BÀI 1
CÁC KHÁI NIỆM CƠ BẢN

1.1. Lịch sử phát triển ngôn ngữ lập trình C

Điều kiện ra đời:

- Đầu những năm 1970, việc lập trình hệ thống vẫn dựa trên hợp ngữ (*Assembly*) nên công việc nặng nề, phức tạp và khó chuyển đổi chương trình giữa các hệ thống máy tính khác nhau. Điều này dẫn đến nhu cầu cần có một ngôn ngữ lập trình hệ thống bậc cao đồng thời có khả năng chuyển đổi dễ dàng từ hệ thống máy tính này sang hệ thống máy tính khác (còn gọi là tính *khả chuyển – portability*) để thay thế hợp ngữ.
- Cũng vào thời gian đó, người ta muốn viết lại hệ điều hành Unix để cài đặt trên các hệ máy tính khác nhau, vì vậy cần có một ngôn ngữ lập trình hệ thống có tính khả chuyển cao để viết lại hệ điều hành Unix.

Ngôn ngữ C ra đời tại phòng thí nghiệm BELL của tập đoàn AT&T (Hoa Kỳ) do Brian W. Kernighan và Dennis Ritchie phát triển vào đầu những năm 1970 và hoàn thành vào năm 1972.

C được phát triển dựa trên nền các ngôn ngữ BCPL (*Basic Combined Programming Language*) và ngôn ngữ B. Cũng vì được phát triển dựa trên nền ngôn ngữ B nên ngôn ngữ mới được Brian W. Kernighan và Dennis Ritchie đặt tên là ngôn ngữ C như là sự tiếp nối ngôn ngữ B.

C có các đặc điểm là một ngôn ngữ lập trình hệ thống mạnh, khả chuyển, có tính linh hoạt cao và có thế mạnh trong xử lí các dạng dữ liệu số, văn bản, cơ sở dữ liệu. Vì vậy C thường được dùng để viết các chương trình hệ thống như hệ điều hành (ví dụ hệ điều hành Unix có 90% mã nguồn được viết bằng C, 10% còn lại viết bằng hợp ngữ) và các chương trình ứng dụng chuyên nghiệp có can thiệp tới dữ liệu ở mức thấp như xử lí văn bản, cơ sở dữ liệu, xử lí ảnh…

W. Kernighan và Dennis Ritchie công bố ngôn ngữ C trong lần xuất bản đầu của cuốn sách "*The C programming language*" (1978). Sau đó người ta đã bổ sung thêm những yếu tố và khả năng mới vào trong ngôn ngữ C (ví dụ như đưa thêm kiểu liệt kê **enum**, cho phép kiểu dữ liệu trả về bởi hàm là kiểu **void**, **struct** hoặc **union**… và đặc biệt là sự bổ sung các thư viện cho ngôn ngữ. Lúc đó đồng thời tồn tại nhiều phiên bản khác nhau của ngôn ngữ C nhưng không tương thích với nhau. Điều này gây khó khăn cho việc trao đổi mã nguồn chương trình C viết trên các phiên bản ngôn ngữ C khác nhau (bạn sẽ rất khó đọc và hiểu chương trình của người khác, và khi bạn muốn sửa nó thành chương trình của mình dịch trên bộ dịch của mình thì sẽ tốn rất nhiều công sức) và dẫn đến nhu cầu chuẩn hóa ngôn ngữ C.

Năm 1989, Viện tiêu chuẩn quốc gia của Hoa Kỳ (*American National Standards Institute* - ANSI) đã công bố phiên bản chuẩn hóa của ngôn ngữ C trong lần tái bản thứ 2 của cuốn sách "*The C programming language*" của các tác giả W. Kernighan và Dennis Ritchie. Và từ đó đến nay phiên bản này vẫn thường được nhắc đến với tên gọi là *ANSI C* hay *C chuẩn* hay C89 (vì được công bố năm 1989). ANSI C là sự kế thừa phiên bản đầu tiên của ngôn ngữ C (do K. Kernighan & D. Ritchie công bố năm 1978) có đưa thêm vào nhiều yếu tố mới và ANSI C đã quy định các thư viện chuẩn dùng cho ngôn ngữ C. Tất cả các phiên bản của ngôn ngữ C hiện đang sử dụng đều tuân theo các mô tả đã được nêu ra trong ANSI C, sự khác biệt nếu có thì chủ yếu nằm ở việc đưa thêm vào các thư viện bổ sung cho thư viện chuẩn của ngôn ngữ C.

Hiện nay cũng có nhiều phiên bản của ngôn ngữ C khác nhau và mỗi phiên bản này gắn liền với một bộ chương trình dịch cụ thể của ngôn ngữ C. Các bộ chương trình dịch phổ biến của ngôn ngữ C có thể kể tên như:

- Turbo C++ và Borland C++ của Borland Inc.
- MSC và VC của Microsoft Corp.
- GCC của GNU project.

  …

Trong các trình biên dịch trên thì Turbo C++ là trình biên dịch rất quen thuộc và sẽ được chọn làm trình biên dịch cho các ví dụ sử dụng trong tài liệu này.

1.2. Các phần tử cơ bản của ngôn ngữ C

1.2.1. Tập kí tự

Chương trình nguồn của mọi ngôn ngữ lập trình đều được tạo nên từ các phần tử cơ bản là tập kí tự của ngôn ngữ đó. Các kí tự tổ hợp với nhau tạo thành các từ, các từ liên kết với nhau theo một quy tắc xác định (quy tắc đó gọi là cú pháp của ngôn ngữ) để tạo thành các câu lệnh. Từ các câu lệnh người ta sẽ tổ chức nên chương trình.

Tập kí tự sử dụng trong ngôn ngữ lập trình C gồm có:

26 chữ cái hoa:	A B C ... X Y Z
26 chữ cái thường:	a b c … x y z.
10 chữ số:	0 1 2 3 4 5 6 7 8 9.
Các kí hiệu toán học:	+ - * / = < >
Các dấu ngăn cách:	. ; , : space tab
Các dấu ngoặc:	( ) [ ] { }
Các kí hiệu đặc biệt:	_ ? $ & # ^ \ ! ‘ “ ~ .v.v.

1.2.2. Từ khóa

Từ khóa (*Keyword*) là những từ có sẵn của ngôn ngữ và được sử dụng dành riêng cho những mục đích xác định.

Một số từ khóa hay dùng trong Turbo C++

break	Case	char	const	continue	default
do	Double	else	enum	float	for
goto	If	int	interrupt	long	return
short	Signed	sizeof	static	struct	switch
typedef	Union	unsigned	void	while	

Chú ý: Tất cả các từ khóa trong C đều viết bằng chữ thường.

Các từ khóa trong C được sử dụng để

- Đặt tên cho các kiểu dữ liệu: **int**, **float**, **double**, **char**, **struct**, **union**…
- Mô tả các lệnh, các cấu trúc điều khiển: **for**, **do**, **while**, **switch**, **case**, **if**, **else**, **break**, **continue**…

1.2.3. Định danh

Định danh (*Identifier* – hoặc còn gọi là *Tên*) là một dãy các kí tự dùng để gọi tên các đối tượng trong chương trình. Các đối tượng trong chương trình gồm có biến, hằng, hàm, kiểu dữ liệu… ta sẽ làm quen ở những mục tiếp theo.

Định danh có thể được đặt tên sẵn bởi ngôn ngữ lập trình (đó chính là các từ khóa) hoặc do người lập trình đặt. Khi đặt tên cho định danh trong C, người lập trình cần tuân thủ các quy tắc sau :

1. Các kí tự được sử dụng trong các định danh của ngôn ngữ C chỉ được gồm có: chữ cái, chữ số và dấu gạch dưới "_" (*underscore*).
2. Bắt đầu của định danh phải là chữ cái hoặc dấu gạch dưới, không được bắt đầu định danh bằng chữ số.
3. Định danh do người lập trình đặt không được trùng với từ khóa.

Ngoài các quy tắc bắt buộc trên lập trình viên có quyền tùy ý đặt tên định danh trong chương trình của mình.

Một số ví dụ về định danh:

i, x, y, a, b, _function, _MY_CONSTANT, PI, gia_tri_1…

Ví dụ về định danh không hợp lệ

1_a, 3d, 55x	bắt đầu bằng chữ số
so luong, ti le	có kí tự không hợp lệ (dấu cách – *space*) trong tên

int, char trùng với từ khóa của ngôn ngữ C

Lưu ý:

- Đôi khi định danh do ta đặt gồm nhiều từ, khi đó để dễ đọc ta nên tách các từ bằng cách sử dụng dấu gạch dưới. Ví dụ định danh danh_sach_sinh_vien dễ đọc và dễ hiểu hơn so với định danh danhsachsinhvien.
- Định danh nên có tính chất gợi nhớ, ví dụ nếu ta muốn lưu trữ các thông tin về các sinh viên vào một biến nào đó thì biến đó nên được đặt tên là danh_sach_sinh_vien hay ds_sv... Và ngược lại định danh danh_sach_sinh_vien chỉ nên dùng để đặt tên cho những đối tượng liên quan đến sinh viên chứ không nên đặt tên cho các đối tượng chứa thông tin về cán bộ viên chức. Việc đặt tên có tính gợi nhớ giúp cho người lập trình và những người khác đọc chương trình viết ra được dễ dàng hơn.
- Ngôn ngữ C phân biệt chữ cái thường và chữ cái hoa trong các định danh, tức là dinh_danh khác với Dinh_danh.
- Một thói quen của những người lập trình là các hằng thường được đặt tên bằng chữ hoa, các biến, hàm hay cấu trúc thì đặt tên bằng chữ thường. Nếu tên gồm nhiều từ thì ta nên phân cách các từ bằng dấu gạch dưới.

  Ví dụ:

Định danh	**Loại đối tượng**
HANG_SO_1, _CONSTANT_2	hằng
a, b, i, j, count	biến
nhap_du_lieu, tim_kiem, xu_li	hàm
sinh_vien, mat_hang	cấu trúc

1.2.4. Các kiểu dữ liệu

Kiểu dữ liệu là gì?

Dữ liệu là tài nguyên quan trọng nhất trong máy tính, song dữ liệu trong máy tính lại không phải tất cả đều giống nhau. Có dữ liệu là chữ viết, có dữ liệu là con số, lại có dữ liệu khác là hình ảnh, âm thanh… Ta nói rằng các dữ liệu đó thuộc các kiểu dữ liệu khác nhau.

Một cách hình thức, kiểu dữ liệu có thể được định nghĩa gồm 2 điểm như sau:

- Một kiểu dữ liệu là một tập hợp các giá trị mà một dữ liệu thuộc kiểu dữ liệu đó có thể nhận được.
- Trên một kiểu dữ liệu ta xác định một số phép toán đối với các dữ liệu thuộc kiểu dữ liệu đó.

Ví dụ:

Trong ngôn ngữ C có kiểu dữ liệu **int**. Một dữ liệu thuộc kiểu dữ liệu **int** thì nó sẽ là một số nguyên (*integer*) và nó có thể nhận giá trị từ - 32,768 (-2^{15}) đến 32,767 ($2^{15} - 1$) . Trên kiểu dữ liệu **int** ngôn ngữ C định nghĩa các phép toán số học đối với số nguyên như

Tên phép toán	**Kí hiệu**
Đảo dấu	-
Cộng	+
Trừ	-
Nhân	*
Chia lấy phần nguyên	/
Chia lấy phần dư	%
So sánh bằng	= =
So sánh lớn hơn	>
So sánh nhỏ hơn	<
…	

Trong máy tính, việc phân biệt kiểu dữ liệu là cần thiết vì qua kiểu dữ liệu máy tính biết được đối tượng mà nó đang xử lí thuộc dạng nào, có cấu trúc ra sao, có thể thực hiện các phép xử lí nào đối với đối tượng đó, hay là cần phải lưu trữ đối tượng đó như thế nào…

1.2.5. Hằng

Định nghĩa: hằng (*constant*) là đại lượng có giá trị không đổi trong chương trình. Để giúp chương trình dịch nhận biết hằng ta cần nắm được cách biểu diễn hằng trong một chương trình C.

Biểu diễn hằng số nguyên

Trong ngôn ngữ C, một hằng số nguyên có thể được biểu diễn dưới những dạng sau

- Dạng thập phân: đó chính là cách viết giá trị số đó dưới hệ đếm cơ số 10 thông thường.
- Dạng thập lục phân: ta viết giá trị số đó dưới dạng hệ đếm cơ số 16 và thêm tiền tố **0x** ở đầu.
- Dạng bát phân: ta viết giá trị số đó dưới dạng hệ đếm cơ số 8 và thêm tiền tố **0** ở đầu.

Ví dụ

Giá trị thập phân	**Giá trị hệ bát phân**	**Giá trị hệ thập lục phân**
2007	03727	0x7D7
396	0614	0x18C

Biểu diễn hằng số thực

Có 2 cách biểu diễn hằng số thực:

- Dưới dạng số thực dấu phẩy tĩnh.
- Dưới dạng số thực dấu phẩy động.

Ví dụ:

Số thực dấu phẩy tĩnh	**Số thực dấu phẩy động**
3.14159	31.4159 E-1
123.456	12.3456 E+1 hoặc 1.23456 E+2

Biểu diễn hằng kí tự

Có 2 cách biểu diễn hằng kí tự:

- Bằng kí hiệu của kí tự đó đặt giữa 2 dấu nháy đơn.
- Bằng số thứ tự của kí tự đó trong bảng mã ASCII (và lưu ý số thứ tự của một kí tự trong bảng mã ASCII là một số nguyên nên có một số cách biểu diễn).

Ví dụ

Kí tự cần biểu diễn	**Cách 1**	**Cách 2**
Chữ cái A	'A'	65 hoặc 0101 hoặc 0x41
Dấu nháy đơn '	'\''	39 hoặc 047 hoặc 0x27
Dấu nháy kép "	'\"'	34 hoặc 042 hoặc 0x22
Dấu gạch chéo ngược \	'\\'	92 hoặc 0134 hoặc 0x5c
Kí tự xuống dòng	'\n'	
Kí tự NUL	'\0'	0 hoặc 00 hoặc 0x0
Kí tự Tab	'\t'	9 hoặc 09 hoặc 0x9

Biểu diễn hằng xâu kí tự

Hằng xâu kí tự được biểu diễn bởi dãy các kí tự thành phần có trong xâu đó và được đặt trong cặp dấu nháy kép.

Ví dụ:

"ngon ngu lap trinh C", "tin hoc dai cuong"…

1.2.6. Câu lệnh

Câu lệnh (*statement*) diễn tả một hoặc một nhóm các thao tác trong giải thuật. Chương trình được tạo thành từ dãy các câu lệnh.

Cuối mỗi câu lệnh đều có dấu chấm phẩy ';' để đánh dấu kết thúc câu lệnh cũng như để phân tách các câu lệnh với nhau.

Câu lệnh được chia thành 2 nhóm chính:

- Nhóm các câu lệnh đơn: là những câu lệnh không chứa câu lệnh khác. Ví dụ: phép gán, phép cộng, phép trừ…
- Nhóm các câu lệnh phức: là những câu lệnh chứa câu lệnh khác trong nó. Ví dụ: lệnh khối, các cấu trúc lệnh rẽ nhánh, cấu trúc lệnh lặp…

Lệnh khối là một số các lệnh đơn được nhóm lại với nhau và đặt trong cặp dấu ngoặc nhọn { } để phân tách với các lệnh khác trong chương trình.

1.2.7. Chú thích

Để giúp việc đọc và hiểu chương trình viết ra được dễ dàng hơn, chúng ta cần đưa vào các lời chú thích (*comment*). Lời chú thích là lời mô tả, giải thích vắn tắt cho một câu lệnh, một đoạn chương trình hoặc cả chương trình, nhờ đó người đọc có thể hiểu được ý đồ của người lập trình và công việc mà chương trình đang thực hiện.

Lời chú thích chỉ có tác dụng duy nhất là giúp chương trình viết ra dễ đọc và dễ hiểu hơn, nó không phải là câu lệnh và nó hoàn toàn không ảnh hưởng gì đến hoạt động của chương trình.

Khi gặp kí hiệu lời chú thích trong chương trình, trình biên dịch sẽ tự động bỏ qua không dịch phần nội dung nằm trong phạm vi của vùng chú thích đó.

Trong C, có 2 cách để viết lời chú thích

- Dùng 2 dấu sổ chéo liên tiếp // để kí hiệu toàn bộ vùng bắt đầu từ 2 dấu sổ chéo liên tiếp đó đến cuối dòng là vùng chú thích. Ví dụ:

  ```
  // khai bao 2 bien nguyen
  int a, b;
  a = 5; b = 3;     // khoi tao gia tri cho cac bien nay
  ```

  Cách này thường dùng nếu đoạn chú thích ngắn, có thể viết đủ trên một dòng.
- Dùng 2 cặp kí hiệu /* và */ để kí hiệu rằng toàn bộ vùng bắt đầu từ cặp kí hiệu /* kéo dài đến cặp kí hiệu */ là vùng chú thích. Ví dụ:

  ```
  /* doan chuong trinh sau khai bao 2 bien nguyen va khoi tao gia tri cho 2 bien nguyen nay */
  int a, b;
  a = 5; b = 3;
  ```

  Cách này thường dùng khi đoạn chú thích dài, phải viết trên nhiều dòng.

1.3. Cấu trúc cơ bản của một chương trình C

Về cơ bản, mọi chương trình viết bằng ngôn ngữ C sẽ có cấu trúc gồm 6 phần có thứ tự như sau:

Khai báo tệp tiêu đề
#include

Định nghĩa kiểu dữ liệu
typedef ...

Khai báo các hàm nguyên mẫu

Khai báo các biến toàn cục

Định nghĩa hàm **main()**
main()
{
...
}

Định nghĩa các hàm đã khai báo nguyên mẫu

- Phần 1: Phần khai báo các tệp tiêu đề. Phần này có chức năng thông báo cho chương trình dịch biết là chương trình có sử dụng những thư viện nào (mỗi tệp tiêu đề tương ứng với một thư viện).
- Phần 2: Định nghĩa các kiểu dữ liệu mới dùng cho cả chương trình.
- Phần 3: Phần khai báo các hàm nguyên mẫu. Phần này giúp cho chương trình dịch biết được những thông tin cơ bản (gồm tên hàm, dach sách các tham số và kiểu dữ liệu trả về) của các hàm sử dụng trong chương trình.
- Phần 4: Phần khai báo các biến toàn cục.
- Phần 5: Phần định nghĩa hàm **main( )**. Hàm **main( )** là một hàm đặc biệt trong C. Khi thực hiện, chương trình sẽ gọi hàm **main( )**, hay nói cách khác chương trình sẽ bắt đầu bằng việc thực hiện các lệnh trong hàm **main( )**. Trong hàm **main( )** ta mới gọi tới các hàm khác.
- Phần 6: Phần định nghĩa các hàm đã khai báo nguyên mẫu. Ở phần 3 ta đã khai báo nguyên mẫu (*prototype*) của các hàm, trong đó chỉ giới thiệu các thông tin cơ bản về hàm như tên hàm, danh sách các tham số và kiểu dữ liệu trả về.

Nguyên mẫu hàm không cho ta biết cách thức cài đặt và hoạt động của các hàm. Ta sẽ làm việc đó ở phần định nghĩa các hàm.

Trong 6 phần trên, thì phần 5 định nghĩa hàm **main( )** bắt buộc phải có trong mọi chương trình C. Các phần khác có thể có hoặc không.

Dưới đây là ví dụ một chương trình viết trên ngôn ngữ C.

```
1.  // Chuong trinh sau se nhap vao tu ban phim 2 so nguyen
2.  // va hien thi ra man hinh tong, hieu tich cua 2 so nguyen vua nhap vao
3.  #include <stdio.h>
4.  #include <conio.h>
5.  void main()
6.  {
7.      // khai bao cac bien trong chuong trinh
8.      int a, b
9.      int tong, hieu, tich;
10.     // Nhap vao tu ban phim 2 so nguyen
11.     printf("\n Nhap vao so nguyen thu nhat: ");
12.     scanf("%d",&a);
13.     printf("\n Nhap vao so nguyen thu hai: ");
14.     scanf("%d",&b);
15.     // Tinh tong, hieu, tich cua 2 so vua nhap
16.     tong = a+b;
17.     hieu = a – b;
18.     tich = a*b;
19.     // Hien thi cac gia tri ra man hinh
20.     printf("\n Tong cua 2 so vua nhap la %d", tong);
21.     printf("\n Hieu cua 2 so vua nhap la %d", hieu);
22.     printf("\n Tich cua 2 so vua nhap la %d", tich);
23.     // Cho nguoi su dung an phim bat ki de ket thuc
24.     getch();
25. }
```

Trong chương trình trên chỉ có 2 phần là khai báo các thư viện và định nghĩa hàm **main( )**. Các phần khai báo hàm nguyên mẫu, khai báo biến toàn cục và định nghĩa hàm nguyên mẫu không có trong chương trình này.

Các dòng 1, 2 là các dòng chú thích mô tả khái quát công việc chương trình sẽ thực hiện.

Dòng thứ 3 và thứ 4 là khai báo các tệp tiêu đề. Bởi vì trong chương trình ta sử dụng các hàm printf() (nằm trong thư viện **stdio** – *standard input/output*, thư viện chứa các hàm thực hiện các thao tác vào ra chuẩn) và getch() (nằm trong thư viện **conio** – *console input/output*, thư viện chứa các hàm thực hiện các thao tác vào ra qua bàn phím, màn hình…) nên ta phải khai báo với chương trình dịch gộp các thư viện đó vào chương trình. Nếu ta không gộp thư viện vào trong chương trình thì ta sẽ không thể sử dụng các hàm có trong thư viện đó.

Để gộp một thư viện vào trong chương trình (nhờ đó ta có thể sử dụng các hàm của thư viện đó), ta khai báo tệp tiêu đề tương ứng với thư viện đó ở đầu chương trình bằng chỉ thị có mẫu sau:

#include <tên_tệp_tiêu_đề>

Ví dụ để gộp thư viện conio vào chương trình ta dùng chỉ thị

#include <conio.h>

Lưu ý: Các tệp tiêu đề có tên là tên của thư viện, có phần mở rộng là .h (viết tắt của từ *header*).

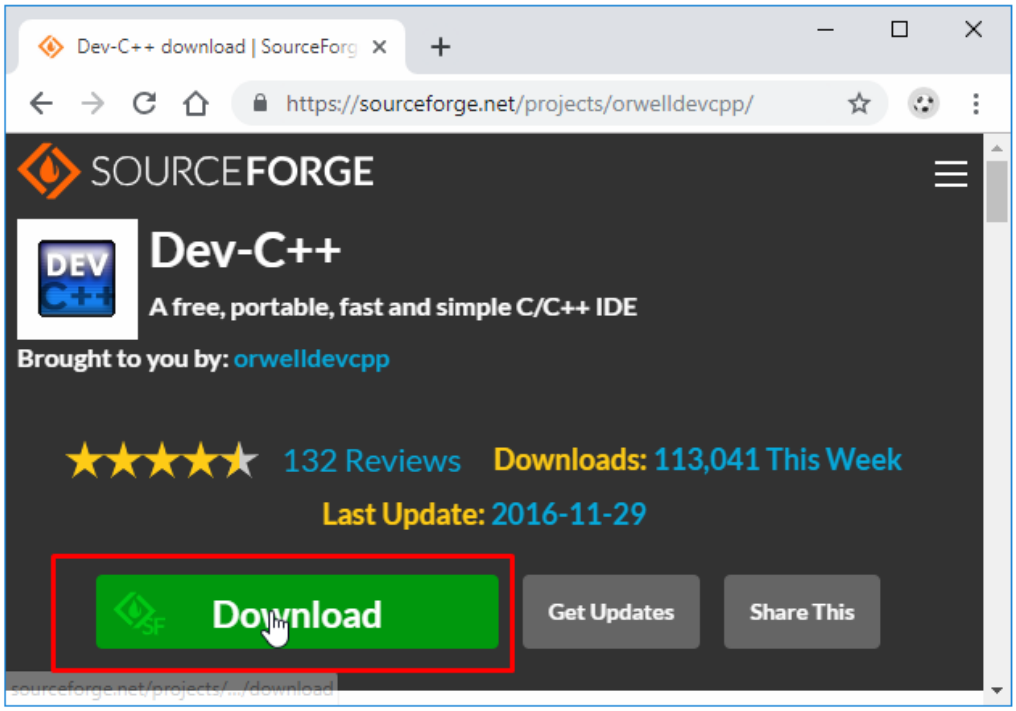
Các dòng tiếp theo (từ dòng thứ 5 đến dòng thứ 25) là phần định nghĩa hàm **main()**, trong đó các dòng 7, 10, 15, 19, 23 là các dòng chú thích mô tả công việc mà các câu lệnh sau đó sẽ thực hiện.

1.4. Biên dịch chương trình viết bằng ngôn ngữ C

1.4.1 Hướng dẫn tải và cài đặt Dev C++

Để tải phần mềm Dev C++ về máy, bạn cần thực hiện theo các bước sau:

- Bước 1: Truy cập vào đường link http://www.bloodshed.net/dev/devcpp.html để tải phần mềm. Kéo trang xuống và tìm đến mục SourceForg, lựa chọn 1 trang để tải về. File tải về sẽ có dạng devcpp4.9.9.2_setup.exe.
- Bước 2: Sau khi hoàn thành quá trình tải phần mềm, hãy mở phần mềm và bắt đầu cài đặt. Lúc này, một số ô cửa sổ sẽ hiển thị yêu cầu bạn chọn ngôn ngữ, đồng ý với các điều khoản sử dụng.
- Bước 3: Phần mềm sẽ yêu cầu bạn lựa chọn vị trí để cài đặt. Thông thường, vị trí mặc định sẽ là ổ C. Sau đó, quá trình cài đặt sẽ được bắt đầu.
- Bước 4: Khi màn hình xuất hiện câu hỏi bạn có muốn cài đặt Dev C++ cho tất cả người dùng không, hãy chọn Yes. Nếu sau đó quá trình cài đặt không thành công, bạn có thể quay lại và chọn No ở bước này.



- Bước 5: Nếu quá trình cài đặt đã được hoàn tất, 1 màn hình thông báo sẽ được hiện lên. Lúc này, chọn Finish -> English -> Newlook -> Yes -> Nhấp OK để hoàn thành quá trình cài đặt.

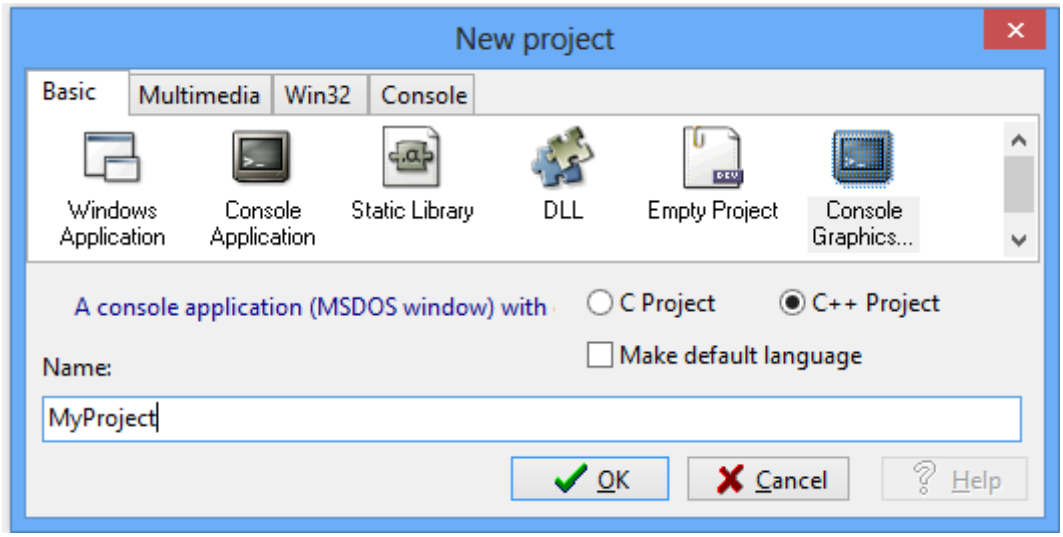
1.4.2 Cách sử dụng Dev C++

Nhìn chung, Dev C++ có giao diện sử dụng khá dễ dàng. Bạn sẽ không phải mất quá nhiều thời gian để làm quen với phần mềm này. Dưới đây là những thao tác cơ bản bạn cần lưu ý.

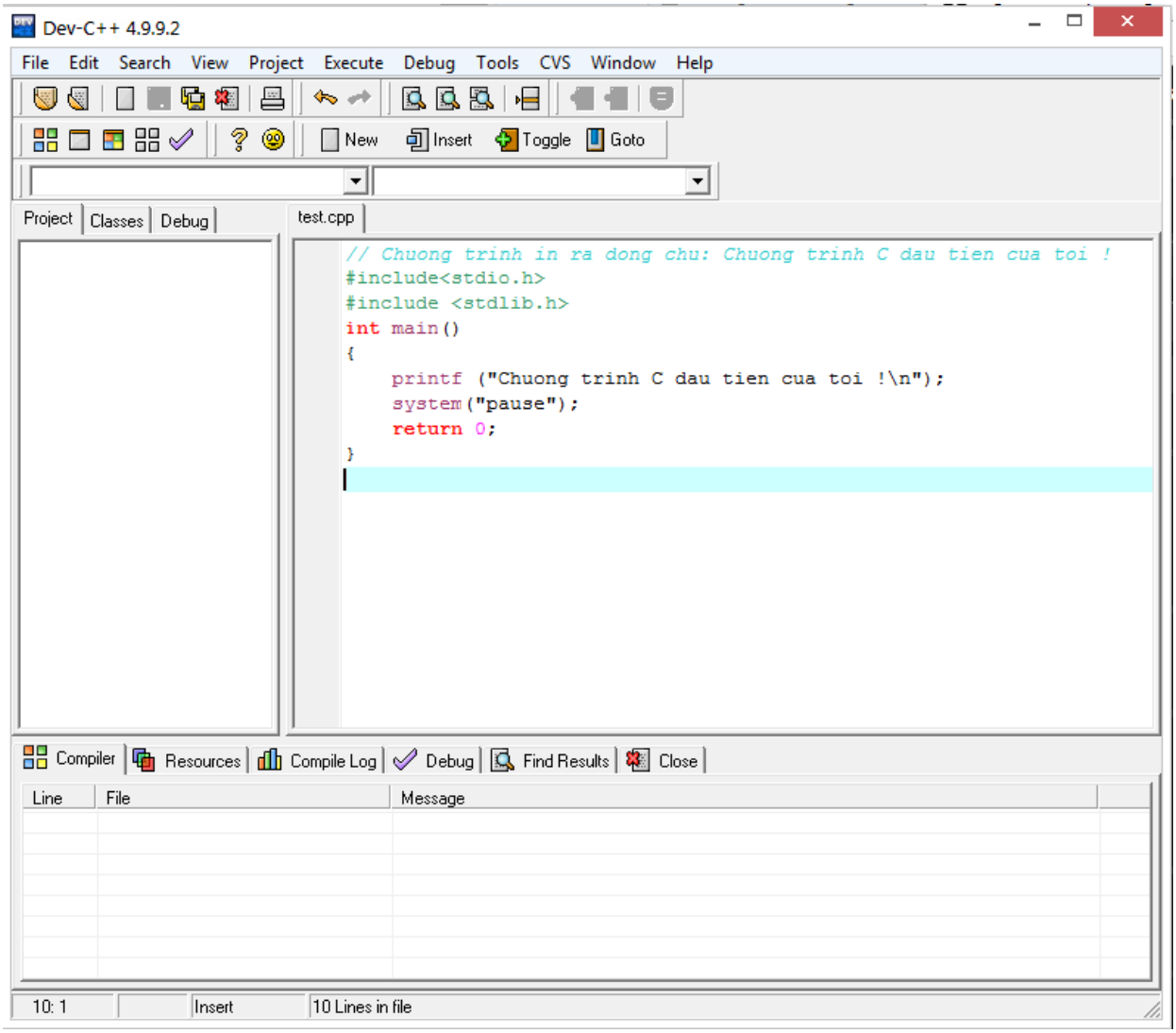
Tạo mới 1 project

Để tạo mới 1 project, bạn thực hiện theo trình tự các bước sau:

- Bước 1: Khởi động chương trình
- Bước 2: Tại vị trí trên cùng bên tay trái màn hình, chọn File -> New -> New Project. 1 cửa sổ mới được hiện ra yêu cầu bạn nhập tên Project, tích chọn mục C Project và cuối cùng nhấn OK để hoàn thành.



- Bước 3: Trong mục Create New Project -> Save
- Bước 4: Tiếp tục vào File -> New -> Source File -> Add source file to current project -> Yes để hiển thị màn hình chỉnh sửa file nguồn.



- Bước 5: Thử nhập và chạy một chương trình đơn giản. Sau đó vào File -> Save As. Lưu ý phần mở rộng của file nên đặt là ".c". Các phần mở rộng khác có thể gặp sự cố khi biên dịch.

Thêm các tệp nguồn

Bạn có thể chuyển đến menu -> chọn tệp -> chọn tệp mới, hoặc sử dụng phím tắt Ctrl + N. Sau đó vào lại menu -> Project -> Chọn 1 tệp mới.

Lưu ý: Dev C++ sẽ không yêu cầu bạn thêm bất kỳ tệp nguồn nào cho đến khi bạn thực hiện các thao tác biên dịch, lưu project, thoát Dev hay lưu tệp nguồn.

Biên dịch

Để thực hiện biên dịch, bạn chỉ cần nhấn thực hiện hoặc Ctrl + F9. Nếu có liên kết lỗi, linker tab sẽ nhấp nháy ở vị trí có lỗi. Thông thường, các lỗi liên kết đều là lỗi cú pháp không cho phép một trong các tệp được biên dịch.

Thực hiện

Bạn nhấp vào Execute -> Compile and Run để chạy thử chương trình.

Chỉnh sửa

Lập trình với những câu lệnh phức tạp thì khó có thể tránh được các lỗi. Nếu như khi chạy thử chương trình mà kết quả không như ý muốn của mình, trình gỡ lỗi sẽ là "cứu cánh" của bạn vào thời điểm này đấy.

Tab gỡ lỗi trên màn hình sẽ giúp bạn xác định được vị trí bạn cần chỉnh sửa câu lệnh.

BÀI 2

BIẾN, MẢNG VÀ BIỂU THỨC

2.1. Các kiểu dữ liệu chuẩn trong C

Kiểu dữ liệu	Ý nghĩa	Kích thước	Miền biểu diễn
unsigned char	Kí tự	1 byte	0 ÷ 255
char	Kí tự	1 byte	-128 ÷ 127
unsigned int	Số nguyên không dấu	2 bytes	0 ÷ 65,535
short int	Số nguyên có dấu	2 bytes	-32,768 ÷ 32,767
Int	Số nguyên có dấu	2 bytes	- 32,768 ÷ 32,767
unsigned long	Số nguyên không dấu	4 bytes	0 ÷ 4,294,967,295
long	Số nguyên có dấu	4 bytes	-2,147,483,648 ÷ 2,147,483,647
float	Số thực dấu phẩy động, độ chính xác đơn	4 bytes	± 3.4E-38 ÷ ± 3.4E+38
double	Số thực dấu phẩy động độ chính xác kép	8 bytes	± 1.7E-308 ÷ ± 1.7E+308
long double	Số thực dấu phẩy động	10 bytes	± 3.4E-4932 ÷ ± 1.1E+4932

- **Khai báo và sử dụng biến, hằng**

Khai báo và sử dụng biến

Một biến trước khi sử dụng phải được khai báo. Cú pháp khai báo:

kiểu_dữ_liệu tên_biến;

Ví dụ:

```
float x;   // biến kiểu thực
float y;   // biến kiểu thực
double z;      // biến kiểu thực
int i;      // biến kiểu nguyên
```

int j; // biến kiểu nguyên

Nếu các biến thuộc cùng kiểu dữ liệu thì C cho phép khai báo chúng trên cùng một dòng:

kiểu_dữ_liệu danh_sách_tên_biến;

Ví dụ:

float x, y;

double z;

int i, j;

Sau khi khai báo, biến có thể nhận giá trị thuộc kiểu dữ liệu đã khai báo. Chúng ta có thể khởi tạo giá trị đầu cho biến nếu muốn với cú pháp:

kiểu_dữ_liệu tên_biến = giá_trị_đầu;

Ví dụ:

int a = 3;// sau lenh nay bien a se co gia tri bang 3

float x = 5.0, y = 2.6;// sau lenh nay x co gia tri 5.0, y co gia tri 2.6

Biến dùng để lưu giữ giá trị, dùng làm toán hạng trong biểu thức, làm tham số cho hàm, làm biến chỉ số cho các cấu trúc lặp (**for**, **while**, **do while**), làm biểu thức điều kiện trong các cấu trúc rẽ nhánh (**if**, **switch**)…

Khai báo hằng

Có 2 cách để khai báo hằng trong C là dùng chỉ thị **#define** hoặc khai báo với từ khóa **const**.

Dùng chỉ thị **#define**

Cú pháp khai báo:

 # define tên_hằng giá_trị

Lưu ý không có dấu chấm phẩy ở cuối dòng chỉ thị.

Với cách khai báo này, mỗi khi chương trình dịch gặp tên_hằng trong chương trình, nó sẽ tự động thay thế bằng giá_trị. Ở đây kiểu dữ liệu của hằng tự động được chương trình dịch xác định dựa theo nội dung của giá_trị.

Ví dụ:

```
#define MAX_SINH_VIEN  50                  // hằng kiểu số nguyên
#define CNTT “Cong nghe thong tin”         // hằng kiểu xâu kí tự (string)
#define DIEM_CHUAN  23.5                   // hằng kiểu số thực
```

Dùng từ khóa **const** để khai báo với cú pháp:

const kiểu_dữ_liệu tên_hằng = giá_trị;

Khai báo này giống với khai báo biến có khởi tạo giá trị đầu, tuy nhiên cần lưu ý:

- Do có từ khóa **const** ở đầu cho nên giá trị của đối tượng tên_hằng sẽ không được phép thay đổi trong chương trình. Những lệnh nhằm làm thay đổi giá trị của tên_hằng trong chương trình sẽ dẫn tới lỗi biên dịch.
- Trong khai báo biến thông thường, người lập trình có thể khởi tạo giá trị cho biến ngay từ khi khai báo hoặc không khởi tạo cũng được. Nhưng trong khai báo hằng, giá trị của tất cả các hằng cần được xác định ngay trong lệnh khai báo.

Các khai báo hằng ở ví dụ trước giờ có thể viết lại theo cách khác như sau:

```
const int  MAX_SINH_VIEN  = 50;
const char  CNTT[20] = “Cong nghe thong tin”;
const float  DIEM_CHUAN = 23.5;
```

Kiểu Enum

Enum trong C là kiểu dữ liệu do người dùng định nghĩa. Nó được sử dụng chủ yếu để gán các tên cho các hằng số, các tên giúp một chương trình dễ đọc và bảo trì.

Từ khóa '**enum**' được sử dụng để khai báo các kiểu liệt kê mới trong C và C++. Sau đây là cú pháp về khai báo enum.

Cú pháp:

```
enum enum_name{constant1, constant2, constant3, .....};
```

Các biến enum có thể được định nghĩa theo 2 cách:

```
// trong ca 2 truong hop duoi day, "day"
// duoc dinh nghia nhu bien cua kieu week
enum week{Mon, Tue, Wed};
enum week day;
 // hoac
 enum week{Mon, Tue, Wed} day;
```

Ví dụ sử dụng enum trong C:

```
1   #include<stdio.h>
2
3   enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
4
5   int main() {
6       enum week day;
7       day = Wed;
8       printf("Chi so cua Wed la %d\n", day);
9
```

```
10          // duyet tat ca cac chi so phan tu cua enum week
11          int i;
12          for (i = Mon; i <= Sun; i++)
13              printf("%d ", i);
14          return 0;
15      }
```

Kết quả:

```
Chi so cua Wed la 2
0 1 2 3 4 5 6
```

Trong ví dụ này, vòng lặp for sẽ chạy từ i = 0 đến i = 6, như ban đầu giá trị của i là Mon là 0 và giá trị của tháng Sun là 6.

2.2. Biểu thức trong C

Khái niệm biểu thức ta đã đề cập ở phần 1.2.8. Ở phần này ta sẽ trình bày về các loại biểu thức trong C.

Biểu thức số học

Biểu thức số học là biểu thức mà giá trị của nó là cái đại lượng số học (số nguyên, số thực). Trong biểu thức số học, các toán tử là các phép toán số học (cộng, trừ, nhân, chia…), các toán hạng là các đại lượng số học. Ví dụ (giả sử a, b, c là các số thực)

3 * 3.7, 8 + 6/3, a + b – c…

Biểu thức logic

Biểu thức logic là biểu thức mà giá trị của nó là các giá trị logic, tức là một trong hai giá trị: Đúng (*TRUE*) hoặc Sai (*FALSE*).

Ngôn ngữ C coi các giá trị nguyên khác 0 (ví dụ 1, -2, -5) là giá trị logic Đúng (TRUE), giá trị 0 là giá trị logic Sai (FALSE).

Các phép toán logic gồm có

- AND (VÀ logic, trong ngôn ngữ C được kí hiệu là &&)
- OR (HOẶC logic, trong ngôn ngữ C được kí hiệu là ||)
- NOT (PHỦ ĐỊNH, trong ngôn ngữ C kí hiệu là !)

Biểu thức quan hệ

Biểu thức quan hệ là những biểu thức trong đó có sử dụng các toán tử quan hệ so sánh như lớn hơn, nhỏ hơn, bằng nhau, khác nhau… Biểu thức quan hệ cũng chỉ có thể nhận giá trị là một trong 2 giá trị Đúng (TRUE) hoặc Sai (FALSE), vì vậy biểu thức quan hệ là một trường hợp riêng của biểu thức logic. Ví dụ về biểu thức quan hệ

```
5 > 7                   // có giá trị logic là sai, FALSE
```

```
9 != 10                 // có giá trị logic là đúng, TRUE
2 >= 2                  // có giá trị logic là đúng, TRUE
a > b                   // giả sử a, b là 2 biến kiểu int
a+1 > a                 // có giá trị đúng, TRUE
```

Ví dụ về biểu thức logic

```
(5 > 7)&&(9!=10)        // có giá trị logic là sai, FALSE
0 || 1                  // có giá trị logic là đúng, TRUE
(5 > 7)||(9!=10)        // có giá trị logic là đúng, TRUE
0                       // có giá trị logic là sai, FALSE
!0                      // phủ định của 0, có giá trị logic là đúng, TRUE
3                       // có giá trị logic là đúng, TRUE
!3                      // phủ định của 3, có giá trị logic là sai, FALSE
(a > b)&&( a < b)       // Có giá trị sai, FALSE. Giả sử a, b là 2 biến kiểu int
```

Sử dụng biểu thức

Trong chương trình, biểu thức được sử dụng cho các mục đích sau:

- Làm vế phải của lệnh gán (sẽ đề cập ở mục sau).
- Làm toán hạng trong các biểu thức khác.
- Làm tham số thực trong lời gọi hàm.
- Làm chỉ số trong các cấu trúc lặp **for**, **while**, **do while**.
- Làm biểu thức kiểm tra trong các cấu trúc rẽ nhánh **if**, **switch**.

2.2.1. Các phép toán (toán tử)

Các phép toán trong C được chia thành các nhóm phép toán cơ bản sau: nhóm các phép toán số học, nhóm các phép toán thao tác trên bit, nhóm các phép toán quan hệ, nhóm các phép toán logic. Ngoài ra C còn cung cấp một số phép toán khác nữa như phép gán, phép lấy địa chỉ…

2.2.2. Phép toán số học

Các phép toán số học (*Arithmetic operators*) gồm có:

Toán tử	**Ý nghĩa**	**Kiểu dữ liệu của toán hạng**	**Ví dụ**
-	Phép đổi dấu	Số thực hoặc số nguyên	int a, b; -12; -a; -25.6;

+	Phép toán cộng	Số thực hoặc số nguyên	float x, y; 5 + 8; a + x; 3.6 + 2.9;
-	Phép toán trừ	Số thực hoặc số nguyên	3 – 1.6; a – 5;
*	Phép toán nhân	Số thực hoặc số nguyên	a * b; b * y; 2.6 * 1.7;
/	Phép toán chia	Số thực hoặc số nguyên	10.0/3.0;(bằng 3.33…) 10/3.0; (bằng 3.33…) 10.0/3; (bằng 3.33…)
/	Phép chia lấy phần nguyên	Giữa 2 số nguyên	10/3; (bằng 3)
%	Phép chia lấy phần dư	Giữa 2 số nguyên	10%3; (bằng 1)

Các phép toán trên bit

Toán tử	**Ý nghĩa**	**Kiểu dữ liệu của toán hạng**	**Ví dụ**
&	Phép VÀ nhị phân	2 số nhị phân	0 & 0 (có giá trị 0) 0 & 1 (có giá trị 0) 1 & 0 (có giá trị 0) 1 & 1 (có giá trị 1) 101 & 110 (có giá trị 100)
|	Phép HOẶC nhị phân	2 số nhị phân	0 | 0 (có giá trị 0) 0 | 1 (có giá trị 0) 1 | 0 (có giá trị 0) 1 | 1 (có giá trị 1) 101 | 110 (có giá trị 111)
^	Phép HOẶC CÓ LOẠI TRỪ nhị phân	2 số nhị phân	0 ^ 0 (có giá trị 0) 0 ^1 (có giá trị 1) 1 ^ 0 (có giá trị 1) 1 ^ 1 (có giá trị 0) 101 ^ 110 (có giá trị 011)
<<	Phép DỊCH TRÁI nhị phân	Số nhị phân	a << n (có giá trị $a*2^n$) 101 << 2 (có giá trị 10100)

>>	Phép DỊCH PHẢI nhị phân	Số nhị phân	a >> n (có giá trị $a/2^n$) 101 >> 2 (có giá trị 1)
~	Phép ĐẢO BIT nhị phân (lấy Bù 1)	Số nhị phân	~ 0 (có giá trị 1) ~ 1 (có giá trị 0) ~ 110 (có giá trị 001)

2.2.3. Phép toán quan hệ

Các phép toán quan hệ (*Comparison operators*) gồm có:

Toán tử	Ý nghĩa	Ví dụ
>	So sánh lớn hơn giữa 2 số nguyên hoặc thực.	2 > 3 (có giá trị 0) 6 > 4 (có giá trị 1) a > b
>=	So sánh lớn hơn hoặc bằng giữa 2 số nguyên hoặc thực.	6 >= 4 (có giá trị 1) x >= a
<	So sánh nhỏ hơn giữa 2 số nguyên hoặc thực.	5 < 3 (có giá trị 0),
<=	So sánh nhỏ hơn hoặc bằng giữa 2 số nguyên hoặc thực.	5 <= 5 (có giá trị 1) 2 <= 9 (có giá trị 1)
==	So sánh bằng nhau giữa 2 số nguyên hoặc thực.	3 == 4 (có giá trị 0) a == b
!=	So sánh không bằng (so sánh khác) giữa 2 số nguyên hoặc thực.	5 != 6 (có giá trị 1) 6 != 6 (có giá trị 0)

2.2.4. Các phép toán logic

Các phép toán logic (*Logical operators*) gồm có

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
&&	Phép VÀ LOGIC. Biểu thức VÀ LOGIC bằng 1 khi và chỉ khi cả 2 toán hạng đều bằng 1	Hai biểu thức logic	3<5 && 4<6 (có giá trị 1) 2<1 && 2<3 (có giá trị 0) a > b && c < d

||	Phép HOẶC LOGIC. Biểu thức HOẶC LOGIC bằng 0 khi và chỉ khi cả 2 toán hạng bằng 0.	Hai biểu thức logic	6 || 0 (có giá trị 1) 3<2 || 3<3 (có giá trị 1) x >= a || x == 0
!	Phép PHỦ ĐỊNH LOGIC một ngôi. Biểu thức PHỦ ĐỊNH LOGIC có giá trị bằng 1 nếu toán hạng bằng 0 và có giá trị bằng 0 nếu toán hạng bằng 1	Biểu thức logic	!3 (có giá trị 0) !(2>5) (có giá trị 1)

2.2.5. Phép toán gán

Phép toán gán có dạng

tên_biến = biểu_thức;

Phép toán gán có chức năng lấy giá trị của biểu_thức gán cho tên_biến. Dấu = là kí hiệu cho toán tử gán.

Ví dụ:

```
int a, b, c;
a = 3;
b = a + 5;
c = a * b;
```

Sau đoạn lệnh trên, biến a có giá trị là 3, b có giá trị là 8 và c có giá trị là 24.

Trong phép toán gán nếu ta bỏ dấu ; ở cuối đi thì ta sẽ thu được biểu thức gán. Biểu thức gán là biểu thức có dạng

tên_biến = biểu_thức

Biểu thức gán là biểu thức nên nó cũng có giá trị. Giá trị của biểu thức gán bằng giá trị của biểu_thức, do đó ta có thể gán giá trị của biểu thức gán cho một biến khác hoặc sử dụng như một biểu thức bình thường. Ví dụ

```
int a, b, c;
a = b = 2007;
c = (a = 20) * (b = 30);
```

Trong câu lệnh thứ 2, ta đã gán giá trị 2007 cho biến *b*, sau đó ta gán giá trị của biểu thức *b* = 2007 cho biến a. Giá trị của biểu thức *b* = 2007 là 2007, do đó kết thúc câu lệnh này ta có *a* bằng 2007, *b* bằng 2007.

Trong câu lệnh thứ 3, ta gán giá trị 20 cho a, gán giá trị 30 cho b. Sau đó ta tính giá trị của biểu thức tích (a = 20) * (b = 30) từ giá trị các biểu thức con *a* = 20 (có giá trị là 20) và *b* = 30 (có giá trị là 30). Cuối cùng ta gán giá trị của biểu thức tích thu được (600) cho biến *c*;

Phép toán gán thu gọn

Xét lệnh gán sau

x = x + y;

Lệnh gán này sẽ tăng giá trị của biến *x* thêm một lượng có giá trị bằng giá trị của *y*. Trong C ta có thể viết lại lệnh này một cách gọn hơn mà thu được kết quả tương đương

x += y;

Dạng lệnh gán thu gọn này còn áp dụng được với các phép toán khác nữa.

Lệnh gán thông thường	Lệnh gán thu gọn
x = x + y	x + = y
x = x - y	x - = y
x = x * y	x * = y
x = x / y	x / = y
x = x % y	x % = y
x = x >> y	x >> = y
x = x << y	x << = y
x = x & y	x & = y
x = x | y	x | = y
x = x ^ y	x^+ = y

2.2.6. Thứ tự ưu tiên các phép toán

Khái niệm thứ tự ưu tiên (operator precedence) của phép toán

Thứ tự ưu tiên của các phép toán dùng để xác định trật tự kết hợp các toán hạng với các toán tử khi tính toán giá trị của biểu thức.

Bảng thứ tự ưu tiên của các phép toán trong C

Mức	Các toán tử	Trật tự kết hợp
1	() [] . -> ++ (hậu tố) -- hậu tố	------>
2	! ~ ++ (tiền tố) -- (tiền tố) - * & sizeof	<------

3	* / %	------>
4	+ -	------>
5	<< >>	------>
6	< <= > >=	------>
7	== !=	------>
8	&	------>
9	^	------>
10	|	------>
11	&&	------>
12	||	------>
13	?:	<------
14	= += -=	<------

Ghi chú: ------> trật tự kết hợp từ trái qua phải

<------ trật tự kết hợp từ phải qua trái.

Nguyên tắc xác định trật tự thực hiện các phép toán

i. Biểu thức con trong ngoặc được tính toán trước các phép toán khác

ii. Phép toán một ngôi đứng bên trái toán hạng được kết hợp với toán hạng đi liền nó.

iii. Nếu toán hạng đứng cạnh hai toán tử thì có 2 khả năng là

a. Nếu hai toán tử có độ ưu tiên khác nhau thì toán tử nào có độ ưu tiên cao hơn sẽ kết hợp với toán hạng

b. Nếu hai toán tử cùng độ ưu tiên thì dựa vào trật tự kết hợp của các toán tử để xác định toán tử được kết hợp với toán hạng.

2.2.7. Một số toán tử đặc trưng của C

Các phép toán tăng giảm một đơn vị

Trong lập trình chúng ta thường xuyên gặp những câu lệnh tăng (hoặc giảm) giá trị của một biến thêm (đi) một đơn vị. Để làm điều đó chúng ta dùng lệnh sau

<tên biến> = <tên biến> + 1;

<tên biến> = <tên biến> - 1;

Ta cũng có thể tăng (hoặc giảm) giá trị của một biến bằng cách sử dụng hai phép toán đặc biệt của C là phép toán ++ và phép toán -- . Phép toán ++ sẽ tăng giá trị của biến thêm 1 đơn vị, phép toán -- sẽ giảm giá trị của biến đi 1 đơn vị. Ví dụ

```
int a = 5;
float x = 10;
```

a ++; // lệnh này tương đương với a = a+1 ;

x --;// tương đương với x = x – 1;

Phép toán tăng, giảm một đơn vị ở ví dụ trên là dạng hậu tố (vì phép toán đứng sau toán hạng). Ngoài ra còn có dạng tiền tố của phép toán tăng, giảm một đơn vị. Trong dạng tiền tố, ta thay đổi giá trị của biến trước khi sử dụng biến đó để tính toán giá trị của biểu thức. Trong dạng hậu tố, ta tính toán giá trị của biểu thức bằng giá trị ban đầu của biến, sau đó mới thay đổi giá trị của biến.

Ví dụ

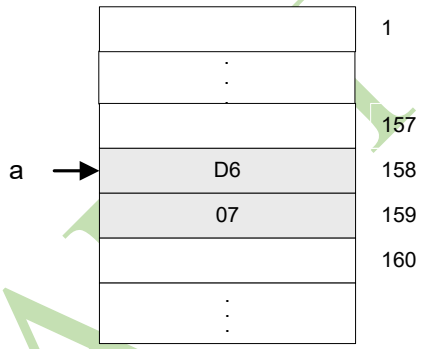
```
int a, b, c;
a = 3;              // a bằng 3
b = a++;    // dạng hậu tố. b bằng 3; a bằng 4
c = ++b;    // dạng tiền tố. b bằng 4, c bằng 4;
```

Sau khi thực hiện đoạn chương trình trên, ta có *a*, *b* và *c* đều có giá trị bằng 4;

Phép toán lấy địa chỉ biến (&)

Một biến thực chất là một vùng nhớ được đặt tên (là tên của biến) trên bộ nhớ của máy tính. Mọi ô nhớ trên bộ nhớ máy tính đều được đánh địa chỉ. Do đó mọi biến đều có địa chỉ.

Địa chỉ của một biến được định nghĩa là địa chỉ của ô nhớ đầu tiên trong vùng nhớ dành cho biến đó. Hình dưới đây minh họa một biến tên là *a*, kiểu dữ liệu **int** được lưu trữ trong bộ nhớ tại 2 ô nhớ có địa chỉ lần lượt là 158 và 159. Giá trị của biến *a* là 2006 = 0x07D6. Khi đó địa chỉ của biến *a* sẽ là 158 hay 0000:9E (vì địa chỉ được mã hóa bởi 2 byte).



Trong C để xác đinh địa chỉ của một biến ta sử dụng toán tử một ngôi & đặt trước tên biến, cú pháp là

& <tên biến>;

Ví dụ

```
&a; // có giá trị là 158 hay 9E
```

Phép toán chuyển đổi kiểu bắt buộc

Chuyển đổi kiểu là chuyển kiểu dữ liệu của một biến từ kiểu dữ liệu này sang kiểu dữ liệu khác. Cú pháp của lệnh chuyển kiểu dữ liệu là như sau:

(<kiểu dữ liệu mới>) <biểu thức>;

Có những sự chuyển đổi được thực hiện hết sức tự nhiên, không có khó khăn gì, thậm chí đôi khi chương trình dịch sẽ tự động chuyển đổi kiểu hộ cho ta, ví dụ chuyển một dữ liệu kiểu số nguyên **int** sang một số nguyên kiểu **long int**, hay từ một số **long int** sang một số thực **float**… Đó là vì một số nguyên kiểu **int** thực ra cũng là một số nguyên kiểu **long int**, một số nguyên kiểu **long int** cũng chính là một số thực kiểu **float**, một số thực kiểu **float** cũng là một số thực kiểu **double**.

Tuy nhiên điều ngược lại thì chưa chắc, ví dụ số nguyên **long int** 50,000 không phải là một số nguyên kiểu **int** vì phạm vi biểu diễn của kiểu **int** là từ (-32,768 đến 32,767). Khi đó nếu phải chuyển kiểu dữ liệu thì ta phải cẩn thận nếu không sẽ bị mất dữ liệu. Ví dụ một số thực **float** khi chuyển sang kiểu số nguyên **int** sẽ bị loại bỏ phần thập phân, còn một số nguyên kiểu **long int** khi chuyển sang kiểu **int** sẽ nhiều khả năng thu được một giá trị xa lạ. Ví dụ 1,193,046 (0x123456) là một số kiểu **long int**, khi chuyển sang kiểu **int** sẽ thu được 13,398 (0x3456). Đoạn chương trình sau minh họa điều đó.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    long int li;
    int i;
    float f;
    clrscr();
    li = 0x123456;
    f = 123.456;
    i = (int) li;
    printf("\n li = %ld",li);
    printf("\n i  = %d",i);
    i = (int) f;
    printf("\n f  = %f",f);
    printf("\n i  = %d",i);
    getch();
```

}

Kết quả

```
li = 1193046
i = 13398
f = 123.456001
i = 123
```

C hỗ trợ chuyển kiểu tự động trong những trường hợp sau

char → int → long int → float → double → long double

Biểu thức điều kiện

Là biểu thức có dạng

biểu_thức_1 ? biểu_thức_2 : biểu_thức_3

Giá trị của biểu thức điều kiện sẽ là giá trị của biểu_thức_2 nếu biểu_thức_1 có giá trị khác 0 (tương ứng với giá trị logic ĐÚNG), và trái lại giá trị của biểu thức điều kiện sẽ là giá trị của biểu_thức_3 nếu biểu_thức_1 có giá trị bằng 0 (tương ứng với giá trị logic SAI).

Ví dụ sau sẽ cho ta xác định được giá trị nhỏ nhất của 2 số nhờ sử dụng biểu thức điều kiện

```
float x, y, z;              // khai báo biến
x = 3.8; y = 2.6;           // gán giá trị cho các biến x, y
z = (x<y) ? x : y;  // z sẽ có giá trị bằng giá trị
                                    // nhỏ nhất trong 2 số x và y
```

Lệnh dãy

Lệnh dãy là lệnh gồm một dãy các biểu thức phân cách nhau bằng dấu phẩy và kết thúc lệnh là dấu chấm phẩy. Nó có dạng

biểu_thức_1, biểu_thức_2, …, biểu_thức_n;

Trong lệnh dãy các biểu thức được tính toán độc lập với nhau.

2.3. Mảng

2.3.1. Khái niệm mảng

Khái niệm mảng

Mảng là một tập hợp hữu hạn các phần tử có cùng kiểu dữ liệu được lưu trữ kế tiếp nhau trong bộ nhớ. Các phần tử trong mảng có cùng tên (và cũng là tên mảng) nhưng phân biệt với nhau ở chỉ số cho biết vị trí của chúng trong mảng.

2.3.2. Khai báo và sử dụng mảng

Cú pháp khai báo

Trong C để khai báo một mảng ta sử dụng cú pháp khai báo sau:

kiểu_dữ_liệu tên_mảng [kích_thước_mảng];

Trong đó kiểu_dữ_liệu là kiểu dữ liệu của các phần tử trong mảng. tên_mảng là tên của mảng. kích_thước_mảng cho biết số phần tử trong mảng.

Ví dụ:

```
int mang_nguyen[10];// khai báo mảng 10 phần tử có kiểu dữ liệu int
float mang_thuc[4];    // khai báo mảng 4 phần tử có kiểu dữ liệu float
char mang_ki_tu[6];    // khai báo mảng 6 phần tử có kiểu dữ liệu char
```

Trong ví dụ trên, mảng mang_nguyen được lưu trữ trên 20 ô nhớ (mỗi ô nhớ có kích thước 1 byte, 2 ô nhớ kích thước là 2 byte lưu trữ được một số nguyên kiểu **int**) liên tiếp nhau. Do C đánh số các phần tử của mảng bắt đầu từ 0 nên phần tử thứ i của mảng sẽ có chỉ số là i-1 và do vậy sẽ có tên là mang_nguyen[i-1]. Ví dụ: phần tử thứ nhất của mảng là mang_nguyen[0], phần tử thứ 2 là mang_nguyen[1], phần tử thứ 5 là mang_nguyen[4]…

mang_nguyen[0]	mang_nguyen[1]	...	...	mang_nguyen[9]

Kích thước của mảng bằng kích thước một phần tử nhân với số phần tử.

Mảng một chiều và nhiều chiều

Các mảng trong ví dụ trên là các mảng một chiều. Mảng là tập hợp các phần tử cùng kiểu dữ liệu, nếu mỗi phần tử của mảng cũng là một mảng khác thì khi đó ta có mảng nhiều chiều. Khái niệm mảng rất giống với khác niệm vector trong toán học.

Ví dụ sau khai báo một mảng gồm 6 phần tử, trong đó mỗi phần tử lại là một mảng gồm 5 số nguyên kiểu **int**. Mảng này là mảng 2 chiều

```
int a [6][5];
```

Còn khai báo

```
int b [3][4][5];
```

thì lại khai báo một mảng gồm 3 phần tử, mỗi phần tử lại một mảng 2 chiều gồm 4 phần tử. Mỗi phần tử của mảng 2 chiều lại là một mảng (1 chiều) gồm 5 số nguyên kiểu **int**. Mảng b ở trên được gọi là một mảng 3 chiều.

Sử dụng mảng

Để truy nhập vào một phần tử của mảng thông qua tên của nó. Tên một phần tử của mảng được tạo thành từ tên mảng và theo sau là chỉ số của phần tử đó trong mảng được đặt trong cặp dấu ngoặc vuông

tên_mảng[chỉ_số_của_phần_tử]

Ví dụ với khai báo

int mang_nguyen[3];

Thì mang_nguyen[0] sẽ là phần tử thứ nhất của mảng

mang_nguyen[1] sẽ là phần tử thứ 2 của mảng

mang_nguyen[2] sẽ là phần tử thứ 3 của mảng

Với mảng nhiều chiều như

int a[6][5];

thì a[0] là phần tử đầu tiên của một mảng, phần tử này bản thân nó lại là một mảng một chiều. Phần tử đầu tiên của mảng một chiều a[0] sẽ là a[0][0]. Phần tử tiếp theo của a[0] sẽ là a[0][1]… Và dễ dàng tính được a[2][3] sẽ là phần tử thứ 4 của phần tử thứ 3 của a.

Một cách tổng quát a[i][j] sẽ là phần tử thứ j+1 của a[i], mà phần tử a[i] lại là phần tử thứ i+1 của a.

2.3.3. Các thao tác cơ bản làm việc trên mảng

Nhập dữ liệu cho mảng

Sau khi khai báo mảng ta phải nhập dữ liệu cho mảng. Nhập dữ liệu cho mảng là nhập dữ liệu cho từng phần tử của mảng. Mỗi một phần tử của mảng thực chất là một biến có kiểu dữ liệu là kiểu dữ liệu chung của mảng. Để nhập dữ liệu cho các phần tử của mảng ta có thể dùng hàm scanf() hoặc lệnh gán tương tự như biến thông thường.

Ví dụ

```
float a[10];    // khai bao mot mang so thuc co 10 phan tu
int i;
// Nhap tu ban phim mot so thuc va gan gia tri so thuc do
// cho phan tu thu 2 cua mang, tuc la a[1]
scanf("%f",&a[1]);
// Gán giá trị cho phần tử a[2]
a[2] = a[1] + 5;
```

Nếu ta muốn gán giá trị cho các phần tử của mảng một cách hàng loạt, ta có thể dùng lệnh **for**. Ví dụ

```
int b[10];
int i;
// Nhap gia tri tu ban phim cho tat ca cac phan tu cua mang b
for(i = 0; i < 10; i++)
{
```

```
        printf("\n Nhap gia tri cho b[%d]", i);
        scanf("%d",&b[i]);
    }
```

Trường hợp ta không biết trước mảng sẽ có bao nhiêu phần tử mà chỉ biết số phần tử tối đa có thể có của mảng. Còn số phần tử thực sự của mảng thì chỉ biết khi chạy chương trình. Khi đó cần khai báo mảng với số phần tử bằng số phần tử tối đa, ngoài ra cần một biến để lưu giữ số phần tử thực sự của mảng.

```
    int a[100];     // khai báo mảng, giả sử số phần tử tối đa của a là 100
    int n;                  // biến lưu giữ số phần tử thực sự của mảng
    int i;
    printf("\n Cho biet so phan tu cua mang: ");
    scanf("%d",&n);
    for(i = 0; i < n; i++)
    {
        printf("\n a[%d] = ", i);
        scanf("%d",&a[i]);
    }
```

Mảng có thể được khởi tạo giá trị ngay khi khai báo, ví dụ

```
    int a[4] = {4, 9, 22, 16};
    float b[3] = {40.5, 20.1, 100};
    char c[5] = {'h', 'e', 'l', 'l', 'o'};
```

Câu lệnh thứ nhất có tác dụng tương đương với 4 lệnh gán

```
    a[0] = 4; a[1] = 9; a[2] = 22; a[3] = 16;
```

Xuất dữ liệu chứa trong mảng

Để hiển thị giá trị của các phần tử trong mảng ta dùng hàm printf(). Ví dụ sau minh họa việc nhập giá trị cho các phần tử của mảng, sau đó hiển thị giá trị của các phần tử đó theo các cách khác nhau.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a[5];
    int i, k;
    // Nhap gia tri cho cac phan tu cua mang a tu ban phim
```

```
    for(i = 0; i < 5; i++)
    {
        printf("\n a[%d] = ", i);
        scanf("%d", &a[i]);
    }
    // Hien thi gia tri cua phan tu bat ki, gia su a[3] len man hinh
    printf("\n a[3] = %d", a[3]);
    // Hien thi gia tri cua tat ca cac phan tu, moi phan tu tren mot dong
    for(i = 0; i < 5; i++)
        printf("\n%d", a[i]);
    // Hien thi gia tri cua tat ca cac phan tu tren cung mot dong, cac gia tri cach nhau 2 vi tri
    printf("\n");  // Xuong dong moi
    for(i = 0; i < 5; i++)
        printf("%d  ", a[i]);
    // Hien thi gia tri cua tat ca cac phan tu, trong do k phan tu tren mot dong.
    // Cac phan tu tren cung dong cach nhau 2 vi tri
    printf("\n Cho biet gia tri cua k = ");
    scanf("%d",&k);
    for(i = 0; i < 5; i++)
    {
        printf("%d  ",a[i]);
        if((i+1)%k == 0)  // da hien thi du k phan tu tren mot dong thi phai xuong dong
            printf("\n");
    }
    getch();
}
```

Kết quả

```
 a[0] = 6
 a[1] = 14
 a[2] = 23
 a[3] = 37
 a[4] = 9
 a[3] = 37
6
```

```
14
23
37
9
6  14  23  37  9
 Cho biet gia tri cua k = 2
6  14
23  37
9
```

Tìm phần tử có giá trị lớn nhất, phần tử có giá trị nhỏ nhất

Để tìm phần tử có giá trị lớn nhất trong mảng ban đầu ta giả sử phần tử đó là phần tử đầu tiên của mảng. Sau đó lần lượt so sánh với các phần tử còn lại trong mảng. Nếu gặp phần tử nhỏ hơn thì chuyển sang so sánh với phần tử tiếp theo. Nếu gặp phần tử lớn hơn thì ta sẽ coi phần tử này là phần tử lớn nhất rồi chuyển sang so sánh với phần tử tiếp theo. Sau khi so sánh với phần tử cuối cùng thì ta sẽ tìm được phần tử lớn nhất trong mảng. Đoạn chương trình sau minh họa giải thuật tìm phần tử lớn nhất

```
int a[100];
int i, n;
int max;
printf("\n Cho biet so phan tu cua mang: ");
scanf("%d",&n);
// Nhap du lieu cho mang
for(i = 0; i < n; i++)
{
    printf("\n a[%d] = ",i);
    scanf("%d",&a[i]);
}
// Tim phan tu lon nhat
max = a[0];   // Ban dau gia su phan tu lon nhat la a[0]
// Lan luot so sanh voi cac phan tu con lai trong mang
for(i = 1; i < n; i++)
    if(max < a[i])              // gap phan tu co gia tri lon hon
        max = a[i];    // coi phan tu nay la phan tu lon nhat
printf("\n Phan tu lon nhat trong mang la: %d", max);
```

Ta cũng làm tương tự với trường hợp tìm phần tử nhỏ nhất trong mảng.

Tìm kiếm trên mảng

Yêu cầu của thao tác tìm kiếm trên mảng: Cho một mảng dữ liệu và một dữ liệu bên ngoài mảng. Hãy tìm (các) phần tử của mảng có giá trị bằng giá trị của dữ liệu bên ngoài trên. Nếu có (các) phần tử như vậy thì hãy chỉ ra vị trí của chúng trong mảng. Nếu không tồn tại (các) phần tử như vậy thì đưa ra thông báo không tìm thấy.

Cách làm là lần lượt duyệt qua từng phần tử của mảng, so sánh giá trị của phần tử đang được duyệt với giá trị của dữ liệu bên ngoài. Nếu phần tử đang xét bằng dữ liệu bên ngoài thì ta ghi nhận vị trí của phần tử đó. Sau đó chuyển qua duyệt phần tử kế tiếp trong mảng. Quá trình được lặp đi lặp lại cho đến khi duyệt xong phần tử cuối cùng của mảng. Để có thể trả lời cho cả tình huống không tồn tại phần tử như vậy trong mảng ta nên sử dụng một biến kiểm tra và khi gặp phần tử bằng giá trị dữ liệu bên ngoài thì ta bật biến đó lên, nếu biến đó không được bật lần nào thì ta trả lời là không có phần tử như vậy trong mảng. Phương pháp trên được gọi là phương pháp tìm kiếm tuần tự (*sequential search*).

Dưới đây là cài đặt của thuật toán tìm kiếm tuần tự cho trường hợp mảng dữ liệu là mảng các số nguyên kiểu **int**.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int m[100], chi_so[100];
    int n;      // n la số phần tử trong mảng
    int i, k, kiem_tra;
    clrscr();  // xóa màn hình để tiện theo dõi
    // Nhập giá trị dữ liệu cho mảng m
    // Trước tiên phải biết số phần tử của mảng
    printf(" Cho biet so phan tu co trong mang: ");
    scanf("%d",&n);
    // Rồi lần lượt nhập giá trị cho các phần tử trong mảng
    for(i  = 0;i<n;i++)
    {
        int temp;
    printf("\n Cho biet gia tri cua m[%d] = ",i);
        scanf("%d",&temp);
```

```
        m[i] = temp;
    }
    // Yêu cầu người sử dụng nhập vào giá trị cho dữ liệu k
    printf("\n Cho biết giá trị của dữ liêu k: ");
    scanf("%d",&k);
    // Bắt đầu quá trình tìm kiếm
    kiem_tra = 0;
    // Duyệt qua tất cả các phần tử
    for(i = 0;i<n;i++)
        if(m[i] = = k)//So sánh phần tử đang xét với dữ liệu k
        {
            // Ghi nhận chỉ số của phần tử đang xét
            chi_so[kiem_tra] = i;
    kiem_tra ++; //Tăng biến kiem_tra thêm 1 đơn vị
        }
    // Kết luận
    if(kiem_tra > 0)
    {
        printf("\n Trong mang co %d phan tu co gia tri bang %d",kiem_tra,k);
        printf("\n Chi so cua cac phan tu la: ");
        for(i = 0;i < kiem_tra;i++)
            printf("%3d",chi_so[i]);
    }
    else
        printf("\n Trong mang khong co phan tu nao co gia tri bang %d",k);
    getch();  // Chờ người sử dụng ấn phím bất kì để kết thúc.
}
```

Sắp xếp mảng

Yêu cầu của bài toán: cho một mảng dữ liệu *m*[*n*] với *n* là số phần tử trong mảng. Hãy sắp xếp các phần tử trong mảng theo một trật tự nào đó, giả sử là theo chiều tăng dần (với chiều giảm dần ta hoàn toàn có thể suy luận từ cách làm với chiều tăng dần).

Sắp xếp kiểu lựa chọn (*Selection sort*): ý tưởng của phương pháp là như sau ta cần thực hiện *n* – 1 lượt sắp xếp, trong đó:

- Ở lượt sắp xếp đầu tiên ta so sánh phần tử đầu tiên của mảng *m*[0] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử *m*[1], *m*[2]…*m*[*n*-1]). Nếu có giá trị *m*[*i*] nào đó (*i* = 1, 2,…*n*–1) nhỏ hơn *m*[0] thì ta hoán đổi giá trị giữa *m*[0] và *m*[*i*] cho nhau. Rõ ràng sau lượt sắp xếp thứ nhất *m*[0] sẽ là phần tử có giá trị nhỏ nhất trong mảng.
- Ở lượt sắp xếp thứ 2 ta so sánh phần tử thứ 2 của mảng *m*[1] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử *m*[2], *m*[3]…*m*[*n*-1]). Nếu có giá trị *m*[*i*] nào đó (*i* = 2, 3,…*n*–1) nhỏ hơn *m*[1] thì ta hoán đổi giá trị giữa *m*[1] và *m*[*i*] cho nhau. Sau lượt sắp xếp thứ 2 thì *m*[1] sẽ là phần tử có giá trị nhỏ thứ 2 trong mảng.

  …
- Ở lượt sắp xếp thứ *k* ta so sánh phần tử thứ *k* của mảng là *m*[*k*-1] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử *m*[*k*], *m*[*k*+1]…*m*[*n*-1]). Nếu có giá trị *m*[*i*] nào đó (*i* = *k*, *k*+1,…*n*–1) nhỏ hơn *m*[*k*] thì ta hoán đổi giá trị giữa *m*[*k*] và *m*[*i*] cho nhau. Sau lượt sắp xếp thứ *k* thì *m*[*k*-1] sẽ là phần tử có giá trị nhỏ thứ *k* trong mảng.

  …
- Ở lượt sắp xếp thứ n-1 ta so sánh phần tử thứ *n*-1 của mảng *m*[*n*-2] với tất cả các phần tử đứng sau nó trong mảng (tức là phần tử *m*[*n*-1]). Nếu *m*[*n*-1] nhỏ hơn *m*[*n*-2] thì ta hoán đổi giá trị giữa *m*[*n*-2] và *m*[*n*-1] cho nhau. Sau lượt sắp xếp thứ *n*-1 thì *m*[*n*-2] sẽ là phần tử có giá trị nhỏ thứ *n*-2 trong mảng. Và dĩ nhiên phần tử còn lại là *m*[*n*-1] sẽ là phần tử nhỏ thứ *n* trong mảng (tức là phần tử lớn nhất trong mảng). Kết thúc *n*-1 lượt sắp xếp ta có các phần tử trong mảng đã được sắp xếp theo thứ tự tăng dần.

Cài đặt giải thuật

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int m[100];
    int n;      // n la số phần tử trong mảng
    int i, j, k;
    clrscr();  // xóa màn hình để tiện theo dõi
    // Nhập giá trị dữ liệu cho mảng m
    // Trước tiên phải biết số phần tử của mảng
    printf(“ Cho biet so phan tu co trong mang: “);
    scanf(“%d”,&n);
```

```
    // Rồi lần lượt nhập giá trị cho các phần tử trong mảng
        for(i  = 0;i<n;i++)
    {
        int temp;
        printf("\n Cho biet gia tri cua m[%d] = ",i);
        scanf("%d",&temp);
        m[i] = temp;
    }
    // Hiển thị ra màn hình mảng vừa nhập vào
    printf("\n Mang truoc khi sap xep:    ");
    for(i=0;i<n;i++)
        printf("%3d",m[i]);
    // Bắt đầu sắp xếp
    for(i = 0; i<n-1;i++)
    {
        // Ở lượt sắp xếp thứ i+1
        for(j = i+1;j<n;j++)
        {
            // So sánh m[i] với các phần tử còn lại
                    // và đổi chỗ khi tìm thấy phần tử < m[i].
            if(m[j]<m[i])
            {
                int temp;
                temp = m[j]; m[j] = m[i]; m[i] = temp;
            }
        }
        // Hiển thị mảng sau lượt sắp xếp thứ i+1
            printf("\n Mang o luot sap xep thu %d",i+1);
        for(k = 0;k < n ;k++)
                    printf("%3d",m[k]);
    }
    getch();  // Chờ người sử dụng ấn phím bất kì để kết thúc.
}
```

Kết quả thực hiện:

```
Cho biet so phan tu co trong mang: 5
Cho biet gia tri cua m[0]: 34
Cho biet gia tri cua m[1]: 20
Cho biet gia tri cua m[2]: 17
Cho biet gia tri cua m[3]: 65
Cho biet gia tri cua m[4]: 21
Mang truoc khi sap xep:    34 20 17 65 21
Mang o luot sap xep thu 1: 17 34 20 65 21
Mang o luot sap xep thu 2: 17 20 34 65 21
Mang o luot sap xep thu 3: 17 20 21 65 34
Mang o luot sap xep thu 4: 17 20 21 34 65
```

BÀI 3

VÀO RA

3.1 Các lệnh vào ra dữ liệu với các biến (printf, scanf)

Để vào ra dữ liệu, C cung cấp 2 hàm vào ra cơ bản là printf() và scanf(). Muốn sử dụng 2 hàm printf() và scanf() ta cần khai báo tệp tiêu đề stdio.h.

Hàm printf()

Cú pháp sử dụng hàm printf ( )

printf(xâu_định_dạng, [danh_sách_tham_số]);

Hàm printf() được dùng để hiển thị ra màn hình các loại dữ liệu cơ bản như số, kí tự và xâu kí tự cùng một số hiệu ứng hiển thị đặc biệt.

xâu_định_dạng là xâu điều khiển cách thức hiển thị dữ liệu trên thiết bị ra chuẩn(màn hình máy tính). Trong xâu_định_dạng có chứa:

- Các kí tự thông thường, chúng sẽ được hiển thị ra màn hình bình thường.
- Các nhóm kí tự định dạng dùng để xác định quy cách hiển thị các tham số trong phần danh_sách_tham_số.
- Các kí tự điều khiển dùng để tạo các hiệu ứng hiển thị đặc biệt như xuống dòng (‘\n’) hay sang trang (‘\f’)…

Phần danh_sách_tham_số là các giá trị biến, hằng, biểu thức mà ta muốn hiển thị ra màn hình. Nhóm kí tự định dạng thứ *k* trong xâu_định_dạng dùng để xác định quy cách hiển thị tham số thứ *k* trong danh_sách_tham_số. Do đó danh_sách_tham_số phải phù hợp về số lượng, thứ tự và kiểu với các nhóm kí tự định dạng trong xâu_định_dạng. Số lượng tham số trong danh_sách_tham_số bằng số lượng nhóm các kí tự định dạng trong xâu_định_dạng.

Ví dụ

```
#include <conio.h>
#include <stdio.h>
void main()
{
    int a = 5;
    float x = 1.234;
    printf(“Hien thi mot so nguyen %d và mot so thuc %f”,a,x);
    getch();
}
```

Kết quả:

```
Hien thi mot so nguyen 5 va mot so thuc 1.234000
```

Trong ví dụ trên "Hien thi mot so nguyen %d và mot so thuc %f" là xâu định dạng, còn a và x là các tham số của hàm printf(). Trong xâu định dạng trên có 2 nhóm kí tự định dạng là %d và %f, với %d dùng để báo cho máy biết rằng cần phải hiển thị tham số tương ứng (biến a) theo định dạng số nguyên và %f dùng để báo cho máy cần hiển thị tham số tương ứng (biến x) theo định dạng số thực.

Lưu ý là mỗi nhóm kí tự định dạng chỉ dùng cho một kiểu dữ liệu, còn một kiểu dữ liệu có thể hiển thị theo nhiều cách khác nhau nên có nhiều nhóm kí tự định dạng khác nhau. Nếu giữa nhóm kí tự định dạng và tham số tương ứng không phù hợp với nhau thì sẽ hiển thị ra kết quả không như ý. Phần sau đây giới thiệu một số nhóm kí tự định dạng hay dùng trong C và ý nghĩa của chúng.

Nhóm kí tự định dạng	**Áp dụng cho kiểu dữ liệu**	**Ghi chú**
%d	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên có dấu hệ đếm thập phân
%i	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên có dấu hệ đếm thập phân
%o	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên không dấu trong hệ đếm cơ số 8.
%u	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên không dấu.
%x	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên hệ đếm 16 (không có 0x đứng trước), sử dụng các chữ cái a b c d e f
%X	**int, long, char**	Hiển thị tham số tương ứng dưới dạng số nguyên hệ đếm 16 (không có 0x đứng trước), sử dụng các chữ cái A B C D E F
%e	**float, double**	Hiển thị tham số tương ứng dưới dạng số thực dấu phẩy động
%f	**float, double**	Hiển thị tham số tương ứng dưới dạng số thực dấu phẩy tĩnh

%g	**float, double**	Hiển thị tham số tương ứng số thực dưới dạng ngắn gọn hơn trong 2 dạng dấu phẩy tĩnh và dấu phẩy động
%c	**int, long, char**	Hiển thị tham số tương ứng dưới dạng kí tự
%s	**char *** (xâu kí tự)	Hiển thị tham số tương ứng dưới dạng xâu kí tự

Để trình bày dữ liệu được đẹp hơn, C cho phép đưa thêm một số thuộc tính định dạng dữ liệu khác vào trong xâu định dạng như độ rộng tối thiểu, căn lề trái, căn lề phải.

Độ rộng tối thiểu

Thông thường khi hiển thị dữ liệu, C tự động xác định số chỗ cần thiết sao cho hiển thị vừa đủ nội dung dữ liệu.

Nếu ta muốn C hiển thị dữ liệu của ta trên một số lượng vị trí xác định bất kể nội dung dữ liệu đó có điền đầy số chỗ được cung cấp hay không, ta có thể chèn một số nguyên vào trong nhóm kí tự định dạng, ngay sau dấu %.

Ví dụ khi hiển thị số nguyên

```
a = 1234;
printf("\n%5d",a); // dành 5 chỗ để hiển thị số nguyên a
printf("\n%5d",34);// dành 5 chỗ để hiển thị số nguyên 34
```

Kết quả

```
□1234
□□□34
```

Ở đây □ kí hiệu thay cho dấu trắng (*space*).

Như vậy với nhóm kí tự định dạng **%md**, m dùng để báo số chỗ cần dành để hiển thị dữ liệu, còn d báo rằng hãy hiển thị dữ liệu đó dưới dạng một số nguyên. Tương tự với các nhóm kí tự định dạng **%mc** khi hiển thị kí tự, và **%ms** khi hiển thị xâu kí tự.

Ví dụ

```
 printf("\n%3d %15s %3c", 1, "nguyen van a", 'g');
 printf("\n%3d %15s %3c", 2, "tran van b", 'k');
```

Kết quả

```
□□1□□□□nguyen van a□□g
□□2□□□□□□tran van b□□k
```

Nếu nội dung dữ liệu không điền đầy số chỗ được cấp thì những chỗ không dùng đến sẽ được điền bởi dấu trắng.

Khi số chỗ cần thiết để hiển thị nội dung dữ liệu lớn hơn m thì C tự động cung cấp thêm chỗ mới để hiển thị chứ không cắt bớt nội dung của dữ liệu để cho vừa m vị trí.

Với dữ liệu là số thực ta sử dụng mẫu nhóm kí tự định dạng **%m.nf** để báo rằng cần dành m vị trí để hiển thị số thực, và trong m vị trí đó dành n vị trí để hiển thị phần thập phân.

Ví dụ:

```
printf("\n%f",12.345);
printf("\n%.2f",12.345);
printf("\n%8.2f",12.345);
```

Kết quả

```
12.345000
12.35
□□□12.35
```

Căn lề trái

Khi hiển thị dữ liệu, mặc định C căn lề phải. Nếu muốn căn lề trái khi hiển thị dữ liệu ta chỉ cần thêm dấu trừ - vào ngay sau dấu %.

Ví dụ

```
printf("\n%-3d %-15s %-4.2f %-3c", 1, "nguyen van a", 8.5, 'g');
printf("\n%-3d %-15s %-4.2f %-3c", 2, "tran van b", 6.75, 'k');
```

Kết quả

```
1   nguyen van a    8.50 g
2   tran van b      6.75 k
```

Dễ thấy các thuộc tính định dạng độ rộng tối thiểu, căn lề… giúp cho việc hiển thị dữ liệu được thẳng, đều và đẹp hơn.

Thuộc tính	**Quy cách kí tự định dạng (m, n là các số nguyên)**	**Ví dụ**
Độ rộng tối thiểu	**%md, %ms, %mc**	printf(“%3d”,10); printf(“%4s”,”CNTT”); printf(“%2c”,’A’);
Độ rộng dành cho phần thập phân	**%m.nf**	printf(“%5.1f”,1234.5);

Căn lề trái	**%-md, %-ms, %-mc**	printf(“%-3d”,10); printf(“%-4s”,”CNTT”); printf(“%-2c”,’A’);

Hàm scanf()

Cú pháp:

scanf(xâu_định_dạng, [danh_sách_địa_chỉ]);

Hàm scanf() dùng để nhập dữ liệu từ bàn phím. Cụ thể nó sẽ đọc các kí tự được nhập từ bàn phím, sau đó căn cứ theo xâu_định_dạng sẽ chuyển những thông tin đã nhập được sang kiểu dữ liệu phù hợp. Cuối cùng sẽ gán những giá trị vừa nhập được vào các biến tương ứng trong danh_sách_địa_chỉ.

xâu_định_dạng trong hàm scanf() xác định khuôn dạng của các dữ liệu được nhập vào. Trong xâu_định_dạng có chứa các nhóm kí tự định dạng xác định khuôn dạng dữ liệu nhập vào.

Địa chỉ của một biến được viết bằng cách đặt dấu & trước tên biến. Ví dụ giả sử ta có các biến có tên là a, x, ten_bien thì địa chỉ của chúng lần lượt sẽ là &a, &x, &ten_bien.

danh_sách_địa_chỉ phải phù hợp với các nhóm kí tự định dạng trong xâu_định_dạng về số lượng, kiểu dữ liệu và thứ tự. Số nhóm kí tự định dạng bằng số địa chỉ của các biến trong danh_sách_địa_chỉ. Dưới đây là một số nhóm kí tự định dạng hay dùng và ý nghĩa

Nhóm kí tự định dạng	**Ghi chú**
%d	Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int**
%o	Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int** hệ cơ số 8
%x	Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int** hệ cơ số 16
%c	Định khuôn dạng dữ liệu nhập vào dưới dạng kí tự kiểu **char**
%s	Định khuôn dạng dữ liệu nhập vào dưới dạng xâu kí tự
%f	Định khuôn dạng dữ liệu nhập vào dưới dạng số thực kiểu **float**
%ld	Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **long**
%lf	Định khuôn dạng dữ liệu nhập vào dưới dạng số thực kiểu **double**

Ví dụ:

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

```
void main()
{
    // khai bao bien
    int a;
    float x;
    char ch;
    char* str;
    // Nhap du lieu
    printf(“Nhap vao mot so nguyen”);
    scanf(“%d”,&a);
    printf(“\n  Nhap vao mot so thuc”);
    scanf(“%f”,&x);
    printf(“\n Nhap vao mot ki tu”);
    fflush(stdin); scanf(“%c”,&ch);
    printf(“\n Nhap vao mot xau ki tu”);
    fflush(stdin); scanf(“%s”,str);
    // Hien thi du lieu vua nhap vao
    printf(“\n Nhung du lieu vua nhap vao”);
    printf(“\n So nguyen: %d”,a);
    printf(“\n So thuc : %.2f”,x);
    printf(“\n Ki tu: %c:,ch);
    printf(“\n Xau ki tu: %s”,str);
}
```

Kết quả:

```
Nhap vao mot so nguyen: 2007
Nhap vao mot so thuc: 18.1625
Nhap vao mot ki tu: b
Nhap vao mot xau ki tu: ngon ngu lap trinh C
Nhung du lieu vua nhap vao
So nguyen: 2007
So thuc: 18.16
Ki tu: b
Xau ki tu: ngon
```

Một số quy tắc cần lưu ý khi sử dụng hàm scanf()

- Quy tắc 1: Khi đọc số, hàm scanf() quan niệm rằng mọi kí tự số, dấu chấm (‘.’) đều là kí tự hợp lệ. Khi gặp các dấu phân cách như tab, xuống dòng hay dấu cách (space bar) thì scanf() sẽ hiểu là kết thúc nhập dữ liệu cho một số.
- Quy tắc 2: Khi đọc kí tự, hàm scanf() cho rằng mọi kí tự có trong bộ đệm của thiết bị vào chuẩn đều là hợp lệ, kể cả các kí tự tab, xuống dòng hay dấu cách.
- Quy tắc 3: Khi đọc xâu kí tự, hàm scanf() nếu gặp các kí tự dấu trắng, dấu tab hay dấu xuống dòng thì nó sẽ hiểu là kết thúc nhập dữ liệu cho một xâu kí tự. Vì vậy trước khi nhập dữ liệu kí tự hay xâu kí tự ta nên dùng lệnh fflush(stdin).

3.2 Các lệnh nhập xuất khác

Hàm gets(), có cú pháp

gets(xâu_kí_tự);

Hàm gets() dùng để nhập vào từ bàn phím một xâu kí tự bao gồm cả dấu cách, điều mà hàm scanf() không làm được.

Hàm puts(), có cú pháp

puts(xâu_kí_tự);

Hàm puts() sẽ hiển thị ra màn hình nội dung xâu_kí_tự và sau đó đưa con trỏ xuống dòng mới. Vì vậy nó tương đương với lệnh printf(“%s\n”,xâu_kí_tự).

Hàm getch(), có cú pháp

getch();

Hàm getch() là hàm không có tham số. Nó đọc một kí tự bất kì nhập vào từ bàn phím nhưng không hiển thị kí tự đó lên màn hình. Lệnh getch() thường dùng để chờ người sử dụng ấn một phím bất kì rồi sẽ kết thúc chương trình.

Để sử dụng các hàm gets(), puts(), getch() ta cần khai báo tệp tiêu đề conio.h.

Ví dụ:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    // khai bao bien
    char* str;
    // Nhap du lieu
    puts(“Nhap vao mot xau ki tu:”);
    fflush(stdin); gets(str);
```

```
    // Hien thi du lieu vua nhap vao
    puts("Xau vua nhap vao: ");
    puts(str);
    puts("An phim bat ki de ket thuc ...");
    getch();
}
```

Kết quả:

```
Nhap vao mot xau ki tu:
ngon ngu lap trinh C
Xau vua nhap vao:
ngon ngu lap trinh C
An phim bat ki de ket thuc ...
```

BÀI 4

CẤU TRÚC RẼ NHÁNH

4.1 Cấu trúc if, if … else

Cú pháp cấu trúc **if**

if (biểu_thức_điều_kiện)

lệnh;

Cú pháp cấu trúc **if … else**

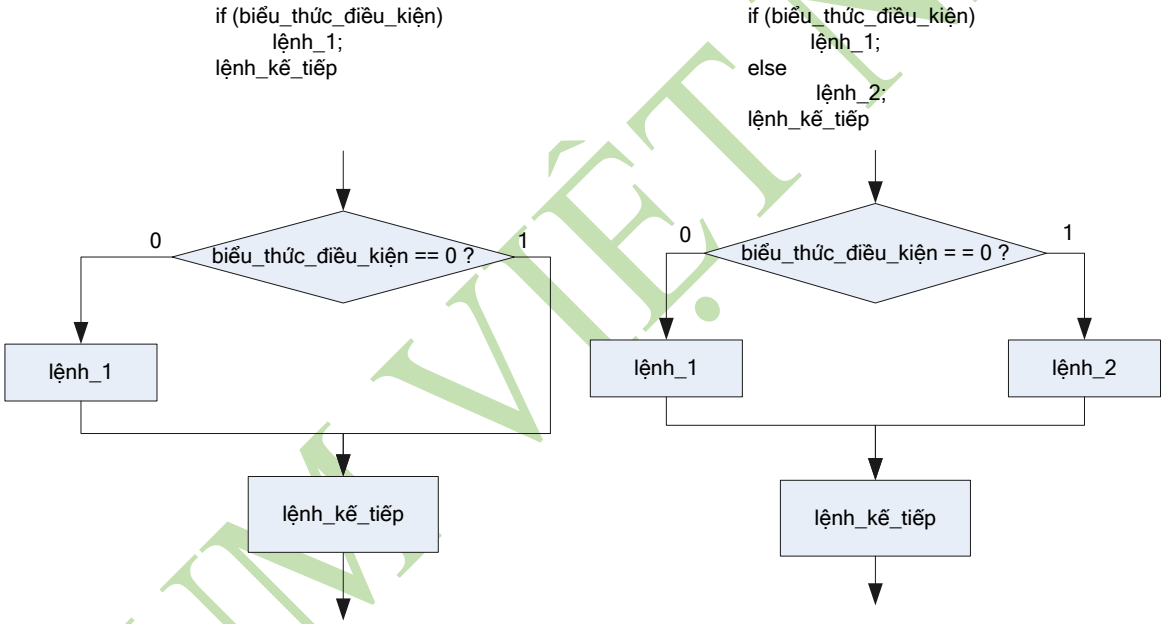
if (biểu_thức_điều_kiện)

lệnh_1;

else

lệnh_2;

Lưu ý là lệnh, lệnh_1 và lệnh_2 có thể là lệnh khối.



Ví dụ: Bài toán tìm số lớn nhất trong 2 số thực *a* và *b*. Cách làm là ta so sánh *a* với *b*, nếu *a* nhỏ hơn *b* thì *b* là số lớn nhất, còn nếu không, tức là *a* ≥ b, thì *a* là số lớn nhất.

```
#include <conio.h>
#include <stdio.h>
void main()
{
// khai bao bien
float a, b;
float max;
printf(“ Nhap gia tri a va b: “);
```

```
scanf("%f %f",&a,&b);
if(a<b)
    max = b;
else
    max = a;
printf("\n So lon nhat trong 2 so %.0f va %.0f la %.0f ",a,b,max);
getch();
}
```

Kết quả:

```
Nhap vao 2 gia tri a va b: 23 247
So lon nhat trong hai so 23 va 247 la 247
```

4.2 Cấu trúc lựa chọn switch

Cú pháp cấu trúc **switch**

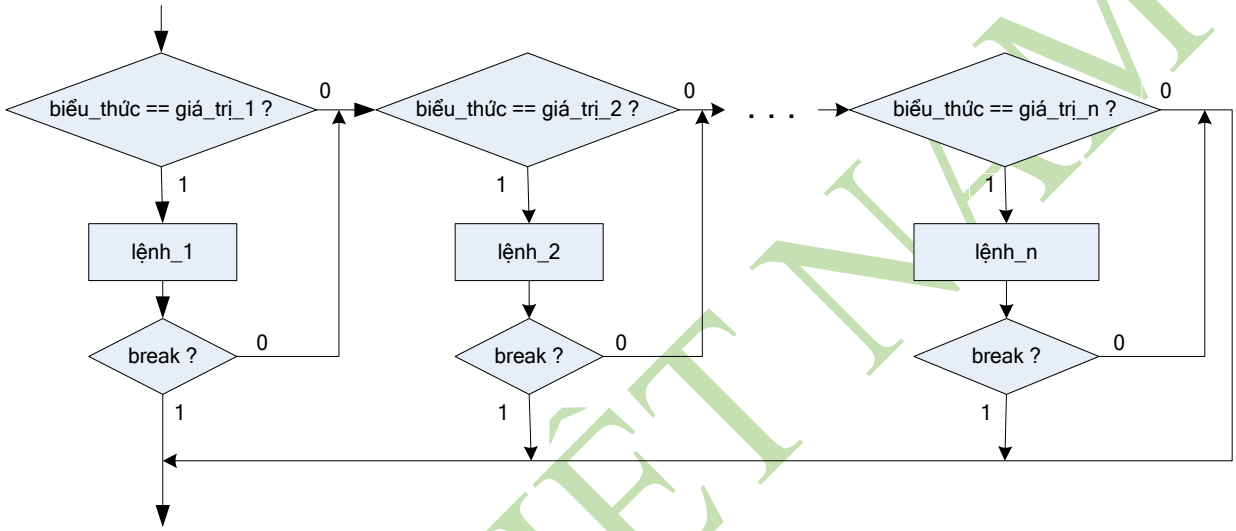
```
switch (biểu_thức)
{
    case giá_trị_1:   lệnh_1; [break;]
    case giá_trị_2:   lệnh_2; [break;]
        …
    case giá_trị_n:   lệnh_n; [break;]
    [default:             lệnh_n+1; [break;]]
}
```

Cơ chế hoạt động: câu lệnh **switch** ban đầu sẽ tính giá trị của biểu_thức, sau đó so sánh với các giá_trị_k với k = 1, 2, …, n đứng sau **case**. Xảy ra 2 trường hợp

- Nếu trong dãy các giá trị giá_trị_1, giá_trị_2, ... tồn tại giá trị bằng biểu_thức. Gọi i là chỉ số của giá trị đầu tiên trong dãy thỏa mãn giá_trị_i bằng biểu_thức, khi đó lệnh_i sẽ được thực hiện. Sau khi thực hiện xong lệnh_i, nếu có lệnh **break** thì chương trình sẽ chuyển sang thực hiện lệnh tiếp sau cấu trúc **switch**. Nếu không có lệnh **break** thì chương trình sẽ chuyển sang thực hiện các lệnh sau lệnh_i nằm trong **switch** (tức là lệnh_i+1, lệnh_i+2…) cho đến khi gặp lệnh **break** đầu tiên hoặc sau khi thực hiện xong lệnh n. Sau đó chương trình sẽ chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.

- Nếu không tồn tại giá_trị_k (với k = 1, 2, …n) nào bằng giá trị của biểu_thức thì sẽ có 2 khả năng:
    - Nếu có nhãn **default**: chương trình sẽ thực hiện lệnh_n+1 rồi chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.
    - Nếu không có nhãn **default**: chương trình chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.

Sơ đồ:



Ví dụ:

/* Ví dụ sau yêu cầu người dùng nhập vào một số nguyên không âm và đưa ra ngày trong tuần tương ứng với số nguyên đó. Và ở đây ta quy ước những số chia hết cho 7 ứng với Chủ nhật, chia 7 dư 1 ứng với thứ Hai, …, chia 7 dư 6 ứng với thứ Bảy.*/

```
#include <conio.h>
#include <stdio.h>
void main()
{
    // khai bao bien
    int a;
    do
    {
        printf("\n Nhap mot gia tri so nguyen khong am: ");
        scanf("%d",&a);
        if(a<0)
```

```
            printf("\n so vua nhap la so am");
        }while(a<0);
        printf("\n Thu trong tuan tuong ung voi so do la: ");
        switch(a % 7)
        {
          case 0: printf(" Chu nhat"); break;
          case 1: printf(" Thu Hai"); break;
          case 2: printf(" Thu Ba"); break;
          case 3: printf(" Thu Tu"); break;
          case 4: printf(" Thu Nam"); break;
          case 5: printf(" Thu Sau"); break;
          case 6: printf(" Thu Bay"); break;
        }
        getch();
    }
```

Kết quả:

```
Nhap vao mot gia tri so nguyen: 2356
Thu tuong ung voi so do la Thu Nam
```

Người ta thường dựa trên tính chất tự động chuyển xuống các câu lệnh sau khi không có lệnh break để viết chung mã lệnh cho các trường hợp khác nhau nhưng cùng được xử lý giống nhau. Ví dụ khi viết chương trình hỗ trợ menu dòng lệnh không phân biệt chữ hoa chữ thường hay bài toán in ra số ngày trong các tháng trong năm dưới đây. Trong một năm các tháng có 30 ngày là 4, 6, 9, 11 còn các tháng có 31 ngày là 1, 3, 5, 7, 8, 10, 12. Riêng tháng hai có thể có 28 hoặc 29 ngày.

```
#include <stdio.h>
#include<conio.h>
int main ()
{
      int thang;
 clrscr();
 printf("\n Nhap vao thangs trong nam ");
 scanf("%d",&thang);
 switch(thang)
```

```
{
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:
    printf("\n Thang %d co 31 ngay ",thang);
    break;
  case 4:
      case 6:
  case 9:
  case 11:
    printf("\n Thang %d co 30 ngay ",thang);
    break;
     case 2:
    printf ("\ Thang 2 co 28 hoac 29 ngay");
    break;
  default :
    printf("\n Khong co thang %d", thang);
    break;
}
getch();
return 0;
}
```

Lưu ý: giá trị của biểu thức kiểm tra phải là số nguyên tức là phải có kiểu dữ liệu là **char**, **int**, **long**. Một cách tương ứng các giá trị sau **case** cũng phải nguyên. Đây là một trong những điểm phân biệt giữa cấu trúc **switch** và **if…else**

4.3 Lệnh goto và nhãn

Câu lệnh goto trong C cung cấp một bước nhảy vô điều kiện từ 'goto' đến một câu lệnh có nhãn trong cùng một hàm.

Chú ý: Việc sử dụng câu lệnh goto không được khuyến khích sử dụng trong bất kỳ ngôn ngữ lập trình nào vì nó rất khó để theo dõi luồng điều khiển của chương trình, làm cho chương

trình khó hiểu và khó bảo trì. Bất kỳ chương trình nào sử dụng goto đều có thể được viết lại theo cách bình thường.

Cú pháp

Cú pháp cho câu lệnh goto trong C như sau:

```
goto label;
……
……
label: statement;
```

Nhãn

Nhãn (label) có thể là bất kỳ văn bản thuần tuý trừ từ khóa C và nó có thể được đặt ở bất kỳ vị trí nào trong chương trình C, bên trên hoặc bên dưới câu lệnh goto.

Ví dụ sử dụng lệnh goto trong C

```
#include <stdio.h>

int main () {
   int a = 10;

   TEST:do {
      if( a == 15) {
         // quay ve do khi a = 15 (bo qua lenh print)
         a = a + 1;
         goto TEST;
      }
      printf("Gia tri cua a: %d\n", a);
      a++;
   } while( a < 20 );

  return 0;
}
```

Kết quả:

```
Gia tri cua a: 10
Gia tri cua a: 11
Gia tri cua a: 12
Gia tri cua a: 13
```

```
Gia tri cua a: 14
Gia tri cua a: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```

BÀI 5

CẤU TRÚC VÒNG LẶP

5.1 Cấu trúc lặp

Trong thuật toán có 3 cấu trúc điều khiển cơ bản là tuần tự, rẽ nhánh và lặp. Để thực hiện cấu trúc lặp, C cung cấp các cấu trúc lặp sau: vòng lặp **for**, vòng lặp **while**, vòng lặp **do...while**.

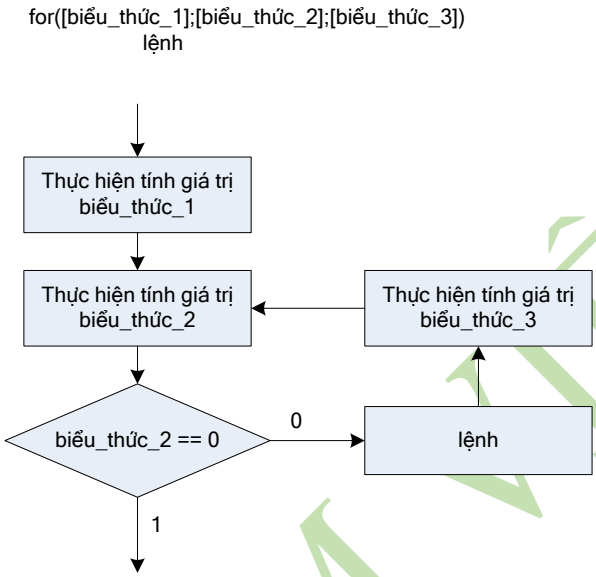
5.1.1 Vòng lặp for

Dạng thường gặp của vòng lặp **for** là

for([biểu_thức_1];[biểu_thức_2];[biểu_thức_3])

lệnh;

Sơ đồ



Câu lệnh **for** thường dùng để thực hiện lặp đi lặp lại một công việc nào đó với số lần lặp xác định.

Ví dụ 1: Hãy đưa ra màn hình các số nguyên dương nhỏ hơn 10.

Cách làm: có 9 số nguyên dương nhỏ hơn 10 là 1, 2, 3, 4, 5, 6, 7, 8 và 9. Để in 9 số nguyên dương này ta cần sử dụng một biến nguyên đặt tên là *i*.

Bước 1: gán cho i giá trị bằng 1.

Bước 2: đưa ra màn hình giá trị của i.

Bước 3: tăng giá trị của i thêm 1 đơn vị.

Bước 4: kiểm tra nếu giá trị của i ≤ 9 thì quay về bước 2, nếu giá trị của i > 9 thì chuyển sang bước 5.

Bước 5: kết thúc.

Chương trình in ra màn hình các số nguyên dương nhỏ hơn 10 sử dụng vòng lặp for:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    for(i = 1;i <= 9;i++)
        printf("%5d",i);// ta dành 5 vị trí để in mỗi số
    getch();
}
```

Kết quả thực hiện

```
  0   1   2   3   4   5  6   7   8   9
```

Ta có thể so sánh cách làm này với việc phải viết một lệnh printf() trong đó phải liệt kê toàn bộ các số nguyên dương lẻ nhỏ hơn 10.

Ví dụ 2: Tính và hiển thị ra màn hình tổng của 100 số tự nhiên lẻ đầu tiên.

Cách làm: 100 số lẻ đầu tiên là 1, 3, 5, …, 199. Ta cần sử dụng một biến nguyên S để chứa giá trị của tổng và một biến nguyên i.

Bước 1: ban đầu gán cho S giá trị bằng 0, gán cho i giá trị bằng 1;

Bước 2: S = S + i;

Bước 3: tăng giá trị của i thêm 2 đơn vị

Bước 4: kiểm tra nếu giá trị của i ≤ 199 thì quay về bước 2, ngược lại nếu i > 199 thì chuyển sang bước 5.

Bước 5: kết thúc.

Chương trình:

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int i;
   int S;
   S = 0;
   for(i = 1; i <= 199; i = i+2)
       S = S+i;
   printf("Tong cua 100 so nguyen duong le dau tien la %d",S);
```

```
    getch();
}
```

Kết quả thực hiện:

```
Tong cua 100 so nguyen duong le dau tien la 10000
```

5.1.2. Vòng lặp while và vòng lặp do {…} while

Cú pháp vòng lặp **while**

while (biểu_thức)

lệnh;

Cú pháp vòng lặp **do {…} while**

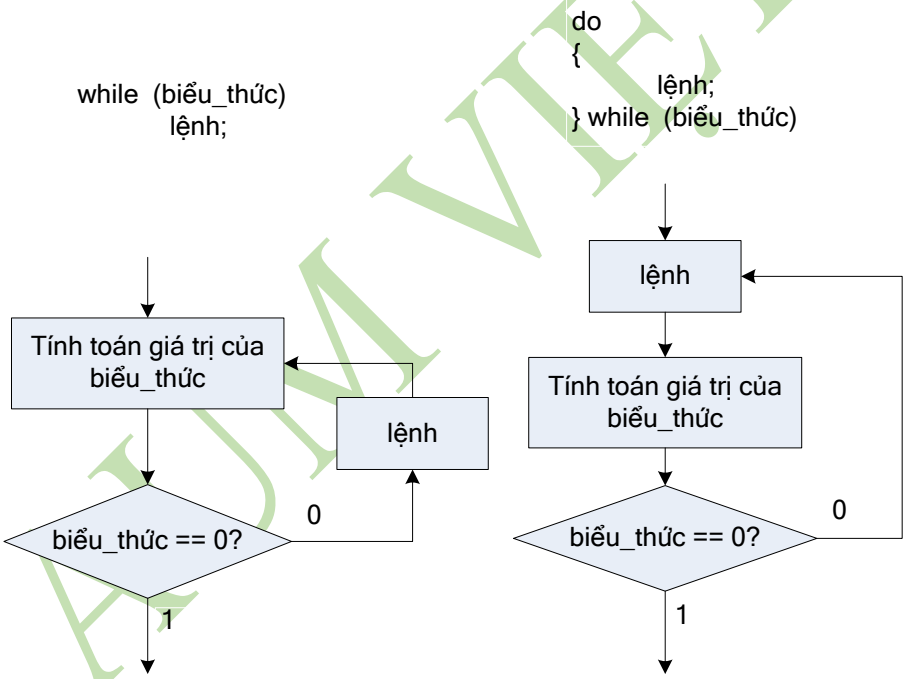
do

{

lệnh;

}while (biểu_thức);

Sơ đồ vòng lặp **while** và **do{…}while**



Sự khác nhau giữa **while** và **do{...}while**

- Lệnh **while** kiểm tra điều kiện vòng lặp (tức là giá trị của biểu thức) trước rồi mới thực hiện lệnh, và do vậy sẽ có khả năng lệnh không được thực hiện lần nào.
- Lệnh **do{...}while** thực hiện lệnh trước rồi mới kiểm tra điều kiện của vòng lặp, và do vậy lệnh sẽ luôn được thực hiện ít nhất một lần.

Cấu trúc **while** và **do{...}while** được dùng để thực hiện lặp đi lặp lại một công việc nào đó với số lần lặp không xác định.

Ví dụ: Sau đây là 2 đoạn chương trình có chức năng tương đương nhau nhưng một đoạn sử dụng cấu trúc **while**, một đoạn sử dụng cấu trúc **do{...}while**.

Chức năng của chương trình là yêu cầu người dùng nhập vào giá trị 1 số nguyên, đưa ra thông báo số đó có phải là số hoàn thiện hay không, sau đó hỏi người dùng có muốn nhập lại số nguyên khác và kiểm tra có phải số hoàn thiện hay không.

Đoạn chương trình sử dụng cấu trúc **do{...}while**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    long int n;
    long int tong;
    long int i;
    char ch;

clrscr();
    do
    {
        tong = 0;
        printf("\n Nhap vao mot so nguyen: ");
        scanf("%ld",&n);
        printf("\n Cac uoc so cua %ld la: ",n);
        for(i = 1;i<n;i++)
            if(n % i == 0)
{
printf("%5d",i);
tong = tong + i;
            }
        printf("\n Tong cac uoc so cua %ld bang %ld",n,tong);
        if(tong == n)
            printf("\n %5ld LA so hoan thien");
        else
```

```
            printf("\n %5ld KHONG LA so hoan thien");
        printf("\n Ban co muon thuc hien lai(c/k)? ");
        fflush(stdin);
        scanf("%c",&ch);
    }while((ch!='k')&&(ch!='K'));
    printf("\n An phim bat ki de ket thuc ...");
    getch();
}
```

Đoạn chương trình viết bằng cấu trúc **while**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    long int n;
    long int tong;
    long int i;
    char ch;

clrscr();
    ch = 'c';
    while((ch != 'k')&&(ch != 'K'))
    {
        tong = 0;
        printf("\n Nhap vao mot so nguyen: ");
        scanf("%ld",&n);
        printf("\n Cac uoc so cua %ld la: ",n);
        for(i = 1;i<n;i++)
            if(n % i == 0)
{
printf("%5d",i);
tong = tong + i;
            }
        printf("\n Tong cac uoc so cua %ld bang %ld",n,tong);
        if(tong == n)
```

```
            printf("\n %5ld LA so hoan thien");
        else
            printf("\n %5ld KHONG LA so hoan thien");
        printf("\n Ban co muon thuc hien lai(c/k)? ");
        fflush(stdin);
        scanf("%c",&ch);
    }
    printf("\n An phim bat ki de ket thuc ...");
    getch();
}
```

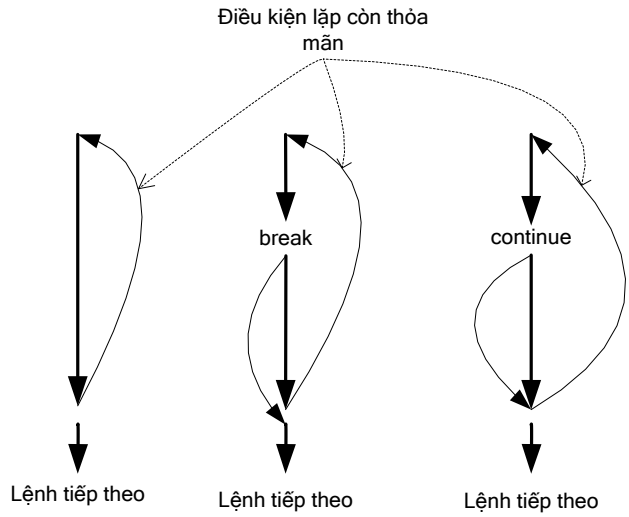
Kết quả thực hiện chương trình

```
Nhap vao mot so nguyen: 10
Cac uoc so cua %10 la: 1   2    5
Tong cac uoc so cua 10 bang 8
10 KHONG LA so hoan thien
Ban co muon thuc hien lai(c/k)? c
Nhap vao mot so nguyen: 12
Cac uoc so cua 12 la: 1   2    3    4    6
Tong cac uoc so cua 12 bang 16
12 KHONG LA so hoan thien
Ban co muon thuc hien lai(c/k)? c
Nhap vao mot so nguyen: 28
Cac uoc so cua %ld la: 1   2    4    7    14
Tong cac uoc so cua 28 bang 28
28 LA so hoan thien
Ban co muon thuc hien lai(c/k)? k
An phim bat ki de ket thuc ...
```

5.2. Các lệnh thay đổi cấu trúc lập trình

Các vòng lặp **while**, **do{...}while**, hay **for** sẽ kết thúc quá trình lặp khi biểu thức điều kiện của vòng lặp không còn được thỏa mãn. Tuy nhiên trong lập trình đôi khi ta cũng cần thoát khỏi vòng lặp ngay cả khi biểu thức điều kiện của vòng lặp vẫn còn được thỏa mãn. Để hỗ trợ người lập trình làm việc đó, ngôn ngữ C cung cấp 2 câu lệnh là **continue** và **break**

Vòng lặp với các lệnh break và continue



5.2.1. continue

Khi gặp lệnh **continue** trong thân vòng lặp, chương trình sẽ chuyển sang thực hiện một vòng lặp mới và bỏ qua việc thực hiện các câu lệnh nằm sau lệnh **continue** trong thân vòng lặp.

Ví dụ sau đây sẽ in ra màn hình các số tự nhiên lẻ và nhỏ hơn 100

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    for(i = 1;i<100;i++)
    {
        if(i%2 == 0) continue;
            printf("%5d",i);
        if((i+1)%20 ==0) printf("\n");
    }
    getch();
}
```

Kết quả thực hiện

```
  1   3   5   7   9  11  13  15  17  19
 21  23  25  27  29  31  33  35  37  39
 41  43  45  47  49  51  53  55  57  59
```

```
61  63  65  67  69  71  73  75  77  79
81  83  85  87  89  91  93  95  97  99
```

5.2.2. break

Khi gặp lệnh **break**, chương trình sẽ thoát khỏi vòng lặp (đối với trường hợp lệnh **break** nằm trong các cấu trúc lặp **while**, **do{...}while**, **for**) hoặc thoát khỏi cấu trúc **switch** (với trường hợp lệnh **break** nằm trong cấu trúc **switch**).

Ví dụ sau sẽ thực hiện việc yêu cầu người dùng nhập vào một kí tự và màn hình thông báo về kí tự vừa nhập. Việc này được lặp đi lặp lại cho đến khi kí tự nhập vào là kí tự 't' hoặc 'T' (viết tắt của từ thoát).

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char ch;
    clrscr();
    do
    {
        printf("\n Nhap vao mot ki tu: ");
        fflush(stdin);
        scanf("%c",&ch);
        printf("\n Ki tu vua nhap vao la %c",ch);
        if((ch =='T')||(ch =='t')) break;
    }while(1);
    printf("\n An phim bat ki de ket thuc chuong trinh...");
    getch();
}
```

Kết quả thực hiện chương trình

```
Nhap vao mot ki tu: a
Ki tu vua nhap vao la a
Nhap vao mot ki tu: 5
Ki tu vua nhap vao la 5
Nhap vao mot ki tu: t
Ki tu vua nhap vao la t
```

An phim bat ki de ket thuc chuong trinh...

BÀI 6
HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

6.1. Khái niệm hàm trong C

Khái niệm chương trình con

Trong khi lập trình chúng ta thường gặp những đoạn chương trình lặp đi lặp lại nhiều lần ở những chỗ khác nhau. Để tránh rườm rà và tiết kiệm công sức, những đoạn chương trình đó được thay thế bởi các chương trình con tương ứng và khi cần ta chỉ việc gọi những chương trình con đó ra mà không phải viết lại cả đoạn chương trình đó.

Lấy ví dụ khi giải các bài toán lượng giác ta thường xuyên cần phải tính giá trị sin của đại lượng lượng giác x nào đó. Như vậy ta nên lập một chương trình con tên là sin và tham số là x để tính giá trị sin(x). Mỗi khi cần tính toán giá trị sin của một đại lượng y nào đó thì ta chỉ cần gọi chương trình con sin đã lập sẵn và truyền đại lượng y làm tham số cho chương trình con sin đó thì ta vẫn thu được kết quả mong muốn mà không phải viết lại cả đoạn chương trình tính giá trị sin(y).

Bên cạnh chương trình con sin còn có rất nhiều chương trình con khác được tạo sẵn như cos, exp (dùng để tính lũy thừa cơ số e), pow (tính lũy thừa), sqrt (tính căn bậc 2), ... giúp người lập trình tính toán giá trị của các đại lượng thông dụng. Những chương trình con này nằm trong *thư viện các chương trình con mẫu* và được trình biên dịch C quản lý, vì vậy chúng còn được gọi là các chương trình con chuẩn. Trình biên dịch Turbo C++ phân loại và đặt các chương trình con chuẩn này trong các đơn vị chương trình khác nhau dưới dạng các tệp tiêu đề như stdio.h, conio.h, math.h, string.h...

Ngoài ra còn có một lý do khác cần đến chương trình con. Khi ta giải quyết một bài toán lớn thì chương trình của ta có thể rất lớn và dài, điều này làm cho việc sửa chữa, gỡ rối, hiệu chỉnh chương trình gặp nhiều khó khăn. Nhưng nếu ta chia bài toán lớn, phức tạp ban đầu thành các bài toán con nhỏ hơn và tương đối độc lập với nhau, rồi lập các chương trình con giải quyết từng bài toán con, cuối cùng ghép các chương trình con đó lại thành một chương trình giải quyết bài toán ban đầu thì sẽ rất tiện lợi cho việc phát triển, kiểm tra và sửa chữa cả chương trình.

Việc này tương tự như trong dây chuyền sản xuất công nghiệp khi ta lắp ráp sản phẩm hoàn thiện từ các bán thành phẩm, các module được chế tạo ở những nơi khác nhau. Vì các bán thành phẩm này được chế tạo độc lập nên khi phát hiện lỗi ở module nào ta chỉ việc tìm đến nơi sản xuất ra nó để sửa chữa.

Việc chia nhỏ một chương trình thành các chương trình con đảm nhận những công việc nhỏ khác nhau chính là tư tưởng chính cho phương pháp lập trình có cấu trúc (*structured programming*).

Cần lưu ý là có khi một chương trình con chỉ sử dụng đúng một lần nhưng nó vẫn làm cho chương trình trở nên sáng sủa và dễ đọc, dễ hiểu hơn.

Phân loại chương trình con:

Có 2 loại chương trình con là hàm (*function*) và thủ tục (*procedure*). Sự khác nhau giữa hàm và thủ tục là ở chỗ hàm sau khi thực hiện xong thì sẽ trả về giá trị, còn thủ tục không trả về giá trị gì cả.

Mặc dù vậy hàm và thủ tục là tương đương nhau, tức là có thể xây dựng được thủ tục có chức năng tương đương với một hàm bất kì và có thể xây dựng được hàm có chức năng tương đương với một thủ tục bất kì. Vì thế có những ngôn ngữ lập trình cho phép chương trình con có thể là hàm và thủ tục (Pascal) và có những ngôn ngữ chỉ cho phép chương trình con là hàm mà thôi (như C, Java).

Lưu ý là nếu chương trình con là hàm thì nó luôn có giá trị trả về. Nếu thực sự không có giá trị gì để trả về (nghĩa là nó hoạt động giống thủ tục) thì ta phải khai báo hàm đó có kiểu giá trị trả về là "*không là kiểu giá trị nào cả*" (kiểu **void** trong C).

6.2. Khai báo và sử dụng hàm trong C

6.2.1. Khai báo hàm

Cú pháp khai báo một hàm trong C là như sau

[<kiểu giá trị trả về>] <tên hàm>([<danh sách tham số>,…])

Thân hàm

Khai báo của một hàm được chia làm 2 phần:

- Dòng đầu hàm:

  [<kiểu giá trị trả về>] <tên hàm>([<danh sách tham số>,…])

- Thân hàm: là tập hợp các khai báo và câu lệnh đặt trong cặp dấu ngoặc nhọn

  {

  <Các khai báo>

  …

  <Các câu lệnh>

  }

  Trong thân hàm có ít nhất một lệnh return.

Ví dụ sau là khai báo và định nghĩa hàm tính giai thừa của một số nguyên dương. Ta quy ước rằng giai thừa của một số âm thì bằng –1, của 0 bằng 0, của một số nguyên dương a là $a! = a \times (a\text{-}1) \times \ldots \times 1$.

```
int giai_thua(int a)                    ----------> Dòng đầu hàm
{
    int ket_qua;                        ----------> Các khai báo
    int i;

    ket_qua = 1;
    for(i = 0;i<a;i++)
        ket_qua = ket_qua * i;
    if(a < 0) ket_qua = -1;             ----------> Các câu lệnh
    if(a == 0) ket_qua = 0;
    return ket_qua;
}
```

Các thành phần của dòng đầu hàm

Dòng đầu hàm là các thông tin được trao đổi giữa bên trong và bên ngoài hàm. Khi nói tới dòng đầu hàm là ta nói tới tên của hàm, hàm đó cần những thông tin gì từ môi trường để hoạt động (các tham số đầu vào), hàm đó cung cấp những thông tin gì cho môi trường (những tham số đầu ra và giá trị trả về).

Dòng đầu hàm phân biệt các hàm với nhau, hay nói cách khác không được có 2 hàm có dòng đầu hàm giống nhau.

Kiểu dữ liệu trả về của hàm

Thông thường hàm sau khi được thực hiện sẽ trả về một giá trị kết quả tính toán nào đó. Để sử dụng được giá trị đó ta cần phải biết nó thuộc kiểu dữ liệu gì. Kiểu dữ liệu của đối tượng tính toán được hàm trả về được gọi là kiểu dữ liệu trả về của hàm.

Trong C, kiểu dữ liệu trả về của hàm có thể là kiểu dữ liệu bất kì (kiểu dữ liệu có sẵn hoặc kiểu dữ liệu do người dùng tự định nghĩa) nhưng không được là kiểu dữ liệu mảng.

Nếu kiểu dữ liệu trả về là kiểu **void** thì hàm không trả về giá trị nào cả.

Trường hợp ta không khai báo kiểu dữ liệu trả về thì chương trình dịch của C sẽ ngầm hiểu rằng kiểu dữ liệu trả về của hàm là kiểu **int**.

Tên hàm

Tên hàm là có thể là bất kì một định danh hợp lệ nào. Tuy nhiên tên hàm nên mang nghĩa gợi ý chức năng công việc mà hàm thực hiện. Ví dụ một hàm có chức năng tính và trả về bình phương của một số thực *x* thì nên có tên là *binh_phuong*. Trong C, các hàm không được đặt tên trùng nhau.

Tham số của hàm

Tham số của hàm là các thông tin cần cho hoạt động của hàm và các thông tin, kết quả tính toán được hàm trả lại. Tức là có những tham số chứa dữ liệu vào cung cấp cho hàm, có những tham số chứa dữ liệu ra mà hàm tính toán được.

Các tham số sử dụng trong lời khai báo hàm được gọi là tham số hình thức. Nó là tham số giả định của hàm. Khi khai báo tham số hình thức của hàm phải chỉ ra tên của tham số và kiểu dữ liệu của tham số.

Các tham số được cung cấp cho hàm trong quá trình thực hiện của hàm được gọi là tham số thực. Kiểu dữ liệu của tham số thực cung cấp cho hàm trong chương trình phải giống kiểu dữ liệu của tham số hình thức tương ứng với tham số thực đó, nếu không sẽ có báo lỗi biên dịch.

Một hàm có thể có một, nhiều hoặc không có tham số nào cả. Nếu có nhiều tham số thì chúng phải được phân cách với nhau bằng dấu phẩy. Lưu ý là nếu hàm không có tham số nào cả thì vẫn phải có cặp dấu ngoặc đơn sau tên hàm, ví dụ **main()**.

Lệnh return

Trong chương trình, một hàm được thực hiện khi ta gặp lời gọi hàm của hàm đó trong chương trình. Một lời gọi hàm là tên hàm theo sau bởi các tham số thực trong chương trình. Sau khi hàm thực hiện xong, nó sẽ trở về chương trình đã gọi nó. Có 2 cách để từ hàm trở về chương trình đã gọi hàm:

- Sau khi thực hiện tất cả các câu lệnh có trong thân hàm.
- Khi gặp lệnh **return**.

Cú pháp chung của lệnh **return** là

*return **biểu_thức**;*

Khi gặp lệnh này, chương trình sẽ tính toán giá trị của ***biểu_thức***, lấy kết quả tính toán được làm giá trị trả về cho lời gọi hàm rồi kết thúc việc thực hiện hàm, trở về chương trình đã gọi nó.

Trong lệnh **return** cũng có thể không có phần ***biểu_thức***, khi đó ta sẽ kết thúc thực hiện hàm mà không trả về giá trị nào cả.

Ví dụ và phân tích.

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

```
#include <conio.h>
int max(int x, int y, int z)
{
    int max;
    max = x>y?x:y;
    max = max>z?max:z;
    return max;
}
void main()
{
    int a,b,c;
    clrscr();
    printf("\n Nhap gia tri cho 3 so nguyen a, b, c: ");
    scanf("%d %d %d",&a,&b,&c);
    printf("\n Gia tri cac so vua nhap: ");
    printf(" a = %-5d b = %-5d c = %-5d");
    printf("\n Gia tri lon nhat trong 3 so la %d",max(a,b,c));
    getch();
}
```

6.2.2. Sử dụng hàm

Một hàm sau khi khai báo thì có thể sử dụng. Để sử dụng một hàm (hay còn nói là *gọi hàm*) ta sử dụng cú pháp sau:

<tên hàm> ([danh sách các tham số])

Ví dụ: chương trình dưới đây sẽ khai báo và định nghĩa một hàm có tên là Uscln với 2 tham số đều có kiểu **unsigned int**. Hàm Uscln tìm ước số chung lớn nhất của 2 tham số này theo thuật toán *Euclid* và trả về ước số chung tìm được. Sau đó ta sẽ gọi hàm Uscln trong hàm **main** để tìm ước số chung lớn nhất của 2 số nguyên được nhập từ bàn phím.

```
#include <stdio.h>
#include <conio.h>
unsigned int Uscln(unsigned int a, unsigned int b)
{
    unsigned int u;
    if (a<b)
    {
```

```
        u = a; a = b; b = u;
    }
    do
    {
        u = a%b;
        a = b;
        b = u;
    }while (u!=0);
    return a;
}

int main()
{
    unsigned int a, b;
    do
    {
        printf("\n Nhap vao 2 so nguyen duong a va b ");
        printf("\n a = "); scanf("%d",&a);
        printf("\n b = "); scanf("%d",&b);
        if(a*b == 0)
        {
            printf("\n Khong hop le");
            continue;
        }
        printf("\n Uoc chung lon nhat cua %d va %d la: %d", a, b, Uscln(a, b));
        }while ((a != 0)||(b != 0));
    printf("\n An phim bat ki de ket thuc chuong trinh...");
    getch();
    return 0;
}
```

Kết quả khi thực hiện:

```
Nhap vao 2 so nguyen duong a va b
a = 6
b = 9
```

```
Uoc chung lon nhat cua 6 va 9 la: 3
Nhap vao 2 so nguyen duong a va b
a = 15
b = 26
Uoc chung lon nhat cua 15 va 26 la: 1
Nhap vao 2 so nguyen duong a va b
a = 3
b = 0
Khong hop le
Nhap vao 2 so nguyen duong a va b
a = 0
b = 0
Khong hop le
An phim bat ki de ket thuc chuong trinh...
```

Lưu ý: – Nếu có nhiều tham số trong danh sách tham số thì các tham số được phân cách với nhau bằng dấu phẩy

– Cho dù hàm có một, nhiều hay không có tham số thì vẫn luôn luôn cần cặp dấu ngoặc đơn đứng sau tên hàm

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tương ứng*.
- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.
- Khi gặp lệnh **return** hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

6.2.3. Phân loại biến sử dụng trong chương trình

Phạm vi của các biến

Một biến sau khi khai báo thì có thể được sử dụng trong chương trình. Tuy nhiên tùy vào vị trí khai báo biến mà phạm vi sử dụng các biến sẽ khác nhau. Nguyên tắc sử dụng biến là biến khai báo trong phạm vi nào thì được sử dụng trong phạm vi đó.

Một biến có thể được khai báo trong chương trình chính hoặc trong các chương trình con hoặc thậm chí trong một lệnh khối. Nếu biến được khai báo trong một lệnh khối thì nó chỉ có thể

được gọi trong lệnh khối đó thôi, không thể gọi từ bên ngoài lệnh khối được. Một biến được khai báo trong một chương trình con chỉ có thể được sử dụng trong phạm vi chương trình con đó. Một biến được khai báo trong chương trình chính thì có thể được sử dụng trong toàn bộ chương trình, trong tất cả các chương trình con cũng như trong các lệnh khối của chương trình.

Lưu ý

- Một số ngôn ngữ lập trình như Pascal cho phép khai báo một chương trình con nằm trong một chương trình con khác, nhưng ngôn ngữ C không cho phép khai báo một chương trình con nằm trong một chương trình con khác.
- Bên trong một lệnh khối thì có thể có chứa lệnh khối khác. Khi đó biến được khai báo ở lệnh khối bên ngoài có thể được sử dụng ở lệnh khối bên trong.
- Việc trùng tên của các biến: Trong cùng một phạm vi ta không được phép khai báo 2 biến có cùng tên nhưng ta có thể khai báo 2 biến trùng tên thuộc 2 phạm vi khác nhau. Nếu có 2 biến trùng tên khai báo ở 2 phạm vi khác nhau thì xảy ra 2 trường hợp:
    - Hai phạm vi này tách rời nhau: khi đó các biến sẽ có tác dụng ở phạm vi riêng của nó, không ảnh hưởng đến nhau.
    - Phạm vi này nằm trong phạm vi kia: khi đó nếu chương trình đang ở phạm vi ngoài (tức là đang thực hiện câu lệnh nằm ở phạm vi ngoài) thì biến khai báo ở phạm vi ngoài có tác dụng, còn nếu chương trình đang ở phạm vi trong (đang thực hiện câu lệnh nằm ở phạm vi trong) thì biến khai báo ở phạm vi trong sẽ có tác dụng và nó che lấp biến trùng tên ở bên ngoài.

Ví dụ:

```
#include <stdio.h>
void main()
{
    {
        int a = 1;
        printf("\n a = %d",a);
        {
            int a = 2;
            printf("\n a = %d",a);
        }
        printf("\n a = %d",a);
    }
```

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

Kết quả thực hiện chương trình

```
a = 1
a = 2
a = 1
a = 3
```

Phân loại biến

Theo phạm vi sử dụng, biến chia làm 2 loại: biến cục bộ (biến địa phương – *local variable*) và biến toàn cục (*global variable*).

Biến địa phương

Là các biến được khai báo trong lệnh khối hoặc trong thân chương trình con. Việc khai báo các biến cục bộ phải được đặt trước phần câu lệnh trong lệnh khối hoặc trong chương trình con.

Biến toàn cục

Là biến được khai báo trong chương trình chính. Vị trí khai báo của biến toàn cục là sau phần khai báo tệp tiêu đề và khai báo hàm nguyên mẫu.

Lưu ý: Hàm **main()** cũng chỉ là một chương trình con, nhưng nó là chương trình con đặc biệt ở chỗ chương trình được bắt đầu thực hiện từ hàm **main()**.

Biến khai báo trong hàm **main()** không phải là biến toàn cục mà là biến cục bộ của hàm **main()**.

Một số lệnh đặc trưng của C: register, static

Chúng ta biết rằng các thanh ghi có tốc độ truy nhập nhanh hơn so với các loại bộ nhớ khác (RAM, bộ nhớ ngoài), do vậy nếu một biến thường xuyên sử dụng trong chương trình được lưu vào trong thanh ghi thì tốc độ thực hiện của chương trình sẽ được tăng lên. Để làm điều này ta đặt từ khóa **register** trước khai báo của biến đó.

Ví dụ

```
register int a;
```

Tuy nhiên có một lưu ý khi khai báo biến **register** là vì số lượng các thanh ghi có hạn và kích thước của các thanh ghi cũng rất hạn chế (ví dụ trên dòng máy 80x86, các thanh ghi có kích

thước 16 bit = 2 byte) cho nên số lượng biến khai báo **register** sẽ không nhiều và thường chỉ áp dụng với những biến có kích thước nhỏ như kiểu **char**, **int**.

Như ta đã biết, một biến cục bộ khi ra khỏi phạm vi của biến đó thì bộ nhớ dành để lưu trữ biến đó sẽ được giải phóng. Tuy nhiên trong một số trường hợp ta cần lưu giá trị của các biến cục bộ này để phục vụ cho những tính toán sau này, khi đó ta hãy khai báo biến với từ khóa **static** ở đầu.

Ví dụ

```
static int a;
```

Từ khóa **static** giúp chương trình dịch biết được đây là một biến tĩnh, nghĩa là nó sẽ được cấp phát một vùng nhớ thường xuyên từ lúc khai báo và chỉ giải phóng khi chương trình chính kết thúc. Như vậy về thời gian tồn tại biến **static** rất giống với biến toàn cục, chỉ có một sự khác biệt nhỏ là biến toàn cục thì có thể truy cập ở mọi nơi trong chương trình (miễn là ở đó không có biến địa phương nào cùng tên che lấp nó), còn biến **static** thì chỉ có thể truy nhập trong phạm vi mà nó được khai báo mà thôi.

Hãy xét ví dụ sau:

```
# include <stdio.h>
# include <conio.h>
void fct()
{
    static int count = 1;
    printf("\n Day la lan goi ham fct lan thu %2d", count++);
}
void main()
{
    int i;
    for(i = 0;  i < 10; i++)
        fct();
    getch();
}
```

Kết quả khi thực hiện

```
Day la lan goi ham fct lan thu  1
Day la lan goi ham fct lan thu  2
Day la lan goi ham fct lan thu  3
Day la lan goi ham fct lan thu  4
```

```
Day la lan goi ham fct lan thu  5
Day la lan goi ham fct lan thu  6
Day la lan goi ham fct lan thu  7
Day la lan goi ham fct lan thu  8
Day la lan goi ham fct lan thu  9
Day la lan goi ham fct lan thu 10
```

6.2.4. Nguyên mẫu hàm trong C.

Ví dụ về nguyên mẫu hàm

```
#include <stdio.h>
#include <conio.h>
int max(int, int, int);    // khai báo nguyên mẫu hàm
void main()
{
      int a,b,c;
      clrscr();
      printf("\n Nhap gia tri cho 3 so nguyen a, b, c: ");
      scanf("%d %d %d",&a,&b,&c);
      printf("\n Gia tri cac so vua nhap: ");
      printf(" a = %-5d b = %-5d c = %-5d");
      printf("\n Gia tri lon nhat trong 3 so la %d",max(a,b,c));
      getch();
}
int max(int x, int y, int z)
{
      int max;
      max = x>y?x:y;
      max = max>z?max:z;
      return max;
}
```

Trong trường hợp ta muốn đặt phần định nghĩa của hàm nằm sau hàm **main()** thì ta phải khai báo nguyên mẫu của hàm để báo cho chương trình dịch biết có một hàm có dòng đầu hàm giống như trong phần nguyên mẫu này. Nhờ đó chương trình dịch có thể kiểm tra được là các lời gọi hàm trong chương trình chính có đúng hay không (có phù hợp về kiểu dữ liệu trả về, các tham số thực có kiểu dữ liệu có phù hợp với kiểu dữ liệu đã khai báo hay không).

Trong hàm nguyên mẫu ta có thể không cần nêu tên các tham số hình thức, nhưng trong phần định nghĩa hàm ta cần phải có các tham số hình thức.

6.3. Con trỏ và địa chỉ

Từ khi bắt đầu môn học cho đến giờ, chúng ta chỉ truy nhập, tác động hay thay đổi nội dung của các biến một cách trực tiếp – qua tên của chúng. Tuy nhiên, ngôn ngữ C cung cấp cho người lập trình một phương tiện gián tiếp khác đôi khi rất tiện dụng để truy xuất đến biến, đó là các con trỏ.

6.3.1. Khái niệm con trỏ

Địa chỉ và giá trị của một biến

Bộ nhớ có thể hiểu như một dãy các byte nhớ được xác định một cách duy nhất qua một *địa chỉ*. Tất cả các biến trong một chương trình được lưu ở một vùng nào đó trong bộ nhớ.

Khi chúng ta khai báo một biến, chương trình dịch sẽ cấp phát cho biến đó một số ô nhớ liên tiếp đủ để chứa nội dung của biến, ví dụ một biến ký tự được cấp phát 1 byte, một biến nguyên được cấp phát 2 byte, một biến thực được cấp phát 4 byte .v.v Địa chỉ của một biến chính là địa chỉ của byte đầu tiên trong số đó.

Một biến luôn có hai đặc tính:

- Địa chỉ của biến.
- Giá trị của biến.

Xét ví dụ sau:

```
int i, j;
i = 3;
j = i;
```

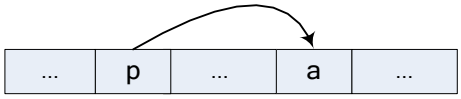
Giả sử chương trình dịch đặt biến i tại địa chỉ FFEC trong bộ nhớ, và biến j ở địa chỉ FFEE, chúng ta có

biến	địa chỉ	giá trị
i	FFEC	3
j	FFEE	3

Hai biến khác nhau có địa chỉ khác nhau. Phép gán i = j; chỉ có tác dụng trên giá trị của các biến, nó chỉ có ý nghĩa nội dung của vùng nhớ lưu trữ tương ứng biến j được sao chép sang nội dung vùng nhớ dành cho i. Hai biến i, j có kiểu nguyên, chúng được lưu trong hai byte.

Khái niệm con trỏ

Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ..Vùng nhớ này có thể chứa các biến thuộc các kiểu dữ liệu cơ sở như int, char, hay double hoặc dữ liệu có cấu trúc như mảng.



Cú pháp khai báo một con trỏ như sau:

Kiểu_dữ_liệu *tên_contrỏ;

Câu lệnh này khai báo một định danh tên_contrỏ gắn với một biến con trỏ có giá trị là địa chỉ của một biến có kiểu chỉ định. Ký tự * xác định rằng một biến con trỏ đang được khai báo. Giá trị của biến con trỏ hoàn toàn có thể thay đổi được.

Kiểu của một con trỏ phụ thuộc vào kiểu biến mà nó trỏ đến. Trong ví dụ sau, ta định nghĩa con trỏ p trỏ đến biến nguyên i:

```
int i = 3;
int *p;
p = &i;
```

Khi gán địa chỉ của i cho p, ta nói rằng p trỏ đến i. Và giá trị của các biến này trong bộ nhớ sẽ là

biến	**địa chỉ**	**giá trị**
i	FFEC	3
p	FFEE	FFEC

Chú ý: một con trỏ chỉ có thể trỏ tới một đối tượng cùng kiểu. Không thể dùng con trỏ float để trỏ vào một biến kiểu int hay ngược lại.

*Toán tử & và **

Có hai toán tử đặc biệt được dùng với con trỏ: & và *. Toán tử & là một toán tử một ngôi và nó trả về địa chỉ của biến. Toán tử thứ hai, toán tử * là một toán tử một ngôi và trả về giá trị chứa trong vùng nhớ được trỏ bởi giá trị của biến con trỏ. Trong ví dụ sau chương trình:

```
main()
{
 int i = 3;
 int *p;

 p = &i;
```

```
 printf("*p = %d \n",*p);
}
```

hiển thị ra màn hình *p = 3. Trong chương trình này i và *p là tương đương. Mọi thay đổi trên *p sẽ thay đổi luôn i. Ví dụ nếu ta gán *p = 0 thì i sẽ có giá trị là 0.

Cả hai toán tử * và & có độ ưu tiên cao hơn tất cả các toán tử số học ngoại trừ toán tử đảo dấu. Chúng có cùng độ ưu tiên với toán tử lấy giá trị âm.

Gán giá trị cho con trỏ

Một cách tổng quát, một biến con trỏ có thể được gán bởi địa chỉ của một biến khác:

```
pointer_variable = &variable;
```

Hay bởi giá trị của một con trỏ khác (tốt nhất là cùng kiểu):

```
pointer_variable2 = pointer_variable;
```

Giá trị NULL cũng có thể được gán đến một biến con trỏ bằng số 0 như sau:

```
pointer_variable = 0;
```

Và cuối cùng, các biến cũng có thể được gán giá trị thông qua con trỏ của chúng.

```
*pointer_variable = 10;
```

Nói chung, các biểu thức có chứa con trỏ cũng theo cùng quy luật như các biểu thức khác trong C. Điều quan trọng cần chú ý phải gán giá trị cho biến con trỏ trước khi sử dụng chúng; nếu không chúng có thể trỏ đến một giá trị không xác định nào đó. Trong chương trình chúng ta có thể thao tác đồng thời trên con trỏ p và *p, nhưng ý nghĩa của chúng là rất khác nhau. Quan sát hai chương trình sau:

```
main()
{
 int i = 3, j = 6;
 int *p1, *p2;
 p1 = &i;
 p2 = &j;
 *p1 = *p2;
}
```

và

```
main()
{
 int i = 3, j = 6;
 int *p1, *p2;
 p1 = &i;
```

```
  p2 = &j;
  p1 = p2;
}
```

Trước lệnh gán cuối cùng, giả sử giá trị các biến trong bộ nhớ như sau:

biến	**địa chỉ**	**giá trị**
i	FFEC	3
j	FFEE	6
p1	FFDA	FFEC
p2	FFDC	FFEE

Sau lệnh gán *p1 = *p2; ở chương trình thứ nhất:

biến	**địa chỉ**	**giá trị**
i	FFEC	6
j	FFEE	6
p1	FFDA	FFEC
p2	FFDC	FFEE

Trong khi đó lệnh gán p1 = p2 trong chương trình thứ hai sẽ dẫn tới kết quả sau :

biến	**địa chỉ**	**giá trị**
i	FFEC	3
j	FFEE	6
p1	FFDA	FFEE
p2	FFDC	FFEE

6.3.2. Các phép toán làm việc liên quan đến biến con trỏ

Một điểm mạnh của ngôn ngữ C là khả năng thực hiện tính toán trên các con trỏ. Các phép toán số học có thể thực hiện trên con trỏ là:

- Cộng con trỏ với một số nguyên (int, long) và kết quả là một con trỏ cùng kiểu.
- Trừ con trỏ với một số nguyên và kết quả là một con trỏ cùng kiểu.
- Trừ hai con trỏ cùng kiểu cho nhau, kết quả là một số nguyên. Kết quả này nói lên khoảng cách (số phần tử thuộc kiểu dữ liệu của con trỏ) ở giữa hai con trỏ.

Chú ý là phép toán cộng hai con trỏ, và nhân chia, lấy phần dư trên con trỏ là không hợp lệ. Mỗi khi con trỏ được tăng giá trị, nó sẽ trỏ đến ô nhớ của phần tử kế tiếp. Mỗi khi nó được giảm giá trị, nó sẽ trỏ đến vị trí của phần tử đứng trước nó.

Xét ví dụ sau:

```
int x, *p1, *p2;
p1= &x;
```

p2= p1+1;

Khi đó p2 sẽ trỏ tới số nguyên nằm ngay kề sau x trong bộ nhớ. Cũng phải chú ý rằng mặc dù 1 chỉ là khoảng cách tương đối tính theo số phần tử kiểu int giữa p1 và p2, còn thực tế giá trị địa chỉ theo byte của p2 hơn p1 là 2. Nếu p2, p1 là con trỏ kiểu float thì khoảng cách sẽ là 4, chính là kích thước của kiểu dữ liệu trỏ bởi con trỏ.

Con trỏ void

Được khai báo như sau:

void *con_trỏ;

Đây là con trỏ đặc biệt, con trỏ không có kiểu, nó có thể nhận giá trị là địa chỉ của một biến thuộc bất kỳ kiểu dữ liệu nào. Con trỏ void được dùng làm đối để nhận bất kỳ địa chỉ nào từ tham số của các lời gọi hàm. Các lệnh sau đây là hợp lệ:

```
void *p, *q;
int x = 21;
float y = 34.34;
p = &x; q = &y;
```

6.4. Sử dụng con trỏ trong làm việc với mảng

Con trỏ và mảng một chiều

Trên thực tế a chính là một hằng địa chỉ và là địa chỉ của phần tử đầu tiên của mảng. Với một mảng có tên là a thì a là một địa chỉ và a có giá trị bằng &a[0].

Như vậy nếu ta khởi tạo một con trỏ p với giá trị bằng địa chỉ của một mảng, ta có thể dùng con trỏ này để duyệt qua các phần từ trong mảng.

Xét khai báo mảng tab gồm 10 số nguyên sau:

int a[10], *p;

nếu p = a; khi đó

p+1 sẽ trỏ tới phần tử cùng kiểu ngay sau phần tử đầu tiên của mảng, chính là a[1], nghĩa là *(p+1) chính là a[1].

p+2 sẽ trỏ tới a[2], hay *(p+2) là a[2]

….

p+i sẽ trỏ tới a[i]

Ví dụ sau đây minh họa việc sử dụng con trỏ để duyệt mảng một chiều

```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
main()
{
```

```
 int i;
 int *p;
 p = tab;
 for (i = 0; i < N; i++)
   {
    printf(" %d \n",*p);
    p++;
   }
}
```

Ngoài ký pháp *(p+i) để chỉ nội dung phần tử thứ i tính từ đầu mảng (trỏ bởi p), ngôn ngữ C còn cho phép chúng ta sử dụng chỉ số i trên con trỏ. Cụ thể là

p[i] có vai trò như *(p+i) và cũng là phần tử thứ i của mảng được trỏ bởi p. Ví dụ trước có thể được viết lại như sau:

```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
main()
{
 int i;
 int *p;
 p = tab;
 for (i = 0; i < N; i++)
   printf(" %d \n", p[i]);
}
```

Lưu ý:

- Con trỏ luôn cần phải được khởi tạo, hoặc bằng cách gán cho nó một địa chỉ nào đó, hoặc qua thao tác cấp phát động bộ nhớ.

Một (tên) mảng là một hằng địa chỉ, nó không bao giờ có thể nằm ở vế trái của một phép gán như con trỏ, cũng như chấp nhận các phép toán số học trên nó ví dụ như tab++;

Con trỏ và mảng nhiều chiều

Con trỏ và mảng 2 chiều

Phần trên chúng ta đã tìm hiểu về con trỏ và mảng 1 chiều, và phần này con trỏ và mảng 2 chiều cũng tương tự như vậy.

Như ta đã biết thực chất trong máy tính thì bộ nhớ lưu mảng 2 chiều giống như mảng 1 chiều. Vì vậy ta hoàn toàn có thể biểu diễn mảng 2 chiều bằng con trỏ giống như mảng 1 chiều.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int main()
05  {
06     double a[10][10], *pa;
07     int n, m, i;
08     pa = (double *) a;
09     printf("Nhap so hang va so cot:\n");
10     scanf("%d %d", &n, &m);
11
12     for (i = 0 ; i < n * m; i++)
13     {
14        printf("Nhap a[%d][%d] = ", i / m, i % m);
15        scanf("%lf", pa + i);
16     }
17
18     for (i = 0 ; i < n * m; i++)
19     {
20        if (i % m == 0) printf("\n"); // xuong dong
21        printf("%-5.2lf", *(pa + i));
22     }
23
24     return 0;
   }
```

Kết quả:

```
Nhap so hang va so cot:
2
3
Nhap a[0][0] = 4.23
Nhap a[0][1] = 5.7
Nhap a[0][2] = 1.2
Nhap a[1][0] = 8.6
```

```
Nhap                    a[1][1]                    =                    3.456
Nhap a[1][2] = 12
4.23                                5.70                                1.20
8.60 3.46 12.00
```

6.5 Mảng con trỏ

Mảng con trỏ là một mảng chứa tập hợp các con trỏ cùng một kiểu.

Float *a[10]; // khai báo một mảng con trỏ. Gồm 10 con trỏ: a[0], a[1], …a[9]; là 10 con trỏ. Liên hệ với mảng 2 chiều thì ta có nhận xét sau: Mỗi hàng của mảng 2 chiều ta coi như 1 mảng 1 chiều. Mà mỗi con trỏ thì lại có mối quan hệ với 1 mảng 1 chiều

```
//Vi du ve mang con tro
#include <stdio.h>
#include <stdlib.h>

int main()
{
   double a[10][10], *pa[10];
   int n, m, i, j;

   printf("Nhap so hang va so cot: ");
   scanf("%d %d", &n, &m);

   for (i = 0 ; i < n; i++)
   {
      pa[i] = a[i]; // con tro thu i tro den hang thu i
      for (j = 0 ; j < m; j++)
      {
         printf("Nhap a[%d][%d] = ", i, j);
         scanf("%lf", &pa[i][j]);
      }
   }

   for (i = 0 ; i < n; i++)
   {
      printf("\n"); // xuong dong
```

```
    for (j = 0 ; j < m; j++)
    {
       printf("%-5.2lf", pa[i][j]);
    }
  }

  return 0;
}
```

Sở dĩ ở VD này ta viết được pa[i][j] đó là do mỗi pa là 1 con trỏ đến mảng một chiều. pa[i][j] tức là phần tử thứ j của con trỏ pa[i].

Các ví dụ ở trên chúng ta đều xét khi mà khai báo mảng a[][] nên không cần cấp phát bộ nhớ cho con trỏ, bây giờ muốn cấp phát bộ nhớ cho con trỏ với mảng 2 chiều giống như cấp phát ở mảng 1 chiều thì ta làm như sau:

```
01    //Vi du mang con tro
02    #include <stdio.h>
03    #include <stdlib.h>
04
05    int main()
06    {
07       double **pa;
08       int n, m, i, j;
09
10       printf("Nhap so hang va so cot: ");
11       scanf("%d %d", &n, &m);
12
13       // cap phat n o nho cho n con tro (n hang)
14       pa = (double**) malloc(n * sizeof(double));
15
16       for (i = 0 ; i < n; i++)
17       {
18          // cap phat m o nho cho moi con tro (moi hang)
19          pa[i] = (double *) malloc(m * sizeof(double));
20          for (j = 0 ; j < m; j++)
21          {
```

```
22              printf("Nhap a[%d][%d] = ", i, j);
23              scanf("%lf", &pa[i][j]);
24          }
25      }
26
27      for (i = 0 ; i < n; i++)
28      {
29          printf("\n"); // xuong dong
30          for (j = 0 ; j < m; j++)
31          {
                printf("%-5.2lf", pa[i][j]);
            }
        }

        return 0;
    }
```

Đây cũng có thể coi là mảng con trỏ, hoặc con trỏ trỏ đến con trỏ (con trỏ đa cấp).

6.6 Đệ quy

Đệ quy trong C là quá trình trong đó một phương thức gọi lại chính nó một cách liên tiếp. Một phương thức trong C gọi lại chính nó được gọi là phương thức đệ quy.

Sử dụng đệ quy giúp code chặt chẽ hơn nhưng sẽ khó để hiểu hơn.

Cú pháp:

```
kieu_tra_ve tenhamdequi() {
  // your code
  tenhamdequi(); /* goi lai chinh no */
}

int main() {
  tenhamdequi();
}
```

Ví dụ : Tính giai thừa

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

```
2
3       int factorial(int n) {
4           if (n == 1)
5               return 1;
6           else
7               return (n * factorial(n - 1));
8       }
9
10      int main() {
11          printf("Giai thua cua 5 la: %d", factorial(5));
12          return 0;
13      }
```

Kết quả:

```
Giai thừa của 5 là: 120
```

BÀI 7
CẤU TRÚC VÀ HỢP

7.1. Khái niệm cấu trúc

Kiểu dữ liệu cấu trúc (**struct**) là kiểu dữ liệu phức hợp bao gồm nhiều thành phần, mỗi thành phần có thể thuộc những kiểu dữ liệu khác nhau.

Các thành phần dữ liệu trong cấu trúc được gọi là các trường dữ liệu (*field*).

Ví dụ: khi cần lưu giữ thông tin về một dạng đối tượng nào đó như đối tượng sinh viên chẳng hạn, ta lưu giữ các thông tin liên quan đến sinh viên như họ tên, tuổi, kết quả học tập… Mỗi thông tin thành phần lại có kiểu dữ liệu khác nhau như họ tên có kiểu dữ liệu là xâu kí tự, tuổi có kiểu dữ liệu là số nguyên, kết quả học tập có kiểu dữ liệu là số thực.

7.2. Khai báo và sử dụng cấu trúc

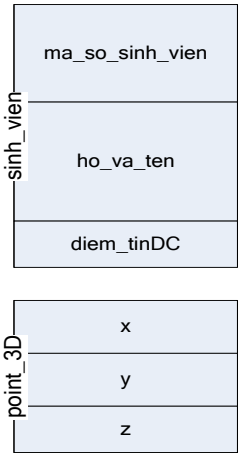
Khai báo kiểu dữ liệu cấu trúc:

Để khai báo một kiểu dữ liệu cấu trúc ta dùng cú pháp khai báo sau:

```
struct tên_cấu_trúc
{
        <khai báo các trường dữ liệu>;
};
```

Ví dụ:

```
struct sinh_vien
{
  char ma_so_sinh_vien[10];
  char ho_va_ten[30];
  float diem_TinDC;
}
struct point_3D
{
  float x;
  float y;
  float z;
}
```



Khai báo thứ nhất là khai báo của một kiểu dữ liệu cấu trúc có tên là sinh_vien gồm có 3 trường dữ liệu là ma_so_sinh_vien kiểu xâu kí tự, ho_va_ten kiểu xâu kí tự và diem_TinDC kiểu số thực **float**.

Ở khai báo thứ 2 ta đã khai báo một kiểu dữ liệu cấu trúc có tên là point_3D gồm có 3 trường và cả 3 trường này đều có cùng kiểu dữ liệu số thực **float**. Vì 3 trường này cùng kiểu dữ liệu nên ta có thể sử dụng mảng để thay thế cấu trúc, tuy nhiên việc sử dụng cấu trúc để mô tả dữ liệu điểm 3 chiều sẽ giúp sự mô tả được tự nhiên hơn, "thật" hơn (nghĩa là trong cấu trúc các tọa độ vẫn độc lập với nhau, nhưng nhìn từ bên ngoài thì các toạ độ này lại là một thể thống nhất cung cấp thông tin về vị trí của một điểm. Còn nếu sử dụng mảng thì các tọa độ của một điểm độc lập và rời rạc với nhau, không thấy mối liên hệ. Khi đó ta sẽ phải tự mình ngầm quy ước rằng các phần tử của mảng có chỉ số là 3**k*, 3**k*+1 và 3**k*+2 với *k* = 0, 1, 2, … là tọa độ của một điểm).

Khai báo biến cấu trúc:

Để khai báo biến cấu trúc ta dùng cú pháp khai báo sau

struct tên_cấu_trúc tên_biến_cấu_trúc;

Ví dụ:

struct sinh_vien a, b, c;

Câu lệnh trên khai báo 3 biến lần lượt tên là a, b, c có kiểu dữ liệu là cấu trúc sinh_vien.

Tuy nhiên ta cũng có thể kết hợp đồng thời vừa khai báo kiểu dữ liệu cấu trúc vừa khai báo biến cấu trúc bằng cách sử dụng cú pháp khai báo sau:

struct [tên_cấu_trúc]

{

<khai báo các trường>;

} tên_biến_cấu_trúc;

Theo cú pháp khai báo trên thì ta thấy phần tên_cấu_trúc có thể có hoặc không. Nếu có tên_cấu_trúc thì sau này ta có thể khai báo bổ sung biến có kiểu dữ liệu là tên_cấu_trúc đó, còn nếu không có tên_cấu_trúc thì cấu trúc khai báo tương ứng không được sử dụng về sau.

Ví dụ:

```
struct diem_thi
{
    float diem_Toan;
    float diem_Ly;
    float diem_Hoa;
}
struct thi_sinh
{
    char SBD[10];          // số báo danh
```

```
        char ho_va_ten[30];
        struct diem_thi ket_qua;
    } thi_sinh_1, thi_sinh_2;
```

Qua ví dụ trên ta cũng thấy rằng các cấu trúc có thể lồng nhau, nghĩa là cấu trúc này có thể là trường dữ liệu của cấu trúc khác, và mức độ lồng là không hạn chế. Để tăng thêm sự tiện lợi, ngôn ngữ C còn cho phép khai báo trực tiếp trường dữ liệu là cấu trúc bên trong cấu trúc chứa nó, vì thế ta có thể viết lại ví dụ ở trên như sau

```
    struct thi_sinh
    {
        char SBD[10];
        char ho_va_ten[30];
        struct diem_thi
        {
            float diem_Toan;
            float diem_Ly;
            float diem_Hoa;
        }ket_qua;
    } thi_sinh_1, thi_sinh_2;
```

Định nghĩa kiểu dữ liệu cấu trúc với typedef

Trong các ví dụ trước ta thấy một khai báo biến cấu trúc bắt đầu bằng từ khóa **struct**, sau đó đến tên cấu trúc rồi mới đến tên biến. Cách khai báo này có phần "rắc rối" hơn so với khai báo biến thông thường và có không ít trường hợp người lập trình quên đặt từ khóa **struct** ở đầu. Để tránh điều đó, ngôn ngữ C cho phép ta đặt tên mới cho kiểu dữ liệu cấu trúc bằng câu lệnh **typedef** có cú pháp khai báo như sau :

typedef struct tên_cũ tên_mới;

Hoặc ta có thể đặt lại tên cho cấu trúc ngay khi khai báo bằng cú pháp

```
    typedef struct [tên_cũ]
    {
        <khai báo các trường>;
    }danh_sách_các_tên_mới;
```

Sau câu lệnh này ta có thể sử dụng tên_mới thay cho tổ hợp struct tên_cũ khi khai báo biến.

Lưu ý: Được phép đặt tên_mới trùng với tên_cũ.

Ví dụ:

```
        struct point_3D
```

```
{
    float x, y, z;
} P;
struct point_3D M;
typedef struct point_3D point_3D;
point_3D N;
```

Trong ví dụ trên ta đã đặt lại tên cho cấu trúc struct point_3D thành point_3D và dùng tên mới này làm kiểu dữ liệu cho khai báo của biến N. Các biến P, M được khai báo theo cách chúng ta đã biết.

Ví dụ:

```
typedef struct point_2D
{
    float x, y;
}point_2D, diem_2_chieu, ten_bat_ki;
point_2D X;
diem_2_chieu Y;
ten_bat_ki Z;
```

Với ví dụ này ta cần chú ý là point_2D, diem_2_chieu và ten_bat_ki không phải là tên biến mà là tên mới của cấu trúc struct point_2D. Các biến X, Y, Z được khai báo với kiểu dữ liệu là các tên mới này.

7.3. Xử lí dữ liệu cấu trúc

Truy nhập các trường dữ liệu của cấu trúc

Dữ liệu của một biến cấu trúc bao gồm nhiều trường dữ liệu, và các trường dữ liệu này độc lập với nhau. Do đó muốn thay đổi nội dung dữ liệu bên trong một biến cấu trúc ta cần truy nhập tới từng trường và thực hiện thao tác cần thiết trên từng trường đó. Để truy nhập tới một trường trong cấu trúc ta dùng cú pháp sau:

tên_biến_cấu_trúc.tên_trường

Dấu chấm ***"."*** sử dụng trong cú pháp trên là toán tử truy nhập thành phần cấu trúc, và nếu như trường được truy nhập lại là một cấu trúc thì ta có thể tiếp tục áp dụng toán tử này để truy nhập tới các trường thành phần nằm ở mức sâu hơn.

Giờ đây ta có thể "đối xử" tên_biến_cấu_trúc.tên_trường giống như một biến thông thường có kiểu dữ liệu là kiểu dữ liệu của tên_trường, tức là ta có thể nhập giá trị, hiển thị giá trị của biến cấu trúc, sử dụng giá trị đó trong các biểu thức…

Ví dụ:

```
// dưới đây là một cấu trúc mô tả một điểm trong không gian 2 chiều.
// các trường dữ liệu gồm: tên của điểm và tọa độ của điểm đó.
// tọa độ là một cấu trúc gồm 2 trường: hoành độ và tung độ
#include <stdio.h>
#include <conio.h>
void main()
{
    struct point_2D
    {
        char ten_diem;
        struct
        {
            float x, y;
        } toa_do;
    } p;
    float temp_float;
    char temp_char;
    printf("\n Hay nhap thong tin ve mot diem");
    printf("\n Ten cua diem: ");
    fflush(stdin);
    scanf("%c",&temp_char);
    p.ten_diem = temp_char;
    printf("\n nhap vao hoanh do cua diem: ");
    scanf("%f",&temp_float);
    p.toa_do.x = temp_float;
    // giả sử điểm đang xét nằm trên đường thẳng y = 3x + 2;
    p.toa_do.y = 3*p.toa_do.x + 2;
    printf("\n %c = (%5.2f,%5.2f)",p.ten_diem, p.toa_do.x, p.toa_do.y);
    getch();
}
```

Kết quả khi chạy chương trình:

```
Hay nhap thong tin ve mot diem
Ten cua diem: A
nhap vao hoanh do cua diem: 5
```

A = (5.00,17.00)

Lưu ý: Cũng như việc nhập giá trị cho các phần tử của mảng, việc nhập giá trị cho các trường của cấu trúc (đặc biệt là các trường có kiểu dữ liệu **float**) nên thực hiện qua biến trung gian để tránh những tình huống có thể làm treo máy hoặc kết quả nhập được không như ý.

Phép gán giữa các biến cấu trúc

Giả sử ta có 2 biến cấu trúc là *a* và *b* có cùng kiểu dữ liệu là một cấu trúc nào đó, và giả sử các trường dữ liệu của *a* đều đã được khởi tạo (tức là giá trị của các trường dữ liệu đó đều đã được xác định). Giờ đây ta cũng muốn giá trị các trường dữ liệu của *b* có giá trị giống với các trường dữ liệu tương ứng của *a*. Ta có thể thực hiện điều đó bằng cách gán giá trị từng trường của *a* cho các trường tương ứng của *b*. Cách này có vẻ rất "thủ công" và rất bất tiện nếu như trong cấu trúc có nhiều trường dữ liệu. Do vậy C cung cấp cho ta một phương tiện để thực hiện việc này như cách thứ 2, đó là sử dung phép gán các biến cấu trúc. Phép gán cấu trúc có cú pháp tương tự như phép gán thông thường

biến_cấu_trúc_1 = biến_cấu_trúc_2;

Câu lệnh trên sẽ gán giá trị của các trường trong biến_cấu_trúc_2 cho các trường tương ứng trong biến_cấu_trúc_1.

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
    struct s
    {
        char ho_ten[20];
        float diem;
    }a, b, c;
    float temp_f;
    printf("\na.ho_ten: ");fflush(stdin);
    gets(a.ho_ten);
    printf("\na.diem = ");scanf("%f",&temp_f);
    a.diem = temp_f;
    strcpy(c.ho_ten, a.ho_ten);
```

```
    c.diem = a.diem;
    b = a;
    printf("\na: %20s %5.2f", a.ho_ten, a.diem);
    printf("\nb: %20s %5.2f", b.ho_ten, b.diem);
    printf("\nc: %20s %5.2f\n", c.ho_ten, c.diem);
}
```

Kết quả thực hiện

```
a.ho_ten: nguyen van minh
a.diem = 7.5
a:      nguyen van minh  7.50
b:      nguyen van minh  7.50
c:      nguyen van minh  7.50
```

Trong chương trình trên ta đã nhập giá trị cho các trường của biến cấu trúc a từ bàn phím, sau đó copy dữ liệu từ biến a sang biến c bằng cách sao chép từng trường, và copy dữ liệu từ biến a sang biến b bằng cách dùng lệnh gán. Kết quả là như nhau và rõ ràng cách thứ 2 ngắn gọn hơn.

Lưu ý : Để copy dữ liệu là xâu kí tự ta phải dùng lệnh strcpy(), không được dùng lệnh gán thông thường để copy nội dung xâu kí tự.

Con trỏ cấu trúc

Tương tự con trỏ kiểu **int** chứa địa chỉ của một biến kiểu **int**, con trỏ kiểu **float** chứa địa chỉ của một biến kiểu **float** thì con trỏ cấu trúc cũng chứa địa chỉ của một cấu trúc.

Để khai báo một biến con trỏ cấu trúc ta dùng cú pháp khai báo

struct <tên cấu trúc> * <tên biến con trỏ cấu trúc> ;

Có 2 cách truy nhập vào trường dữ liệu của cấu trúc từ biến con trỏ cấu trúc là

(*<tên biến con trỏ cấu trúc>).<tên trường dữ liệu>

<tên biến con trỏ cấu trúc>-><tên trường dữ liệu>

7.4. Mảng cấu trúc

Trong phần trước ta đã biết đến mảng các số nguyên, mảng các số thực… là tập hợp các phần tử có cùng kiểu dữ liệu là số nguyên (hoặc số thực nếu là mảng số thực) và được lưu trữ trên những ô nhớ kế tiếp nhau. Nếu như kiểu dữ liệu của các phần tử là kiểu cấu trúc thì khi đó ta sẽ có mảng cấu trúc, đó là tập hợp các biến cấu trúc (cùng loại cấu trúc) và được lưu trữ trên những ô nhớ kế tiếp nhau.

Để khai báo một biến mảng cấu trúc ta sử dụng cú pháp khai báo sau:

struct <tên cấu trúc> <tên mảng cấu trúc> [số phần tử];

Ví dụ:

```
struct sinh_vien
{
    char ho_ten[20];
    float diem_thi;
};
struct sinh_vien lop_KHMT[50];
```

Câu lệnh trên khai báo một biến mảng tên là lop_KHMT gồm 50 phần tử, trong đó mỗi phần tử đều có kiểu dữ liệu là kiểu cấu trúc sinh_vien.

Nhìn chung mảng cấu trúc cũng giống như mảng các số nguyên hay mảng các số thực, duy chỉ có điều lưu ý là một phần tử của mảng cấu trúc là một biến cấu trúc, do vậy để truy nhập vào dữ liệu thực sự trong mảng cấu trúc thì ta phải qua 2 bước truy nhập là truy nhập vào phần tử của mảng, sau đó truy nhập vào trường dữ liệu quan tâm.

Mảng cấu trúc là một dạng cấu trúc dữ liệu rất hay gặp trong thực tế. Khi ta muốn lưu thông tin về nhiều đối tượng khác nhau và thông tin về mỗi đối tượng lại gồm nhiều khía cạnh khác nhau thì không có gì thích hợp hơn là sử dụng một mảng các cấu trúc trong đó mỗi phần tử của mảng lưu thông tin về một đối tượng, cả mảng sẽ lưu thông tin của cả nhóm đối tượng. Sau đây sẽ là một ví dụ tổng hợp, một chương trình nhỏ cho phép ta nhập vào các thông tin đơn giản liên quan đến các sinh viên như mã số sinh viên, họ tên, điểm thi, sau đó hiển thị những thông tin vừa nhập được.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
    struct sinh_vien
    {
        char ma_sv[10];
        char ho_ten[20];
        float diem_thi;
    };
    struct sinh_vien sv[3];
    int i;
    clrscr();
```

```
    for(i=0;i<3;i++)
    {
        char str[20];
        float diem;
        printf("\n Nhap thong tin cho sinh vien thu %d",i+1);
        printf("\n Ma so sinh vien:");
        fflush(stdin); gets(str);
        strcpy(sv[i].ma_sv,str);
        printf("\n Ho va ten: ");
        fflush(stdin); gets(str);
        strcp(sv[i].ho_ten,str);
        printf("\n      Diem thi: ");
        scanf("%f",&diem);
        sv[i].diem_thi = diem;
    }
    printf("\n Thong tin ve cac sinh vien");
    for(i=0;i<3;i++)
    {
        printf("\n Sinh vien thu %d ",i+1);
        printf("%-10s %-20s %-3.1f",sv[i].ma_sv, sv[i].ho_ten, sv[i].diem_thi);
    }
    getch();
}
```

Kết quả:

```
Nhap thong tin cho sinh vien thu 1
Ma so sinh vien: SV0032
Ho va ten: Nguyen Thanh Binh
Diem thi: 8.5
Nhap thong tin cho sinh vien thu 2
Ma so sinh vien: SV0002
Ho va ten: Pham Hong Phuc
Diem thi: 9
Nhap thong tin cho sinh vien thu 3
Ma so sinh vien: SV0046
```

```
Ho va ten: Le Minh Hoa
Diem thi: 10
Thong tin ve cac sinh vien
Sinh vien thu 1: SV0032    Nguyen Thanh Binh   8.5
Sinh vien thu 2: SV0002    Pham Hong Phuc      9.0
Sinh vien thu 3: SV0046    Le Minh Hoa         10.0
```

Chương trình có thể mở rộng thêm với các chức năng như tìm kiếm, sắp xếp thông tin theo tiêu chí nào đó như theo thứ tự điểm giảm dần chẳng hạn.

7.5. Con trỏ cấu trúc và địa chỉ cấu trúc

Một con trỏ trỏ đến cấu trúc hay biến con trỏ có kiểu cấu trúc (struct) chỉ đơn thuần là con trỏ đó trỏ đến địa chỉ của cấu trúc đó. Lưu ý con trỏ có kiểu struct không thể tự biến đổi biến con trỏ đó thành struct được.

Cú pháp:

```
struct <structure_tag_name >   /* Khai báo cấu trúc */
{
  <data_type member_name_1>;
  <data_type member_name_2>;
    .
    .
  <data_type member_name_n>;
} *ptr;
```

Hoặc

```
struct <structure_tag_name >   /* Khai báo cấu trúc */
{
  <data_type member_name_1>;
  <data_type member_name_2>;
    .
    .
  <data_type member_name_n>;
};
struct <structure_tag_name > *ptr;
```

Con trỏ ptr có thể được gán cho một con trỏ khác có cùng kiểu struct.

Để truy vấn đến các thành phần của cấu trúc. Chúng ta sử dụng cú pháp sau :

Cú pháp

```
(*ptr).member_name;
```

Hoặc

```
ptr-> member_name;
```

Để thiết lập cho các thành phần của struct thông qua biến con trỏ, chúng ta sử dụng cú pháp sau :

Cú pháp

*(*ptr).member_name_x = constant;*

Hoặc

ptr-> member_name_x = constant;

Ví dụ minh họa

```
struct test1  /** Khai báo struct "test" */
{
  char a;
  int i;
  float f;
};
struct test1 pt; /* Khai báo biến con trỏ có kiểu cấu truct "test"*/
pt->a='K';   /* Thiết lập giá trị cho char a */
pt->i=15;    /* Thiết lập giá trị cho  int i */
pt->f=27.89; /* Thiết lập giá trị cho float f */
```

7.6. Hàm trên các cấu trúc

Struct có thể làm tham đối để truyền cho hàm, giống như các tham đối có kiểu dữ liệu của hàm. Khi một struct đóng vai trò là tham đối truyền vào của hàm thì mỗi thành phần trực thuộc struct sẽ được sao chép thông tin. Thực tế mỗi thành phần của struct được truyền giá trị vào cho hàm bằng cơ chế truyền tham đối thông qua giá trị. Khi thành phần của struct là array thì sẽ thực hiện bằng cơ chế copy mảng để truyền giá trị.

Với các struct có kích cỡ lớn thì việc sử dụng truyền tham đối của hàm thông qua con trỏ sẽ tỏ ra hiệu quả hơn.

Cú pháp khai báo tham đối của hàm với struct như sau :

struct structure_tag function_name(struct structure_tag structure_variable);

Trong đó

structure_tag : Tên struct

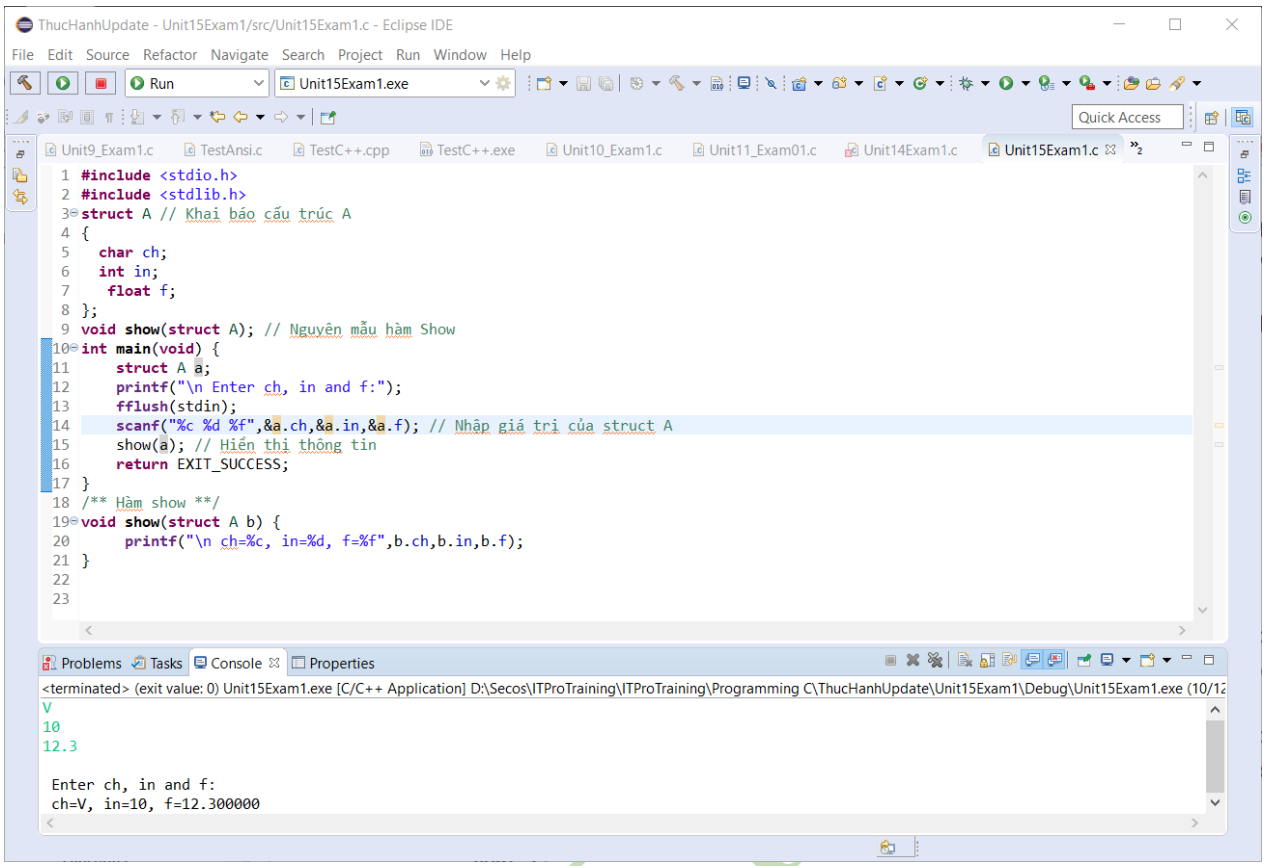
function_name: Tên hàm

structure_variable: Tên tham đối truyền vào cho hàm (Tên tham số hình thức).

Ví dụ:

```
struct A // Khai báo cấu trúc A
{
    char ch;
    int in;
    float f;
};
```

void show(struct A b); // Khai báo hàm show thực hiện vai trò hiển thị thông tin của struct A với b tham số hình thức truyền vào của A;



Hình số 9: Hàm có tham đối có kiểu struct

Trong ví dụ trên, hàm show(struct A b) là hàm có sử dụng b làm tham đối truyền vào.

Với tham đối là con trỏ, thay vì chúng ta sử dụng hàm :

scanf("%c %d %f",&a.ch,&a.in,&a.f);

Để nhập giá trị cho biến struct A a; thì chúng ta viết hàm với tham đối có kiểu con trỏ như sau :

```
void read(struct A *p){
printf("\n Enter ch, in and f:");  /* Nhập thông tin các thành phần struct  */
      fflush(stdin); /* Xóa các dòng dữ liệu khi nhập thông tin */
scanf("%c %d %f",&p->ch,&p->in,&p->f); /* Nhập giá trị */
}
```

Hàm read có tham số p truyền vào có kiểu con trỏ. Việc sử dụng con trỏ sẽ làm tăng tính hiệu quả khi lập trình.

7.7. Cấp phát bộ nhớ động

Tại sao cần cấp phát bộ nhớ động?

Như chúng ta đã biết, mảng là một tập hợp của các phần tử nằm liên tiếp nhau trên bộ nhớ và có cùng kiểu dữ liệu. Khi khai báo mảng, bạn phải chỉ định rõ kích thước tối đa (số lượng

phần tử tối đa). Và sau khi khai báo, bạn không thể thay đổi kích thước của mảng – **Cấp phát tĩnh**.

Đôi khi kích thước của mảng bạn khai báo có thể không đủ sài. Để giải quyết vấn đề này, bạn có thể cấp phát thêm bộ nhớ theo cách thủ công trong thời gian chạy chương trình. Đó cũng chính là khái niệm **cấp phát động trong C**.

Bảng dưới đây so sánh giúp bạn sự khác biệt giữa cấp phát bộ nhớ động và tĩnh.

Cấp phát bộ nhớ tĩnh	**Cấp phát bộ nhớ động**
Bộ nhớ được cấp phát trước khi chạy chương trình (trong quá trình biên dịch)	Bộ nhớ được cấp phát trong quá trình chạy chương trình.
Không thể cấp phát hay phân bổ lại bộ nhớ trong khi chạy chương trình	Cho phép quản lý, phân bổ hay giải phóng bộ nhớ trong khi chạy chương trình
Vùng nhớ được cấp phát và tồn tại cho đến khi kết thúc chương trình	Chỉ cấp phát vùng nhớ khi cần sử dụng tới
Chương trình chạy nhanh hơn so với cấp phát động	Chương trình chạy chậm hơn so với cấp phát tĩnh
Tốn nhiều không gian bộ nhớ hơn	Tiết kiệm được không gian bộ nhớ sử dụng

Ưu điểm chính của việc sử dụng cấp phát động là giúp ta **tiết kiệm được không gian bộ nhớ** mà chương trình sử dụng. Bởi vì chúng ta sẽ chỉ cấp phát khi cần dùng và có thể giải phóng vùng nhớ đó ngay sau khi sử dụng xong.

Nhược điểm chính của cấp phát động là bạn phải **tự quản lý vùng nhớ** mà bạn cấp phát. Nếu bạn cứ cấp phát mà quên giải phóng bộ nhớ thì chương trình của bạn sẽ tiêu thụ hết tài nguyên của máy tính dẫn đến tình trạng tràn bộ nhớ (memory leak).

Cấp phát bộ nhớ động trong C

Để cấp phát vùng nhớ động cho biến con trỏ trong ngôn ngữ C, bạn có thể sử dụng hàm malloc() hoặc hàm calloc(). Sử dụng hàm free() để giải phóng bộ nhớ đã cấp phát khi không cần sử dụng, sử dụng realloc() để thay đổi (phân bổ lại) kích thước bộ nhớ đã cấp phát trong khi chạy chương trình.

Sử dụng hàm malloc()

Từ **malloc** là đại diện cho cụm từ memory allocation (cấp phát bộ nhớ).

Hàm malloc() thực hiện cấp phát bộ nhớ bằng cách chỉ định **số byte** cần cấp phát. Hàm này trả về con trỏ kiểu void cho phép chúng ta có thể ép kiểu về bất cứ kiểu dữ liệu nào.

Cú pháp của hàm malloc()**:**

```
ptr = (castType*) malloc(size);
```

Ví dụ:

```
ptr = (int*) malloc(100 * sizeof(int));
```

Ví dụ trên thực hiện cấp phát cho việc lưu trữ 100 số nguyên. Giả sử sizeof int là 4, khi đó lệnh dưới đây thực hiện cấp phát 400 bytes. Khi đó, con trỏ ptr sẽ có giá trị là địa chỉ của **byte dữ liệu đầu tiên** trong khối bộ nhớ vừa cấp phát.

Trong trường hợp không thể cấp phát bộ nhớ, nó sẽ trả về một con trỏ NULL.

Sử dụng hàm calloc()

Từ calloc đại diện cho cụm từ contiguous allocation (cấp phát liên tục).

Hàm malloc() khi cấp phát bộ nhớ thì vùng nhớ cấp phát đó không được khởi tạo giá trị ban đầu. Trong khi đó, hàm calloc() thực hiện cấp phát bộ nhớ và khởi tạo tất cả các ô nhớ có giá trị bằng 0.

Hàm calloc() nhận vào 2 tham số là **số ô nhớ** muốn khởi tạo và **kích thước của 1 ô nhớ**.

Cú pháp của hàm calloc():

```
ptr=(castType*)calloc(n,size);
```

Ví dụ:

```
ptr=(int*) calloc(100,sizeof(int));
```

Trong ví dụ trên, hàm calloc() thực hiện cấp phát 100 ô nhớ liên tiếp và mỗi ô nhớ có kích thước là số byte của kiểu int. Hàm này cũng trả về con trỏ chứa giá trị là địa chỉ của byte đầu tiên trong khối bộ nhớ vừa cấp phát.

BÀI 8

QUẢN LÝ MÀN HÌNH VÀ CỬA SỔ

Người học sau khi học xong bài 8 sẽ có các khái niệm cơ bản về các vấn đề sau:

- Chọn kiểu màn hình văn bản
- Đặt màu nền và chữ
- Xây dựng và sử dụng cửa sổ

8.1. Chọn kiểu màn hình văn bản

Hàm **textmode:** cho phép định dạng màu sắc của màn hình.

Cú pháp:

```
void textmode(int mode)
```

Ký hiệu	Giá trị	Mode video
Lastmode	-1	Mode trước đó
BW40	0	Đen trắng 40 cột
C40	1	16 màu, 40 cột
BW80	2	Đen trắng 80 cột
C80	3	16 màu, 80 cột
MONO	7	Đơn sắc, 80 cột

Ví dụ 1:

- textmode(C40) sẽ cho màn hình gồm:
  - 25 hàng
  - 40 cột
  - 16 màu

Ví dụ 2:

- textmode(C80) sẽ cho màn hình gồm:
  - 25 hàng
  - 80 cột
  - 16 màu

Chú ý: Màn hình được đánh tọa độ, góc trên cùng bên trái sẽ có tọa độ là 1,1. Góc dưới cùng bên phải sẽ có tọa độ là 40,25 hoặc 80,25

8.2 Màu nền và màu chữ

void textbackground(int color); void textcolor(int color);

color là 1 biến integer có giá trị từ 0 đến 15 với ý nghĩa thể hiện ở các bảng bên.
Tùy theo từng giá trị mà giá trị của màu chữ hoặc màu nền được gán.
Với giá trị >=8 thì chỉ được gán cho màu chữ.
Còn giá trị <8 thì có thể gán cho đồng thời cả màu chữ và màu nền.

Ký hiệu	Giá trị	Màu chữ / màu nền
Black	0	Cả hai
Blue	1	Cả hai
Green	2	Cả hai
Cyan	3	Cả hai
Red	4	Cả hai
Magenta	5	Cả hai
Brown	6	Cả hai
BlackƠ]	7	Cả hai
Blue	8	Chữ
Green	9	Chữ
Cyan	10	Chữ
Red	11	Chữ
Magenta	12	Chữ
Brown	13	Chữ
Red	14	Chữ
Magenta	15	Chữ

8.3 Xây dựng cửa sổ và sử dụng cửa sổ

Để xây dựng và sử dụng cửa sổ, ta sử dụng hàm window với cú pháp như sau:

void window(int xt,int yt, int xd, int yd);

- window: tên hàm cửa sổ
- xt, yt: tọa độ góc trên cùng bên trái của cửa sổ
- xd,yd: tọa độ góc dưới cùng bên phải của cửa sổ

Chú ý:

- xd>=xt
- yd>=yt
- Giá trị của 4 tham số này phải nằm trong giới hạn hiển thị của cửa sổ.

Ví dụ:
textbackground(RED) và windows(5,5,35,20)

- Tọa độ trên bên trái là 5,5
- Tọa độ dưới bên phải là 35,20
- Nền cửa sổ có thể là màu đỏ nếu trước đó chưa hiển thị gì. Muốn chắc chắn là mầu đỏ, ta cần phải xóa sạch màn hình trước khi điều chỉnh màu nền thông qua lệnh:
  void clrscr(void);

8.4 Các hàm cprintf và cscanf

- Hàm cprintf, cscanf dùng để nhập dữ liệu từ bàn phím và đưa kết quả ra màn hình.
- Màu của ký tự sẽ lấy giá trị được thiết lập gần nhất
- Khi dùng hàm cscanf, nếu bấm sai thì không dùng được các phím chức năng
- Hàm printf có phạm vi làm việc là toàn bộ cửa sổ màn hình, do đó có thể xảy ra hiện tượng tràn dữ liệu hiển thị. Trong khi đó, hàm cprintf chỉ có phạm vi hiển thị trong cửa sổ khai báo. Nếu thông tin quá dài thì có thể bị mất thông tin hiển thị

8.6 Các hàm khác

- **Hàm xóa màn hình**
  Cú pháp:
  void clrscr(void);
  Chức năng:
  - Xóa cửa sổ hiện tại
  - Đưa con trỏ lên góc trên cùng bên trái
    Màu sẽ được xác định bởi hàm textbackground
- **Hàm xóa dòng**
  Cú pháp:
  void clreol(void);
  Chức năng:
  - Xóa mọi ký tự đứng sau con trỏ đến cuối dòng
  - Không thay đổi vị trí con trỏ
- **Hàm xóa 1 dòng trong cửa sổ**
  Cú pháp:
  void delline(void);
  Chức năng:
  - Xóa dòng cửa sổ đang chứa con trỏ
  - Không thay đổi vị trí con trỏ
- **Hàm di chuyển con trỏ**
  Cú pháp:
  void gotoxy(int x, int y);
  Chức năng:
  - Chạy đến vị trí x,y trong cửa sổ hiện tại
  - Thay đổi vị trí con trỏ
- **Hàm lấy tọa độ ngang của con trỏ**
  Cú pháp:
  int wherex(void);
  Chức năng:
  - Cho vị trí ngang của con trỏ trong cửa sổ hiện hành
  - Giá trị trả về là kiểu int
  - Không thay đổi vị trí con trỏ
- **Hàm lấy tọa độ dọc của con trỏ**
  Cú pháp:
  int wherey(void);
  Chức năng:
  - Cho vị trí dọc của con trỏ trong cửa sổ hiện hành
  - Giá trị trả về là kiểu int
- **Hàm lấy thông tin hiển thị văn bản**
  Cú pháp:

void gettextinfo(struct text_info *r);

Chức năng:

- r: con trỏ tới địa chỉ của text_info
- Gửi các thông tin liên quan đến kiểu hiển thị màn hình văn bản đang sử dụng vào các thành phần của biến cấu trúc.

BÀI 9

ĐỒ HỌA

9.1. Khởi động đồ hoạ

Mục đích của việc khởi động hệ thống đồ hoạ là xác định thiết bị đồ hoạ (màn hình) và mode đồ hoạ sẽ sử dụng trong chương trình. Để làm công việc này, ta có hàm sau :

```
void initgraph(int *graphdriver,int graphmode,char *driverpath);
```

Trong đó :

driverpath là xâu ký tự chỉ đường dẫn đến thư mục chứa các tập tin điều khiển đồ hoạ.

graphdriver cho biết màn hình đồ hoạ sử dụng trong chương trình.

graphmode cho biết mode đồ hoạ sử dụng trong chương trình.

Bảng dưới đây cho các giá trị khả dĩ của graphdriver và graphmode :

Các giá trị khả dĩ của graphdriver và graphmode

graphdriver**DETECT (0)**	graphmode	Độ phân giải
CGA (1)	CGAC0 (0)CGAC1 (1)CGAC2 (2)CGAC3 (3)CGAHi (4)	320x200320x200320x200320x200640x200
MCGA (2)	MCGA0 (0)MCGA1 (1)MCGA2 (2)MCGA3 (3)MCGAMed (4)MCGAHi (5)	320x200320x200320x200320x200640x200640x480
EGA (3)	EGAL0 (0)EGAHi (1)	640x200640x350
EGA64 (4)	EGA64LO (0)EGA64Hi (1)	640x200640x350

EGAMONO (5)	EGAMONOHi (0)	640x350
VGA (9)	VGALO (0)VGAMED (1)VGAHI (2)	640x200640x350640x480
HERCMONO (7)	HERCMONOHI	720x348
ATT400 (8)	ATT400C0 (0)ATT400C1 (1)ATT400C2 (2)ATT400C3 (3)ATT400MED (4)ATT400HI (5)	320x200320x200320x200320x200640x400640x400
PC3270 (10)	PC3270HI (0)	720x350
IBM8514 (6)	PC3270LO (0)PC3270HI (1)	640x480 256 mầu1024x768 256 mầu

- Bảng trên cho ta các hằng và giá trị của chúng mà các biến graphdtriver và graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0 vv...Khi lập trình ta có thể thay thế vào vị trí tương ứng của chúng trong hàm tên hằng hoặc giá trị của hằng đó.

Ví dụ :

Giả sử máy tính có màn hình VGA, các tập tin đồ hoạ chứa trong thư mục C:\TC \BGI, khi đó ta khởi động hệ thống đồ hoạ như sau :

```
#include "graphics.h"
   main()
   {
   int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
   initgraph(&mh,&mode,"C:\\TC\\BGI");
   /* Vì kí tự \ trong C là kí tự đặc biệt nên ta phải gấp đôi nó */
```

```
}
```

Bảng trên còn cho thấy độ phân giải còn phụ thuộc cả vào màn hình và mode. Ví dụ như trong màn hình EGA nếu dùng EGALo thì độ phân giải là 640x200 (Hàm getmaxx() cho giá trị cực đại của số điểm theo chiều ngang của màn hình. Với màn hình EGA trên : 639, Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình. Với màn hình EGA trên : 199).

Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver bằng DETECT hay giá trị 0. Khi đó, kết quả của initgraph sẽ là :

Kiểu màn hình đang sử dụng được phát hiện, giá trị của nó được gán cho biến graphdriver.

Mode đồ hoạ ở độ phân giải cao nhất ứng với màn hành đang sử dụng cũng được phát hiện và trị số của nó được gán cho biến graphmode.

Như vậy dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ hoạ với màn hình hiện có theo mode có độ phân giải cao nhất mà còn giúp ta xác định kiểu màn hình đang sử dụng.

Ví dụ :

Chương trình dưới đây xác định kiểu màn hình đang sử dụng :

```
#include "graphics.h"

   #include "stdio.h"
   main()
   {
   int mh=0, mode;
   initgraph(&mh,&mode,"C:\\TC\\BGI");
   printf("\n Gia tri so cua man hinh la : %d",mh);
   printf("\n Gia tri so mode do hoa la : %d",mode);
   closegraph();
   }
```

- Nếu chuỗi dùng để xác định driverpath là chuỗi rỗng thì chương trình dịch sẽ tìm kiếm các file điều khiển đồ hoạ trên thư mục chủ (Thư mục hiện thời).

9.2 Các hàm đồ hoạ

9.2.1 Mẫu và màu

- **Đặt màu nền**

Để đặt màu cho nền ta dùng thủ tục sau :

```
void setbkcolor(int màu);
```

- **Đặt màu đường vẽ**

Để đặt màu vẽ đường ta dùng thủ tục sau :

```
void setcolor(int màu);
```

- **Đặt mẫu (kiểu) tô và màu tô**

Để đặt mẫu (kiểu) tô và màu tô ta dùng thủ tục sau :

```
void setfillstyle(int mẫu, int màu);
```

Trong cả ba trường hợp **màu** xác định mã của màu.

Các giá trị khả dĩ của **màu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của màu

Bảng các giá trị khả dĩ của màu

Tên hằng	Giá trị số	Màu hiển thị
BLACK	0	Đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIGHTGRAY	7	Xám nhạt
DARKGRAY	8	Xám đậm
LIGHTBLUE	9	Xanh xa trời nhạt

LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	16	Trắng

Các giá trị khả dĩ của **mẫu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của mẫu

Bảng các giá trị khả dĩ của mẫu

Tên hằng	Giá trị số	Kiểu mẫu tô
EMPTY_FILL	0	Tô bằng mầu nền
SOLID_FILL	1	Tô bằng đường liền nét
LINE_FILL	2	Tô bằng đường --------
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa

CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

Chọn giải màu

Để thay đổi giải màu đã được định nghĩa trong bảng trên, ta sử dụng hàm :

```
void setpalete(int số_thứ_tự_màu, int màu );
```

Ví dụ :

Câu lệnh :

```
setpalete(0,lightcyan);
```

biến màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng.

- **Lấy giải màu hiện thời**

+ Hàm getcolor trả về mầu đã xác định bằng thủ tục setcolor ngay trước nó.

+ Hàm getbkcolor trả về mầu đã xác định bằng hàm setbkcolor ngay trước nó.

9.2.2. Vẽ và tô màu

Có thể chia các đường và hình thành bốn nhóm chính :

- Cung tròn và hình tròn.
- Đường gấp khúc và đa giác.
- Đường thẳng.
- Hình chữ nhật.

Cung tròn và đường tròn

Nhóm này bao gồm : Cung tròn, đường tròn, cung elip và hình quạt.

- **Cung tròn**

Để vẽ một cung tròn ta dùng hàm :

```
void arc(int x, int y, int gd, int gc, int r);
```

Trong đó :

(x,y) là toạ độ tâm cung tròn.

gd là góc đầu cung tròn(0 đến 360 độ).

gc là góc cuối cung tròn (gd đến 360 độ).

r là bán kính cung tròn .

Ví dụ :

Vẽ một cung tròn có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán kính 30.

```
arc(100,50,0,180,30);
```

- **Đường tròn**

Để vẽ đường tròn ta dùng hàm :

```
void circle(int x, int y, int r);
```

Trong đó :

(x,y) là toạ độ tâm cung tròn.

r là bán kính đường tròn.

Ví dụ :

Vẽ một đường tròn có tâm tại (100,50) và bán kính 30.

```
circle(100,50,30);
```

- **Cung elip**

Để vẽ một cung elip ta dùng hàm :

```
void ellipse(int x, int y, int gd, int gc, int xr, int yr);
```

Trong đó :

(x,y) là toạ độ tâm cung elip.

gd là góc đầu cung tròn(0 đến 360 độ).

gc là góc cuối cung tròn (gd đến 360 độ).

xr là bán trục nằm ngang.

yr là bán trục thẳng đứng.

Ví dụ **:**

Vẽ một cung elip có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán trục ngang 30, bán trục đứng là 20.

```
ellipse(100,50,0,180,30,20);
```

- **Hình quạt**

Để vẽ và tô màu một hình quạt ta dùng hàm :

```
void pieslice(int x, int y, int gd, int gc, int r);
```

Trong đó :

(x,y) là toạ độ tâm hình quạt.

gd là góc đầu hình quạt (0 đến 360 độ).

gc là góc cuối hình quạt (gd đến 360 độ).

r là bán kính hình quạt .

Ví dụ :

Chương trình dưới đây sẽ vẽ một cung tròn ở góc phần tư thứ nhất, một cung elip ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
# include "graphics.h"
   #include "stdio.h"
   #include "conio.h"
   main()
   {
   int md=0,mode;
   initgraph(&md,&mode,"C:\\TC\\BGI");
   setbkcolor(BLUE);
   setcolor(YELLOW);
   setfillstyle(SOLID_FILL,RED);;
   arc(160,50,0,90,45);
   circle(160,150,45);
   pieslice(480,150,90,360,45);
   getch();
   closegraph();
   }
```

9.2.3. Vẽ đường gấp khúc và đa giác

- **Vẽ đường gấp khúc**

Muốn vẽ đường gấp khúc đi qua n điểm : (x1,y1), (x2,y2), ..., (xn,yn) thì trước hết ta phải gán các toạ độ (xi,yi) cho một mảng a kiểu int nào đó theo nguyên tắc sau :

Toạ độ x1 gán cho a[0]

Toạ độ y1 gán cho a[1]

Toạ độ x2 gán cho a[2]

Toạ độ y2 gán cho a[3]

....

Toạ độ xn gán cho a[2n-2]

Toạ độ yn gán cho a[2n-1]

Sau đó gọi hàm :

drawpoly(n,a);

Nếu điểm cuối cùng (xn,yn) trùng với điểm đầu (x1,y1) thì ta nhận được một đường gấp khúc khép kín.

- **Tô màu đa giác**

Giả sử ta có a là mảng đã đề cập đến trong mục trên, khi đó ta gọi hàm :

```
fillpoly(n,a);
```

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm (x1,y1), (x2,y2), ..., (xn,yn)

Ví dụ :

Vẽ một đường gấp khúc và hai đường tam giác.

```
#include "graphics.h"
   #include "stdio.h"
   #include "conio.h"
   int poly1[]={5,200,190,5,100,300};
   int poly2[]={205,200,390,5,300,300};
   int poly3[]={405,200,590,5,500,300,405,200};
   main()
   {
   int md=0,mode;
   initgraph(&md,&mode,"C:\\TC\\BGI");
   setbkcolor(CYAN);
   setcolor(YELLOW);
   setfillstyle(SOLID_FILL,MAGENTA);
   drawpoly(3,poly1);
   fillpoly(3,poly2);
   fillpoly(4,poly3);
   getch();
   closegraph();
   }
```

- **Vẽ đường thẳng**

Để vẽ đường thẳng nối hai điểm bất kỳ có toạ độ (x1,y1) và (x2,y2) ta sử dụng hàm sau :

```
void line(int x1, int y1, int x2, int y2);
```

Con chạy đồ hoạ giữ nguyên vị trí.

Để vẽ đường thẳng nối từ điểm con chạy đồ hoạ đến một điểm bất có toạ độ (x,y) ta sử dụng hàm sau :

```
void lineto(int x, int y);
```

Con chạy sẽ chuyển đến vị trí (x,y).

Để vẽ một đường thẳng từ ví trí con chạy hiện tại (giả sử là điểm x,y) đến điểm có toạ độ (x+dx,y+dy) ta sử dụng hàm sau :

```
void linerel(int dx, int dy);
```

Con chạy sẽ chuyển đến vị trí (x+dx,y+dy).

- **Di chuyển con chạy đồ hoạ**

Để di chuyển con chạy đến vị trí (x,y), ta sử dụng hàm sau :

```
void moveto(int x, int y);
```

- **Chọn kiểu đường**

```
Hàm  void setlinestyle(int kiểu_đường, int mẫu, int độ_dày);
```

tác động đến nét vẽ của các thủ tục vẽ đường line, lineto,linerel , circle, rectangle (hàm vẽ hình chữ nhật, ta sẽ học trong phần vẽ miền ở dưới).

Hàm này sẽ cho phép ta xác định ba yếu tố khi vẽ đường thẳng, đó là : Kiểu đường, bề dày và mẫu tự tạo.

Dạng đường do tham số **kiểu_đường** xác định. Bảng dưới đây cho các giá trị khả dĩ của **kiểu_đường :**

Các giá trị khả dĩ của kiểu_đường

Tên hằng	Giá trị số	Kiểu đường
SOLID_LINE	0	Nét liền
DOTTED_LINE	1	Nét chấm
CENTER_LINE	2	Nét chấm gạch

DASHED_LINE	3	Nét gạch
USERBIT_LINE	4	Mẫu tự tạo

Bề dày của đường vẽ do tham số **độ_dày** xác định,. bảng dưới đây cho các giá trị khả dĩ của **độ_dày** :

các giá trị khả dĩ của độ_dày :

Tên hằng	Giá trị số	Bề dày
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Mẫu tự tạo : Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số **mẫu.**

Ví dụ ta xét đoạn chương trình :

```
int pattern = 0x1010;
   setlinestile(USERBIT_LINE,pattern,NORM_WIDTH);
   line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010, trong hệ 2 là :

0001 0000 0001 0000

Bit 1 sẽ cho điểm sáng, bit 0 sẽ làm tắt điểm ảnh.

Ví dụ :

Chương trình vẽ một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau :

```
(20,20),(620,20),(620,180),(20,180) và (320,100)
   #include "graphics.h"
   #include "stdio.h"
   #include "conio.h"
   main()
   {
   int mh=0, mode;
   initgraph(&mh,&mode,"C:\\TC\\BGI");
```

```
setbkcolor(BLUE);
setcolor(YELLOW);
setlinestyle(SOLID-LINE,0,THICK_WIDTH);
moveto(320,100); /* con chạy ở vị trí ( 320,100 ) */
line(20,20,620,20); /* con chạy vẫn ở vị trí ( 320,100 ) */
linerel(-300,80);
lineto(620,180);
lineto(620,20);
getch();
closegraph();
}
```

9.2.4. Vẽ điểm, miền

- **Vẽ điểm**

Hàm :

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo mầu xác định bởi **color.**

Hàm

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y).

Nếu điểm này chưa được tô màu bởi các hàm vẽ hoặc hàm putpixel (mà chỉ mới được tạo màu nền bởi setbkcolor) thì hàm cho giá trị 0.

- **Tô miền**

Để tô màu cho một miền nào đó trên màn hình, ta dùng hàm sau :

```
void floodfill(int x, int y, int border);
```

ở đây :

(x,y) là toạ độ của một điểm nào đó gọi là điểm gieo.

Tham số border chứa mã của màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x,y,border và trạng thái màn hình.

+ Khi trên màn hình có một đường cong khép kín hoặc đường gấp khúc khép kín mà mã màu của nó bằng giá trị của border thì :

- Nếu điểm gieo (x,y) nằm trong miền này thì miền giới hạn phía trong đường sẽ được tô màu.
- Nếu điểm gieo (x,y) nằm ngoài miền này thì miền phía ngoài đường sẽ được tô màu.

+ Trong trường hợp khi trên màn hình không có đường cong nào như trên thì cả màn hình sẽ được tô màu.

Ví dụ :

Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Toạ độ (x,y) của điểm gieo được nạp từ bàn phím. Tuỳ thuộc giá trị cụ thể của x,y chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include "graphics.h"
   #include "stdio.h"
   main()
   {
   int mh=mode=0, x, y;
   printf("\nVao toa do x,y:");
   scanf("%d%d",&x,&y);
   initgraph(&mh,&mode,"");
   if (graphresult != grOk) exit(1);
   setbkcolor(BLUE);
   setcolor(RED);
   setfillstyle(11,YELLOW);
   circle(320,100,50);
   moveto(1,150);
   floodfill(x,y,RED);
   closegraph();
   }
```

9.2.5. Hình chữ nhật

- Hàm

```
void rectangle(int x1, int y1, int x2, int y2);
```

sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm

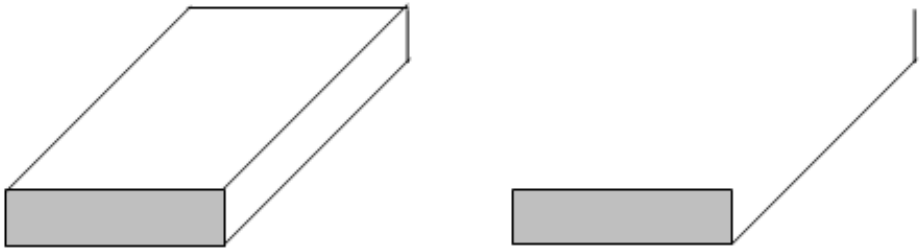
```
void bar(int x1, int y1, int x2, int y2);
```

sẽ vẽ và tô màu một hình chữ nhật. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm

```
void bar3d(int x1, int y1, int x2, int y2, int depth, int top);
```

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các toạ độ (x1,y1), (x2,y2). Hình chữ nhật này được tô màu thông qua hàm setfillstyle . Tham số **depth** xác định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số **top** có thể nhận các giá trị 1 hay 0 và khối 3 chiều tương ứng sẽ có nắp hoặc không.



Khối 3 chiều

Ví dụ :

Chương trình dưới đây tạo nên một hình chữ nhật, một khối hình chữ nhật và một hình hộp có nắp :

```
#include "graphics.h"
   main()
   {
   int mh=mode=0;
   initgraph(&mh,&mode,"");
   if (graphresult != grOk) exit(1);
   setbkcolor(GREEN);
   setcolor(RED);
   setfillstyle(CLOSE_DOT_FILL,YELLOW);
   rectangle(5,5,300,160);
```

```
bar(3,175,300,340);
bar3d(320,100,500,340,100,1);

closegraph();
}
```

9.3. Cửa sổ (Viewport)

- **Thiết lập viewport**

Viewport là một vùng chữ nhật trên màn hình đồ hoạ. Để thiết lập viewport ta dùng hàm :

```
void setviewport(int x1, int y1, int x2, int y2, int clip);
```

trong đó (x1,y1) là toạ độ góc trên bên trái, (x2,y2) là toạ độ góc dưới bên phải. Bốn giá trị này vì thế phải thoả mãn :

$0 \le x1 \le x2$

$0 \le y1 \le y2$

Tham số clip có thể nhận một trong hai giá trị :

clip=1 không cho phép vẽ ra ngoài viewport.

clip=0 cho phép vẽ ra ngoài viewport.

Ví dụ :

```
setviewport(100,50,200,150,1);
```

Lập nên một vùng viewport hình chữ nhật có toạ độ góc trái cao là (100,50) và toạ độ góc phải thấp là (200,150) (là toạ độ trước khi đặt viewport).

Sau khi lập viewport, ta có hệ toạ độ mới mà góc trên bên trái sẽ có toạ độ (0,0).

- **Nhận diện viewport hiện hành**

Để nhận viewport hiện thời ta dùng hàm :

```
void getviewsetting(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau :

```
struct viewporttype
  {
  int left,top,right,bottom;
  int clip;
  };
```

- **Xóa viewport**

Sử dụng hàm :

```
void clearviewport(void);
```

- **Xoá màn hình, đưa con chạy về tạo độ (0,0) của màn hình :**

Sử dụng hàm :

```
void cleardevice(void);
```

- **Toạ độ âm dương**

Nhờ sử dụng viewport có thể viết các chương trình đồ hoạ theo toạ độ âm dương. Muốn vậy ta thiết lập viewport và cho clip bằng 0 để có thể vẽ ra ngoài giới hạn của viewport.

Sau đây là đoạn chương trình thực hiện công việc trên :

```
int xc,yc;
   xc=getmaxx()/2;
   yc=getmaxy()/2;
   setviewport(xc,yc,getmaxx(),getmaxy(),0);
```

Như thế, màn hình sẽ được chia làm bốn phần với toạ độ âm dương như sau :

Phần tư trái trên : x âm, y âm.

x : từ -getmaxx()/2 đến 0.

y : từ -getmaxy()/2 đến 0.

Phần tư trái dưới : x âm, y dương.

x : từ -getmaxx()/2 đến 0.

y : từ 0 đến getmaxy()/2.

Phần tư phải trên : x dương, y âm.

x : từ 0 đến getmaxx()/2.

y : từ -getmaxy()/2 đến 0.

Phần tư phải dưới : x dương, y dương.

x : từ 0 đến getmaxx()/2.

y : từ 0 đến getmaxy()/2.

Ví dụ :

Chương trình vẽ đồ thị hàm sin x trong hệ trục toạ độ âm dương. Hoành độ x lấy các giá trị từ -4π đến 4π. Trong chương trình có sử dụng hai hàm mới là settextjustify và outtextxy ta sẽ đề cập ngay trong phần sau.

```
#include "graphics.h"
   #include "conio.h"
   #include "math.h"
   #define TYLEX 20
   #define TYLEY 60
   main()
   {
   int mh=mode=DETECT;
   int x,y,i;
   initgraph(mh,mode,"");
   if (graphresult!=grOK ) exit(1);
   setviewport(getmaxx()/2,getmaxy()/2,getmaxx(),getmaxy(),0);
   setbkcolor(BLUE);
   setcolor(YELLOW);
   line(-getmaxx()/2,0,getmaxx()/2,0);
   line(0,-getmaxy()/2,0,getmaxy()/2,0);
   settextjustify(1,1);
   setcolor(WHITE);
   outtextxy(0,0,"(0,0)");
   for (i=-400;i<=400;++i)
   {
   x=floor(2*M_PI*i*TYLEX/200);
   y=floor(sin(2*M_PI*i/200)*TYLEY);
   putpixel(x,y,WHITE);
   }
   getch();
   closegraph();
   }
```

9.4 Xử lý văn bản trên màn hình đồ hoạ

- Hiển thị văn bản trên màn hình đồ hoạ

Hàm :

```
void outtext(char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí con trỏ đồ hoạ hiện thời.

Hàm :

```
 void outtextxy(int x, int y,char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí (x,y).

Ví dụ :

Hai cách viết dưới đây :

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
   outtext(" Say HELLO");
```

cho cùng kết quả.

- Sử dụng các Fonts chữ

Các Fonts chữ nằm trong các tập tin *.CHR trên đĩa. Các Fonts này cho các kích thước và kiểu chữ khác nhau, chúng sẽ được hiển thị lên màn hình bằng các hàm outtext và outtextxy. Để chọn và nạp Fonts ta dùng hàm :

```
void settextstyle(int font, int direction, int charsize);
```

Tham số font để chọn kiểu chữ và nhận một trong các hằng sau :

DEFAULT_FONT=0

TRIPLEX_FONT=1

SMALL_FONT=2

SANS_SERIF_FONT=3

GOTHIC_FONT=4

Tham số derection để chọn hướng chữ và nhận một trong các hằng sau :

HORIZ_DIR=0 văn bản hiển thị theo hướng nằm ngang từ trái qua phải.

VERT_DIR=1 văn bản hiển thị theo hướng thẳng đứng từ dưới lên trên.

Tham số charsize là hệ số phóng to của ký tự và có giá trị trong khoảng từ 1 đến 10.

Khi charsize=1, font hiển thị trong hình chữ nhật 8*8 pixel.

Khi charsize=2 font hiển thị trong hình chữ nhật 16*16 pixel.

.....

Khi charsize=10, font hiển thị trong hình chữ nhật 80*80 pixel.

Các giá trị do settextstyle lập ra sẽ giữ nguyên tới khi gọi một settextstyle mới.

Ví dụ :

Các dòng lệnh :

```
settextstyle(3,VERT_DIR,2);
   outtextxy(30,30,"GODS TRUST YOU");
```

sẽ hiển thị tại vị trí (30,30) dòng chữ GODS TRUST YOU theo chiều từ dưới lên trên, font chữ chọn là SANS_SERIF_FONT và cỡ chữ là 2.

- **Đạt vị trí hiển thị của các xâu ký tự cho bởi outtext và outtextxy**

Hàm settextjustify cho phép chỉ định ra nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy và của outtextxy theo quan hệ với toạ độ (x,y);

Hàm này có dạng sau :

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau :

LEFT_TEXT=0 (Văn bản xuất hiện bên phải con chạy).

CENTER_TEXT (Chỉnh tâm văn bản theo vị trí con chạy).

RIGHT_TEXT (Văn bản xuất hiện bên trái con chạy).

Tham số vert có thể là một trong các hằng số sau :

BOTTOM_TEXT=0 (Văn bản xuất hiện phía trên con chạy).

CENTER_TEXT=1 (Chỉnh tâm văn bản theo vị trí con chạy).

TOP_TEXT=2 (Văn bản xuất hiện phía dưới con chạy).

Ví dụ :

```
settextjustify(1,1);
   outtextxy(100,100,"ABC");
```

sẽ cho dòng chữ ABC trong đó điểm (100,100) sẽ nằm dưới chữ B.

- **Bề rộng và chiều cao văn bản**

Chiều cao

Hàm :

```
textheight(char *s);
```

cho chiều cao (tính bằng pixel) của chuỗi do con trỏ s trỏ tới.

Ví dụ 1 :

Với font bit map và hệ số phóng đại là 1 thì textheight("A") ch giá trị là 8.

Ví dụ 2 :

```
#include "stdio.h"
   #include "graphics.h"
   main()
   {
   int mh=mode=DETECT, y,size;
   initgraph(mh,mode,"C:\\TC\\BGI");
   y=10;
   settextjustify(0,0);
   for (size=1;size<5;++size)
   {
   settextstyle(0,0,size);
   outtextxy(0,y,"SACRIFICE");
   y+=textheight("SACRIFICE")+10;
   }
   getch();
   closegraph();
   }
```

Bề rộng

Hàm :

```
textwidth(char *s);
```

cho bề rộng chuỗi (tính theo pixel) mà con trỏ s trỏ tới dựa trên chiều dài chuỗi, kích thước font chữ, hệ số phóng đại.