thức cho phép kết xuất giá trị của các đối mục ra màn hình tạm gọi là mã định dạng. Sau đây là các dấu mô tả định dạng:

%c : Ký tự đơn

%s : Chuỗi

%d : Số nguyên thập phân có dấu

%f : Số chấm động (ký hiệu thập phân)

%e : Số chấm động (ký hiệu có số mũ)

%g : Số chấm động (%f hay %g)

%x : Số nguyên thập phân không dấu

%u : Số nguyên hex không dấu %o :

Số nguyên bát phân không dấu

l : Tiền tố dùng kèm với %d, %u, %x, %o để chỉ số nguyên dài (ví dụ %ld)

+ Các ký tự điều khiển và ký tự đặc biệt

\n : Nhảy xuống dòng kế tiếp canh về cột đầu tiên.

\t : Canh cột tab ngang.

\r : Nhảy về đầu hàng, không xuống hàng. \a

: Tiếng kêu bip.

\\ : In ra dấu \
\" : In ra dấu "

\' : In ra dấu '

%%: In ra dấu %

Ví dụ 1: printf("Bai hoc C dau tien. \n");

Kí tự điều khiển Chuỗi kí tự

➔ Kết quả in ra màn hình

Bai hoc C dau tien: -

Ví dụ 2: giả sử float a = 6.4, b= 1234.56, c=62.3

printf("%7.2d%7.2d%7.2d.\n", a, b, c);

7: độ rộng trường, .2 : số chữ số thập phân

Kết quả in ra màn hình:

6.40 1234.56 62.30 -

Bề rộng trường bao gồm : phần nguyên, phần lẻ và dấu chấm động.

Ví dụ 3: giả sử float a = 6.4, b = 1234.55, c = 62.34
printf("%10.1d%10.1d%10.1d.\n", a, b, c);
printf("%10.1d%10.1d%10.1d.\n", 165, 2, 965); Kết quả in ra màn hình:

6.4 1234.6 62.3 165.0 2.0 965.0	Số canh về bên phải bề rộng trường

printf("%-10.2d%-10.2d%-10.2d.\n", a, b, c); printf("%-10.2d%-10.2d%-10.2d.\n", 165, 2, 965);

6.40 1234.55 62.34 165.0 2.00 965.00	Số canh về bên trái bề rộng trường

 Đưa dữ liệu vào từ bàn phím

Định dạng khi nhập liệu ➢

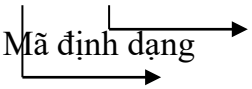
Cú pháp :

scanf(“chuỗi định dạng”[, đối mục 1, đối mục 2,….]);

Khi sử dụng hàm phải khai báo tiền xử lý #include <stdio.h>

- scanf : tên hàm, *phải viết bằng chữ thường*
- Khung định dạng : được đặt trong cặp nháy kép (“ ”) là hình ảnh dạng dữ liệu nhập vào.
- Đối mục 1,….. là danh sách các đối mục cách nhau bởi dấu phẩy, mỗi đối mục sẽ tiếp nhận giá trị nhập vào.

Ví dụ 1: scanf(“%d”, &i);



Nhập vào 12abc, biến i chỉ nhận giá trị 12. Nhập 3,4 chỉ nhận giá trị 3

Ví dụ 2: scanf("%d%d", &a, &b);

 Nhập vào 2 số a, b phải cách nhau bằng **khoảng trắng** hoặc **enter**.

Ví dụ 3: scanf("%d/%d/%d", &ngay, &thang, &nam);

 Nhập vào ngày, tháng, năm theo dạng ngay/thang/nam (20/12/2002) **Ví dụ 4:** scanf("%d%*c%d%*c%d", &ngay, &thang, &nam);

 Nhập vào ngày, tháng, năm với dấu phân cách /, -,…; ngoại trừ số. **Ví dụ 5**: scanf("%2d%2d%4d", &ngay, &thang, &nam);

 Nhập vào ngày, tháng, năm theo dạng dd/mm/yyyy.

VI. Cách chạy chương trình

Trình đơn Run (Alt – R) gồm các chức năng để chạy một chương trình của Turbo C đang hiện hữu trên màn hình, cụ thể gồm các lệnh như sau: - Run <Ctrl – F9> : Biên dịch và chạy chương trình hiện hành.

- Program reset <Ctrl -F2> : Khởi tạo lại chương trình đang chạy (Reset running program) sau khi đã gỡ rối.
- Go to cursor <F4> : Biên dịch và chạy từ đầu chương trình cho đến vị trí con trỏ.

  - Trace into <F7> và Step over <F8> : Chạy chương trình theo từng bước (lệnh), giúp ta quan sát, kiểm tra và sửa chữa chương trình.

- User screen <Alt – F5> : Xem kết quả trên màn hình nếu trong đoạn chương trình không có lệnh gọi hàm getch(). Sau khi xem xong bấm phím bất kỳ để trở lại màn hình của turbo C.

Chương 3: **CÁC LỆNH CẤU TRÚC**

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

1. Lệnh rẽ nhánh có điều kiện if:

1.1. Lệnh if-else :

Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau :

```
if ( biểu thức )
khối lệnh 1; /*
Dạng một */ if
( biểu thức )
khối lệnh 1;
else khối lệnh
2 ;
/* Dạng hai */
```

Hoạt động của biểu thức dạng 1 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ :

Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình. Chương trình có thể viết bằng cả hai cách trên như sau :

```
#include "stdio.h"
main()
{ float a,b,max; printf("\n Cho a="); scanf("%f",&a); printf("\n Cho b=");
        scanf("%f",&b); max=a; if (b>max) max=b; printf(" \n Max cua hai so
        a=%8.2f va b=%8.2f la Max=%8.2f",a,b,max);
}
```

```
#include "stdio.h"
main()
{ float a,b,max; printf("\n Cho a="); scanf("%f",&a); printf("\n Cho b=");
        scanf("%f",&b); if (a>b) max=a; else max=b; printf(" \n Max cua hai so
        a=%8.2f va b=%8.2f la Max=%8.2f",a,b,max);
}
```

Sự lồng nhau của các toán tử if :

C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else. Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất. Chẳng hạn như đoạn chương trình ví dụ sau :

```
if ( n>0 ) /* if thứ nhất*/
if ( a>b ) /* if thứ hai*/
z=a; else z=b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với :

```
if ( n>0 ) /* if thứ nhất*/
{ if ( a>b ) /* if thứ hai*/ z=a;
        else z=b;
}
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau :

```
if ( n>0 ) /* if thứ nhất*/
{ if ( a>b ) /* if thứ hai*/ z=a;
} else z=b;
```

1.2. Lệnh else-if :

Khi muốn thực hiện một trong n quyết định ta có thể sử dụng cấu trúc sau :

```
if ( biểu thức 1 )
khối lệnh 1; else if (
biểu thức 2 ) khối
lệnh 2;
.....
else if ( biểu thức n-1
) khối lệnh n-1; else
khối lệnh n;
```

Trong cấu trúc này, máy sẽ đi kiểm tra từ biểu thức 1 trở đi đến khi gặp biểu thức nào có giá trị khác 0.

Nếu biểu thức thứ i (1,2, ...n-1) có giá trị khác 0, máy sẽ thực hiện khối lệnh i, rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Nếu trong cả n-1 biểu thức không có biểu thức nào khác 0, thì máy sẽ thực hiện khối lệnh n rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Ví dụ :

Chương trình giải phương trình bậc hai.

```
#include "stdio.h" main()
```

```
{ float a,b,c,d,x1,x2; printf("\n
        Nhap a, b, c:");
        scanf("%f%f%f,&a&b&c); d=b*b-4*a*c; if (d<0.0)
        printf("\n Phuong trinh vo nghiem "); else if (d==0.0)
        printf("\n Phuong trinh co nghiem kep x1,2=%8.2f",-
        b/(2*a));
else
{ printf("\n Phuong trinh co hai nghiem ");
printf("\n x1=%8.2f",(-b+sqrt(d))/(2*a));
printf("\n x2=%8.2f",(-b-sqrt(d))/(2*a)); }
```

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

Là cấu trúc tạo nhiều nhánh đặc biệt. Nó căn cứ vào giá trị một biểu thức nguyên để để chọn một trong nhiều cách nhảy.

Cấu trúc tổng quát của nó là :

```
switch ( biểu thức nguyên )
{
        case n1
        khối lệnh 1
        case n2
        khối lệnh 2
        .......
        case nk
        khối lệnh k
        [ default
        khối lệnh k+1 ]
}
```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch. default là một thành phần không bắt buộc phải có trong thân của switch. Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc ( ) như sau :

- Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni. Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau :
- Khi có default máy sẽ nhảy tới câu lệnh sau nhãn default.
- Khi không có default máy sẽ nhảy ra khỏi cấu trúc switch.

Chú ý :Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch. Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ :

Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch :

```
#include "stdio.h" main()
{ int diem; tt: printf("\nVao du lieu
:"); printf("\n Diem =");
scanf("%d",&diem);
switch (diem)
{ case 0: case 1: case 2: case 3:printf("Kem\n");break;
case 4:printf("Yeu\n");break;
case 5: case 6:printf("Trung binh\n");break;
case 7: case 8:printf("Kha\n");break;
case 9: case 10:printf("Gioi\n");break;
default:printf(Vao sai\n);
} printf("Tiep tuc 1, dung 0 :")
scanf("%d",&diem);
if (diem==1) goto tt; getch(); return;
}
```

3. Các lệnh break, continue, goto :

3.1. 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);
}
```

3.2. 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); }
```

3.3. 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:

Cú pháp:

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 vòng lặp for :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:

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:

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 ");
}
```

Chương 4: **HÀM**

Mục tiêu:

- Hiểu được khái niệm hàm
- Trình bày được qui tắc xây dụng 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.

1. Khái niệm :

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().

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.

2. Quy tắc xây dựng một hàm :

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

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.

Trước khi sử dụng một hàm ta phải khai báo kiểu giá trị của nó theo cú pháp sau:

Kiểu tên hàm ();

4. Nguyên tắc hoạt động của hàm :

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.

5. Cách truyền tham số : 5.1 Truyền bằng trị VD :

```
void hoanvi(a,b)
{
    int tam;
    tam=a;
a=b; b=tam;
}
main()
{
        int a,b;
    printf("\n nhap so thu nhat:"); scantf("%d", &a);
    printf("\n nhap so thu hai:"); scantf("%d", &b);
    hoanvi(a,b);
   printf("\n sau khi hoan vi so thu nhat
   =%d",a);  printf("\n con so thu hai =%d",b);
   getch();
}
```

* Giải thích :

5.2 Truyền bằng biến

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.

6. Hàm không cho các giá trị :

- Các hàm không cho giá trị giống như thủ tục (procedure) trong ngôn ngữ lập trình PASCAL. Trong trường hợp này, kiểu của nó là void.
- Ví dụ hàm tìm giá trị max trong ba số là max3s ở trên có thể được viết thành thủ tục hiển thị số cực đại trong ba số như sau : *void htmax3s(float a, float b, float c)*

```
{ float max; max=a; if
        (max<b) max=b; if
        (max<c) max=c;
}
```

Lúc này, trong hàm main ta gọi hàm htmax3s bằng câu lệnh :

htmax3s(x,y,z);

Chương 5: **KIỂU MẢNG**

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.

I. Khai báo mảng

Là tập hợp các phần tử có cùng dữ liệu. Giả sử bạn muốn lưu n số nguyên để tính trung bình, bạn không thể khai báo n biến để lưu n giá trị rồi sau đó tính trung bình.

Ví dụ 1 : bạn muốn tính trung bình 10 số nguyên nhập vào từ bàn phím, bạn sẽ khai báo 10 biến: a, b, c, d, e, f, g, h, i, j có kiểu int và lập thao tác nhập cho 10 biến này như sau:

printf("Nhap vao bien a:

"); scan f("%d", &a);

10 biến bạn sẽ thực hiện 2 lệnh trên 10 lần, sau đó tính trung bình:

(a + b + c + d + e + f + g + h + i + j)/10

Điều này chỉ phù hợp với n nhỏ, còn đối với n lớn thì khó có thể thực hiện được. Vì vậy khái niệm mảng được sử dụng

➢ **Cách khai báo mảng**

KieuDLptu Tenmang[sopt];

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***. ➢

Tham chiếu đến từng phần tử mảng

Sau khi mảng được khai báo, mỗi phần tử trong mảng đều có chỉ số để tham chiếu. Chỉ số bắt đầu từ 0 đến n-1 (với n là kích thước mảng). Trong ví dụ trên, ta khai báo mảng 10 phần tử thì chỉ số bắt đầu từ 0 đến 9. 0 1 2 3 4 5 6 7 8 9

Cách tham chiếu : Tên mảng [chỉ số]

➢ **Nhập dữ liệu 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]);
}
```

- **Đọc dữ liệu từ mảng** *for(i = 0; i < 10; i++) printf("%3d ", ia[i]);*

```
VD : /* Tinh trung binh cong n so nguyen */
#include <stdio.h>
#include <conio.h> void
main(void)
{ int ia[50], i, in, isum = 0;
        printf("Nhap vao gia tri n: ");
        scanf("%d", &in); //Nhap du
        lieu vao mang for(i = 0; i <
        in; i++)
        { printf("Nhap vao phan tu thu %d: ", i + 1);
        scanf("%d", &ia[i]); //Nhap gia tri cho phan tu thu
        i
        }
        //Tinh tong gia tri cac phan tu for(i = 0; i < in;
        i++) isum += ia[i]; //cong don tung phan tu vao
        isum printf("Trung binh cong: %.2f\n", (float)
        isum/in); getch();
}
```

II. Mảng và tham số của hàm

Khi truyền mảng sang hàm, không tạo bản sao mảng mới. Vì vậy mảng truyền sang hàm có dạng tham biến. Nghĩa là giá trị của các phần tử trong mảng sẽ bị ảnh hưởng nếu có sự thay đổi trên chúng

VD1 : /* Chuong trinh tim so lon nhat su dung ham */

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

```
#define MAX 20
//Khai bao prototype
int max(int, int);
//ham tim so lon nhat trong mang 1 chieu
int max(int ia[], int in)
{ int i, imax; imax = ia[0]; //cho phan tu dau tien
        la max for (i = 1; i < in; i++) if (imax <
        ia[i]) //neu so dang xet > max imax =
        ia[i]; //gan so nay cho max return imax;
        //tra ve ket qua so lon nhat
}
void main(void)
{
        int ia[MAX]; int
        i = 0, inum; do
        { printf("Nhap vao mot so:
        "); scanf("%d", &ia[i]); }
        while (ia[i++] != 0);
        i--;
        inum = max(ia, i); printf("So lon
        nhat la: %d.\n", inum); getch();
}
```

Chương trình ban đầu hàm max có hai tham số truyền vào và kết quả trả về là giá trị max có kiểu nguyên, một tham số là mảng 1 chiều kiểu int và một tham số có kiểu int. Với chương trình sau khi sửa hàm max chỉ còn một tham số truyền vào nhưng cho kết quả như nhau. Do sau khi sửa chương trình mảng a[MAX] được khai báo lại là biến toàn cục nên hàm max không cần truyền tham số mảng vào cũng có thể sử dụng được. Tuy vậy, khi lập trình bạn nên viết như chương trình ban đầu là truyền tham số mảng vào (dạng tổng quát) để hàm max có thể thực hiện được trên nhiều

mảng khác nhau. Còn với chương trình sửa lại bạn chỉ sử dụng hàm max được với mảng a mà thôi.

VD2 : Tìm số lớn nhất của 3 mảng a, b, c

```
#include <stdio.h>
#include <conio.h>
#define MAX 20
//Khai bao prototype
int max(int, int); int
input(int);
//ham tim phan tu lon nhat trong mang 1 chieu int
max(int ia[], int in)
{ int i, imax; imax = ia[0]; //cho phan tu dau tien
        la max for (i = 1; i < in; i++) if (max <
        ia[i]) //neu so dang xet > max max = ia[i];
        //gan so nay cho max return imax; //tra ve
        ket qua so lon nhat
}
//ham nhap lieu vao mang 1 chieu int
input(int ia[])
{
        int i = 0;
        do
        { printf("Nhap vao mot so:
        "); scanf("%d", &ia[i]); }
        while (ia[i++] != 0);
        i--;
        return i;
}
void main(void)
{
```

```
    int ia[MAX], ib[MAX], ic[MAX]; int inum1, inum2, inum3;
    printf("Nhap lieu cho mang a: \n"); inum1 = max(ia, input(ia));
    printf("Nhap lieu cho mang b: \n"); inum2 = max(ib, input(ib));
    printf("Nhap lieu cho mang c: \n"); inum3 = max(ic, input(ic));
    printf("So lon nhat cua mang a: %d, b: %d, c: %d.\n", inum1,
    inum2, inum3); getch();
}
```

Giải thích chương trình

Hàm input có kiểu trả về là int thông qua biến i (cho biết số lượng phần tử đã nhập vào) và 1 tham số là mảng 1 chiều kiểu int. Dòng 41, 43, 45 lần lượt gọi hàm input với các tham số là mảng a, b, c. Khi hàm input thực hiện việc nhập liệu thì các phần tử trong mảng cũng được cập nhật theo.

III. Sắp xếp mảng

Trong thực tế ta thường gặp những công việc cần phải sắp xếp theo một thứ tự nào đó, chẳng hạn danh sách học sinh trong lớp có thể được sắp xếp theo Họ hoặc Điểm trung bình. Việc sắp xếp thứ tự 1 mảng giúp ta dễ dàng tìm kiếm các phần tử của mảng.

Từ trước đến nay người ta đã tìm ra nhiều phương pháp sắp xếp mảng như:

- Phương pháp chèn vào thẳng (Straightinsertion)
- Phương pháp chèn vào chia đôi hay còn gọi là phương pháp chèn nhị phân (binaryinsertion)
- Phương pháp chọn lựa thẳng (Straightselection) - Các phương pháp đổi chỗ gồm: ○ Phương pháp nổi bọt (bubble sort) ○ Phương pháp rung (shakesort)
- Các phương pháp sắp xếp khác gồm có:
    - Phương pháp shell ○ Phương pháp Heapsort ○ Phương pháp Quicksort

Phần dưới đây trình bày hai giải thuật sắp xếp thường được sử dụng nhiều nhất đó là phương pháp Bubble sort và Quicksort

4. Phương pháp Bubble sort

Phương pháp nổi bọt (Bubble sort) hay còn gọi là phương pháp sắp xếp cổ điển dựa trên nguyên lý so sánh và đổi chỗ liên tiếp các cặp phần tử đứng liền kề nhau cho tới khi tất cả các phần tử được sắp xếp theo thứ tự tăng dần (hoặc giảm dần). Các bước sắp xếp gồm:

- Trước hết tìm phần tử lớn nhất của mảng, đặt nó vào vị trí n, tiếp đó:
    - Rồi lại tìm phần tử lớn thứ nhì của mảng, đặt nó vào vị trí thứ n – 1 ○ Cuối cùng tìm phần tử lớn thứ n -1 (tức là bé thứ nhì) của mảng, đặt vào vị trí thứ hai.
    - Như vậy phần tử x[1] còn lại hiển nhiên là phần tử bé nhất của mảng.

Quá trình trên có thể viết vắn tắt) như sau:

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

- Để tìm phần tử lớn thứ i của mảng :
    - So sánh x[1] với x[2]. Nếu x[1] > x[2] thì nghĩa là 2 phần tử này phải đổi chỗ cho nhau sao cho x[1] phải nhỏ hơn x[2] bằng giải thuật đổi chỗ:

```
tam:=x[1];
x[1]:=x[2];
x[2]:=tam;
```

    - Tới đây ta có x[1] <= x[2]. Ta lại so sánh x[2] với x[3] để xem x[2] có hơn x[3] hay không? Nếu không ta lại đổi chỗ cho chúng. Quá trình này cứ tiếp diễn cho tới phần tử cuối cùng của mảng.

Quá trình trên có thể viết vắn tắt như sau:

```
for(j=1; j<n; j++)
{
    if (x[i] > x[j])
    { tam:=x[j];
        x[j]:=x[i];
        x[i]:=tam
        ;
```

```
                    }
                }
```

Tóm lại toàn bộ giải thuật Bubble sort để sắp xếp thứ tự tăng dần của một mảng như sau:

```
#define SIZE 10 main()
        {
            Int a[SIZE];
            …..
        }
sapxep(int x[ ])
        {
            Int i, j, tam;
      for(i=0; i< SIZE; i++)
                    for(j=1; j<n; j++)
                    {
                        if (x[i] > x[j])
{
tam:=x[j];
x[j]:=x[i];
x[i]:=tam;
                        }
                    }
}
```

Ví dụ 1:

- Mảng ban đầu : 5 9 10 8 i = 1 j 10 9 = 1 5 8 10 j = 2 5 9 8 8 j = 3 5 9 10

i = 2 j = 1 5 9 8

j = 2 5 8 9

i = 3 j = 1 5 8

Mảng sắp xếp: 5 8 9 10

Ví dụ 2: Tạo mảng gồm 1000 phần tử ngẫu nhiên sau đó sắp xếp lại theo thứ tự tăng dần.

```
#include <stdio.h>
#include <stdlib.h> #define
MAX 1000
int a[MAX], I, j, tam; main()
{
        randomize();
        printf("\n Tao mot mang gom 1000 phan tu ngau nhien");
        printf("\n Bam phim bat ky de tao mang"); getch();
        for(i=0; i<MAX; a[i++] = random(3000));
        for(i=0; i<MAX; printf("%8d", a[i++]));
        for(i=0; i< MAX; i++)
                for(j=1;    j<MAX;    j++)
                        if (a[i] < a[j])
                           { tam:=a[j];
                                a[j]:=a[i];
                                a[i]:=tam
                                ;
                           }
        printf("\n Mang sau khi sap xep.");
        for(i=0; i<MAX; printf("%8d", a[i++]));
        getch()
}
```

5. Phương pháp Quicksort

Thông thường đối với những mảng có kích thước lớn (nhiều phần tử) người ta hay dùng phương pháp Quicksort để sắp xếp mảng.

Nguyên tắc của phương pháp này là chia mảng thành 2 phần bên trái(l) và bên phải (r) lấy giá trị của phần tử ở giữa (X) làm chuẩn để so sánh.

- Đi tìm một phần tử A của phần bên trái có giá trị lớn hơn X
- Đi tìm một phần tử B của phần bên phải có giá trị nhỏ hơn X
- Hoán vị 2 phần tử A và B (nếu sắp xếp theo chiều tăng dần)
- Tiếp tục thực hiện như vậy cho đến khi đạt được phần bên trái của mảng chứa những giá trị nhỏ hơn X, phần bên phải của mảng chứa những giá trị lớn hơn X.
- Chia mảng thành 2 mảng con và thực hiện lại phương pháp sắp xếp cho 2 mảng này (bằng cách sử dụng thuật toán đệ quy). Như vậy từ 2 mảng con thành 4 mảng…cho đến khi không thể chia được nữa thì việc sắp xếp cũng hoàn tất.

➢ Giải thuật:

- Chia mảng thành 2 phần: bên trái (L) và bên phải (R). Lấy giá trị phần tử ở giữa và gán vào biến X để làm chuẩn so sánh: x = a[(l + r)/2]
- Cho i ban đầu là L chỉ thứ tự các phần tử ở phía bên trái: i = l
  Cho j ban đầu là R chỉ thứ tự các phần tử ở phía bên phải: j = r
- Lặp lại:
  - Chừng nào a[i] < X thì tăng i : i = i+1
  - Chừng nào X < a[i] thì giảm j : j = j -1

  {Đến tại vị trí này i là thứ tự của phần tử có giá trị lớn hơn X và j là thứ tự của phần tử có giá trị nhỏ hơn X}
  - Nếu i<=j thì: Hoán vị 2 phần tử a[i] và a[j]. Tăng i(i++). Giảm j (j--).

  Cho đến khi i>j

  {Đến đây chúng ta được 2 mảng con:

Mảng L,j và mảng i,R (vì i lớn hơn j), với mảng L, j chứa các giá trị nhỏ hơn giá trị trong mảng i,R

Tiếp tục thực hiện giải thuật trên với 2 mảng con này bằng thuật toán đệ quy}

- Sắp xếp mảng từ thành phần L đến thành phần j ○ Sắp xếp mảng từ thành phần i đến thành phần R ○ Chấm dứt

Đoạn chương trình dưới đây minh họa giải thuật Quicksort vừa trình bày kể trên được viết dưới một hàm tự tạo.

```
……
quicksort(int l, int r)
{    int i, j, x , y;   i
= j;            j = r;
     x:= a[(l+r)/2];
        do
                {
                        while (a[i] < x) i =
                                i+1; while(
                                x<a[j]) j=j-1;
                                if(i<=j)
                                {
                                    y=a[i];
                                    a[i]=a[j];
                                    a[j]=y;
                                    i++;    j--;

                                }
        } while(i>j); if(l<j)
        quicksort(l,r); if(i<r)
        quicksort(i,r);
}
```

Nếu không cần hàm tự tạo, ta có thể sử dụng trực tiếp hàm Qsort của Turbo C với cú pháp sau:

qsort(<Tên mảng>, Số phần tử, Kích thước, so sánh); - Số phần tử: Là một biến nguyên chỉ số phần tử muốn sắp xếp.

- Kích thước : Là kích thước một phần tử của mảng.
- So sánh : Là một hàm tác động lên địa chỉ các phần tử cho ra các giá trị.
    - o >0 : Nếu lớn hơn (theo nghĩa nào đó)
    - o =0: Nếu bằng o <0: Nếu nhỏ hơn

IV. Gán giá trị cho mảng

Hai mảng A và B có cùng kiểu thành phần (các phần tử mảng có cùng kiểu dữ liệu) có thể chuyển giá trị cho nhau bằng lệnh memmove, cú pháp như sau:

memmove(b, a, sizeof(a)) a: tên mảng nguồn b: tên mảng đích sizeof(a): Số phần tử chép.

Ví dụ : Viết chương trình tạo mảng ngẫu nhiên A gồm 10 phần tử. Gán giá trị của A cho mảng B. #include <stdio.h>

```
#include <stdlib.h>
#define MAX 10 int
a[MAX], b[MAX], i;
main()
{
    randomize();
    printf("\n Tao mang a");
for(i=0; i<MAX; i++) a[i] = 3*i
+ 5; memmove(b,a, sizeof(a));
for(i=0; i<MAX; i++);
        printf("\n Ta có a[%d] = 2%d va b[%d] = %2d", i, a[i], i, b[i]); getch()
}
```

Chương 6: **CHUỖI KÝ TỰ**

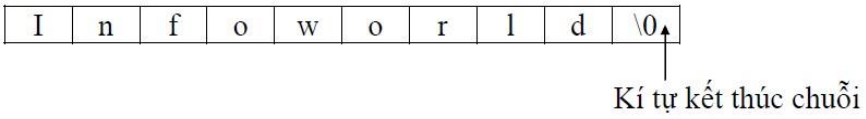
Mục tiêu:

- Hiểu được thế nào là chuỗi kí tự
- 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. ***I. Khái niệm***

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:



II. Khai báo biến chuỗi

Cách khai báo chuỗi : Char tenbienchuoi[kichthuoc];

Ví dụ : char cname[30];

Ý nghĩa: ***Khai báo chuỗi cname có chiều dài 30 kí tự.*** Do chuỗi kết thúc bằng kí tự null, nên khi bạn khai báo chuỗi có chiều dài 30 kí tự chỉ có thể chứa 29 kí tự.

```
VD : /* Chuong trinh nhap va in ra ten*/
#include <stdio.h>
#include <conio.h>
void main(void)
{
        char cname[30]; printf("Cho
        biet ten cua ban: "); scanf("%s",
        cname); printf("Chao ban %s\n",
        cname); getch();
}
```

III. Nhập chuỗi ký tự

Dùng hàm scanf để nhập chuỗi có hạn chế như sau: Khi bạn thử lại chương trình trên với dữ liệu nhập vào là Mai Lan, nhưng khi in ra bạn chỉ nhận được Mai. Vì hàm scanf nhận vào dữ liệu đến khi gặp khoảng trắng thì kết thúc

*** Hàm nhập (gets), xuất (puts) chuỗi**

Sử dụng hàm gets, puts phải khai báo #include <stdio.h>

VD : /* Chuong trinh nhap va in ra ten*/

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
        char cname[30]; puts("Cho
        biet ten cua ban: ");
        gets(cname); puts("Chao ban
        "); puts(cname); getch();
      }
```

Đối với hàm puts kí tự kết thúc chuỗi null (\0) được thay thế bằng kí tự newline (\n). Hàm gets và puts chỉ có 1 đối số và không sử dụng dạng thức trong nhập liệu cũng như xuất ra màn hình.

IV. Các phép toán chuỗi ký tự

V. Các thao tác trên chuỗi ký tự

Các hàm cơ bản trong thư viện string.h

1. gets(s1) : nhập dữ liệu vào chuỗi s1.
2. n = strlen(s1) : cho biết độ dài của chuỗi s1.
3. 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.

4. 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.

5. strcat (s1,s2) : nối s1 và s2 .

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

6. 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.

7. 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".
8. 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] = "\0").
9. 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)] .

10. 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

11. 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>

Chương 7: **BIẾN CON TRỎ**

Mục tiêu:

- Hiểu được về con trỏ trong ngôn ngữ lập trình
- 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

I. Biến con trỏ

1. Khái niệm:

Các biến được sử dụng từ trước đến nay đều là biến có kích thước và kiểu dữ liệu xác định, người ta gọi những biến này là biến tĩnh (static)

Khi chạy chương trình, gặp những biến này, máy sẽ cung cấp lượng bộ nhớ và địa chỉ cho các biến đó mà không cần biết các biến đó sử dụng lúc nào hoặc thậm chí có được sử dụng hay không. Các biến tĩnh sẽ tồn tại trong suốt thời gian thực hiện chương trình, vì vậy nếu ta chạy một chương trình lớn trong khi máy của ta lại hạn chế bộ nhớ thì sẽ xảy ra tình trạng không đủ bộ nhớ.

Nhưng khi dùng mảng ta phải khai báo kích thước mảng. Vì vậy đối với những chương trình mà ta không dự đoán trước được kích thước của chúng ra sao thì sẽ xảy ra tình trạng:

- Cấp dư, gây lãng phí bộ nhớ.
- Cấp thiếu, chạy không được.

➔ Đó là những nhược điểm của biến tĩnh.

Vậy *"Biến tĩnh là biến có kích thước, kiểu của biến và địa chỉ của chúng là không đổi, các biến này tồn tại trong suốt quá trình chạy chương trình"*

Để khắc phục những nhược điểm trên, các ngôn ngữ lập trình thường sử dụng những biến động vì có những đặc điểm sau:

- Không sinh ra lúc bắt đầu chương trình mà sinh ra trong quá trình thực hiện chương trình.

- Có thể thay đổi được kích thước, vùng nhớ và địa chỉ của vùng nhớ được cấp phát lúc chạy chương trình.
- Có thể giải phóng biến động sau khi đã sử dụng để tiết kiệm bộ nhớ.

Thế nhưng biến động cũng có nhược điểm là không thể truy nhập đến nó được bởi vì biến động không chứa địa chỉ nhất định

Để khắc phục nhược điểm này người ta sử dụng một loại biến đặc biệt gọi là biến con trỏ (pointer)

Biến con trỏ có những đặc điểm sau:

- Không chứa dữ liệu nhưng chứa địa chỉ của dữ liệu tức địa chỉ của biến khác, thông thường là các biến động.
- Kích thước của biến con trỏ không phụ thuộc vào đối tượng mà nó trỏ tới là kiểu gì. Kích thước cố định của biến con trỏ là 2 byte dùng để lưu địa chỉ của biến. Khi nó đang lưu địa chỉ của biến nào, ta nói nó đang trỏ tới biến ấy. Vậy : *"biến con trỏ là loại biến chuyên dùng để chứa địa chỉ của biến động, giúp ta truy nhập đến biến động"*

2. Khai báo biến con trỏ

Con trỏ là một biến dùng để chứa địa chỉ, vì có nhiều loại địa chỉ nên cũng có nhiều kiểu con trỏ tương ứng:

- Con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu nguyên.
- Con trỏ kiểu float, double dùng để chứa địa chỉ của các biến kiểu float, double.

Khai báo : ***Kiểu * Tên con trỏ;***

VD : int x, y, *px, *c; → Khai báo 2 biến kiểu int, 2 biến con trỏ kiểu int là px và c.

3. Quy định vùng trỏ tới

Tên con trỏ = & biến

Ví dụ:

- Khai báo các biến con trỏ:

…

```
int x, y, *px, *c; float
*t, *d;
```

- Quy định vùng trỏ tới: c= &y; (1) px = &x; (2) t = &y; (3)
- Câu lệnh (1): Gán địa chỉ của y cho con trỏ c.
- Câu lệnh (2) : Gán địa chỉ của biến x cho con trỏ px. Như vậy trong con trỏ c chứa địa chỉ của biến y và trong con trỏ px chứa địa chỉ của biến x.
- Câu lệnh (3): Vì t là con trỏ kiểu float nó chỉ chứa được địa chỉ của biến kiểu float, trong khi đó y lại là biến kiểu int nên không thể chấp nhận được. Hiệu ứng của các câu lệnh xác định (1) và (2) kể trên là nội dung của biến con trỏ sẽ là nội dung của biến mà nó trỏ tới(tức biến động đang chứa dữ liệu). lVD: Ở câu lệnh (2) : *px = 567 sẽ có tác dụng giống với câu lệnh : x = 567;

4. Kiểu giá trị trong khai báo

Nếu kiểu của con trỏ và kiểu của biến mà nó trỏ tới không cùng một kiểu sẽ gây ra lỗi.

VD : float a, b[7], f(), *px; → mọi thành phần trong khai báo này đều cho hoặc nhận giá trị kiểu float.

"Mọi thành phần của cùng một khai báo (biến, phần tử mảng, con trỏ) khi xuất hiện trong biểu thức đều cho cùng một kiểu giá trị"

5. Sử dụng con trỏ trong các biểu thức

Sau khi khai báo biến con trỏ, bạn có thể sử dụng các biến này trong các biểu thức. Ví dụ đối với con trỏ px kể trên ta có thể sử dụng cách viết các toán hạng trong biểu thức:

- px : Theo tên con trỏ.
- *px: Theo dạng khai báo của con trỏ.

a. Sử dụng tên con trỏ và phép gán.

- Vì con trỏ cũng là một biến nên khi tên của nó xuất hiện trong biểu thức thì giá trị của nó cũng được sử dụng trong biểu thức này. Giá trị của nó có nghĩa là địa chỉ của biến nào đó (biến động).

- Khi tên con trỏ ở bên trái của toán tử gán thì giá trị của biểu thức ở bên phải được gán cho con trỏ. VD : int a, *p, *a; (1) p = &a; (2) q = p; (3)

Câu lệnh (1) : khai báo biến a kiểu int và 2 biến con trỏ p và q cũng thuộc kiểu int.

Câu lệnh (2) : gán địa chỉ của biến a cho con trỏ p Câu

lệnh (3) : gán giá trị của p cho q.

Kết quả con trỏ q cũng chứa địa chỉ của biến a.

- Cũng giống như các biến khác, nội dung của biến con trỏ cũng có thể thay đổi. Ví dụ nếu con trỏ p chứa địa chỉ của phần tử mảng a[i] thì sau khi thực hiện phép toán

++p hoặc p++ nó sẽ chứa địa chỉ của phần tử a[i+1]

b. Sử dụng dạng khai báo của con trỏ.

Nếu con trỏ px trỏ tới biến x thì các cách viết x và *px là tương đương nhau. Sau khi khai báo float x, y, z, *px, *py thì các câu lệnh sau đây đều có tác dụng như nhau:

```
y = 3*x + z;
*px = 3*x + z;
*py = 3*(*px) + z;
```

Như vậy khi đã biết địa chỉ của một biến thì không những sử dụng giá trị của nó mà còn có thể gán cho nó một giá trị mới để thay đổi nội dung của biến.

6. Hàm có tham số con trỏ.

Ở mục này chúng ta sẽ nghiên cứu các tham số thực là các biến tĩnh sẽ truyền cho các tham số hình thức của hàm là các biến con trỏ.

Đa số các trường hợp người ta sử dụng cách truyền theo giá trị. Còn nếu muốn truyền theo quy chiếu (theo biến) tức là truyền cả địa chỉ và nội dung biến thì phải sử dụng biến con trỏ. VD :

```
hoanvi(px, py)
    int *px, *py;
    {
```

```
            int tam;
        tam = *px;
        *px = *py;
                *py = tam;
          }
main()
          {
                int a,b;
                printf("\n nhap so thu nhat:"); scanf("%d",&a);
                printf("\n nhap so thu hai:");
                scanf("%d",&b); hoanvi(&a, &b);  printf("\n
                sau khi hoan vi:"); printf("\n so thu nhat =
                %d",a); printf("\n so thu hai= %d",b);
                getch();
}
```

Người ta chia tham số của hàm thành 2 loại:

- Các tham số dùng để chứa các giá trị do các biến (tham số thực) nhập vào và ta gọi các tham số này là các tham số vào.
- Các tham số dùng để chứa kết quả do xử lý, tính toán, ta gọi các tham số này là các tham số ra.

VD : cần lập hàm giải PT bậc 2 : $ax^2 + bx + c = 0$

- Các hệ số a, b , c là các tham số vào
- Các nghiệm x1, x2 là các tham số ra.

Để trả lời câu hỏi "Khi nào thì sử dụng tham số con trỏ". Câu trả lời là: "Chỉ sử dụng cho các tham số ra"

7. Cấp phát vùng nhớ, thay đổi kích thước và giải phóng vùng nhớ.

a. Cấp phát vùng nhớ để lưu dữ liệu.

Thông thường chúng ta chỉ quan tâm đến khai báo và cung cấp vùng nhớ để lưu địa chỉ của biến con trỏ mà không lưu ý đến việc cung cấp vùng nhớ để lưu trữ dữ liệu nên khi chạy chương trình thường mắc sai lầm.

Khi ta khai báo :int *px; chỉ có tác dụng cung cấp cho bản thân biến px một vùng nhớ là 2 byte. Khai báo này không hề cung cấp vùng nhớ để lưu trữ dữ liệu mà px trỏ tới.

Vì vậy khi sử dụng biến con trỏ, ta phải:

- Cung cấp vùng nhớ cho biến con trỏ
- Cung cấp vùng nhớ để lưu dữ liệu

Khi khai báo : int x, *px; có nghĩa là đã cung cấp vùng nhớ cho bản thân con trỏ px và cũng đã cung cấp vùng nhớ cho biến x.

Còn câu lệnh: px = &x; có nghĩa là chỉ định vùng nhớ mà px trỏ tới là vùng nhớ của x mà vùng nhớ này đã được cung cấp rồi bằng hiệu ứng int x;

Để tránh những sai lầm đáng tiếc, Turbo C có sẵn 2 câu lệnh malloc hoặc calloc định nghĩa trong alloc.h hay stdlib.h để cung cấp vùng nhớ trực tiếp cho biến con trỏ.

Cú pháp tổng quát là:

Biến con trỏ = malloc(sizeof(tên kiểu));
Biến con trỏ = calloc(n, sizeof(tên kiểu));

Sự khác biệt duy nhất giữa 2 lệnh ở chỗ calloc vùng nhớ bằng n lần số bytes xác định bởi sizeof(tên kiểu)

VD : các câu lệnh sau đây có giá trị tương đương:

px = malloc (sizeof(int));

px = calloc (1, sizeof(int)); px = calloc (2, sizeof(char)); malloc và calloc cho ra các con trỏ trỏ tới kí tự, cho nên nếu muốn cấp phát bộ nhớ cho các loại biến khác thì phải dùng phép biến đổi kiểu cưỡng bức.

b. Thay đổi kích thước vùng nhớ động

Sau khi đã cấp phát vùng nhớ để lưu dữ liệu bằng 1 trong 2 cách:

- Khi khai báo biến con trỏ như : int *px;
- Hoặc dùng lệnh malloc, calloc;

Ta có thể thay đổi kích thước vùng nhớ đã cấp phát bằng lệnh realloc định nghĩa trong alloc.h có cú pháp như sau:

void *realloc(void *block, size – t size)

Trong đó : *block : Khối bộ nhớ động

* size-t size: Dạng sử dụng kích thước cho từng đối tượng VD :

```
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int k; typedef char
ten[26]; ten *p[10];
main()
{
        for( k = 0; k<=10; k++)
        {
 p[k] = malloc(sizeof(ten));   printf("\n ten nguoi
thu %d:", k+1);   gets(*p[k]);
                p[k] = realloc(*p[k], strlen(*p[k] +1);
        }
clrscr; printf("\n danh sach 10 nguoi vua
nhap: \n") for(k=0; k<10; k++)
printf("\n nguoi thu %d ten la %s", k+1, *p[k]); getch();
}
```

c. Cho biết kích thước vùng nhớ còn lại -

Đối với kiểu bộ nhớ Tyni, Small, Medium:

unsigned coreleft(void)

- Đối với kiểu bộ nhớ Compact, Large, Huge:

unsigned long coreleft(void)

d. Giải phóng vùng nhớ động

void free(void *block)

Trong đó *block : Vùng nhớ cần giải phóng tức là tên của biến con trỏ đang chiếm giữ vùng nhớ.

8. Ưu điểm của biến con trỏ

- Linh động trong việc cung cấp vùng nhớ: Khi sử dụng biến con trỏ trong chương trình thì việc cấp phát vùng nhớ sẽ được thực hiện trong quá trình chạy chương trình, không phải cung cấp ngay từ đầu nên thời gian nạp chương trình sẽ nhanh chóng hơn. Mặt khác các biến sau khi không còn sử dụng ta có thể giải phóng vùng nhớ (xóa dữ liệu) để sử dụng phần bộ nhớ này vào việc khác nên tiết kiệm được bộ nhớ.
- Thay đổi cách thức truyền tham số: Việc truyền tham số thực cho các tham số hình thức ở trong các hàm có 2 cách: Truyền theo giá trị và truyền theo biến. sự khác nhau cơ bản của 2 cách truyền này là:
  - Đối với truyền theo trị : Sau khi thực hiện xong các lệnh trong hàm thì các tham số thực không bị thay đổi giá trị dù rằng trong hàm đó có thay đổi chúng hay không.
  - Đối với truyền theo biến : Sau khi thực hiện xong hàm thì các tham số thực bị thay đổi nếu như trong hàm có sự thay đổi chúng

  Theo quy ước mặc định là truyền theo trị, còn nếu muốn truyền theo biến thì phải dùng biến con trỏ.

  VD : Truyền tham số theo biến:

```
main()
        {
                int a,b;
                printf("\n nhap so thu nhat:"); scanf("%d",&a);
               printf("\n nhap so thu hai:");
               scanf("%d",&b); truyenbien(&a, &b);
               printf("\n sau khi hoan vi:"); printf("\n so thu
```

```
                nhat = %d",a); printf("\n so thu hai= %d",b);
                getch();
} truyenbien(int *x,int
*y)
        {
                int tam;
                tam = *x;
                *x = *y;
                *y = tam;
                printf("\n so thu nhat = %d",*x);
                printf("\n so thu hai= %d",*y);

        }
```

II. Con trỏ và mảng 1 chiều

1. Địa chỉ của các phần tử mảng

Giả sử ta khai báo : int a[10];

Nghĩa là ta đã khai báo mảng a là một mảng 1 chiều có 10 phần tử kiểu số nguyên.

Để lấy địa chỉ của một phần tử nào đó ta dùng lệnh: &a[i]

Với i là phần tử trong khoảng từ 0 đến 9

2. Địa chỉ của phần tử mảng đầu tiên

Với khai báo int a[10] máy sẽ bố trí cho mảng a một vùng nhớ liên tiếp 20 bytes cho 10 phần tử mảng kiểu nguyên (mỗi phần tử 2 bytes). Như phần trên thì địa chỉ của từng phần tử mảng được xác định bằng phép toán &a[i]. Vậy địa chỉ của phần tử mảng đầu tiên sẽ là &a[0]. Do đó trong Turbo C quy định:

&a[0] tương đương với a(tên mảng)

&a[i] tương đương với a + i a[i]

tương đương với *(a +i)

Như vậy tên mảng đồng nghĩa với phần tử đầu tiên của mảng, do đó có thể nói:

"Tên mảng là một hằng địa chỉ"

3. Con trỏ trỏ tới phần tử mảng

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.

4. Tham số thực và tham số hình thức trong mảng 1 chiều:

Giả sử tham số thực là tên mảng a kiểu int(hoặc float, double…) thì tham số hình thức px tương ứng phải là một con trỏ cùng kiểu int (hoặc float, double..)

Tham số hình thức có thể khai báo:

- Theo kiểu con trỏ:

```
int *px; float
*px; double
*px;
```

- Theo tên mảng:

int px[ ]; float

px[ ]; double

px[ ];

Khi hàm bắt đầu làm việc thì giá trị thực của a được truyền cho tham số hình thức px. Vì a là hằng địa chỉ xác định địa chỉ của phần tử đầu tiên của mảng nên con trỏ px sẽ chứa địa chỉ của phần tử này (tức là phần tử đầu tiên của mảng). Sau đó nếu muốn trỏ tới phần tử a[i] ta có thể sử dụng 1 trong 2 dạng sau trong thân hàm:

*(px +i) và px[i]

VD : Tính tổng các phần tử của mảng a bằng cách truyền tham số thực là tên mảng cho các tham số hình thức trong hàm tong()

```
#include <stdio.h> main()
{ int a[10],i, tong();
        clrscr; for
        (i=0;i<4;++i)
        { printf("\n a[%d]=",i);
                scanf("%d",&a[i]);
        }
        printf("\n Tong cac phan tu mang la :%d ",tong(a,10))
        getch();
} int
tong(a,n) int
n, *a;
{ int i, s=0;
        for (i=0;i<n;++i)
s +=a[i];          return(s);
}
```

Ta có thể viết hàm tính tổng theo nhiều các khác nhau:

Cách 1:

```
int tong(a,n)
int n, *a;
{
        int i, s=0; for
        (i=0;i<n;++i) s
        +=*(a+i);

        return(s);
}
```

Cách 2:

```
int tong(a,n)
int n, *a;
{
        int i, s=0;
        While(n--) s
        +=*a++);

        return(s);
}
```

Cách 3:

```
int tong(a,n)
int n, a[];
{
        int i, s=0; for
        (i=0;i<n;++i) s
        +=*(a+i);

        return(s);
}
```

Cách 4:

```
int tong(a,n)
int n, a[];
{
        int i, s=0; for
        (i=0;i<n;++i) s
        +=a[i];

        return(s);
}
```

Cách 5:

```
int tong(a,n)
int n, a[];
{   int   i,   s=0;
        while(n--
        )
        s +=*a++;
        return(s);
}
```

5. Mảng và chuỗi kí tự

- Chuỗi kí tự là một dãy các kí tự kể cả kí tự trống được rào trong cặp dấu nháy kép (“”)
- Khi gặp một chuỗi kí tự máy sẽ cấp phát một vùng nhớ cho một mảng kiểu char để chứa các kí tự và chứa thêm kí tự “\0”(kí tự kết thúc một chuỗi). Mỗi kí tự trong chuỗi tương ứng với mỗi phần tử trong mảng. Vì vậy chuỗi kí tự nào cũng là một hằng địa chỉ biểu thị của phần tử đầu tiên. Do đó nếu ta khai báo biến ten như một con trỏ kiểu char thì sau đó có thể gán dữ liệu chuỗi cho biến này. Ví dụ:

```
char *ten;
ten = “Nguyen Van Xuan”;
```

- Sau đó nếu có in chuỗi này ra màn hình, ta có thể thực hiện 1 trong 2 câu lệnh sau:

printf(“Nguyen Van Xuan”); Hoặc

printf(ten);

Nếu muốn nhập một chuỗi (chẳng hạn tên của một người) ta có thể thực hiện bằng 1 trong 2 cách:

- Nếu dùng mảng: char t[24];

  printf(“\n cho biet

  ten:”); scanf(“%s”,t);

  …..

- Nếu dùng con trỏ:

  char *ten, t[24]; ten

  = t;

  printf(“\n Cho biết tên:”);

  scanf(“%s”,ten); hoặc scanf(“%s”,t);

  ……..

III. Con trỏ và mảng nhiều chiều

1. Nhập số liệu cho mảng nhiều chiều.

Mảng nhiều chiều phức tạp hơn mảng một chiều, ta không thể áp dụng các quy tắc của mảng 1 chiều cho mảng nhiều chiều được.

- Ta không thể sử dụng phép toán lấy địa chỉ để nhập giá trị cho từng phần tử mảng.
- Ta không thể sử dụng địa chỉ của phần tử mảng để nhập dữ liệu cho từng phần tử mảng được.
- Ta không thể sử dụng biến con trỏ để nhập dữ liệu cho mảng hai chiều được.
- Tóm lại cả 3 cách nhập số liệu cho mảng 1 chiều đều không thể áp dụng cho mảng nhiều chiều. Vậy muốn nhập dữ liệu cho mảng hai chiều ta thực hiện theo nguyên tắc sau:

- Nhập dữ liệu và gán cho biến trung gian (x): scanf(“%f”, &x);

- Gán biến trung gian cho phần tử mảng a[i][j]: a[i][j] = x;

VD : *#include "stdio.h" main()*

{ float a[2][3], x; int

i,j;

for (i=0;i<2;++i)

for (j=0;j<2;++j)

{ printf("\n

a[%d][%d]=",i,j);

scanf("%8.2f",&x);

a[i][j]=x;

}

}

2. Phép cộng địa chỉ trong mảng hai chiều:

Giả sử ta có mảng hai chiều a[2][3] có 6 phần tử úng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự sau :

Phần tử	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
Địa chỉ	1	2	3	4	5	6

Tên mảng a biểu thị địa chỉ đầu tiên của mảng. Phép cộng địa chỉ ở đây được thực hiện như sau : C coi mảng hai chiều là mảng (một chiều) của mảng, như vậy khai báo float a[2][3]; thì a là mảng mà mỗi phần tử của nó là một dãy 3 số thực (một hàng của mảng).

Vì vậy :

a trỏ phần tử thứ nhất của mảng : phần tử a[0][0] a+1 trỏ

phần tử đầu hàng thứ hai của mảng : phần tử a[1][0]

.....

3. Con trỏ và mảng hai chiều :

Để lần lượt duyệt trên các phần tử của mảng hai chiều ta có thể dùng con trỏ như minh họa ở ví dụ sau :

*float *pa,a[2][3]; pa=(float*)a;*

lúc đó :

pa	trỏ tới	a[0][0]
pa+1	trỏ tới	a[0][1]
pa+2	trỏ tới	a[0][2]
pa+3	trỏ tới	a[1][0]
pa+4	trỏ tới	a[1][1]
pa+5	trỏ tới	a[1][2]

Ví dụ :

Dùng con trỏ để vào số liệu cho mảng hai chiều.

Cách 1 :

```
#include "stdio.h" main()
{ float a[2][3],*pa;
        int i;
        pa=(float*)a; for
        (i=0;i<6;++i)
        scanf("%f",pa+i)
        ;
 }
```

Cách 2 :

```
#include "stdio.h" main()
{ float a[2][3],*pa;
        int i;
        for (i=0;i<6;++i)
        scanf("%f",(float*)a+i); }
```