ĐẠI HỌC QUỐC GIA HÀ NỘI
KHOA CÔNG NGHỆ

GIÁO TRÌNH TIN HỌC CƠ SỞ
PHẦN LẬP TRÌNH TRÊN NGÔN NGỮ C

VŨ BÁ DUY

HÀ NỘI THÁNG 1 NĂM 2003

Đây là phiên bản đầu tiên, có thể còn nhiều sai sót. Chung tôi mong nhận được ý kiến đóng góp của bạn đọc. Các ý kiến gửi về theo địa chỉ dkquoc@vnu.edu.vn hoặc dkquoc@yahoo.com .

Cảm ơn bạn đọc đóng góp để hoàn thiện giáo trình.

Thay mặt các tác giả

Đào Kiến Quốc

MÔC LÔC

I. Mở đầu

C là ngôn ngữ lập trình được thiết kế bởi Dennis Ritchie tại phòng thí nghiệm Bell Telephone năm 1972. Nó được viết với mục tiêu chính là xây dựng hệ điều hành UNIX. Vì thế ban đầu nó không hướng tới sự tiện dụng cho người lập trình. C được phát triển từ một ngôn ngữ lập trình có tên là B (B là ngôn ngữ lập trình được viết bởi Ken Thompson tại Bell Labs, và tên ngôn ngữ lấy theo tên của Bell Labs).

C là ngôn ngữ mạnh và mềm dẻo, linh hoạt, nó nhanh chóng trở thành ngôn ngữ phổ biến không chỉ trong phạm vi của Bell, C được các lập trình viên sử dụng viết nhiều loại ứng dụng ở các mức độ khác nhau.

Cũng vì nó được dùng nhiều nơi nên xuất hiện những đặc điểm khác nhau, các phiên bản phát triển không thống nhất. Để giải quyết vấn đề này, năm 1983 Viện tiêu chuẩn Mỹ (ANSI) đã thành lập một chuẩn cho C và có tên ANSI C (ANSI standard C). Nói chung các chương trình dịch C ngày nay đều tuân theo chuẩn này ngoại trừ một số khác biệt nhỏ.

Hiện nay có rất nhiều ngôn ngữ lập trình bậc cao như C, Pascal, BASIC,.. mỗi ngôn ngữ đều có điểm mạnh riêng của nó và phù hợp cho một số lĩnh vực nào đó, C cũng không ngoại lệ, C được phổ biến bởi nó có các đặc điểm sau:

- C là ngôn ngữ mạnh và mềm dẻo. Có thể nói rằng sự hạn chế của C chỉ phụ thuộc vào người lập trình, tức là với C bạn có thể làm tất cả những điều theo ý tưởng của bạn. C được dùng cho những dự án từ nhỏ tới lớn như: Hệ điều hành, Đồ hoạ, Chương trình dịch,...
- C dễ chuyển đổi sang hệ hệ thống khác (tính khả chuyển), tức là một chương trình C được viết trên hệ thống này có thể dễ dàng dịch lại chạy được trên hệ thống khác
- C là ngôn ngữ cô đọng, số lượng từ khoá không nhiều.
- C là ngôn ngữ lập trình cấu trúc. Mã lệnh của chương trình C được viết thành các hàm, các hàm này có thể sử dụng lại trong các ứng dụng khác.

Với các đặc điểm trên C là ngôn ngữ tốt cho việc học lập trình, hơn nữa sau này chúng ta còn có thể tiếp cận với lập trình hướng đối tượng, và một trong những ngôn ngữ lập trình chúng ta lựa chọn đầu tiên cho lập trình hướng đối tượng là C++, những kiến thức về C vẫn có ích cho bạn vì C++ là ngôn ngữ được phát triển từ C và bổ sung đặc tính hướng đối tượng.

I.1. Bảng chữ cái, tên và từ khoá

- ***Bảng chữ cái***: Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ kí tự nào đó và các quy tắc trên đó để xây dựng các từ, các câu lệnh và cấu trúc chương trình. Ngôn ngữ lập trình C sử dụng bộ ký tự ASCII (American Standard Code for Informations Interchange). Theo chuẩn này, bộ kí tự gồm có 256 kí tự đó là:

- Các chữ cái: A,..,Z, a,..,z
- Các chữ số: 0,..,9
- Các dấu phép toán số học: +,-,*,/,...
- Các dấu ngoặc: (,), [,],...
- Các ký tự khác

Mỗi kí tự có tương ứng 1 số duy nhất gọi là mã, trong đó có 128 kí tự đầu (có mã từ 0 tới 127) là kí tự cố định và 128 kí tự còn lại (có mã từ 128 tới 255) là các kí tự mở rộng, tức là nó có thể thay đổi tuỳ theo ngôn ngữ mỗi quốc gia sử dụng.

- ***Từ khoá và tên***: Tên là một xâu (dãy) các kí tự, trong ngôn ngữ lập trình nói chung đều yêu cầu tên phải tuân theo những ràng buộc nhất định.

Với C tên là xâu kí tự chỉ có thể gồm

- các chữ cái
- chữ số
- dấu gạch nối

Tên phải bắt đầu bằng chữ cái hoặc dấu gạch dưới, độ dài không quá 32 kí tự, không được trùng với từ khoá của ngôn ngữ. Và vì C phân biệt chữ hoa và chữ thường nên các tên chữ hoa như XY và xy là khác nhau.

Mỗi ngôn ngữ đều có riêng một tập các từ với ý nghĩa đặc biệt đó là các từ khoá, chúng được dùng với mục đích định trước như tên kiểu dữ liệu, tên toán tử,..

Sau đây là một số từ khoá của C

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

I.2.Các bước lập trình giải bài toán

Để giải một bài dù mức nào thì bước đầu tiên chúng ta cũng phải phát biểu bài toán, tức là chúng ta phải hiểu bài toán yêu cầu gì thì mới có thể tìm được thuật giải, và cài đặt thuật toán đó và sau khi đã có chương trình bạn phải chạy để kiểm nghiệm tính đúng đắn của nó.

Như vậy để giải bài toán bằng chương trình chúng ta theo các bước sau:

1. Xác định đối tượng của chương trình

2. Xác định phương pháp và thuật giải

3. Viết chương trình (lập trình)

4. Chạy chương trình và kiểm tra kết quả.

Để có một chương trình chúng ta cần phải viết các lệnh (lập trình) trong một ngôn ngữ lập trình nào đó, như C chẳng hạn, nhưng máy tính không chạy trực tiếp được chương trình viết bằng các ngôn ngữ lập trình bậc cao (gọi là chương trình nguồn), nó chỉ có thể thực hiện được các chương trình dạng mã máy (chương trình đích). Vì vậy sau khi đã có chương trình nguồn, chúng ta cần thực hiện chuyển chương trình nguồn thành chương trình đích, công việc này chúng ta cần đến trình biên dịch (compiler) và liên kết (linker). Như vậy ta thấy chu trình phát triển một chương trình như sau:

1. Soạn thảo chương trình nguồn

Chúng ta có thể sử dụng một trình soạn thảo văn bản chuẩn (ASCII) nào đó để soạn thảo chương trình, sau đó ghi vào file chương trình nguồn (ngầm định với phần mở rộng là .C).

Do C cũng như hầu hết các ngôn ngữ lập trình phổ biến đều sử dụng bảng chữ cái ASCII nên bạn có thể sử dụng bất kỳ một hệ soạn thảo văn bản chuẩn để viết chương trình, tuy nhiên hầu hết các trình biên dịch của C trên môi trường MS-DOS hoặc WINDOWS đều có tích hợp trình soạn thảo và bạn nên sử dụng trình soạn thảo tích hợp này sẽ thuận lợi hơn.

2. Biên dịch chương trình nguồn

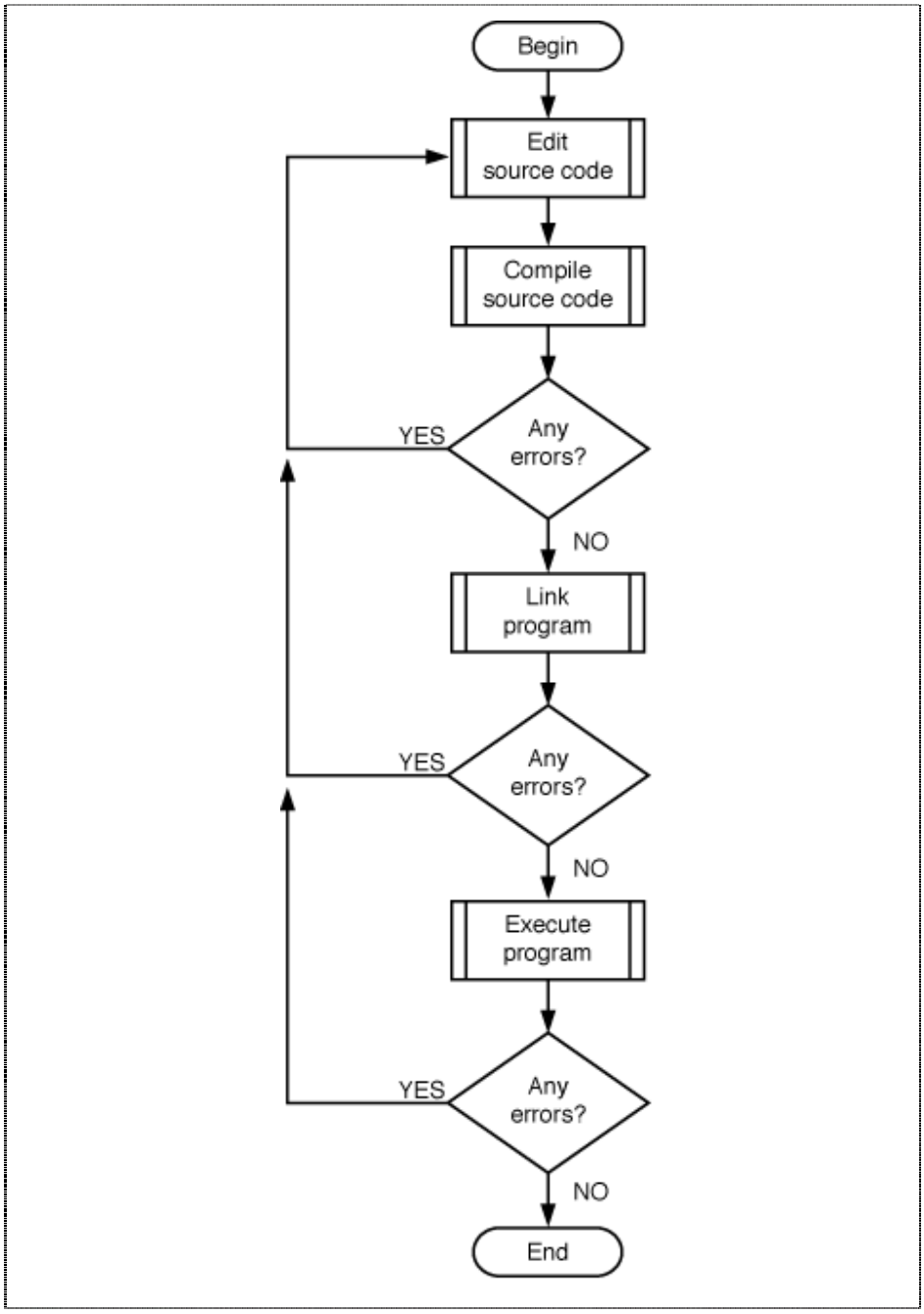
Hiện nay có rất nhiều chương trình dịch cho C như: Turbo C, BC, Microsoft C,.. mục đích của bước này là chuyển chương trình nguồn thành chương trình mã đối tượng (object). Sau bước này (nếu thành công) chúng ta thu được file chương trình đối tượng (có phần mở rộng là .OBJ)

3. ***Liên kết chương trình***

Sau bước biên dịch hoàn thành ta có chương trình đối tượng, đây chưa phải là chương trình có thể chạy được trên máy tính, bước này chúng ta phải sử dụng một trình liên kết để liên kết các hàm thư viện với chương trình đối tượng để tạo ra chương trình đích . Bạn có thể sử dụng trình liên kết độc lập nào đó, nhưng với các trình biên dịch của C trên môi trường DOS hay WINDOWS đều có sẵn trình liên kết.

4. ***Chạy và kiểm tra kết quả chương trình***

Khi đã có chương trình đích, chúng ta cần phải kiểm tra tính đúng đắn của nó. bạn chạy chương trình với các bộ dữ liệu mẫu và kiểm tra kết quả có như dự kiến hay không, nếu có sai sót thì phải xác định nguyên nhân gây lỗi và quay lại bước 1 để hiệu chỉnh. và chúng ta lặp lại quá trình này cho tới khi được chương trình giải đúng bài toán mong đợi.



Hình 1 – Các bước phát triển chương trình

Hiện nay có rất nhiều chương trình dịch cho C và hầu hết (trên nền DOS hoặc Windows) trong đó được tích hợp cả trình soạn thảo, biên dịch, liên kết - gọi là môi trường tích hợp. Trong giáo trình này chúng ta sử dụng BC (Borland C) hoặc turbo C làm môi trường lập trình.

II. Biến, hằng và các kiểu dữ liệu trong C

II.1.Biến

- **Khái niệm**

Biến là đại lượng có giá trị thuộc một kiểu dữ liệu nào đó mà được chấp nhận bởi ngôn ngữ (*xem phần các kiểu dữ liệu*), giá trị của biến có thể thay đổi trong thời gian tồn tại của biến (hay ta nói trong vòng đời của biến).

Các thành phần của chương trình sẽ được lưu trong bộ nhớ trong và biến cũng không ngoại lệ. Tức là biến cũng được cấp phát một vùng nhớ để lưu giữ giá trị thuộc một kiểu dữ liệu xác định. Vì thế theo một khía cạnh nào đó có thể nói biến là một cái tên đại diện cho ô nhớ trong máy tính, chương trình có thể truy xuất ô nhớ (lấy hoặc ghi giá trị) thông qua tên biến.

Một biến nói chung phải có các đặc trưng sau:

- Tên biến

- Kiểu dữ liệu: kiểu của biến

- Giá trị hiện tại nó đang lưu giữ (giá trị của biến)

(tuy nhiên sau này chúng ta thấy trong C có biến kiểu void, ban đầu coi đây là biến không kiểu nhưng dần quan niệm đó cũng là 1 tên kiểu và là kiểu không xác định)

- **Tên biến**

Trong C cũng như các ngôn ngữ lập trình khác các biến đều phải có tên, các tên biến hay nói chung là tên (gồm tên biến, tên hằng, tên hàm, hoặc từ khoá) là một xâu kí tự và phải tuân theo các quy định của ngôn ngữ đó là:

- Tên chỉ có thể chứa kí tự là chữ cái (‘a’ ,..,’z’; ‘A’,..,’Z’); chữ số(‘0’,..,’9’) và kí tự gạch dưới (_), số kí tự không quá 32.
- Kí tự đầu tiên của tên phải là chữ cái hoặc kí tự gạch dưới
- Trong tên phân biệt chữ hoa và chữ thường. Tức là hai xâu cùng các kí tự nhưng khác nhau bởi loại chữ hoa hoặc chữ thường là hai tên khác nhau, ví dụ như với 2 xâu kí tự “AB” và “Ab” là hai tên hoàn toàn phân biệt nhau.
- Các từ khoá của ngôn ngữ không được dùng làm tên biến, tên hằng, hay tên hàm. Hay nói khác đi, trong chương trình có thể bạn phải dùng đến tên, tên này do bạn đặt theo ý tưởng của bạn nhưng không được trùng với các từ khoá.

- **Ví dụ các tên hợp lệ và không hợp lệ**

Tên biến	*hợp lệ / không hợp lệ*
Percent	hợp lệ

y2x5__fg7h	hợp lệ
ho_ten	hợp lệ
_1990_tax	hợp lệ
A	hợp lệ
ngay-sinh	không hợp lệ vì có kí tự -(dấu trừ)
double	không hợp lệ vì trùng với từ khoá
9winter	không hợp lệ vì kí tự đầu tiên là số

- **Câu lệnh định nghĩa biến**

Trong ngôn ngữ lập trình có cấu trúc nói chung và trong C nói riêng, mọi biến đều phải được định nghĩa trước khi sử dụng. Câu lệnh định nghĩa biến báo cho chương trình dịch biết các thông tin tên, kiểu dữ liệu và có thể cả giá trị khởi đầu của biến.

Cú pháp khai báo biến :

<kiểu_dữ_liệu> <biến_1> [= <giá_trị_1>] [, <biến_2>[= <giá_trị_2>,..];

trong đó:

- <kiểu_dữ_liệu> là tên một kiểu dữ liệu đã tồn tại, đó có thể là tên kiểu dữ liệu chuẩn hoặc kiểu dữ liệu định nghĩa bởi người lập trình.
- <biến_1>, <biến_2> là các tên biến cần khai báo, các tên này phải tuân theo quy tắc về tên của ngôn ngữ.
- <giá_tri_1>, <giá_trị_2> là các giá trị khởi đầu cho các biến tương ứng <biến_1>, <biến_2>. Các thành phần này là tuỳ chọn, nếu có thì giá trị này phải phù hợp với kiểu của biến.

Trên một dòng lệnh định nghĩa có thể khai báo nhiều biến cùng kiểu, với tên là <biến_1>, <biến_2>,.. các biến cách nhau bởi dấu phẩy (,) dòng khai báo kết thúc bằng dấu chấm phẩy (;).

Ví dụ:

```
int a  = 4, b = 6;
float x =4.5,y,z;
unsigned u ;
char c =’A’;
```

Khi gặp các lệnh định nghĩa biến, chương trình dịch sẽ cấp phát vùng nhớ có kích thước phù hợp với kiểu dữ liệu của biến, nếu có thành phần khởi đầu thì sẽ gán giá trị khởi đầu vào vùng nhớ đó.

II.2. Hằng

- **Khái niệm**

Hằng là đại lượng có giá trị thuộc một kiểu dữ liệu nhất định, nhưng giá trị của hằng không thể thay đổi trong thời gian tồn tại của nó.

Có hai loại hằng một là các hằng không có tên (chúng ta sẽ gọi là hằng thường) đó là các giá trị cụ thể tức thời như : 8, hay 9.5 hoặc ‘d’.

Loại thứ hai là các hằng có tên (gọi là hằng ký hiệu). Các hằng ký hiệu cũng phải định nghĩa trước khi sử dụng, tên của hằng được đặt theo quy tắc của tên. Sau đây nếu không có điều gì đặc biệt thì chúng ta gọi chung là hằng

- **Định nghĩa hằng**

Các hằng được định nghĩa bằng từ khoá **const** với cú pháp như sau:

const <kiểu_dữ_liệu> <tên_hằng> = <giá_trị>;

hoặc **const <tên_hằng> = <giá_trị>;**

Trong dạng thứ hai, chương trình dịch tự động ấn định kiểu của hằng là kiểu ngầm định, với BC hay TC là int và như vậy chương trình dịch sẽ tự động chuyển kiểu của <giá_trị> về kiểu int.

Ví dụ:

```
const int a = 5; // định nghĩa hằng a kiểu nguyên, có giá trị là 5
const float x = 4; // hằng x kiểu thực, có giá trị là 4.0
const d = 7; // hằng d kiểu int, giá trị là 7
const c = ‘1’; // hằng c kiểu int giá trị = 49
const char * s  = “Ngon ngu C”;// s là hằng con trỏ, trỏ tới xâu “Ngo ngu C”
```

Các hằng số trong C được ngầm hiểu là hệ 10, nhưng bạn có thể viết các hằng trong hệ 16 hoặc 8 bằng cú pháp, giá trị số hệ 16 được bắt đầu bằng 0x, ví dụ như 0x24, 0xA1 các số hệ 8 bắt đầu bởi số 0, ví dụ 025, 057.

Các hằng kí tự được viết trong cặp dấu ‘’ ví dụ ‘a’, ‘2’ các giá trị này được C hiểu là số nguyên có giá trị bằng mã của kí tự; ‘a’ có giá trị là 97, ‘B’ có giá trị bằng 66.

Các xâu kí tự là dãy các kí tự được viết trong cặp “”, ví dụ “Ngon ngu C”, “a” (*xâu kí tự sẽ được giới thiệu trong phần sau*)

***Chú ý**: Các biến, hằng có thể được định nghĩa ngoài mọi hàm, trong hàm hoặc trong một khối lệnh. Với C chuẩn thì khi định nghĩa biến, hằng trong một khối thì dòng định nghĩa phải ở các dòng đầu tiên của khối, tức là trước tất cả các lệnh khác của khối, nhưng trong C++ bạn có thể đặt dòng định nghĩa bất kỳ vị trí nào.*

II.3. Các kiểu dữ liệu chuẩn đơn giản trong C

Một trong mục đích của các chương trình là xử lý, biến đổi thông tin, các thông tin cần xử lý phải được biểu diễn theo một cấu trúc xác định nào đó ta gọi là các kiểu dữ liệu. Các kiểu dữ liệu này được quy định bởi ngôn ngữ lập trình, hay nói khác đi mỗi ngôn ngữ có tập các kiểu dữ liệu khác nhau. Không hoàn toàn giống như khái niệm kiểu dữ liệu trong toán học, trong các ngôn ngữ lập trình nói chung mỗi kiểu dữ liệu chỉ biểu diễn được một miền giá xác định nào đó. Chẳng hạn như số nguyên chúng ta hiểu là các số nguyên từ - ∞ tới +∞, nhưng trong ngôn ngữ lập trình miền các giá trị này bị giới hạn, sự giới hạn này phụ thuộc vào kích thước của vùng nhớ biểu diễn số đó. Vì vậy khi nói tới một kiểu dữ liệu chúng ta phải đề cập tới 3 thông tin đặc trưng của nó đó là:

- tên kiểu dữ liệu
- kích thước vùng nhớ biểu diễn nó,miền giá trị
- các phép toán có thể sử dụng.

Các kiểu dữ liệu đơn giản trong C chỉ là các kiểu số, thuộc hai nhóm chính đó là số nguyên và số thực (số dấu phẩy động).

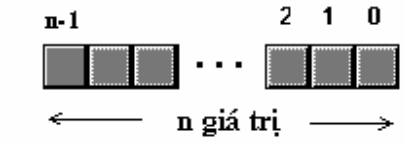
- *Nhóm các kiểu nguyên gồm có*: char, unsigned char, int, unsigned int, short, unsigned short, long, unsigned long được mô tả trong bảng sau:

Kiểu dữ liệu	tên kiểu (từ khoá tên kiểu)	kích thước	miền giá trị
kí tự có dấu	char	1 *byte*	từ -128 tới 127
kí tự không dấu	unsigned char	1 *byte*	từ 0 tới 255
số nguyên có dấu	int	2 *byte*	từ -32768 tới 32767
số nguyên không dấu	unsigned int	2 *byte*	từ 0 tới 65535
số nguyên ngắn có dấu	short	2 *byte*	từ -32768 tới 32767
số nguyên ngắn có dấu	unsigned short	2 *byte*	từ 0 tới 65535
số nguyên dài có dấu	long	4 *byte*	từ -2,147,483,648 tới 2,147,438,647
số nguyên dài không dấu	unsigned long	4 *byte*	từ 0 tới 4,294,967,295

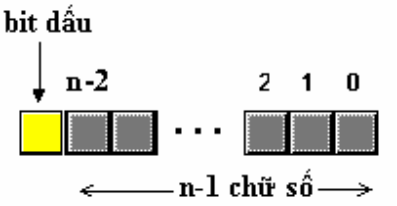
Khuôn dạng số nguyên: mặc dù như trên chúng ta có kiểu số nguyên và kí tự (char) nhưng bản chất trong C chúng đều là các số nguyên mà thôi. Hệ thống biểu diễn các số nguyên dưới dạng dãy các bit (số nhị phân). Như chúng ta đã biết, một bit chỉ có thể biểu diễn được 2 giá trị là 0 và 1.

Ta thấy với một nhóm có 2 bit (2 số nhị phân) thì có thể lưu được giá trị nhỏ nhất khi cả 2 bit đều bằng 0 và lớn nhất khi cả 2 bit bằng 1 có nghĩa là nó có thể biểu diễn được các số 0,1,2,3 tức 2^2 giá trị khác nhau. Với số nguyên 1 byte (unsigned char) thì giá trị nó có thể lưu trữ là 0,1,..,255.

Tổng quát nếu kiểu dữ liệu có kích thước n bit thì có thể biểu diễn 2^n giá trị khác nhau là: $0,1,..(2^n-1)$.



Nhưng đó là trong trường hợp tất cả các bit dùng để biểu diễn giá trị số(các con số), tức là ta có số nguyên không dấu (***số dương – unsigned***). Nhưng số nguyên chúng ta cần có thể là số âm (***số có dấu – signed***), trong trường hợp này bit cao nhất được dùng biểu diễn dấu, như vậy chỉ còn n-1 bit để biểu diễn giá trị. Nếu số âm (có dấu) thì bit dấu có giá trị =1, ngược lại, nếu số có giá trị dương thì bit dấu có giá trị =0.



Ví dụ với kiểu char (signed char) một byte thì có 7 bit để biểu diễn các con số, vậy nó có thể biểu diễn các số dương 0,1,..,127 và (theo cách biểu diễn số âm – xem phần hệ đếm và biểu diễn số âm) nó biểu diễn được các số âm –1,..-128. Miền giá trị của các kiểu số nguyên khác được diễn giải tượng tự.

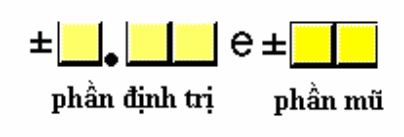
Các bạn có thể đặt câu hỏi tại sao đã có kiểu int lại vẫn có kiểu short hay có sự khác nhau giữa int và short hay không?. Thực ra sự khác nhau giữa chúng phụ thuộc vào hệ thống mà bạn dùng. Trên môi trường 32 bit thì int có kích thước là 4 byte, short có kích thước 2 byte, còn trên môi trường 16 bit thì chúng giống nhau.

Thực ra sự quy định kích thước của các kiểu nguyên chỉ là:

- *kiểu char kích thước là 1 byte*
- *kiểu short kích thước là 2 byte*
- *kiểu long kích thước là 4 byte*
- *kích thước kiểu short <= kích thước kiểu int <= kích thước kiểu long*

➢ *Nhóm các kiểu số thực gồm: float, double, long double*

Khuôn dạng biểu diễn của số thực không giống như số nguyên. Một số thực nói chung được biểu diễn theo ký pháp khoa học gồm phần định trị và phần mũ.



Trong giáo trình này chúng tôi không có ý định trình bày chi tiết định dạng của số thực. Bạn đọc cần quan tâm tới vấn đề này hãy tham khảo [3 - Chương 14]. Chính vì

khuôn dạng khác mà miền giá trị của số thực so với số nguyên có cùng kích thước cũng khác.

Kiểu dữ liệu	**tên kiểu**	**kích thước**	**(trị tuyệt đối)miền giá trị**
số thực với độ chính xác đơn	float	4 *byte*	3.4e-38 -> 3.4e38
số thực với độ chính xác kép	double	8 *byte*	1.7e-308 -> 1.7e308
số thực dài với độ chính xác kép	long double	10 *byte*	3.4e-4832 -> 1.1e 4932

Trong bảng trên miền giái trị chúng ta nói tới giá trị dương lớn nhất mà số thực có thể biểu diễn (giá trị âm nhỏ nhất lấy đối) và giá trị dương nhỏ nhất còn phân biệt được với 0.

Ví dụ với kiểu float, giá trị dương lớn nhất là 3.4e38 =$3.4*10^{38}$ và số dương nhỏ nhất có thể biểu diễn là 3.4e-38 = $3.4*10^{-38}$.

Tuy nhiên, do số chữ số trong phần định trị là giới hạn nên số chữ số đáng tin cậy (hay ta nói là số chữ số có nghĩa) cũng giới hạn với kiểu float là 7-8 chữ số, double là 15 chữ số, và long double là 18-19 chữ số.

- **Kiểu con trỏ và địa chỉ**

Ngoài hai kiểu dữ liệu số mà chúng ta vừa đề cập trong C còn kiểu dữ liệu rất hay sử dụng đó là kiểu con trỏ. Chúng ta biết là các thành phần: biến, hằng, hàm,.. được lưu trong bộ nhớ, tức là chúng được định vị tại một vùng nhớ có được xác định. Một thành phần (biến, hằng) có thể lưu giá trị là địa chỉ của một thành phần khác được gọi là con trỏ.

Giá sử p là một con trỏ lưu địa chỉ của a thì ta nói p trỏ tới a và kiểu của con trỏ p là kiểu của thành phần mà p trỏ tới.

Khai báo con trỏ

<kiểu> * <tên_con_trỏ>; // khai báo biến con trỏ

Ví dụ:

```
int * p,*q; // p, q là 2 con trỏ kiểu int
```

Kiểu void : Ngoài các kiểu dữ liệu trong C còn có những thành phần (con trỏ) không xác định kiểu, hoặc hàm không cần trả về giá trị trong trường hợp này chúng ta có con trỏ, hàm kiểu void. Hay nói các khác void là một kiểu nhưng là kiểu không xác định.

II.4. Biểu thức và các phép toán

- **Biểu thức**

Biểu thức là sự kết hợp giữa các toán hạng và toán tử theo một cách phù hợp để diễn đạt một công thức toán học nào đó. Các toán hạng có thể là hằng, biến, hay lời gọi hàm hay một biểu thức con. Các toán tử thuộc vào tập các toán tử mà ngôn ngữ hỗ trợ.

Biểu thức được phát biểu như sau:

- Các hằng, biến, lời gọi hàm là biểu thức
- Nếu A, B là biểu thức và ⊗ là một phép toán hai ngôi phù hợp giữa A và B thì A⊗B là biểu thức.
- Chỉ những thành phần xây dựng từ hai khả năng trên là biểu thức.

Một biểu thức phải có thể ước lượng được và trả về giá trị thuộc một kiểu dữ liệu cụ thể. Giá trị đó được gọi là giá trị của biểu thức và kiểu của giá trị trả về được gọi là kiểu của biểu thức, ví dụ một biểu thức sau khi ước lượng trả lại một số nguyên thì chúng ta nói biểu thức đó có kiểu nguyên (nói ngắn gọn là biểu thức nguyên).

Ví dụ : p = (a+b+c)/2;
s = sqrt((p-a)*(p-b)*p-c));
trong đó a, b, c là 3 biến số thực.

Biểu thức logic trong C: theo như trên chúng ta nói thì biểu thức logic là biểu thức mà trả về kết quả kiểu logic. Nhưng trong ngôn ngữ lập trình C không có kiểu dữ liệu này (như boolean trong Pascal). Trong C sử dụng các số để diễn đạt các giá trị logic (‘đúng’ hay ‘sai’). Một giá trị khác 0 nếu được dùng trong ngữ cảnh là giá trị logic sẽ được coi là ‘đúng’ và nếu giá trị bằng 0 được xem là sai. Ngược lại một giá trị ‘sai’(chẳng hạn như giá trị của biểu thức so sánh sai (5==3)) sẽ trả lại số nguyên có giá trị 0, và giá trị của biểu thức (ví dụ như 5 < 8) ‘đúng’ sẽ trả lại một số nguyên có giá trị 1. Sau này chúng ta còn thấy không phải chỉ có các số được dùng để diễn đạt giá trị ‘đúng’ hay ‘sai’ mà một con trỏ có giá trị khác NULL (rỗng) cũng được coi là ‘đúng’, và giá trị NULL được xem là ‘sai’.

➢ Các toán tử (phép toán) của ngôn ngữ C

a. Phép gán

Cú pháp

<biến> = <giá trị>

Trong đó vế trái là tên một biến và vế phải là một biểu thức có kiểu phù hợp với kiểu của biến. Với phép gán hệ thống sẽ ước lượng giá trị của vế phải sau đó gán giá trị vào biến bên trái.

Ví dụ:

```
int a, b;
a = 5;
b = a +15;
```

Sự phù hợp kiểu giữa vế bên phải và bên trái được hiểu là hoặc hai vế cùng kiểu hoặc kiểu của biểu thức bên phải có thể được chuyển tự động (ép kiểu) về kiểu của biến bên trái theo quy tắc chuyển kiểu tự động của ngôn ngữ C là từ thấp tới cao:

char → int → long → double.

Tuy nhiên trong thực tế sự ép kiểu phụ thuộc vào chương trình dịch, một số chương trình dịch cho phép tự chuyển các kiểu số bên phải về kiểu cúa vế trái bằng mà không cần phải tuân theo quy tắc trên, bằng cách cắt bỏ phần không phù hợp. Ví dụ bạn có thể gán bên phải là số thực (float) vào vế trái là một biến nguyên (int), trường hợp này chương trình dịch sẽ cắt bỏ phần thập phân và các byte cao, nhưng kết quả có thể không như bạn mong muốn.

Với C chúng ta có thể thực hiện gán một giá trị cho nhiều biến theo cú pháp:

<biến_1>=<biến_2> = ,..=<giá_trị>

với lệnh trên sẽ lần lượt gán <giá_trị> cho các biến từ phải qua trái.

b. Các phép toán số học

phép toán	cú pháp	ý nghĩa
+	<th_1> + <th_2>	phép cộng giữa <th_1> và <th_2>là số thực hoặc nguyên
-	<th_1> - <th_2>	phép trừ giữa <th_1> và <th_2>là số thực hoặc nguyên
*	<th_1> * <th_2>	phép nhân giữa <th_1> và <th_2>là số thực hoặc nguyên
/	<th_1> / <th_2>	phép chia lấy phần nguyên giữa <th_1> và <th_2>là số nguyên. ví dụ 9/2 kết quả là 4
/	<th_1> / <th_2>	phép chia giữa <th_1> và <th_2>là số thực ví dụ 9.0/2.0 kết quả là 4.5
%	<th_1> % <th_2>	phép chia lấy phần dư giữa <th_1> và <th_2>là số nguyên ví dụ 15 % 4 = 3; 12%3 =0

Trong các phép toán số học nói trên, khi hai toán hạng cùng kiểu thì kết quả là số có kiểu chung đó. Nếu hai toán hạng không cùng kiểu (trừ %) thì toán hạng có kiểu nhỏ hơn sẽ được tự động chuyển về kiểu của toán hạng còn lại, đây cũng là kiểu của kết quả.

c. Các phép toán so sánh (quan hệ)

phép toán	cú pháp	ý nghĩa
= =	th_1 == th_2	so sánh bằng, kết quả 'đúng' nếu 2 toán hạng bằng nhau, ngược lại trả lại 'sai'.
!=	th_1> != th_2	so sánh khác nhau, kết quả 'đúng' nếu 2 toán hạng khác nhau, ngược lại trả lại 'sai'.
>	th_1 > th_2	so sánh lớn hơn, kết quả 'đúng' nếu toán hạng thứ nhất lớn hơn, ngược lại trả lại 'sai'.
>=	th_1 >= th_2	so sánh lớn hơn hoặc bằng, kết quả 'đúng' nếu toán hạng thứ nhất lớn hơn hay bằng toán hạng thứ 2, ngược lại trả lại 'sai'.
<	th_1 < th_2	so sánh nhỏ hơn, ngược của >=
<=	th_1 <= th_2	so sánh nhỏ hơn hoặc bằng, ngược với >

Trong phần các kiểu dữ liệu chúng ta không có kiểu dữ liệu tương tự như boolean trong Pascal để biểu diễn các giá trị logic (true, false). Vậy kết quả các phép toán so sánh mà chúng ta thu được 'đúng', 'sai' là gì? Ngôn ngữ C dùng các số để biểu thị giá trị 'đúng' hay 'sai'. Một số có giá trị bằng 0 nếu dùng với ý nghĩa là giá trị logic thì được xem là 'sai' ngược lại nếu nó khác 0 được xem là 'đúng'. Thực sự thì các phép so sánh trên cũng đều trả về giá trị là số nguyên, nếu biểu thức so sánh là 'sai' sẽ có kết quả = 0, ngược lại nếu biểu thức so sánh là đúng ta thu được kết quả = 1.

Ví dụ:

5 > 2 trả lại giá trị = 1
5 <= 4 trả lại giá trị = 0
'a'!='b' trả lại giá trị = 1

d. Các phép toán logic

– **Phép toán ! (phủ định)**:

Cú pháp:

! <toán_hạng>

với <toán_hạng> là biểu thức số nguyên hoặc thực, nếu <toán_hạng> có giá trị khác 0 thì kết quả sẽ =0 và ngược lại, nếu <toán_hạng> ==0 thì kết quả sẽ = 1.

- **Phép toán && (phép hội - and)**:

  Cú pháp:

  <toán_hạng_1> && <toán_hạng_2>

trong đó 2 toán hạng là các biểu thức số, kết quả của phép toán này chỉ 'đúng' (!=0) khi và chỉ khi cả 2 toán hạng đều có giá trị 'đúng' (!=0).

<toán_hạng_1>	**<toán_hạng_2>**	**<toán_hạng_1> && <toán_hạng_2>**
0	0	0
0	khác 0	0
khác 0	0	0
khác 0	khác 0	1

- **Phép toán || (phép tuyển - or)**:

  Cú pháp:

  <toán_hạng_1> || <toán_hạng_2>

trong đó 2 toán hạng là các biểu thức số, kết quả của phép toán này chỉ 'sai' (0) khi và chỉ khi cả 2 toán hạng đều có giá trị 'sai' (=0).

<toán_hạng_1>	**<toán_hạng_2>**	**<toán_hạng_1> || <toán_hạng_2>**
0	0	0
0	khác 0	1
khác 0	0	1
khác 0	khác 0	1

e. Các phép toán thao tác trên bit

Trong ngôn ngữ C có nhóm các toán tử mà thao tác của nó thực hiện trên từng bit của các toán hạng và chúng được gọi là các toán tử trên bit, các toán hạng của chúng phải có kiểu số nguyên.

- **Phép & (phép and theo bit - phép hội)**

  Cú pháp: <toán_hạng_1> & <toán_hạng_2>

Chức năng của toán tử & là thực hiện phép and trên từng cặp bit tương ứng của 2 toán hạng và trả về kết quả. Tức là phép toán trả về 1 số nguyên (cùng kích thước với 2 toán hạng), bit thứ nhất của kết quả có giá trị bằng bit thứ nhất của <toán_hạng_1> hội với bit thứ nhất của <toán_hạng_2>,...

&	0	1
0	0	0
1	0	1

Bảng giá trị chân lý của &

Ví dụ int a,b, c;

1. nếu a=7; b = 14; c = a & b;
   thì c = 6;
2. nếu a= 2; b = 15; c = a & b;
   thì c = 0;
3. nếu a=-2; b = 45; c = a & b;
   thì c = 44;
4. nếu a=-2; b = -3; c = a & b;
   thì c = -4;

(nếu kết quả các ví dụ trên gây thắc mắc tại sao lại như vậy thì bạn đọc có thể tham khảo: cách biểu diễn số âm, phép AND trong phần hợp ngữ)

- **Phép | (phép or theo bit)**

Cú pháp

<toán_hạng_1> | <toán_hạng_2>

Kết quả của trả về 1 số nguyên (cùng kích thước với 2 toán hạng), các bit của giá trị trả về được tính bằng kết quả của phép tuyển (or) giữa hai bit tương ứng của <toán_hạng_1> với <toán_hạng_2>.

|	0	1
0	0	1
1	1	1

Bảng giá trị chân lý phép tuyển |

Ví dụ int a,b, c;

1. nếu a=7; b = 14; c = a | b;
   thì kết quả c = 15;
2. nếu a= 2; b = 15; c = a | b =15;

- **Phép ~ (phép đảo bit)**

Đây là toán tử một ngôi thực hiện đảo các bit của toán hạng, các bít giá trị 1 trở thành 0 và bít giá trị 0 thành 1.

Cú pháp ~<toán_hạng>

Ví dụ: 1. unsigned char c =3, d;

d = ~c; kết quả d = 252;

2. unsigned int c =3, d;

d = ~c; kết quả d = 65532;

- **Phép ^ (phép XOR - tuyển loại trừ)**

Phép tuyển loại trừ trên hai bit là phép toán xác định nếu hai bit (toán hạng) khác nhau thì kết quả theo phép tuyển, nếu hai bit có cùng giá trị thì kết quả là 0(loại trừ).

Cú pháp

<toán_hạng_1> ^ <toán_hạng_2>

^	0	1
0	0	1
1	1	0

Bảng giá trị chân lý phép tuyển loại trừ ^

Ví dụ: 1. unsigned char c = 3, d=10;

kết quả c ^ d = 2;

2. unsigned int c =10, d=10;

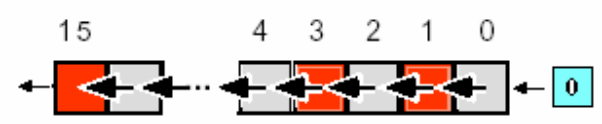
   kết quả c^d =0;

- **Phép << (dịch trái)**

Cú pháp:

toán_hạng_1 << toán *hạng*_2

Chức năng: dịch tuần tự các bit của <toán_hạng_1> sang trái số vị trí dịch bằng giá trị của <toán_hạng_2>. Khi dịch các bit của 1 toán hạng sang trái 1 thì: bit trái nhất sẽ bị loại bỏ, các bit bên phải sẽ tuần tự được dịch sang trái 1 vị trí, bit bên phải nhất sẽ được lấp bằng 0. Khi dịch trái k bit một số nào đó có thể coi là k lần liên tiếp dịch trái 1.



Dịch trái

Ví dụ : char a =12,b;

b = a<<1 thì b = 24

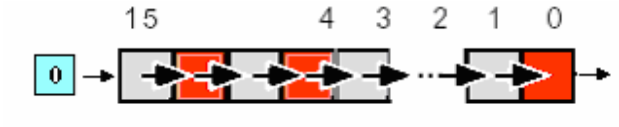
Khi dịch trái số a với số bước là k, nếu chưa xảy ra các bit có giá trị 1 của a bị mất thì kết quả sẽ là a 2^k, nhưng có khả năng khi dịch trái k bit một số a thì một số bit cao của a sẽ bị loại, tổng quát có thể tính giá trị như sau: gọi l là số bit cửaa thì kết quả là(a * 2^k % 2^l).*

- **Phép >> (dịch phải)**

Cú pháp

toán_hạng_1 >> toán_hạng_2

Lệnh này thực hiện tương tự như SHL nhưng dịch các bit của <toan_hạng_1> sang phải, các bit bên trái sẽ được điền bằng 0, các bit bên phải sẽ bị ghi đè bởi bit bên trái.



Minh hoạ toán tử >>

Khi dịch số n sang phải k bit, kết quả thu được($n/2^k$)

e. Các phép toán tích luỹ (gán số học)

Trong các biểu thức toán số học chúng ta rất hay gặp các biểu thức dạng như a = a +k, tức là chúng ta tăng a lên một lượng bằng k, hoặc như a = a << k, tức là dịch các bít của a

sang trái k vị trí rồi lại gán vào a. Trong C có các phép toán thực hiện chức năng này và ta gọi là các phép toán tích luỹ.

Cú pháp chung:

<đích> <tt>= <nguồn>;

Trong đó <đích> là một biến thuộc kiểu số nguyên hoặc thực, <nguồn> là một giá trị phù hợp. <tt> là dấu phép toán số học hay trên bit (hai ngôi): +,-,*,/,%, <<,>>,&,|,^
Với ý nghĩa

<đích> <tt>= <nguồn> ≡ <đích> = <đích> <tt> <nguồn>

toán tử	ví dụ về cách dùng	ý nghĩa
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b

Lưu ý: hai kí tự <tt> và dấu = phải viết liền và trở thành 1 dấu toán tử của ngôn ngữ C.

Bằng cách dùng toán tử tích luỹ, trong một số trường hợp chúng ta có thể giảm sự phức tạp (về cách viết) của một biểu thức rất nhiều và như vậy sẽ giảm khả năng viết sai một cách đáng kể.

Ví dụ: a[f(i)+b[j*2]-srtlen(s)] = a[f(i)+b[j*2]-strlen(s)] +6;
được viết thành a[f(i)+b[j*2]-strlen(s)] += 6;

f. ***Toán tử điều kiện***

Cú pháp:

<bt>?<gt1>:<gt2>

Trong đó <bt>, <gt1> và <gt2> là các biểu thức, nếu <bt> có giá trị 'đúng' thì kết quả của biểu thức là <gt1> ngược lại nếu <bt> có giá trị 'sai' thì biểu thức trả lại <gt2>.
Ví dụ: (a>b?a:b)
ý nghĩa của biểu thức trên là nếu a >b thì kết quả là a ngược lại là b, tức là trả lại giá trị lớn nhất của a và b.

g. *Phép tăng và giảm 1*

Với biểu thức dạng a = a + 1 hoặc a = a - 1 thì trong C có dạng khác viết ngắn gọn hơn bằng cách dùng toán tử ++ hoặc --. Mỗi toán tử này lại có hai dạng khác nhau đó là toán tử viết trước toán hạng (gọi là toán tử trước) và dạng toán tử viết sau toán hạng (gọi là toán tử sau - như vậy có 4 toán tử).

Cú pháp:

<tên_biến> ++
++<tên_biến>
<tên_biến> --
--<tên_biến>

Ví dụ: int a=5,b,c=2;

b= a++;

c = ++ a +b;

Ý nghĩa của ++ là tăng toán hạng (là biến) lên 1 đơn vị và -- là giảm toán hạng 1. Sự khác nhau giữa toán tử trước và toán tử sau được minh hoạ bằng ví dụ sau:

a = 4; b = 2;
c= b + a++ ; thì sau khi thực hiện ta có c = 6 và a = 5
hay x = b++; thì b = 3 và x=2;
nhưng nếu
a = 4; b = 2;
c= ++a +b ; thì sau khi thực hiện ta có c = 7 và a = 5
và x = ++b thì x=3, b=3.

Như vậy bạn thấy sự khác nhau giữa x= b++ ;(1) và x=++b (2); là trong (1) giá trị của b được gán cho x trước khi nó được tăng 1, còn trong (2) thì tăng giá trị của b lên 1 sau đó mới gán b cho x.

Tức là có thể hiểu: x = b++ ; ⇔ { x = b; b = b+1;}
còn x = ++b ; ⇔ { b = b+1; x = b; }

Tương tự cho toán tử --;
x = b-- ; ⇔ { x = b; b = b - 1;}
còn x = --b ; ⇔ { b = b -1; x = b; }

Vậy :

- Trong biểu thức đơn dạng a++, ++a, b--, --b thì ý nghĩa của toán tử trước và sau là như nhau (cùng là tăng hay giảm 1)
- Trong biểu thức nói chung mà có a++ (a--) hay ++b (--b) thì giá trị của a được sử dụng trong biểu thức trước khi a được tăng (giảm) 1, và giá trị của b được sử dụng sau khi b đã được tăng (giảm) 1.

Lưu ý: Bạn có thể dùng kết hợp nhiều lần toán tử ++, -- với một biến. Vì ++,-- có cùng độ ưu tiên và được kết hợp từ phải sang trái do vậy các phép toán dạng ++++a, ----a là được phép trong khi đó a++++, a---- là không được phép.

h. ***Toán tử & - lấy địa chỉ***

Các biến và hằng là các được lưu trong bộ nhớ và được cấp tại địa chỉ nào đó, toán tử & trả lại địa chỉ của một biến hay hằng.

Cú pháp: &<tên_biến>

hoặc &<tên_hằng>

i. ***Toán tử *** *(truy xuất giá trị qua con trỏ)*

Phần trên chúng ta biết * là phép nhân, nhưng nó còn có ý nghĩa là toán tử 1 ngôi với chức năng lấy giá trị của một thành phần thông qua con trỏ.

Cú pháp: * <tên_con_trỏ>

Như vậy với một biến được cấp phát tại một vùng nhớ nào đó trong bộ nhớ thì chúng ta có thể truy xuất giái trị của nó thông qua tên biến hoặc qua địa chỉ (con trỏ) của nó.

Giá sử pa là con trỏ và pa trỏ tới biến a (có kiểu phù hợp) thì *pa chính là giá trị của a. và cách truy xuất theo tên biến a hoặc qua con trỏ *pa là như nhau.

Ví dụ:
```
int a, b, c;
int *p;
p=&a;
*p = 5; b = a + 3; c =*p -1;
```

Sau các lệnh trên thì a có giá trị là 5, b là 8 và c là 4 (*truy xuất a theo cách *p gọi là truy xuất gián tiếp thông qua con trỏ)*.

j. Toán tử ,

Dấu phẩy (,) thường được dùng trong như dấu phân cách giữa các biến, các hằng được khai báo trên cùng một dòng, hoặc giữa các tham số của hàm. Trong một số trường hợp nào đó nó được dùng như một toán tử để tạo ra một biểu thức dạng A,B (với A, B là hai biểu thức con hợp lệ). Các biểu thức con được tính từ trái qua phải và giá trị của biểu thức con cuối (bên phải) chính là giá trị trả về của toàn biểu thức.

Ví dụ: a=5; b=6; c=2;

x= (a+b, a *2 +c);

kết quả x = 12

nhưng nếu x =a+b,a*2+c; thì x =11.

k. ***Phép chuyển kiểu***

Trong C cũng như một số ngôn ngữ lập trình khác, trong một biểu thức thì các toán hạng phải cùng kiểu. Tuy nhiên trong thực tế thì không thể cứng nhắc như vậy, chẳng hạn

cộng một số nguyên với một số thực rõ ràng là khác kiểu nhưng bạn vẫn có thể thực hiện được. Thực ra thì trước khi thực hiện toán tử cộng đó chương trình dịch đã thực hiện thao tác chuyển đổi kiểu của số nguyên thành số thực chúng ta gọi là phép chuyển kiểu (ép kiểu). Trong một số tình huống việc chuyển kiểu trong C có thể được chương trình dịch thực hiện tự động (gọi là ép kiểu tự động) hoặc được ép kiểu kiểu tường minh (người lập trình viết câu lệnh - toán tử chuyển kiểu).

Nói chung sự chuyển kiểu tự động xảy ra trong bốn trường hợp sau:

- Các toán hạng trong một biểu thức khác kiểu.
- Gán một biểu thức vào một biến khác kiểu.
- Truyền tham số thực sự khác kiểu với tham số hình thức.
- Giá trị trả về của hàm sau câu lệnh return khác với kiểu hàm được khai báo.

Trong trường hợp thứ nhất quy tắc chuyển kiểu từ thấp lên cao được áp dụng, tức là toán hạng có kiểu thấp hơn sẽ được tự động chuyển thành kiểu của toán hạng cao hơn theo trật tự:

char $\Rightarrow$ int $\Rightarrow$ long $\Rightarrow$ float $\Rightarrow$ double

Trong ba trường hợp cuối kiểu của giá trị vế phải được chuyển theo kiểu của biến bên trái, kiểu các tham số thực sự được chuyển theo kiểu của tham số hình thức, kiểu giá trị trả về (sau return) phải chuyển thành kiểu của hàm.

Lưu ý là chỉ chuyển kiểu giá trị tức thời của toán hạng rồi thực hiện phép toán chứ kiểu của bản thân toán hạng thì không thay đổi.

Trong một số yêu cầu chúng ta cần sự chuyển kiểu rõ ràng (ép kiểu tường minh) chứ không sử dụng quy tắc chuyển kiểu ngầm định, trong trường hợp này bạn có thể sử dụng toán tử chuyển kiểu theo cú pháp sau:

(kiểu_mới) (biểu_thức)

trong cấu trúc này (kiểu_mới) là tên một kiểu hợp lệ nào đó, và giá trị của (biểu_thức) trả về bắt buộc phải chuyển thành (kiểu_mới)

Ví dụ 1:

```
int a=5, b =2;
float c;
c = (float) a /b; thì c có giá trị =2.5.
nhưng c= a/b ; thì c lại có giá trị =2.0
```

Ví dụ 2:

```
float a =7.0;
int b;
b = (int)a % 3;
```

Trong C yêu cầu phải dùng cặp ngoặc () bao tên kiểu_mới, còn C++ thì với những kiểu_mới là tên kiểu đơn giản thì không bắt buộc phải dùng cặp (), ví dụ trong C++ bạn có thể dùng phép chuyển kiểu như int (a).

I. *Độ ưu tiên các toán tử*

Trong biểu thức có thể có nhiều toán tử, vậy điều gì giúp cho chương trình dịch thực hiện các toán tử một cách đúng đắn?. Trong các biểu thức nếu có các cặp (), thì nó sẽ quyết định thứ tự thực hiện các phép toán: trong ngoặc trước, ngoài ngoặc sau. Nhưng có những khả năng dấu ngoặc không có hoặc không đủ để quyết định tất cả các trường hợp thì khi đó C thực hiện các toán tử căn cứ vào độ ưu tiên của chúng và sử dụng một số quy tắc về các toán tử (ví dụ như khi chúng cùng độ ưu tiên thì thực hiện từ trái qua phải hay từ phải qua trái). Ví dụ với các phép toán số học +, - có cùng độ ưu tiên, nên nếu trong biểu thức có nhiều phép toán +, - và không có các dấu ngoặc quy định thứ tự thực hiện thì chúng sẽ được thực hiện từ trái qua phải. Nhưng với phép toán ++, hay các phép gán,.. chẳng hạn như

++++ a; hoặc a=b=c=d trình tự kết hợp lại từ phải qua trái.

Sau đây là bảng các toán tử và độ ưu tiên của chúng, các phép toán trên cùng dòng (thứ tự) có cùng độ ưu tiên, các toán tử trên dòng có thứ tự nhỏ hơn sẽ có độ ưu tiên cao hơn, trong bảng này có một số toán tử không được mô tả trong phần các phép toán như [], (), ., -> chúng sẽ được mô tả trong các phần thích hợp.

STT	Các phép toán	trình tự kết hợp
1.	(), [], ->,.	trái qua phải
2.	!, ~, & *(địa chỉ)*, * *(truy xuất gián tiếp)*, - *(đổi dấu)*, ++, --, (ép kiểu), sizeof	phải sang trái
3.	**(phép nhân)*, /, %	trái sang phải
4.	+, - *(phép trừ)*	trái sang phải
5.	<<, >> *(dịch bit)*	trái sang phải
6.	<, <=, >, >=	trái sang phải
7.	==, !=	trái sang phải
8.	& *(and trên bit)*	trái sang phải
9.	^	trái sang phải
10.	|	trái sang phải
11.	&&	trái sang phải
12.	||	trái sang phải

13.	? :	trái sang phải
14.	=, +=, -=, *=, /=, %=, <<=, >>=, &=, \+, ^=, |=	phải sang trái
15.	, *(dấu phẩy)*	trái sang phải

(bảng độ ưu tiên các toán tử)

III. Chương trình C

Trước khi nói đến cấu trúc tổng quát của một chương trình nguồn C, chúng ta hãy xem một ví dụ đơn giản sau đây – *chương trình in xâu* **'Chao cac ban!'** *ra màn hình*

```
1:    #include <stdio.h>
2:    #include <conio.h>
3:    void main()
4:    {
5:        clrscr();
6:        printf("\n\n Chao cac ban !");
7:        getch();
8:    }
```

(trong đoạn mã nguồn trên chúng ta thêm các số dòng và dấu : để tiện cho việc giải thích, còn trong chương trình thì không được có chúng)

Trong chương trình trên gồm hai phần chính đó là :

- Các dòng bao hàm tệp – dòng 1, 2; đăng ký sử dụng các tệp tiêu đề. Trong chương trình này chúng ta cần dùng hai file tiêu đề **stdio.h** và **conio.h.**
- Hàm main từ dòng 3 tới dòng 8. Đây là hàm chính của chương trình , dòng 3 là tiêu đề hàm cho biết tên: **main**, kiểu hàm: **void**, và đối của hàm (trong ví dụ này không có đối). Thân của hàm main bắt đầu ngay sau dấu **{** *(dòng 4),* và kết thúc tại dấu **}** *(dòng 8).*

III.1.Cấu trúc chương trình

Một chương trình C nói chung có dạng như sau

```
1:    [ các bao hàm tệp ]
2:    [ các khai báo nguyên mẫu hàm của người dùng ]
3:    [ các định nghĩa kiểu ]
4:    [ các định nghĩa macro ]
5:    [ các định nghĩa biến, hằng ]
6:    <kiểu_hàm> main ( [khai báo tham số ] )
7:       {
8:              < thân hàm main>
9:       }
10:   [ các định nghĩa hàm của người dùng]
```

(trong cú pháp trên chúng ta thêm số hiệu dòng và dấu: để cho việc giải thích được thuận lợi, các thành phần trong ngoặc [] là các thành phần tuỳ chọn)

a. ***Các bao hàm tệp*** *(dòng 1)*

Trong chương trình C (trong hàm main cũng như các hàm khác do người lập trình viết) có thể sử dụng các hàm, hằng, kiểu dữ liệu,..(gọi chung là các thành phần) đã được định nghĩa trong thư viện của C. Để sử dụng các thành phần này chúng ta phải chỉ dẫn cho chương trình dịch biết các thông tin về các thành cần sử dụng, các thông tin đó được khai báo trong tệp gọi là tệp tiêu đề (có phần mở rộng là H – viết tắt của header). Và phần các bao hàm tệp là các chỉ dẫn để chương trình gộp các tệp này vào chương trình của chúng ta. trong một chương trình chúng ta có thể không dùng hoặc dùng nhiều tệp tiêu đề.

Cú pháp của một dòng bao hàm tệp:

#include <tên_tệp>

hoặc

#include "tên_tệp"

trong đó **tên_tệp** là tên có thể có cả đường dẫn của tệp tiêu đề (.H) mà chúng ta cần sử dụng, mỗi lệnh bao hàm tệp trên một dòng.

Ví dụ:

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

#include “phanso.h”

Sự khác nhau giữa cặp <> và “” bao quanh tên tệp là với cặp <> chương trình dịch tìm tên tệp tiêu đề trong thư mục ngầm định xác định bởi đường dẫn trong mục Option/Directories, còn với cặp “” chương trình dịch tìm tệp trong thư mục hiện tại, nếu không có mới tìm trong thư mục các tệp tiêu đề ngầm định như trường hợp <>.

b. Các khai báo nguyên mẫu và định nghĩa hàm của người dùng

Trong phần này chúng tôi nêu một số thông tin về khai báo nguyên mẫu và định nghĩa hàm để giải thích cấu trúc chương trình chứ không có ý định trình bày về hàm, chi tiết về hàm sẽ được trình bày trong phần về hàm.

- *Các nguyên mẫu (dòng 2)*

Nguyên mẫu một hàm là dòng khai báo cung cấp các thông tin: tên hàm, kiểu hàm, số đối số và kiểu từng đối số của hàm.

Cú pháp khai báo nguyên mẫu

<kiểu_hàm> <tên_hàm> ([khai báo đối]);

Ví dụ:

```
int min (int, int);
float binhphuong (float y);
float giatri(int , float);
```

Lưu ý: Phần khai báo đối của nguyên mẫu, mục đích là xác định số đối số và kiểu của từng đối số, do vậy bạn có thể không viết tên của đối số nhưng kiểu của chúng thì phải có và bạn phải liệt kê đầy đủ kiểu của từng đối.

- *Các định nghĩa hàm của người dùng (dòng 10)*

Trong phần này chúng ta định nghĩa các hàm của người dùng, một định nghĩa hàm bao gồm dòng tiêu đề của hàm và thân của hàm, với cú pháp như sau:

```
<kiểu_hàm> <tên_hàm> ([ khai báo đối ])
  {
     < thân  hàm >
  }
```

Ví dụ:

```
int min(int a, int b)     {
      if(a>=b)
            return b;
      else  return a;
   }
```

Lưu ý:
- *Tiêu đề hàm trong định nghĩa hàm phải tương ứng với nguyên mẫu hàm*
- *Nếu trong chương trình định nghĩa hàm xuất hiện trước khi gặp lời gọi hàm đó thì có thể không nhất thiết phải có dòng khai báo nguyên mẫu hàm.*

c. ***Định nghĩa kiểu mới*** *(dòng 4)*

Ngoài những kiểu chuẩn đã được cung cấp sẵn của ngôn ngữ, người lập trình có thể định nghĩa ra các kiểu mới từ những kiểu đã có bằng cách sử dụng từ khoá **typedef** (*type define)* Với cú pháp như sau

typedef <mô_tả_kiểu> <tên_kiểu_mới>;

Trong đó <tên_kiểu_mới> là tên kiểu cần tạo do người lập trình đặt theo quy tắc về tên của ngôn ngữ, và <mô_tả_kiểu> là phần chúng ta định nghĩa các thành phần cấu thành lên kiểu mới.

Ví dụ:

typedef unsigned char byte;

typedef long nguyendai;

Sau định nghĩa này các tên mới byte được dùng với ý nghĩa là tên kiểu dữ liệu nó tương tự như unsigned char, và nguyendai tương tự như long.

Ví dụ: chúng ta có thể định nghĩa biến a, b kiểu byte như sau

byte a,b;

d. ***Định nghĩa macro*** *(dòng 5)*

Khái niệm macro là gì? Giả sử như bạn có một nội dung (giá trị) nào đó và bạn muốn sử dụng nó nhiều lần trong chương trình, nhưng bạn không muốn viết trực tiếp nó vào chương trình lúc bạn soạn thảo vì một vài lý do nào đó (chẳng hạn như nó sẽ làm chương trình khó đọc, khó hiểu, hoặc khi thay đổi sẽ khó,..). Lúc này bạn hãy gán cho nội dung đó một 'tên' và bạn sử dụng 'tên' đó để viết trong chương trình nguồn. Khi biên dịch chương trình, chương trình dịch sẽ tự động thay thế nội dung của 'tên' vào đúng vị trí của 'tên' đó. Thao tác này gọi là phép thế macro và chúng ta gọi 'tên' là tên của macro và nội dung của nó được gọi là nội dung của macro.

Một macro được định nghĩa như sau:

#define tên_macro nội_dung

Trong đó tên macro là một tên hợp lệ, nội dung (giá trị) của macro được coi thuần tuý là 1 xâu cần thay thế vào vị trí xuất hiện tên của macro tương ứng, giữa tên và nội dung cách nhau 1 hay nhiều khoảng trống (dấu cách). Nội dung của macro bắt đầu từ kí tự khác dấu trống đầu tiên sau tên macro cho tới hết dòng.

Ví dụ :

```
# define SOCOT 20
# define max(a,b) (a>?b a:b)
```

Với hai ví dụ trên, khi gặp tên SOCOT chương trình dịch sẽ tự động thay thế bởi 20 và max(a,b) sẽ được thay thế bởi (a>b?a:b)

Chú ý:

- *Phép thay thế macro đơn giản chỉ là thay nội dung macro vào vị trí tên của nó do vậy sẽ không có cơ chế kiểm tra kiểu.*
- *Khi định nghĩa các macro có 'tham số' có thể sau khi thay thế biểu thức mới thu được có trật tự tính toán không như bạn mong muốn. Ví dụ ta có macro tính bình phương 1 số như sau:*

  *# define bp(a) a*a*

  *và bạn có câu lệnh bp(x+y) sẽ được thay là x+y*x+y và kết quả không như ta mong đợi. Trong trường hợp này bạn nên sử dụng dấu ngoặc cho các tham số của macro*

  # define bp(a) (a)(a)*

e. ***Định nghĩa biến, hằng*** (dòng 5)

Các biến và hằng được định nghĩa tại đây sẽ trở thành biến và hằng toàn cục. Ý nghĩa về biến, hằng, cú pháp định nghĩa đã được trình bày trong mục biến và hằng.

f. ***Hàm main*** *(dòng 6-9)*

Đây là thành phần bắt buộc duy nhất trong một chương trình C, thân của hàm main bắt đầu từ sau dấu mở móc { (dòng 7) cho tới dấu đóng móc } (dòng 8). Không giống như chương trình của Pascal luôn có phần chương trình chính, chương trình trong C được phân thành các hàm độc lập các hàm có cú pháp như nhau và cùng mức, và một hàm đảm nhiệm phần thân chính của chương trình, tức là chương trình sẽ bắt đầu được thực hiện từ dòng lệnh đầu tiên và kết thúc sau lệnh cuối cùng trong thân hàm main .

Trong định nghĩa một hàm nói chung đều có hai phần đó là tiêu đề của hàm, dòng này bao gồm các thông tin : Tên hàm, kiểu hàm (kiểu giá trị hàm trả về), các tham số hình thức (tên tham số và kiểu của chúng). Phần thứ hai là thân của hàm, đây là tập các lệnh (hoặc khai báo) thực hiện các thao tác theo yêu cầu về chức năng của hàm đó. Hàm main cũng chỉ là một trường hợp riêng của hàm nhưng có tên cố định là main, có thể có hoặc không có các đối số, và có thể trả về giá trị cho hệ điều hành, kiểu của giá trị này được xác định bởi <kiểu_hàm> (dòng 6) – chi tiết về đối, kiểu của hàm main sẽ được đề cập kỹ hơn trong các phần sau.

Thân hàm main được bao bởi cặp {*(dòng 7)*, và } *(dòng 9)* có thể gồm các lệnh, các khai báo hoặc định nghĩa biến, hằng, kiểu, các thành phần này trở thành cục bộ trong hàm main - *vấn đề cục bộ, toàn cục sẽ đề cập tới trong phần phạm vi.*

➢ **Lưu ý:**

- Các thành phần của chương trình mà chúng ta vừa nêu trừ hàm main là thành phần phải có và duy nhất trong một chương trình C, còn các thành phần khác là tuỳ chọn, có thể không có hoặc có nhiều.
- Thứ tự các thành phần không bắt buộc theo trật tự như trên mà chúng có thể xuất hiện theo trật tự tuỳ ý nhưng phải đảm bảo yêu cầu mọi thành phần phải được khai báo hay định nghĩa trước khi sử dụng.
- Các biến, hằng khai báo ngoài mọi hàm có phạm vi sử dụng là toàn cục (tức là có thể sử dụng từ sau lệnh khai báo cho tới hết file chương trình). Các hằng, biến khai báo trong 1 hàm (hoặc trong 1 khối) là thành phần cụ bộ (có phạm vi sử dụng trong hàm hoặc trong khối đó mà thôi).
- Các hàm trong C là một mức (tức là trong hàm không chứa định nghĩa hàm khác).

Ví dụ: *chương trình nhập bán kính từ bàn phím, tính và in diện tích hình tròn*

```
#include <stdio.h>
#include <conio.h>
#define PI 3.1415
float r;                    // Khai báo biến r có kiểu float
void main()
 { printf("\nNhap ban kinh duong tron r =");
   scanf("%f",&r);              //nhập số thực từ bàn phím vào r
   printf("Dien tich = %5.2f", r*r*PI); //tính và in diện tích
   getch();
 }
```

III.2. Câu lệnh và dòng chú thích

III.2.1. Câu lệnh

Trong chương trình có thể có nhiều câu lệnh, mỗi câu lệnh đảm nhiệm một chức năng nào đó. Trong C một lệnh nói chung có thể viết trên một hay nhiều dòng (trừ xâu kí tự và macro) và kết thúc bởi dấu chấm phẩy (**;**) và cũng có thể viết nhiều lệnh trên một dòng, giữa các thành phần của lệnh có thể có các dấu cách.

Ví dụ:

```
a = b +5;
a  = b +
```

```
        5;
    printf("Dien tich = %5.2f", r*r*PI);
```

Một lệnh có thể viết trên nhiều dòng nhưng trong 1 xâu kí tự hay định nghĩa macro thì chúng ta phải viết trên 1 dòng, trường hợp nhất thiết phải viết trên nhiều dòng thì bạn phải thêm kí tự \ vào cuối dòng trên để báo cho chương trình dịch nối nội dung dòng dưới vào cuối của dòng trên.

Ví dụ

```
    printf("Dien tich \
            = %5.2f", r*r*PI);
```

III.2.2. Lệnh và khối lệnh

Các lệnh cúa chương trình C bao gồm 2 loại đó là câu lệnh đơn và khối lệnh (câu lệnh ghép - nhóm lệnh).

Câu lệnh đơn là những lệnh đơn giản (chỉ một phát biểu, kết thúc bởi ;) như phép gán, một lời gọi hàm,..

Khối lệnh là nhóm các lệnh được bao bởi cặp **{** và **}**, bên trong khối lệnh là dãy các lệnh có thể là lệnh đơn hoặc khối lệnh con khác, tức là khối lệnh có thể lồng nhau, các dấu móc { và } phải xuất hiện tương ứng theo cặp.

Ví dụ:

```
if (a>0)
  {  d  = b*b - 4*a*c;
     if(d>=0)
        {  x1  =  (-b - sqrt(d))/(2*a);
           x2  =  (-b + sqrt(d))/(2*a);
           printf(" nghiem x1 = %4.2f, x2 = %4.2f",x1,x2);
        }
     else
        printf("phuong trinh khong co nghiem thuc");
  }
```

III.2.3. Lời chú thích

Trong chương trình chúng ta có thể thêm các lời chú thích để giải thích câu lệnh hoặc chức năng của chương trình ,.. nhằm cho chương trình dễ đọc.

Các chú thích được đặt giữa cặp /* và */, có thể trên một hoặc nhiều dòng. Với các chương trình dịch của C++ bạn có thể sử dụng // để ghi một chú thích trong chương trình, với cách này nội dung lời chú thích bắt đầu sau dấu // tới hết dòng.

Các lời chú thích chỉ có tác dụng với người đọc chứ không ảnh hưởng tới chương trình, tức là chương trình dịch sẽ bỏ qua các lời chú thích.

Ví dụ:

```
scanf("%f",&r);              /*nhập số thực từ bàn phím vào r */
printf("Dien tich = %5.2f", r*r*PI); //tính và in diện tích
```

III.3.Nhập và xuất dữ liệu

Trong phần này chúng ta giới thiệu cú pháp và ý nghĩa một số hàm cơ bản để nhập dữ liệu từ thiết bị vào chuẩn là bàn phím và xuất dữ liệu ra màn hình máy tính. Để sử dụng các hàm nói chung của thư viện bạn phải bao hàm các tệp tiêu đề (tệp .h) chứa khai báo nguyên mẫu của chúng vào chương trình.

- ***Một số hàm nhập dữ liệu từ bàn phím***

a. Hàm* getch, getche *nhập 1 ký tự

Cú pháp:

int getch();
int getche();

Chức năng: Hai hàm này thực hiện đợi người dùng nhập một ký tự từ bàn phím và trả về một số nguyên là mã của kí tự được bấm, ví dụ bạn gõ phím 'a' thì hàm sẽ trả về 97.

Sự khác nhau giữa hai hàm là hàm getche hiện kí tự được nhập lên màn hình, còn getch thì không.

Khi phím được bấm là phím mở rộng thì hệ thống sẽ đẩy vào bộ đệm nhập liệu 2 byte, byte thứ nhất có giá trị 0, byte thứ 2 là mã mở rộng của phím đó. Ví dụ khi bạn bấm phím mũi tên lên ↑ thì hai byte có giá trị là 0 72 và hàm getch hay getche trả về 0, byte có giá trị 72 vẫn còn lưu trong bộ đệm nhập liệu, nếu ta gọi getch hoặc getche sẽ nhận được giá trị này.

b. Hàm scanf

Đây là một trong những hàm nhập dữ liệu phổ biến nhất của C, nó cho phép nhập nhiều loại dữ liệu (có các kiểu khác nhau). Khi nhập dữ liệu bằng hàm này bạn phải xác định địa chỉ (vùng nhớ, hay biến) để lưu dữ liệu và kiểu của dữ liệu cần nhập.

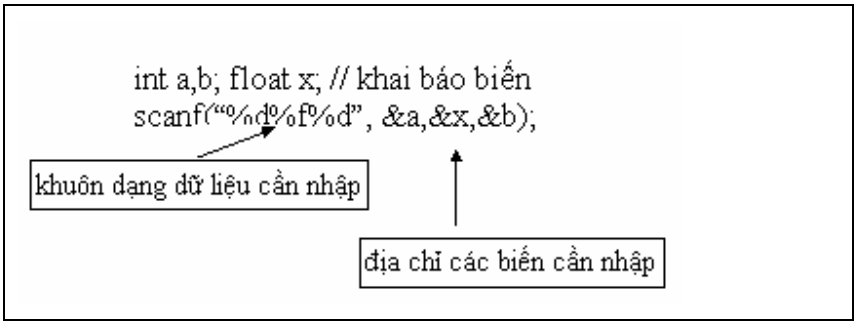
cú pháp

int scanf(const char * format, ds_các_con_trỏ);

chức năng

Hàm scanf cho phép chúng ta nhập dữ liệu từ bàn phím theo khuôn dạng được xác định bởi xâu kí tự **format**, dữ liệu nhập vào sẽ lưu vào các biến hoặc vùng nhớ có địa chỉ tương ứng là các con trỏ trong **ds_các_con_trỏ** (có thể có nhiều con trỏ, mỗi con trỏ cách nhau bởi dấu phẩy).

Ví dụ: nhập giá trị cho 3 biến a có kiểu int, x có kiểu float, và b có kiểu int



Trong cú pháp trên **format** là một xâu quy định quy cách dữ liệu cần nhập, gồm nhiều **đặc tả** dữ liệu tương ứng với các kiểu của con trỏ trong phần ds_các_con_trỏ, có bao nhiêu con trỏ thì cần đúng bấy nhiêu đặc tả, đặc tả thứ nhất quy định khuôn dạng dữ liệu cho con trỏ thứ nhất, đặc tả thứ 2 quy định khuôn dạng dữ liệu cho con trỏ thứ 2,...

Mỗi đặc tả bắt đầu bằng dấu **%** có dạng sau (*các thành phần trong* [] *là tuỳ chọn*):

%[*][n]<ký_tự_định_kiểu>

Trong đó

- **n** là một số nguyên dương quy định độ dài tối đa (tính theo số kí tự) được nhập cho thành phần tương ứng
- **<ký_tự_định_kiểu>** là kí tự quy định kiểu dữ liệu cần nhập ví dụ bạn muốn nhập số nguyên kiểu int thì kí tự định kiểu là **d,** kiểu ký tự là **c.** Các kí tự định kiểu khác bạn xem bảng sau.

Kí tự định kiểu	dữ liệu nhập	kiểu con trỏ của đối nhập liệu
d	integer	int *arg
D, ld	integer	long *arg
e, E	Float	float *arg
f	Float	float *arg
g, G	Float	float *arg
o	Octal	int *arg
O	Octal	long *arg
i	Decimal,octal, hex	int *arg
I	Decimal,octal, hex	long *arg

u	Unsigned int	unsigned int *arg
U	Unsigned int	unsigned long *arg
x	Hexadecimal	int *arg
X	Hexadecimal	int *arg
s	Character string	char arg[]
c	Character	char *arg

- * đây cũng là thành phần tuỳ chọn, nếu có thì tác dụng của nó là sẽ bỏ qua một thành phần dữ liệu được xác định bởi đặc tả này, như vậy sẽ không có đối tương ứng với đặc tả này.

  Ví dụ:

  scanf("%d%*c%d",&a,&b);

  trong dòng này chúng ta sẽ nhập 1 thành phần (gọi là 1 trường) số nguyên vào a, sau đó bỏ qua một thành phần là kí tự, và tiếp theo là một số nguyên vào b.

- ***Quy cách nhập dữ liệu***

Khi chúng ta nhập dữ liệu từ bàn phím, kết thúc nhập bằng Enter (↵), thì tất cả những kí tự chúng ta gõ trên bàn phím đều được lưu trong vùng đệm nhập dữ liệu (gọi là dòng vào- stdin) - dòng vào kết thúc bởi (↵), dữ liệu trên dòng vào này sẽ được cắt thành từng trường tuần tự từ trái qua phải và gán vào các biến (hoặc vùng nhớ) xác định tương ứng bởi các con trỏ, các phần đã tách được sẽ bị loại khỏi dòng vào.

Trước khi tách giá trị một trường thì các khoảng trắng phía trước của trường nếu có sẽ bị loại bỏ. Nếu trong đặc tả không có thành phần (**n**) quy định độ dài tối đa một trường thì **c**ác trường được xác định bởi các ký tự dấu cách, tab, enter (gọi chung là khoảng trắng ký hiệu là ␣) hoặc khi gặp ký tự không phù hợp với đặc tả hiện tại.

Nếu trên dòng vào có nhiều hơn các thành phần yêu cầu của hàm nhập thì các thành phần chưa được nhận vẫn còn lưu trên dòng vào.

Ví dụ:

int a,b; float x;

scanf("%d%%d%f",&a,&b, &x);

với dòng vào là:

␣143 ␣ ␣ 535 ␣ 34 ↵

thì :

- khoảng trắng đầu tiên bị loại bỏ, 143 là trường thứ nhất được gán vào a,
- hai khoảng trắng bị loại bỏ, 535 là trường thứ hai được gán vào b,
- một khoảng trắng bị loại bỏ, 34 được gán vào x (còn lại ↵ trong dòng vào)

Nếu trong đặc tả có thành phần xác định độ rộng tối đa (**n**) thì một trường sẽ kết thúc hoặc khi gặp khoảng trống, hay kí tự không phù hợp hoặc đã đủ độ dài **n**

Ví dụ

int a,b; float x;

scanf("%d%2d%3f",&a,&b, &x);

với dòng vào là:

␣143 ␣ ␣ 537 ␣ 34 ↵

thì :

- khoảng trắng đầu tiên bị loại bỏ, 143 là trường thứ nhất được gán vào a,
- hai khoảng trắng bị loại bỏ, 53 là trường thứ hai được gán vào b,
- một khoảng trắng bị loại bỏ, 7 được gán vào x (còn lại ␣34↵ trong dòng vào)

Lưu ý:

- *Số các đặc tả phải tương ứng với số con trỏ trong danh sách con trỏ*
- *Ký tự định kiểu trong đặc tả phải phù hợp với kiểu của con trỏ cần nhập liệu.*
- *Dữ liệu nhập từ bàn phím phải phù hợp với các đặc tả.*
- *Hàm scanf trả về số nguyên là số trường được nhập dữ liệu*

c. Hàm gets

Cú pháp:

char * gets(char * s);

Chức năng của hàm gets là nhập một xâu kí tự từ bàn phím, khác với hàm scanf với đặc tả "%s" kết thúc nội xâu khi gặp dấu cách hoặc enter, tức là xâu không thể có dấu cách, hàm gets chỉ kết thúc khi gặp enter (kí tự '\n'). Xâu kí tự được ghi vào s (với s là mảng các kí tự hoặc con trỏ kí tự), dấu kết thúc xâu ('\0' - kí tự có mã 0) được tự động thêm vào cuối xâu. Hàm trả về địa chỉ của xâu được nhập.

Chú ý: hàm gets loại bỏ ký tự Enter('\n') trên dòng vào nhưng ký tự này không được đưa vào s mà tự động thêm ký tự kết thúc xâu ('\0') vào cuối của s.

➢ *Một số hàm xuất dữ liệu ra màn hình*

a. Hàm printf

Hàm prinntf là hàm in dữ liệu ra màn hình rất đa dạng của ngôn ngữ C. Cũng như hàm scanf, hàm printf cũng yêu cầu chúng ta phải cung cấp các giá trị và định dạng của dữ liệu cần in thông qua các đối của hàm.

Cú pháp

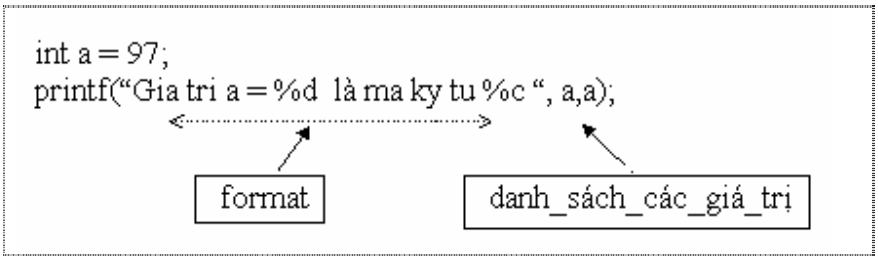
int printf (const char * format [, <danh_sách_các_giá trị>]);

Trong đó: **<danh_sách_các_giá trị>** là phần tuỳ chọn, nếu có thì đó là các giá trị cần in, các giá trị (có thể là biến, hằng, lời gọi hàm, hay biểu thức nói chung) cách nhau bởi dấu phẩy.

*Lưu ý: số giá trị trong **<danh_sách_các_giá trị>** có thể nhiều hơn số các đặc tả, khi đó các giá trị cuối (không có đặc tả tương ứng) sẽ bị bỏ qua.*

format là xâu ký tự điều khiển, nhiệm vụ chính của nó là điều khiển khuôn dạng thông tin được in ra màn hình.

Ví dụ:



Trong format gồm ba loại: các kí tự điều khiển, các đặc tả, các kí tự thường

- *Các ký tự điều khiển*

  Đây là các kí tự đặc biệt, bắt đầu bằng kí tự \ tiếp theo là 1 kí tự dùng để điều khiển: chuyển con trỏ màn hình, vị trí in dữ liệu,..
  - \n : chuyển con trỏ màn hình xuống dòng mới
  - \t : dấu tab
  - \b : (backspace) lùi một kí tự (xoá kí tự trước vị trí con trỏ hiện tại)

- *Các ký tự thường*

  Là những kí tự không thuộc loại điều khiển và đặc tả, các kí tự này được in ra màn hình đúng như nó xuất hiện trong format. Ngoài ra còn có một vài kí tự đặc biệt mà khi muốn in ra màn hình chúng ta phải đặt nó ngay sau kí tự \, đó là:
  - \\ : để in chính dấu \
  - \' : in dấu nháy đơn(')
  - \\" : in dấu nháy kép (")

- *Các đặc tả*

  Trong format có thể có nhiều đặc tả, các đặc tả quy định khuôn dạng dữ liệu cần in ra, mỗi đặc tả có dạng như sau :

  %[-][n[.m]]<ký_tự_định_kiểu>

Ý nghĩa các thành phần

- *Thành phần <ký_tự_định_kiểu>* đây là kí tự quy định kiểu của dữ liệu cần in ví dụ bạn muốn in một giá trị int thì <ký_tự_định_kiểu> là **d**, bạn muốn in một kí tự thì kí tự định kiểu là **c. C**ác kiểu khác được cho trong bảng sau:

Kí tự định kiểu	Kiểu của giá trị cần in	Khuôn dạng in ra
d	int	số nguyên hệ 10
i	int	số nguyên hệ 10
o	int	số nguyên không dấu hệ 8
u	unsigned int	số nguyên không dấu hệ 10
x,X	int,unsigned	số nguyên không dấu hệ 16
f	float	số thực (dạng dấu phẩy tĩnh)
e, E	float	số thực (dấu phẩy tĩnh hoặc kí pháp khoa học)
c	char	kí tự
s	char *	xâu kí tự
p	con trỏ	in giá trị con trỏ dạng Segment:Offset hoặc Offset tuỳ mô hình bộ nhớ được sử dụng

Lưu ý: Có thể dùng kết hợp ld, lu, lx,.. để định kiểu dữ liệu in ra là số nguyên dài (long), số nguyên dài không dấu (unsigned long),..

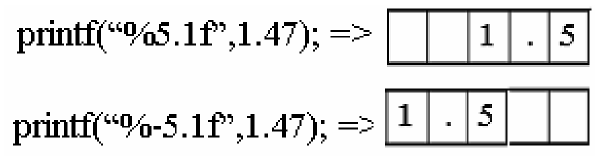
- *Thành phần* [n[.m]] : n, m là các số nguyên dương, n quy định độ rộng của thông tin (tính theo số ký tự) được in ra màn hình, m số chữ số cho phần thập phân (chỉ dùng cho số thực), nếu có m thì số thực được làm tròn với m chữ số thập phân. Nếu độ rộng thực sự của giá trị cần in < độ rộng được dành cho nó (n) thì các dấu trống được thêm vào (bên trái hay bên phải tuỳ vào sự có mặt của thành phần [-] hay không).

  *Lưu ý: có thể thay số n bằng kí tự *, khi đó thông tin sẽ được in ra theo đúng độ rộng thực sự của nó.*

  ví dụ printf("%5.1f",1.37); sẽ in ra 1.4 và chiếm 5 vị trí trên màn hình, bên trái của số được điền 2 dấu cách.

- *Thành phần* [-]: Xác định kiểu căn bên trái hay bên phải. Khi một giá trị được in ra trên màn hình, nếu độ rộng thực sự của nó nhỏ hơn độ rộng xác định bởi thành phần n, ngầm định chúng được căn bên phải (trong vùng n kí tự trên màn hình), nếu có dấu - thì dữ liệu được căn trái.

Ví dụ:

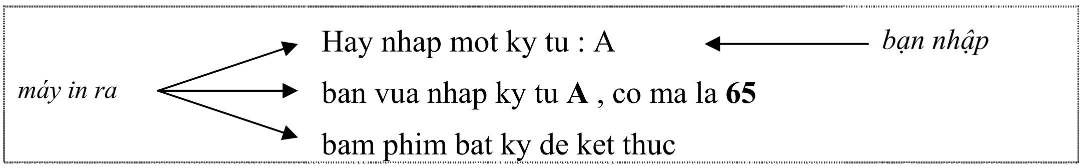


Giá trị trả về của hàm printf là tổng độ dài thông tin được in (tính theo ký tự).

Ví dụ 1: chương trình nhập 1 kí tự từ bàn phím, in kí tự và mã của nó ra màn hình

```
#include <stdio.h>
#include <conio.h>
void main(){
   int c;
   printf("\nHay nhap mot ky tu : ");
   c = getch();
   printf("\nban vua nhap ky tu %c, co ma la %d ",c,c);
   printf("\nbam phim bat ky de ket thuc");
   getch();
 }
```

khi thực hiện chương trình trên chúng ta sẽ được:



như vậy cùng là giá trị c (mã của kí tự chúng ta gõ từ bàn phím), nhưng với đặc tả khác nhau sẽ cho chúng ta khuôn dạng khác nhau (*trong ví dụ với đặc tả %d in ra 65, nhưng với %c lại in ra kí tự A).*

Ví dụ 2: *chương trình nhập 2 số nguyên a, b từ bàn phím (b !=0), in tổng, hiệu, tích, thương phần nguyên a/b*

```
#include <stdio.h>
#include <conio.h>
void main() { int a,b,c;
  printf("\nnhap a, b :");
  scanf("%d%d",&a,&b);
  printf("\na+b= %5d \na-b= %5d", a+b, a-b);
  printf("\na*b= %5d \na/b= %5d", a*b, a/b);
  getch();
  }
```

kết quả chạy chương trình là

```
nhap a, b : 5 2 ↵
a+b= 7
a-b= 3
a*b= 10
a/b= 2
```

b. ***Hàm putch***

Cú pháp:

int putch(int ch);

Chức năng: Hàm này in kí tự có mã là ch ra màn hình tại vị trí hiện tại của con trỏ, chuyển con trỏ sang phải 1 ký tự, hàm trả về số nguyên chính là mã kí tự in ra.

Ví dụ: minh hoạ putch

```
1:    #include <stdio.h>
2:    #include <conio.h>
3:    void main(){
4:       int c;
5:       c = 97;
6:       printf("\nprint c = %d", c);
7:       printf("\nputch c = "); putch(c);
8:       c = 354;
9:       printf("\nprint c = %d" , c);
10:      printf("\nputch c = "); putch(c);
11:      getch();
12:   }
```

khi thực hiện chương trình trên các bạn sẽ thu được kết quả như sau:

```
   print c = 97
s  putch c = a
   c = 354
   print c = 354
   putch c = b
```

Các bạn biết là một kí tự chỉ có kích thước 1 byte, nhưng trong hàm putch lại có đối là int (2 byte), trong trường hợp giá trị của ch >255 thì kí tự được in ra là kí tự có mã (ch % 256), và đây cũng là giá trị mà putch trả về. Cũng giống như với printf một số kí tự đặc biệt được coi là ký tự điều khiển chứ không phải kí tự in ra màn hình.

c. ***Hàm puts***

Cú pháp:

int puts(char * s);

Chức năng: Hàm này in xâu kí tự s ra màn hình tại vị trí hiện tại của con trỏ, sau đó tự động chuyển con trỏ sang dòng mới. Trong trường hợp in thành công hàm trả về số nguyên dương, ngược lại trả về -1(EOF).

IV - Các cấu trúc điều khiển chương trình

Một chương trình là tập nhiều câu lệnh, thông thường một cách trực quan chúng ta hiểu chương trình thực hiện tuần tự các lệnh từ trên xuống dưới, bắt đầu từ lệnh thứ nhất trong hàm main và kết thúc sau lệnh cuối cùng của nó. Nhưng thực tế chương trình có thể phức tạp hơn sự tuần tự nhiều, chẳng hạn như một câu lệnh (hay khối lệnh) chỉ được thực hiện khi có một điều kiện nào đó đúng, còn ngược lại nó sẽ bị bỏ qua, tức là xuất hiện khả năng lựa chọn một nhánh nào đó. Hay một chức năng nào đó có thể phải lặp lại nhiều lần. Như vậy với một ngôn ngữ lập trình có cấu trúc nói chung phải có các cấu trúc để điều khiển trình tự thực hiện các lệnh trong chương trình (gọi ngắn gọn là các cấu trúc hoặc các toán tử điều khiển)

Sau đây chúng ta sẽ tìm hiểu từng cấu trúc điều khiển chương trình của C.

IV.1. Cấu trúc tuần tự

Đây là cấu trúc đơn giản nhất của các ngôn ngữ lập trình nói chung, điều khiển thực hiện tuần tự các lệnh trong chương trình (bắt đầu từ các lệnh trong thân hàm main) theo thứ tự từ trên xuống dưới (nếu không có điều khiển nào khác).

Ví dụ 1.1: Chương trình nhập năm sinh của một người từ bàn phím, sau đó in ra lời chào và tuổi của người đó.

```
#include <stdio.h>
#include <conio.h>
void main()
  { int namsinh;
    printf("Nhap nam sinh cua ban : ");
    scanf("%d", &namsinh);
    printf("\n\nChao ban! nam nay ban %4d tuoi",2002-
    namsinh);
    getch();
  }
```

Ví dụ 1.2: Viết chương trình nhập ba số thực a,b,c từ bàn phím là số đo 3 cạnh tam giác, sau đó tính và in chu vi và diện tích của tam giác.

Giải

Dữ liệu vào : a,b,c kiểu float là 3 cạnh một tam giác

Tính toán : chu vi p = (a+b+c),
diện tích s = sqrt(q*(q-a)*(q-b)*(q-c))
với q = p/2, sqrt là hàm tính căn bậc 2

Chúng ta có chương trình như sau:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{ float a,b,c, p,q,s;
  printf("Nhap so do 3 canh cua tam giac ");
  printf("\na = "); scanf("%f", &a);
  printf("\nb = "); scanf("%f", &b);
  printf("\nc = "); scanf("%f", &c);
  p = a+b+c;  q = p/2;
  s = sqrt(q*(q-a)*(q-b)*(q-c));
  printf("\n\nChu vi la %5.1f, dien tich la %5.2f ",p,s);
  getch();}
```

ví dụ về kết quả thực hiện chương trình

```
Nhap so do 3 canh cua tam giac
a = 3 ↵
b = 4 ↵
c = 5 ↵
Chu vi la 12.0, dien tich la 6.00
```

Lưu ý:

- *Trong chương trình ví dụ trên chúng ta sử dụng hàm tính căn bậc 2 sqrt, hàm này được khai báo trong tệp tiêu đề math.h*
- *Chương trình trên chưa xử lý trường hợp a,b,c không hợp lệ (ba số a,b,c có thể không thoả mãn là 3 cạnh một tam giác)*

IV.2. Cấu trúc rẽ nhánh

Chúng ta hãy xem lại chương trình trong ví dụ 2 trên, điều gì xảy ra nếu dữ liệu không thoả mãn, tức là khi bạn nhập 3 số a,b,c từ bàn phím nhưng chúng không thoả mãn là số đo 3 cạnh một tam giác trong khi chương trình của chúng ta vẫn cứ tính và in diện tích.

Rõ ràng là có hai khả năng:

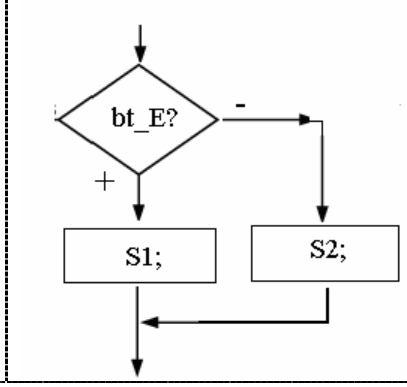
- Nếu a,b,c thoả mãn là 3 cạnh tam giác thì tính chu vi, diện tích và in kết quả
- Ngược lại phải thông báo dữ liệu không phù hợp

Như vậy cần phải có một sự lựa một trong hai nhánh tuỳ vào điều kiện a,b,c có là ba cạnh một tam giác hay không. Điều này gọi là rẽ nhánh, và C cung cấp cho chúng ta một cấu trúc điều khiển rẽ nhánh **if.**

- **Cú pháp**

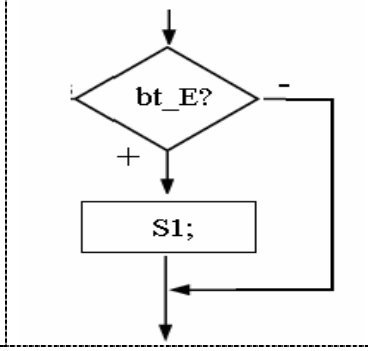
dạng đủ

```
if (bt_E)
    S1;
else
    S2;
```



hoặc dạng khuyết

```
if (bt_E)
    S1;
```



Trong cú pháp trên S1, S2 chỉ là 1 lệnh, <bt_E> là biểu thức điều kiện của if

Sự hoạt động của cấu trúc if:

Trước hết biểu thức điều kiện <bt_E> được tính, có hai khả năng:

→ Nếu giá trị của <bt_E> là 'đúng' (!=0) thì S1 được thực hiện và ra khỏi cấu trúc if.

→ Ngược lại nếu <bt_E> là 'sai' thì

- với dạng đầy đủ : thực hiện S2 rồi kết thúc if
- với dạng khuyết : kết thúc cấu trúc if

Ví dụ 2.1: chương trình nhập hai số nguyên a, b từ bàn phím, in số lớn nhất ra màn hình.

```
#include <stdio.h>
#include <stdio.h>
void main() {
int a,b;
printf("\nNhap so thu nhat :  ");
scanf("%d", &a);
printf("\nNhap so thu hai :  ");
scanf("%d", &b);
if (a>b)
   printf('\nSo lon nhat la %d", a);
else
   printf('\nSo lon nhat la %d", b);
getch();
}
```

Ví dụ 2.2: Viết chương trình theo ví dụ 1.2 với yêu cầu nếu a,b,c thỏa mãn là 3 cạnh một tam giác thì tính và in chu vi, diện tích, nếu không thoả mãn thì in thông báo ra màn hình.

Giải: Chúng ta biết 3 số a, b, c là 3 cạnh một tam giác nếu nó thoả mãn tính chất tổng hai cạnh lớn hơn cạnh thứ ba. Theo cách tính từ ví dụ 1.2 chúng ta có chương trình:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{ float a,b,c, p,q,s;
  printf("Nhap so do 3 canh cua tam giac ");
  printf("\na = "); scanf("%f", &a);
  printf("\nb = "); scanf("%f", &b);
  printf("\nc = "); scanf("%f", &c);
  if(a+b>c)&&(a+c>b)&&(b+c>a)
    {
      p = a+b+c;  q = p/2;
      s = sqrt(q*(q-a)*(q-b)*(q-c));
      printf(" \n\nChu vi la %5.1f,\
               dien tich la %5.2f ",p,s);
     }
  else
      printf("\nBa so da cho la ba canh tam giac");
  getch();}
```

*Lưu ý: Cấu trúc **if** cũng như các cấu trúc khác của ngôn ngữ C, chúng có thể lồng nhau, tức là chúng có thể chứa các cấu trúc điều khiển khác.*

Ví dụ 2.3: Chương trình giải phương trình bậc 2: $ax^2 + bx + c = 0$, với các hệ số a, b, c là các số thực nhập từ bàn phím.

Giải: Để tìm nghiệm của phương trình bậc 2 một ẩn ta thực hiện theo các bước sau:

- Nhập a,b,c từ bàn phím
- Tính delta $= b^2 + 4*a*c$
- Nếu delta = 0 (phương trình có nghiệm kép x)
  - tính nghiệm x = -b/(2*a).
- Ngược lại
  - Nếu delta > 0 (phương trình có 2 nghiệm x1,x2)
    - x1 = (-b - sqrt(delta))/(2*a)
    - x1 = (-b + sqrt(delta))/(2*a)
  - Ngược lại (delta < 0)
    - phương trình không có thực

Chương trình giải phương trình bậc 2

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main(){
  float a,b,c, delta, x1,x2;
  printf("\nNhap cac he so a,b,c : ");
  scanf("%f%f%f",&a,&b,&c);
  delta = b*b-4*a*c;
  if(delta==0)
      printf("\nPhuong trinh co nghiem kep \
               x = %5.2f", -b/(2*a));
  else
      if(delta > 0)
      { x1 =(-b-sqrt(delta))/(2*a);
        x2 =(-b+sqrt(delta))/(2*a);
        printf("\nPhuong trinh cos 2 nghiem phan biet\
         \n x1= %5.2f \n x2= %5.2f",x1,x2);
```

```
        }
      else
        printf("\nPhuong trinh khong co nghiem thuc");
}
```

IV.3. Cấu trúc switch

Cấu trúc switch cho phép lựa chọn một nhánh trong nhiều khả năng tuỳ vào điều kiện xẩy ra. *Cú pháp như sau*

```
switch (<bt_E>)
  {
            case <bt_1>:
                        S1 ;
            case <bt_2>:
                        S2 ;
                  . . .
            case <bt_n>:
                        Sn ;
          [ default :   S  ;   ]
    }
```

Trong đó

- <bt_E> là biểu thức nguyên.
- <bt_1>, <bt_2>,..,<bt_n> là các biểu thức hằng, nguyên và chúng phải khác nhau.
- $S_1, S_2, .., S_n, S$ là một hoặc nhiều lệnh được gọi là thân của cấu trúc switch.
- case, default là các từ khoá

Sự hoạt động của switch

- Đầu tiên <bt_E> được tính, sau đó lần lượt so sánh giá trị của <bt_E> với các biểu thức hằng <bt_1>, <bt_2>,..<bt_n>.
- Nếu giá trị của một biểu thức hằng thứ k trong các biểu thức này trùng với giá trị của <bt_E> thì chương trình sẽ thực hiện các lệnh bắt đầu từ S_k và tiếp tục các lệnh phía dưới cho tới khi:
    - gặp câu lệnh : break (tất nhiên nếu gặp các câu lệnh return, exit thì cũng kết thúc)
    - gặp dấu đóng móc } hết cấu trúc switch
- Nếu <bt_E> không trùng với giá trị nào trong các biểu thức hằng thì câu lệnh S (các lệnh sau mệnh đề default nếu có) sẽ được thực hiện, rồi ra khỏi cấu trúc switch.

Ví dụ 3.1: Chương trình nhập một biểu thức đơn giản dạng a⊗b (a, b là các số nguyên, ⊗ là một trong các dấu phép toán số học +, -, *, x, /, :,) tính và in kết quả.

Giải : Ở đây chúng ta giả sử có thể dùng phép nhân là dấu * hoặc chữ x, phép chia có thể là dấu : hay /. Giả thiết người dùng nhập biểu thức đúng dạng a⊗b.

Chúng ta có chương trình như sau:

```
 1:  #include <stdio.h>
 2:  #include <conio.h>
 3:  void main()
 4:  {
 5:  int a,b;
 6:  char tt; // dấu toán tử
 7:  printf("\nnhap bieu thuc don gian :");
 8:  scanf("%d%c%d",&a,&tt,&b);
 9:  switch(tt)
10:  {
11:  case '+':   printf("\n %d %c %d = %d ",a,tt,b, a+b);
12:              break;
13:  case '-':   printf("\n %d %c %d = %d ",a,tt,b, a-b);
14:              break;
15:  case 'x':
16:  case '*':   printf("\n %d %c %d = %d ",a,tt,b, a*b);
17:              break;
18:  case ':':
19:  case '/': if(b!=0)
20:                 printf("\n %d %c %d = %d ",a,tt,b, a/b);
21:              else
22:                 printf("loi chia cho 0");
23:               break;
24:  default:  printf("\n\nkhong hieu phep toan %c",tt);
25:      }
26:  getch();
27:  }
```

Trong chương trình ví dụ này nếu bạn nhập biểu thức ví dụ như 9+2 tức là ta có a=9, b=2, tt (dấu toán tử) = '+'.
Như vậy mệnh đề case '+' (dòng 11) đúng, chương trình thực hiện câu lệnh

printf("\n %d %c %d = %d ",a,tt,b, a+b);

break;

và chương trình in ra kết quả

9 + 2 = 11

và thoát khỏi cấu trúc switch (nhảy tới dòng 26)

Nếu bạn nhập 9*2 hoặc 9x2, thì sự hoạt động cũng tương tự và kết quả in ra là
9 * 2 = 18
hoặc 9 x 2 = 18
Nếu bạn nhập không đúng phép toán hoặc không đúng khuôn dạng thì mệnh đề default được thực hiện và bạn sẽ nhận được thông báo 'khong hieu phep toan..'.
Lưu ý :

- Khi hết các lệnh (S_i) tương ứng của mệnh đề case <bt_i> nếu không có câu lệnh break (hoặc một lệnh kết thúc khác) thì điều khiển được chuyển tới lệnh S_{i+1} mà không cần kiểm tra biểu thức hằng <bt_i+1> có bằng <bt_E> hay không.
- Mệnh đề default có thể xuất hiện tại vị trí tuỳ ý trong thân của switch chứ không nhất thiết phải là cuối của cấu trúc (tất nhiên khi đó cần câu lệnh break).
- Cấu trúc switch có thể lồng nhau hoặc có thể chứa các cấu trúc điều khiển khác.

IV.4. Cấu trúc while

while là cấu trúc điều khiển lặp thực hiện một lệnh hay khối lệnh nào đó với số lần lặp được xác định tuỳ theo một điều kiện (gọi là điều kiện lặp).

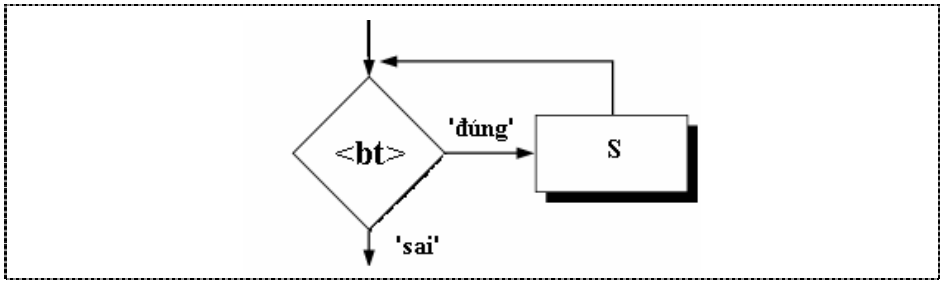
Cấu trúc của while như sau:

while (<bt>)
S;

Trong đó bt là một biểu thức nào đó là biểu thức điều kiện lặp, S là thân của while và chỉ là một câu lệnh.
Sự hoạt động của while như sau:

bước 1: tính giá trị của <bt>
bước 2: nếu giá trị tính được của <bt> là 'sai' (==0) thì kết thúc while
bước 3: nếu giá trị của <bt> là 'đúng' (!=0) thì thực hiện S
bước 4: quay lại bước 1

ta có sơ đồ điều khiển của while như sau



Ví dụ 4.1: Tìm ước số chung lớn nhất của 2 số nguyên x, y theo thuật toán sau:
a = x; b = y;
b1: nếu (a = b) thì ước số chung lớn nhất là a, kết thúc

ngược lại (a !=b) thì tới bước 2

b2:

- nếu a > b thì ta tính a = a- b
- ngược lại ta tính b =b – a
- quay lại bước 1

Như vậy chúng ta có thể phát biểu như sau

Chừng nào (a !=b) thì lặp lại

nếu a >b thì a = a-b

ngược lại b = b -a

kết thúc vòng lặp này thì a=b và là ước chung lớn nhất của x và y.

Đó chỉ xét trường hợp x, y là số nguyên >0. Trong trường hợp nếu một trong hai số bằng 0 thì ước số chung lớn nhất là trị tuyệt đối của số còn lại, nếu cả 2 số bằng 0 thì không xác định được ước số chung lớn nhất.

chúng ta có chương trình sau

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

void main() {
      int a,b,x,y;
      printf("nhap 2 so x, y : ");
      scanf("%d%d", &x,&y);
      a = abs(x); // a bằng trị tuyệt đối của x
      b = abs(y); // b bằng trị tuyệt đối của y
      if(a+b==0)
          printf("ca hai so bang 0");
      else
      if(a*b==0)
          printf("uoc so chung lon nhat la %d ", a+b);
      else
          {
         while(a!=b)
            if(a>b) a-=b;
               else b-=a;
         printf("uoc so chung lon nhat la %d", a);
       }
    getch();
  }
```

Ví dụ 4.2: Viết chương trình cho phép người sử dụng nhập một ký tự trên bàn phím, in kí tự và mã của nó ra màn hình, kết thúc chương trình khi người dùng bấm phím ESC (mã 27)

Giải: chúng ta có thể mô tả các bước của chương trình như sau

1: nhận 1 ký tự từ bàn phím, mã lưu trong biến ch

ch= getch();

2 : nếu ch ==ESC thì kết thúc, ngược lại chuyển sang bước 3

3 : in ký tự và mã của nó

printf("\nKy tu %c co ma %d", ch,ch)

4: quay lại 1

Vậy có chương trình

```
// In ki tu
#include <stdio.h>
#include <conio.h>
#include <math.h>
const int ESC =27; // ma phim ESC
void main(){
     int ch;
     while((ch=getch())!=ESC)
     printf("\nKi tu %c co ma %d",ch,ch);
}
```

Nhận xét:

- *while là cấu trúc điều khiển lặp với điều kiện trước, tức là điều kiện lặp được kiểm tra trước khi vào thân của vòng lặp, do vậy nếu biểu thức điều kiện có giá trị 'sai' ngay từ đầu thì thân của while có thể không được thực hiện lần nào.*
- *trong thân của while phải có lệnh để làm thay đổi giá trị của biểu thức điều kiện. Hoặc nếu biểu thức điều kiện luôn có giá trị 'đúng' - chúng ta gọi vòng lặp không kết thúc, thì trong thân của while phải có lệnh để chấm dứt vòng lặp (xem lệnh break).*

  ví dụ

```
while (1) // biểu thức điều kiện luôn đúng
  { printf("\n Hay bam mot phim: ");
    ch = getch();
```

```
        if(ch==27) break;
        printf("ky tu %c co ma la %d",ch,ch);
    }
```

Ví dụ 4.3: Viết chương trình nhập số nguyên n từ bàn phím, n <=10 và n >0, tính và in giá trị n! (giai thừa của n), với n! = 1*2*,..,*(n-1)*n

Giải:

Theo yêu cầu n nhập từ bàn phím phải thoả mãn n > 0 và n <=10, vì vậy nếu n người dùng nhập n không hợp lệ thì chương trình sẽ yêu cầu nhập lại, và sẽ lặp lại việc nhập cho tới khi thoả mãn.

Việc tính gt = n! theo các bước sau:

```
gt = 1 ; i = 1;
   while ( i <=n)
      {  gt = gt *i;
         i++;
      }
```

```
// giai thua
#include <stdio.h>
#include <conio.h>
const int Max =10; // giới hạn giá trị cần tính
void main(){
    int n,  i;
    long gt;
    printf("\nNhap n = ");
    scanf("%d",&n);
    while((n<=0)||(n>Max))
    {
       printf("\nNhap lai n (0<n<=%d) : ", Max);
       scanf("%d",&n);
    }
    gt=i=1;
    while(i<=n)
    {
        gt*=i;
      i++;
    }
    printf("\nGia tri %d! = %ld",n,gt);
    getch();
  }
```

Ví dụ 4.4: Viết chương trình tính và in S = 1 +1/(2!) +1/(3!)+..+1/(n!) với n nhập từ bàn phím, 0<n<10.

Giải: Bạn thấy là để tính S thì phải tính được các số hạng là 1/(k!) với k 1,2,..,n sau đó cộng vào tổng. Như vậy chúng ta có thể thực hiện như sau:

1: S=1; k=2;
2: nếu k > n thì thực hiện 4:
ngược lại thì thực hiện 3:
3: thực hiện các thao tác
    3.1: tính d = k!
    3.2: tính pt = 1/d
    3.3: S = S + pt
    3.4: k = k+1
    3.5: lặp lại 2:
4: in kết quả là S và kết thúc

Nhưng như vậy với mỗi số hạng (pt) của tổng chúng ta phải tính giai thừa một lần, trong khi đó chúng ta thấy rằng số hạng (pt) thứ k là 1/(k1!) = 1/(((k-1)!)*k) tức là bằng giá trị của pt thứ k-1 nhân với 1/k. Vì vậy ta có thể sửa đổi lại cách tính như sau:

1: S=1; pt=1; k = 2
2: nếu k > n thì thực hiện 4:
ngược lại thì thực hiện 3:
3: thực hiện các thao tác
    3.1: tính pt = pt * 1/k
    3.2: S = S + pt
    3.3: k = k+1
    3.4: lặp lại 2:
4: in kết quả là S và kết thúc

Chương trình là

```
#include <stdio.h>
#include <conio.h>
const int Max =10; // gioi han
void main(){
  int n, k ;  float S, pt;
  printf("\nNhap n = ");
  scanf("%d",&n);
  while((n<=0)||(n>Max)){
    printf("\nNhap lai n (0<n<=%d) : ", Max);
    scanf("%d",&n);
         }
```

```
        S=pt=1; k=2;
        while(k<=n){
             pt /=k;
             S+=pt;
               k++;
        }
        printf("\nGia tri tong S = %8.4f",S);
}
```

Ví dụ 4.5: Viết chương trình nhập số nguyên dương n từ bàn phím, hãy kiểm tra và thông báo ra màn hình số đó có là số nguyên tố hay không ?

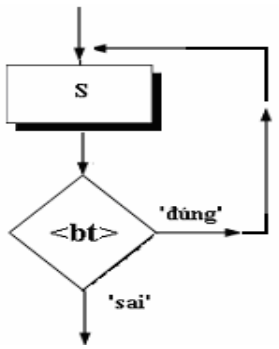
IV.5. **Cấu trúc do .. while**

Cấu trúc while mà chúng ta khảo sát ở trên luôn while kiểm tra điều kiện lặp trước khi thực hiện thân của nó. Tức là đòi hỏi trước khi vào cấu trúc while chúng ta phải xác lập được điều kiện lặp cho nó.

Có lẽ bạn cũng đồng ý rằng có những cấu trúc lặp mà thân của nó phải thực hiện ít nhất một lần, tức bước lặp đầu tiên được thực hiện mà không cần điều kiện gì, sau đó các bước lặp tiếp theo sẽ được xem xét tuỳ vào trạng thái của bước lặp trước đó. Với tình huống như thế thì while không thuận lợi bằng một cấu trúc điều khiển khác mà C cung cấp, đó là cấu trúc do while.

Cấu trúc của do while:

<table>
<tr>
<td>do
S;

while (<bt>);</td>
<td> 
</td>
</tr>
</table>

Trong cú pháp trên S là 1 câu lệnh là thân của vòng lặp, <bt> là biểu thức điều kiện có vai trò kiểm soát vòng lặp.

Sự hoạt động của cấu trúc do while

bước 1: thực hiện câu lệnh S

bước 2: kiểm tra giá trị biểu thức <bt>, nếu có giá trị 'đúng' (khác 0) thì lặp lại bước 1, nếu 'sai' (=0) thì kết thúc vòng lặp.

Ví dụ 5.1: Viết chương trình cho phép người sử dụng nhập một ký tự trên bàn phím, in kí tự và mã của nó ra màn hình, kết thúc chương trình khi người dùng bấm phím ESC (mã 27)

Giải: tương tự như ví dụ 4.2 nhưng ở đây chúng ta sử dụng cấu trúc do, vậy có chương trình

```
// In ki tu
#include <stdio.h>
#include <conio.h>
#include <math.h>
const int ESC =27; // ma phim ESC
void main(){
     int ch;
     do{
        printf("\n Hay nhap mot ki tu : ")
        ch = getch()
        printf("\nKi tu %c co ma %d",ch,ch);
      }while(ch!=ESC)
   }
```

Ví dụ 5.2: Chương trình tính tổng sin(x) theo công thức khai triển Taylor:

Các bạn biết rằng sin(x) được tính theo tổng chuỗi vô hạn đan dấu như sau:

$$\mathbf{S = x - x^3/3! + x^5/5! - x^7/7! + ...}$$

Nhưng rõ ràng chương trình không thể tính vô hạn được, chúng ta chỉ có thể tính với một giới hạn nào đó. Có hai cách để giới hạn một là theo số các số hạng trong tổng tức là chỉ tính tổng với n số hạng đầu tiên của chuỗi, cách này có ưu điểm là số bước lặp xác định, nhưng không ước lượng được sai số. Cách thứ hai chúng ta hạn chế số bước theo độ chính xác của kết quả. Chúng ta có thể phát biểu lại bài toán là: Tính sin(x) theo công thức khai triển trên với độ chính xác ε (epsilon) cho trước. Có nghĩa là kết quả tính được (S) có sai số so với giá trị thực của nó không quá ε, hay fabs(S-sin(x)) <= ε. Yêu cầu này có thể thoả mãn khi chúng ta tính tổng đến số hạng thức k nào đó mà giá trị tuyệt đối của phần tử này <= ε.

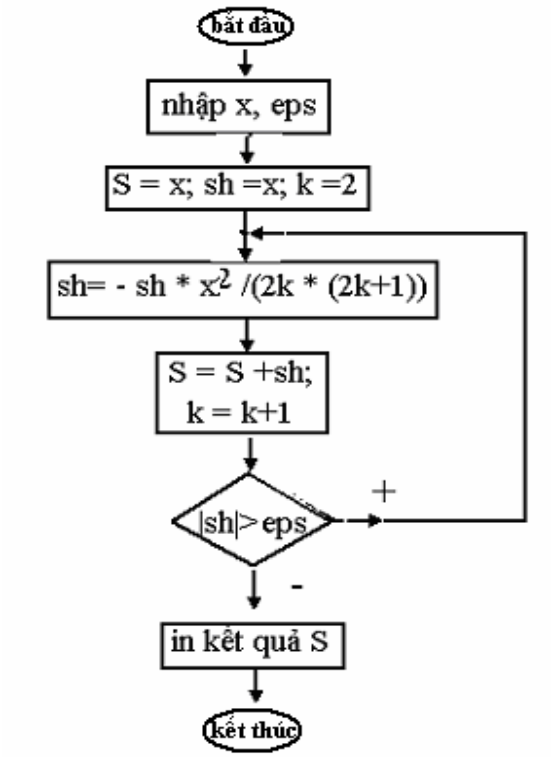
Cácbạn thấy rằng các phần tử trong chuỗi có tính chất đan dấu và giả sử ta ký hiệu phần tử thứ k trong chuỗi là p_k thì

$$p_k = (-1)^{2k} x^{2k+1} / (2k+1)!$$

và $$p_{k+1} = (-1)^{2k+1} x^{2(k+1)+1} / (2(k+1)+1)!$$

$$= -p_k * x^2 / (2k*(2k+1))$$

Chúng ta có sơ đồ khối thuật giải như sau:



sơ đồ khối tính S sin(x)

các bạn có chương trình tính sin(x)

```
#include <math.h>
#include <stdio.h>
void main(){
    float x, eps;
    float s, sh;
    int k;
    printf("\nNhap gia tri (radian) x  = ");
    scanf("%f",&x);
    printf("\nNhap sai so duoc phep eps = ");
    scanf("%f",&eps);
    s=x;sh=x; k=1;
    do {
         sh =-sh*x*x/(2*k*(2*k+1));
         s+=sh;
         k++;
      }  while(fabs(sh)>eps);
    printf("s= %f ",s);
}
```

Ví dụ 5.3: Viết chương trình nhập một số nguyên dương n từ bàn phím, kiểm tra và thông báo số đó có là số nguyên tố hay không.
Yêu cầu

- Chương trình chỉ kiểm tra số n >2
- Sau khi kiểm tra xong một số, chương trình hỏi người dùng có muốn kiểm tra tiếp hay không, nếu trả lời c(C) thì chương trình vẫn tiếp tục cho nhập và kiểm tra số tiếp, ngược lại sẽ kết thúc chương trình.

Giải :Để kiểm tra số n có là số nguyên tố hay không chúng ta cần kiểm tra các số từ 2 tới $\sqrt{n}$ xem có số nào là ước của n hay không, nếu không thì thông báo n là số nguyên tố, ngược lại thông báo n là hợp số.

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main(){
    int k, n, tl;
    do{
        printf("\nNhap gia tri can kiem tra  = ");
        scanf("%d",&n);
        if(n<2) printf("Khong kiem tra so <2");
        else { k=2;
              while((k<=sqrt(n))&&(n%k)) k++;
              if(k>sqrt(n))
                 printf("\n%d La so nguyen to",n);
              else
                 printf("\n%d Khong la so nguyen to",n);
              }
         printf("\nban co kiem tra so khac khong :");
         tl = getch();
      } while(toupper(tl)=='C');
    }
```

IV.6. **Cấu trúc for**

Đây cũng là toán tử điều khiển thực hiện lặp một số lệnh nào đó nhưng có cú pháp khác với hai chương trình lặp mà chúng ta đã xem xét ở phần trên, for trong C được dùng hết sức mềm dẻo nhưng nói chung nó sẽ trực quan và dễ hiểu hơn trong những tình huống lặp mà số bước lặp xác định.

Trước khi trình bày cú pháp tổng quát của for chúng ta hãy xem xét một đoạn lệnh (in các kí tự từ ‘a’ tới ‘z’ và mã của chúng) như sau:

for(i=’a’; i<’z’; i++)

printf(“ gia tri %d là mã của ki tu %c”, i,i);

đoạn lệnh trên có thể viết lại bằng chương trình while là

i =‘a’;

while(i<=’z’)

{ printf(“ gia tri %d là mã của ki tu %c”, i,i);

i++;

}

➢ ***Cú pháp***

for([bt_1]; [bt_2]; [bt_3])

S;

Trong đó **S** là một lệnh (đơn hoặc khối) được gọi là thân của vòng lặp, **bt_1, bt_2, bt_3** là các biểu thức hợp lệ, với ý nghĩa là:

- bt_1: biểu thức khởi đầu
- bt_2: biểu thức điều kiện - *điều kiện lặp*
- bt_3: bước nhảy - *thường dùng với ý nghĩa là thay đổi bước nhảy*

Cả 3 biểu thức này đều là tuỳ chọn, chúng có thể vắng mặt trong câu lệnh cụ thể nhưng các dấu chấm phẩy vẫn phải có.

➢ ***Hoạt động của for***

Hoạt động của for theo các bước sau:

b1: Thực hiện biểu thức khởi đầu - **bt_1**

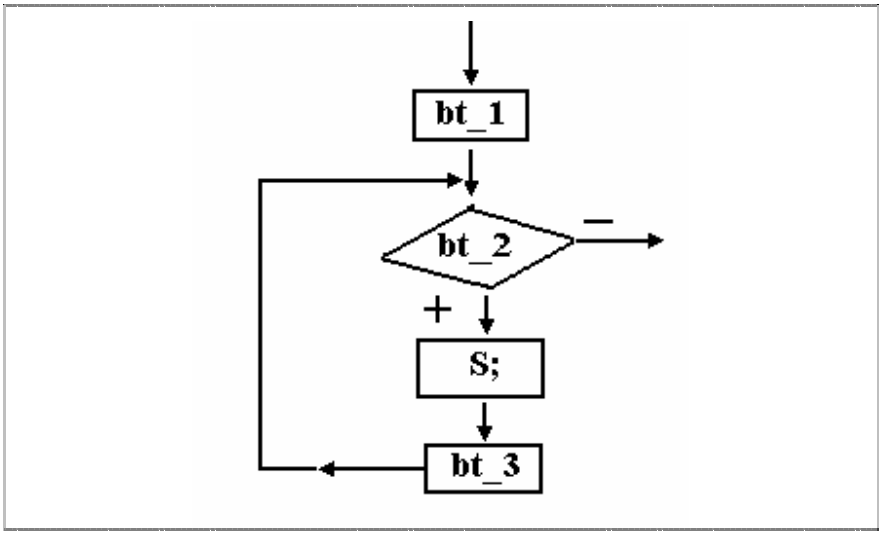
b2: Tính giá trị **bt_2** để xác định điều kiện lặp.

Nếu **bt_2** có giá trị 'sai' (==0) thì ra khỏi vòng lặp

Ngược lại, nếu **bt_2** có giá trị 'đúng' (khác 0) thì chuyển tới bước 3

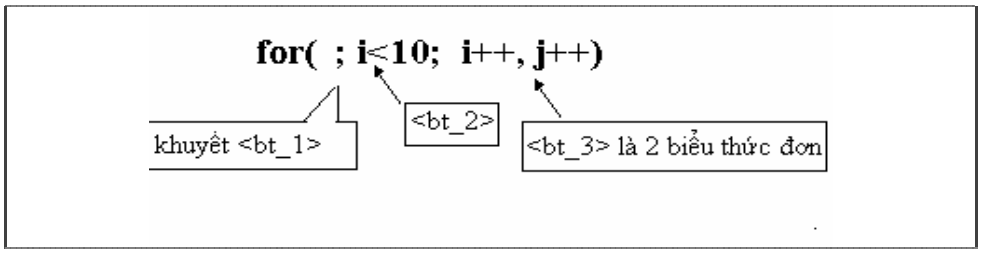
b3: Thực hiện lệnh **S (** thân của for **),** chuyển tới b4

b4: Thực hiện bt_3, rồi quay về b2.



sơ đồ cấu trúc for

Như trong cú pháp các bạn thấy các biểu thức (bt_1, bt_2, bt_3) của for có thể vắng, hơn nữa mỗi thành phần (biểu thức) lại có thể là một hoặc nhiều biểu thức(đơn) phân cách nhau bởi dấu phẩy (,) ví dụ như:



Nếu <bt_1>, <bt_3> vắng mặt thì đơn thuần đó là các lệnh rỗng (không thực hiện gì), nếu chúng có nhiều biểu thức đơn cách nhau bởi dấu phẩy thì các biểu thức đơn đó được thực hiện tuần tự từ trái qua phải - *thực ra vẫn có thể coi đây chỉ là một biểu thức, trong đó có các toán tử dấu phẩy (,) và trật tự tính toán theo độ ưu tiên của các toán tử trong biểu thức.*

Tương tự như bt_1, bt_3; biểu thức điều kiện trong trường hợp nó chỉ gồm một biểu thức đơn thì giá trị của nó quyết định vòng lặp có còn được tiếp tục hay không, nhưng nếu nó có nhiều biểu thức đơn ví dụ như:

```
for(i=0, j= i +2;  i<2,j <6;  i++,j++)
        printf("\ni = %d, j = %d);
```

thì kết quả sẽ là:

```
i=0, j=2
i=1, j=3
i=2, j=4
i=3, j=5
```

Như vậy vòng lặp dừng không theo điều kiện i <2 mà theo điều kiện j <6. Tức là giá trị của biểu thức bên phải nhất trong danh sách các biểu thức quyết định điều kiện của của vòng lặp – *điều này do toán tử (,) trả về toán hạng bên phải.*

Ví dụ 6.1: Chương trình in các kí tự có mã từ 32 tới 255 trong bảng mã ASCII

```
1: #include <stdio.h>
2: void main(void)
3:   {
4:      int i;
5:      for(i=32; i<256; i++)
6:       {
7:         if ((i - 31) %10==0) printf("\n");
8:         printf("%c -%d;",i,i);
9:        }
10:   getch();
11:   }
```

Giải thích: trong chương trình trên chúng ta sử dụng vòng lặp for

- biểu thức khởi đầu : i =32
- biểu thức điều kiện : i <256
- bước nhảy i ++

như vậy vòng lặp sẽ được thực hiện 224 lần, mỗi bước lặp nếu (i –31)%10 ==0 thì chúng ta chuyển con trỏ xuống dòng mới *(lệnh trên dòng 7)* – có nghĩa là cứ sau 10 bước thì chúng ta chuyển sang dòng mới, với mỗi i chúng ta in kí tự có mã là i cùng giá trị của nó *(lệnh printf trên dòng 8)*.

Ví dụ 6.2*:* chương trình nhập n số nguyên từ bàn phím, tìm và in số lớn nhất, nhỏ nhất.
Giải:
Giả sử chúng ta có một dãy số $a_1, a_2, .., a_n$ để xác định giá trị lớn nhất (gọi là max) và số nhỏ nhất (min)chúng ta thực hiện theo cách sau:

1. max = min = a_1 (xem một số đầu tiên là lớn nhất và cũng là nhỏ nhất)
2. i=2
3. nếu i > n thì thì kết thúc, ngược lại thì
   - nếu a_i > max thì max = a_i
   - ngược lại, nếu a_i < min thì min =a_i
   - i =i+1
4. lặp lại bước 3

Khi kết thúc chúng ta có giá trị lớn nhất là max, giá trị nhỏ nhất là min.
Nhưng cho tới bây giờ chúng ta chưa thể lưu được n số (trong yêu cầu này chúng ta cũng không cần phải lưu chúng) , vì thế chúng ta thực hiện theo phương pháp sau:

1: Nhập số thứ nhất từ bàn phím vào a
2: max = min = a (xem một số đầu tiên là lớn nhất và cũng là nhỏ nhất)
3: i=2
4: nếu i > n thì thì kết thúc, ngược lại thì
   - Nhập số thứ i từ bàn phím vào a
   - nếu a > max thì max = a

- ngược lại, nếu a < min thì min =a
- i =i+1

5: lặp lại bước 4

Các bạn có chương trình như sau

```
#include <stdio.h>
#include <conio.h>
void main(){
     int n,a,max,min,i;
     do{
         printf("Nhap so phan tu cua day : ");
         scanf("%d", &n);
     }while(n<1);
     printf("Nhap so thu nhat : ");
     scanf("%d",&a);
     max=min=a;
     for(i=2; i<=n; i++)
     {
        printf("Nhap so thu nhat : ");
        scanf("%d",&a);
        if(a>max) max=a;
        else
          if(a<min)
              min =a;
     }
     printf("\n gia tri lon nhat = %d\n \
              gia tri nho nhat = %d",max,min);
}
```

Ví dụ 6.3 : Viết chương trình giải bài toán sau:

"Trăm trâu trăm cỏ
Trâu đứng ăn năm
Trâm nằm ăn ba
Lụ khụ trâu già, ba con một cỏ
hỏi mỗi loại có mấy con "

Giải:

Rõ ràng là có ba loại trâu (phương trình ba ẩn) nhưng chỉ có hai phương trình đó là tổng số số trâu là 100 và tổng số cỏ cũng là 100. Chúng ta có

d + n + g =100 *(tổng số trâu)*

5*d + 3 * n + g/3 =100 *(tổng số cỏ)*

*(*d: số trâu đứng, n : số trâu nằm, g : số trâu già)

Như vậy bài toán này không có nghiệm duy nhất, và nếu có thì chỉ lấy nghiệm là các số nguyên mà thôi.

Ở đây chúng ta sử dụng cách kiểm tra các bộ số gồm 3 số nguyên dương (d,n,g) tương ứng với số trâu của từng loại với d,n,g ∈ [1,..100], nếu thoả mãn hai phương trình trên thì đó là một nghiệm. Vậy ta thực hiện như sau:

Với d = 1 tới 20 *// tối đa chỉ có 20 trâu đứng*
  thì thực hiện
  Với n = 1 tới 33 *// tối đa chỉ có 23 trâu nằm*
    thực hiện
    g = 100 – d – n ; *// số trâu già*
    nếu (g%3==0) và (5*d + 3 * n + g/3 ==100)
    thì in (d,n,g) là một nghiệm

```
#include <stdio.h>
#include <conio.h>
void main(){
int d,n,g;
clrscr();
printf("\nCac nghiem la\n");
   printf("\ntrau_dung  trau_nam  trau_gia\n");
for(d=1; d<=20;d++)
for(n=1; n<=33; n++)
  {
       g=100-d-n;
    if((g%3==0)&&(5*d+3*n+g/3==100))
    printf("%d  \t %d \t   %d\n",d,n,g);
  }
}
```

- **Chú ý :** Ngoài các cấu trúc điều khiển chúng ta vừa nêu trên, trong ngôn ngữ C còn một cấu trúc điều khiển khác nữa là goto. Đây là lệnh nhảy không điều kiện tới một vị trí nào đó trong chương trình, vị trí đó được xác định bằng một nhãn (label).
  Cú pháp

  goto <label>;

  trong đó **<label>** là tên một nhãn hợp lệ. Khi gặp lệnh này điều khiển sẽ được chuyển tới lệnh tại vị trí tại nhãn <label>
  Ví dụ:

  goto ketthuc; // với kết thúc là một nhãn

  Người ta chứng minh được là có thể dùng các cấu trúc điều khiển rẽ nhánh, lặp thay thế được goto, hơn nữa lạm dụng goto làm mất đi tính trong sáng và chặt chẽ của lập trình cấu trúc, do vậy trong giáo trình này chúng tôi không sử dụng goto.

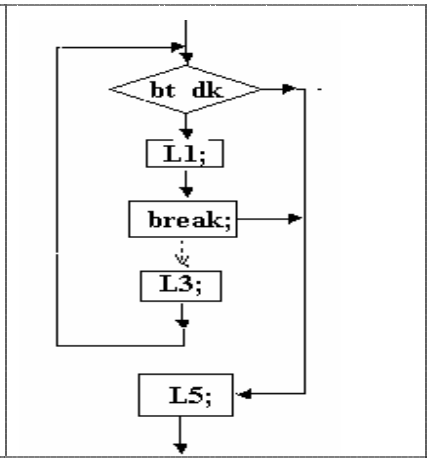
IV.7. Câu lệnh continue và break

Trong thân của for cũng như các cấu trúc lặp khác, có thể có câu lệnh continue và break, với chức năng là:

- **break** : kết thúc vòng lặp (trong cùng) chứa nó. break cho ta khả năng kết thúc một vòng lặp từ một vị trí bên trong của vòng lặp mà không cần đến giá trị của biểu thức điều kiện. Nếu trong nhiều cấu trúc lặp lồng nhau thì break chỉ có tác dụng kết thúc một cấu trúc lặp trong cùng chứa nó mà thôi.
- **continue**: Trái với break, continue là câu lệnh có chức năng chuyển chu trình về đầu bước lặp tiếp theo. Có nghĩa là sẽ bỏ qua các lệnh trong thân của vòng lặp kể từ lệnh sau continue cho tới hết thân của vòng lặp. Nếu có nhiều cấu trúc lặp bao nhau thì lệnh continue cũng chỉ có tác dụng với cấu trúc lặp trong cùng chứa nó.

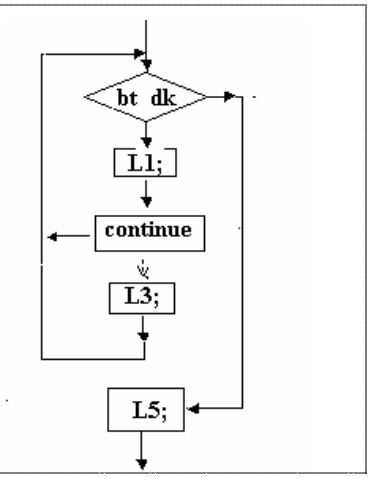
Ta có thể minh hoạ break và continue như sau:

```
.....
while (bt_dk)
     {
    L1;
    break;
    L3;

     }
  L5;
....
```



minh hoạ sự hoạt động của break

```
.....
while (bt_dk)
     {
    L1;
    continue;
    L3;

     }
  L5;
....
```



minh hoạ sự hoạt động của break

Chú ý: Trong for khi gặp continue thì các lệnh phía sau continue tới hết khối bị bỏ qua và chuyển tới thao tác thực hiện **bt_3** (bước nhảy) sau đó bắt đầu vòng lặp mới (kiểm tra điều kiện).

Ví dụ 6.4 : chương trình nhập số nguyên dương n từ bàn phím, tìm và in các ước của n và tổng các ước ra màn hình.

```
#include <stdio.h>
#include <conio.h>
void main(){
   int n,i, tonguoc=0;
   do{
       printf("Nhap so n : ");
       scanf("%d", &n);
   }while(n<2);
   printf("\nCac uoc cua %d la\n",n);
   for(i=1;i<n/2; i++)
    {  if(n%i)
          continue;
       printf("%d, ",i);
       tonguoc+=i;
    }
    printf("tong cac uoc la %d",tonguoc);
   }
```

Bài tập:

1: Nhập 2 số x, y, in bội số chung nhỏ nhất
2: Nhập tử số, mẫu số của một phân số, in phân số dạng tối giản
3: Giải phương trình bậc 2 có tính nghiệm phức
4: Tính sin(x), cos(x)
5: in ra các số nguyên tố 2..n
6: Kiểm tra 1 số có là số chính phương?
7: Kiểm tra 1 số có là số hoàn chỉnh?
8: Tìm giá trị lớn nhất, nhỏ nhất trong 1 dãy
9: Nhập một dãy số, hãy cho biết trật tự dãy đó
10: Nhập một số kiểm tra số đó có là số thuộc dãy fibonaxi hay không?
11: Nhập một số nin các số thuộc dãy fibonaxi <=n

V - Mảng và con trỏ

V.1. Khái niệm Mảng

Một biến (biến đơn) tại một thời điểm chỉ có thể biểu diễn được một giá trị. Vậy để có thể lưu trữ được một dãy các giá trị cùng kiểu chẳng hạn như các thành phần của vector trong không gian n chiều chúng ta cần n biến a_1, a_2,..,a_n. rất cồng kềnh và rất bất tiện nhất là khi n lớn và lại không phải là cố định. Các ngôn ngữ lập trình đưa ra một khái niệm mảng để giải quyết vấn đề này.

Mảng là một tập các phần tử cùng kiểu dữ liệu, các phần tử cùng tên phân biệt nhau bởi chỉ số. Từng phần tử của mảng có thể sử dụng như một biến đơn, kiểu của mảng chính là kiểu của các phần tử.

- *Các thông tin về mảng*: Với một mảng phải xác định các thông tin: tên mảng, kiểu các phần tử (kiểu mảng), số phần tử trong mảng (kích thước mảng). Ví dụ như chúng ta nói a là mảng có 20 phần tử, kiểu nguyên.

Mảng cũng như các biến đơn khác trong ngôn ngữ C, trước khi sử dụng nó phải đảm bảo là nó đã được cấp phát trong bộ nhớ và đã sẵn sàng để sử dụng

- *Số chiều của mảng*: trong ví dụ chúng ta nêu trên về vector, chúng ta có một dãy n các số, nếu như chúng ta dùng một mảng để lưu trữ các số đó thì chúng ta cần mảng có n phần tử và chỉ cần 1 chỉ số để xác định các phần tử - đây là mảng một chiều. Nếu như chúng ta cần một mảng để biểu diễn một bảng có n dòng, m cột, và để xác định một phần tử trong mảng chúng ta cần 2 chỉ số: chỉ số dòng và chỉ số cột, như vậy chúng ta có mảng 2 chiều. Một cách tương tự chúng ta cũng có thể có mảng 3 chiều, 4 chiều,.. hay nói cách ngắn gọn hơn: mảng một chiều là mảng có một chỉ số, mảng 2 chiều có 2 chỉ số,...Trong giáo trình này chúng ta cũng chỉ sử dụng đến mảng 2 chiều.

V.2. Mảng 1 chiều

V.2.1 - Định nghĩa mảng

Cú pháp

Kiểu_mảng tên_mảng [số_phần_tử];

Trong đó:

- **Kiểu_mảng:** đây là kiểu của mảng, là tên một kiểu dữ liệu đã tồn tại, có thể là kiểu chuẩn hoặc kiểu dữ liệu do người lập trình định nghĩa .
- **tên_mảng :** là tên của mảng, do người lập trình đặt, theo quy tắc về tên của C.

- **số_phần_tử :** là hằng (hoặc biểu thức hằng) nguyên, dương là số phần tử của mảng.

Ví dụ:

int vector [15]; // *tên mảng: vector, có 15 phần tử, kiểu int*

float MT[10], D[20]; // *có hai mảng kiểu float: MT có 10 phần tử, D có 20 phần tử*

char * s[30]; // s là mảng có 30 phần tử kiểu char * (mảng các con trỏ)

*Khi gặp (dòng lệnh) định nghĩa một mảng, chương trình dịch sẽ cấp phát một vùng nhớ (lên tiếp) cho đủ các phần tử liên tiếp của mảng, ví dụ vector[15] sẽ được cấp phát một vùng nhớ có kích thước 15*sizeof(int) =30 byte.*

V.2.2 - Truy xuất các phần tử

Cú pháp :

tên_mảng [chỉ_số]

ví dụ vector[1], MT[3], D[0];

***c*hỉ_số** là số thứ tự của phần tử trong mảng, các phần tử của mảng được đánh chỉ số bắt đầu từ 0. Với mảng có n phần tử thì các phần tử của nó có chỉ số là 0, 1,..,n-1.

ví dụ mảng vector có các phần tử vector[0], vector[1],...,vector[14]

Lưu ý: Các chương trình dịch của C không bắt lỗi khi người dùng truy xuất phần tử mảng vượt ra ngoài phạm vi của mảng, tức là có chỉ số nhỏ hơn 0 hoặc lớn hơn số_phần_tử-1.

V.2.3 - Khởi tạo giá trị các phần tử mảng một chiều

Các phần tử của mảng cũng như các biến đơn, chúng ta có thể khởi tạo giá trị ban đầu cho chúng trên dòng định nghĩa mảng (gọi là khởi đầu) với cú pháp sau:

Kiểu_mảng tên_mảng [số_phần_tử] = {gt_0, gt_1,..,gt_k};

hoặc

Kiểu_mảng tên_mảng [ ] = {gt_0, gt_1,..,gt_k};

Trong đó các thành phần **Kiểu_mảng , tên_mảng, số_phần_tử** như trong phần định nghĩa (V.1). **gt_0, gt_1,.., gt_k** là các giá trị khởi đầu (gọi là ***bộ khởi đầu***) cho các phần tử tương ứng của mảng, tức là gán tuần tự các giá trị trong bộ khởi đầu cho các phần tử của mảng từ trái qua phải.

Trong dạng thứ nhất, số giá trị trong bôn khởi đầu chỉ có thể <= số phần tử của mảng (k ≤ số_phần_tử). Khi đó những phần tử mảng thừa ra (không có giá trị khởi đầu) sẽ tự động được gán bằng 0 (trong trường hợp mảng số, nếu là con trỏ sẽ là NULL (rỗng)).

Ví dụ:

int a[3] ={ 1,3,4}; thì giá trị của a[0] là 1, a[1] là 3, a[2] là 4.

int b[5] ={1,2}; thì giá trị của b[0] là 1, b[1] là 2, b[3]=b[4] là 0.

với mảng các ký tự hoặc xâu ký tự thì có hai cách khởi đầu như sau

char c[4] ={‘a’,’b’,’c’ }; // c[0] là ‘a’, c[1] là ‘b’, c[2] là ‘c’, c[3] là ‘\0’

char s[10] =”ABC”; // tương đương với char s[10] ={‘A’,’B’,’C’,’\0’}

(*nếu số giá trị trong bộ khởi đầu > số phần tử mảng chương trình dịch sẽ báo lỗi)*

Trong dạng thứ hai, chúng ta không xác định số phần tử của mảng, trong trường hợp này chương trình biên dịch sẽ tự động xác định kích thước (số phần tử) của mảng theo số giá trị trong bộ khởi đầu.

Ví dụ:

int a[] ={1,3,4};

thì a là mảng có 3 phần tử, giá trị của a[0] là 1, a[1] là 3, a[2] là 4.

V.2.4 - Một số ví dụ

Ví dụ V.1: Chương trình nhập một mảng A có n phần tử kiểu nguyên , n<=20 nhập từ bàn phím, in các phần tử của mảng theo trật tự xuôi, ngược (theo thứ tự chỉ số).

Giải: Trong chương trình chúng ta cần định nghĩa một mảng A có 20 phần tử kiểu int. Một biến nguyên n là số phần tử thực sự của A sẽ được nhập. Số nguyên n được nhập từ bàn phím phải thoả mãn 1<=n<=20.

Các phần tử A[i] (i=0,1,.,n-1) của A được nhập từ bàn phím tương tự như các biến đơn bằng câu lệnh (hàm) scanf: scanf(“%d”,&A[i]).

Và được in ra bằng hàm printf, như sau printf (“%d ”, A[i]), nếu ta sử dụng vòng lặp với i từ 0 tới n-1 ta được các phần tử theo trật tự xuôi của chỉ số:

```
for(i=0; i<n; i++)
    printf("%d, ",A[i]);
```

ngược lại các bạn cho chạy theo thứ tự từ n-1 tới 0 chúng sẽ có các phần tử theo số thứ tự ngược

```
for(i= n-1; i>=0; i--)
     printf("%d, ",A[i]);
```

Chương trình minh hoạ như sau:

```
#include <stdio.h>
#include <conio.h>
void main(){
    clrscr();
    const int max =20;
    int A[max];
    int n,i;
    do{
          printf("\nNhap so phan tu mang  = ");
          scanf("%d",&n);
      }while(n<1 || n>max); //nhập số pt mảng 1<=n<=max
```

```
  printf("\nNhap mang co %d phan tu \n",n);
  for(i=0; i<n; i++)
   {
        printf("A[%d]= ",i);
        scanf("%d",&A[i]);
   }
  printf("\nCac phan tu mang theo thu tu xuoi la  \n");
  for(i=0; i<n; i++)
        printf("%d, ",A[i]);

  printf("\nCac phan tu mang theo thu tu nguoc la  \n");
  for(i=n-1; i>=0; i--)
        printf("%d, ",A[i]);
  getch();
}
```

(Ví dụ V.1: *chương trình nhập và in mảng)*

Ví dụ V.2: Viết chương trình nhập 2 mảng A, B có n phần tử (n<=10) các số nguyên, tính và in mảng C = A+B.

Giải: Việc nhập 2 mảng A, B cũng tương tự như trong ví dụ trước. Mảng C là tổng của A và B tức là các phần tử C[i] = A[i]+B[i] (i =0,1,.., n-1).

```
#include <stdio.h>
#include <conio.h>
void main(){
    clrscr();
    const int max = 10;
    int A[max], B[max], C[max];
    int n,i;
    do{
        printf("\nNhap so phan tu mang  = ");
        scanf("%d",&n);
    }while(n<1 || n>max); //nhập số pt mảng 1<=n<=max
    printf("\nNhap mang A co %d phan tu \n",n);
    for(i=0; i<n; i++)
      {
        printf("A[%d]= ",i);
        scanf("%d",&A[i]);
      }
```

```
     printf("\nNhap mang B co %d phan tu \n",n);
     for(i=0; i<n; i++)
      {
           printf("B[%d]= ",i);
           scanf("%d",&A[i]);
      }
     // Tính C=A+B
     for(i=0; i<n; i++)
           C[i]= A[i]+B[i];
     printf("\nCac phan tu mang ket qua C la \n");
     for(i = 0; i < n; i++)
           printf("%d, ",C[i]);
     getch();
  }
```

(Ví dụ V.2 - Tính tổng 2 mảng)

V.2.3 - Sắp xếp và tìm kiếm trên mảng một chiều

Trong thực tế chúng ta rất hay gặp yêu cầu phải sắp xếp một dãy các phần tử theo một trật tự nào đó, hoặc là tìm kiếm một phần tử có trong một dãy các phần tử hay không. Một cách thuận lợi nhất (nếu có thể) đó là biểu diễn các phần tử đó là một mảng các phần tử.

➢ ***Sắp xếp mảng:*** Bài toán sắp xếp mảng nói chung được phát biểu như sau: Cho một mảng có n phần tử, và k là khoá của mỗi phần tử, các phần tử có thể so sánh với nhau theo khoá k, hãy sắp xếp mảng theo thứ tự tăng (hoặc giảm) của khoá k.

Khoá k chúng ta có thể hiểu là một thành phần nào đó của các phần tử như là tuổi của một người, hay điểm trung bình học tập của một sinh viên, hoặc là một tiêu chí nào đó áp dụng cho các phần tử của mảng.

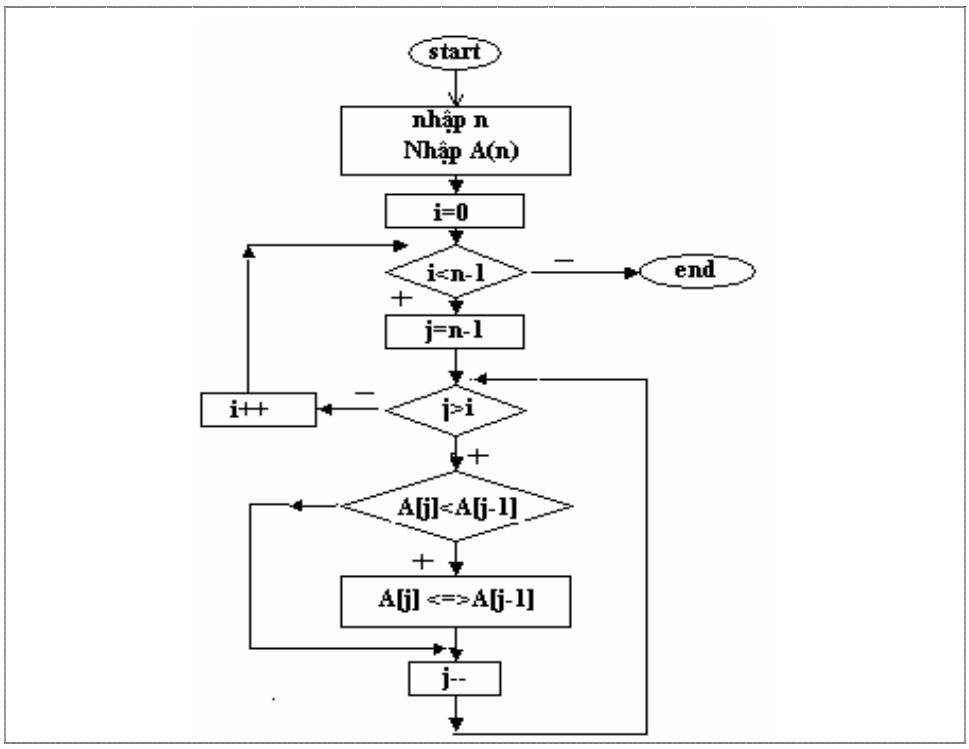
Trong trường hợp đơn giản như các mảng số mà chúng ta sẽ nói trong ví dụ sau đây thì khoá k chính là giá trị của các phần tử.

Hiện nay có nhiều thuật toán để sắp xếp một mảng: thuật toán nối bọt, thuật toán đổi chỗ, thuật toán chọn, thuật toán chia đôi,.. trong giáo trình này chúng tôi giới thiệu ba thuật toán sắp xếp đơn giản để sắp một mảng A có n phần tử kiểu số nguyên.

a. Sắp xếp bằng phương pháp nổi bọt

Ý tưởng của phương pháp này là có n phần tử ("bọt nước") đều có xu hướng nổi lên trên mặt nước, thì phần tử nào nhỏ hơn ('nhẹ hơn') sẽ được ưu tiên nổi lên trên. Tức là với mọi cặp phần tử **kề nhau** nếu phần tử sau (dưới) nhỏ hơn phần tử phía trước thì phần tử nhỏ hơn sẽ nổi lên trên, phần tử nặng hơn sẽ chìm xuống dưới.

Sơ đồ thuật toán sắp xếp mảng A(n) như sau:



(sắp xếp bằng phương pháp nổi bọt)

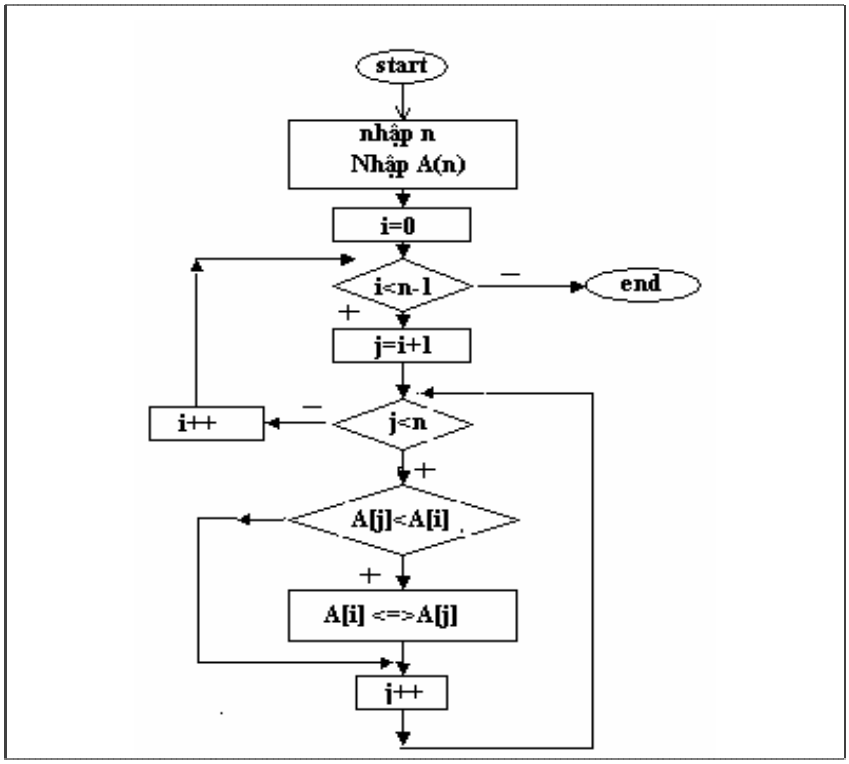
b. Sắp xếp bằng phương pháp đổi chỗ trực tiếp

Ý tưởng của phương pháp này cũng rất đơn giản là: giả sử các phần tử đầu mảng A[0], A[1],.., A[i-1] đã được sắp đúng vị trí tức là đã có:

A[0] <=A[1] <=,..<A[i-1] <= min{A[i],A[i+1],..,A[n-1]} (i=0,1,..)

Công việc tiếp theo là sắp các phần tử còn lại vào đúng vị trí của nó. Các bạn thấy vị trí thứ i là vị trí đầu tiên chưa được sắp, nếu được sắp thì A[i] phải có giá trị nhỏ nhất trong các phần tử còn lại đó {A[i],A[i+1],..,A[n-1]}, vậy chúng ta sẽ duyệt các phần tử mảng trong phần còn lại A[j] với j =i+1 tới n-1, nếu A[j] < A[i] thì chúng ta đổi chỗ A[i] với A[j]. Như vậy phần tử i đã được xếp đúng vị trí.

Vậy chúng ta thực hiện lặp công việc trên với i từ 0 tới n-2 chúng ta sẽ có mảng được sắp.



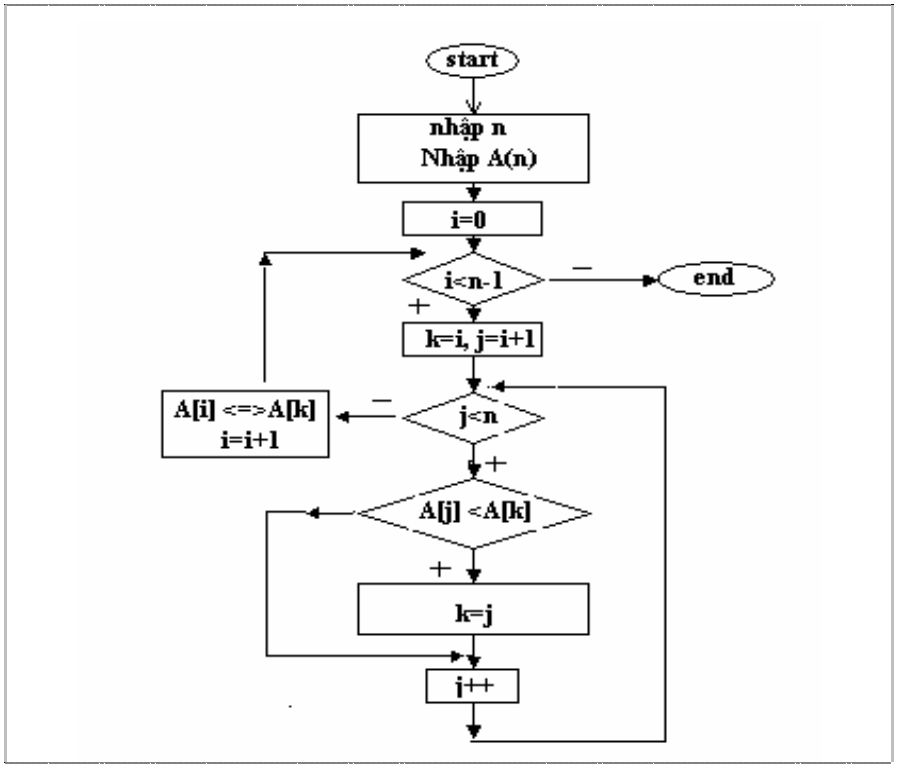
(sắp xếp bằng phương pháp đổi chỗ trực tiếp)

c. Sắp xếp bằng phương pháp chọn

Các bạn có nhận xét là trong phương pháp đổi chỗ trực tiếp để đặt được phần tử vào vị trí i có thể phải sử dụng (n-1) phép đổi chỗ. Trong khi đó chỉ có một phần tử sẽ đặt tại đó.

Phương pháp chọn cũng xuất phát từ ý tưởng như phương pháp đổi chỗ trực tiếp nhưng thay vì đổi chỗ A[i] với A[j] trong mỗi bước duyệt (theo j) thì chúng ta xác định phần tử nhỏ nhất trong các phần tử A[i+1],..A[n-1] giả sử là A[k], sau đó dổi chỗA[k] và A[i]. Như vậy với mỗi vị trí i chương trình chỉ thực hiện đổi chỗ một lần, và người ta tính thời gian thực hiện trung bình của phương pháp này ít hơn thời gian trung bình của hai phương pháp trên.

Các bạn có sơ đồ khối như sau



(*sắp xếp bằng phương pháp chọn)*

Ví dụ V.3: chương trình minh hoạ sắp xếp mảng bằng phương pháp nổi bọt

```
#include <stdio.h>
#include <conio.h>
void main(){
   const max=10;
   int n,a[max], i,j,tg;
   do{
          printf("Nhap so n : ");
          scanf("%d", &n);
      }while(n<2);
   printf("\nNhap mang co %d phan tu \n",n);
   for(i=0;i<n; i++)
      {   printf("a[%d]= ",i);
          scanf("%d",&a[i]);
       }
    for(i = 0; i<n-1; i++)
    for(j =n-1; j>i;j--)
    if(a[j]<a[j-1])
       {  tg=a[j]; a[j]=a[j-1]; a[j-1]=tg;}
    printf("Mang sau khi sap la \n");
    for(i=0;i<n; i++)
       printf("%d, ",a[i]);
}
```

➢ ***Tìm kiếm trên mảng***

Giả sử cho trước một mảng các số nguyên A(n), và một số x. hãy kiểm tra x có thuộc mảng A hay không?

Với bài toán tìm kiếm trên mảng nói chung, chúng ta phân hai trường hợp:

- *Trường hợp 1:* Mảng A không có trật tự (chưa được sắp) thì để tìm kiếm một giá trị nào đó thì chúng ta phải duyệt tuần tự mảng từ phần tử đầu tiên cho tới khi gặp giá trị đó hoặc tới phần tử cuối cùng thì mới khẳng định được giái trị đó có thuộc mảng hay không. Như vậy trong trường hợp kém nhất thì số lần so sánh là n.

có thể minh hoạ như sau:

```
// Nhập n, A(n), x
   i = 0 ;
   while((i <n)&&(A[i] !=x))  i++;
   if(i >n-1) printf("%d khong co trong mang",x );
   else  printf("%d  co trong mang tai vi tri  %d", x, i);
```

- *Trường hợp 2:* Mảng A đã được sắp (không mất tổng quát giả sử tăng dần), trong trường hợp này chúng ta có thể áp dụng phương pháp tìm kiếm nhị phân để giảm số bước phải so sánh. Ý tưởng là ta chia đôi mảng A thành hai phần, so sánh x với phần tử ở giữa của mảng A[g] xảy ra ba trường hợp :
  - A[g] = = x thì kết luận x thuộc vào A và kết thúc
  - A[g] > x thì chúng ta lặp lại việc tìm x trong nửa cuối của mảng
  - A[g] < x thì chúng ta lặp lại việc tìm x trong nửa đầu của mảng

Như vậy sau một bước so sánh chúng ta có thể giảm số phần tử cần duyệt còn một nửa. Như vậy số lần kiểm tra so sánh trung bình sẽ giảm so với phương pháp duyệt tuần tự.

có thể minh hoạ như sau:

```
// Nhập n, A(n), x
// Mảng A theo thứ tụ tăng dần
   l = 0, r =n-1 ;        // l, r chỉ số đầu, cuối của các phần tử cần duyệt
   while(l <= r)
      {  g =  (l+r)/2;        // lấy phần tử giữa
         if (A[g] ==x) printf(" %d thuộc vào mảng "); return;
         if (A[g] > x)
               l = g+1 ; // lặp timg trong nửa cuối
```

```
        else
                r = g-1;  // tim trong nửa đầu.
      }
    printf("%d khong co trong mang",x );
   //.....
```

Ví dụ V.4: Chương trình sinh ngẫu nhiên một mảng có n phần tử, sắp xếp mảng đó theo thứ tự tăng bằng phương pháp chọn, Nhập x từ bàn phím kiểm tra x có trong mảng hay không

V.3 - Mảng 2 chiều

V.3.1 - Định nghĩa mảng hai chiều

Mảng hai chiều có thể hiểu như bảng gồm các dòng các cột, các phần tử thuộc cùng một kiểu dữ liệu nào đó. Mảng hai chiều được định nghĩa như sau.

Cú pháp

Kiểu_mảng tên_mảng [sd][sc];

Trong đó:

- **Kiểu_mảng:** đây là kiểu của mảng, là tên một kiểu dữ liệu đã tồn tại, có thể là kiểu chuẩn hoặc kiểu dữ liệu do người lập trình định nghĩa.
- **tên_mảng :** là tên của mảng, do người lập trình đặt, theo quy tắc về tên của C.
- **sd, sc :** là hằng (hoặc biểu thức hằng) nguyên, dương tương ứng là số dòng và số cột mảng, số phần tử của mảng sẽ là sd*sc.

Ví dụ:

```
int  a[2][5];       // a là mảng số nguyên có 2 dòng, 5 cột (có 10 phần tử)
float D[3][10];   // D là mảng số thực có 3 dòng, 10 cột (có 30 phần tử)
char DS[5][30];  // DS là mảng kí tự có 5 dòng, 30 cột
```

Khi gặp một định nghĩa mảng, chương trình dịch sẽ cấp phát một vùng nhớ liên tiếp có kích thước là **sd*sc*sizeof (Kiểu_mảng)** *cho mảng.*

Có thể coi mảng 2 chiều n dòng, m cột là mảng 1 chiều có n phần tử, mỗi phần tử lại là 1 mảng một chiều có m phần tử (mảng của mảng). Ví dụ với **float D[3][10]** *có thể xem D là mảng có 3 phần tử D[0], D[1], D[2], mỗi phần tử này là mảng có 10 phần tử.*

V.3.2 – Truy xuất các phần tử mảng hai chiều

Một phần tử của mảng 2 chiều được xác định qua tên (tên của mảng) và chỉ số dòng, chỉ số cột của nó trong mảng theo cú pháp sau:

tên_mảng [csd][csc]

Với **csd** là số nguyên xác định chỉ số dòng và **csc** là số hiệu cột cũng như trong mảng 1 chiều các chỉ số được tính từ 0. Tức là **0 ≤ csd ≤ sd-1** và **0 ≤ csc ≤ sc-1.**

Lưu ý: Các phần tử của mảng 2 chiều cũng được dùng như các biến đơn, trừ trường hợp khi nhập giá trị cho các phần tử mảng kiểu float bằng hàm scanf thì bạn nên sử dụng biến (đơn) trung gian, sau đó gán giá trị của biến đó vào phần tử mảng chứ không nên sử dụng toán tử & để nhập trực tiếp phần tử của mảng.

V.3.3 – Khởi đầu giá trị các phần tử mảng hai chiều

Các phần tử mảng hai chiều cũng có thể được khởi đầu giá trị theo cú pháp (4 dạng sau):

1. **Kiểu_mảng tên_mảng [sd][sc] = {{kđ_dòng_1},{ kđ_dòng_2},..,{ kđ_dòng_k}};**
2. **Kiểu_mảng tên_mảng [ ][sc] = {{kđ_dòng_1},{ kđ_dòng_2},..,{ kđ_dòng_k}};**
3. **Kiểu_mảng tên_mảng [sd][sc] = { gt_1, gt_2,...,gt_n };**
4. **Kiểu_mảng tên_mảng [ ][sc] = { gt_1, gt_2,...,gt_n };**

Cú pháp trên có thể giải thích như sau:

- **dạng 1**: có k bộ giá trị sẽ được gán cho k dòng đầu tiên của mảng (**k ≤ sd**), với mỗi dòng (được coi như mảng một chiều) được khởi tạo giá trị như mảng một chiều:

  dòng thứ nhất được khởi đầu bởi {kđ_dòng_1}, dòng thứ hai được khởi đầu bởi {kđ_dòng_1},.., dòng thứ k được khởi đầu bởi {kđ_dòng_k}. Yêu cầu k ≤ sd, ngược lại chương trình sẽ báo lỗi.

  Các dòng cuối của mảng nếu không có bộ khởi đầu tương ứng thì sẽ được tự động gán giá trị 0 (hoặc NULL nếu là con trỏ).

- **dạng 2: (**không xác định số dòng) chương trình dịch sẽ tự động ấn định số dòng của mảng bằng số bộ khởi đầu (= k), sau đó thực hiện khởi đầu như dạng 1.

- **dạng 3: n** giá trị trong bộ khởi đầu được gán cho các phần tử mảng theo cách: **sc** giá trị đầu tiên trong các giá trị khởi đầu (gt_1,..,gt_sc) được gán tuần tự cho các phần tử của dòng thứ nhất trong mảng, **sc** phần tử kế tiếp sẽ gán cho các phần tử ở dòng thứ 2,... nếu phần tử nào của mảng không có giá trị khởi đầu sẽ được gán 0 (con trỏ là NULL) - với điều kiện **n ≤ sd*sc,** ngược lại là lỗi**.**

- **dạng 4**: số dòng của mảng sẽ được chương trình tự tính theo số giá trị trong bộ khởi đầu theo công thức sd = (**n/sc) +1,** và khởi đầu như dạng 3.

  Ví dụ:

- int a[3][2] = {{1,2},{3},{4,5}}; thì các phần tử của a như sau:
  a[0][0]=1, a[0][1]=2, a[1][0]=3, a[1][1]= 0, a[2][0]=4,a[2][1]=5;
- int b[ ][2] = {{1,2},{3},{4,5}};
  thì là mảng 3 dòng, 2 cột các phần tử của a như sau:
  b[0][0]=1, b[0][1]=2, b[1][0]=3,b[1][1]= 0, b[2][0]=4,b[2][1]=5;
- int c[ ][2] = {1,2,3,4,5};
  thì số dòng của c là mảng 5/2 +1 =3 dòng, các phần tử của a như sau:
  c[0][0]=1, c[0][1]=2, c[1][0]=3,c[1][1]= 4, b[2][0]=5,b[2][1]=0;

V.3.3 - Một số ví dụ về mảng hai chiều

*Ví dụ V.5: C*hương trình nhập mảng A(n,m), 1≤ n,m ≤ 5, các số nguyên từ bàn phím, in mảng ra màn hình theo yêu cầu các phần tử cùng một hàng được in trên một dòng của *màn hình*, các phần tử cách nhau một dấu trống.

```
#include <stdio.h>
#include <conio.h>
void main(){
    clrscr(); //xóa màn hình
    const int max =5; // kích thước tối đa
    int A[max][max];
    int n,m,i,j;
    do{printf("\nNhap so dong cua mang  = ");
        scanf("%d",&n);
        printf("\nNhap so cot cua mang  = ");
        scanf("%d",&m);
      } while(n<1 || n>max|| m<1 || m>max);
    printf("\nNhap mang co %d dong, %d cot \n",n,m);
    for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      {
        printf("A[%d][%d]= ",i,j);
        scanf("%d",&A[i][j]);
      }
    printf("\nCac phan tu mang la  \n");
    for(i=0; i<n; i++)
     {  printf("\n");
        for(j=0; j<m; j++)
        printf("%d ",A[i][j]);
       }
    getch();}
```

*Ví dụ V.6: C*hương trình nhập 2 ma trận A(n,m), B(n,m), 1≤ n,m ≤ 5, các số thực từ bàn phím, tính in ra màn hình ma trận C = A+B.

Giải: Trước khi viết chương trình chúng ta lưu ý đến mấy vấn đề:

- C = A+B có nghĩa là các phần của C được tính C[i][j] = A[i][j] + B[i][j]
- chỉ có thể cộng hai ma trận A,B cùng kích thước và C cũng cùng kích thước với A,B
- do các phần tử mảng có kiểu là float vì vậy khi nhập ta ***nên*** dùng biến phụ.

```
#include <stdio.h>
#include <conio.h>
void main(){
    clrscr();
    const int max =5; //số dòng, cột tối đa
    float A[max][max],B[max][max],C[max][max];
    int n,m,i,j;
    float x;
    do{
        printf("\nNhap so dong cua ma tran  = ");
        scanf("%d",&n);
        printf("\nNhap so cot cua ma tran  = ");
        scanf("%d",&m);
      } while(n<1 || n>max|| m<1 || m>max);
    printf("\nNhap A co %d dong, %d cot \n",n,m);
    for(i=0; i<n; i++)
    for(j=0; j<m; j++)
      {
        printf("A[%d][%d]= ",i,j);
        scanf("%f",&x);A[i][j]=x;
      }
    printf("\nNhap B co %d dong, %d cot \n",n,m);
    for(i=0; i<n; i++)
    for(j=0; j<m; j++)
       {
          printf("B[%d][%d]= ",i,j);
          scanf("%f",&x);B[i][j]=x;
        }
    for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        C[i][j]=A[i][j]+B[i][j];
```

```
printf("\nCac phan tu ma tran C la  \n");
for(i=0; i<n; i++)
  { printf("\n");
    for(j=0; j<m; j++)
    printf("%4.1f ",C[i][j]);
   }
getch();
}
```

(ví dụ V.6 - chương trình tính tổng 2 ma trận)

Ví dụ V.7: Chương trình nhập ma trận A(n,n), 1≤ n,n ≤ 5, các số nguyên từ bàn phím, sau đó kiểm tra và thông báo ma trận đó có đối xứng hay không .

Giải: Một ma trận A là đối xứng trước hết nó phải là ma trận vuông và các phần tử của nó đối xứng nhau qua đường chéo chính, tức là A[i][j] =A[j][i]. Vậy chúng ta kiểm tra một ma trận đối xứng theo cách sau:

Với mỗi i,j (0≤ i,j ≤n-1) nếu tồn tại i,j mà A[i][j] != A[j][i] thì ta kết luận A không đối xứng, ngược lại A là đối xứng.

Tất nhiên chúng ta không cần phải xét mọi cặp (i,j) có thể, vì nếu như vậy thì:

- cặp (A[i][j] A[j][i]), và (A[j][i], A[i][j]) thực chất là một nhưng lại hai lần so sánh
- phần tử trên đường chéo chính A[i][i] được so sánh với chính nó.

Vì thế chúng ta chỉ xét các chỉ số (i,j) mà phần tử A[i][j] nằm thực sự phía trên của đường chéo chính. tức là chỉ cần xét các cặp phần tử (A[i][j], A[j][i]) với i chạy từ 0 tới n-1 và với j chạy từ i+1 tới n-1 là đủ. Hơn nữa chúng ta chỉ duyệt các cặp (A[i][j], A[j][i]) nếu chưa phát hiện được cặp nào khác nhau.

Vậy ta có thể mô tả như sau:

```
d=0; // d là biến để đánh dấu ghi nhận có gặp một cặp (A[i][j]!= A[j][i])  thì d=1
for(i=0; (i<n) && (d= =0); i++)
for(j= i+1; (j<n) && (d= =0); j++)
   if(A[i][j]!=A[j][i]) d=1;
```

Kết thúc đoạn lệnh lặp trên chỉ có hai khả năng

- nếu d=1 tức là có cặp (A[i][j]!=A[j][i]) tức là ma trận không đối xứng.
- ngược lại,(d=0) thì ma trận là đối xứng.

```
#include <stdio.h>
#include <conio.h>
void main(){
   clrscr();
   const int max =5; //
   int A[max][max];
   int n,d,i,j;
   do{
        printf("\nNhap so dong, so cot cua ma tran = ");
        scanf("%d",&n);
     }  while(n<1 || n>max);
   printf("\nNhap ma tran vuong cap %d \n",n,n);
   for(i=0; i<n; i++)
   for(j=0; j<n; j++)
      {
        printf("A[%d][%d]= ",i,j);
        scanf("%d",&A[i][j]);
     }
   for(i=0,d=0; (i<n)&&(d==0); i++)
   for(j=0; (j<n)&&(d==0); j++)
   if(A[i][j]!=A[j][i]) d=1;
   if(d) printf("\nMa tran khong doi xung");
   else
        printf("\nMa tran doi xung");
   getch();
}
```

(Ví dụ V.6 - kiểm tra ma trận đối xứng)

V.4 - Con trỏ và mảng

Trong phần này chúng xem xét kỹ hơn về các tổ chức của mảng trong bộ nhớ; liên hệ giữa mảng, các phần tử của mảng với con trỏ, các phép toán trên con trỏ. Tuy nhiên con trỏ là một kiểu quan trong của C. Trong phần này chúng tôi chưa đề cập tới hết tất cả các khía cạnh của con trỏ như cấp phát động, tryền tham số hàm là con trỏ, danh sách liên kết. Các nội dung này sẽ được giới thiệu trong chuyên đề kỹ hơn về C.

V.4.1 - Con trỏ và các phép toán trên con trỏ

Trong phần đầu trình bày về kiểu dữ liệu và các phép toán chúng ta cũng đã đề cập tới kiểu con trỏ, trong phần này chúng ta dề cập chi tiết hơn về con trỏ và các phép toán có thể sử dụng trên chúng.

Con trỏ là kiểu dữ liệu mà một thành phần kiểu này có thể lưu trữ địa chỉ của một thành phần nào đó (có thể là biến, hằng, hàm), hoặc ta nói nó trỏ tới thành phần đó.

Một con trỏ lưu trữ địa chỉ của một thành kiểu T thì ta nói p là con trỏ kiểu T, đặc biệt nếu T là một kiểu con trỏ, hay nói cách khác, p lưu trữ địa chỉ của một con trỏ khác thì ta nói p là con trỏ trỏ tới con trỏ.

Cú pháp khai báo con trỏ

<kiểu> * <tên_con_trỏ>;

Ví dụ:

```
int *p;  // p là con trỏ kiểu int
float * q ;  // q là con trỏ kiểu float
char *s ;   // s là con trỏ kiểu char hay xâu kí tự
int ** r; // r là con trỏ tới con trỏ kiểu int
```

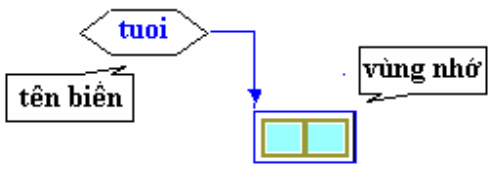
Cũng giống như biến bình thường khi khai báo một biến con trỏ, chương trình dịch cũng cấp phát vùng nhớ cho biến đó, ***lưu ý rằng*** giá trị trong vùng nhớ đó đang là bao nhiêu thì quan niệm đó là địa chỉ mà con trỏ này trỏ tới. Vì vậy các bạn phải chú ý khi dùng con trỏ phải bảo đảm nó trỏ tới đúng vùng nhớ cần thiết.

Một con trỏ chưa lưu trữ địa chỉ của thành phần nào ta gọi là con trỏ rỗng và có giá trị là **NULL** (*là một hằng định nghĩa sẵn thực ra = 0*).

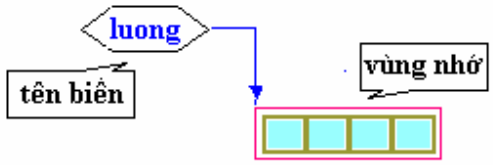
Khi gặp các lệnh khai báo biến trong chương trình thì chương trình dịch sẽ cấp phát vùng nhớ phù hợp và '***gắn***' tên biến với vùng nhó đó.

Ví dụ:

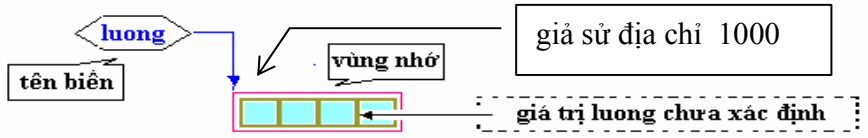
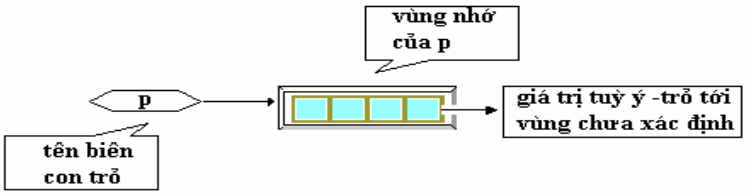
int tuoi;

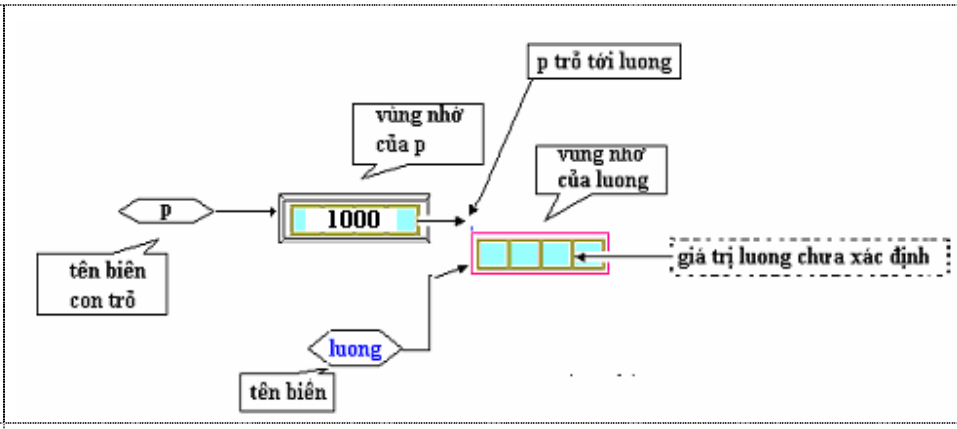
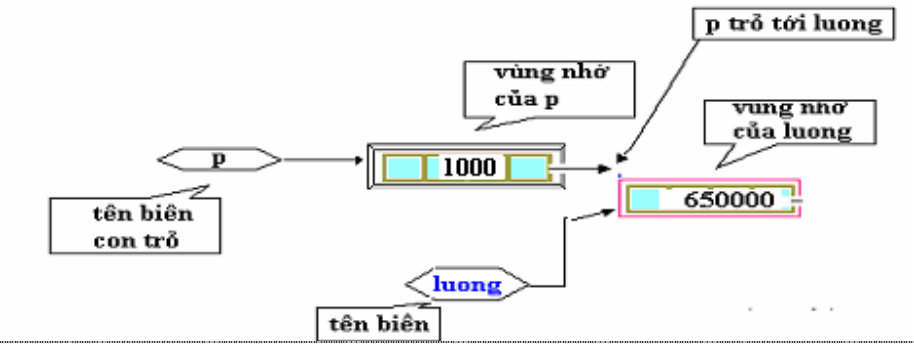


float luong;



Nếu chúng ta có con trỏ **p** kiểu **float, p** lưu địa chỉ của **luong** và luong 65000 như sau:

float luong;	
float * p;	

p = &luong;	
*p = 650000	

Khi con trỏ trỏ tới một vùng nhớ ví dụ như **p** trỏ tới **luong** thì khi truy xuất ***p** chính là giá trị của vùng nhớ do **p** trỏ tới tức là *p ⇔ luong.

Với con trỏ trỏ tới một con trỏ khác chẳng hạn như ví dụ sau:

```
int a = 10;
int *pa;
int **ppa;
pa  = &a;   // p trỏ tới a
ppa = &pa; // ppa trỏ tới pa
```

thì chúng ta có:

*ppa ⇔ pa ⇔ &a;

**ppa ⇔ *pa ⇔ a;

- Các phép toán trên con trỏ (địa chỉ)

a. Phép so sánh hai con trỏ

Trên con trỏ tồn tại các phép so sánh (= =, !=, <, <=, >,>=) hai con trỏ bằng nhau là hai con trỏ cùng trỏ tới một đối tượng (có giá trị bằng nhau), ngược lại là khác nhau. Con trỏ trỏ tới vùng nhớ có địa chỉ nhỏ hơn là con trỏ nhỏ hơn.

b. Phép cộng con trỏ với số nguyên

Giả sử p là con trỏ kiểu T, k là số nguyên thì (p + k) cũng là con trỏ kiểu T, không mất tổng quát giả sử p trỏ tới phần tử **t** thì

- **p+1** là con trỏ trỏ tới một phần tử kiểu T kế tiếp sau t
- **p+2** trỏ tới một phần tử kiểu T kế tiếp sau t 2 phần tử,...
- **p -1** là con trỏ trỏ tới một phần tử kiểu T kế tiếp trước t
- **p -2** trỏ tới một phần tử kiểu T kế tiếp trước t hai phần tử,...
- tổng quát **p+k** trỏ tới phần tử cách t một khoảng k phần tử kiểu T (nếu k >0 dịch về phía địa chỉ lớn, k<0 thì dịch về phía địa chỉ nhỏ).

Ví dụ:

int a; // giả sử a có địa chỉ 150

int *p;

p = &a;

thì **p+1** là con trỏ kiểu nguyên và p+1 trỏ tới địa chỉ 152; p + k trỏ tới 150 +2*k.

c. Phép trừ hai con trỏ

Nếu **p, q** là hai con trỏ cùng kiểu T thì **p-q** là số nguyên là số các phần tử kiểu T nằm giữa hai phần tử do p và q trỏ tới.

Ví dụ:

int *p, *q;

giả sử p trỏ tới phần tử có địa chỉ 180, q trỏ tới phần tử có địa chỉ 160 thì

(p-q) = = 10;

float *r1, *r2;

giả sử r1 trỏ tới phần tử có địa chỉ 120, r2 trỏ tới phần tử có địa chỉ 100 thì

(r1-r2) = = 5;

V.4.2 - Tổ chức vùng nhớ của mảng

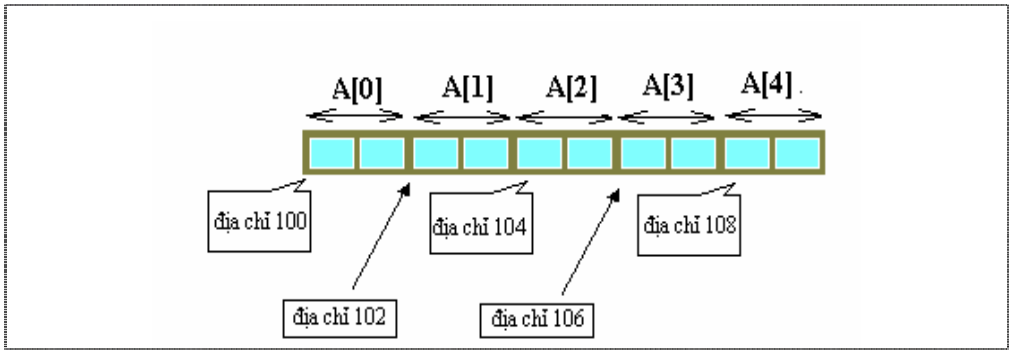
Như trong phần trên chúng ta đã nói, khi có một định nghĩa mảng thì chương trình biên dịch cấp phát một vùng nhớ (liên tiếp - các ô nhớ liền kề nhau) có kích thước bằng tổng kích thước của các phần tử trong mảng, các phần tử của mảng xếp tuần tự trong bộ nhớ, phần tử đầu tiên có địa chỉ thấp nhất trong vùng đó, và đây cũng chính là địa chỉ của mảng, phần tử thứ hai của mảng sẽ là ô nhớ kề sát sau (ô nhớ có địa chỉ cao hơn) phần tử thứ nhất,... Ở đây chúng ta nói ô nhớ có thể là 1 byte, 2 byte, 4 byte,.. tùy theo kiểu dữ

liệu của các phần tử mảng là gì (tương ứng là 1,2,4,.. byte). Và địa chỉ của ô nhớ là địa chỉ của byte đầu tiên trong các byte đó.

Ví dụ 1: chúng ta định nghĩa mảng A kiểu nguyên:

int A[5];

Chương trình dịch sẽ cấp phát một vùng nhớ 5 × 2 = 10 byte cho mảng A, giả sử rằng vùng nhớ đó có địa chỉ là **100** (byte đầu tiên có địa chỉ là 100). thì các phần tử của A như hình sau:



(mảng A có 5 phần tử kiểu int)

Ví dụ 2: chúng ta định nghĩa mảng X kiểu float:

float X[6];

Chương trình dịch sẽ cấp phát một vùng nhớ 6 × 4 = 24 byte cho mảng X, giả sử rằng vùng nhớ đó có địa chỉ là 2**00** (byte đầu tiên có địa chỉ là 200) thì các phần tử của X được cấp phát là địa chỉ của X[0] là 200 (&X[0] = 200), &X[1] = 204,..,&X[5] =216.

Với mảng 2 chiều, giả sử mảng **D** có **n** dòng, **m** cột, kiểu **int:**

int D[n][m]; // n, m là hằng nguyên

Tức là có **n×m** phần tử kiểu nguyên, như trên chúng ta nói D được xem là mảng có **n** phần tử, mỗi phần tử lại là một mảng, mảng thành phần này có **m** phần tử. Như vậy D được cấp phát một vùng nhớ liên tiếp, trong vùng đó có **n** vùn con cho n phần tử (dòng), trong mỗi vùng con có **m** ô nhớ (mỗi ô là một phần tử, 2byte). Hay nói cách khác các phần tử của mảng được cấp phát liên tiếp, đầu tiên là **m** phần tử của hàng 0, sau đó là **m** phần tử của hàng 1,...

Giả sử địa chỉ của mảng D là xxxx thì các phần tử của nó như sau:

D[0] có địa chỉ là xxxx

D[0][0] có địa chỉ là xxxx (&D[0][0] = =xxxx)

D[0][1] có địa chỉ là xxxx + 2 (&D[0][1] = =xxxx + 2)

....

D[0][m-1] có địa chỉ là xxxx+2(m-1) (&D[0][m-1] = = xxxx +2(m-1))

D[1] có địa chỉ là xxxx +2m

D[1][0] có địa chỉ là xxxx +2m (&D[0][0] = =xxxx+2m)

D[1][1] có địa chỉ là xxxx + 2m +2 (&D[0][1] = =xxxx + 2m+2)

....

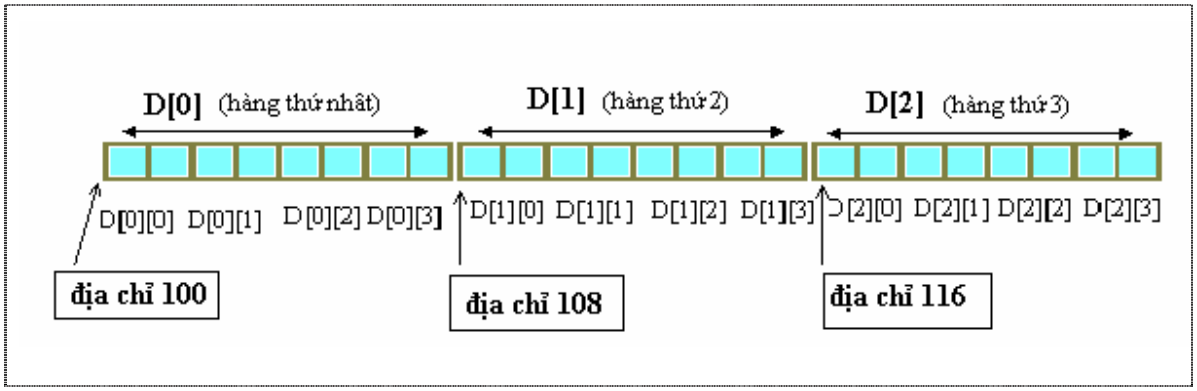
D[1][m-1] có địa chỉ là xxxx+2m +2(m -1) (&D[0][m-1] = = xxxx +2m+2(m-1))

...

Ví dụ:

int D[3][4];

Giả sử D được cấp phát tại vùng nhớ có địa chỉ 100 thì các phần tử của D như sau:



➢ ***Hạn chế số phần tử của mảng***

Tuy rằng ngôn ngữ không đưa ra con số cụ thể giới hạn các phần tử của mảng, nhưng kích thước của mảng bị hạn chế bởi các yếu tố sau:

- Các phần tử mảng được cấp phát liên tiếp, trong 1 đoạn bộ nhớ (64kb), do vậy tổng kích thước của mảng ≤ 64kb (số_pt × sizeof(kiểu_mảng) ≤ 65535)
- Kích thước mảng có thể cấp phát phụ thuộc lượng bộ nhớ tự do mà chương trình dịch có thể cấp phát được.

Ví dụ nếu bộ nhớ tự do (trong 1 đoạn) có thể cấp phát còn lại là 100 byte thì nếu là mảng nguyên 1 chiều kiểu int thì kích thước tối đa có thể là 50, với mảng một chiều kiểu float thì chỉ có thể là 25 phần tử,..

V.4.3 - Liên hệ giữa con trỏ và mảng

Trong C con trỏ và mảng có liên hệ rất chặt chẽ với nhau, như trên chúng ta biết tên mảng là một hằng con trỏ. Hơn nữa các phần tử của mảng có thể được truy xuất thông qua chỉ số hoặc thông qua con trỏ.

Như trên chúng ta biết, mảng được cấp phát tại vùng nhớ nào đó và địa chỉ của vùng nhớ đó chính là địa chỉ của mảng. Tên mảng là con trỏ trỏ tới chính địa chỉ của nó hay nói khác tên mảng là con trỏ lưu địa chỉ của mảng, nhưng là hằng con trỏ. Chúng ta giải thích cụ thể hơn qua ví dụ sau:

Ví dụ với khai báo

int A[3], D[2][5];

thì A, D là các con trỏ và: A chính là địa chỉ của mảng A, hay bằng &A[0]; tương tự cho mảng D ta cũng có D ⇔ &D[0].

Và theo như cách gọi của con trỏ thì ta nói A là con trỏ trỏ tới phần tử đầu tiên của mảng. Với mảng hai chiều D thì cũng tương tự, D cũng là một con trỏ trỏ tới D[0], và D[0] lại là một con trỏ trỏ tới D[0][0], có thể nói D là con trỏ trỏ tới con trỏ.

Như vậy A là mảng kiểu int tức là các phần tử của nó có kiểu int, và như vậy A là con trỏ kiểu int, hay A có kiểu là int *.

Tương tự, D[i] (nó chung là các hàng của mảng D) là con trỏ kiểu int, tức là D[i] có kiểu là int *, và D là con trỏ trỏ tới D[0] - Các D[i] là mảng int[5], vậy D là con trỏ kiểu int [5].

Tên mảng có thể gán cho các (biến) con trỏ có kiểu phù hợp.

Ví dụ: với các con trỏ và mảng sau

```
int A[10];
int D[2][4];
int *p;
int (*q)[4]; // khai báo q là con trỏ kiểu int [4],
```

chúng ta có thể thực hiện các phép gán sau:

```
p = A;
q = D;
p = D[i];
```

- ***Truy xuất các phần tử mảng qua con trỏ***

Giả sử A là mảng một chiều như trên chúng ta có:

- A là con trỏ trỏ tới A[0] hay A tương đương với &A[0]
- (A +1) là con trỏ trỏ tới phần tử kiểu T kế tiếp sau A[0] tức là A+1 trỏ tới A[1] hay (A+1) ⇔ &A[1]
- Tổng quát (A+i) ⇔ &A[i]

*Như chúng ta biết từ trước là nếu p là con trỏ lưu địa chỉ của biến x thì * p là giá trị của x.*

Như vậy *A chính là giá trị của A[0], *(A+1) là A[1],...

tổng quát chúng ta có ***(A+i) ⇔A[i]**

Ví dụ chúng ta có thể minh họa bằng đoạn lệnh nhập và in các phần tử mảng A như sau:

```
int A[10];
int i;
....
// nhập mảng A có 10 phần tử
for(i = 0; i<10; i++)
    {   printf("A[%d] = ", i);
        scanf("%d", A+i);
    }
// in mảng A có 10 phần tử,
for(i = 0; i<10; i++)
    {   printf("A[%d] = ", i);
        scanf("%d", *(A+i));
    }
```

Với mảng 2 chiều D[n][m] cũng tương tự như trên ta có:

- D là con trỏ trỏ tới hàng dầu tiên trong mảng tức là: D ⇔ &D[0]
  - D[0] là con trỏ trỏ tới phần tử đầu tiên là D[0][0] hay D[0] ⇔ &D[0][0] nên *D[0] ⇔ D[0][0];

    hay ** D ⇔ D[0][0]
  - (D[0]+j) con trỏ tới phần tử D[0][j], hay (D[0]+j) ⇔ &D[0][j] nên ta có *(D[0]+j) ⇔ D[0][j]

    hay *(*D+j) ⇔ D[0][j]
- Tương tự tổng quát ta có (D+i) ⇔ &D[i].
  - D[i] là con trỏ trỏ tới phần tử đầu tiên là D[i][0] hay D[i] ⇔ &D[i][0] nên *D[i] ⇔ D[i][0];

    hay *(*D+i) ⇔ D[i][0]
  - (D[i]+j) con trỏ tới phần tử D[i][j], hay (D[i]+j) ⇔ &D[i][j] nên ta có *(D[i]+j) ⇔ D[i][j]

    hay tổng quát ta có ***(*(D+i)+j) ⇔ D[i][j]**

Bài tập

1. tích vô hướng 2 vector
2. Nhập mảng, tìm phần tử lớn nhất, nhỏ nhất, trung bình các phần tử dương, âm
3. Nhập mảng A(n), các phần tử là số nguyên, hãy cho biết trật tự của mảng
4. bài toán sắp xếp bằng phương pháp chọn và đổi chỗ
5. tìm kiếm trên mảng không thứ tự
6. tìm kiếm trên mảng có thứ tự
7. in các phần tử khác nhau của mảng không có thứ tự
8. in các phần tử khác nhau của mảng có thứ tự
9. ghép hai mảng tăng
10. Viết chương trình nhập A(n,m), B(n,m), tính và in C= A+B
11. Viết chương trình nhập A(n,m), B(m,p), tính và in C= A*B
12. Viết chương trình nhập A(n,n) kiểm tra A có là ma trận đối xứng hay không?
13. Viết chương trình nhập A(n,n) kiểm tra A có là ma trận đơn vị hay không?
14. Viết chương trình nhập A(n,n) kiểm tra A điểm yên ngựa hay không? nếu có hãy in giá trị, chỉ số của nó.
15. Viết chương trình xây dựng ma trận xoắn ốc
16. Viết chương trình xây dựng ma phương bậc lẻ
17. Viết chương trình tính định thức ma trận vuông bằng phương pháp khử
18. Viết chương trình giải hệ phương trình bậc nhất n ẩn

VI – Các vấn đề cơ bản về hàm

Trong các ngôn ngữ lập trình có cấu trúc thì việc xây dụng và sử dụng các chương trình con có ý nghĩa quan trọng nó giúp chúng ta phân chia chương trình thành các modul độc lập nhỏ hơn, dễ kiểm soát, dễ phát triển hơn và có thể sử dụng lại các modul đó ở nhiều nơi mà không phải viết lại. Khác với một số ngôn ngữ lập trình khác, chương trình con có thể là hàm hoặc thủ tục, trong C chỉ có một loại đó là hàm.

Trong phần này chúng ta xem xét hàm ở mức độ đơn giản nhất, giúp bạn đọc có khái niệm cơ bản ban đầu về hàm và có thể viết được các hàm đơn giản

Hàm là một là một đơn vị độc lập của chương trình, mỗi hàm có một chức năng xác định, có thể được gọi thực hiện bởi hàm hoặc chương trình khác. Trong C các hàm đều ngang mức, tức là trong định nghĩa hàm không thể chứa định nghĩa hàm khác (gọi là hàm 1 mức). Có hai loại hàm đó là hàm của thư viện và hàm do người lập trình định nghĩa (hay còn gọi là hàm của người dùng)

Với một hàm nói chung thì các thông tin xác định là: Tên hàm, kiểu giá trị trả về của hàm (gọi là kiểu hàm), và các tham số của nó. Tức là với một hàm cần phải xác định 3 thông tin để 'nhận diện'

- tên hàm
- dữ liệu vào
- kiểu quả trả về (kiểu hàm)

Nói chung để xây dựng một hàm thường có hai phần đó là khai báo nguyên mẫu hàm và định nghĩa hàm. Vị trí của hai phần này bạn đọc xem lại phần cấu trúc chương trình.

VI.1 - Nguyên mẫu (prototype) hàm

Nguyên mẫu hàm là dòng khai báo cho chương trình dịch biết các thông tin về hàm bao gồm: tên hàm, kiểu hàm và kiểu các tham số (đầu vào) của hàm.

Cú pháp khai báo nguyên mẫu hàm

<kiểu_hàm> <tên_hàm>([Các_khai_báo_kiểu_tham_số]);

Trong đó

- **tên_hàm**: là một tên hợp lệ theo quy tắc về tên của ngôn ngữ C. mỗi hàm có tên duy nhất và không được trùng với các từ khóa. Tên hàm sẽ được dùng để gọi hàm.
- **kiểu_hàm** : Hàm có thể trả về một giá trị cho nơi gọi, giá trị đó thuộc một kiểu dữ liệu nào đó, kiểu đó được gọi là kiểu hàm. Kiểu hàm có thể là kiểu chuẩn cũng có thể là kiểu do người dùng định nghĩa. Nếu hàm không trả về giá trị thì kiểu hàm là **void.**
- **Các_khai_báo_kiểu_tham_số:** Hàm có thể nhận dữ liệu vào thông qua các tham số của nó (tham số hình thức), các tham số này cũng thuộc kiểu dữ liệu xác định. Có thể

có nhiều tham số, các tham số cách nhau bởi dấu phẩy (,). Trong nguyên mẫu không bắt buộc phải có tên tham số nhưng kiểu của nó thì bắt buộc. Nếu hàm không có tham số chúng ta có thể để trống phần này hoặc có thể khai báo là void.

Ví dụ:

- **int max(int a, int b)**; // *khai báo nguyên mẫu hàm* **max***, có hai tham số kiểu int, kết quả trả về kiểu int*
- **float f(float, int)**; // *nguyên mẫu hàm* **f***, có hai tham, tham số thứ nhất kiểu float, tham số thứ 2 kiểu int, kết quả trả về kiểu float*
- **void nhapmang(int a[], int)**; // *hàm* **nhapmang***, kiểu void (không có giá trị trả về), tham số thứ nhất là một mảng nguyên, tham số thứ 2 là một số nguyên*
- **void g()**; // *hàm* **g** *không đối, không kiểu.*

VI.2 - Định nghĩa hàm

Cú pháp:

```
<kiểu_hàm>  <tên_hàm>([khai_báo_tham_số])
   {
          < thân hàm>
     }
```

Dòng thứ nhất là tiêu đề hàm (dòng tiêu đề) chứa các thông tin về hàm: tên hàm, kiểu của hàm (*hai thành phần này giống như trong nguyên mẫu hàm)* và khai báo các tham số (tên và kiểu) của hàm, nếu có nhiều hơn một thì các tham số cách nhau bởi dấu phẩy(,).

Thân hàm là các lệnh nằm trong cặp **{ },** đây là các lệnh thực hiện chức năng của hàm. Trong hàm có thể có các định nghĩa biến, hằng hoặc kiểu dữ liệu; các thành phần này trở thành các thành phần cục bộ của hàm.

Nếu hàm có giá trị trả về (kiểu hàm khác void) thì trong thân hàm trước khi kết thúc phải có câu lệnh trả về giá trị:

```
return <giá trị>;
```

<giá_trị> sau lệnh return chính là giá trị trả về của hàm, nó phải có kiểu phù hợp với kiểu của hàm được khai báo trong dòng tiêu đề. *Trường hợp hàm void chúng ta có thể dùng câu lệnh return (không có giá trị) để kết thúc hàm hoặc khi thực hiện xong lệnh cuối cùng (gặp } cuối cùng) hàm cũng kết thúc.*

Ví dụ 1: Hàm max trả lại giá trị lớn nhất trong 2 số nguyên a, b

```
void max (int a, int b)
   { if(a>b)
         return a;
     else
         return b;
   }
```

Ví dụ 2: Hàm nhập một mảng có n phần tử nguyên:

- tên hàm: **nhapmang**
- giá trị trả về: không trả về
- tham số: có hai tham số là mảng cần nhập A và số phần tử cần nhập N

nguyên mẫu hàm như sau:

void nhapmang (int [], int);

định nghĩa hàm như sau:

```
void nhapmang (int A[], int N)  {
    int i;
    printf("\nNhap mang co %d phan tu \n",N);
    for(i=0;i<N; i++)
      {    printf("a[%d]= ",i);
           scanf("%d",&a[i]);
         }
    return ;
   }
```

VI.3 - Lời gọi hàm và truyền tham số

Một hàm có thể gọi thực hiện thông qua tên hàm, với những hàm có tham số thì trong lời gọi phải truyền cho hàm các tham số thực sự (đối số) tương ứng với các tham số hình thức.

Khi hàm được gọi và truyền tham số phù hợp thì các lệnh trong thân hàm được thực hiên bắt đầu từ lệnh đầu tiên sau dấu mở móc { và kết thúc khi gặp lệnh return, exit hay gặp dấu đóng móc } kêt thúc hàm.

Cú pháp:

<tên_hàm> ([danh sách các tham số thực sự]);

Các tham số thực sự phải phù hợp với các tham số hình thức:

- số tham số thực sự phải bằng số tham số hình thức.
- Tham số thực sự được truyền cho các tham số hình thức tuần tự từ trái sang phải, tham số thực sự thứ nhất truyền cho tham số hình thức thứ nhất, tham số thực sự thứ 2 truyền cho tham số hình thức thứ 2,.. kiểu của các tham số hình thức phải phù hợp với kiểu của các tham số hình thức. Sự phù hợp ở đây được hiểu là kiểu trùng nhau hoặc kiểu của tham số thực sự có thể ép về kiểu của tham số hình thức.

Ví dụ: *giả sử có các hàm max, nhapmang như đã định nghĩa ở trên*

int a=4, b=6,c;

int D[10];

```
c = max(a,b);
nhapmang(D,5);
```

Lưu ý sau này chúng ta thấy hàm có thể có đối số thay đổi và chúng có thể được truyền tham số với giá trị ngầm định vì vậy tham số hình thức có thể ít hơn tham số thực sự.

➢ Một số ví dụ

Ví dụ VI.1*: Viết chương trình nhập một số n từ bàn phím (n >2), in các số nguyên tố từ 2 tới n.*

Giải: Để in các số nguyên tố trong khoảng từ 2 tới n chúng ta thực hiện như sau: với mỗi số k ∈ [2,n] kiểm tra xem k có là nguyên tố hay không, nếu đúng thì in k ra màn hình. Vậy chúng ta sẽ xây dựng hàm để kiểm tra một số có là nguyên tố hay không,

- tên hàm: nguyento
- đầu vào: k là một số nguyên cần kiểm tra
- giá trị trả về: 1 nếu đúng k là số nguyên tố, ngược lại trả về 0.

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
int nguyento(int k);//nguyên mẫu hàm kiểm tra k là số nguyên tố
void main(){
    int k, n;
    do{
          printf("\nNhap gia tri n (n>=2) = ");
          scanf("%d",&n);
       } while(n<2);
    printf("\nCac so nguyen to tu 2 toi %d la \n",n);
    for(k=2; k<=n; k++)
        if (nguyento(k)) printf("%d, ",k);
    }
//-----định nghĩa hàm nguyento ------------

int nguyento(int k){
    int i=2;
    while((i<=sqrt(k))&&(k%i)) i++;
        if(i>sqrt(k))
           return 1;
```

```
    else
            return 0;
        }
```

(ví dụ VI.1 - in các số nguyên tố từ 2 tới n)

Ví dụ VI.2 - Viết chương trình nhập 2 mảng A, B có n phần tử nguyên từ bàn phím, tính mảng tổng C = A+B, in 3 mảng A, B, C ra màn hình.

Yêu cầu:

- n <20;
- chương trình gồm 3 hàm: hàm nhập mảng, hàm in mảng, hàm tổng hai mảng.

Giải: Chúng ta cần xây dựng 3 hàm như sau

1. *Hàm nhập mảng*
   - Tên hàm: nhapmang
   - Giá trị trả về: không trả về (void)
   - Tham số: mảng cần nhập (int A[]) và số phần tử kiểu int
   - nguyên mẫu: void nhapmang(int A[], int n);
2. *Hàm in mảng*
   - Tên hàm: inmang
   - Giá trị trả về: không trả về (void)
   - Tham số: mảng cần in (int A[]) và số phần tử của mảng kiểu int
   - nguyên mẫu: void inmang(int A[], int n);
3. *Hàm tính tổng hai mảng*
   - Tên hàm: tong
   - Giá trị trả về: không trả về (void)
   - Tham số: hai mang A, B, mảng kết quả C và số phần tử kiểu int
   - nguyên mẫu: void tong (int A[], int B[], int C[], int n);

Các bạn có chương trình minh họa như sau

```
#include <stdio.h>
#include <conio.h>
void nhapmang(int a[], int n);
void inmang(int a[], int n);
void tong(int A[],int B[],int C[],int n);
void main(){
      clrscr();
      const int max = 20; //
      int A[max], B[max],C[max];
      int n;
      do{
              printf("\nNhap so phan tu mang  = ");
              scanf("%d",&n);
            } while(n<1 || n>max);
      printf("\nNhap A \n");
      nhapmang(A,n);
      printf("\nNhap B \n");
      nhapmang(B,n);
      tong(A,B,C,n);
      printf("\nmang A: ");
      inmang(A,n);
      printf("\nmang B: ");
      inmang(B,n);
      printf("\nmang C: ");
      inmang(C,n);
      getch();
   }
void nhapmang(int a[], int n){
      int i;
      printf("\nNhap mang co %d phan tu \n",n);
      for(i=0; i<n; i++)
        {
            printf("pt 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 tong(int A[],int B[],int C[],int n){
      int i;
     for (i = 0; i<n; i++) C[i]=A[i]+B[i];
   }
```

Hàm và truyền tham số

Với C việc tryền tham số cho hàm được thực hiện qua cơ chế truyền tham trị. Tức là trong hàm chúng ta sử dụng tham số hình thức như là một bản sao dữ liệu của tham số được truyền cho hàm, do vậy chúng không làm thay đổi giá trị của tham số truyền vào. Hay nói khác đi, các tham số hình thức là các biến cụ bộ trong hàm, sự thay đổi của nó trong hàm không ảnh hưởng tới các biến bên ngoài.

Vậy trong trường hợp thực sự cần thay đổi giá trị của tham số thì thế nào? chẳng hạn bạn cần hàm để hoán đổi giá trị của a và b.

Nếu bạn viêt hàm

```
void doicho(int x, int y)  {
        int  tg;
        tg = x;
        x=y;
        y=tg;
    }
```

hàm này đúng cú pháp nhưng với các lệnh sau:

```
int a = 4;
int b = 6;
printf ("\ntruoc khi goi ham doi cho a=%d, b=%d",a,b);
doicho(a,b);
printf ("\nsau khi goi ham doi cho a=%d, b=%d",a,b);
```

kết quả in ra là

```
truoc khi goi ham doi cho a=4,b=6
sau khi goi ham doi cho a=4,b=6
```

Rõ ràng hàm đổi chỗ (doicho) thực hiện không đúng, nguyên nhân là với hàm doicho x, y là hai biên cục bộ, khi gọi hàm doicho(a,b) chương trình dịch cấp phát vùng nhớ cho hai biến (tham số hình thức) x, y và sao chép giá trị của a vào x, b vào y, mọi thao tác trong hàm doicho đều thực hiên trên x, y mà không ảnh hưởng tới a và b, kết quả là a, b không đổi.

Để khắc phục điều này chúng ta định nghĩa hàm với tham số là con trỏ và khi gọi các bạn hãy truyền cho nó địa chỉ của tham số thực sự, ví dụ:

```
void doicho2(int * x, int *y)    {
        int  tg;
        tg  =  *x;
        *x = *y;
        *y = tg;
    }
```

Lúc này với các lệnh sau:

```
int a = 4;
int b = 6;
printf ("\ntruoc khi goi ham doi cho a=%d, b=%d",a,b);
doicho(&a,&b);
printf ("\nsau khi goi ham doi cho a=%d, b=%d",a,b);
```

kết quả in ra là

```
truoc khi goi ham doi cho a = 4,b = 6
sau khi goi ham doi cho a = 6 , b = 4
```

Tài liệu tham khảo

1. Đỗ Phúc, Tạ Minh Châu - Kỹ thuật lập trình Turbo C
2. Phạm Văn Ất - Kỹ thuật lập trình Turbo C - Cơ sở và nâng cao
3. Scott Robert Ladd - C++ Kỹ thuật và ứng dụng, Nguyễn Hùng dịch