MỜI CÁC BẠN TÌM ĐỌC

1. GIÁO TRÌNH NGÔN NGỮ LẬP TRÌNH PASCAL
2. SÁNG TẠO TRONG THUẬT TOÁN VÀ LẬP TRÌNH (3 TẬP)
3. KỸ NGHỆ PHẦN MỀM
4. GIÁO TRÌNH TIN HỌC CƠ SỞ
5. GIÁO TRÌNH MẠNG MÁY TÍNH
6. GIÁO TRÌNH KỸ THUẬT VI XỬ LÝ (02 TẬP)
7. LÝ THUYẾT HỆ THỐNG VÀ ĐIỀU KHIỂN HỌC
8. NHẬP MÔN PHÂN TÍCH THÔNG TIN CÓ BẢO MẬT
9. GIÁO TRÌNH CƠ SỞ DỮ LIỆU
10. GIÁO TRÌNH CƠ SỞ DỮ LIỆU PHÂN TÁN
11. GIÁO TRÌNH CƠ SỞ DỮ LIỆU: LÝ THUYẾT VÀ THỰC HÀNH
12. BÀI TẬP CƠ SỞ DỮ LIỆU
13. CƠ SỞ DỮ LIỆU QUAN HỆ & ỨNG DỤNG
14. CƠ SỞ DỮ LIỆU QUAN HỆ VÀ CÔNG NGHỆ PHÂN TÍCH - THIẾT KẾ

Giá: ...đ

TS. NGUYỄN NGỌC CƯƠNG (Chủ biên)

Giáo trình Ngôn ngữ lập trình C/C++

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

TS. Nguyễn Ngọc Cương (Chủ biên)
ThS. Nguyễn Đình Nghĩa - ThS. Đỗ Quốc Huy - CN. Trần Nghi Phú - KS. Phạm Thành Công

Giáo trình
Ngôn ngữ lập trình C/C++

NHÀ XUẤT BẢN
THÔNG TIN VÀ TRUYỀN THÔNG

TS. Nguyễn Ngọc Cương (Chủ biên)
ThS. Nguyễn Đình Nghĩa - ThS. Đỗ Quốc Huy - CN. Trần Nghi Phú - KS. Phạm Thành Công

Giáo trình
Ngôn ngữ lập trình C/C++

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

Mã số: GD 14 HM 11

LỜI NÓI ĐẦU

Từ ngàn năm nay, con người liên lạc với nhau thông qua ngôn ngữ, tạo ra các mẫu từ ngữ và âm thanh. Ngôn ngữ lập trình cũng tương tự như vậy, đó là một tập từ ngữ và ký hiệu cho phép lập trình viên hoặc người dùng có thể tương tác với máy tính. Việc hiểu biết và nắm vững các ngôn ngữ lập trình là yêu cầu bắt buộc đối với các lập trình viên. Mặc dù đã có hàng trăm ngôn ngữ lập trình được sinh ra, song chỉ có một số ít là được sử dụng rộng rãi và được xem là một chuẩn công nghiệp. Các ngôn ngữ này đều có thể được sử dụng trên nhiều loại máy tính khác nhau. Hiện nay, ngôn ngữ lập trình C/C++ là ngôn ngữ bậc cao đang được sử dụng khá rộng rãi trong lập trình.

Nhằm giúp các bạn sinh viên, nghiên cứu sinh và các lập trình viên có được một tài liệu chuẩn về ngôn ngữ lập trình C/C++, Nhà xuất bản Thông tin và Truyền thông trân trọng giới thiệu cuốn sách ***"Giáo trình ngôn ngữ lập trình C/C++"*** do TS. Nguyễn Ngọc Cương (Trưởng khoa Toán tin, Học viện An ninh Nhân dân) làm chủ biên.

Nội dung cuốn sách gồm 7 chương:

Chương 1: Giới thiệu C/C++

Chương 2: Sử dụng các phép toán của C/C++

Chương 3: Sử dụng các lệnh của C/C++

Chương 4: Mảng và con trỏ

Chương 5: Cấu trúc và vào/ra tệp

Chương 6: Kỹ thuật đồ họa trong C/C++

Chương 7: Giới thiệu lập trình hướng đối tượng

Hy vọng cuốn sách không chỉ dành riêng cho các Học viên Học viện An ninh Nhân dân, Học viện Cảnh sát, còn là tài liệu tham khảo bổ ích cho các cán bộ giảng dạy, sinh viên các trường đại học cũng như các kỹ sư, kỹ thuật viên chuyên ngành Công nghệ Thông tin, Điện tử Viễn thông và Tự động hóa.

Nhà xuất bản xin trân trọng giới thiệu cùng bạn đọc và rất mong nhận được ý kiến đóng góp của quý vị. Mọi ý kiến đóng góp xin gửi về ***Nhà xuất bản Thông tin và Truyền thông*** - 18 Nguyễn Du, Hà Nội hoặc gửi trực tiếp cho tác giả theo địa chỉ: *cuongnnhvan@yahoo.com*.

Xin trân trọng cảm ơn./.

NXB THÔNG TIN VÀ TRUYỀN THÔNG

Chương 1

GIỚI THIỆU C/C++

Ngôn ngữ C/C++ là một trong các ngôn ngữ lập trình được sử dụng rộng rãi nhất trên thế giới. Đã có rất nhiều công ty tin học xây dựng chương trình dịch cho ngôn ngữ C/C++ mà dẫn đầu là Borland và Microsoft. Hiện nay ngôn ngữ C/C++ đã trở thành công cụ chuẩn, hiệu quả và thông dụng cho các lập trình viên chuyên nghiệp.

1.1. LỊCH SỬ C/C++

Những năm cuối của thập kỷ 60 đầu 1970, xuất hiện yêu cầu cần có ngôn ngữ bậc cao để hỗ trợ các nhà tin học trong việc xây dựng phần mềm hệ thống, hệ điều hành và do đó đã ra đời ngôn ngữ C tại phòng thí nghiệm điện thoại Bell Labs (Mỹ) để viết hệ điều hành Unix cho máy tính mini PDP11 của công ty DEC (Mỹ) và trở nên rất quen thuộc với các chuyên gia tin học.

Vào những năm 1980, Bjourn Stroustrup, làm việc tại Công ty Điện báo và Điện thoại (*AT&T - American Telephone & Telegraph*) đã bổ sung các đặc tính mới "hướng đối tượng" cho ngôn ngữ C. Các khái niệm về lập trình hướng đối tượng đã có trước đó trong các ngôn ngữ lập trình khác như Smalltalk, nhưng C++ là ngôn ngữ lập trình hướng đối tượng mạnh.

1.2. CHƯƠNG TRÌNH MÁY TÍNH LÀ GÌ?

Chương trình máy tính là dãy các lệnh cho máy tính biết cần phải thực hiện.

C/C++ là ngôn ngữ lập trình máy tính giúp chúng ta xây dựng chương trình để máy tính thực hiện đúng các công việc mà ta yêu cầu.

Để cung cấp các lệnh cho máy tính bằng ngôn ngữ lập trình C/C++ chúng ta cần có chương trình soạn thảo (Editor) và chương trình biên dịch (C++ compiler).

Các bước xây dựng chương trình máy tính bằng C/C++

1. Xác định vấn đề cần giải quyết bằng máy tính
2. Định nghĩa đầu ra chương trình
3. Chia nhỏ vấn đề thành các bước logic
4. Viết chương trình
5. Dịch chương trình
6. Kiểm tra, thử nghiệm chương trình

Các lệnh trong chương trình C/C++ được gọi là mã nguồn. Sau khi nhập mã nguồn ta có thể ghi lên đĩa từ dưới dạng tệp trước khi dịch và chạy chúng. Các tên tệp mã nguồn có phần mở rộng là .C cho ngôn ngữ C và .CPP cho ngôn ngữ C++.

Ví dụ: Tên tệp của chương trình C có thể là myprog.c còn trong C++ có thể là sales.cpp,...

Sau khi nhập mã chương trình ta phải dịch chương trình. Nếu có thông báo lỗi trên màn hình thì phải trở lại trình soạn thảo mã nguồn để sửa lỗi.

Trước khi tìm hiểu kỹ về C/C++ ta hãy làm quen với trình soạn thảo mã nguồn và trình biên dịch C++ của Borland thông qua ví dụ sau đây:

Ví dụ: Chương trình sau được viết bằng ngôn ngữ C:

```
#include <stdio.h>
main ()
{
printf("Xin chao");
}
```

Ví dụ trên sẽ đưa ra màn hình dòng chữ:

```
Xin chao
```

```
//Ten tep hello.cpp
// Chuong trinh hien thi dong chu "Hello C++!" tren man hinh
#include <iostream.h>
#include <conio.h>
main()
{
cout << "Hello C++!\n";
getch();
return 0;
}
```

Sau khi nhập xong mã chương trình, chúng ta nhấn tổ hợp phím (Alt + R) để chạy thử chương trình. Trên màn hình sẽ xuất hiện dòng chữ Hello C++! và chờ nhấn phím bất kỳ để thoát khỏi chương trình.

1.3. CẤU TRÚC CHƯƠNG TRÌNH C/C++

Để thấy rõ được cấu trúc chương trình của C/C++ chúng ta sẽ xét một số ví dụ viết trên ngôn ngữ C, sau đó sẽ xem xét viết trên C++

1.3.1. Một số chương trình ví dụ

Ví dụ 1: Lập chương trình sắp xếp một dãy số n thực a_1, a_2,... a_n theo thứ tự tăng dần.

Đây là bài toán ứng dụng nhiều trong thực tế như: sắp xếp kết quả thi, bảng kết quả học tập, sắp xếp danh sách cán bộ của một đơn vị theo tuổi…

Thuật toán:

1) Nhập số n và dãy số a_1, a_2,... a_n vào mảng a[1], a[2],... a[n];

2) for i →1 to n-1 do

3) for j → i+1 to n do

Kiểm tra điều kiện: nếu a[i] >a[j] thì đổi giá trị a[i], a[j] cho nhau;

4) In ra dãy số a[1], a[2],... a[n];

5) Kết thúc chương trình

Chương trình như sau:

```
# include <stdio.h>
# include <conio.h>
# define Nmax 50
short i,j, n;
float tg, A[Nmax];
void main()
{
clrscr();
printf("Nhap n="); scanf("%d",&n);
for (i=1; i<=n; i++)
{ printf("Nhap so thu %d ",i);
scanf("%f",&A[i]);
}
printf("\n");
for (i=1; i<=n-1; i++)
for (j=i+1; j<=n; j++)
if (A[i]>A[j])
{ tg=A[i];
A[i]=A[j];
A[j]=tg;
}
printf("Day so da duoc sap xep la \n \n");
for (i=1; i<=n; i++) printf("%4.1f",A[i]);
printf("\n");
getch();
}
```

Ví dụ 2: Lập chương trình tính số cách chọn k sinh viên vào học khoa Toán tin trong số n sinh viên trúng tuyển vào Học viện An ninh Nhân dân.

Thuật toán

Ta sử dụng hàm đệ quy TH để tính tổ hợp chập k của n.

```
TH(n,0) = TH(n,n) = 1;
TH (n, k) = TH (n-1,k) + TH(n-1,k-1) ;
// Chuong trinh de quy tinh to hop chap k cua n
# include <stdio.h>
# include <conio.h>
short nL, kL;
short TH(short n, short k)
{
if (n== k||k== 0) return 1;
else return TH(n-1,k)+ TH(n-1,k-1);
}
void main()
{
clrscr();
printf("Cho biet tong so hoc sinh n=");
scanf("%d", &nL);
printf("Cho biet so hoc sinh duoc chon");
scanf("%d", &kL);
printf("So cach chon la: %d",TH(nL, kL));
getch();
}
```

Nhận xét:

+ Chương trình C bắt đầu bằng #include để khai báo thư viện toán học, đồ họa, các đơn vị chương trình mà được gọi là chỉ thị tiền xử lý. Các tệp conio.h hay math.h phải có trong thư viện của C.

+ Dòng define để định nghĩa các hằng số, macro. Tương ứng với các biến kiểu nguyên, thực, ký tự chúng ta cũng sẽ có các hằng kiểu dữ liệu đó.

Bảng dưới đây cho ta một cấu trúc chung của chương trình viết trên ngôn ngữ C.

1.3.2. Cấu trúc chương trình C

# include <...>	Chỉ thị tiền xử lý bao hàm các tệp tiêu đề và tệp ghép
# define <Tên hằng> <Giá trị>	Định nghĩa các hằng, macro
typedef <Kiểu> <Tên kiểu>	Khai báo kiểu dữ liệu mới
<Tên kiểu> <Danh sách biến>;	Khai báo các biến toàn cục
<Tên kiểu> <Tên hàm 1>	Khai báo hàm 1
{ <Tên kiểu> <Danh sách biến>;	Khai báo các biến địa phương
Dãy câu lệnh	Các lệnh thực hiện trong hàm
....	
return <Biểu thức_1>;}	Trả kết quả cho hàm
main()	Chương trình chính - hàm main
{	
<Tên kiểu> <Danh sách biến>;	Khai báo các biến địa phương
Dãy câu lệnh;	Các câu lệnh thực hiện trong chương trình chính
.....	
}	

1.4. CÁC KÝ HIỆU CƠ BẢN TRONG NGÔN NGỮ C

Tập các ký hiệu cơ bản được gọi là bảng chữ cái của ngôn ngữ. Nó bao gồm nhóm các chữ cái, nhóm các chữ số và nhóm các ký hiệu đặc biệt.

a) Các chữ cái: A, B, C,... Z, a, b, c,... z, _ (dấu gạch nối)

b) Các chữ số: 0, 1, 2,..., 9

c) Các dấu phép toán

- Các phép toán số học: +, -, *, /, %
- Các phép so sánh: <, <=, >=, >, ==, !=
- Các phép toán logic: !, ||, &&
- Các phép toán trên bit: ~, |, &, ^, <<, >>
- Các phép tăng, giảm: ++, --
- Dấu phép gán: =

d) Các ký hiệu đặc biệt:

- Các dấu ngoặc (,), [,], {, }
- Các dấu chú thích /*, */, //
- Các ký hiệu khác ->, #, \,:, ?, ., ';

1.5. CÁC TỪ KHÓA CỦA NGÔN NGỮ C

Ngôn ngữ lập trình C sử dụng 38 từ tiếng Anh để tạo nên các khai báo, các biểu thức, câu lệnh. Đây là các từ khóa của ngôn ngữ C và được viết bằng chữ thường. Mỗi từ khóa mang một ý nghĩa nhất định do đó trong lập trình không được sử dụng từ khóa để đặt tên cho các đối tượng khác trong chương trình.

asm	far	short
auto	float	signed
break	for	sizeof
case	goto	Static
cdecl	huge	struc
char	if	switch
continue	int	typedef
default	interrupt	union
do	long	unsigned
double	near	void
else	pascal	volatile
enum	register	while
extern	return	

Ngoài các từ khóa, ngôn ngữ C còn đặt một số tên chuẩn cho các hằng, các kiểu, các biến, các hàm mẫu. Ví dụ: BLUE, M_PI, SEEK_CUR, clrscr; printf, sizeof, strcmp, stdin, textattr...

Khi đặt tên cho các đối tượng trong chương trình, chúng ta không được phép đặt trùng với tên chuẩn đã có.

1.6. CÁC MỞ RỘNG CỦA C++

Về thực chất C++ giống như C nhưng bổ sung thêm một số mở rộng khác như: đối tượng, lập trình hướng đối tượng. Từ các năm 70 của thế kỷ XX đã có các ý tưởng về cấu trúc trong C++ từ ngôn ngữ Simula 70 và Algol 68.

1.6.1. Các từ khóa mới của C++

Để bổ sung các tính năng mới vào C, các từ khóa sau đã được đưa vào C++:

asm	catch	class	delete	friend	inline
new	operator	private	protected	public	template
this	throw	try	virtual		

Như vậy, khi chúng ta chuyển chương trình từ C sang C++, cần phải thay đổi một số tên nếu trùng với từ khóa của C++ sau đó mới biên dịch lại.

1.6.2. Lệnh nhập và xuất dữ liệu chuẩn

Trong chương trình C, chúng ta thường sử dụng các hàm nhập/xuất dữ liệu là scanf() và printf(), còn trong C++ có thể dùng dòng nhập/xuất chuẩn (standard input/output stream) để nhập/xuất dữ liệu thông qua hai biến đối tượng của dòng (stream object) là *cout* và *cin*.

Xét ví dụ: Viết chương trình nhập vào hai số, sau đó tính tổng, hiệu, tích của hai số đó.

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

```
{ int a,b;
clrscr();
cout<<"Nhap so a=";
cin>>a;
cout<<"Nhap so b=";
cin>>b;
cout<<"Tong hai so la:"<<a+b<<"\n";
cout<<"Hieu hai so la:"<<a-b<<"\n";
cout<<"Tich hai so la:"<<a*b<<"\n";
getch();
return 0;
}
```

Để thực hiện dòng xuất chúng ta sử dụng biến *cout* (console output) kết hợp với toán tử chèn (insertion operator) <<. Còn để thực hiện dòng nhập, chúng ta sử dụng biến *cin* (console input) kết hợp với toán tử trích (extration operator) >>. Cần phải khai báo tệp iostream.h bằng include.

1.6.3. Cách chuyển đổi kiểu dữ liệu

Trong C++ có thêm cách chuyển đổi kiểu dữ liệu giống như lệnh gọi hàm.

Ví dụ:

```
# include<iostream.h>;
# include<conio.h>;
int main()
{
int X=200;
long Y = (long) X; //Chuyen doi theo cach cua C
long Z = long(X);//Chuyen doi theo cach moi cua C++
cout<<"X="<<X<<"\n";
```

```
cout<<"Y="<<Y<<"\n";
cout<<"Z="<<Z<<"\n";
getch();
return 0;
}
```

1.6.4. Các thay đổi khác trong C++

Vị trí khai báo biến

Chương trình C đòi hỏi tất cả các khai báo bên trong một phạm vi cho trước phải đặt ở ngay đầu của phạm vi đó. Tức là, các khai báo toàn cục phải đặt trước tất cả các hàm và các khai báo cục bộ phải được tiến hành trước tất cả các lệnh thực hiện.

Ngược lại, trong C++, cho phép chúng ta khai báo linh hoạt bất kỳ vị trí nào trong một phạm vi cho trước (không nhất thiết phải ngay đầu của phạm vi), có thể xen kẽ khai báo dữ liệu với các câu lệnh thực hiện. Khi khai báo một biến trong chương trình, biến đó sẽ có hiệu lực trong phạm vi của chương trình kể từ vị trí nó xuất hiện, vì vậy chúng ta không thể sử dụng một biến được khai báo bên dưới nó.

Ta xét ví dụ sau đây được viết trên C:

```
void makeit(void)
{ float i;
  char *cp;
/* Doan chuong trinh */
/* Cấp 100 bytes cho cp */
cp = malloc(100); /* lần thứ nhất dùng cp */
for (i=0; i<100;  ++i) /* lần thứ nhất dùng i */
{
/* Một số lệnh trong chu trình */
}
/* chương trình tiếp tục */
```

Sau đoạn chương trình không gian cho cp được cấp và i cuối cùng cũng được dùng trong chu trình for. Tuy nhiên, người lập trình quên biến i đã khai báo rồi và do i là float nên vòng lặp for được tạo ra dùng float nên nó chạy chậm hơn thông thường. Các khai báo ở xa nơi sử dụng dữ liệu thường hay gây nhầm lẫn nên dẫn đến lỗi.

C++ cho phép khai báo được đặt gần nơi sử dụng hơn. Chương trình trên được viết lại như sau:

```
 void makeit(void)
{
   char *cp;
/* Doan chuong trinh */
/* Cấp 100 bytes cho cp */
char *cp = new char[100] ;
for (i=0; i<100;  ++i) /* lần thứ nhất dùng i */
{
/* Một số lệnh trong chu trình */
}
/* chương trình tiếp tục */
```

Chương trình này dễ theo dõi hơn bằng cách khai báo biến i và cp lần đầu tiên dùng chứ không phải khai báo trên đoạn lệnh.

Các hằng const

Trong C, muốn định nghĩa một hằng có kiểu nhất định thì ta dùng khai báo *const* vì nếu dùng #define thì tạo ra các hằng không chứa thông tin về kiểu. Còn trong C++, khai báo *const* rất linh hoạt:

- C++ xem *const* cũng như #*define* để định nghĩa hằng trong chương trình, ví dụ:

```
const int n =100;
int X[n];
```

Trong C++, chúng ta phải khởi tạo giá trị ban đầu của hàm, còn trong C thì nó tự động gán giá trị zero cho mọi hằng định nghĩa bằng *cout* nếu ta không định nghĩa giá trị ban đầu cho nó.

Trong C, các hằng định nghĩa bằng *const* được khai báo ở bên ngoài mọi hàm thì chúng có phạm vi toàn cục, tức là chúng có thể nhìn thấy cả ở bên ngoài file mà chúng được định nghĩa trừ khi chúng được khai báo là static.

Nhưng trong C++, các hằng định nghĩa bằng *const* được hiểu mặc định là static. Các giá trị const trong C++ có thể linh hoạt hơn rất nhiều, có thể được dùng ở mọi vị trí nên cho phép lập trình viên không cần dùng #define để tạo hằng không có thông tin kiểu.

Đoạn chương trình sau được chấp nhận trong C++ nhưng sẽ bị lỗi trong C:

```
const int  ArraySize =100;
int Array[Arraysize];
```

Còn có sự khác biệt về các kiểu dữ liệu *struct, union* và *enum*, các toán tử *new* và *delete* giữa C và C++, chúng ta sẽ xem xét sau.

C++ cho phép dùng hai kiểu chú thích là /* */ như trong C và //. Trình biên dịch sẽ bỏ qua mọi thứ nằm trong /* */ và //, tuy nhiên kiểu // được dùng cho chú thích trên một dòng ít được sử dụng hơn.

1.7. BIẾN VÀ HẰNG

1.7.1. Biến

Biến là một đại lượng thay đổi khi thực hiện chương trình. Biến là một thành phần quan trọng trong một chương trình, dù chương trình được viết bằng ngôn ngữ nào. Biến có thể lưu trữ các kiểu dữ liệu khác nhau, dưới đây là các kiểu biến cơ sở và khả năng lưu trữ của chúng trong C/C++.

Tên khai báo	Kiểu	Khả năng lưu trữ
short	số nguyên	từ –128 đến 127
unsigned char	ký tự không dấu	từ 0 đến 255
int	số nguyên	từ –32768 đến 32767
unsigned int	số nguyên không dấu	từ 0 đến 65535
long	số nguyên dài	từ –2147483648 đến 2147483647
unsigned long	số nguyên dài không dấu	từ 0 đến 4294967295
float	số dấu phảy di động	từ -3.4E-38 đến 3.4E+308
double	số dấu phảy di động dài gấp đôi	từ –1.7E-308 đến 1.7E+308
char	Kiểu ký tự	1 giá trị trong bảng ASCII có 256 ký tự
Không định nghĩa tường minh	Kiểu logic	1 biểu diễn giá trị logic đúng; 0 biểu diễn giá trị logic sai

Trong ngôn ngữ C/C++, biến phải được khai báo trước khi dùng. Việc định nghĩa biến phải xác định được hai thành phần: tên biến, kiểu biến. Trong đó:

Kiểu biến xác định giới hạn miền giá trị mà biến có thể lưu giữ và biểu diễn. Các kiểu biến có thể được ngôn ngữ C/C++ định nghĩa sẵn hoặc do người dùng có thể tự định nghĩa dựa trên các kiểu đã có sẵn.

Tên biến do người dùng tự đặt nhưng phải tuân thủ theo quy tắc là một dãy ký tự không có dấu cách, có thể có cả chữ, số và dấu gạch dưới nhưng phải bắt đầu bằng chữ cái hoặc dấu gạch dưới. Tên biến có phân biệt chữ hoa và chữ thường, và mang tính gợi nhớ để người lập trình dễ sử dụng.

Chú ý:

- Tên biến thường được viết bằng chữ thường

- Tên biến cần được đưa ra theo một tên gợi nhớ, tức là tên nêu ra được một chỉ dẫn nào đó về việc dùng của nó trong chương trình.

- Tên biến có nhiều từ thì các từ nối với nhau bằng dấu gạch dưới (_).

C++ dành riêng một tập các từ để dùng bên trong ngôn ngữ xem như các từ khóa. Người dùng không được dùng tên gọi cho các từ khóa làm tên gọi riêng cho chương trình.

Có thể khai báo nhiều biến có cùng một kiểu trên cùng một hay nhiều dòng và ngăn cách nhau bởi dấu phảy kết thúc bằng dấu chấm phảy.

Khi định nghĩa biến ta có thể gán giá trị khởi đầu cho biến. Đây là điểm khác của C/C++ so với Pascal.

Cú pháp khai báo biến:

<Kiểu dữ liệu> **<Danh sách tên biến>**;

Trong đó:

Kiểu dữ liệu: Tên kiểu dữ liệu của một hoặc nhiều biến cần khai báo.

Danh sách tên biến: Nếu có nhiều biến thuộc cùng một kiểu dữ liệu, ta có thể liệt kê danh sách các tên biến, phân cách nhau bởi dấu phảy (,). Kết thúc khai báo là dấu chấm phảy (;).

Việc khai báo biến nêu trên chưa tạo ra việc cấp phát bộ nhớ cho biến, nó chỉ xác nhận rằng biến đó đã được định nghĩa đâu đó trong chương trình. Một biến có thể được khai báo nhiều lần trong chương trình.

Ví dụ 1: khai báo biến:

```
int i; /* biến tên i có kiểu nguyên */
float f1, f2; /*các biến tên f1 và f2 đều có kiểu số thực dấu phảy
động*/
int j = 5; /* khai báo biến j và khởi tạo giá trị là 5*/
```

Ví dụ 2:

```
# include <stdio.h>
main()
{
int a, b,c ; /*khai báo 3 biến kiểu int */
long ad, biểu diễn, cd; /*khai báo 3 biến kiểu long*/
```

```
unsigned au, bu, cu; /*khai báo 3 biến kiểu unisigned */
char k1, k2; /* khai báo 2 biến kiểu char */
float x, y; /*khai báo 2 biến kiểu float */
double xd, yd; /*khai báo 2 biến kiểu double */
```

Một biến phải được định nghĩa hay khai báo cho chương trình trước khi nó được sử dụng.

Chú ý: Việc khai báo biến trong C/C++ và Pascal có sự khác nhau.

Đối với Turbo C	Đối với Pascal
Kiểu DS biến;	DS biến:Kiểu;

```
# include <stdio>
main()
{
int a, b, c;
..
}
```

```
Var
a, b, c: Integer;
Begin
....
End;
```

Vị trí khai báo biến

Các biến trong C/C++ có thể được khai báo bên ngoài hoặc bên trong hàm hay khối lệnh.

Khai báo biến ngoài

Vị trí các biến đặt ở bên ngoài tất cả các hàm. Các biến này sẽ tác động tới toàn bộ chương trình của C/C++.

Ví dụ:

```
int n, tong ; /*Khai báo biến ngoài */
main ()
{
n = tong +10;
...
}
```

Khai báo biến trong

Các biến được đặt ở đầu của một hàm hoặc khối lệnh. Các biến này chỉ tác động trong phạm vi hàm hoặc khối lệnh đó.

Ví dụ:

```
func1()
{
int i, j ; /*Khai báo biến trong */
....
for (i=j;
...
}
```

Khai báo các đối số hàm

Vị trí biến đặt trong phần định nghĩa tham số của hàm.

Ví dụ:

```
func2(x,c);
    int x /*Khai báo đối số hàm */
    char c;
    { x= x-1;
    ....
    }
```

Trong C, khi khai báo các biến trong, chúng ta cần phải đặt biến ngay sau dấu { đầu tiên của thân hàm và cần phải đứng trước các câu lệnh khác. Như vậy sau một lệnh khác (chẳng hạn như câu lệnh gán) thì không được khai báo biến nữa, nếu không C sẽ báo sai như đoạn chương trình sau:

Ví dụ:

```
    /* Khai báo sai vị trí biến */
    main()
    { int a, b, c; /*Khai báo đúng */
    a = 35;
    int d ; /*Khai báo sai vì sau câu lệnh gán */
```

Trong C++ thì việc khai báo ở những vị trí nào là tùy ý.

Việc gán giá trị đầu tiên cho biến có thể thực hiện bằng 1 trong 2 cách: khai báo trước, gán giá trị sau hoặc vừa khai báo vừa gán giá trị.

Ví dụ:

```
main()
    { int a, b = 20, c, d = 40;
    float e = -35.1, f = 23.0;
    int i;
    i = 10;
    ...
    }
```

1.7.2. Hằng

Biến kiểu hằng được định nghĩa như một biến thường và thêm vào đằng trước từ khóa *const*. Biến hằng được gán giá trị khởi đầu khi định nghĩa và không được phép thay đổi giá trị của nó trong toàn bộ chương trình.

Trong C/C++ mỗi hằng hoặc biểu thức hằng đều có một trị và kiểu dữ liệu của chúng. Đó có thể là: hằng số thực (*float, double*), hằng nguyên (*int, long*), hằng ký tự, hằng xâu ký tự.

1.8. MẢNG KÝ TỰ VÀ XÂU KÝ TỰ

Trong các loại biến thường dùng trong ngôn ngữ C/C++, loại biến mảng các ký tự là loại biến thường được sử dụng. Biến mảng ký tự để lưu trữ liên tiếp các ký tự.

Cú pháp:

char tên_biến[kích cỡ];

với: - char: kiểu ký tự.

- tên_biến: tên của biến mảng ký tự.
- []: ký hiệu mảng.
- kích cỡ: số phần tử của mảng.

Ví dụ: Để lưu trữ tên của một sinh viên, ta khai báo một biến mảng ký tự như sau: *char TenSV[15];*

Có nghĩa là ta khai báo một biến mảng kiểu ký tự có độ dài 15 ký tự. Ta có thể gán giá trị cho biến mảng ký tự ngay khi khai báo:

```
char TenSV[15] = "Nguyen Van A";
```

hoặc khai báo biến mảng ký tự và gán giá trị theo kiểu sau:

```
char TenSV[15] = {'N', 'g', 'u', 'y', 'e', 'n', '', 'V', 'a', 'n', ' ', 'A'};
```

Trong bộ nhớ máy tính sẽ có 15 ô nhớ liên tiếp, mỗi ô chứa một ký tự gọi là một phần tử: phần tử thứ 12 (ký tự \0) là ký tự kết thúc xâu ký tự.

Nếu gán ngay giá trị cho mảng ký tự lúc khai báo biến thì có thể không cần khai báo kích cỡ của mảng, chương trình sẽ tự động đếm và xác định số ký tự cho biến mảng.

```
char TenSV[] = "Nguyen Van A";
```

Ta có thể đọc hay ghi dữ liệu từ biến mảng ký tự.

Ví dụ: Ta thực hiện phép gán sau:

```
TenSV[11] = 'B';
```

Không như Pascal, trong C/C++, chỉ số của mảng phải được bắt đầu từ 0, truy nhập đến phần tử tiếp theo bằng cách tăng chỉ số lên 1 đơn vị.

Ví dụ: TenSV[i+1] xác định phần tử thứ i+1 là phần tử đứng ngay sau phần tử i. Tương tự khi truy nhập đến phần tử đứng ngay trước nó thì giảm chỉ số đi 1 đơn vị.

Ví dụ: TenSV[i-1] xác định phần tử thứ i-1 là phần tử đứng ngay trước phần tử i.

Xâu ký tự là mảng ký tự có phần tử tận cùng là ký tự kết thúc xâu '\0'.

Với xâu ký tự, chúng ta không thể gán trực tiếp dữ liệu như với các kiểu biến khác:

Ví dụ:

```
char TenSV[20];
TenSV = "Nguyen Van A"; // C++ sẽ báo lỗi khi dịch
```

Ngoài kiểu khai báo và gán khởi đầu như trên thì chỉ có thể thực hiện gán trực tiếp từng phần tử như sau:

```
TenSV[0] = 'N';
TenSV[1] = 'g'
TenSV[2] = 'u';
TenSV[3] = 'y';
TenSV[4] = 'e';
TenSV[5] = 'n';
TenSV[6] = '\0';
```

Ngôn ngữ C++ cung cấp cho người dùng hàm thư viện strcpy() trong thư viện hàm và được dùng như sau:

```
strcpy(TenSV, "Nguyen");
```

Hàm này tự động thêm ký tự kết thúc xâu vào cuối mảng ký tự TenSV"[].

1.9. CÁC LỆNH ĐIỀU KHIỂN TIỀN DỊCH CHƯƠNG TRÌNH

Chương trình dịch của *C/C++* không dịch ngay ra chương trình nguồn mà thực hiện các tiền xử lý trước, sau đó mới dịch chương trình. Công việc tiền xử lý bao gồm: thực hiện chỉ thị bao hàm tệp, định nghĩa các hằng, macro, dịch và thực hiện chương trình có điều kiện.

C/C++ có một thư viện chuẩn bao gồm các tệp tiêu đề (*header*) chuẩn, chứa các thông tin về thư viện mà người dùng cần tới. Để thâm nhập vào các biến hay hàm được định nghĩa bên trong thư viện chuẩn, ta phải khai báo các tệp tiêu đề liên kết vào chương trình của mình. Tệp tiêu đề phải là tệp mã ASCII (tệp văn bản).

Để sử dụng tệp tiêu đề trong chương trình ta phải sử dụng chỉ thị *#include*. Chỉ thị này được xử lý trước khi gọi tới trình biên dịch ngôn ngữ.

Chỉ thị #include có hai dạng:

#include <tên tệp> ví dụ: #include <stream.h>

#include "tên tệp" ví dụ: #include "myIO.h"

Nếu tên tệp được bao bởi dấu ngoặc góc (<...>) thì tệp có tên nằm trong móc nhọn là tệp tiêu đề đã được xác định sẵn, hay chuẩn. Việc tìm kiếm nó được thực hiện bằng cách duyệt trong tập các vị trí định sẵn bằng chương trình dịch.

Nếu tên tệp được bao quanh bởi dấu nháy kép ("...") thì tệp có tên nằm trong dấu nháy là tệp tiêu đề do người dùng cung cấp. Việc tìm tệp này được thực hiện bắt đầu từ thư mục hiện thời (*current dir*). Nếu không tìm thấy thì sẽ duyệt tìm trong tập các vị trí đã định sẵn.

Tệp được đưa vào bản thân nó lại có thể chứa chỉ thị *#include*. Do các tệp được đưa vào lồng nhau nên một tệp tiêu đề đôi khi có thể được kéo vào nhiều lần đối với một tệp gốc. Có thể dùng các chỉ thị điều kiện để khắc phục lại việc xử lý nhiều lần cho một tệp tiêu đề.

Ví dụ:

```
#ifndef STRING_H
#define STRING_H
/* nội dung của string.h*/
#endif
```

Chỉ thị điều kiện *#ifndef* cho lại giá trị đúng nếu tên gọi đi sau nó chưa được định nghĩa. Khi một chỉ thị điều kiện cho lại giá trị đúng thì các dòng đi sau nó được thực hiện cho đến khi gặp *#endif*.

Chỉ thị *#define* định nghĩa tên đi sau nó. Như ví dụ trên nó định nghĩa STRING_H

Chỉ thị *#define* còn có ý nghĩa để định nghĩa giá trị một hằng số nào đó. Sau đó ta có thể sử dụng chúng về sau trong chương trình.

Ví dụ:

```
#define BUFSIZE 512
#define MONTHS 12
```

1.10. CÁC LỆNH VÀO/RA ĐƠN GIẢN

Các thiết bị vào ra chuẩn thông dụng gồm có thiết bị vào là bàn phím, thiết bị ra là màn hình, máy in.

Sau đây ta xem xét việc sử dụng một số hàm vào ra đơn giản với các thiết bị chuẩn trong ngôn ngữ C/C++.

1.10.1. Câu lệnh nhập dữ liệu vào biến nhớ

Cú pháp:

scanf("ký tự định dạng", *danh sách biến*);

Ký tự định dạng bắt đầu bằng dấu % cho phép mô tả dữ liệu nhập hoặc dữ liệu xuất. Danh sách biến quy định như sau:

- Mỗi biến phải bắt đầu bằng dấu & để định địa chỉ biến và phân cách nhau bằng dấu phẩy. Có bao nhiêu biến phải có bấy nhiêu định dạng và thứ tự của các định dạng phải phù hợp với thứ tự của danh sách.

Để nhập được dòng ký tự có khoảng trắng bằng hàm scanf phải viết:

```
scanf("%[^\n]",a)) /*a là chuỗi xâu ký tự được nhập */
```

Ví dụ 1: Định dạng chỉ nhập số nguyên, không nhập số thực cho 2 biến x và y: *scanf ("%d%d",&x, &y);*

Ví dụ 2: Định dạng nhập 1 số thực có tối đa là 2 số lẻ

```
scanf("%.2f",&y);
```

Ví dụ 3: Định dạng để nhập 10/20 như sau:

```
scanf("%d%c%d", &x, &y);
```

Có nghĩa là 10 gán cho x, 20 gán cho y, bỏ qua dấu (/) không gán cho biến nào cả nhưng vẫn đọc được.

Ví dụ 4:

```
#include <stdio.h>
void main()
{
 float total_sale;
float vat_tax;
printf("\nTong so tien hang ban duoc trong ngay: ");
scanf("%f", &total_sale);
vat_tax = total_sale * 0.1;
printf("\nThue vat phai nop cua %.2f la: %.2f", total_sale, vat_tax);
}
```

Trong C/C++ thường dùng hàm *gets* để nhận một chuỗi từ bàn phím cho đến khi gặp ký tự \n.

Cú pháp:

gets(tên của mảng ký tự);

Khi nhập chuỗi ta không cần phải lấy địa chỉ.

Ví dụ:

```
#include <stdio.h>
char  a[5]; /*Khai bao chuoi a do dai 5 ky tu */
main()
/* ham chinh */
int i;
printf("Nhap chuoi 5 ky tu:  ");
gets(a);
/* gia su nhap cuong */
print f("%s", a);     /*In ra cuong */
```

Để nhận một ký tự từ bàn phím và trả về mã ký tự nhận được ta dùng hàm getchar.

Cú pháp:

int **getchar**(*void*).

Ví dụ:

```
int  j;
printf ("Nhap 1 ky tu:");
j = getchar();  /* Nhap mot ky tu tu ban phim */
printf("%c\n",j)  /*in ra man hinh ma ASCII cua ky tu nhap vao */
```

Giả sử ta gõ 'B' thì chương trình sẽ in ra 66 là mã ASCII của 'B'.

Lưu ý về các hàm nhập dữ liệu scanf, gets, getchar

Đây là các hàm nhận dữ liệu từ vùng nhớ đặc biệt stdin theo nguyên tắc:

- Nếu trên *stdin* có đủ dữ liệu thì chúng sẽ nhận một phần dữ liệu mà nó yêu cầu, phần dữ liệu còn lại vẫn ở trên *stdin*.

- Nếu trên *stdin* không đủ dữ liệu theo yêu cầu của hàm thì máy tạm dừng để chờ người sử dụng đưa dữ liệu từ bàn phím lên *stdin* cho đến khi gặp phím.

- Xóa khỏi *stdin* phần dữ liệu đã lấy.

- Hàm *gets* sẽ xóa ký tự \n trong *stdin*

- Hàm *scanf* và *getchar* không xóa \n trong *stdin*

Như vậy nếu trong chương trình có sử dụng các lệnh *scanf*, *getchar* thì các lệnh sử dụng sau sẽ không có tác dụng do mã phím \n còn lại trong *stdin* của lệnh *scanf* hoặc *getchar* trước đó.

Ví dụ:

```
# include <stdio.h>
main ()  /* Ham chinh * /
{
char a, b;
printf("Nhap mot so: ");
```

```
scanf("%d", &a);
printf("Nhap mot ky tu: ");
scanf("%c", &b);
printf("\n%d %c", a, b);
}
```

Giả sử ta nhập 65 sau đó gõ Enter. Khi đó lệnh *scanf* thứ hai bị trôi và kết quả là a = 65 và b = <enter>

Trường hợp ta nhập 65ABC sau đó gõ Enter thì lệnh *scanf* thứ hai cũng trôi như trường hợp đầu và a = 65 và b = A.

Vì vậy, để các hàm sau hoạt động đúng, chúng ta cần phải khử ký tự \n còn lưu trong *stdin* bằng lệnh *fflush*(stdin) sau lệnh *scanf* hoặc dùng ký tự đặc tả %*c trong lệnh *scanf*.

Như vậy, ta có thể sửa lại đoạn chương trình trên như sau:

```
# include <stdio.h>
main ()  /* Ham chinh * /
{
char a, b;
printf("Nhap mot so: ");
scanf("%d", &a);
fflush(stdin);
/* Hoặc scanf("%d%*c", &a) */
printf("Nhap mot ky tu: ");
scanf("%c", &b);
printf("\n%d %c", a, b);
}
```

Trong thư viện conio.h còn có các hàm getch() và getche().

Cú pháp: int ***getch***(void)

int ***getche***(void)

Hai hàm này chờ nhận một ký tự trực tiếp từ bộ nhớ đệm của bàn phím. Nếu bộ đệm rỗng thì chờ. Khi một phím được nhấn thì nhận

ngay ký tự đó mà không cần ta gõ phím Enter như các hàm nhập từ *stdin*. Kết quả trả về của hàm là ký tự được nhấn.

Điểm khác nhau giữa *getche()* và *getch()* là: *getche()* cho hiện ký tự lên màn hình còn *getch()* thì không cho hiện.

Để nhận các phím đặc biệt (các phím chức năng như F1, F2...) cần chú ý khi được nhấn sẽ có hai giá trị được gửi lên bộ đệm bàn phím. Ví dụ, nhấn phím F1, có giá trị đầu là 0 và giá trị tiếp là 59 được gửi lên bộ đệm, do đó để nhận được mã các phím này ta phải đọc bộ đệm bàn phím thêm một lần nữa.

Trong C++ thường sử dụng: Toán tử cin

cin để nhận dữ liệu từ bàn phím và lưu trữ vào biến nhớ.

Cú pháp:

cin >> value [>> value];

value: biến dữ liệu

Khi gặp lệnh *cin*, chương trình sẽ chờ người sử dụng nhập dữ liệu vào từ bàn phím, quá trình nhập liệu cho biến dữ liệu kết thúc khi người sử dụng nhấn phím Enter hoặc phím Space.

Ví dụ:

```
#include <iostream.h>
void main()
{
char name[30];
cout << "\nHay nhap vao ten của ban: ";
cin >> name;
cout << "\nChao ban "<< name <<". Rat vui vi ban da den voi C++ !";
}
```

Kết quả hiển thị:

Hay nhap vao ten cua ban:...

Chao ban... Rat vui vi ban da den voi C++ !

Trong ngôn ngữ C thường hay sử dụng hàm printf() và scanf() cho các thao tác vào/ra dữ diệu với các thiết bị vào ra chuẩn.

1.10.2. Câu lệnh xuất giá trị

Cú pháp:

printf("định dạng", Danh sách biến);

Trong đó, *định dạng* quy định:

- Có bao nhiêu biến phải có bấy nhiêu định dạng và thứ tự của các định dạng phải phù hợp với thứ tự của danh sách biến.

- Mã định dạng phải phù hợp với kiểu dữ liệu biến và bắt đầu bằng dấu phần trăm %.

Ví dụ:

```
#include <stdio.h>
void main()
{
 char hoten[] = “Nguyen Van A”;
int tuoi=32;
float hesoluong = 5.25;
printf (“\n%s”, hoten);
printf(“, Tuoi: %d”, tuoi);
printf(“, Luong: %f”, hesoluong);
}
```

Kết quả hiển thị:

```
Nguyen Van A, Tuoi: 32, He so luong: 5.25
```

Ngoài hàm printf, trong C/C++ còn có các hàm xuất dữ liệu sau:

Hàm putchar

Hàm putchar để xuất một ký tự lên màn hình. Kết quả của hàm trả về ký tự xuất nếu thành công và cho giá trị EOF(-1) nếu không thành công.

Cú pháp:

int ***putchar*** (int ch);

Ví dụ:

```
char c= 'C' ;
putchar (c) ; /* in ra màn hình ký tự C */
```

Hàm puts

Hàm puts có chức năng xuất một chuỗi ký tự ra màn hình.

Cú pháp:

int ***puts***(char *s);

Hàm này sẽ xuất chuỗi lên màn hình với *s là con trỏ kiểu ký tự char trỏ tới ô nhớ đầu của vùng nhớ chứa chuỗi ký tự cần xuất. Hàm này khi xuất sẽ đưa thêm ký tự \n vào cuối, kết quả của hàm =\n nếu thành công, ngược lại hàm cho giá trị là EOF.

Hàm fprintf

Hàm fprintf cho phép xuất thông tin ra máy in.

Cú pháp:

fprintf(stdprn, "Định dạng xuất, <danh sách các biểu thức>);

Hàm này tương tự như hàm in ra màn hình nhưng yêu cầu chúng ta thêm vào tên quy ước của máy in.

Trong C++ thường sử dụng toán tử cout để xuất dữ liệu.

Toán tử cout

cout đưa dữ liệu ra thiết bị ra chuẩn.

Cú pháp:

cout << data [<<data];

data: có thể là một biến dữ liệu, một hằng, một biểu thức hoặc kết hợp của các loại trên.

Ví dụ:

```
#include <iostream.h>
void main()
{
 int i = 1;
 char Head[] = “Gia tri cua i la: ”;
 cout << “\nKiem tra lenh cout !\n” << Head << i;
}
```

Kết quả hiển thị:

```
Kiem tra lenh cout
Gia tri cua i la: 1
```

Danh sách các mã định dạng thông dụng của C/C++

Khuôn dạng	**Ý nghĩa**
%s	Đưa ra dữ liệu dưới dạng xâu ký tự (đến khi có dấu kết thúc xâu)
%c	Đưa ra dữ liệu dạng ký tự
%d	Đưa ra dữ liệu dạng số nguyên
%f	Đưa ra dữ liệu dạng số float
%u	Đưa ra dữ liệu dạng số nguyên dương
%x	Đưa ra dữ liệu dạng số cơ số 16 (số Hexa)
%%	In ra dấu %

Ví dụ 1: Viết chương trình nhập vào 4 số thực a, b, c, d; sau đó tính và đưa ra màn hình tổng và trung bình cộng của 4 số đó.

```
include <stdio.h>
main()
{
 float a, b, c, d, e, f ;
clrscr();
printf(“\n - nhap so thu nhat= ”  ); scanf(“%f”, &a);
printf(“\n - nhap so thu hai=” ); scanf(“%f”, &b);
```

```
printf(“\n - nhap so thu ba=”); scanf(“%f”, &c);
printf(“\n - nhap so thu tu=”); scanf(“%f”, &d);
e = a+b+c+d;
printf(“\n Tong cua 4 so vua nhap vao= %f”,e);
f = (a+b+c+d)/4;
printf(“\n trung binh cong cua 4 so vua nhap vao= %f”,f);
getch();
}
```

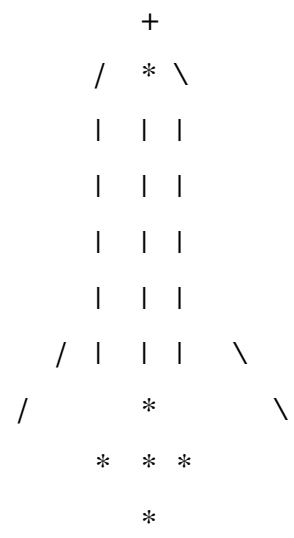
Ví dụ 2: Hãy chạy chương trình sau để xem tác dụng của mã định dạng.

```
/* Khai bao va gan tri dau tien cho bien */
main()
{
int i,j=3,k; /* Chi co j duoc gan gia tri */
float x=15;
float y=1.2e-3; /* Mo ta so thuc o dang khoa hoc */
char c1=“A”;
char c2=“\x41”; /* Mo ta ky tu bang tri ASCII o dang thap luc */
long l1=123L; /* Hang 123 luu tru o dang long */
double z1=.5E8;
printf("KET QUA IN RA MAN HINH \n");
printf("j=%d\n",j);
printf("x=%f,y=%e\n",x,y);
printf("c1=%x,c2=%c\n",c1,c2);
printf("l1=%Ld,z1=%LE\n",l1,z1);
getch();
}
```

Bài tập chương 1

1. Viết chương trình C++ để lưu trữ ba biến: trọng lượng, chiều cao và cỡ giày của anh, chị. Khai báo biến sau đó gán giá trị cho chúng.
2. Bổ sung lệnh cout hoặc/và printf() vào chương trình trong ví dụ trên để hiển thị các giá trị đã lưu trữ.
3. Viết chương trình lưu trữ giá trị và hiển thị các kiểu biến của C++ mà các anh chị biết.
4. Viết chương trình lưu trữ tên hai chương trình truyền hình mà anh/chị thích nhất.
5. Viết chương trình hiển thị tên, tuổi của anh chị. Lưu trữ chúng trong tệp với tên tùy ý. Sau đó viết tệp thứ hai gộp tệp thứ nhất với khả năng hiển thị tên và tuổi của anh chị.
6. Viết chương trình định nghĩa 10 tên hằng từ 0 đến 9, ví dụ ZERO đến NINE. Sử dụng chúng để hiển thị lên màn hình.
7. Viết chương trình nhập điểm thi của các môn học trong học kỳ vừa qua của anh chị. Sau đó hiển thị điểm trung bình cộng.
8. Sửa đổi chương trình trong bài tập 7 để nhập điểm và tên sinh viên bất kỳ trong lớp.
9. Viết chương trình nhập vào ba cạnh của một tam giác a, b, c sau đó tính diện tích tam giác theo công thức: $S = \sqrt{p(p-a)(p-b)(p-c)}$ trong đó p = (a + b + c)/2.
10. Viết chương trình nhập từ bàn phím và sau đó xuất ra màn hình, các thông tin của một mặt hàng bao gồm: tên mặt hàng, trọng lượng, đơn giá, mã chất lượng.

11. Sử dụng lệnh cout để viết chương trình hiển thị bức tranh như dưới đây:



Chương 2

SỬ DỤNG CÁC PHÉP TOÁN CỦA C/C++

Nội dung của chương sẽ trình bày các vấn đề sau:

- Biểu thức
- Các phép toán số học
- Các phép toán quan hệ và logic
- Các phép toán khác, lệnh if
- Thứ tự ưu tiên giữa các phép toán

2.1. BIỂU THỨC

Biểu thức là sự kết hợp hợp lệ một hay nhiều phép toán thực hiện trên các biến, hằng hoặc các giá trị của hàm.

Ví dụ: float a, b=3.5; b=(a=b*25);

Đối tượng của các phép toán được gọi là toán hạng. Các phép toán được biểu thị thông qua các toán tử.

Toán tử tác động lên một toán hạng gọi là toán tử một ngôi. Toán tử tác động lên hai toán hạng gọi là toán tử hai ngôi.

Việc tính một biểu thức sẽ thực hiện một hay nhiều phép toán và cho lại một kết quả, kiểu dữ liệu của kết quả được xác định bởi kiểu dữ liệu của các toán hạng. Khi hai hay nhiều toán hạng được tổ hợp với nhau qua toán tử, biểu thức được xem như là một biểu thức tính toán. Trật tự tính toán được xác định bởi thứ tự ưu tiên và luật kết hợp của các toán tử.

Cần lưu ý, trong các biểu thức của C/C++ không dùng các cặp dấu [,], {, } như trong các biểu thức toán học mà đều sử dụng dấu ngoặc tròn (và).

2.2. CÁC PHÉP TOÁN SỐ HỌC

Toán tử	Chức năng	Cách dùng
+	Cộng	Expr + Expr
-	Trừ	Expr - Expr
*	Nhân	Expr * Expr
/	Chia	Expr / Expr
%	Lấy phần dư	Expr % Expr

trong đó:

- Expr là toán hạng.

- Phép chia giữa hai số nguyên cho kết quả là một số nguyên. Nếu thương số chứa phần thập phân thì nó sẽ bị làm tròn.

- Phép toán lấy phần dư (%) tính phần dư của phép chia giữa hai giá trị. Nó chỉ áp dụng cho các toán hạng thuộc kiểu nguyên tức là cả hai toán hạng cùng phải có kiểu nguyên.

Ví dụ:

```
int so1 = 10;
int so2 = 3;
int so3 = so1 / so2; /* phep chia hai so nguyen cho ket qua la so
nguyen 10/3 = 3 */
int so4 = so1 % so2; /* phep modulo (%) cho phan du cua phep chia
10 % 3 = 1 */
```

Các trường hợp đặc biệt:

Có thể xảy ra các trường hợp đặc biệt như: chia cho không hoặc tràn ô.

Tràn ô xảy ra trong phép gán, ví dụ ta xem xét các câu lệnh gán sau:

```
unsigned char uc=32;
int i = 8;
uc = i*uc; // tràn ô
```

Lúc này máy tính sẽ dừng thực hiện chương trình và thông báo lỗi *overflow*.

2.3. CÁC PHÉP TOÁN QUAN HỆ VÀ LOGIC

Các phép toán quan hệ và logic cho lại một giá trị đúng hay sai. Một điều kiện đúng sẽ cho giá trị là 1 và sai cho giá trị là 0. Các phép toán quan hệ gồm: == (bằng), != (không bằng), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Các phép toán logic bao gồm: ! (phủ định hay NOT), && (nhân logic hay phép AND), || (cộng logic hay phép hoặc OR). Trong lập trình, các phép toán logic có thể kết hợp nhau để tạo nên biểu thức logic.

Toán tử so sánh bằng

Cú pháp: ***expr1 = = expr2***

Ý nghĩa: là toán tử so sánh 2 toán hạng.

Toán tử AND (và)

Cú pháp: ***expr1 && expr2***

Ý nghĩa: Trả lại giá trị đúng chỉ khi cả hai toán hạng của nó có giá trị đúng.

1 && 1 = 1 1 && 0 = 0

0 && 1 = 0 0 && 0 = 0

Toán tử OR (hoặc):

Cú pháp: ***expr1*** || ***expr2***

Ý nghĩa: Trả lại giá trị đúng chỉ khi một trong hai toán hạng của nó có giá trị đúng

1 || 1 = 1 1 || 0 = 1

0 || 1 = 1 0 || 0 = 0

Với các toán tử và && toán tử hoặc ||, các toán hạng được tính từ trái qua phải. Việc tính toán sẽ dừng lại ngay khi giá trị đúng hay sai của biểu thức được xác định, chẳng hạn với: expr1 && expr2 và

expr1 || expr2 thì expr2 sẽ không được tính nữa nếu: Trong biểu thức expr1 && expr2, expr1 có giá trị sai, hoặc trong biểu thức expr1 || expr2, expr1 có giá trị đúng.

Toán tử phủ định NOT

Cú pháp: ***! expr***

Ý nghĩa: Trả lại giá trị đúng nếu toán hạng của nó có giá trị sai và ngược lại.

Toán tử gán

Cú pháp: ***<Biến> = <biểu thức>***

Ví dụ:

```
x = a; // cất giữ giá trị a vào vùng nhớ của biến x
```

Ý nghĩa: Cất giữ một giá trị mới vào vùng bộ nhớ liên kết với toán hạng bên trái.

Kết quả của toán tử gán là giá trị của biểu thức ở vế phải của toán tử. Kiểu dữ liệu của kết quả là kiểu của toán hạng bên trái.

Thường dùng câu lệnh gán tích lũy:

Cú pháp: ***<biến> op = <biểu thức>;***

trong đó, op là một phép toán hai ngôi nào đó như +, -, *, /, % hoặc các phép thao tác bit đối với toán hạng nguyên & (AND), | (OR), ^(XOR), << (dịch trái), >> (dịch phải).

Ví dụ:

```
a+ = b-c; tương đương với a= a+b-c;
a/=b+c; tương đương với a=a/(b+c);
a&=b tương đương với a=a&b
a+ = (b = 5); /* gán giá trị 5 vào biến b rồi cộng 5 vào a*/
a+ = (b+ =2); /* nếu a = 5 và b = 6 thì sau khi thi hành câu
lệnh trên ta được b = 8 và a = 13*/
```

Toán tử gán được thực hiện từ phải sang trái vì vậy với hai ví dụ trên có thể bỏ dấu ngoặc.

a = b = c = 0 /* biểu thức này hoàn toàn đúng */.

Toán tử phẩy

Đây là một toán tử đặc biệt có ký hiệu là ",". Biểu thức phẩy gồm một cặp biểu thức cách nhau bằng dấu phẩy và được tính toán từ biểu thức bên trái sau đó đến biểu thức bên phải. Kết quả của toàn bộ biểu thức phẩy sẽ là kết quả của biểu thức bên phải. Toán tử phẩy là đặc thù riêng của C/C++ mà trong ngôn ngữ Pascal không có.

Ví dụ:

```
int x, y, z;
x = (y = 2, z = y*y + 3);
```

Kết quả sẽ là y = 2; z =7 và x = 7.

Chuyển đổi kiểu dữ liệu

- Khi hai toán hạng trong một phép toán khác kiểu dữ liệu thì kiểu dữ liệu thấp được nâng thành kiểu dữ liệu cao trước khi tính toán.

Ví dụ: nếu f có kiểu *float* và i có kiểu *int* thì trong biểu thức *f+i,* i sẽ tạm thời được chuyển sang kiểu *float* để thực hiện phép cộng.

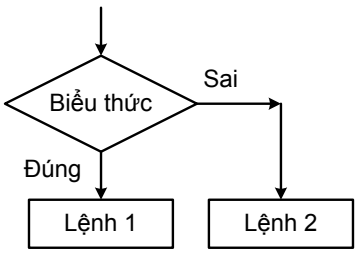
- Nếu *f* có kiểu *float, i1 và i2* có kiểu *int* và có giá trị *10* và *3* thì biểu thức *f =i1/i2* sẽ có giá trị 3.0. Để thu được kết quả chính xác trong trường hợp này cần sử dụng *phép ép kiểu*: *f = (float) i1/i2*.

2.4. LỆNH IF, ELSE

Thông thường sử dụng các phép toán quan hệ với lệnh if. Biểu thức chứa phép toán quan hệ được xem như biểu thức quyết định. Cú pháp sử dụng như sau:

if(<biểu thức điều kiện>) <Câu lệnh 1>; [else <Câu lệnh 2>;]

Khi thực hiện câu lệnh này, máy tính sẽ kiểm tra biểu thức điều kiện. Nếu <biểu thức> cho giá trị đúng thì các lệnh 1 được thực hiện, ngược lại các lệnh 2 được thực hiện. Lưu đồ:



Ví dụ 1: Tìm số lớn nhất trong 2 số nhập vào từ bàn phím

Phân tích thuật toán:

```
Bắt đầu:
    int a, b, max; /* Khai báo 3 biến*/
    Nhập giá trị a, b;
    max = a;
    Nếu (b>a)
    max = b;
    In kết quả a, b, max;
    Lệnh dừng;
Kết thúc
```

Chương trình như sau:

```
#include<stdio.h>
#include<conio.h>
main()
{
clrscr();
float a,b,max;
printf("\n Nhap so thu nhat=");
scanf("%f",&a);
```

```
printf("\n Nhap so thu hai=");
scanf("%f",&b);
max=a;
if (b>a)
max=b;
printf("\n So thu nhat= %f",a);
printf("\n So thu hai = %f",b);
printf("\n So lon nhat= %f",max);
getch();
}
```

Ví dụ 2: Lập chương trình tính giá trị hàm số:

$$y = \begin{cases} x + 2 \text{ nếu } x \le -1 \\ x^2 + 1 \text{ nếu } -1 < x < 2 \\ 2x^3 - 4 \text{ nếu } x \ge 2 \end{cases}$$

Tại các điểm nằm trong đoạn [a,b] được chia thành n đoạn con đều nhau.

Chương trình như sau:

```
# include <stdio.h>
# include <conio.h>
int i,n;
float a,b,h,x,y;
void main()
{
clrscr();
printf("Nhap a, b:"); scanf("%f%f", &a, &b);
printf("Nhap so diem:"); scanf("%d",&n);
h=(b-a)/n;
```

```
printf("\n BANG GIA TRI HAM SO \n*");
for (i=0; i<=n; i++)
{
x=a + i*h;
if (x<=-1) y=x+2; else
if (x<2) y=x*x+1;
else y=2*x*x*x+4;
printf("\n%5.2f%8.2f",x,y);
}
printf("\n");
getch();
}
```

Lệnh else được sử dụng với lệnh if như sau:

```
if(điều kiện)
        {một lệnh hay một khối lệnh};
else
        {một lệnh hay một khối lệnh }
```

Nếu điều kiện được thỏa mãn thì khối lệnh sau if được thực hiện, nếu không thì khối lệnh sau else được thực hiện.

Có thể lặp cặp lệnh if-else bao nhiêu lần tùy ý, ví dụ:

```
if (đk1)
        {....}
else if (đk2)
        {....}
else
        {...}
```

2.5. CÁC PHÉP TOÁN KHÁC

Toán tử tăng và giảm

Toán tử tăng (++) và giảm (--) đưa ra cách viết gọn và thuận tiện biểu thị sự tăng hay giảm đi 1 từ giá trị của một biến. Mỗi toán tử đều có dạng tiền tố và hậu tố. Dạng tiền tố và hậu tố cho lại các giá trị khác nhau đối với biến sử dụng.

Có hai dạng:

1. ++ <biến> và <biến>++

2. - - <biến> và <biến> - -

Để dễ hiểu chúng ta xét ví dụ sau:

```
i =3;
s = ++ i;
```

Kết quả s là 4, vì lệnh ++i sẽ thực hiện phép tăng biến trước sau đó mới thực hiện những phép toán khác.

Còn nếu viết

```
i =3;
s = i++;
```

kết quả của s là 3 vì lệnh i ++ không được thực hiện trước lệnh gán.

Như vậy, ++<biến> và -- <biến> sẽ thực hiện phép toán tăng hoặc giảm giá trị của biến trước rồi mới thực hiện những phép toán khác, còn <biến> ++ và <biến> -- thì ngược lại như đã xét trong ví dụ trên.

Ví dụ: i++; ++i; j--; --j;

Toán tử sizeof

Toán tử sizeof cho lại kích cỡ tính theo byte của một biểu thức hay một bộ xác định kiểu.

Ví dụ: sizeof(int) cho lại giá trị 2.

Toán tử ba ngôi

Toán tử điều kiện hay toán tử ba ngôi ký hiệu là ? gồm ba thành phần:

<Điều kiện> ? <biểu thức 1>: <biểu thức 2>

Kết quả của biểu thức trên bằng biểu thức 1 nếu điều kiện có kết quả khác 0, ngược lại kết quả của biểu thức này bằng biểu thức 2.

Ví dụ:

```
a = 1>5 ? 15:  -15  /* biểu thức a cho kết quả bằng -1 */
char  c, t;
t = 'E';
c = (t >= 'A') &&  (t <= 'Z') ? t:'A';
/* biểu thức c cho kết quả bằng ký tự E */
```

Biểu thức b = a?a: -a; cho kết quả b chứa giá trị tuyệt đối của a. Biểu thức này tương đương với: if(a > 0) b = a; else b = -a;

Toán tử thao tác bit

Toán tử này cho phép xử lý từng bit của một số nguyên không dùng cho kiểu *float* và *double*. Các tác vụ bao gồm AND(&), OR(|), XOR (^), dịch trái (<<), dịch phải (>>), lấy phần bù (~) theo bit.

Toán tử thao tác bit coi các toán hạng của nó như là một tập hợp các bit. Mỗi bit có thể chứa giá trị 1 hoặc 0. Toán tử thao tác bit cho phép người lập trình kiểm thử hay đặt giá trị cho từng bit hay từng nhóm bit.

Toán tử	Chức năng	Cách dùng
~	phủ định (NOT)	~Expr
<<	dịch chuyển trái	Expr1 << Expr2
>>	dịch chuyển phải	Expr1 >> Expr2
&	và bit (AND)	Expr1 & Expr2
^	hoặc bit (XOR)	Expr1 ^ Expr2
|	hoặc bit (OR)	Expr1 ^ Expr2

trong đó, Exp, Exp1, Exp2 là các biểu thức

Toán hạng của toán tử bit phải có kiểu nguyên.

Toán tử phủ định bit (~):

Đảo lại giá trị từng bit của toán hạng, bit 0 được đặt thành 1 và bit 1 đặt thành 0.

Ví dụ: char c = 0x0f; nếu c1 = ~c; thì c1 có giá trị 0xf0;

Các phép toán dịch chuyển bit

Toán tử dịch chuyển trái (<<) và toán tử dịch chuyển phải (>>) sẽ chuyển các bit của toán hạng bên trái đi một số vị trí hoặc sang trái hoặc sang phải.

Các bit vượt quá trong toán hạng sẽ bị bỏ qua. Nếu toán hạng là không dấu thì toán tử dịch chuyển trái sẽ thêm các bit có giá trị 0 vào bên phải.

Đối với kiểu không dấu thì:

a << n = a* (2^n)

a << = a/ (2^n)

Toán tử dịch chuyển phải sẽ thêm các bit có giá trị 0 vào bên trái

Nếu toán hạng là có dấu thì nó có thể thêm bản sao của bit dấu hoặc thêm các bit giá trị 0 (tùy thuộc vào máy)

Toán tử và bit (&)

Toán tử và bit là toán tử hai ngôi. Với mỗi vị trí bit thì kết quả trả lại là bit 1, nếu cả hai toán hạng đều chứa bit 1, ngược lại kết quả là bit 0.

Toán tử XOR bit (^)

Toán tử XOR bit là toán tử hai ngôi. Với mỗi vị trí bit thì kết quả cho lại là bit 1 nếu một trong hai chứa bit 1, tức không cần đồng thời cả hai có chứa bit 1, ngược lại kết quả là bit 0.

Toán tử hoặc bit (|)

Toán tử hoặc bit là toán tử hai ngôi. Với mỗi vị trí bit thì kết quả trả lại là bit 1 nếu một trong hai toán hạng có chứa bit 1, ngược lại kết quả là bit 0.

2.6. THỨ TỰ ƯU TIÊN THỰC HIỆN CÁC PHÉP TÍNH

Thứ tự ưu tiên của các toán tử là trình tự thực hiện chúng trong một biểu thức hợp thành. Đây là điều quan trọng cần phải nhớ để tránh các lỗi chương trình thông thường.

Bảng sau cho ta một tập hợp đầy đủ các toán tử của C++ theo trật tự ưu tiên tính toán, mức ưu tiên càng cao thì phép toán càng có quyền ưu tiên thực hiện trước.

R: thứ tự kết hợp từ phải qua trái

L: thứ tự kết hợp từ trái qua phải

Ví dụ 17R có nghĩa là số ưu tiên mức 17, kết hợp từ phải qua trái

Mức	Toán tử	Chức năng
17R	::	Phạm vi toàn cục (một ngôi)
17L	::	Phạm vi lớp (hai ngôi)
16L	->, .	Chọn thành phần của cấu trúc, lớp
16L	[ ]	Chỉ số mảng
16L	( )	Gọi hàm
16L	( )	Xây dựng kiểu
16L	sizeof	Tính kích cỡ theo byte
15R	++, --	Phép tăng, giảm
15R	~	Phép thao tác bit NOT
15R	!	Phép toán logic NOT
15R	+, -	Cộng trừ một ngôi
15R	*, &	Phép lấy giá trị, lấy địa chỉ
15R	( )	Chuyển đổi kiểu
15R	new, delete	Quản lý bộ nhớ tự do
14L	->*, .*	Bộ chọn con trỏ thành phần
13L	*, /, %	Các toán tử nhân, chia
12L	+, -	Các toán tử số học
11L	<<, >>	Dịch chuyển bit
10L	<, <=, >, >=	Toán tử quan hệ

9L	==, !=	So sánh bằng, không bằng
8L	&	Phép toán và bit (AND)
7L	^	Phép toán hoặc loại bit (XOR)
6L	|	Phép toán hoặc bit (OR)
5L	&&	Phép logic và (AND)
4L	| |	Phép logic hoặc (OR)
3L	?:	Phép if số học
2R 2R	=, *=, /=, %=, +=, -= <<=, >>=, &=, | =, ^=	Các toán tử gán
1L	,	Toán tử phảy

Ví dụ 1:

Cho mảng một chiều kiểu số thực như sau:

a[] ={3.5, 0, 23.3, 0, 4.7, -2, 10, 30, 40}

Viết chương trình để xác định:

- Phần tử âm đầu tiên là phần tử thứ mấy
- Số phần tử dương đầu tiên
- Tổng các phần tử dương đầu tiên
- Max của phần tử dương đầu tiên

Chương trình như sau:

```
float a[]={3.5, 0, 23.3, 0, 4.7, -2, 10, 30, 40};
main()
{
 float s=0,max=0;
 int k=0,i;
 clrscr();
 for (i=0;i<9;++i)
{
if(a[i] <0)
```

```
break;
if(a[i] >0)
{
s+=a[i];
k++;
if (max < a[i])
max=a[i];
}
}
printf("\nPhan tu am dau tien la phan tu thu %d\co tri = %8.2f",i+1,a[i]);
printf("\nSo phan tu duong dau tien la: %d",k);
printf("\nTong cac phan tu duong dau tien la: %8.2f",s);
printf("\nMax cua phan tu duong dau tien la: %8.2f",max);
getch();
}
```

Ví dụ 2:

Viết chương trình tính căn bậc hai của một số nhập vào từ bàn phím.

```
/* Chuong trinh tinh can bac 2 */
main()
{
    double a,xn,c,fabs();
    int i;
    tt:printf("\nNhap vao mot so a= ");
    scanf("%lf",&a);
    if(a<0)
    {
    printf("\nKhong the tinh can bac 2 cua so am\n");
    printf("\nBam phim bat ky de nhap lai");
    getch();
```

```
        goto tt;
        }
        if(a==0)
        {
        xn=0;
        goto kq;
        }
        xn=a;
        do
        {
              c=xn;
              xn=(xn*xn+a)/(2*xn);
          }
        while(fabs((xn-c)/c) > 1e-5);
        kq:printf("\nNeu a= %8.2f,thi can bac hai cua a=%8.2f\ ",a,xn);
        printf("\n\n Tiep tuc nua khong ?(tiep=1,khong=0)");
        scanf("%d",&i);
        if(i==1) goto tt;
}
```

Bài tập chương 2

1. Ví dụ trong chương trình có khai báo như sau:

   unsigned int ui1=3, ui2=7;

   Hãy cho biết kết quả của các phép toán sau:

   (a) ui1 & ui2

   (b) ui1 && ui2

   (c) ui1 | ui2

   (d) ui1 || ui2

   (e) ui1 << 2

   (f) ui1 ^ ui2

   (g) ui1 >> 3

2. Theo thứ tự ưu tiên trong C++ thì phép toán sau trả lại kết quả bao nhiêu: 6 + 3 * 4 / 2 + 2

3. Viết chương trình tính diện tích hình tròn có bán kính r.

4. Viết chương trình hiển thị số dư của phép chia hai số nguyên nhập từ bàn phím.

5. Viết chương trình chuyển đổi số nguyên sang số nhị phân.

6. Các biểu thức sau đây cho giá trị True hay False:

   a) 'C' < 'c'

   b) '0' > '0'

   c) '?' > ')'

7. Viết chương trình cho biết số nhập từ bàn phím là chẵn hay lẻ.

8. Cho trước các lệnh: int i = 12, j = 10, k = 5;

Cho biết kết quả (True hay False) của các lệnh sau:

a) i&&j

b) 12-i || k

c) j!=k && i!=k

9. Viết chương trình chuyển đổi chữ hoa thành chữ thường. Các ký tự được nhập từ bàn phím, nếu ký tự là chữ thường thì không chuyển đổi.

10. Ông B làm việc 50 giờ cho một cửa hàng. Ông ta được nhận 8 nghìn/giờ cho 40 giờ đầu. Năm giờ tiếp theo ông được trả gấp rưỡi số tiền bình thường nhận được cho một giờ. Nhận được gấp hai số tiền trong một giờ cho các giờ làm việc còn lại. Ông B phải trả 10% thuế thu nhập. Hãy viết chương trình tính tổng thu nhập, số thuế phải nộp của ông B.

11. Lập chương trình giải hệ:

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

Các hệ số a, b, c, d, e, f nhập vào từ bàn phím. Yêu cầu xét tất cả các trường hợp có thể.

12. Viết chương trình nhập ba cạnh của một tam giác. Hãy cho biết tam giác đó là tam giác gì?

13. Cho ba số thực, hãy thay tất cả các số âm bằng giá trị tuyệt đối của nó. Ví dụ nhập a = 2; b = -5; c = -8 thì kết quả là: a = 2; b = 5; c = 8.

14. Viết chương trình giải phương trình bậc hai.

15. Viết chương trình nhập ngày, tháng, năm. Hãy cho biết tháng đó có bao nhiêu ngày và ngày, tháng, năm tiếp theo.

Chương 3

SỬ DỤNG CÁC LỆNH CỦA C/C++

3.1. CÁC VÒNG LẶP WHILE, DO-WHILE, FOR

Trong lập trình giải các bài toán thực tế, chúng ta luôn gặp phải cấu trúc lặp để thực hiện một công việc nào đó lặp đi lặp lại theo một quy luật nào đó. Ví dụ: trong bài toán quản lý cán bộ, để nhập dữ liệu vào danh sách gồm n người thì công việc lặp lại chính là: nhập thông tin của một cán bộ (gồm họ tên, ngày sinh, giới tính, chức vụ, lương...) n lần vào máy tính.

Trong nhiều trường hợp, số lần lặp có thể không biết trước mà chỉ biết điều kiện để dừng lặp lại. Ví dụ bài toán nhập thông tin về một cán bộ nào đó trong một cơ quan cho đến khi nhấn phím ESCAPE thì dừng nhập.

Trong các ngôn ngữ lập trình, cấu trúc lặp cho phép thực hiện lặp đi, lặp lại nhiều lần một câu lệnh hay một khối câu lệnh nào đó (chính là *khối công việc* trong các bài toán thực tế).

Trong C/C++ một dãy các lệnh đặt trong cặp dấu { và } được gọi là một khối lệnh và có thể coi khối lệnh này như là một câu lệnh riêng lẻ (tương tự như lệnh ghép trong ngôn ngữ Pascal).

3.1.1. Câu lệnh while

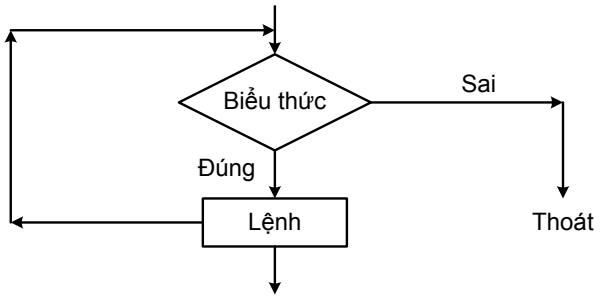
Đây là câu lệnh lặp số lần không xác định có kiểm tra điều kiện trước khi thực hiện vòng lặp.

Cú pháp:

while(<*biểu thức điều kiện*>)

<câu lệnh>;

Sơ đồ khối của cấu trúc lặp như sau:



Trình tự thực hiện:

1. Chương trình sẽ tính toán giá trị <biểu thức điều kiện>

2. Nếu biểu thức điều kiện cho giá trị đúng thì chương trình sẽ thực hiện <câu lệnh>, nếu biểu thức điều kiện cho lại giá trị sai thì kết thúc câu lệnh while.

3. Quay về bước 1

Nếu ngay lần tính đầu tiên giá trị biểu thức điều kiện cho lại là sai thì phần câu lệnh sẽ không bao giờ được thực hiện.

Các vòng lặp while có thể lồng nhau theo cú pháp sau:

```
while(biểu thức điều kiện 1)
{
câu lệnh...
while (biểu thức điều kiện 2)
{
  câu lệnh...
}
câu lệnh...
}
```

Ví dụ 1: Hãy viết chương trình để in ra 10 hàng chữ cho biết số thứ tự của hàng:

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

```
#include<conio.h>
int i=0;
main()
{
clrscr();
printf("\n Theo chieu tang");
while (i<10)
{
printf("\n Hang thu %2d= %2d",i+1,i);
++ i;
}
getch();
}
```

Ví dụ 2: Viết chương trình tạo mật khẩu, chỉ khi nào gõ đúng mật khẩu thì mới thoát khỏi vòng lặp để thực hiện câu lệnh khác. Trong ví dụ này mật khẩu là m.

```
#include<stdio.h>
#include<conio.h>
main()
{
char mk;
clrscr();
printf("\n cho biet mat khau:");
scanf("%s",&mk);
while (mk!='m')
{
printf("\n Cho biet mat khau:");
scanf("%s",&mk);
}
```

```
printf("Ban da go dung mat khau");
getch();
}
```

Ví dụ 3: Viết chương trình tính $1^2 + 2^2 + ... + N^3$.

```
# include <stdio.h>
# include <conio.h>
void main()
{
int s = 0; int i = 1;  int n;
printf("Nhap n:= ");
scanf("%d", &n);
s = 0;
while (i <= n)
{
s = s + i* i;
i = i + 1;
}
printf(" Tong s = %d", s);
getch();
}
```

3.1.2. Câu lệnh do..while

Là câu lệnh kiểm tra điều kiện sau (kiểm tra điều kiện sau khi thực hiện vòng lặp). Cấu trúc này thực hiện khối lệnh cho đến khi biểu thức có giá trị bằng 0.

Cú pháp:

do

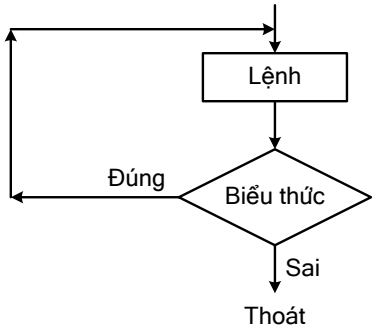
{

<*câu lệnh*>;

}

while(<*biểu thức điều kiện*>)

Sơ đồ khối cấu trúc lệnh như sau:



Thực hiện:

1. Chương trình sẽ thực hiện khối câu lệnh sau lệnh do,

2. Kiểm tra giá trị biểu thức điều kiện của lệnh while, nếu giá trị biểu thức điều kiện là đúng, chương trình sẽ thực hiện tiếp bước 1, nếu giá trị biểu thức điều kiện là sai, nó sẽ kết thúc lệnh do-while.

Chú ý:

Khác với lệnh while, lệnh do-while bao giờ cũng cho phép thực hiện khối lệnh sau ít nhất một lần, sau đó nó mới kiểm tra giá trị biểu thức điều kiện để quyết định có thực hiện khối lệnh đó vòng tiếp theo hay không (kiểm tra điều kiện sau). Còn lệnh while thì kiểm tra giá trị biểu thức điều kiện ngay từ ban đầu trước khi quyết định cho thực hiện khối câu lệnh (kiểm tra điều kiện trước).

Ví dụ 1: Viết chương trình tính trung bình cộng của n số nguyên nhập vào từ bàn phím. Khi nào bấm số 0 thì kết thúc nhập và tính giá trị trung bình.

```
# include <stdio.h>
# include<conio.h>
main()
{ int so, tong=0, n=0;
float tb;
```

```
clrscr();
do
{
printf("\n Nhap vao 1 so (so 0 de dung)");
scanf("%d", &so);
tong+=so;
n+=1;
}
while (so!=0);
n-=1;
tb=tong/n;
printf("\n Da nhap vao: %d so ",n);
printf("\n Trung binh cong= %f ", tb);
getch();
}
```

Ví dụ 2: Viết chương trình tính tổng bình phương của n số nguyên theo công thức: $S_n = 1^2 + 2^2 + ... + n^2$

```
# include<stdio.h>
# include<conio.h>
main()
{
int s=0, i=1, n;
clrscr();
printf("\n Tinh tong binh phuong cua bao nhieu so nguyen:");
scanf("%d",&n);
do
      {
      s+ = (i * i);
      ++ i;
      }
```

```
while (i<= n);
printf("\n Tong binh phuong cua: %d so nguyen dau tien lap: %d",n,s);
getch();
}
```

Ví dụ 3: Lập chương trình tính gần đúng giá trị của hàm sin(x) theo công thức sau:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ...(-1)^i \frac{x^{2i+1}}{(2i+1)!} + ...$$

với sai số không vượt quá ε cho trước.

Đây thực chất là một bài toán tính tổng các số hạng, trong đó mỗi số hạng thứ i + 1 được tính qua số hạng thứ i. Điều kiện để thoát khỏi vòng lặp chính là khi giá trị số hạng thứ n nào đó nhỏ hơn hoặc bằng ε. Để giải bài toán chúng ta cần tìm được công thức biểu diễn số hạng thứ i + 1 theo số hạng thứ i bằng công thức truy hồi. Dễ dàng tìm được: $a_i = -a_{i-1} \frac{x^2}{(2i-1)(2i-2)}$ với i: 2.. n.

Chương trình như sau:

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
long i;
float x,y,a;
void main()
{
clrscr();
printf("Nhap gia tri x:");
scanf("%f",&x);
fflush(stdin);
y=a=x;
```

```
i=1;
do
{
i+=2;
a*=-x*x/(2*i-1)/(2*i-2);
y+=a;
}
while (fabs(a) >0.0001);
printf("\n Ham sin la: %0.4f\n",y);
printf("Sai so la: %0.4f\n",y-sin(x));
getch();
}
```

Ví dụ 4: Lập chương trình giải phương trình x = sin(x) + 0,1 trên đoạn [0.3, 0.9] bằng phương pháp lặp đơn với độ chính xác ε cho trước.

Phân tích thuật toán: Phương pháp lặp đơn giải phương trình tổng quát x = φ(x) trên đoạn [a, b] với độ chính xác ε như sau:

Bắt đầu, chúng ta chọn nghiệm ban đầu x_0 trong đoạn [a, b], sau đó tính: $x_1 = \varphi(x_0)$, $x_2 = \varphi(x_1)$,... $x_{n+1} = \varphi(x_n)$

Quá trình lặp dừng lại khi: $|x_{n+1} - x_n| < \varepsilon$

Chương trình như sau:

```
//Chuong trinh giai phuong trinh bang phuong phap lap don
#include <stdio.h>
#include <math.h>
#include <conio.h>
float x1, x2, saiso;
float Phi(float x)
{
return sin(x)+0.1;
}
```

```
void main()
{
clrscr();
printf("Cho biet sai so:"); scanf("%f",&saiso);
printf("Cho xap xi dau tien:"); scanf("%f",&x2);
do
{
x1=x2;
x2 = Phi(x1);
}
while (fabs(x2-x1)>saiso);
printf("\n Nghiem phuong trinh la %0.5f",x2);
getch();
}
```

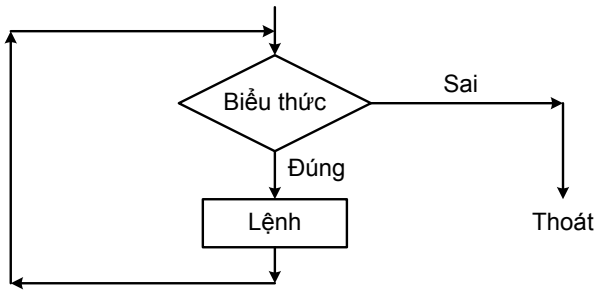
3.1.3. Câu lệnh for

Đây là câu lệnh lặp có kiểm tra điều kiện trước, nên sử dụng khi lập trình viên biết trước được số lần lặp.

Cú pháp:

for (<câu lệnh khởi đầu>;<biểu thức 1>;<biểu thức 2>)
{
    <câu lệnh>;
}

Lưu đồ:



Thực hiện:

Bước 1: Thực hiện câu lệnh khởi đầu, câu lệnh khởi đầu có thể hoặc là một khai báo biến hay là một biểu thức. Nói chung nó được dùng để khởi đầu hay gán giá trị cho một hay một tập các biến hoặc nó cũng có thể là câu lệnh rỗng.

Phần câu lệnh khởi đầu có thể có các dạng sau:

```
for (int i = 0; …
for (; … // Câu lệnh khởi đầu rỗng
for (i = 0; …
for (int lo = 0, hi=MAX, mid=MAX/2; …
for (char *ptr=getStr (); …
```

Bước 2: Tính giá trị của biểu thức 1. Biểu thức 1 để điều khiển chu trình, nếu biểu thức 1 còn đúng thì việc lặp còn tiếp tục và phần câu lệnh còn được thực hiện, nếu giá trị biểu thức 1 cho lại là sai thì sẽ kết thúc câu lệnh for. Cũng như trong câu lệnh while, nếu ở lần tính đầu tiên, giá trị của biểu thức 1 đã cho giá trị sai thì phần câu lệnh sẽ không bao giờ được thực hiện.

Biểu thức 1 có thể có các dạng sau:

```
(…; index < arrSize; …)
(…; ptr; …)
(…; *st1++ = *st2++; …)
(…; ch = getNextChar (); …)
```

Bước 3: Tính giá trị biểu thức 2. Biểu thức 2 được tính sau mỗi lần lặp chu trình. Nói chung là nó được dùng để thay đổi giá trị các biến đã được khởi đầu trong phần câu lệnh khởi đầu. Nếu việc tính giá trị lần đầu của biểu thức 1 cho kết quả sai thì biểu thức 2 sẽ không bao giờ được tính. Biểu thức 2 có thể có các dạng sau:

```
(…;;i++)
```

```
(…;…;ptr = ptr->next)
(…;…;i++,j--,++cnt)
(…;…;) //chương trình không điều kiện
```

Các vòng lặp for có thể lồng nhau theo cú pháp sau:

```
for(câu lệnh khởi đầu 1;biểu thức 1;biểu thức 2)
{
câu lệnh...
for (câu lệnh khởi đầu 2;biểu thức 3;biểu thức 4)
{
    câu lệnh...
}
câu lệnh...
}
```

Chú ý: Lệnh for sẽ giống như lệnh while trong trường hợp không có giá trị ban đầu và không tính giá trị tiếp theo cho biến chạy:

```
for (; điều kiện giá trị của run;)
{
câu lệnh;
}
```

Ví dụ 1: Viết chương trình tính tổng bình phương của n số nguyên theo công thức: $S_n = 1^2 + 2^2 + ... + n^2$

```
# include<stdio.h>
# include<conio.h>
main()
{
int s=0, i, n;
clrscr();
```

```
printf("\n Tinh tong binh phuong cua bao nhieu so nguyen:");
scanf("%d",&n);
for (i=1; i<=n; ++i)
      s+ = (i * i);
printf("\n Tong binh phuong cua: %d so nguyen dau tien lap: %d",n,s);
getch();
}
```

Ví dụ 2: Viết chương trình tính tổng các số lẻ nguyên dương nhỏ hơn n.

```
# include<stdio.h>
# include<conio.h>
void main()
{
int i, n, s;
printf("nhap n = ");
scanf("%d", &n);
for (i = 1, s = 0; i <= n ; i = i +  2)
s = s + i;
printf("\n Tong N so le nguye duong dau tien la: %d",s);
getch();
}
```

3.2. CÁC LỆNH NHẢY

Các câu lệnh nhảy gồm *break*, *continue*, *go to* và *switch*. Trong đó các lệnh *break, continue, go to* là các lệnh nhảy trong và nhảy ra khỏi vòng lặp, còn lệnh *switch* là lệnh nhảy lựa chọn.

Sau đây ta sẽ xét từng câu lệnh.

3.2.1. Câu lệnh switch

Câu lệnh *switch* cũng có ý nghĩa như tập các câu lệnh if...else nhưng nó xác định rõ hơn giá trị cụ thể của biểu thức điều kiện. Nếu

như câu lệnh *if* chỉ cho chọn một trong hai nhánh tính toán, tùy theo giá trị của một biểu thức nguyên.

Cú pháp:

switch (***Expr***)

{

case <*constant expr1*>:

câu lệnh 1;

break;

case <*constant expr2*>:

câu lệnh 2;

break;

case <*constant exprn*>:

câu lệnh n;

break;

default:

câu lệnh m;

break*;*

}

Expr: biểu thức kiểu nguyên

<*constant expr*> phải có giá trị là một hằng số *integer* và phải có giá trị duy nhất.

Thực hiện:

Bước 1: Chương trình tính toán giá trị cho biểu thức ***Expr***

Bước 2: So sánh giá trị trả về của biểu thức ***Expr*** và giá trị của <*constant expr1*>, <*constant expr2*>,..., <*constant exprn*>; nếu biểu thức **Expr** có giá trị bằng một trong các giá trị <*constant expr1*>, <*constant expr2*>,..., <*constant exprn*> thì các khối lệnh tương ứng sẽ được thực hiện và sau đó câu lệnh switch sẽ được kết thúc bằng câu

lệnh *break*. Nếu giá trị trả về của biểu thức ***Expr*** không nằm trong các giá trị của <*constant expr1*>, <*constant expr2*>,..., <*constant exprn*> thì khối lệnh trong **default** sẽ được thực hiện.

Khi chương trình đã thực hiện xong câu lệnh của case <*constant expri*> nào đó, nó sẽ thực hiện luôn các lệnh thuộc **case** bên dưới nó mà không xét lại điều kiện do các <*constant expri*> được xem như các nhãn. Vì vậy để chương trình thoát khỏi lệnh ***swithch*** sau khi thực hiện xong một chương trình ta dùng lệnh ***break***.

Ví dụ 1: Viết chương trình nhập vào một năm, máy sẽ thông báo năm đó là năm nhuận hay không?

```
# include<stdio.h>
# include<conio.h>
main()
{
int nam;
clrscr();
printf("\n Cho biet nam can xem?");
scanf("%d",&nam);
switch (nam % 4)
{
case 0: printf("\n Nam %4d",nam);
        printf("\n La nam nhuan");
case 1: printf("\n Nam ngoai%4d",nam-1);
        printf("\n Moi la nam nhuan");
        break;
case 2: printf("\n Nam kia %4d",nam-2);
        printf("\n Moi la nam nhuan");
        break;
default: printf("\n Sang nam %4d", nam+1);
         printf("\n Moi la nam nhuan");
         break; } }
```

Ví dụ 2: Viết chương trình thực hiện một trong 4 phép toán +, -, *, / bằng cách nhấn các phím mũi tên ←↑→↓ trên bàn phím.

```
# include<stdio.h>
# include<conio.h>
int c;
void main()
{
int a, b;
clrscr();
printf("Nhap hai so nguyen a va b:");
scanf("%d%d", &a, &b);
printf("\n Nhan phim mui ten chon phep toan thuc hien");
c = getch();
if (!c)
switch (getch( ) )
{
case 75: printf("\n Tong a va b la: %d", a + b);
break;
case 77: printf("\n Hieu  a va b la: %d", a - b);
break;
case 72: printf("\n Tich  a va b la: %d", a * b);
break;
case 80: printf("\n Hieu  a va b la: %d", a / b);
}
getch ();
}
```

3.2.2. Câu lệnh break

Câu lệnh break cho phép thoát khỏi vòng lặp của các lệnh while, do, for hay switch... case (như trong ví dụ trên). Lệnh break tương tự như lệnh EXIT trong FOXPRO. Khi có nhiều vòng lặp lồng nhau, lệnh break có tác dụng ra khỏi vòng lặp (hoặc lệnh **switch**) bên trong nhất

(gần nhất) chứa nó mà không cần kiểm tra điều kiện kết thúc vòng lặp. Nói một cách khác, máy sẽ bỏ qua các câu lệnh còn lại để thoát ra khỏi vòng lặp.

3.2.3. Câu lệnh continue

Câu lệnh continue làm cho lần lặp hiện tại của chu trình ***while***, ***do*** hay ***for*** gần nhất bao quanh nó được kết thúc, trái với lệnh ***break***

Với lệnh ***while*** hay ***do*** việc thực hiện lần lặp tiếp theo tùy thuộc vào giá trị biểu thức cho lại, còn với lệnh ***for*** thì việc thực hiện lần lặp tiếp theo tùy thuộc vào giá trị biểu thức 2.

Chú ý: Khác với câu lệnh ***break***, câu lệnh ***continue*** chỉ làm kết thúc lần lặp hiện tại và không áp dụng cho ***switch.***

Ví dụ: Viết chương trình in ra tất cả các số chẵn nhỏ hơn 101 trừ các số 6, 8, 94.

```
# include <stdio.h>
# include <conio.h>
void main()
{
int i;
for (i = 0; i < 101; i + = 2)
{
if (( i==6) || (i== 8) || (i== 94)
continue;
printf("%5d",i);
}
getch();
}
```

3.2.4. Câu lệnh goto

Câu lệnh *goto* cũng có khả năng giống như các câu lệnh *break* hay *continue* để nhảy ra khỏi một chu trình hay câu lệnh *switch*, nhưng khác các câu lệnh trên là *goto* có thể chuyển điều khiển đến một câu

lệnh bất kỳ tùy thuộc vào nhãn đích, còn *break* chỉ có thể chuyển điều khiển đến lệnh tiếp ngay sau lệnh lặp *while, do*, *for* hay *switch*.

Cú pháp:

goto ***nhãn***;

Các quy định sử dụng nhãn:

Nhãn là tên do người dùng xác định, một nhãn có thể chỉ được dùng làm đích cho một *goto* và phải được kết thúc bằng dấu hai chấm. Nhãn không được đứng trước dấu ngoặc nhọn đóng (có thể dùng câu lệnh rỗng đi sau nhãn để xử lý cho ràng buộc này).

Câu lệnh *goto* không được phép nhảy qua định nghĩa của một biên có bộ khởi đầu tường minh hay không tường minh trừ khi định nghĩa này xuất hiện trong một khối và toàn bộ khối bị nhảy qua.

Hiện nay, với những người lập trình có kinh nghiệm, người ta không dùng goto trong chương trình, vì như vậy chương trình có thể nhảy đến nhiều nơi nên rất khó đọc, khó hiểu và khó kiểm soát lỗi.

Ví dụ:

```
main()
{
    int i,k;
    tt: printf("\n Nhap vao mo ta so nguyen");
    scanf("%d",&i);
    switch(i);
    {
        case 0: printf ("\n vua nhap so 0");
        break;
        case 1: printf ("\n vua nhap so 1");
        break;
        case 2: printf ("\n vua nhap so 2");
        break;
      default: printf ("\n vua nhap mo ta so nguyen");
    }
```

```
printf("\n Tiep tuc nua hay khong?");
printf("\n Bam so (1) de tiep tuc, so (2) de dung");
scanf("%d",&k);
      if(i= =1)
      goto tt;
}
```

3.3. HÀM VÀ CHƯƠNG TRÌNH CON

3.3.1. Khái niệm và quy tắc xây dựng hàm

Hàm dùng để chia một chương trình lớn thành nhiều phần nhỏ giúp người lập trình dễ đọc, dễ hiểu, dễ viết và dễ sửa khi có lỗi.

Sử dụng hàm rất có ý nghĩa trong lập trình vì việc sử dụng hàm chính là việc sử dụng và tái sử dụng các phần mã chương trình đã viết, nó rất có ý nghĩa trong làm việc lập trình theo nhóm và việc sử dụng hàm cũng đồng nghĩa với việc tiết kiệm bộ nhớ cho hệ thống.

Trong ngôn ngữ C/C++, mỗi chương trình con là một ***hàm*** được xây dựng theo cấu trúc chung. Bản thân chương trình cũng là một hàm. Đó là hàm *main()* đóng vai trò điều khiển toàn bộ chương trình. Hàm là một khối chương trình hoàn chỉnh để thực hiện một tính toán nào đó mà ta có thể gọi để thực hiện ở bất cứ chỗ nào trong chương trình. Bởi vậy khi lập trình, nếu gặp những đoạn chương trình phải sử dụng lại nhiều lần ở những chỗ khác nhau trong chương trình thì chúng ta nên viết những đoạn chương trình đó thành các hàm và sử dụng lời gọi hàm đó thay cho việc phải viết toàn bộ chương trình.

Có thể coi hàm là một phép toán do người dùng xác định. Hàm được biểu thị bởi một tên chứ không phải là một toán tử.

Các toán hạng của hàm, được gọi là các đối số của nó được xác định trong danh sách đối phân cách nhau bởi dấu phảy và được bao trong ngoặc tròn.

Kết quả của hàm thường được coi là kiểu trả lại của nó, hàm không cho giá trị trả lại có kiểu cho lại là void.

Các hành động thực tại mà một hàm thực hiện được xác định trong thân của hàm, thân của hàm được bao trong ngoặc nhọn ("{ }") và đôi khi còn được coi như một khối.

Cấu trúc của hàm gồm hai phần: *phần khai báo* và *phần câu lệnh*.

a. Phần khai báo

<tên kiểu> **<tên hàm>** (*<khai báo các tham biến hình thức>*

Trong đó:

- <tên kiểu> là kiểu giá trị trả về của hàm. Nếu hàm phải trả về một giá trị thì trước hết phải khai báo kiểu dữ liệu của hàm

- <tên hàm> là tên gọi do ta đặt để gọi thực hiện hàm này trong chương trình. Tên hợp lệ của hàm gồm 8 ký tự viết liền nhau. Nên đặt tên hàm mang tính gợi nhớ để tiện sử dụng.

- < Khai báo các biến hình thức>: là dãy khai báo các tham biến hình thức cách nhau bằng dấu phảy. Những tham số này khi hàm được gọi sẽ được truyền tham số thực. Nếu có nhiều tham số thì phải có dấu phảy (,) để phân cách chúng. Ví dụ: float max3s(a, b, c).

Tham số của một hàm có hai công dụng:

- Cung cấp các giá trị cho hàm khi ta gọi nó thực hiện.

- Lưu các kết quả tính toán được trong quá trình hàm hoạt động.

Như vậy ta có các loại tham số:

+ Tham số vào: cung cấp giá trị cho hàm

+ Tham số ra: lưu kết quả tính toán được trong hàm

+ Loại tham số vừa vào, vừa ra: vừa cung cấp giá trị cho hàm, vừa lưu kết quả tính toán được trong hàm.

Khi viết một hàm ta cần xác định xem hàm có bao nhiêu tham số.

Ví dụ: Phần khai báo của hàm GIAITHUA

long **giaithua** (*int n*);

b. Phần thân hàm

Phần này thể hiện các lệnh cần phải thực hiện khi hàm được gọi. Nó chính là một câu lệnh ghép (lệnh hợp thành) bao gồm khai báo hằng, kiểu, biến trong hàm và các câu lệnh. Thân hàm phải được đặt trong dấu { và }. Cấu trúc thân hàm như sau:

{ <*các khai báo hằng, kiểu, biến*>;

<các câu lệnh>;

...

[return <biểu thức>;]

}

Nếu hàm cho đúng một giá trị thì <biểu thức> trong câu lệnh return sẽ cho giá trị trả về của hàm này, tất nhiên kiểu của biểu thức phải trùng với kiểu của hàm đã khai báo trong phần khai báo thân hàm.

Nếu hàm không có giá trị trả về thì không cần viết câu lệnh *return* và khi đó kiểu của hàm là *void*. Các hàm không cho giá trị trả về giống như thủ tục *(procedure*) trong ngôn ngữ lập trình Pascal.

Ví dụ, ta có thể viết hàm tìm giá trị lớn nhất (max) trong ba số là max3s theo kiểu thủ tục hiển thị số cực đại trong ba số như sau:

```
# include <stdio.h>
# include <conio.h>
# include <iostream.h>
void max3s(float a, float b, float c)
{
float max;
max = a;
if (max <b) max = b;
if (max <c) max = c;
cout<<"max="<<max<<"\n";
}
```

```
int main(void)
{
clrscr();
float x,y,z;
x=3; y=8; z=1;
max3s(x,y,z);
getch();
return 0;
}
```

Trong ví dụ trên, hàm max3s không được trả về giá trị nên ta khai báo là void, các tham số a, b, c là tham số hình thức còn x, y, z là các tham số thực sự.

Trong trường hợp hàm cho nhiều giá trị thì các giá trị trả về này phải đặt trong các tham biến hình thức thông qua các biến con trỏ.

Trong dãy câu lệnh của hàm có thể sử dụng các tham biến hình thức, các *biến địa phương* được khai báo trong hàm này và cả các *biến toàn cục* được khai báo ở đầu chương trình. Do vậy, hàm có thể không có tham biến hình thức. Khi đó trong phần tham biến hình thức ta viết void. Trong hàm *main()* không được sử dụng các tham biến hình thức và các biến địa phương khai báo trong các hàm khác.

Tuy vậy cũng có hàm không có tham số nhưng vẫn phải để cặp dấu ngoặc đơn như hàm *main(),* hàm *clrscr()* chẳng hạn.

- *Khai báo kiểu cho các tham số*: nếu hàm có các tham số hình thức thì phải khai báo kiểu dữ liệu cho các biến. Nếu có nhiều tham số có cùng kiểu dữ liệu thì phải có dấu phảy (,) để phân cách các tham số. Ví dụ: *float a, b, c*;

- *Thân hàm*: thân hàm là phần được giới hạn bởi cặp dấu móc nhọn { và }. Trong thân hàm gồm:

+ *Khai báo các biến cục bộ*: Ví dụ *float max*. Các biến cục bộ chỉ có tác dụng trong thân của hàm, nó không có mối liên hệ gì với các

biến của các hàm khác trong chương trình. Trái lại, các tham số hình thức như *float a, b, c* kể trên lại được dùng để trao đổi dữ liệu từ hàm này cho hàm khác.

+ Trong thân hàm có thể sử dụng một câu lệnh return, giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

- *Sử dụng hàm*: Sau khi đã tạo lập hàm, khi cần sử dụng hàm trong chương trình chính, ta thực hiện lời gọi hàm có hoặc không có các tham số thực như sau:

Tên hàm *([Danh sách các tham số thực]);*

Chú ý: trước khi sử dụng hàm phải khai báo kiểu giá trị của nó: kiểu dữ liệu tên hàm(). Ví dụ: float x, y, z, max3s();

3.3.2. Cách thức truyền tham số

Đối với những hàm có tham số hình thức thì khi thực hiện phải được truyền giá trị thực cho tham số này. Có 2 cách truyền tham số: truyền theo trị và truyền theo biến.

Truyền theo trị

Trong C/C++ thì chế độ mặc định là truyền tham số theo giá trị, điều này có nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức.

Ví dụ 1:

```
/* Truyền theo trị*/
Void hoanvi(a,b)
{
int tam;
tam=a;
a=b;
b=tam;
}
```

```
main()
{
int a, b;
printf("\n Nhap so thu nhat:");
scanf("%d",&a);
printf("\n Nhap so thu hai:");
scanf("%d",&b);
hoanvi(a,b);
printf("\n Sau khi hoan vi, so thu nhat=%d",a);
printf("\n Sau khi hoan vi, so thu hai=%d",b);
getch();
}
```

Khi chạy chương trình nếu ta cung cấp giá trị:

Nhap so thu nhat: 12

Nhap so thu hai: 24

Thì sau khi thực hiện, kết quả là:

Sau khi hoan vi, so thu nhat = 12

Sau khi hoan vi, so thu hai = 24

Như vậy nếu ta muốn đổi giá trị hai biến a và b cho nhau thì đoạn chương trình này đã không thực hiện được mục đích đó. Lý do là: các tham số thực (các giá trị thực của hai biến a, b) khi truyền cho các tham số hình thức trong hàm **hoanvi(a,b)** chỉ truyền giá trị chứ không truyền địa chỉ, do đó sự thay đổi trong các hàm này (hoán vị 2 số) không thể ghi lại được.

Truyền theo biến

Nếu muốn truyền cả nội dung và địa chỉ của biến thì phải sử dụng biến con trỏ (sẽ được trình bày chi tiết trong phần sau).

Ví dụ:

```
/* Truyền theo biến*/
Void hoanvi(int *a,int *b) /*a,b là biến kiểu con trỏ*/
{
int tam;
tam =*a;
*a=*b;
*b=tam;
}
main()
{
int a, b;
printf("\n Nhap so thu nhat:");
scanf("%d",&a);
printf("\n Nhap so thu hai:");
scanf("%d",&b);
hoanvi(a,b);
printf("\n Sau khi hoan vi, so thu nhat=%d",a);
printf("\n Sau khi hoan vi, so thu hai=%d",b);
getch();
}
```

Khi chạy chương trình nếu ta cung cấp giá trị:

Nhap so thu nhat: 12

Nhap so thu hai: 24

Thì sau khi thực hiện, kết quả là:

Sau khi hoan vi, so thu nhat = 24

Sau khi hoan vi, so thu hai = 12

Lý do của việc chúng ta đổi được giá trị hai biến a và b trên là do: việc truyền tham biến là truyền địa chỉ của tham biến thực sự cho tham

biến hình thức tương ứng. Hai tham biến này tuy khác nhau nhưng lại có chung một địa chỉ, do vậy chúng cũng có chung dữ liệu.

Truyền tham số kiểu tham chiếu cho hàm trong C++

Trong C++, ngoài việc truyền tham số kiểu tham trị và truyền tham số kiểu con trỏ đã nêu trên còn cho phép truyền tham số cho hàm kiểu tham chiếu. Có nghĩa là *giá trị của tham số bị hàm thay đổi cũng sẽ được giữ lại khi ra khỏi hàm.*

Bản chất của cách truyền tham số kiểu tham chiếu khác *bản chất* của hai cách truyền tham số truyền thống: kiểu tham trị và kiểu con trỏ. Theo cách truyền tham số truyền thống thì chỉ có bản sao giá trị (hoặc giá trị của biến hoặc giá trị là địa chỉ biến) truyền vào hàm, còn trong cách truyền tham số kiểu tham chiếu thì hàm làm việc trực tiếp trên biến truyền vào hàm, thông qua bí danh của nó.

Ví dụ:

```
# include <stdio.h>
# include <conio.h>
void hoandoi (int  &, int  &);
void main ()  {
int x = 10, y = 5;
clrscr ();
hoandoi (x, y);
printf ("x = %d  y = %d\n", x, y);
getch (); }
void hoandoi (int &x, int &y)
{ int  z = x; x = y ; y = z; }
```

Về mặt cú pháp, khi truyền tham số cho hàm theo kiểu tham chiếu ta chỉ cần báo hiệu địa chỉ của biến ở phần khai báo, còn trong thân hàm và trong lời gọi hàm, ta vẫn viết tên biến một cách bình thường.

Chú ý với cách truyền tham số kiểu tham chiếu ta không được cấp phát bộ nhớ động cho biến tham chiếu và không dùng phép toán tăng

giảm địa chỉ cho biến tham chiếu vì biến tham chiếu gắn liền với địa chỉ tĩnh trong bộ nhớ và do đó không được phép thay đổi địa chỉ này.

3.3.3. Phạm vi biến và lập trình mô-đun

a. Phạm vi biến

Phạm vi cục bộ (local variable)

Biến cục bộ là các biến được khai báo trong các hàm. Biến cục bộ được cấp phát bộ nhớ vào lúc hàm được gọi và được gán khởi đầu bằng một giá trị không xác định nào đó. Đến khi kết thúc hàm thì vùng nhớ dành cho các biến cục bộ trong hàm được giải phóng, tức là các biến này không còn truy nhập được. Không nên truyền địa chỉ của biến cục bộ ra ngoài phạm vi của nó.

Các biến cục bộ có phạm vi hoạt động chỉ trong hàm khai báo và nó có thể trùng tên với các biến toàn cục (global) mà không ảnh hưởng gì. Trong trường hợp này, việc gọi tên truy nhập biến toàn cục để sử dụng trong hàm chỉ cần thêm toán tử phạm vi "::" đằng trước tên biến.

Vì ta còn có thể khai báo biến ở trong một khối lệnh của một hàm nào đó, do vậy ngoài biến cục bộ có phạm vi trong hàm, còn có biến cục bộ có phạm vi trong một khối lệnh, nếu biến cục bộ khai báo trong khối lệnh thì nó chỉ có phạm vi trong khối lệnh, khi ra khỏi khối lệnh, biến cục bộ đó sẽ bị giải phóng và không sử dụng được nữa.

Biến cục bộ tĩnh (Static Local Variable)

Biến cục bộ tĩnh là biến cục bộ được xác định bằng từ khóa static.

Cú pháp:

static **type** *varName*;

Trong đó:

type: kiểu biến

varName: tên biến.

Khi dùng biến cục bộ, sau khi ra khỏi hàm, biến cục bộ sẽ bị giải phóng hết vùng nhớ và giá trị của biến sẽ bị xóa tức là không thể truy

nhập biến cục bộ từ bên ngoài hàm. Biến cục bộ tĩnh, cũng giống biến cục bộ là chỉ gọi và truy nhập trong hàm, nhưng giá trị của nó không bị xóa khi kết thúc hàm. Ví dụ, để theo dõi một hàm đệ quy đã được gọi lặp bao nhiêu lần, ta có thể dùng biến tĩnh cục bộ để đếm.

Biến toàn cục (Global variable)

Biến toàn cục là biến được khai báo ngoài tất cả các hàm. Có thể truy nhập biến toàn cục từ bất cứ hàm nào, từ bất cứ nơi đâu trong hàm và có thể sử dụng biến toàn cục để gán giá trị trả về trong hàm.

Khi chương trình bắt đầu được khởi tạo thì biến toàn cục được cấp phát bộ nhớ và nó sẽ tồn tại suốt thời gian tồn tại của chương trình.

Các biến toàn cục được tự động gán giá trị ban đầu là 0.

Với các biến ngoài, khi đã khai báo ở một mô-đun, có thể gọi sử dụng ở một mô-đun khác bằng cách dùng từ khóa extern. Từ khóa *extern* chỉ là cách để báo cho chương trình dịch biết rằng, biến đã được khai báo tường minh ở đâu đó (ở một mô-đun khác), mà không cần phải cấp phát bộ nhớ cũng như không phải gán giá trị khởi đầu lại cho biến.

Biến toàn cục tĩnh

Biến toàn cục tĩnh là biến toàn cục có thêm từ khóa static đứng trước. Biến toàn cục tĩnh khác biến toàn cục là chỉ có thể truy nhập đến nó từ bất cứ đâu bên trong tệp khai báo nó, tức là không thể truy nhập được biến toàn cục tĩnh theo kiểu dùng cách gọi extern (không có extern biến toàn cục tĩnh). Như vậy có thể có nhiều biến toàn cục tĩnh có cùng tên nhưng được khai báo ở các tệp khác nhau trong chương trình.

b. Truyền giá trị cho hàm

Có hai phương pháp truyền giá trị cho hàm: dùng biến toàn cục trong chương trình và dùng danh sách đối hình thức.

Phương pháp dùng biến toàn cục

Khả năng thâm nhập chung vào biến toàn cục từ bất cứ đâu trong chương trình là lợi ích chính của nó. Tính thấy được của biến toàn cục làm cho nó trở thành phương pháp thuận tiện cho việc liên lạc giữa các

thành phần khác nhau của chương trình. Nhưng việc sử dụng biến toàn cục để truyền giá trị cho hàm cũng có nhược điểm:

- Các hàm có dùng tới biến toàn cục phụ thuộc vào sự tồn tại và kiểu của biến toàn cục đó làm cho việc dùng lại hàm này trong ngữ cảnh khác trở lên khó khăn.

- Nếu cần sửa đổi chương trình thì những sự phụ thuộc toàn cục sẽ làm tăng khả năng sinh lỗi. Hơn nữa việc đưa vào những thay đổi cục bộ đòi hỏi việc hiểu biết toàn bộ chương trình.

- Nếu một biến toàn cục nhận một giá trị không đúng thì phải tìm kiếm trong toàn bộ chương trình để xác định xem lỗi xuất hiện ở đâu. Không có việc cục bộ hóa.

- Việc đệ quy lại còn khó đi đến đúng chỗ hơn khi hàm dùng một biến toàn cục.

Phương pháp dùng danh sách đối

Danh sách đối cùng với kiểu cho lại của hàm xác định ra giao diện chung của hàm. Với việc truyền theo danh sách đối, một chương trình nhìn chung là không phải thay đổi khi hàm thay đổi. Mặt khác một hàm có thể dùng qua nhiều chương trình, không nhất thiết bị giới hạn vào một ứng dụng riêng. Nhưng với một hàm, nếu số lượng đối số quá lớn thì hàm sẽ rất khó dùng vì dễ gây ra lỗi do truyền sai kiểu đối. Để dễ dùng hơn thì một hàm thường chỉ dùng tối đa đến tám đối. Nếu một hàm mà phải dùng đến số đối nhiều hơn thì ta có thể định nghĩa ra một kiểu lớp (hoặc cấu trúc) chứa các giá trị đối. Điều này có hai lợi ích:

- Độ phức tạp của danh sách đối được giảm đi

- Việc kiểm tra tính hợp lệ của các giá trị đối có thể được thực hiện bằng các hàm thành phần của lớp chứ không phải là bên trong hàm, làm cho giảm kích cỡ của hàm và hàm sẽ dễ hiểu hơn.

Cú pháp danh sách đối

Một hàm không có đối có thể được biểu thị bằng danh sách đối rỗng hoặc một danh sách đối chỉ chứa một từ khóa *void*.

Danh sách đối còn được gọi là dấu hiệu của hàm, bởi vì nó thường dùng để phân biệt một thể nghiệm của hàm này với hàm khác. Tên và dấu hiệu của một hàm là xác định duy nhất nó.

Dấu hiệu bao gồm một danh sách các kiểu đối phân cách nhau bởi dấu phảy, có thể có tên biến đi theo bộ xác định kiểu nhưng không bắt buộc.

Không có hai tên đối nào xuất hiện trong danh sách đối được phép trùng nhau. Tên đối cho phép đối đó được thâm nhập từ bên trong thân hàm, do vậy tên đối không cần viết trong khai báo hàm.

Bộ khởi đầu ngầm định

Một hàm có thể xác định một giá trị ngầm định cho một hay nhiều đối của nó bằng cách dùng cú pháp khởi đầu bên trong dấu hiệu. Một hàm có cung cấp các bộ khởi đầu ngầm định có thể được gọi với các đối thực tại tương ứng hoặc không có các đối đó. Nếu một đối được đưa ra thì nó sẽ thay thế cho giá trị ngầm định, nếu không thì giá trị ngầm định sẽ được dùng.

Chú ý rằng khi gọi hàm, không thể cung cấp một giá trị cho tham số đứng sau mà không cung cấp giá trị cho các tham số đứng trước, tức là các đối của lời gọi hàm phải được gán giá trị theo thứ tự từ trái qua phải.

Truyền đối

Khi chạy chương trình, mỗi khi một hàm được gọi, các đối được cấp phát bộ nhớ trong ngăn xếp. Vùng nhớ trong ngăn xếp dành cho đối tồn tại cho tới khi hàm kết thúc, lúc đó vùng nhớ này tự động bị loại bỏ.

Danh sách các đối của hàm được mô tả gọi là đối hình thức, mỗi đối hình thức được cấp phát bộ nhớ trong ngăn xếp, kích cỡ của chúng được xác định theo kiểu đối.

Các biểu thức nằm giữa hai dấu ngoặc tròn của lời gọi hàm được gọi là các đối thực của hàm. Việc truyền đối chính là một tiến trình khởi đầu cho vùng nhớ của các đối hình thức theo các đối thực của hàm.

Trong ngôn ngữ C/C++, có hai phương pháp truyền đối, truyền theo giá trị và truyền theo địa chỉ.

Truyền tham số theo giá trị:

Phương pháp khởi đầu ngầm định cho việc truyền đối là sao chép giá trị các đối thực vào vùng nhớ của đối hình thức gọi là truyền đối theo giá trị.

Đặc điểm:

- Giá trị đối thực được sao chép vào vùng nhớ của các đối số hình thức tương ứng của hàm, tạo thành các bản sao cục bộ riêng của nó.

- Giá trị các đối số hình thức của hàm thay đổi trong quá trình tính toán, nhưng giá trị của đối thực không thay đổi trong quá trình tính toán đó. Người lập trình không cần phải cất giữ và khôi phục các giá trị đối khi tiến hành gọi hàm. Do vậy phương pháp này là biện pháp an toàn ít yêu cầu người dùng chung phải để ý đến.

- Tuy nhiên, việc truyền theo giá trị không phải là phương pháp thích hợp cho mọi hàm. Những tình huống mà truyền theo giá trị tỏ ra không thích hợp bao gồm:

+ Khi phải truyền một đối tượng lớp lớn như một đối. Phí tổn về thời gian và không gian nhớ để cấp phát và sao chép đối tượng lên lớp thường cao so với các chương trình ứng dụng thực.

+ Không thay đổi được giá trị đối thực khi cần thay đổi.

Truyền tham số theo địa chỉ:

Hàm thực hiện truyền các giá trị địa chỉ của đối thực và các đối hình thức được gọi là các đối tham chiếu.

Việc truyền tham số theo địa chỉ khắc phục được các nhược điểm của phương pháp truyền tham số theo giá trị.

Chương trình gọi hàm sẽ truyền cho hàm giá trị địa chỉ của đối thực;

Các thao tác tính toán trong hàm là thao tác trực tiếp với giá trị của đối thực, chứ không phải là làm việc với bản sao của đối thực, do vậy nó làm thay đổi giá trị đối thực.

c. Giá trị cho lại của hàm

Kết quả của hàm cho lại phải có kiểu.

Hàm không cho lại một giá trị nào có kiểu là void.

Kiểu cho lại của hàm có thể là kiểu cơ sở, kiểu do người dùng định nghĩa hay kiểu suy dẫn.

Hàm không có giá trị cho lại tường minh, theo ngầm định được coi như cho lại một giá trị có kiểu int.

Câu lệnh return kết thúc việc thực hiện hàm hiện tại, điều khiển của chương trình trở về hàm đã gọi hàm vừa kết thúc.

Có hai dạng câu lệnh return:

return;

return expr;

- Dạng thứ nhất dùng cho hàm có kiểu *void* (hàm không có giá trị trả về). Câu lệnh return này không thực sự cần thiết cho hàm kiểu này. Nó được dùng chủ yếu để kết thúc sớm hàm theo một điều kiện nào đó.

- Dạng thứ hai của câu lệnh *return* xác định ra kết quả của hàm. Biểu thức *Expr* có thể là một biểu thức phức tạp, tùy ý, bản thân nó cũng có thể chứa cả lời gọi hàm (trong ví dụ về hàm đệ quy).

Nếu giá trị cho lại không đúng với kiểu cho lại đã khai báo thì việc chuyển đổi không tường minh sẽ được tiến hành.

Một hàm có thể cho lại một giá trị. Nếu logic chương trình đòi hỏi phải cho lại nhiều giá trị thì ta có thể sử dụng các phương pháp sau để nhận giá trị trả về của hàm:

- Trả giá trị trả về qua biến toàn cục

- Các đối hình thức có thể được định nghĩa hoặc như kiểu con trỏ, hoặc như kiểu tham khảo

- Cho lại kiểu dữ liệu tập hợp có chứa nhiều giá trị. Với cách này thì nên dùng lớp mềm dẻo hơn mảng.

3.3.4. Các ví dụ minh họa về hàm

Ví dụ 1: Tạo một hàm tìm một số lớn nhất trong 3 số.

```
#include<stdio.h>
#include<conio.h>
float max3s(a, b, c)   /*Khai bao ham*/
float a,b,c;
{
float max;             /*Bien cuc bo*/
max=a>b?a:b;           /*lenh dieu kien*/
return(max>c?max:c);
}                      /*Ket thuc ham*/
#include<stdio.h>
main()
{
clrscr();
float x,y,z, max3s();
int s;
tt:printf("\nNhap 3 so:");
printf("\n So thu nhat=");
scanf("%f",&x);
printf("\n So thu hai=");
scanf("%f",&y);
```

```
printf("\n So thu ba=");
scanf("%f",&z);
printf("\n So lon nhat la %8.2f",max3s(x,y,z));
getch();
}
```

Ví dụ 2: Tạo một hàm tính giai thừa của n số nguyên đầu tiên.

```
long int gt(n)
int n;
{
int s=1,i;
for (i=1;i<=n;++i)
s*=i;
return(s);
      }
main()
      {
            long gt();
            int n;
printf("\nTinh giai thua cua n= ");
scanf("%d",&n);
printf("\nNeu n=%3d,thi n! = %8ld",n,gt(n));
}
```

Ví dụ 3: Viết chương trình nhập vào tọa độ của 2 véc-tơ, mỗi véc-tơ 5 thành phần, tính mô-đun và tổng của 2 véc-tơ.

Ở đây mảng được sử dụng làm tham số hình thức của một hàm trong chương trình nên ta phải khai báo theo cú pháp sau:

<Kiểu> <Tên hàm> (<kiểu> <tên mảng[],...);

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

```
float modul(int x[]);
main()
{
int A[5], B[5];
clrscr();
printf("\n");
nhap(A,'A');
nhap(B,'B');
printf("\n KET QUA TINH TOAN CUA 2 VECTO A va B");
trinhbay(A,'A');
trinhbay(B,'B');
printf("\n ----------");
tong(A,B);
printf("\n Module cua vecto A la: %f",modul(A));
printf("\n Module cua vecto B la: %f",modul(B));
getch();
}
nhap(int x[], char chu);
{ int k;
for (k=0;k<5,k++)
{
printf("\n %c[%d]=",chu,k+1);
scanf("%d",&x[k]);
}
printf("\n");
}
trinhbay(int x[], char chu)
{
printf("\n %c={%d\,%d\,%d\,%d\,%d\}",chu,x[0],x[1],x[2],x[3],x[4]);
}
float modul(int x[])
{
int k;
```

```
float s=0;
for (k=0; k<5; k++)
s=s+x[k]*x[k];
s=sqrt((double),s);
return(s);
}
tong int X[], int Y[])
{
int z[5],k;
for (k=0; k<5; k++)
z[k]= x[k]+ y[k]
trinhbay(z,'C');
}
```

Ví dụ 4: Viết chương trình nhập ngẫu nhiên một mảng gồm 1000 phần tử là các số nguyên nhỏ hơn 100 và tìm hai số liên tiếp có tổng bằng số nguyên X nhập vào từ bàn phím.

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
int a[1000];
void tao()
{
int k;
randomize();
for (k=0;k<1000;k++)
a[k]= random(100);
}
void tim()
{
int x,k;
printf("\n Nhap vao gia tri cua x=");
```

```
scanf("%d%,&x");
for (k=0; k<999;k++)
if(a[k]+a[k+1]==x)
printf("a[%d]+[%d]=%d\n",k,k+1,x);
puts("ket thuc");
}
main()
{
tao();
tim();
getch();
}
```

Ví dụ 5: Tính tích phân $I = \int_a^b f(x)dx$ bằng phương pháp hình thang.

Phân tích thuật toán: Nếu hàm f(x) liên tục trong khoảng [a, b] thì tích phân I được tính gần đúng bằng tổng các hình thang nhỏ. Giả sử ta chia [a, b] thành m đoạn có bước tích phân h = (b-a)/m thì x_i = a+ ih (i = 0, 1.. m) và khi đó:

$$I = \int_a^b f(x)dx = h[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{m-1} f(a+ih)] + R_m(f)$$

trong đó phần dư: $R_m(f) = -\frac{(b-a)^3}{12m^2} f'(\xi), \xi \in [a,b]$

Chương trình bao gồm các công việc sau:

1. Xác định hàm cần lấy tích phân (chẳng hạn $y = e^{-x}\sin x$) bằng hàm float ham(float x).

2. Đưa số liệu vào gồm cận dưới a, cận trên b và số điểm tính m.

3. Tính tổng giá trị tích phân theo phương pháp hình thang bằng vòng lặp. Ở đây có thể sử dụng for là vòng lặp số lần xác định vì số lần lặp n được biết trước.

Chương trình được xây dựng như sau:

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
float ham(float);
void main()
{
clrscr();
int i,n;
float a,b,c,x,s,h,tp;
printf("\n Nhap can duoi a=");scanf("%f",&a);
printf("\n Nhap can tren b=");scanf("%f",&b);
printf("\n Cho so diem tinh n=");scanf("%d",&n);
h=(b-a)/n;
x=a;
s= (ham(a)+ham(b))/2;
for (i=1; i<n-1; i++)
{ x=x+i*h;s=s+ham(x);}
tp=s*h;
printf("\n Gia tri cua tich phan =%f",tp);
getch();
return;}/*het chuong trinh chinh*/
/*ham duoi dau tich phan*/
float ham(float x)
{ return exp(-x)*sin(x); }
```

Trong chương trình này chúng ta khai báo tên hàm dưới dấu tích phân **ham** trên đầu chương trình chính main() còn nội dung hàm này được xây dựng sau return của chương trình chính main().

3.3.5. Hàm đệ quy

Trong thực tế, chúng ta gặp rất nhiều đối tượng mà khó có thể định nghĩa nó một cách tường minh, nhưng lại dễ dàng định nghĩa đối tượng qua chính nó. Kỹ thuật định nghĩa đối tượng qua chính nó được gọi là kỹ thuật đệ quy (*recursion)*. Hàm gọi đến chính nó, trực tiếp hay gián tiếp, đều được coi là hàm đệ quy. Khái niệm đệ quy giúp chúng ta mô tả một tập vô hạn bằng các phát biểu hữu hạn.

Ví dụ: chúng ta có thể phát biểu định nghĩa n! bằng hai cách sau:

1) n! là tích của các số nguyên dương liên tiếp từ 1 tới n.

2) n! bằng 1 nếu n = 0 hoặc bằng n*(n-1)! nếu n > 0.

Các giải thuật đệ quy đều được xây dựng thông qua hai bước: bước phân tích và bước thay thế ngược lại.

Ví dụ: Để tính tổng n số nguyên: S(n)= 1+2+... + n, ta có thể thực hiện thông qua hai bước sau:

Bước phân tích:

- Để tính được S(n), đầu tiên phải tính trước được S(n-1) sau đó tính S(n) = S(n-1) + n

- Để tính được S(n-1), phải tính trước được S(n-2) sau đó tính: S(n-1) = S(n-2) + n-1

- Tiếp tục tính S(n-2), S(n-3)... cho đến S(1).

- Để tính được S(2) ta phải tính được S(1) sau đó tính:

S(2) = S(1)+2

- Và cuối cùng tính S(1) chính là 1.

Bước thay thế ngược lại:

Xuất phát từ S(1), thay thế ngược lại ta xác định được S(n):

S(1) = 1

S(2) = S(1)+2

S(3) = S(2) + 3

...

S(n) = S(n-1) + n

Cú pháp:

```
type fname(parameter_list)
{
if (condition_stop==true) // điều kiện dừng
 return(value_atStop); // giá trị tại thời điểm dừng
else
 return(fname(value_of_func_with_next_paramater));
// giá trị được tính bởi việc gọi lại hàm với tham số khác
}
```

Hàm đệ quy bao giờ cũng phải xác định điều kiện dừng, nếu không nó sẽ đệ quy mãi và chương trình sẽ bị treo do tràn bộ nhớ (trong ví dụ trên điều kiện dừng là n<=1).

Hàm đệ quy có thể chạy chậm hơn hàm không đệ quy nhưng chương trình viết lại nhỏ gọn, ít phức tạp hơn.

Ta xem xét một số ví dụ sau:

Ví dụ 1: Tính a^n bằng giải thuật đệ quy với mọi số thực a và số tự nhiên n.

```
double luythua (float a, int n) {
if (n == 0)
return(1);
return(a*luythua(a,n-1)); }
```

Ví dụ 2: Thuật toán đệ quy tìm ước số chung lớn nhất của hai số nguyên dương a và b với b>a.

Phân tích: ta có thể rút gọn về bài toán tìm ước số chung lớn nhất của (b mod a) và a, vì USCLN(b mod a, a) = USCLN(a, b).

Dãy các rút gọn liên tiếp có thể đạt được cho tới khi đạt điều kiện dừng USCLN(0, a) = USCLN(a, b) = a.

```
int USCLN(int a, int b) {
if (a==0)
return (b);
return(USCLN(b % a, a));
}
```

Ví dụ 3: Thuật toán đệ quy tính n!

```
long giaithua(int n) {
if (n==1)
return(1);
return(n*giaithua(n-1));
```

3.3.6. Hàm inline

Trong C++ có thể định nghĩa các hàm được thay thế trực tiếp thành mã lệnh máy tại chỗ gọi (inline) mỗi lần được tham chiếu. Hàm kiểu *inline* thường áp dụng cho các hàm nhỏ, có thân hàm thường là một vài dòng và là hàm hay được sử dụng.

Một hàm *inline* được định nghĩa và được sử dụng giống như bình thường. Điểm khác nhau duy nhất là phải đặt mô tả ***inline*** trước khai báo hàm.

Cú pháp: ***inline type fname(parameter_list)***

Ưu điểm của các hàm *inline* là chúng không đòi hỏi các thủ tục bổ sung khi gọi hàm và trả giá trị về. Do vậy hàm *inline* được thực hiện nhanh hơn so với các hàm thông thường.

Từ khóa *inline* yêu cầu chương trình biên dịch xử lý hàm *min()* khác với các hàm thông thường. Cụ thể là, mỗi lần gọi *min()*, trình biên dịch ghép trực tiếp các chỉ thị tương ứng của hàm vào trong chương trình (ở dạng ngôn ngữ máy). Do đó cơ chế quản lý lời gọi và trở về không cần nữa (không cần lưu ngữ cảnh, sao chép các thông

số...), nhờ vậy tiết kiệm thời gian thực hiện. Bên cạnh đó các chỉ thị tương ứng sẽ được sinh ra mỗi khi gọi hàm, do đó bộ nhớ lưu trữ tăng lên khi hàm được gọi nhiều lần.

Điểm bất lợi khi sử dụng các hàm *inline* là nếu chúng quá lớn và được gọi thường xuyên thì kích thước chương trình sẽ tăng lên rất nhanh. Vì lý do này, chỉ những hàm đơn giản, không chứa các cấu trúc lặp mới được khai báo là hàm *inline*.

Bài tập chương 3

1. Viết chương trình nhận ký tự vào từ bàn phím, nếu ký tự nhận là ESC (0x1b) thì hiển thị tổng số ký tự đã nhận và kết thúc chương trình.
2. Viết chương trình với vòng lặp do-while hiển thị các số từ 10 đến 20.
3. Viết chương trình kiểm tra từ khóa là số nguyên lưu trước trong máy. Nếu vào đúng từ khóa chương trình thông báo dòng chữ "Đã vào đúng từ khóa" kết thúc chương trình, nếu vào sai từ khóa, chương trình yêu cầu vào lại và nếu ba lần vào sai liên tiếp chương trình thông báo "Từ khóa không đúng" và kết thúc chương trình.
4. Viết chương trình hiển thị bảng mã ASCII. Chú ý: không hiển thị 32 ký tự đầu tiên.
5. Viết chương trình với lệnh for tính giai thừa của một số nhập vào từ bàn phím.
6. Viết chương trình với switch-case để đếm số nguyên âm và phụ âm nhập từ bàn phím.
7. Viết chương trình đếm các từ và các dòng trong tệp chương trình C++.
8. Cho chương trình

```
main()
{
cout<< "Lop hoc C/C++";
}
```

Hãy sửa đổi chương trình trên nhưng vẫn giữ nguyên hàm main() để chương trình có thể hiển thị trên màn hình như sau:

HVANND don chao Lop hoc C/C++

9. Viết chương trình tính tích phân số $I = \int_a^b f(x)dx$ bằng phương pháp Simson.
10. Viết hàm đổi chỗ hai số nguyên và viết hàm main() sử dụng chúng.
11. Viết các hàm tính cực đại và cực tiểu của hai số. Xây dựng ví dụ sử dụng chúng.

Cho đoạn chương trình sau:

```
#include<iostream.h>
void test(float x);
void main()
{
float x, y;
Test(x);
cout << “x= ” << x << “ , y= ”<< y;
}
void Test(float x)
{
float y;
y = 25.0;
x = y;
}
```

Cho biết giá trị x và y mà chương trình sẽ in ra.

Cho biết giá trị x và y mà chương trình sẽ in ra nếu thay void Test(float x) bằng void Test(float &x).

Cho biết giá trị x và y mà chương trình sẽ in ra nếu thay void Test(float x) bằng void Test(float &x) và gọi trong hàm main() là Test(y).

12. Viết hàm đếm và hiển thị số lần các hàm khác gọi nó. Cho ví dụ sử dụng.

13. Viết chương trình có 3 hàm main(), fun1() và fun2(). Hãy khai báo mảng chứa 10 ký tự trong hàm main(), nạp các ký tự từ A đến J vào chúng trong hàm fun1(), sử dụng hàm fun2() để hiển thị mảng nói trên.
14. Viết chương trình chứa 3 hàm, một hàm tính bình phương, hàm thứ hai tính lập phương của số nguyên. Hàm thứ ba là hàm main() để hiển thị bảng các giá trị bình phương và lập phương của 15 chữ số nguyên đầu tiên.
15. Viết chương trình giải phương trình bậc hai:
    - Xây dựng hàm nhập các hệ số a, b, c
    - Xây dựng hàm để tính nghiệm số
    - Xây dựng hàm để in kết quả
16. Viết hàm tính $S = x^y$. Ví dụ tính $S = 0.45^{1.34}$

Chương 4

MẢNG VÀ CON TRỎ

Trong lập trình để biểu diễn các dữ liệu thực tế thì các kiểu dữ liệu chuẩn trong các ngôn ngữ lập trình thường chưa đủ để mô phỏng nó, do đó các ngôn ngữ lập trình thường cho phép người dùng tự xây dựng các kiểu dữ liệu phức tạp từ các kiểu dữ liệu chuẩn hoặc các kiểu dữ liệu đã được xây dựng trước.

Nội dung chương này sẽ trình bày các vấn đề sau đây:

- Kiểu dữ liệu mảng và xử lý
- Mảng nhiều chiều
- Con trỏ
- Con trỏ và mảng

4.1. KIỂU DỮ LIỆU MẢNG VÀ XỬ LÝ MẢNG

Như ta biết, mỗi biến chỉ có thể biểu diễn một giá trị. Muốn biểu diễn một dãy số hay một bảng số ta có thể sử dụng nhiều biến nhưng rất bất tiện. Vì vậy trong hầu hết các ngôn ngữ lập trình đều thiết kế một kiểu dữ liệu chuyên dụng cho loại này được gọi là *kiểu mảng*. *Mảng* là tập hợp nhiều phần tử có *cùng một kiểu dữ liệu* và có *chung một tên*. Mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu dữ liệu thì cũng có từng đó kiểu mảng.

Cú pháp khai báo mảng:

<Tên kiểu> ***<tên mảng>***[<**Kích thước**>];

<Tên kiểu>: là kiểu dữ liệu của các phần tử mảng.

<*Tên mảng*> là tên gọi đặt cho mảng để truy cập tới nó và các phần tử của nó

<Kích thước> cho biết số phần tử mảng

Ví dụ, có các mảng sau đây:

int a[10], b[4][2]

float x[5], y[3][3]

Mảng a[10]: mảng 1 chiều a, có 10 phần tử thuộc kiểu số nguyên (**int**) được đánh số như sau: a[0], a[1],... a[9].

Mảng b[4][2]: mảng 2 chiều b có 8 phần tử thuộc kiểu nguyên (int) gồm 4 hàng và 2 cột được sắp xếp như sau:

b[0][0] b[0][1]

b[1][0] b[1][1]

b[2][0] b[2][1]

b[3][0] b[3][1]

Mỗi phần tử b[i][j] biểu diễn một giá trị kiểu nguyên.

Mảng x[5]: mảng 1 chiều x có 5 phần tử thuộc kiểu số thực

Mảng 2 chiều y gồm 3 hàng, 3 cột có 9 phần tử, mỗi phần tử y[i][j] chứa một giá trị kiểu số thực (float).

Một định nghĩa mảng bao gồm một bộ xác định: kiểu mảng, tên gọi mảng và số chiều của mảng. Chiều của mảng xác định số phần tử có trong mảng, được bao trong dấu ngoặc vuông ("[ ]"), mảng phải có số chiều lớn hơn hoặc bằng một. Giá trị của chiều phải là một biểu thức hằng, tức là phải có thể tính ra được giá trị của nó vào lúc dịch, không thể dùng biến không xác định để xác định chiều của mảng.

Ta có thể truy nhập các đối tượng trong mảng thông qua giá trị chỉ số xác định vị trí của chúng trong mảng. Các phần tử trong mảng được đánh chỉ số bắt đầu từ 0, ví dụ với mảng có 10 phần tử thì các phần tử sẽ được đánh số từ 0 đến 9.

Mảng có thể khởi đầu tường minh bằng cách xác định danh sách các giá trị cách nhau bởi dấu phảy và được bao trong ngoặc nhọn. Một mảng được khởi đầu tường minh không cần phải xác định giá trị chiều, trình biên dịch sẽ xác định kích cỡ mảng theo số phần tử được liệt kê.

Nếu kích cỡ của chiều được xác định thì số phần tử khởi đầu phải không được vượt quá kích cỡ đó, ngược lại trình biên dịch sẽ báo lỗi. Nếu kích cỡ của chiều lớn hơn số phần tử khởi đầu được liệt kê thì các phần tử mảng không được khởi đầu tường minh sẽ được đặt giá trị là 0.

Có thể khởi đầu một mảng ký tự với hoặc là một danh sách các ký tự cách nhau bởi dấu phảy đặt trong ngoặc nhọn hoặc một hằng xâu. Tuy nhiên hai dạng này là không tương đương vì hằng xâu còn chứa thêm ký tự kết thúc xâu '\0'.

Một mảng không thể khởi đầu với mảng khác, cũng không thể gán mảng này sang mảng khác.

Để sao chép mảng này sang mảng khác phải sao chép lần lượt các phần tử của mảng.

Một mảng, khi được truyền cho hàm sẽ được chuyển thành một con trỏ tới phần tử đầu tiên của mảng, thực tế là thông tin về chiều của mảng bị mất, do vậy một mảng được truyền cho hàm phải bằng cách nào đó thông báo thêm về chiều của nó. Ví dụ có thể truyền thông tin về chiều của mảng như là tham số tiếp theo.

Mọi biểu thức cho kết quả nguyên đều có thể được dùng làm chỉ số cho mảng. Ngôn ngữ không đưa ra việc kiểm tra miền chỉ số cả lúc dịch cũng như lúc chạy. Không có gì ngăn cản người lập trình vượt quá cận mảng, do vậy phải chú ý vào chi tiết và kiểm thử kỹ lưỡng chương trình.

Lấy địa chỉ của phần tử mảng

Địa chỉ của các phần tử trong mảng một chiều được xác định thông qua toán tử & theo cú pháp:

& tên_biến [i] (i là chỉ số của mảng).

Tên của mảng sẽ chứa địa chỉ đầu của mảng.

Ví dụ: nếu có int a[10] thì a = &a[0].

Nhập xuất dữ liệu cho các phần tử mảng

Để nhập hay xuất dữ liệu cho các phần tử mảng có hai cách:

Cách 1: Nhập trực tiếp

Chỉ sử dụng được với mảng một chiều và mảng hai chiều mà các phần tử là kiểu *int*.

Ví dụ: Viết chương trình để nhập dữ liệu vào mảng a có 5 phần tử, sau đó in các giá trị của mảng ra màn hình.

```
# include <stdio.h>
# include <conio.h>
void main ()
{
int a[5]; int i;
clrscr ();
/* Nhap du lieu vao mang a */
for ( i = 0; i < 5 ; i ++)
{
printf("\na [%d] ", i) ;
scanf("%d", &a[i]) ; /* nhap truc tiep cac phan tu mang a */
}
/* Xuat cac gia tri mang ra man hinh */
for ( i = 0; i < 5 ; i ++)
printf("%d", a[i]);
getch();
}
```

Cách 2: Nhập gián tiếp

Đây là cách sử dụng tốt cho mảng một chiều cũng như nhiều chiều. Việc nhập dữ liệu vào mảng thông qua một biến trung gian *temp* nào đó, sau đó gán giá trị biến *temp* cho các phần tử mảng.

Ví dụ: Viết chương trình để nhập dữ liệu vào ma trận vuông a có 3x3 phần tử, sau đó in các giá trị của mảng ra màn hình.

```
# include <stdio.h>
# include <conio.h>
void main ()
{
float temp, a[3] [3]; int i, j;
clrscr ();
/* Nhap du lieu vao ma tran a */
for ( i = 0; i < 3 ; i ++)
for ( j = 0; j < 3 ; j ++)
{
printf("\na [%d] [%d] ", i, j);
scanf("%d", &temp); /* nhap gian tiep vao bien trung gian temp */
a[i] [j] = temp;
/* Gan gia tri temp cho phan tu ma tran */
}
/* Xuat cac gia tri ma tran ra man hinh */
for ( i = 0; i < 3 ; i ++)
{
printf(" \n");
for ( j = 0; j < 3 ; j ++)
printf("%.2f", a[i] [j] );
}
getch();
}
```

4.2. MẢNG NHIỀU CHIỀU

Ta có thể khai báo một mảng nhiều chiều, mỗi chiều được xác định theo cặp dấu ngoặc vuông của nó.

Ví dụ ta khai báo mảng hai chiều sau: int a [4][3]; xác định ra một mảng hai chiều, chiều thứ nhất là hàng, chiều thứ hai là cột. Ở đây biến a có 4 hàng, mỗi hàng 3 cột (mảng hai chiều còn gọi là ma trận).

Có thể cho giá trị khởi đầu cho mảng nhiều chiều ngay sau khai báo. Việc lấy chỉ số cho mảng nhiều chiều cần cặp dấu ngoặc vuông cho từng chiều.

Mảng nhiều chiều là sự mở rộng của mảng một chiều. Do vậy, việc xử lý các mảng nhiều chiều trở nên phức tạp. Phép lấy địa chỉ như đã trình bày ở trên không áp dụng được cho các phần tử mảng nhiều chiều trừ mảng hai chiều các số nguyên. Vì vậy, để khắc phục hạn chế này của mảng, thường dùng con trỏ và các phép toán trên con trỏ.

4.3. CÁC VÍ DỤ VỀ MẢNG

Ví dụ 1: Viết chương trình sắp xếp một mảng gồm n số thực theo thứ tự tăng dần (hoặc giảm dần)

```
# include<stdio.h>;
# include<conio.h>;
# include<math.h>;
main()
{
clrscr();
float x[10];
float tam;
int i,j,n;
printf("\n nhap so hang n=");
scanf("%d",&n);
for (i=1;i<=n;i++)
{ printf("\n Nhap so thu %d=",i);
scanf("%f",&x[i]);
}
for (i=1; i<=n-1; i++)
```

```
for (j=i+1;j<=n;j++)
{
if (x[i]<x[j])
{
tam=x[i];
x[i]=x[j];
x[j]=tam;
}
}
printf("\n mang sau sap xep la");
for (i=1; i<=n; i++)
printf("\n %f",x[i]);
getch();
}
```

Ví dụ 2: Tìm phần tử âm đầu tiên của ma trận số thực a(m,n)

```
#include <stdio.h>
#include<conio.h>
#include<math.h>
main()
{ clrscr();
float a[20][20];
int i,j,n,m;
printf("\n Cho so hang ma tran m=");
scanf("%d",&m);
printf("\n Cho so cot ma tran n=");
scanf("%d",&n);
for (i=1;i<=m;i++)
for (j=1; j<=n;j++)
{ printf("Nhap A[%d,%d]=",i,j);
scanf("%f",&a[i][j]);
if (a[i][j]<0)
```

```
goto timthay;
}
printf("\n ma tran khong phan tu am");
goto kethuc;
timthay:
printf("n Phan tu am dau tien la a[%d,%d]= %f",i,j,a[i][j]);
kethuc:
;
getch();
}
```

Ví dụ 3: Tìm và đưa ra màn hình các giá trị lớn nhất và nhỏ nhất trên mỗi hàng của một ma trận cho trước.

```
# include<stdio.h>
#include<conio.h>
# include<math.h>
main()
{
clrscr();
int i,j,n,m;
float a[10][10],max,min;
printf("\n Nhap so hang cua ma tran=");
scanf("%d",&m);
printf("\n Nhap so cot cua ma tran=");
scanf("%d",&n);
for (i=1;i<=m;i++)
for (j=1;j<=n;j++)
{
printf("\n Nhap a[%d,%d]=",i,j);
scanf("%f",&a[i][j]);
}
for (i=1;i<=m;i++)
```

```
{
max=a[i][1];
min=a[i][1];
for (j=2;j<=n;j++)
{
if (a[i][j]>=max)
{ max=a[i][j];
}
if (a[i][j]<=min)
{ min=a[i][j];}
}
printf("\n Hang %d max=%f min=%f",i,max,min);
}
getch();
}
```

Ví dụ 4: Tính tích ma trận C = A.B

Phân tích: Nếu A là ma trận mxp phần tử và B là ma trận pxn phần tử thì ma trận tích C gồm mxn phần tử và được tính như sau:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Đây thực chất là bài toán tính tổng các số hạng.

Chương trình như sau:

```
# include <stdio.h>
# include <conio.h>
typedef float mt[10][10];
void main()
{
int i,j,k,m,n,p;
float x;
mt a,b,c;
```

```
/*Nhap du lieu ma tran A*/
printf("Ma tran A n hang va m cot:\n\n");
printf("So hang ma tran A n=");
scanf("%d",&n);
printf("So cot ma tran A m=");
scanf("%d",&m);
for (i=1;i<=n;i++)
for (j=1;j<=m;j++)
{ printf("\n a[%d][%d]=",i,j);
scanf("%f",&x);
a[i][j]=x;
}
printf("\n");
/*Nhap du lieu ma tran B*/
printf("Ma tran B n hang va m cot:\n\n");
printf("So hang ma tran B p=");
scanf("%d",&p);
printf("So cot ma tran B n=");
scanf("%d",&n);
for (i=1;i<=p;i++)
for (j=1;j<=n;j++)
{ printf("\n b[%d][%d]=",i,j);
scanf("%f",&x);
b[i][j]=x;
}
printf("\n");
/*Tinh tich c=a*b*/
for (i=1; i<=m;i++)
for (j=1; j<=p; j++)
{c[i][j]=0;
for (k=1; k<=n; k++)
```

```
c[i][j]+=a[i][k]*b[k][j];
}
/*Hien thi ket qua*/
printf("ma tran ket qua tich C\n\n");
for (i=1; i<=m; i++)
{
for (j=1; j<=p; j++)
printf("%4.2f",c[i][j]);
printf("\n");
}
getch();
return;}
```

4.4. CON TRỎ VÀ BIẾN ĐỘNG

Các biến nhớ có kiểu dữ liệu như đã trình bày ở trên là các biến tĩnh (*static*). Đây là các biến mà kích thước, kiểu và địa chỉ của nó không thay đổi, tồn tại trong suốt quá trình chạy chương trình.

Biến tĩnh có nhược điểm là khi chúng ta khai báo dư: gây lãng phí bộ nhớ, còn nếu khai báo thiếu thì không chạy được chương trình. Để khắc phục những nhược điểm trên trong các ngôn ngữ lập trình cấp cao thường sử dụng biến động (*dynamic*).

Biến động khác biến tĩnh, không sinh ra lúc bắt đầu chương trình mà sinh ra trong quá trình thực hiện chương trình. Nó có thể thay đổi được kích thước và địa chỉ vùng nhớ cấp phát lúc chạy chương trình và có thể giải phóng khi đã sử dụng xong để tiết kiệm bộ nhớ.

Tuy nhiên không thể truy nhập đến biến động vì nó không chứa địa chỉ nhất định. Người ta phải sử dụng một loại biến đặc biệt gọi là biến con trỏ (*pointer*). Biến con trỏ là loại biến chuyên dùng để chứa *địa chỉ của biến động* giúp người lập trình truy nhập đến biến động.

Khai báo biến con trỏ:

Con trỏ là biến để chứa địa chỉ của một biến nhớ khác hoặc một hàm. Vì có nhiều loại dữ liệu nên có nhiều kiểu con trỏ tương ứng.

- Con trỏ kiểu nguyên dùng để chứa địa chỉ các biến nguyên.

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

*Cú pháp: <tên kiểu> *<**Biến con trỏ**>*;

<Tên kiểu> là kiểu dữ liệu của biến động mà biến con trỏ lưu địa chỉ của nó.

<Biến con trỏ>: được đặt theo quy tắc đặt tên.

Ví dụ:

```
int *ip1, *ip2;
unsigned char *ucp;
double *dp;
float a=4.5, b= 6.0, *p;
```

Trong ví dụ này ip1, ip2 trỏ đến dữ liệu kiểu nguyên còn p trỏ đến dữ liệu kiểu thực.

Thông qua con trỏ, một đối tượng có thể được tham chiếu tới một cách gián tiếp. Cách dùng điển hình của con trỏ là trong việc tạo ra danh sách móc nối và quản lý các đối tượng được cấp phát bộ nhớ trong khi thực hiện chương trình.

Mọi con trỏ đều có một kiểu liên kết dữ liệu. Kiểu dữ liệu xác định ra kiểu của đối tượng dữ liệu mà con trỏ trỏ tới. Ví dụ một con trỏ kiểu int, sẽ trỏ tới một đối tượng có kiểu int.

Bộ nhớ được cấp phát cho một con trỏ chính là kích cỡ cần thiết để lưu trữ một địa chỉ bộ nhớ, tức là một con trỏ kiểu int và một con trỏ kiểu double sẽ có cùng kích cỡ. Kiểu liên kết của con trỏ xác định ra cách thức thông dịch nội dung và kích cỡ của vùng bộ nhớ mà nó trỏ tới.

Một con trỏ có thể được khởi đầu bằng giá trị địa chỉ của một đối tượng cùng kiểu. Để lấy được địa chỉ của một đối tượng, cần phải dùng một toán tử đặc biệt, gọi là toán tử lấy địa chỉ, có ký hiệu là dấu "&".

Để định vị con trỏ trỏ tới địa chỉ một biến nào đó, ta sử dụng cú pháp: *Tên con trỏ* = **&biến**

Ví dụ:

- Khai báo các biến con trỏ:

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

- Quy định vùng trỏ tới:

```
c = &y; //* gán địa chỉ của y cho con trỏ c *//
px = &x; // gán địa chỉ của x cho con trỏ px
```

Như vậy trong con trỏ c chứa địa chỉ của biến y và trong con trỏ px chứa địa chỉ của biến x.

```
t = &y;
```

Vì t là con trỏ kiểu float nó chỉ chứa được địa chỉ của biến float, trong khi đó y lại là biến kiểu int nên không thể chấp nhận được.

Nội dung của biến con trỏ là nội dung của biến mà nó trỏ tới (tức là biến động đang chứa dữ liệu).

Ví dụ: *px = 567 sẽ giống như x = 567

```
#include <stdio.h>
#include<stdlib.h>
#include<conio.h>
main()
{
int x, *px;
px =&x;
*px=567;
```

```
clrscr();
printf("Vi tri cua bien x la: %p \n",&x);
printf("Noi dung cua bien x la: %d\n",x);
printf("Vi tri cua bien con tro p la: %p \n",px);
printf("Noi dung cua bien con tro px la: %d\n",*px);
getch();
}
```

Con trỏ cũng có thể được khởi đầu bằng một con trỏ khác có cùng kiểu.

Việc gán giá trị một biến cho một con trỏ và ngược lại bao giờ cũng là không hợp lệ vì con trỏ luôn chứa giá trị kiểu địa chỉ.

Có thể gán giá trị 0 (hay còn gọi là giá trị NULL) cho một con trỏ có kiểu bất kỳ để chỉ ra rằng con trỏ hiện tại không trỏ tới đối tượng dữ liệu.

Để thâm nhập vào đối tượng dữ liệu mà con trỏ trỏ tới, cần áp dụng toán tử lấy giá trị, ký hiệu "*".

Tạo ra các biến nhớ động

Việc tạo ra một biến động (dynamic) được thực hiện nhờ các hàm sau:

1. Hàm malloc, *khai báo*:

void ***malloc**(<kích thước>);

void ***farmalloc**(<kích thước>);

Hàm malloc cho phép cấp phát một vùng nhớ có <*kích thước*> nhỏ hơn 64Kb, còn nếu muốn cấp phát vùng nhớ lớn hơn phải sử dụng farmalloc. Nếu máy tính còn đủ bộ nhớ để cấp phát thì các hàm này cho con trỏ trỏ tới vùng nhớ vừa được cấp, ngược lại hàm cho con trỏ rỗng NULL.

2. Hàm calloc, *khai báo*:

void ***calloc**(<số lượng>, <kích thước>);

void ***farcalloc**(<số lượng>, <kích thước>);

Hàm **calloc** xin cấp phát một vùng nhớ có dung lượng là <*số lượng*>x<*kích thước*>. Nếu máy tính còn đủ bộ nhớ để cấp phát thì các hàm này cho con trỏ trỏ tới vùng nhớ vừa được cấp, ngược lại hàm cho con trỏ rỗng NULL. Trường hợp muốn xin cấp vùng nhớ trên 64KB thì phải sử dụng **farcalloc**.

Giải phóng vùng nhớ

Khi không cần dùng đến các biến động có thể sử dụng hai hàm sau đây để thu hồi lại vùng nhớ đã cấp phát cho chúng.

Hàm: *void* **free**(*void* *<*khối*>) giải phóng vùng nhớ đã được cấp phát bằng *malloc, calloc*.

Hàm: *void* **farfree**(*void* *<*khối*>) giải phóng vùng nhớ đã được cấp phát bằng *farmalloc, farcalloc*.

4.5. CON TRỎ VÀ MẢNG

Trong C/C++, khái niệm mảng và con trỏ liên quan chặt chẽ với nhau. Các thao tác thực hiện thông qua chỉ số của mảng đều có thể thực hiện biến con trỏ. Các con trỏ trỏ vào các dữ liệu cùng một kiểu có thể gộp lại thành một mảng. Để khai báo mảng các con trỏ, ta viết:

<Tên kiểu> * ***<Tên mảng>*** [<*kích thước mảng*>];

Ví dụ: int *a_nguyen[10]; char *b_ten[20];

Với dữ liệu kiểu mảng, tên gọi của mảng xác định địa chỉ trong bộ nhớ của phần tử đầu tiên nằm trong mảng.

Ta xét một biến mảng: *char buf [8];* khai báo một mảng có 8 phần tử có kiểu char.

Ta có: buf ⇔ &buf [0]

Vì việc áp dụng toán tử lấy địa chỉ cho một đối tượng dữ liệu sẽ cho lại con trỏ thuộc kiểu đối tượng đó. Trong trường hợp này, đối tượng có kiểu char, có nghĩa là buf (tên gọi của mảng) phải cho lại một giá trị kiểu char*. Hay ta có thể viết như sau:

char *pBuf = buf;

pBuf và buf là tương đương vì mỗi tên gọi đều chỉ ra phần tử đầu tiên của mảng.

Để trỏ đến địa chỉ phần tử tiếp theo, có thể có hai cách:

```
pBuf + 1;
hoặc &buf [1];
```

Lấy địa chỉ phần tử thứ i:

```
pBuf + i;
hoặc &buf [i];
```

Và ta có thể thâm nhập phần tử dữ liệu đầu tiên:

```
*pBuf;
hoặc  buf [0];
```

Thâm nhập phần tử dữ liệu thứ i:

```
*(pBuf + i );
hoặc  buf [ i ];
```

Ta xét khai báo sau đây: float t, *p, A[9];

Câu lệnh này khai báo biến thực t, con trỏ thực p và mảng A gồm 9 phần tử: A[0], A[1],... A[9].

Lệnh gán p = &A[0]; sẽ cho biến p trỏ đến phần tử đầu tiên của mảng A. Chúng ta cần nhớ rằng việc sử dụng *&<tên biến>* là cho ta địa chỉ của biến. Bây giờ nếu ta viết t=*p sẽ sao chép nội dung của A[0] vào biến t.

Một con trỏ và một mảng khác nhau về phương pháp cấp bộ nhớ:

Với một mảng dữ liệu, bộ nhớ cấp phát cho kiểu mảng là một khối bộ nhớ liên tục, tên gọi của mảng chứa địa chỉ đầu của vùng nhớ và nó được xử lý như một hằng.

Việc tăng kích cỡ của mảng là không thể thực hiện được vì số phần tử của mảng đã được xác định ngay từ đầu.

Nhưng dữ liệu kiểu con trỏ là để lưu giá trị của một địa chỉ bộ nhớ. Người dùng trước hết phải đặt con trỏ trỏ vào đối tượng đã được

cấp phát bộ nhớ trước đó trước khi sử dụng chúng một cách an toàn. Điều này thường được thực hiện bằng cách cấp phát bộ nhớ trong khi chương trình đang chạy thông qua việc dùng toán tử *new*. Hủy bỏ vùng nhớ được cấp phát bằng new nhờ toán tử *delete*.

Chúng ta xét ví dụ sau:

Ví dụ: Lập trình sử dụng con trỏ để sắp xếp một dãy số theo thứ tự tăng dần.

```
# include<stdio.h>
# include<conio.h>
short i,j,n;
float tg, *p;
void main()
{
clrscr();
printf("Nhap so phan tu mang n=");
scanf("%d",&n);
for (i=1; i<=n; i++)
{
printf("Nhap so thu %d=",i);
scanf("%f",p+i);
}
printf("\n");
for (i=1; i<=n-1; i++)
for (j=i+1; j<=n; j++)
if (*(p+i)>*(p+j))
{
tg=*(p+i);
*(p+i)=*(p+j);
*(p+j)=tg;
}
```

```
printf("day so sap xep tang dan nhu sau:\n\n");
for (i=1; i<=n; i++)
printf("%6.1f",*(p+i));
printf("\n");
getch();
}
```

Trong chương trình trên chúng ta sử dụng *(p+i) là để truy nhập đến phần tử thứ i của mảng A[i].

Bài tập chương 4

1. Đoạn chương trình sau dịch không có lỗi, nhưng nó vẫn sai. Hãy tìm ra chỗ sai đó: char buf [ ] = "Xau kiem thu";

```
void main ()
{
 char *ptr = 0;
for (int i=0; buf [i] != '\0'; i++)
      ptr [i] = buf [i];
}
```

2. Trong các khai báo mảng sau, khai báo nào là không hợp lệ? Tại sao?

```
int    getSize();
int   bufSize = 1024;
```

   a) int ia[bufSize];

   b) int ia[getSize()];

   c) int ia[4*7 - 14];

3. Trong các khai báo con trỏ sau, khai báo nào là không hợp lệ? Tại sao?

```
int ival = 1024;
int *iptr;
double *dptr;
```

   a) ival = *iptr;

   b) ival = iptr;

   c) *iptr = ival;

   d) iptr = ival;

   e) *iptr = & ival;

   f) iptr = &ival;

   g) dptr = iptr;

   h) dptr = *iptr;

4. Viết chương trình lưu trữ tuổi của 6 người bạn trong mảng một chiều bằng toán tử gán. Hiển thị tuổi trên màn hình theo thứ tự tăng dần.
5. Viết chương trình lưu trữ và hiển thị các số từ 1 đến 21 trong bảng 3x7.
6. Viết chương trình lưu trữ điểm học trung bình của học kỳ vừa qua của 5 lớp học, mỗi lớp có 20 sinh viên. Sử dụng cin để nhập điểm, hiển thị bảng điểm trên màn hình.
7. Viết chương trình lưu trữ tên thành viên trong gia đình trong bảng hai chiều.
8. Viết chương trình nhập dãy số từ bàn phím. Sử dụng con trỏ để tìm xem dãy số đó có chứa số 0 hay không?
9. Viết chương trình nhập dãy số từ bàn phím. Sử dụng con trỏ để tìm ra số lớn nhất và số nhỏ nhất vừa nhập. Hiển thị lại dãy số vừa nhập, số lớn nhất và số nhỏ nhất trong dãy đó.
10. Lập chương trình nhập mảng một chiều, tính tổng các phần tử dùng con trỏ.
11. Sử dụng việc cấp phát bộ nhớ động (không sử dụng mảng) để nhập hai dãy số có số phần tử mỗi dãy được nhập từ bàn phím. Sau đó in ra tổng của mỗi dãy và tổng của hai dãy này.
12. Viết chương trình nhập vào mảng, hãy xuất ra màn hình:
    - Dòng 1: phần tử âm lớn nhất của mảng.
    - Dòng 2: phần tử dương nhỏ nhất của mảng.
    - Dòng 3: Tổng các phần tử có căn bậc hai nguyên.
    - Dòng 4: Gồm các số lẻ và số các số lẻ.
    - Dòng 5: Gồm các số chẵn và số các số chẵn

Chương 5

CẤU TRÚC VÀ VÀO/RA TỆP

Trong các bài toán thực tế chúng ta hay gặp các kiểu dữ liệu có nhiều thành phần khác nhau chứ không nhất định là thuộc kiểu dữ liệu nguyên hay thực hoặc ký tự. Ví dụ, trong hồ sơ cán bộ thường bao gồm các thuộc tính: họ và tên, ngày sinh, quê quán, giới tính, lương, địa chỉ... cụ thể:

- Họ và tên: dữ liệu kiểu ký tự.
- Lương: dữ liệu kiểu số thực
- Ngày sinh: dữ liệu kiểu ngày/tháng/năm...

Mỗi một thuộc tính này ta gọi là một trường, còn tập các thuộc tính trên sẽ tạo thành một hồ sơ và trong C/C++ ta gọi là một kiểu cấu trúc.

Trong chương này chúng ta sẽ nghiên cứu cách sử dụng kiểu cấu trúc trong chương trình. Nội dung chương này sẽ trình bày các vấn đề sau đây:

- Cấu trúc, hợp nhất
- Mảng cấu trúc
- Các thao tác vào/ra tệp

5.1. CẤU TRÚC, HỢP NHẤT

5.1.1. Cấu trúc

Cấu trúc là tập hợp của một hay nhiều biến, chúng có thể khác kiểu nhau, được nhóm lại dưới một tên duy nhất để tiện xử lý. Trong ngôn ngữ Pascal cấu trúc được gọi là bản ghi (record).

Như vậy, dữ liệu kiểu cấu trúc cho phép lập trình viên tổ chức gộp nhiều kiểu dữ liệu khác nhau trong một cấu trúc đơn, nó là một kiểu dữ liệu dẫn xuất từ các kiểu dữ liệu đã được định nghĩa.

Ví dụ cấu trúc dữ liệu mô tả nhân viên gồm họ tên, giới tính, tuổi...

Cú pháp khai báo kiểu cấu trúc:

```
struct <Tên kiểu>
{
        <kiểu dữ liệu 1 > <trường 1>;
        <kiểu dữ liệu 2> <trường >2;
       ...
        <kiểu dữ liệu n> <trường n>;
};
```

struct: từ khóa

<kiểu 1>, <kiểu 2>, <kiểu n>: là các kiểu dữ liệu của các thành phần cấu trúc.

<trường1>, <trường 2>,.. <trường n>: là tên các thành phần cấu trúc.

Ví dụ: Để mô tả kiểu dữ liệu hồ sơ cán bộ ta có thể viết:

```
struct hoso
{
char hoten[20];
int namsinh;
float luong;
char gioitinh;
}
struct hoso canbo1, canbo2;
```

Khi khai báo cấu trúc như trên tức là ta định nghĩa ra một kiểu dữ liệu mới, chưa có bộ nhớ cho cấu trúc. Bộ nhớ được cấp phát khi ta khai báo biến dữ liệu có kiểu cấu trúc.

Cú pháp khai báo biến kiểu cấu trúc:

<*kiểu cấu trúc*> <***tên biến cấu trúc***>;

Để có thể khai báo biến kiểu cấu trúc, trước hết ta phải khai báo cấu trúc, tức là hiển nhiên ta phải khai báo kiểu biến trước khi khai báo biến.

Sau khi khai báo biến kiểu cấu trúc, ta có thể thâm nhập các thành phần cấu trúc và thực hiện các thao tác dữ liệu cần thiết bằng toán tử "." như sau: <tên biến cấu trúc>.<tên trường>

Ví dụ: canbo1.hoten="nguyen van A"; canbo1.luong=890.5;

Phép gán với kiểu cấu trúc

Với dữ liệu kiểu cấu trúc, ta cũng có thể thực hiện các phép toán:

- *Gán cấu trúc*: Có thể gán giá trị hai biến có cùng kiểu cấu trúc.

Ví dụ: canbo1 = canbo2;

Khi thực hiện phép gán, các thành phần dữ liệu tương ứng sẽ được tự động sao chép sang nhau.

- *Sử dụng biến cấu trúc* như là một đối của hàm.

Ví dụ 1: Hãy xây dựng một cấu trúc hồ sơ để nhập vào danh sách: Họ và tên, ngày vào trường, ngày tốt nghiệp. Trong đó yêu cầu ngày vào trường và ngày tốt nghiệp là kiểu dữ liệu ngày tháng (bao gồm: ngày, tháng, năm).

Chương trình như sau:

```
/*Chuong trinh minh hoa du lieu kieu cau truc*/
#include <stdio.h>
#include <conio.h>
#define maxTen 40 /*Ho ten dai nhat 40 ky tu*/
struct ngaythang
{
int ngay,thang,nam;};
```

```
struct hoso
{
char Ten[maxTen];
struct ngaythang ngayvao; /*ngay vao truong*/
struct ngaythang ngayra; /*ngay tot nghiep truong*/
};
void VaoDS(struct hoso *ds);
void main()
{
struct hoso nhvat;
clrscr();
VaoDS(&nhvat);
printf("\nHo ten:%s\nNgay vao:%d%d%d\nNgay tot nghiep
%d%d%d",nhvat.Ten,nhvat.ngayvao.ngay,nhvat.ngayvao.thang,
nhvat.ngayvao.nam,nhvat.ngayra.ngay,nhvat.ngayra.thang,
nhvat.ngayra.nam);
getch();
return;}
void VaoDS(struct hoso *ds)
{
char traloi;
printf("\nCho ho va ten:");
gets(ds->Ten);
printf("\n Ngay vao truong:");
scanf("%d%d%d",&ds->ngayvao.ngay,&ds->ngayvao.thang,
&ds->ngayvao.nam);
printf("\nNgay = Ngay tot nghiep?y/n:");
getchar();
traloi=getchar();
if (traloi=='y') ds->ngayra=ds->ngayvao;
else
{printf("\nNgay tot nghiep:");
```

```
scanf("%d%d%d",&ds->ngayra.ngay,&ds->ngayra.thang,
&ds->ngayra.nam);
}
}
```

Trong chương trình trên, chúng ta đã khai báo kiểu dữ liệu *ngaythang* là kiểu dữ liệu cấu trúc gồm các trường *ngay*, *thang* và *nam*.

Kiểu dữ liệu **hoso** (Hồ sơ) bao gồm:

trường *tên* (kiểu ký tự);

ngayvao, *ngayra* đều thuộc kiểu cấu trúc dữ liệu *ngaythang* ta đã tạo ra ở trên.

Khai báo *void VaoDS* là khai báo hàm không định kiểu (giống thủ tục trong Pascal) để thực hiện công việc nhập danh sách cán bộ. Hàm này được khai báo sau *return* của chương trình chính. Trong chương trình này chúng ta đã sử dụng các cấu trúc lồng nhau, tức là một trường trong kiểu cấu trúc (*ngayvao, ngayra*) lại là một kiểu cấu trúc khác (*ngaythang*).

Để truy nhập đến các trường như vậy chúng ta sử dụng lệnh:

<Tên biến bản ghi>.*<Tên trường ngoài>*.*<Tên trường trong>*

Chúng ta cũng đã sử dụng đến biến con trỏ cấu trúc. Biến cấu trúc là một biến trong bộ nhớ nên có thể có con trỏ trỏ đến nó. Khai báo biến con trỏ *trỏ* đến cấu trúc như sau:

struct *<tên kiểu cấu trúc>* * *<Biến con trỏ>*;

Ví dụ: struct cb *tro_hoso;

Việc truy cập vào các trường của biến cấu trúc thông qua **con trỏ cấu trúc** được viết như sau:

<Biến con trỏ cấu trúc> **->** *<Tên trường>*

Ví dụ: gets(ds->Ten);

Dấu -> ở đây là bắt buộc không được dùng dấu chấm (.).

Ví dụ 2: Viết chương trình nhập vào họ tên và điểm thi của sinh viên một lớp, sau đó in ra danh sách những sinh viên đạt điểm khá từ 7,0 trở lên.

```
 #include <stdio.h>
#include <conio.h>
# include <string.h>
#define maxTen 40 /*Ho ten dai nhat 40 ky tu*/
 struct hoc_sinh
{
char   ho_ten[maxTen];
float diem;
} dshs[100];
 int n ; / * So luong sinh vien lop * /
int k; / * So luong sinh vien lop dat diem kha * /
int i;
char name[20]; float d;
/* Nhap thong tin sinh vien */
n = 0;
do {
clrscr();
printf(" Nhap thong tin sinh thu %d\n", n+1) ;
 printf ("Ho va ten : "); gets(name);
if (strcmp(name, " ")!=0)
{
strcpy(dshs[n].ho_ten.name);
printf(" Diem : ");
scanf("%d", &d);
dshs[n++].diem = d;
}
}  while (strcmp(name," ")!=0);
```

```
if (!n) /* Khong nhap duoc sinh vien nao */
{
printf ("Chua nhap hoc sinh nao \n");
return 0;
}
else
{
k =0;
printf ("Danh sach sinh vien dat diem kha \n");
 for (i = 0; i<n; i++)
if (dshs[i].diem >= 7)
printf("%d.%s%6.3f", ++k, dshs[i].ho_ten, dshs[i].diem);
if (!k)
printf ("Khong co sinh vien nao dat diem kha \n"
}
getch ();
}
```

5.1.2. Hợp nhất

Ngoài kiểu dữ liệu cấu trúc, ngôn ngữ C/C++ còn cho phép người lập trình tạo ra một kiểu dữ liệu dẫn xuất khác, đó là kiểu hợp nhất (*union*). Kiểu dữ liệu union cũng cho phép người dùng gộp nhiều kiểu dữ liệu trong một tổ chức nhất định như kiểu dữ liệu cấu trúc.

Cú pháp:

```
union Uname
{
        type1 id_list1;
        type2 id_list2;
        ...
        typen id_listn;
};
```

Trong đó:

+ *union*: từ khóa hợp nhất

+ **Uname**: tên hợp nhất

+ type1, type2, typen: các kiểu dữ liệu của các thành phần hợp nhất

+ id_list1, id_list2, id_listn: tên các thành phần hợp nhất

Điểm khác biệt của *union* với kiểu dữ liệu cấu trúc (struct) là: các thành phần của biến cấu trúc được cấp phát các vùng nhớ khác nhau, còn các thành phần của biến union được cấp phát chung một vùng nhớ. Vùng bộ nhớ của *union* thường có độ lớn bằng độ lớn vùng bộ nhớ để lưu trữ thành phần dữ liệu có kích cỡ lớn nhất trong *union*.

Cách dùng kiểu dữ liệu hợp nhất:

Kiểu dữ liệu hợp nhất cũng được dùng như kiểu dữ liệu cấu trúc. Trước khi khai báo biến có kiểu hợp nhất, ta phải khai báo hợp nhất như là một kiểu và các thành phần của nó.

Ta có thể truy nhập vào các thành phần của hợp nhất bằng các toán tử " . "

Ví dụ:

```
struct  ng{
int ngay;
int thang;
int nam;
};
struct diachi {
int sonha;
char *tenpho;
};
union u{
struct ng date;
```

```
struct diachi address;
} diachi_ngaysinh;
```

Trong ví dụ này chúng ta đã khai báo một union tên là u có các thành phần là cấu trúc.

5.2. MẢNG CẤU TRÚC

Cũng như các kiểu dữ liệu đơn khác, ta cũng có thể dùng kiểu dữ liệu là một mảng các cấu trúc như là một tập hợp các cấu trúc. Mảng mà các thành phần có kiểu cấu trúc được gọi là mảng cấu trúc. Khai báo một mảng có cấu trúc hoàn toàn tương tự như đối với khai báo một mảng bình thường nhưng thay cho tên kiểu dữ liệu bình thường là tên kiểu dữ liệu cấu trúc.

Cú pháp:

<*Kiểu cấu trúc*> **Tên mảng** [*số phần tử*];

Ví dụ:

struct hoc_sinh dshs[100]; /*hoc_sinh là kieu cau truc */

Với hoc_sinh là kiểu cấu trúc được khai báo như sau:

```
struct hoc_sinh
{
char ho_ten[20];
float diem;
} hs, dshs[100];
```

Mảng cấu trúc cũng có đầy đủ các tính chất của một mảng dữ liệu, tức là ta có thể truy nhập đến từng cấu trúc theo chỉ số mảng. Muốn truy nhập vào các thành phần cấu trúc ta lại dùng toán tử thành phần " . ".

Truy nhập đến các thành phần cấu trúc theo thứ tự:

Dùng toán tử [ ] để xác định thứ tự của phần tử cấu trúc trong mảng cấu trúc. Dùng toán tử " . " để truy nhập vào thành phần của cấu trúc.

Khi dùng mảng hay mảng cấu trúc, người lập trình luôn phải chú ý rằng số các phần tử của mảng là có giới hạn *size* khi khai báo. Khi thực hiện các phép toán với các phần tử cấu trúc, luôn phải kiểm tra thứ tự giới hạn nằm trong kích thước của mảng cấu trúc.

5.3. CÁC THAO TÁC VÀO/RA TỆP

Các tệp dữ liệu là giải pháp cần thiết cho việc lưu trữ dữ liệu của các chương trình khi phải xử lý dữ liệu đưa vào hoặc kết quả quá trình xử lý có khối lượng lớn không thể chứa hết trong bộ nhớ trong của máy tính. Hơn nữa, khi chương trình kết thúc, dữ liệu trong bộ nhớ trong của máy sẽ bị xóa nên ta không thể lưu trữ dữ liệu lâu dài để sử dụng nhiều lần. Ngoài ra, các tệp dữ liệu còn đóng vai trò là "hòm thư" phục vụ cho việc trao đổi dữ liệu giữa các tiến trình trong hệ thống. Việc vào/ra dữ liệu trên tệp luôn phải tuân thủ theo một quy tắc chung là phải làm việc tuần tự theo ba bước: Mở tệp (Open) → Xử lý (đọc/ghi) → Đóng tệp (close).

Trong ngôn ngữ C/C++, có thể xử lý việc vào/ra tệp dữ liệu theo hai cách: Sử dụng lớp ifstream và ofstream hoặc sử dụng hệ thống thư viện vào ra chuẩn.

Trong chương trình C, kiểu tệp được khai báo như sau:

FILE *<Biến trỏ tệp>;

Ví dụ: FILE *f, *tep;

f = fopen("data.dat", "wt");

5.3.1. Sử dụng lớp ifstream và ofstream

Ngôn ngữ C/C++ cung cấp khái niệm luồng dữ liệu (stream). Đây là khái niệm trừu tượng làm trung gian giữa chương trình và các thiết bị vào - ra.

Lớp ifstream và ofstream được định nghĩa trong thư viện fstream, nên khi sử dụng phải có dòng #include <fstream.h> ở đầu tệp nguồn chương trình.

Một số hàm thông dụng trong lớp ifstream và ofstream.

a. Các hàm mở tệp

void ofstream::open(const char *filename,int mode,int nProt);

Mở một tệp để ghi dữ liệu, với tham số:

int mode: xác định mode mở tệp, có giá trị mặc định là ios::out.

void ifstream::open(char *filename,int mode,int nProt);

Mở một tệp để đọc dữ liệu, với tham số:

int mode: xác định mode mở tệp, có giá trị mặc định là ios::in.

const char *filename: xác định tên của tệp.

int nProf: xác định mode sử dụng chung.

b. Hàm đóng tệp

void ofstream::close();

void ifstream::close();

Đóng tệp dữ liệu đã mở để ghi hoặc đọc dữ liệu.

c. Các hàm xử lý trên tệp

ifstream::get(char ch);

Đọc vào một ký tự từ tệp và đưa vào char ch.

ifstream::getline(char *str,int num_bytes,char='\n');

Đọc các ký tự vào vùng đệm str cho tới khi gặp ký tự phân cách (tham số thứ 3) hoặc ký hiệu kết thúc tệp hoặc khi số ký tự đã được đọc bằng num_bytes-1. Ký hiệu kết thúc xâu ('\0') được tự động thêm vào cuối xâu. Có lỗi khi không ký tự nào được đọc (fail()!=0).

char *str: xâu ký tự đọc được từ tệp

int num_bytes: độ lớn của xâu str.

char= '\n': ký tự phân cách.

ofstream::put(char ch);

Ghi vào tệp một ký tự char ch.

Toán tử <<

Ghi giá trị một biến nằm bên phải toán tử vào tệp dữ liệu đã mở bằng ofstream::open

Toán tử >>

Đọc dữ liệu từ tệp ra một biến nằm bên phải toán tử.

ifstream::eof()

Kiểm tra cuối tệp dữ liệu

ifstream::read(char* str, int n);

ifstream::read(signed char* str, int n);

ifstream::read(unsigned char* str, int n);

Đọc n ký tự từ tệp dữ liệu vào vùng đệm str. Sử dụng hàm thành phần gcount() để biết số ký tự thực sự được đọc.

ofstream outf;

ofstream:: write(const char*, int n);

ofstream:: write(const signed char*, int n);

ofstream:: write(const unsigned char*, int n);

Ghi n ký tự từ vùng đệm str vào tệp dữ liệu.

5.3.2. Sử dụng hệ thống thư viện vào ra chuẩn

Khi thao tác với tệp dữ liệu sử dụng các hàm thư viện vào/ra chuẩn, ta sử dụng kiểu FILE là kiểu dữ liệu sử dụng cho việc thao tác với tệp dữ liệu.

Phải thêm dòng lệnh #include <stdio.h> vào đầu tệp chương trình nguồn khi xâm nhập tệp.

a. Hàm mở tệp: fopen()

Cú pháp:

FILE * fopen(const char *fname,const char *mode);

Trong đó:

const char *fname: tên tệp dữ liệu sẽ mở.

const char *mode:

r: Mở để đọc;

w: Tạo một tệp để ghi dữ liệu (nếu tệp có tên đã tồn tại sẽ bị ghi đè);

a: Mở để ghi thêm vào cuối tệp (tạo mới nếu tệp chưa tồn tại);

r+: Mở tệp đã tồn tại để cập nhật (reading & writing).

w+: Tạo một tệp mới để cập nhật (reading & writing). Nếu tệp đã tồn tại sẽ bị ghi đè.

a+: Mở để cập nhật vào cuối tệp (tạo mới nếu tệp chưa tồn tại).

Hàm này trả lại con trỏ tới tệp được mở, NULL nếu không mở được tệp.

b. Hàm đóng tệp: fclose()

Cú pháp:

```
int fclose(FILE *fp);
```

Đóng tệp đã được mở. Trả lại 0 nếu thành công, EOF nếu có lỗi.

c. Các thao tác tệp dữ liệu

Đọc ký tự: fgetc()

Cú pháp:

```
int fgetc(FILE *fp);
```

Đọc một ký tự. Trả về giá trị (mã ASSCI) của ký tự nếu thành công, EOF nếu có lỗi.

Đọc xâu ký tự: fgets()

Cú pháp:

```
char *fgets(char *s,int n,FILE *stream);
```

Đọc xâu ký tự cho đến khi gặp ký tự xuống dòng hoặc đã đọc được n -1 ký tự.

Trả về con trỏ tới xâu kết quả s nếu thành công, NULL nếu có lỗi.

Đọc dữ liệu: fread()

Cú pháp:

```
size_t fread(void *ptr,size_t size,size_t n,FILE *stream);
```

Đọc dữ liệu từ tệp .fread đọc n phần tử, mỗi phần tử có độ lớn size bytes (tổng số bytes = n*size) từ tệp vào vùng nhớ được xác định bởi ptr. Tổng số byte đọc một lần là < 64kb.

Trả về số phần tử thực sự đọc được.

Ghi ký tự vào tệp: fputc()

Cú pháp:

```
int fputc(int c,FILE *stream);
```

Trả về c nếu thành công, EOF nếu không thành công.

Ghi xâu ký tự vào tệp: fputs()

Cú pháp:

```
int fputs(const char *s,FILE *stream);
```

Ghi xâu s vào tệp (không ghi ký tự '\0' và '\n')

Trả về giá trị >0 nếu thành công, EOF nếu thất bại.

Ghi dữ liệu vào tệp: fwrite()

Cú pháp:

```
size_t fwrite(const void *ptr,size_t size,size_t n,FILE *stream);
```

Ghi bổ sung n phần tử, mỗi phần tử có độ lớn size bytes vào tệp.

Trả về số phần tử thực sự được ghi.

Dịch chuyển con trỏ tệp: fseek()

Cú pháp:

int fseek(FILE *stream, long offset, int whence);

trong đó:

whence: SEEK_SET0 offset tính từ đầu tệp

SEEK_CUR 1 offset tính từ vị trí hiện hành

SEEK_END 2 offset tính từ cuối tệp

fseek trả về 0 nếu thành công, khác 0 nếu thất bại.

Xét ví dụ sau:

Ví dụ 1: Lập chương trình đọc từ file dữ liệu kiểu văn bản vào một ma trận. Cấu trúc tệp dữ liệu gồm: dòng đầu là hai số nguyên cho biết số hàng và cột của ma trận; các hàng tiếp theo cho biết các phần tử ma trận theo hàng. Ví dụ:

2 3

4.5 8.0 9

3 5 7

Chương trình như sau:

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#define Mmax 10
#define Nmax 20
char xau[20];
FILE *f;
int i,j,m,n;
float A[Mmax][Nmax];
void main()
{
clrscr();
```

```
printf("Cho biet ten tep:");
fflush(stdin);
gets(xau);
if (!(f=fopen(xau,"r+")))
{
puts("khong mo duoc tep!");
getch();
/*exit(1);*/
}
fscanf(f,"%d%d",&m,&n);
for (i=0; i<m; ++i)
{for (j=0;j<n;++j)
{
fscanf(f,"%f",&A[i][j]);
printf("%8.3f",A[i][j]);
}
printf("\n");
}
getch();
}
```

Ví dụ 2: Viết chương trình cho phép tìm một tệp xem có trên đĩa không bằng cách nhập vào tên tệp cần tìm kiếm và thông báo kết quả.

```
#include<stdio.h>
#include<conio.h>
# include<iostream.h>
main()
{    clrscr();
     FILE *f;
     char duong_dan[79];
     printf("\n Chuong trinh tim mot tep tren dia");
     printf("\n Nhap ten tep hoac ten duong dan");
```

```
        printf("\n Nhap ten tep can tim: ");
        gets(duong_dan);
        f=fopen(duong_dan,"r");
        if (f==NULL)
                puts("\n Tep khong co tren dia");
        else
        puts("\t Co tep nay tren dia");
        fclose(f);
        printf("\n Nhan phim bat ky de ket thuc");
        getch();
}
```

Bài tập chương 5

1. Sự khác nhau giữa cấu trúc và trường?
2. Có thể khai báo cấu trúc và biến cấu trúc là tổng thể hay riêng?
3. Viết cấu trúc trong chương trình để lưu trữ băng hình trong một cửa hàng. Cấu trúc phải có khả năng chứa tên băng, độ dài, giá khi mua, giá cho thuê và ngày tháng sản xuất phim trong băng.
4. Viết chương trình nhập các dữ liệu cho cấu trúc trong bài tập 3 và hiển thị danh sách băng hình của cửa hàng trên màn hình.
5. Viết chương trình lưu trữ danh mục điện thoại của sinh viên trong lớp, bao gồm họ và tên, địa chỉ, số điện thoại.
6. Bổ sung sắp xếp họ và tên trong danh sách điện thoại (ví dụ 5) theo thứ tự abc.
7. Viết chương trình hiển thị tệp autoexec.bat trong máy của bạn.
8. Viết chương trình hiển thị 30 byte đầu tiên của tệp command.com trong máy tính.
9. Viết chương trình sao lưu tệp autoexec.bat thành autoexec.bak và tệp config.sys thành config.bak trong máy tính.
10. Viết chương trình lưu trữ danh sách điện thoại được tạo ra từ ví dụ 5. Sau đó đọc chúng để hiển thị lại trên màn hình.
11. Viết chương trình hiển thị 10 ký tự cuối cùng của tệp command.com trong máy tính.
12. Viết chương trình nhập dữ liệu và ghi vào đĩa một dãy số nguyên bất kỳ. Sắp xếp dãy số đó theo thứ tự giảm dần rồi lại ghi vào đĩa.
13. Viết chương trình nhập dữ liệu ghi vào đĩa các thành phần: HO, TEN, TUOI, CHUCVU, LUONG thành tệp LUONG.DAT, khi nào không muốn nhập nữa nhấn phím ESC.

14. Viết chương trình liệt kê tên các tệp có trong một thư mục và trong tất cả các thư mục con của nó.

15. Viết chương trình gồm hai chức năng:

- Nhập và lưu các hệ số a, b, c của các phương trình bậc 2 vào 1 tệp.

- Tìm nghiệm của các phương trình bậc 2 có hệ số a, b, c được lưu trong tệp trên và lưu kết quả vào một tệp khác.

Chương 6

KỸ THUẬT ĐỒ HỌA TRONG C/C++

Ngay từ khi máy tính ra đời người ta đã sử dụng để tạo đồ họa vì không có sản phẩm của máy tính nào mà không liên quan đến đồ họa. Trong những thập kỷ 70 và 80, người ta sử dụng cách hiển thị đồ họa cổ điển bằng các chương trình viết bằng BASIC hoặc FORTRAN, thông tin hoặc dữ kiện trôi nhanh trên màn hình rồi biến mất nên có nhiều nhược điểm hạn chế đến việc ứng dụng đồ họa. Ngày nay do tốc độ phát triển nhanh của phần cứng, có nhiều thiết bị chuyên dùng cho đồ họa nên kỹ thuật đồ họa phát triển vượt bậc. Ngôn ngữ C/C++ có nhiều ưu thế về đồ họa.

Nội dung chương này sẽ trình bày các vấn đề sau đây:

- Giới thiệu hệ thống đồ họa
- Những cơ sở đồ họa của máy PC
- Mô hình đồ họa trong Turbo C/C++
- Màn hình hiển thị và hệ tọa độ
- Chế độ màn hình
- Xây dựng một chương trình đồ họa
- Một số hàm đồ họa cơ bản

6.1. GIỚI THIỆU HỆ THỐNG ĐỒ HỌA

Kỹ thuật đồ họa là một trong những kỹ thuật khá phức tạp và ngôn ngữ C++ được sử dụng trong kỹ thuật đồ họa rất thành công. Có thể nói đồ họa là một trong những sự thể hiện sức mạnh của C++.

Hình ảnh là sản phẩm của kỹ thuật đồ họa. Quá trình tạo ảnh là việc đặt dữ liệu vào dạng trung gian sau đó hiển thị chúng trên màn hình.

Các thành phần cơ bản (nguyên sơ) tạo nên đồ họa bao gồm điểm, đường, đa giác, hình tròn, văn bản...

Mỗi phần tử cơ bản được gắn thêm thuộc tính vẽ (kiểu vẽ, màu vẽ, kiểu chữ, kiểu tô màu).

Trong chương trình ứng dụng C++ (chương trình thể hiện bằng ngôn ngữ C++) các thuộc tính vẽ được xác định trước các phần tử sẽ được vẽ.

Hệ tọa độ vẽ trên màn hình: sử dụng hệ tọa độ Đề-các hai chiều, chiều x tăng về bên phải, chiều y tăng lên trên. Gốc tọa độ có giá trị (0, 0) nằm trên góc trên trái màn hình.

Kiểu biến biểu diễn cho tọa độ: nguyên hoặc dấu phảy động. Tuy nhiên trên máy tính biến nguyên sẽ được sử dụng để biểu diễn tọa độ vì như thế mới đảm bảo tốc độ xử lý dữ liệu.

6.2. NHỮNG CƠ SỞ ĐỒ HỌA CỦA MÁY PC

Thiết bị hiển thị thông tin ra màn hình là thiết bị hiển thị thông dụng nhất. Trên máy tính IBM PC và các máy tương thích, đầu ra của màn hình phụ thuộc vào hai thành phần: thiết bị ghép nối màn hình (display adapter) và màn hình (display monitor).

Adapter: là vỉ mạch cắm trong máy tính, trên đó có bộ nhớ màn hình

Monitor: là màn hình để hiển thị thông tin (kết quả ra) dưới dạng văn bản hay hình ảnh. Mỗi màn hình có thể hiển thị một mảng (hai chiều) các điểm, mỗi điểm được gọi là một pixel. Độ phân giải của màn hình là số lượng điểm theo chiều ngang và chiều đứng của màn hình.

Khi máy tính muốn hiển thị một hình ảnh, hình ảnh này sẽ được thể hiện trên một mảng điểm và mảng điểm này sẽ được gửi tới monitor thông qua adapter. Ví dụ: khi máy tính muốn hiển thị các ký

tự, mỗi ký tự được xây dựng bằng một mẫu vẽ bởi các pixel và các mẫu (ký tự) này sẽ được gửi tới monitor thông qua *adapter*.

Mỗi loại adapter có thể đưa dữ liệu ra theo một hoặc hai chế độ sau: Văn bản (text) hay đồ họa (graphics).

6.2.1. Các loại Adapter và Monitor

Đối với các máy tính IBM PC

Adapter có loại chuẩn sau:

1. MDA: Monochrome Display Adapter

2. CGA: Color Graphics Adapter

3. EGA: Enhanced Graphics Adapter

4. VGA: Video Graphics Array

5. SVGA: Super VGA

Ngoài ra còn một loại thứ sáu: HGC (Hercules Graphics Adapter) có khả năng hiển thị chế độ đồ họa trong màn hình monochrome.

Monitor: Monochrome display, color display, enhanced color display.

Mối tương quan giữa Adapter và Monitor:

MDA: có thể hiển thị văn bản (text) trên 25 dòng (rows), mỗi dòng 80 ký tự (columns) trong chế độ text trên monochrome display.

HGC: Có thể hiển thị theo chế độ text và chế độ đồ họa trên màn hình monochrome. Trong chế độ đồ họa, độ phân giải tối đa 720x348.

CGA: Có thể hiển thị theo chế độ text và chế độ đồ họa trên màn hình màu. Trong chế độ text: màu chữ: 16, màu nền: 8.

Trong chế độ graphics:

- 4 màu, độ phân giải: 320x200.

- 2 màu (đen trắng), độ phân giải: 640x200.

EGA: Có thể hiển thị theo chế độ text và chế độ đồ họa trên màn hình màu, bao hàm các chế độ của CGA.

Trong chế độ graphics: có khả năng thể hiện 16 màu với độ phân giải: 640x350.

Các loại Adapter trong máy tính IBM PS/2

MCGA: Multi Color Graphics Array

VGA: Video Graphics Array

- 16 màu: 640x480

- 256 màu: 320x200

SVGA: Super VGA

- 256 màu, 64K màu hay màu thực

- Độ phân giải cao hơn VGA

6.2.2. Lập trình cho display adapter

Bộ nhớ ảnh màn hình (video RAM): nằm trên adapter được truy cập tới thông qua địa chỉ như địa chỉ của RAM. Mỗi điểm (pixel) trên màn hình tương ứng với 1 hay nhiều bit trong bộ nhớ màn hình.

Các dữ liệu cần hiển thị ra màn hình sẽ được lưu trữ trong Video RAM, có một vòng lặp đọc các giá trị từ Video RAM và hiển thị từng điểm pixel lên màn hình. Người lập trình có thể đưa dữ liệu lên màn hình bằng cách ghi dữ liệu trực tiếp vào bộ nhớ màn hình.

Trong chế độ text: Mỗi ký tự được vẽ trong một mảng điểm hai chiều. Mỗi ký tự được lưu trữ trong 2 bytes: 1 byte chứa mã ASCII của ký tự, 1 byte chứa thuộc tính hiển thị của ký tự (màu, màu nền, độ sáng ...).

Font chữ: Một mảng bộ nhớ chứa các mẫu vẽ thể hiện cho mỗi ký tự.

Trong chế độ đồ họa: Số bits để thể hiện cho một pixel phụ thuộc vào số màu được dùng để vẽ pixel.

Trong chế độ B/W: 1 pixel ~ 1 bit (on/off)

Trong chế độ 4 (2^2) colors: 1 pixel ~ 2 bits

Trong chế độ 16 (2^4) colors: 1 pixel ~ 4 bits

Trong chế độ 256 (2^8) colors: 1 pixel ~ 8 bits

6.3. MÔ HÌNH ĐỒ HỌA TRONG TURBO C/C++

6.3.1. Màn hình hiển thị và hệ tọa độ

Tọa độ màn hình trong chế độ Text và Graphics:

Trong chế độ Text: Màn hình được chia thành lưới các cells (thông thường 25 dòng 80 cột), mỗi cell sẽ hiển thị 1 ký tự.

Trong chế độ Graphics: Màn hình được xem như một ma trận điểm hai chiều. Kích cỡ của ma trận được gọi là độ phân giải, phụ thuộc vào chế độ đồ họa và Adapter. Tọa độ gốc của hệ thống: xác định ở góc trên bên trái.

View Port: Cửa sổ hiển thị dữ liệu (vùng nhìn), là một hình chữ nhật mà các dữ liệu đồ họa sẽ hiển thị bên trong.

Các hàm trong Turbo C/C++ làm việc với tọa độ trong View Port.

6.3.2. Chế độ màn hình

Xác định chế độ màn hình Text hay Graphics

Chế độ màn hình xác định số lượng màu thể hiện được và độ phân giải của màn hình trong chế độ Graphics. Với mỗi loại vỉ điều khiển màn hình Borland cung cấp tệp điều khiển màn hình tương ứng, ví dụ EGAVGA.BGI cho màn hình VGA, EGA...

Bảng chế độ màn hình đồ họa trong Turbo C/C++ đối với từng loại Adapter như sau:

Graphics driver	graphics_modes	Value	Column x Row	Palette	Pages
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 Color	1
MCGA	MCGA0	0	320 x 200	C0	1
	MCGA1	1	320 x 200	C1	1
	MCGA2	2	320 x 200	C2	1
	MCGA3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 Color	1
	MCGAHI	5	640 x 480	2 Color	1
EGA	EGALO	0	640 x 200	16 Color	4
	EGAHI	1	640 x 350	16 Color	2
EGA64	EGA64LO	0	640 x 200	16 Color	1
	EGA64HI	1	640 x 350	4 Color	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 Color	1*
	EGAMONOHI	3	640 x 350	2 Color	2**
HERC	HERCMONOHI	0	720 x 348	2 Color	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 Color	1
	ATT400HI	5	640 x 200	2 Color	1
VGA	VGALO	0	640 x 200	16 Color	2
	VGAMED	1	640 x 350	16 Color	2
	VGAHI	2	640 x 480	16 Color	1
PC3270	PC3270HI	0	720 x 350	2 Color	1
IBM8514	IBM8514HI	1	1024 x 760	256 Color	
	IBM8514LO	0	640 x 480	256 Color	

* 64K on EGAMONO card

** 256K on EGAMONO card

6.3.3. Xây dựng một chương trình đồ họa

Để thực hiện các thao tác đồ họa, tính toán kích thước đối tượng vẽ hay kiểm tra độ phân giải màn hình... công việc đầu tiên là phải khởi động màn hình ở chế độ đồ họa.

Turbo C++ gọi khởi động chế độ màn hình đồ họa với hàm:

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

Hàm initgraph khởi động hệ thống đồ họa bằng việc nạp các điều khiển đồ họa từ đĩa và đặt hệ thống trong chế độ đồ họa. Nó khởi tạo lại và đặt lại tất cả các tham số đồ họa (như màu, vị trí hiện tại, viewport...) về giá trị mặc định của chúng.

Các tham số:

*int far *graphdriver* xác định điều khiển đồ họa sẽ được sử dụng. Có thể sử dụng các hằng số của graphdriver được định nghĩa trên bảng 1.

*int far *graphmode* xác định chế độ đồ họa.

Nếu graphmode = DETECT, hàm initgraph sẽ đặt graphmode ở chế độ phân giải cao nhất mà hệ thống có thể có. Có thể đặt graphmode theo các hằng số trong bảng chế độ màn hình đồ họa trong Turbo C/C++.

char far pathtodriver chỉ ra đường dẫn đến thư mục BGI (Borland Graphics Interface) của trình biên dịch của Borland.

Có thể kiểm tra kết quả của việc khởi động chế độ đồ họa bằng hàm *int graphresult()*, hàm này trả về số hiệu lỗi của việc khởi động chế độ đồ họa. Có thể đưa ra thông báo lỗi bằng việc gọi hàm *grapherrormsg(int)* với đối số của hàm là số hiệu lỗi trả lại.

Sau khi khởi động màn hình ở chế độ đồ họa, ta có thể thực hiện tiếp các bước vẽ các đối tượng đồ họa theo ý đồ biểu diễn thông qua các hàm đồ họa cơ bản của thư viện ngôn ngữ C/C++.

Khi kết thúc chế độ đồ họa, ta phải dùng hàm *closegraph()* để trả về các điều khiển đồ họa và chuyển về chế độ văn bản như bình thường.

Ví dụ:

```
#include <graphics.h>
#include <stdio.h>
#include <process.h>
main()
{
 int errorcode;
int graphdriver;
int graphmode;
graphdriver = DETECT;
initgraph(&graphdriver, &graphmode, "..");
errorcode = graphresult();
if (errorcode != grOk)
{
printf("graphics error: %s\n",grapherrormsg(errorcode));
exit(1);
}
// xen các hàm vẽ đồ họa vào đây
getch();
/* kết thúc chế độ đồ họa */
closegraph();
return(0);}
```

6.3.4. Một số hàm cơ bản của thư viện đồ họa

Turbo C++ cung cấp cho người lập trình một số lượng rất phong phú và khá đa dạng các hàm thư viện đồ họa. Sau đây là thống kê và mô tả của một số hàm đồ họa cơ bản hay được dùng trong các chương trình đồ họa.

1. Hàm void far initgraph(int far *graphdriver,int far *graphmode, char far pathtodriver);

Ý nghĩa: Khởi động màn hình ở chế độ đồ họa (đã mô tả ở phần trên)

2. Hàm void far closegraph(void);

Ý nghĩa: Giải phóng các điều khiển đồ họa, đặt màn hình trở về chế độ trước khi khởi động chế độ đồ họa.

3. Hàm void far setcolor(int color);

Ý nghĩa: Đặt màu vẽ cho bút vẽ hiện thời.

Tham số: int color: màu bút vẽ được đặt, color có giá trị từ 0 đến giá trị trả về của hàm *getmaxcolor()*.

4. Hàm int far getmaxcolor()

Ý nghĩa: Trả về số màu lớn nhất mà chế độ màn hình hiện thời có thể thể hiện được.

5. Hàm int far getcolor(void);

Ý nghĩa: Trả về giá trị màu của bút vẽ hiện thời

6. Hàm void far putpixel(int x,int y,int color);

Ý nghĩa*:* Đặt một pixel có màu color vào điểm có tọa độ (x, y) trên màn hình.

7. unsigned far getpixel(int x,int y);

Ý nghĩa*:* Nhận giá trị màu của một điểm có tọa độ (x, y) trên màn hình.

8. void far moveto(int x,int y);

Ý nghĩa: Đặt bút vẽ đến vị trí hiện thời có tọa độ (x, y).

9. Hàm void far lineto (int x,int y)

Ý nghĩa: Vẽ một đoạn thẳng từ vị trí hiện thời đến điểm có tọa độ (x, y), với màu vẽ hiện thời, kiểu đường và độ dày đường hiện thời. Nếu không chọn các tham số trên thì giá trị mặc định của chúng sẽ là: màu bút vẽ là màu trắng, độ dày đường là 1 pixel, kiểu đường là kiểu liền nét.

10. Hàm void far line(int x1,int y1,int x2,int y2);

Ý nghĩa*:* Vẽ một đoạn thẳng nối hai điểm có tọa độ (x1, y1) và (x2, y2) với màu vẽ hiện thời, kiểu đường và độ dày đường hiện thời.

11. Hàm void far rectangle(int left,int top,int right,int bottom);

Ý nghĩa*:* Vẽ hình chữ nhật có tọa độ đỉnh trên bên trái là (left, top) và tọa độ đỉnh dưới bên phải là (right, bottom), với màu vẽ hiện thời, kiểu đường và độ dày đường hiện thời.

12. Hàm void far circle(int x,int y,int radius)

Ý nghĩa*:* Vẽ đường tròn, có tọa độ tâm là (x, y) và bán kính là radius, với màu vẽ là màu hiện thời.

13. Hàm void far arc(int x,int y,int stangle,int endangle,int radius);

Ý nghĩa*:* Vẽ cung tròn.

(int x, int y): tọa độ tâm cung tròn

int stangle: độ lớn góc bắt đầu tính theo độ (tính từ phía phải trục nằm ngang)

int endangle: độ lớn góc kết thúc tính theo độ (tính từ phía phải trục nằm ngang)

int radius: độ lớn bán kính cung tròn.

14. Hàm void far ellipse(int x,int y,int stangle,int endangle, int xradius,int yradius)

Ý nghĩa: Vẽ cung ellipse với màu của bút vẽ hiện thời.

(int x, int y) tọa độ tâm hình ellipse

int stangle: độ lớn góc bắt đầu tính theo độ (tính từ phía phải trục nằm ngang)

int endangle: độ lớn góc kết thúc tính theo độ (tính từ phía phải trục nằm ngang)

int xradius: độ lớn bán kính theo trục x.

int yradius: độ lớn bán kính theo trục y.

15. Hàm void far fillellipse(int x,int y,int xradius,int yradius);

Ý nghĩa*:* Vẽ hình ellipse và tô màu hình ellipse theo màu và kiểu tô hiện thời.

(int x, int y) tọa độ tâm hình ellipse

int xradius: độ lớn bán kính theo trục x.

int yradius: độ lớn bán kính theo trục y.

16. Hàm int far getmaxx(void);

Ý nghĩa*:* Trả về giá trị số pixel của màn hình theo chiều nằm ngang, trong chế độ điều khiển đồ họa hiện thời.

17. int far getmaxy(void);

Ý nghĩa*:* Trả về giá trị số pixel của màn hình theo chiều thẳng đứng, trong chế độ điều khiển đồ họa hiện thời.

18. void far outtextxy(int x,int y,char far *textString);

Ý nghĩa: Hiển thị xâu ký tự textString trong khung nhìn ở vị trí tọa độ (int x, int y).

6.4. VÍ DỤ VỀ VẼ ĐỒ HỌA

Viết chương trình vẽ hàm số y = f(x), với y_1 = sinx và y_2 = sinx/x.

Để vẽ đồ thị y = f(x) ta cần phải biết các giá trị x và tương ứng với nó là giá trị y tức là dãy số x, y. Ta sẽ sử dụng hàm putpixel(x,y,màu) để chấm các điểm (x,y) đó.

Chương trình như sau:

```
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
#define pi 3.14159255
#define TileX 20
#define TileY 100
int i,x,y; float t;
int KhoiTaoDoHoa(void);
void main()
{
KhoiTaoDoHoa();
setbkcolor(CYAN);
setviewport(0,getmaxy()/2,getmaxx(),getmaxy(),0);
setcolor(LIGHTRED);
outtextxy(getmaxx()/2,10,"(0,0)");
setcolor(RED);
moveto(getmaxx()/2-5,10);
linerel(0,-(getmaxy()/2-20));
outtextxy((getmaxx()/2-5),-(getmaxy()/2-20),"y");
moveto(getmaxx()/2-5,10);
linerel(0,getmaxy()/2-60);
moveto(0,0);
linerel(getmaxx()-10,0);
outtextxy(getmaxx()-10,0,"x");
setcolor(BLUE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
outtextxy(200,200,"VE DO THI HAM SO Y=F(x)");
for (i=1; i<=2000; i++) / *Ham y=sin(x)* /
{
x=(int)(2*pi*i*TileX/400);
```

```
y=-(int)(sin((2*i*pi)/400)*TileY);
putpixel(x,y,YELLOW);
}
getch();
for (i=1;i<=2000;i++) /*Ham y=sin(x)/x */
{
t=pi*i;
x=(int)(t*TileX/400+320);
y=-(int)(2*sin((t)/400)*TileY/t*200);
putpixel(x,y,WHITE);
}
getch();
closegraph();
return;
}
int KhoiTaoDoHoa(void)
{
int graphdriver=DETECT, graphmode,error_code;
initgraph(&graphdriver,&graphmode,"..\\bgi");
error_code=graphresult();
if (error_code!=grOk)
return(-1);
if((graphdriver!=EGA)&&(graphdriver!=VGA))
{
closegraph();
return 0;
}
return(1); }
```

Bài tập chương 6

1. Viết chương trình vẽ đoạn thẳng từ tọa độ (20, 20) đến (400, 300) trong màn hình đồ họa 640x480x16màu.
2. Viết chương trình hiển thị kích thước của màn hình đồ họa hiện hành. (Gợi ý: sử dụng hàm getmaxx() và getmaxy()).
3. Viết chương trình vẽ đường tròn tâm (200, 150) bán kính 50 với đường nét đứt.
4. Viết chương trình vẽ hình chữ nhật với độ rộng nét là 3 điểm ảnh và tô màu đỏ.
5. Viết chương trình hiển thị xâu ký tự "HVAN" với phông không chân, kích thước 24.
6. Viết chương trình hiển thị xâu ký tự "Ngôn ngữ lập trình C/C++" theo hướng thẳng đứng.
7. Viết chương trình vẽ đồ thị hàm số y = sin(x).
8. Viết chương trình vẽ đa giác đều n cạnh với n nhập vào từ bàn phím.
9. Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Tọa độ (x, y) của điểm gieo được nhập vào, tùy thuộc giá trị cụ thể của x,y mà chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

Chương 7

GIỚI THIỆU LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nội dung chương này sẽ trình bày các vấn đề sau đây:

- Giới thiệu lập trình hướng đối tượng
- Lớp và các thành phần của lớp
- Cấp phát bộ nhớ động - các thành phần tĩnh - friend
- Thừa kế và đa hình
- Nạp chồng toán tử và các hàm chuyển đổi
- Các hàm mẫu (template)

7.1. KHÁI NIỆM VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

Lập trình hướng đối tượng OOP (Object - Oriented Programming) là một phương pháp mới trên bước đường tiến hóa của lập trình máy tính, nhằm làm cho chương trình trở nên linh hoạt, tin cậy và dễ phát triển. Kỹ thuật OOP không theo cách lập trình cổ điển, thủ tục từ trên xuống (top - down procedureal) mà phân tích bài toán thành các thực thể gọi là đối tượng, sau đó xây dựng các hàm xung quanh đối tượng. Để hiểu được OOP chúng ta hãy xem xét lịch sử của quá trình lập trình.

7.1.1. Lập trình tuyến tính

Máy tính đầu tiên được lập trình bằng mã nhị phân, sử dụng các công tắc cơ khí để nạp chương trình. Cùng với sự xuất hiện của các thiết bị lưu trữ lớn và bộ nhớ máy tính có dung lượng lớn nên các ngôn ngữ lập trình cấp cao đầu tiên được đưa vào sử dụng. Thay vì phải suy nghĩ trên một dãy các bit và byte, lập trình viên có thể viết

một loạt lệnh gần với tiếng Anh và sau đó chương trình dịch thành ngôn ngữ máy.

Các ngôn ngữ lập trình cấp cao đầu tiên được thiết kế để lập các chương trình làm các công việc tương đối đơn giản như tính toán. Các chương trình ban đầu chủ yếu liên quan đến tính toán và không đòi hỏi gì nhiều ở ngôn ngữ lập trình. Hơn nữa, phần lớn các chương trình này tương đối ngắn, thường ít hơn 100 dòng.

Khi khả năng của máy tính tăng lên thì khả năng để triển khai các chương trình phức tạp cũng tăng lên. Các ngôn ngữ lập trình trước đây không còn thích hợp đối với việc lập trình đòi hỏi cao hơn.

Chúng ta biết rằng, một đoạn lệnh thường được chép lặp lại mỗi khi chúng ta dùng trong nhiều chương trình nên chương trình càng dài dòng, logic của chương trình càng khó hiểu. Trong lập trình tuyến tính không có các phương tiện cần thiết để sử dụng lại các phần mã chương trình đã viết.

Ngoài ra, chương trình được điều khiển để nhảy đến nhiều chỗ mà thường không có sự giải thích rõ ràng. Làm thế nào để chương trình đến được chỗ cần thiết hoặc tại sao như vậy?

Ngôn ngữ lập trình tuyến tính không có khả năng kiểm soát phạm vi nhìn thấy của các dữ liệu. Mọi dữ liệu trong chương trình đều là dữ liệu toàn cục, nghĩa là chúng có thể bị sửa đổi ở bất kỳ phần nào của chương trình. Việc dò tìm các thay đổi không mong muốn đó của các phần tử dữ liệu trong một dãy mã lệnh dài và vòng vèo đã từng làm cho các lập trình viên mất rất nhiều thời gian.

7.1.2. Lập trình cấu trúc

Rõ ràng là các ngôn ngữ mới với các tính năng mới cần phải được phát triển để có thể tạo ra các ứng dụng tinh vi hơn. Vào thập kỷ 60 và 70 của thế kỷ XX, ngôn ngữ lập trình có cấu trúc ra đời. Các chương trình có cấu trúc được tổ chức theo các công việc mà chúng thực hiện.

Về bản chất, chương trình được chia thành các chương trình con riêng rẽ (còn gọi là hàm hay thủ tục) thực hiện các công việc rời rạc trong quá trình lớn hơn, phức tạp hơn. Các hàm này được giữ càng độc lập với nhau càng nhiều càng tốt, mỗi hàm có dữ liệu và logic riêng. Thông tin được chuyển giao giữa các hàm thông qua tham số, các hàm có thể có các biến cục bộ mà các đối tượng bên ngoài phạm vi của hàm không thể truy xuất được. Như vậy, các hàm có thể được xem là các chương trình con được đặt chung với nhau để xây dựng nên một ứng dụng.

Mục tiêu là làm sao cho việc triển khai các phần mềm dễ dàng hơn đối với các lập trình viên mà vẫn cải thiện được tính tin cậy và dễ bảo quản chương trình. Một chương trình có cấu trúc được hình thành bằng cách "bẻ gãy" các mảnh nhỏ mà sau đó trở thành các hàm. Bằng cách "cô lập" các công việc vào trong các hàm, chương trình có cấu trúc có thể làm giảm khả năng của một hàm này ảnh hưởng đến một hàm khác. Việc này cũng làm cho việc phân tách các vấn đề trở nên dễ dàng hơn. Sự gói gọn này cho phép chúng ta có thể viết các chương trình sáng sủa hơn và giữ được điều khiển trên từng hàm. Các biến toàn cục không còn nữa và được thay thế bằng các tham số và các biến cục bộ có phạm vi nhỏ hơn và dễ kiểm soát hơn. Cách tổ chức tốt hơn này nói lên rằng chúng ta có khả năng quản lý logic của cấu trúc chương trình, làm cho việc triển khai và bảo dưỡng chương trình nhanh hơn, hữu hiệu hơn và hiệu quả hơn.

Một khái niệm lớn đã được đặt ra trong lập trình có cấu trúc là sự trừu tượng hóa (*Abstraction*). Sự trừu tượng hóa có thể xem như khả năng quan sát một sự việc mà không cần xem xét đến các chi tiết bên trong của nó. Trong một chương trình có cấu trúc, chúng ta chỉ cần biết một hàm đã cho có thể làm được một công việc cụ thể gì là đủ. Còn làm thế nào mà công việc đó lại thực hiện được là không quan trọng, chừng nào hàm còn tin cậy được thì còn có thể dùng nó mà không cần phải biết nó thực hiện đúng chức năng của mình như thế nào. Điều này gọi là sự trừu tượng hóa theo chức năng (Funtional Abstraction) và là nền tảng của lập trình có cấu trúc.

Tuy nhiên lập trình cấu trúc có những điểm yếu nhất định. Khi độ phức tạp của một chương trình tăng lên, sự phụ thuộc của chương trình vào các kiểu dữ liệu cơ bản mà nó xử lý cũng tăng theo. Vấn đề trở nên rõ ràng là cấu trúc dữ liệu trong chương trình quan trọng chẳng kém gì các phép toán thực hiện trên chúng nhất là khi kích thước của chương trình càng tăng.

Các kiểu dữ liệu được xử lý trong nhiều hàm khác nhau bên trong một chương trình có cấu trúc nên khi có sự thay đổi trong các dữ liệu này thì cũng cần phải thực hiện cả các thay đổi ở mọi nơi có thao tác tác động trên chúng. Đây có thể là một công việc tốn thời gian và kém hiệu quả đối với các chương trình có hàng ngàn dòng lệnh và hàng trăm hàm trở lên.

Một yếu điểm nữa của việc lập trình có cấu trúc là khi có nhiều lập trình viên làm việc theo nhóm cùng một ứng dụng nào đó. Trong một chương trình có cấu trúc, các lập trình viên được phân công viết một tập hợp các hàm và các kiểu dữ liệu. Vì có nhiều lập trình viên khác nhau quản lý các hàm riêng, có liên quan đến các kiểu dữ liệu dùng chung nên các thay đổi mà lập trình viên tạo ra trên một phần tử dữ liệu sẽ làm ảnh hưởng đến công việc của tất cả các người còn lại trong nhóm.

7.1.3. Sự trừu tượng hóa dữ liệu

Sự trừu tượng hóa dữ liệu (*Data abstraction*) tác động trên các dữ liệu cũng tương tự như sự trừu tượng hóa theo chức năng. Khi có trừu tượng hóa dữ liệu, các cấu trúc dữ liệu và các phần tử có thể được sử dụng mà không cần bận tâm đến các chi tiết cụ thể. Chẳng hạn như các số dấu chấm động đã được trừu tượng hóa trong tất cả các ngôn ngữ lập trình, chúng ta không cần quan tâm đến cách biểu diễn nhị phân chính xác nào cho số dấu chấm động khi gán một giá trị, cũng không cần biết tính bất thường của phép nhân nhị phân khi nhân các giá trị dấu chấm động. Điều quan trọng là các số dấu chấm động hoạt động đúng đắn và hiểu được.

Sự trừu tượng hóa dữ liệu giúp chúng ta không phải bận tâm về các chi tiết không cần thiết. Nếu lập trình viên phải hiểu biết về tất cả các khía cạnh của vấn đề ở mọi lúc và về tất cả các hàm của chương trình thì chỉ ít hàm mới được viết ra. Nhưng sự trừu tượng hóa theo dữ liệu đã tồn tại sẵn trong mọi ngôn ngữ lập trình đối với các dữ liệu phức tạp như số dấu chấm động. Tuy nhiên chỉ mới gần đây, người ta mới phát triển các ngôn ngữ cho phép chúng ta định nghĩa các kiểu dữ liệu trừu tượng riêng.

7.1.4. Lập trình hướng đối tượng

Khái niệm hướng đối tượng được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu. Sự thay đổi căn bản ở chỗ, một chương trình hướng đối tượng được thiết kế xoay quanh dữ liệu mà chúng ta làm việc trên đó hơn là bản thân chức năng của chương trình. Điều này là hoàn toàn tự nhiên khi chúng ta hiểu rằng mục tiêu của chương trình là xử lý dữ liệu. Suy cho cùng, mọi việc mà máy tính thực hiện vẫn thường được gọi là xử lý dữ liệu. Dữ liệu và thao tác liên kết với nhau ở mức cơ bản (còn có thể gọi là mức thấp nhất), mỗi thứ đều đòi hỏi ở thứ kia có mục tiêu cụ thể, các chương trình hướng đối tượng làm tường minh mối quan hệ này.

Lập trình hướng đối tượng liên kết cấu trúc dữ liệu với các thao tác, theo cách mà tất cả thường nghĩ về thế giới quanh mình. Chúng ta thường gắn một số các hoạt động cụ thể với một loại hoạt động nào đó và đặt các giả thiết của mình trên các quan hệ đó.

Ví dụ 1: Chúng ta biết rằng một chiếc xe có các bánh xe, di chuyển được và có thể đổi hướng của nó bằng cách quẹo tay lái. Tương tự như thế, một cái cây là một loại thực vật có thân, gỗ và lá. Một chiếc xe không phải là một cái cây, mà cái cây không phải là một chiếc xe và chúng ta có thể giả thiết rằng cái mà chúng ta làm được với một chiếc xe thì không thể làm được với một cái cây. Chẳng hạn, thật là vô nghĩa khi muốn lái một cái cây, còn chiếc xe thì lại chẳng lớn thêm được khi chúng ta tưới nước cho nó.

Lập trình hướng đối tượng cho phép chúng ta sử dụng các quá trình suy nghĩ như vậy với các khái niệm trừu tượng được sử dụng trong máy tính. Một mẫu tin (*record*) nhân sự có thể được đọc ra, thay đổi và lưu trữ lại; còn số phức thì có thể được dùng trong các tính toán. Tuy vậy, không thể nào lại viết một số phức vào tập tin làm mẫu tin nhân sự và ngược lại hai mẫu tin nhân sự lại không thể cộng với nhau được. Một chương trình hướng đối tượng sẽ xác định đặc điểm và hành vi cụ thể của các kiểu dữ liệu, điều đó cho phép chúng ta biết một cách chính xác rằng chúng ta có thể có được những gì ở các kiểu dữ liệu khác nhau.

Chúng ta còn có thể tạo ra các quan hệ giữa các kiểu dữ liệu tương tự nhưng khác nhau trong một chương trình hướng đối tượng. Người ta thường tự nhiên phân loại ra mọi thứ, thường đặt mối liên hệ giữa các khái niệm mới với các khái niệm đã có và thường có thể thực hiện suy diễn giữa chúng trên các quan hệ đó. Chúng ta có thể quan niệm thế giới theo cấu trúc cây, với các mức xây dựng chi tiết hơn kế tiếp nhau cho các thế hệ sau so với các thế hệ trước. Đây là phương pháp hiệu quả để tổ chức thế giới quanh chúng ta. Các chương trình hướng đối tượng cũng làm việc theo một phương thức tương tự, trong đó chúng cho phép xây dựng các cơ cấu dữ liệu và thao tác mới dựa trên các cơ cấu có sẵn, mang theo các tính năng của các cơ cấu nền mà chúng dựa trên đó, trong khi vẫn thêm vào các tính năng mới.

Lập trình hướng đối tượng cho phép chúng ta tổ chức dữ liệu trong chương trình theo một cách tương tự như các nhà sinh học tổ chức các loại thực vật khác nhau. Theo cách nói lập trình đối tượng: xe hơi, cây cối, các số phức, các quyển sách đều được gọi là các *lớp (Class)*.

Một lớp là một bản mẫu mô tả các thông tin cấu trúc dữ liệu, lẫn các thao tác hợp lệ của các phần tử dữ liệu. Khi một phần tử dữ liệu được khai báo là một phần tử của một lớp thì nó được gọi là một *đối tượng (Object)*. Các hàm được định nghĩa hợp lệ trong một lớp được gọi là *các phương thức (Method)* và chúng là các hàm duy nhất có thể

xử lý dữ liệu của các đối tượng của lớp đó. *Một thực thể (Instance)* là một vật thể có thực bên trong bộ nhớ, thực chất đó là một đối tượng được cấp phát vùng nhớ.

Mỗi một đối tượng có riêng cho mình một bản sao các phần tử dữ liệu của lớp còn gọi là các *biến thực thể (Instance variable)*. Các phương thức định nghĩa trong một lớp có thể được gọi bởi các đối tượng của lớp đó. Điều này được gọi là gửi một thông điệp (*Message*) cho đối tượng. Các thông điệp này phụ thuộc vào đối tượng, chỉ đối tượng nào nhận thông điệp mới phải làm việc theo thông điệp đó. Các đối tượng đều độc lập với nhau vì vậy các thay đổi trên các biến thể hiện của đối tượng này không ảnh hưởng gì trên các biến thể hiện của các đối tượng khác và việc gửi thông điệp cho đối tượng này không ảnh hưởng gì đến các đối tượng khác.

7.2. MỘT SỐ KHÁI NIỆM MỚI TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Trong lập trình hướng đối tượng, các khái niệm như sự đóng gói, tính kế thừa và tính đa hình là các khái niệm căn bản, là nền tảng tư tưởng của nó. Hiểu được khái niệm này, chúng ta bước đầu tiếp cận với phong cách lập trình mới, phong cách lập trình dựa vào đối tượng làm nền tảng mà trong đó quan điểm che dấu thông tin thông qua sự đóng gói là quan điểm trung tâm của vấn đề.

7.2.1. Sự đóng gói (Encapsulation)

Sự đóng gói là cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất, tránh được các tác động bất ngờ từ bên ngoài. Thể thống nhất này gọi là đối tượng.

Trong một đối tượng, dữ liệu hay thao tác hay cả hai có thể là riêng (private) hoặc chung (public) của đối tượng đó.

Thao tác hay dữ liệu riêng là thuộc về đối tượng chỉ được truy cập bởi các thành phần của đối tượng, điều này nghĩa là thao tác hay dữ liệu không thể truy cập bởi các phần khác của chương trình tồn tại

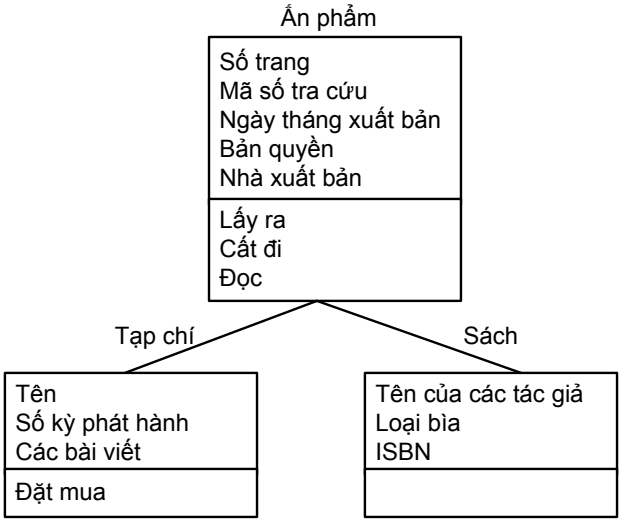
ngoài đối tượng. Khi thao tác hay dữ liệu chung, các phần khác của chương trình có thể truy cập nó mặc dù nó được định nghĩa trong một đối tượng. Các thành phần chung của một đối tượng dùng để cung cấp một giao diện có điều khiển cho các thành phần riêng của đối tượng.

Cơ chế đóng gói là phương thức tốt để thực hiện cơ chế che dấu thông tin so với các ngôn ngữ lập trình cấu trúc.

7.2.2. Tính kế thừa (Inheritance)

Chúng ta có thể xây dựng các lớp mới từ các lớp cũ thông qua sự kế thừa. Một lớp mới còn được gọi là *lớp dẫn xuất* (derived class) có thể thừa hưởng dữ liệu và các phương thức của lớp cơ sở (base class) ban đầu. Trong lớp này, có thể bổ sung các thành phần dữ liệu và các phương thức mới vào những thành phần dữ liệu và các phương thức mà nó thừa hưởng từ lớp cơ sở. Mỗi lớp (kể cả lớp dẫn xuất) có thể có một số lượng bất kỳ các lớp dẫn xuất. Qua cơ cấu kế thừa này, dạng hình cây của các lớp được hình thành. Dạng cây của các lớp trông giống như cây gia phả vì thế các lớp cơ sở còn được gọi là lớp cha (parent class) và các lớp dẫn xuất được gọi là lớp con (child class).

Ví dụ: Chúng ta sẽ xây dựng một tập các lớp mô tả cho thư viện các ấn phẩm. Có hai kiểu ấn phẩm: tạp chí và sách. Chúng ta có thể tạo ra một ấn phẩm tổng quát bằng cách định nghĩa các thành phần dữ liệu tương ứng với số trang, mã số tra cứu, ngày tháng xuất bản, bản quyền và nhà xuất bản. Các ấn phẩm có thể được lấy ra, cất đi và đọc. Đó là các phương thức thực hiện trên một ấn phẩm. Tiếp đó, chúng ta định nghĩa hai lớp dẫn xuất tên là tạp chí và sách. Tạp chí có tên, số ký phát hành và chứa nhiều bài của các tác giả khác nhau. Các thành phần dữ liệu tương ứng với các yếu tố này được đặt vào định nghĩa của lớp tạp chí. Tạp chí cũng cần có một phương thức nữa đó là đặt mua. Các thành phần dữ liệu xác định cho sách sẽ bao gồm tên của các tác giả, loại bìa (cứng hay mềm) và số hiệu ISBN của nó. Như vậy, chúng ta có thể thấy: sách và tạp chí có chung các đặc trưng ấn phẩm trong khi vẫn có các thuộc tính riêng của chúng.



Hình 7.1: Lớp ấn phẩm và các lớp dẫn xuất của nó

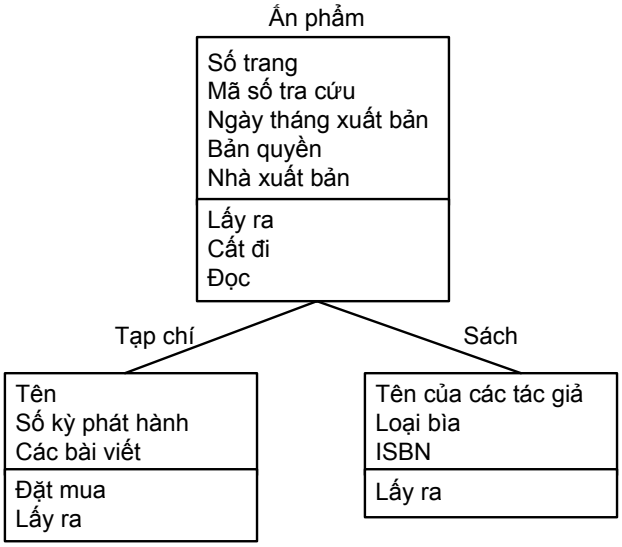
Với tính kế thừa, chúng ta không phải mất công xây dựng lại từ đầu các lớp mới, chỉ cần bổ sung để có được trong các lớp dẫn xuất các đặc trưng cần thiết.

7.2.3. Tính đa hình (Polymorphism)

Xét lại ví dụ trên, chúng ta thấy rằng cả tạp chí và sách đều phải có khả năng lấy ra. Tuy nhiên phương pháp lấy ra cho tạp chí có khác so với phương pháp lấy ra cho sách, mặc dù kết quả cuối cùng giống nhau. Khi phải lấy ra tạp chí, thì phải sử dụng phương pháp lấy ra riêng cho tạp chí (dựa trên một bản tra cứu) nhưng khi lấy ra sách thì lại phải sử dụng phương pháp lấy ra riêng cho sách (dựa trên hệ thống phiếu lưu trữ). Tính đa hình trong lập trình hướng đối tượng cho phép chúng ta xác định một phương thức để lấy ra một tạp chí hay một cuốn sách. Khi lấy ra một tạp chí nó sẽ dùng phương thức lấy ra riêng cho tạp chí, còn khi lấy ra một cuốn sách thì nó sử dụng phương thức lấy ra tương ứng với sách. Kết quả là chỉ cần một tên phương thức duy nhất được dùng cho cả hai công việc tiến hành trên hai lớp dẫn xuất có liên quan, mặc dù việc thực hiện của phương thức đó thay đổi tùy theo từng lớp.

Như vậy, tính đa hình (Polymorphism) là khả năng để cho một thông điệp đưa ra cách thực hiện của nó theo lớp cụ thể của đối tượng nhận thông điệp. Khi một lớp dẫn xuất được tạo ra (ví dụ lớp tạp chí và lớp sách) nó có thể thay đổi cách thực hiện các phương thức nào đó mà nó thừa hưởng từ lớp cơ sở của nó (ví dụ phương thức lấy ra từ lớp ấn phẩm). Một thông điệp khi được gửi đến một đối tượng của lớp cơ sở, sẽ dùng phương thức đã định nghĩa cho nó trong lớp cơ sở. Nếu một lớp dẫn xuất định nghĩa lại một phương thức thừa hưởng từ lớp cơ sở của nó thì một thông điệp có cùng tên với phương thức này, khi được gửi tới một đối tượng của lớp dẫn xuất sẽ gọi phương thức đã định nghĩa cho lớp dẫn xuất.

Tính đa hình dựa trên sự nối kết (Binding), đó là quá trình gắn một phương thức với một hàm thực sự. Khi các phương thức kiểu đa hình được sử dụng thì trình biên dịch chưa thể xác định hàm nào tương ứng với phương thức nào sẽ được gọi. Hàm cụ thể được gọi sẽ tùy thuộc vào việc phần tử nhận thông điệp lúc đó thuộc lớp nào, do đó hàm được gọi chỉ xác định được vào lúc chương trình chạy. Điều này được gọi là sự kết nối muộn (Late binding) hay kết nối lúc chạy (Runtime binding) vì nó xảy ra khi chương trình đang thực hiện.



Hình 7.2: Lớp ấn phẩm và các lớp dẫn của nó

7.3. LỚP VÀ CÁC THÀNH PHẦN CỦA LỚP

7.3.1. Khái niệm lớp

Trong phần trên ta đã đưa ra khái niệm lớp trong lập trình hướng đối tượng. Cơ chế lớp của C++ cho phép người dùng định nghĩa ra kiểu dữ liệu riêng của mình.

Một lớp của C++ có bốn thuộc tính liên kết:

- Một tập hợp các thành phần dữ liệu, đại diện cho các thuộc tính của lớp.

- Một tập hợp các hàm thành phần, là tập các phép toán có thể được áp dụng vào các đối tượng của lớp. Chúng còn được gọi là giao diện của lớp.

- Phạm vi thâm nhập lớp từ chương trình. Thành phần của một lớp có thể được xác định là private, protected hay public.

- Một tên nhãn lớp dùng làm bộ xác định kiểu cho lớp do người dùng định nghĩa.

Lớp thường được khai báo theo cú pháp sau:

class **<tên lớp>** {

private:

<khai báo các thành phần riêng >

protected:

<khai báo các thành phần bảo vệ >

public:

<khai báo các thành phần công cộng>

};

<định nghĩa của các hàm thành phần chưa được định nghĩa bên trong khai báo lớp>

Những thành phần được khai báo trong lớp chia làm hai nhóm:

- *Những thành phần private* không thể truy nhập được bởi những hàm bên ngoài lớp mà chỉ có thể truy nhập được bởi các hàm thành phần khác bên trong chính lớp đó, tức là chúng chỉ được sử dụng bên trong thân các hàm thành phần của lớp.

- *Những thành phần public* có thể được truy nhập bởi các hàm thành phần bên trong lớp và các hàm bên ngoài lớp. Nếu truy nhập bởi các hàm bên ngoài lớp thì phải sử dụng thông qua đối tượng của lớp. Các đối tượng thuộc lớp sẽ truy nhập tới các thành phần public thông qua toán tử xác định '.'.

7.3.2. Các thành phần của lớp (Data member and function member)

Dữ liệu thành phần trong một lớp có thể có các kiểu sau đây:

1. Kiểu private - dữ liệu riêng

2. Kiểu public - dữ liệu chung

3. Kiểu protect - dữ liệu được bảo vệ

4. Kiểu class - dữ liệu kiểu lớp

5. Kiểu static - dữ liệu tĩnh

Hàm thành phần trong một lớp có thể có các kiểu sau đây:

1. Kiểu private - hàm thành phần riêng

2. Kiểu public - hàm thành phần chung

3. Kiểu protect - hàm thành phần được bảo vệ

4. Kiểu static - hàm thành phần tĩnh

5. Constructor (hàm khởi tạo) là hàm thành phần cho phép khởi tạo dữ liệu của đối tượng một cách tự động khi nó được khai báo sau tên lớp.

6. Destructor (hàm hủy bỏ) là hàm thành phần cho phép hủy bỏ đối tượng một cách tự động khi không còn dùng đến đối tượng trong chương trình nữa.

7. Friend - hàm thành phần thân thiện

8. Overloading Operator - Toán tử tải bội

9. Overloading Function - Hàm tải bội

10. Virtual function - Hàm ảo

Thành phần dữ liệu (data member)

Các thành phần dữ liệu được khai báo trong lớp theo các quy tắc như khai báo các biến dữ liệu thường, chỉ khác là các thành phần dữ liệu không được phép có khởi đầu tường minh.

Một đối tượng lớp có thể được khai báo như một thành phần dữ liệu.

Định nghĩa các hàm thành phần (member function)

Các hàm thành phần của lớp được khai báo ngay bên trong thân lớp.

Định nghĩa ngay bên trong lớp

```
Class <tên lớp>
{ private:
// khai báo định nghĩa các thành phần private
 public:
 // khai báo các biến public
return_type fun_name(arg_list)
{
// Các lệnh trong thân hàm
}
// các hàm thành phần khác
}
```

Sau đây ta sẽ xét một ví dụ định nghĩa lớp *thanhvien*, trong đó thành phần *private* gồm các thông tin của một cá nhân: tên, tuổi; thành phần *public* gồm hai phương thức: phương thức *nhap()* để nhập thông tin và phương thức *xuat* để hiển thị thông tin của một cá nhân đó.

Các hàm *nhap()* và *xuat()* đều được định nghĩa trực tiếp ngay trong định nghĩa lớp:

Ví dụ 1:

```
# include <iostream.h>
# include <conio.h>
# include <stdio.h>
class thanhvien
{ char ten[25];
      int tuoi;
public:
void nhap(void)
{ cout<< "Nhap ten:"; gets(ten);
cout <<"Nhap tuoi:"; cin >>tuoi; }
void xuat(void)
{ cout<< "Ten:"<<ten<<"\n";
cout <<"Tuoi:"<<tuoi<<endl; }
} a;
int main() {
clrscr();
a.nhap();
a.xuat();
getch();
return 0; }
```

Các định nghĩa hàm ngoài thân lớp phải theo cú pháp sau:

type **classname**:**functionName**(*parameter_list*)
{
    function body;
}

parameter_list: danh sách tham số của hàm

function body: thân hàm

Ta xét ví dụ, định nghĩa lớp SINHVIEN có thành phần private gồm các thông tin riêng của một sinh viên: họ tên (kiểu ký tự), năm sinh (kiểu số nguyên), điểm trung bình (số thực); thành phần public gồm hai phương thức nhap() và xem() các dữ liệu cho sinh viên.

Các hàm nhap() và xem() minh họa các định nghĩa hàm thành phần bên ngoài định nghĩa lớp.

Ví dụ 2:

```
# include <iostream.h>
# include <conio.h>
# include <stdio.h>
class SINHVIEN
{ char hoten[25];
      int namsinh;
      float dtb;
public:
void nhap(void);
void xem(void);
};
void SINHVIEN::nhap()
{ cout<< "Nhap ho ten:"; gets(hoten);
cout <<"Nhap nam sinh:"; cin >>namsinh;
cout <<"Nhap diem trung binh:"; cin >>dtb;}
void SINHVIEN::xem(void)
{ cout<< "Ho ten:"<<hoten<<endl;
cout <<"Nam sinh:"<<namsinh<<endl;
cout <<"Diem trung binh:"<<dtb<<endl;}
int main() {
clrscr();
SINHVIEN s;
s.nhap();
s.xem();
```

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

Chú ý, truy nhập đến thành phần *public* thông qua đối tượng như sau:

obj_name.var_member;

obj_name.fun_member;

Chúng ta sẽ xét ví dụ sau đây để phân biệt thành phần (dữ liệu và hàm) *private* khác thành phần *public* như thế nào.

Ví dụ 3:

```
Class A
{ private:
      long x;
      void f()
public:
      double y;
      void g();
};
int main()
{ A a;
a.x =10; // Sai vì x là dữ liệu thành phần private
a.f(); // Sai vì f() là dữ liệu thành phần private
a.y =20; // Đúng, vì y là dữ liệu thành phần public
a.g(); // Đúng vì g() là dữ liệu thành phần public
}
```

Ví dụ 4: Chúng ta xây dựng chương trình quản lý sinh viên bao gồm các chức năng:

- Nhập dữ liệu cho các sinh viên

- Đưa ra màn hình danh sách các sinh viên có năm sinh nhập vào từ bàn phím

- Đưa ra màn hình danh sách các sinh viên được sắp tăng dần theo tên.

Để giải quyết bài toán này, trước hết chúng ta sẽ xây dựng lớp (class) các sinh viên bao gồm các thuộc tính: số báo danh, tên và năm sinh. Ta sẽ sử dụng lớp sinh viên này để tạo ra một mảng các đối tượng là các sinh viên.

Chương trình như sau:

```
# include <iostream.h>
# include <conio.h>
# include <stdio.h>
# include <string.h>
class SINHVIEN
{ int sbd;
char ten[30];
int ns;
public:
char *getten();
int getns() {return ns;}
void nhapsv();
void xemsv();
};
char *SINHVIEN::getten()
{ int len=strlen(ten);
char *s = new char[len];
strcpy(s,ten);
return s;}
void SINHVIEN::nhapsv()
{ cout <<"so bao danh:"; cin>>sbd;
cout<<"ten:";gets(ten);
cout<<"nam sinh:"; cin>>ns;}
void SINHVIEN::xemsv()
```

```
{ cout<<sbd<<" "
<<ten<<" "
<<ns<<endl;}
const n=4;
SINHVIEN a[n];
void nhapdssv()
{ for (int i=0; i<n; i++) {a[i].nhapsv();}}
void xemdssv()
{ for (int i=0; i<n; i++) {a[i].xemsv();}
cout<<endl;}
void xemns()
{ int x;
cout<<"nhap nam sinh:"; cin>>x;
cout<<"danh sach sinh vien sinh nam"<<x<<":\n";
for (int i=0; i<n; i++)
if (a[i].getns()==x) a[i].xemsv();
cout<< endl;}
void swap(int i, int j)
{SINHVIEN tg=a[i]; a[i]=a[j]; a[j]=tg;}
void sapxepten()
{int i,j,k;
for (i=0; i<n; i++)
{ k=i;
for (j=i+1; j<n; j++)
if (strcmp(a[j].getten(), a[k].getten())<0)k=j;
if (k!=i) swap(i,k); }}
void main()
{ nhapdssv();
xemns();
sapxepten();
xemdssv();
getch();}
```

Ngoài thành phần *private* và *public*, trong lớp còn có thành phần *protected* được dùng trong quan hệ kế thừa giữa các lớp.

Ngoài cách dùng từ khóa *class* ta còn có thể sử dụng cách sau đây để định nghĩa lớp:

1. Sử dụng cấu trúc (*struct)* để định nghĩa lớp, khi đó ngầm định các trường trong struct là thành phần public. Ta có thể quy định lại các trường thuộc thành phần private hay public.

2. Sử dụng hợp (*union*) để định nghĩa lớp, khi đó ngầm định các trường trong *union* là thành phần *public* và không thể thay đổi được.

Các hàm thành phần của lớp *khác* với hàm thông thường ở các tính chất sau:

- Hàm thành phần của lớp có quyền thâm nhập vào tất cả các loại thành phần của lớp đó, trong khi các hàm thông thường chỉ được quyền thâm nhập vào các thành phần public của lớp.

- Các hàm thành phần được định nghĩa bên trong phạm vi của lớp, các hàm thông thường được định nghĩa bên trong phạm vi tệp.

Che giấu thông tin:

Che giấu thông tin là một cơ chế hình thức để hạn chế việc thâm nhập của người dùng vào biểu diễn bên trong của một kiểu lớp. Nó được xác định bởi việc gắn nhãn chung (*public*), riêng (*private*) hay bảo vệ (*protected*) bên trong thân lớp. Một thành phần công cộng là thành phần có thể thâm nhập tới được từ bất kỳ đâu trong chương trình.

- Một thành phần được bảo vệ là thành phần công cộng đối với các lớp suy diễn, nó là thành phần riêng đối với phần còn lại của chương trình.

- Một thành phần riêng chỉ có thể được thâm nhập tới từ các hàm thành phần và hàm bạn *friend* của lớp của nó.

Hàm cấu tử và hàm hủy tử

Hàm cấu tử - constructor

- Hàm cấu tử là một hàm thành phần đặc biệt không thể thiếu được trong một lớp. Nó cho phép khởi tạo dữ liệu của đối tượng một cách tự động khi nó được khai báo sau tên lớp, tức là nó được gọi tự động mỗi khi có một đối tượng được khai báo.

- Các đặc trưng của hàm cấu tử:

+ Hàm cấu tử có cùng tên với tên của lớp.

+ Hàm cấu tử phải có thuộc tính public.

+ Hàm cấu tử không có giá trị trả về. Và không cần khai báo void

Có thể có nhiều hàm cấu tử trong cùng lớp (chồng các hàm cấu tử).

Hàm cấu tử có thể được khai báo với các tham số có giá trị ngầm định.

Hàm hủy tử - destructor

Ngược với hàm cấu tử, hàm hủy bỏ được gọi khi đối tượng tương ứng bị xoá khỏi bộ nhớ. Nói cách khác, nó là hàm thành phần cho phép hủy bỏ đối tượng một cách tự động khi không còn dùng đối tượng trong chương trình nữa. Hàm hủy tử có đặc trưng sau:

- Tên của hàm hủy tử bắt đầu bằng dấu '~' theo sau là tên của lớp tương ứng. Hàm hủy tử phải có thuộc tính public.

- Nói chung hàm hủy tử không có tham số, mỗi lớp chỉ có một hàm hủy tử.

- Giống như hàm cấu tử, hàm hủy tử không có giá trị trả về.

7.4. CẤP PHÁT BỘ NHỚ ĐỘNG - CÁC THÀNH PHẦN TĨNH - FRIEND

7.4.1. Cấp phát và giải phóng bộ nhớ động

Vùng bộ nhớ tự do được cấp phát bằng cách áp dụng toán tử *new*. Việc cấp phát vùng nhớ cho các đối tượng được thực hiện vào lúc chương trình chạy được gọi là cấp phát bộ nhớ động.

Cú pháp của toán tử new:

<*pointer_to_name*> = **new** <*name*> [<*name_initializer*>];

Đặc trưng của bộ nhớ tự do là nó không có tên. Các đối tượng được cấp phát bộ nhớ tự do cần phải thao tác gián tiếp qua con trỏ. Mặc định, toán tử *new* gán giá trị 0 khi không còn đủ bộ nhớ tự do để thoả mãn yêu cầu.

Việc cấp phát bộ nhớ khi dịch gọi là cấp phát bộ nhớ tĩnh. Ví dụ, bộ nhớ dành cho bản thân con trỏ được cấp phát lúc dịch được gọi là cấp phát tĩnh.

Bộ nhớ tự do bị hủy bỏ thông qua toán tử *delete*.

Cú pháp của toán tử delete:

delete <**pointer_to_name**>;

Việc áp dụng toán tử *delete* cho vùng bộ nhớ không được cấp phát trong miền bộ nhớ tự do có thể gây ra lỗi nguy hiểm cho chương trình khi chạy.

7.4.2. Các thành phần tĩnh

Đôi khi ta cần mọi đối tượng của một lớp có quyền thâm nhập vào cùng một biến. Thành phần tĩnh của lớp là một giải pháp giúp ta làm việc này.

Về hình thức: Nếu đặt từ khóa *static* trước các khai báo thành phần thì dữ liệu đó là dữ liệu thành phần tĩnh (hoặc gọi là dữ liệu thành phần *static*).

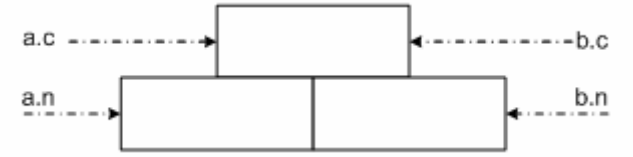
Về bản chất: Dữ liệu thành phần *static* chỉ có một bản sao duy nhất dùng chung cho tất cả các đối tượng trong một lớp.

Ví dụ khai báo:

```
class A
{ static int c;
```

```
int n;
...
} a,b;
```

Sẽ tạo ra 2 đối tượng a và b có chung thành phần dữ liệu c:



Một thành phần tĩnh có ý nghĩa như một biến toàn cục cho lớp, nhưng nó còn có nhiều ưu điểm hơn biến toàn cục.

Mỗi một đối tượng trong lớp đương nhiên đã có thành phần dữ liệu *private* là thành phần riêng của nó và được bao bọc, che giấu. Thành phần dữ liệu *public* của đối tượng suy cho cùng cũng vẫn là sở hữu riêng của đối tượng nhưng có thể được đối tượng đó thể hiện ra bên ngoài. Một thành phần dữ liệu *static* được khai báo trong một lớp thì sẽ không là của riêng bất kỳ đối tượng nào trong lớp đó và tất cả các đối tượng của lớp sẽ dùng chung thành phần dữ liệu *static* này. Thành phần *static* trong lớp là cần thiết để đảm bảo tính đầy đủ khi mô tả thế giới thực. Các thực thể cùng loại khi tồn tại có những cái riêng được đóng kín, có cái riêng có thể bộc lộ và có cái chung (cái chung đó chỉ có ở loại thực thể đang xét).

Có thể tăng cường che giấu thông tin. Thành phần tĩnh cũng có thể được đặt là thành phần công cộng, riêng hay bảo vệ.

Thành phần tĩnh không nằm trong "không gian tên" toàn cục của chương trình, do vậy loại bỏ được khả năng trùng tên ngẫu nhiên.

Chỉ tồn tại một thể nghiệm cho thành phần tĩnh của lớp. Cú pháp khai báo thành phần dữ liệu tĩnh chỉ là ghép thêm vào phần đầu khai báo từ khóa static.

Khi đối tượng đầu tiên của lớp được tạo lập thì những biến static được tự khởi gán bằng 0.

Thành phần dữ liệu tĩnh chỉ được khởi đầu một lần trong chương trình. Có thể thâm nhập thành phần dữ liệu tĩnh như thâm nhập với các thành phần khác. Vì thành phần *static* dùng chung cho tất cả các đối tượng của lớp nên không thể truy nhập thành phần *static* thông qua một đối tượng cụ thể nào trong lớp mà phải thông qua chính tên lớp cùng toán tử phân giải miền xác định.

Định nghĩa của hàm thành phần tĩnh cũng giống như định nghĩa các hàm thành phần khác. Việc gọi các hàm thành phần tĩnh cũng có thể thông qua một đối tượng lớp hay một con trỏ tới đối tượng lớp, hay cũng có thể được gọi trực tiếp bằng cách dùng cú pháp lớp thành phần.

Trước khi sử dụng các đối tượng trong lớp phải có một lệnh đặc biệt khởi tạo biến *static*, nếu không có lệnh này chương trình sẽ báo lỗi.

static_type **class_name**:: *var_static*;

Ví dụ sau đây tạo ra một lớp HOSO với biến *tuoi* là một biến nguyên kiểu *static* và *diem* là một biến nguyên thuộc thành phần *private*.

```
# include <iostream.h>
# include <conio.h>
class HOSO
{ static int tuoi;
      int diem;
public:
void get(int a)
{diem = a;tuoi++;}
void put(void)
{cout<<"tuoi:"<<tuoi<<"diem="<<diem<<endl;}};
int HOSO::tuoi;
int main() {
clrscr();
HOSO a,b,tuoi;
```

```
a.get(20); b.get(30); tuoi.get(40);
a.put(); b.put(); tuoi.put();
getche(); return 0;}
```

Vì *tuoi* là biến *static* nên nó chỉ có một bản sao duy nhất dùng chung cho ba đối tượng a, b, tuoi trong chương trình. Mỗi khi một đối tượng mới được tạo, giá trị của biến *static* sẽ được xét tới. Trong trường hợp này, giá trị của biến static là c lần lượt tăng đến 3 (vì giá trị khởi tạo đầu tiên của static ngầm định là 0) và dùng chung cho cả a, b, tuoi. Ngược lại với *tuoi,* biến *diem* không phải là biến *static* nên diem có ba bản sao tương ứng với ba đối tượng khác nhau.

Chúng ta sẽ xét tiếp một ví dụ sau để thấy được ý nghĩa thực tế của thành phần dữ liệu *static*.

Giả sử trong một hệ thống mạng máy tính, một số người trong cùng phòng làm việc có chung tài khoản truy nhập mạng (gồm tên và mật khẩu) và tạo thành một nhóm (NHOM) làm việc. Ta có thể xem nhóm (NHOM) các thành viên trên như là một lớp, còn tên (TEN) và mật khẩu (MATKHAU) được xem như đối tượng của lớp này. Mỗi một thành viên khi khai báo tên và mật khẩu sẽ được cấp các quyền riêng và có các thiết lập khác nhau về hệ thống. Như vậy ta có thể xem Tên và mật khẩu sẽ là thành phần dữ liệu tĩnh (của từng đối tượng) của lớp. Chương trình đăng nhập như sau:

```
# include <iostream.h>
# include <conio.h>
# include <string.h>
class MEMBER
{ static long taikhoan;
static long matkhau;
char *loginten;
double loginkhoa;
public:
void get(char *s, double k)
{ int len = strlen(s);
```

```
loginten=new char[len+1];
strcpy(loginten,s);
loginkhoa=k;}
void put()
{ cout<<"ten truy nhap:"<<loginten<<endl;
cout<<"Ma khoa truy nhap:"<<loginkhoa<<endl;
public_infor(); }
void public_infor()
{ cout<<"tai khoan:"<<taikhoan<<endl;
cout <<"mat khau:"<<matkhau<<endl<<endl;}
};
long MEMBER::taikhoan=922011;
long MEMBER::matkhau=23081968;
int main() { clrscr();
MEMBER t1,t2;
t1.get("Phuong Thao",1721994);
t2.get("Ngoc Quynh",711997);
t1.put(); t2.put();
getche(); return 0;
}
```

7.4.3. Hàm thành phần thân thiện (Friend function)

Hàm thành phần thân thiện là một trong những hàm đặc biệt và quan trọng trong lớp. Hàm thành phần thân thiện không thuộc miền xác định của lớp, nơi nó được khai báo là thân thiện (nhờ từ khóa *friend*). Khi một lớp hay một hàm trở thành "bạn" với lớp khác, các hàm thành phần của lớp "bạn" sẽ được trao quyền thâm nhập vào các thành phần riêng của lớp kia.

Khai báo hàm "bạn" bắt đầu bằng từ khóa *friend*. Nó chỉ xuất hiện bên trong định nghĩa lớp.

Hàm thành phần thân thiện nếu được định nghĩa bên ngoài lớp thì nó được định nghĩa như một hàm bình thường. Ta không truy nhập hàm thân thiện bởi đối tượng của lớp mà truy nhập nó như là một hàm bình thường.

Hàm tự do là bạn của một lớp

Cú pháp khai báo:

```
class <class name>
{
private:
public:
friend function_type Function_name(parameter_list);
}
// định nghĩa hàm bạn
function_type Function_name(parameter_list)
{
 // thân hàm bạn
}
```

Hàm tự do là bạn của nhiều lớp

Một hàm tự do là bạn của nhiều lớp, thì nó cũng được khai báo là bạn trong tất cả các lớp mà nó coi là bạn. Khi đó hàm tự do có thể truy nhập vào tất cả các thành phần dữ liệu private của các lớp bạn.

Phương thức của một lớp là bạn của một lớp khác

Một phương thức của một lớp này cũng có thể là bạn của lớp kia. Với cách làm việc này, phương thức có thể truy nhập vào thành phần riêng tư của cả hai lớp.

Ta sẽ xét ví dụ sau để minh họa cho việc sử dụng hàm thân thiện. Giả sử cần xây dựng lớp các phân số, trong đó các phép toán: nhập, xem, tối giản và cộng phân số được thể hiện bằng các hàm thân thiện.

```
 # include <iostream.h>
# include <conio.h>
# include <math.h>
class FRAC
{ int t,m;
public:
void read();
friend void write(FRAC);
friend void reduce(FRAC&);
friend FRAC sum(FRAC, FRAC);
};
void FRAC::read()
{ cout<<"tu so:"; cin>> t;
do {cout<<"mau so:"; cin>>m;
} while (m==0);
}
void write (FRAC P)
{ reduce(P);
cout<<P.t<<"/"<<P.m;}
void reduce(FRAC& P)
{ int i,j;
i=abs(P.t);
i=P.t; j=P.m;
j= abs(P.m);
while (i!=j) if(i>j) i-=j; else j-=i;
P.t= P.t/i;
P.m=P.m/j;
}
FRAC sum(FRAC P, FRAC Q)
{FRAC R;
R.t = P.t*Q.m + Q.t*P.m;
```

```
R.m =P.m* Q.m;
reduce(R);
return R;
}
int main() {
clrscr();
FRAC p,q;
p.read();
q.read();
write(p); cout<<"+"; write(q);
cout<<"=";write(sum(p,q));
getche();
return 0; }
```

Vai trò của hàm thân thiện lúc này cho phép sử dụng chúng như những hàm bình thường để thao tác trên đối tượng hoặc xử lý chính dữ liệu của lớp mà không cần truy nhập qua đối tượng của lớp.

7.5. THỪA KẾ VÀ ĐA HÌNH

7.5.1. Thừa kế

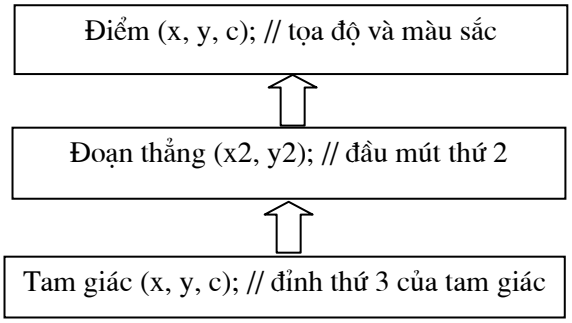
Thừa kế nền tảng cho việc sử dụng lại các bộ phận của chương trình. Thừa kế cho phép ta định nghĩa một lớp mới, gọi là lớp *dẫn xuất*, từ một lớp đã có (gọi là *lớp cơ sở*). Lớp dẫn xuất sẽ thừa kế các thành phần (dữ liệu, hàm) của lớp cơ sở, đồng thời thêm vào các thành phần mới. Nói cách khác, lớp dẫn xuất chỉ kế thừa các thuộc tính và phương thức của lớp cơ sở chứ không kế thừa các hàm cấu tử và hủy tử.

Lớp dẫn xuất thường bổ sung thêm các thành phần dữ liệu và các hàm thành phần trong định nghĩa, ta nói lớp dẫn xuất cụ thể hơn so với lớp cơ sở và vì vậy thường mô tả một lớp các đối tượng có phạm vi hẹp hơn lớp cơ sở.

Hàm cấu tử của lớp dẫn xuất thường tự động gọi các hàm cấu tử của các lớp cơ sở để khởi tạo giá trị cho các thành phần trong lớp cơ sở. Hàm hủy tử được gọi theo thứ tự ngược lại. Khi người sử dụng không khai báo tường minh sử dụng hàm cấu tử nào của lớp cơ sở được sử dụng trong hàm cấu tử của lớp dẫn xuất thì trình biên dịch tự động sử dụng hàm cấu tử mặc định không có đối số.

Ta có thể hiểu kế thừa đơn giản như sau: Nếu lớp B là lớp kế thừa từ lớp A thì toàn bộ các thành phần của lớp A cũng có mặt trong lớp B mà không cần phải khai báo trong lớp B. Lớp A thường được gọi là lớp cha (hay lớp cơ sở) còn lớp B là lớp con (hay lớp dẫn xuất).

Ví dụ:



Khai báo: Cách khai báo một lớp dẫn xuất trong C++

```
class  A
{
  ...
};
 class  B: public A
{
  ...
};
```

Các kiểu dẫn xuất khác nhau

Tùy thuộc vào từ khóa đứng trước lớp cơ sở trong khai báo lớp dẫn xuất, người ta phân biệt ba loại dẫn xuất như sau: *public*, *private* và *protected*.

Dẫn xuất public

Trong dẫn xuất *public*, các thành phần, các hàm bạn và các đối tượng của lớp dẫn xuất không thể truy nhập đến các thành phần *private* của lớp cơ sở. Các thành phần *protected* trong lớp cơ sở trở thành các thành phần *private* trong lớp dẫn xuất. Các thành phần *public* của lớp cơ sở vẫn là *public* trong lớp dẫn xuất.

Dẫn xuất private

Trong trường hợp này, các thành phần *public* trong lớp cơ sở không thể truy nhập được từ các đối tượng của lớp dẫn xuất, nghĩa là chúng trở thành các thành phần *private* trong lớp dẫn xuất. Các thành phần *protected* trong lớp cơ sở có thể truy nhập được từ các hàm thành phần và các hàm bạn của lớp dẫn xuất.

Dẫn xuất protected

Trong dẫn xuất loại này, các thành phần *public*, *protected* trong lớp cơ sở trở thành các thành phần *protected* trong lớp dẫn xuất.

Có hai kiểu thừa kế:

Đơn thừa kế:

Truy nhập các thành phần lớp cơ sở từ lớp dẫn xuất: Các thành phần *private* trong lớp cơ sở không thể truy nhập được từ các lớp dẫn xuất. Các lớp dẫn xuất chỉ có thể truy nhập vào các thành phần *protected* và thành phần *public* của lớp cơ sở.

Định nghĩa lại các thành phần lớp cơ sở trong lớp dẫn xuất.

Đa thừa kế:

Đa thừa kế cho phép một lớp có thể là dẫn xuất của nhiều lớp cơ sở, do vậy những gì đã đề cập trong phần đơn thừa kế được tổng quát hóa cho trường hợp đa thừa kế.

Giống như trong đơn thừa kế, trong hàm thành phần của lớp dẫn xuất có thể sử dụng tất cả các hàm thành phần public (hoặc protected) của lớp cơ sở.

Khi có nhiều hàm thành phần cùng tên trong các lớp khác nhau (định nghĩa lại một hàm), ta có thể loại bỏ sự nhập nhằng bằng cách sử dụng toán tử phạm vi "::".

7.5.2. Đa hình

Tính đa hình được xuất hiện khi có sự kế thừa giữa các lớp. Lớp cơ sở được coi là tổng quát hóa của các lớp dẫn xuất, ngược lại các lớp dẫn xuất là chi tiết hóa của lớp cơ sở. Do vậy, có những phương thức tổng quát cho mọi lớp dẫn xuất nên có mặt ở lớp cơ sở nhưng nội dung của nó chỉ được xác định ở các lớp dẫn xuất cụ thể. Ví dụ một lớp cơ sở hình đa giác phải có phương thức tính diện tích của nó nhưng tính diện tích như thế nào sẽ được xác định ở các lớp dẫn xuất hình tam giác, tứ giác, ngũ giác,... Tính chất như trên trong kế thừa gọi là tính đa hình.

Một ví dụ trực quan nữa thể hiện tính đa hình là một ti vi có thể vừa là đối tượng của mặt hàng vừa là đối tượng của lớp mặt hàng điện tử dân dụng. Các đối tượng hình học như hình vuông, hình tròn, hình chữ nhật đều có cùng cách vẽ như nhau: xác định hai điểm đầu và cuối, nối hai điểm này. Do vậy thuật toán tuy giống nhau đối với tất cả các đối tượng hình, nhưng cách vẽ thì phụ thuộc vào từng lớp đối tượng cụ thể. Ta nói phương thức nối điểm của các đối tượng hình học có tính đa hình. Tính đa hình còn được thể hiện trong cách thức hiển thị thông tin trong các đối tượng điểm màu/không màu.

Tính đa hình còn thể hiện khi một hàm thành phần trong lớp cơ sở được gọi từ một đối tượng lớp dẫn xuất, còn bản thân hàm đó thì gọi tới hàm thành phần được định nghĩa đồng thời trong cả lớp cơ sở (khai báo *virtual* có mặt ở đây) và trong các lớp dẫn xuất.

7.6. NẠP CHỒNG TOÁN TỬ VÀ HÀM CHUYỂN ĐỔI

7.6.1. Nạp chồng toán tử

Trong C++, có thể định nghĩa chồng đối với hầu hết các phép toán (một ngôi hoặc hai ngôi) trên các lớp, nghĩa là một trong số các toán hạng tham gia phép toán là các đối tượng. Đây là một khả năng mạnh vì nó cho phép xây dựng trên các lớp các toán tử cần thiết, làm cho chương trình được viết ngắn gọn dễ đọc hơn và có ý nghĩa hơn. Chẳng hạn, khi định nghĩa một lớp complex để biểu diễn các số phức, có thể viết trong C++: a+b, a-b, a*b, a/b với a,b là các đối tượng complex.

Để có được điều này, ta định nghĩa chồng các phép toán +, -, * và / bằng cách định nghĩa hoạt động của từng phép toán giống như định nghĩa một hàm, chỉ khác đây là hàm toán tử (operator function). Hàm toán tử có tên được ghép bởi từ khóa operator và ký hiệu của phép toán tương ứng. Bảng sau đưa ra một số ví dụ về tên hàm toán tử.

Bảng liệt kê một số tên hàm toán tử quen thuộc

Tên hàm	Ý nghĩa
operator **+**	định nghĩa phép +
operator *****	định nghĩa phép nhân *
operator **/**	định nghĩa phép chia /
operator **+=**	định nghĩa phép tự cộng +=
operator **!=**	định nghĩa phép so sánh khác nhau

Hàm toán tử có thể dùng như là một hàm thành phần của một lớp hoặc là hàm tự do, khi đó hàm toán tử phải được khai báo là bạn của các lớp có các đối tượng mà hàm thao tác.

Hàm toán tử phải có thuộc tính public vì nếu không chương trình dịch không thể thực hiện được nó ở ngoài phạm vi lớp.

Ký hiệu đứng sau từ khóa operator phải là một trong số các ký hiệu toán tử áp dụng cho các kiểu dữ liệu cơ sở, không thể dùng các ký hiệu mới. Tất nhiên là ta không thể tạo ra các ký hiệu phép toán mới từ việc định nghĩa chồng các toán tử.

Phải đảm bảo các đặc tính nguyên thủy của toán tử được định nghĩa chồng, chẳng hạn: độ ưu tiên, trật tự kết hợp, số ngôi.

Mỗi hàm toán tử chỉ có thể áp dụng với kiểu toán hạng nhất định. Cần chú ý rằng các tính chất vốn có, chẳng hạn tính giao hoán của toán tử không thể áp dụng một cách tùy tiện cho các toán tử được định nghĩa chồng.

Phần lớn toán tử trong C++ đều có thể định nghĩa chồng. Một số toán tử không thể định nghĩa chồng (chẳng hạn toán tử truy nhập thành phần cấu trúc ".", toán tử phạm vi "::", toán tử điều kiện "?:") và có một số toán tử ta phải tuân theo các ràng buộc sau:

- Phép =, [] nhất định phải được định nghĩa như hàm thành phần của lớp.

- Phép << và >> dùng với cout và cin phải được định nghĩa như hàm bạn.

Hai phép toán ++ và -- có thể sử dụng theo hai cách khác nhau ứng với dạng tiền tố ++a, --b và dạng hậu tố a++, b--. Điều này đòi hỏi hai hàm toán tử khác nhau.

Chiến lược sử dụng hàm toán tử

Về nguyên tắc, định nghĩa chồng một phép toán là khá đơn giản, nhưng việc sử dụng phép toán định nghĩa chồng lại không phải dễ dàng và đòi hỏi phải cân nhắc bởi lẽ nếu bị lạm dụng sẽ làm cho chương trình khó hiểu.

Phải làm sao để các phép toán vẫn giữ được ý nghĩa trực quan nguyên thủy của chúng. Chẳng hạn không thể định nghĩa cộng "+"

như phép trừ "-" hai giá trị. Phải xác định trước ý nghĩa các phép toán trước khi viết định nghĩa của các hàm toán tử tương ứng.

a) Các phép toán một ngôi

Các phép toán một ngôi là: *, &, ~, !, ++, --, sizeof (kiểu)

Các hàm toán tử tương ứng chỉ có một đối số và phải trả về giá trị cùng kiểu với toán hạng, riêng sizeof có giá trị trả về kiểu nguyên không dấu và toán tử (kiểu) dùng để trả về một giá trị có kiểu như đã ghi trong dấu ngoặc.

b) Các phép toán hai ngôi

Các phép toán hai ngôi như:

*, /, %, +, -, <<, >>, <, >, <=, >=, ==, !=, &,|, ^, &&, ||

Hai toán hạng tham gia các phép toán không nhất thiết phải cùng kiểu, mặc dù trong thực tế sử dụng thì thường là như vậy. Như vậy chỉ cần một trong hai đối số của hàm toán tử tương ứng là đối tượng là đủ.

c) Các phép toán gán

Các toán tử gán gồm có:

=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=

Do các toán tử gán được định nghĩa dưới dạng hàm thành phần, nên chỉ có một tham số tường minh và không có ràng buộc gì về kiểu đối số và kiểu giá trị trả về của các phép gán.

d) Toán tử truy nhập thành phần "->"

Phép toán này được dùng để truy xuất các thành phần của một cấu trúc hay một lớp và cần phân biệt với những cách sử dụng khác để tránh dẫn đến sự nhầm lẫn. Có thể định nghĩa phép toán lấy thành phần giống như đối với các phép toán một ngôi.

e) Toán tử truy nhập thành phần theo chỉ số

Toán tử lấy thành phần theo chỉ số được dùng để xác định một thành phần cụ thể trong một khối dữ liệu (cấp phát động hay tĩnh).

Thông thường phép toán này được dùng với mảng, nhưng cũng có thể định nghĩa lại nó khi làm việc với các kiểu dữ liệu khác. Chẳng hạn với kiểu dữ liệu véc-tơ có thể định nghĩa phép lấy theo chỉ số để trả về một thành phần tọa độ nào đó véc-tơ. Và phải được định nghĩa như hàm thành phần có một đối số tường minh.

7.6.2. Các hàm chuyển đổi

Với các kiểu dữ liệu chuẩn, ta có thể thực hiện các phép chuyển kiểu ngầm định, chẳng hạn có thể gán một giá trị int vào một biến long, hoặc cộng giá trị long vào một biến float. Thường có hai kiểu chuyển kiểu: chuyển kiểu ngầm định (tự động) và chuyển kiểu tường minh (ép kiểu). Phép chuyển kiểu ngầm định được thực hiện bởi chương trình biên dịch. Phép chuyển kiểu tường minh xảy ra khi sử dụng phép ép kiểu bắt buộc. Phép ép kiểu thường được dùng trong các câu lệnh gọi hàm để gửi các tham số có kiểu khác với các tham số hình thức tương ứng.

Các kiểu lớp không thể dễ dàng chuyển sang các kiểu khác được mà phải do người lập trình tự làm lấy. C++ cũng cung cấp cách thức định nghĩa phép chuyển kiểu ngầm định và tường minh. Phép chuyển kiểu ngầm định được định nghĩa bằng một hàm cấu tử chuyển kiểu (conversion constructor), còn phép chuyển kiểu tường minh được xác định thông qua toán tử chuyển kiểu hoặc ép kiểu (cast operator).

Phép chuyển kiểu ngầm định được định nghĩa thông qua một hàm cấu tử chuyển kiểu cho lớp. Với đối số có kiểu cần phải chuyển thành một đối tượng của lớp đó. Tham số này có thể có kiểu cơ sở hay là một đối tượng thuộc lớp khác. Hàm cấu tử một tham số trong lớp point trong các chương trình point.cpp ở chương trước là ví dụ cho hàm cấu tử chuyển kiểu.

Các toán tử chuyển kiểu có thể được dùng để chuyển các đối tượng sang các kiểu khác và cũng có thể được dùng cho các mục đích khác ngoài phép gán và khởi tạo giá trị. C++ quy định rằng một hàm toán tử chuyển kiểu như thế buộc phải là hàm thành phần của lớp liên

quan và không có tham số hoặc kiểu trả về. Tên của nó được cho theo dạng như sau:

```
operator type();
```

trong đó type là tên của kiểu dữ liệu mà một đối tượng sẽ được chuyển sang; có thể là kiểu dữ liệu cơ sở (khi đó ta phải chuyển kiểu từ đối tượng sang kiểu cơ sở) hay một kiểu lớp khác (khi đó ta phải chuyển kiểu từ đối tượng lớp này sang lớp khác).

Trong các chuyển đổi kiểu từ lớp này sang một lớp khác có thể được mở rộng cho hai kiểu lớp khác nhau:

- Trong lớp A, ta có thể định nghĩa một hàm toán tử để thực hiện chuyển đổi từ kiểu A sang kiểu B (cast operator).

- Hàm thiết lập của lớp A chỉ với một tham số kiểu B sẽ thực hiện chuyển đổi kiểu từ B sang A.

7.7. CÁC HÀM MẪU (TEMPLATE)

Trong lập trình, nhiều khi có những đoạn chương trình tương đồng về công việc cần làm. Để tránh những công việc có tính chất lặp đi lặp lại như vậy, C++ hỗ trợ khái niệm *khuôn mẫu (template)*.

Khuôn mẫu cho phép ta định nghĩa một công việc có tính chất tổng quát , chung chung, để từ đó có thể tạo nên hàng loạt những công việc cụ thể theo một cách nào đó. Có hai khuôn mẫu: hàm mẫu (function template) và lớp mẫu (class template).

7.7.1. Tạo một hàm khuôn mẫu

Ta đã biết định nghĩa chồng hàm cho phép dùng một tên duy nhất cho nhiều hàm thực hiện các công việc khác nhau. Khái niệm khuôn hình hàm cũng cho phép sử dụng cùng một tên duy nhất để thực hiện các công việc khác nhau, tuy nhiên so với định nghĩa chồng hàm, nó có phần mạnh hơn và chặt chẽ hơn; mạnh hơn vì chỉ cần viết định nghĩa khuôn hình hàm một lần, rồi sau đó chương trình biên dịch làm cho nó thích ứng với các kiểu dữ liệu khác nhau; chặt chẽ hơn bởi vì

dựa theo khuôn hình hàm, tất cả các hàm thể hiện được sinh ra bởi trình biên dịch sẽ tương ứng với cùng một định nghĩa và như vậy sẽ có cùng một giải thuật.

Khai báo và định nghĩa hàm mẫu

template <**class** *Etype1*, **class** *Etype2*,...)

return_type func_name (*Etype1* **v1**, *Etype2* **v2**,...)

{ // *nội dung hàm*

}

Ví dụ: Giả thiết rằng chúng ta cần viết một hàm *min* đưa ra giá trị nhỏ nhất trong hai giá trị có cùng kiểu. Ta có thể viết một định nghĩa như thế đối với kiểu *int* như sau:

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

Giả sử, ta lại phải viết định nghĩa hàm min() cho kiểu *double*, *float*, *char*, *char**...

```
float min(float a, float b) {
 if (a < b) return a;
       return else b; }
```

Nếu tiếp tục như vậy, sẽ có khuynh hướng phải viết rất nhiều định nghĩa hàm hoàn toàn tương tự nhau; chỉ có kiểu dữ liệu các tham số là thay đổi. Các chương trình biên dịch C++ hiện có cho phép giải quyết đơn giản vấn đề trên bằng cách định nghĩa một khuôn hình hàm duy nhất theo cách như sau:

```
#include <iostream.h>
//tạo một khuôn hình hàm
template <class   T> T min(T a, T b)
```

```
{
    if (a < b) return a;
    else return b;
}
```

So sánh với định nghĩa hàm thông thường, ta thấy chỉ có dòng đầu tiên bị thay đổi:

template <**class** *T*> *T* **min** (*T* a, *T* b)

trong đó:

template <**class** *T*>: xác định rằng đó là một khuôn hình với một tham số kiểu *T*;

Phần còn lại: *T* **min**(*T* a, *T* b): nói rằng, *min()* là một hàm với hai tham số hình thức kiểu *T* và có giá trị trả về cũng là kiểu *T*.

7.7.2. Sử dụng hàm khuôn mẫu

Hàm khuôn mẫu cho kiểu dữ liệu cơ sở

Để sử dụng hàm khuôn mẫu *min()* vừa tạo ra, chỉ cần sử dụng hàm *min()* trong những điều kiện phù hợp (ở đây có nghĩa là hai tham số của hàm có cùng kiểu dữ liệu). Như vậy, nếu trong một chương trình có hai tham số nguyên n và p, với lời gọi min(n,p) chương trình biên dịch sẽ tự động sản sinh ra hàm *min()* (ta gọi là một hàm thể hiện) tương ứng với hai tham số kiểu nguyên *int*. Nếu chúng ta gọi *min()* với hai tham số kiểu *float*, chương trình biên dịch cũng sẽ tự động sản sinh một hàm thể hiện min khác tương ứng với các tham số kiểu *float* và cứ thế.

Hàm khuôn mẫu min với kiểu dữ liệu lớp

Để áp dụng hàm khuôn mẫu *min()* ở trên với kiểu lớp, cần phải định nghĩa lớp sao cho có thể áp dụng phép toán so sánh "<" với các đối tượng của lớp này, nghĩa là ta phải định nghĩa một hàm toán tử operator < cho lớp.

Ví dụ: Xây dựng lớp mẫu SWAP gồm 2 thành phần dữ liệu có kiểu bất kỳ và cho phép hoán đổi giá trị của 2 thành phần dữ liệu đó.

```
# include <iostream.h>
# include <conio.h>
template <class  Etype>
class   SWAP
{  Etype   x,y ;
public:
SWAP (Etype  a,  Etype  b)  { x = a ;  y = b ; }
void  swap ();
void print ();
} ;
template <class  Etype>
void  SWAP <Etype> :: swap ()
{  Etype z = x ; x = y; y = z ;}
template <class  Etype>
void  SWAP <Etype> :: print ()
{  cout << " x = "<<x<<" y = " << y << endl;  }
int  main ()
{  SWAP <int>  s1(4, 5);
SWAP <char>  s2 ('A', 'M') ;
s1.print () ; s1.swap () ; s1.print () ;
s2.print () ; s2.swap () ; s2.print () ;
return 0 ;
}
```

Bài tập chương 7

1. Hãy xây dựng lớp để lưu trữ bản ghi về cá nhân, bao gồm các thành phần dữ liệu sau:

```
char  name[25];
float  salary;
char  date_of_birth[9];
```

Hãy tạo lập hai cấu tử, một khởi động bản ghi bởi các giá trị hợp lệ và một cấu tử khác tạo lập bản ghi rỗng. Tạo lập hàm thành phần để hiển thị tên, lương và ngày sinh của từng cá nhân.

2. Cho trước chương trình như sau:

```
#include <iostream.h>
main()
{
cout<< "Lop hoc C/C++";
}
```

Hãy sửa đổi chương trình trên nhưng vẫn giữ nguyên hàm main() để chương trình có thể hiển thị trên màn hình như sau:

```
HVAN don chao Lop hoc C/C++
```

3. Xây dựng lớp tên box. Hàm cấu tử của nó nhận ba giá trị kiểu double là kích thước ba chiều của khối hình hộp. Lớp có hàm tính thể tích của hộp và hàm hiển thị thể tích của mỗi đối tượng.

4. Xây dựng hàm có tên min() chấp nhận đối số là số và cho lại giá trị nhỏ nhất giữa hai số. Hãy nạp chồng hàm min() để chấp nhận xâu ký tự làm đối số.

5. Xây dựng lớp phân số cho phép thực hiện các phép toán số học

Session

- Khi hiện ra thông báo có lưu Session hiện tại hay không → Chọn No.

- Vào Session → Import→ From SAM file, sau đó nhấn Browse, chọn tệp SAM trong thư mục C:\WINOWS\SYSTEM32\CONFIG, chương trình sẽ hiện ra mật khẩu tìm được.

TÀI LIỆU THAM KHẢO

Tiếng Anh

1. Borland, *Borland C++ 3.0 User's Guide*, Borland Pub, 1992.
2. Borland, *Borland C++ 3.0 Programmer's Guide*, Borland Pub, 1992.
3. Greg Pery, *C++ by Example*, Que, 1992.
4. Herbert Schildt, *Teach Yourself C++*, Second Edition, Osborne McGraw-hill, 1994.
5. *The C programming language,* ANSI, W.Kernighan and D.M.Ritchie, Prentice Hall, 1988.
6. Mark Allen Weiss - Florida Internationnal University, 1998, *"Algorithms, data Structures and Problem Solving with C++"* Addison - Wesley Publishing Company, Inc.
7. Scott Robert Ladd - United States of America, 1996, *"C++: Commponents And Algorithms",* M&T Books. A Division of MIS: Press, Inc. A Subsidiary of Henry Holt and Company, Inc. 15 Wec 18th Street. New York, New York 10011.

Tiếng Việt

8. *Giáo trình lý thuyết & Bài tập ngôn ngữ C (tập 1, 2)*, Nguyễn Đình Tê - Hoàng Đức Hải, Nhà xuất bản Lao động Xã hội, 2009.
9. *Giáo trình lập trình hướng đối tượng với C++*, Vũ Thanh Hiền, Nhà xuất bản Thống kê, 2003.
10. *101 thuật toán và chương trình bằng ngôn ngữ C*, Lê Văn Doanh - Trần Khắc Tuấn - Lê Đình Anh, Nhà xuất bản Khoa học và Kỹ thuật, 2002.

11. *Lập trình trên ngôn ngữ C*, Hoàng Chí Thành, Nhà xuất bản Khoa học và Kỹ thuật, 2009.

12. *Lập trình hướng đối tượng với C++*, Nguyễn Chí Trung (chủ biên), Nhà xuất bản Hà Nội, 2005.

13. *Bài tập lập trình hướng đối tượng với C++,* Nguyễn Thanh Thủy (chủ biên), nhà xuất bản khoa học và kỹ thuật, 2004.

14. *Giáo trình Tin học đại cương A2,* Lê Hoài Bắc (chủ biên), Nhà xuất bản đại học Quốc gia TP. Hồ Chí Minh, 2006.

MỤC LỤC

Giáo trình
Ngôn ngữ lập trình C/C++

Chịu trách nhiệm xuất bản
NGUYỄN THỊ THU HÀ

Biên tập:	NGÔ MỸ HẠNH
	TRỊNH THU CHÂU
Trình bày sách:	BÙI CHÂU LOAN
Sửa bản in:	TRỊNH THU CHÂU
Thiết kế bìa:	TRẦN HỒNG MINH

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG

Trụ sở: 18 Nguyễn Du, TP. Hà Nội
ĐT Biên tập: 04.35772143, 04.35772145 ĐT Phát hành: 04.35772138
E-mail: nxb.tttt@mic.gov.vn Fax: 04.35772037
Website: www.nxbthongtintruyenthong .vn
Chi nhánh TP. Hồ Chí Minh: 8A đường D2, P25, Quận Bình Thạnh, TP. Hồ Chí Minh
Điện thoại: 08.35127750, 08.35127751 Fax: 08.35127751
E-mail: cnsg.nxbtttt@ mic.gov.vn
Chi nhánh TP. Đà Nẵng: 42 Trần Quốc Toản, Quận Hải Châu, TP. Đà Nẵng
Điện thoại: 0511.3897467 Fax: 0511.3843359
E-mail: cndn.nxbtttt@ mic.gov.vn

In 1000 bản, khổ 16x24cm tại Công ty In Hải Nam
Số đăng ký kế hoạch xuất bản: 410-2011/CXB/3-102/TTTT
Số quyết định xuất bản: 141/QĐ-NXB TTTT ngày 23 tháng 6 năm 2011
In xong và nộp lưu chiểu tháng 7 năm 2011