```
int &fr(int *d,int i);
void main() {
  clrscr();
  cout<<"Nhap gia tri cho mang a:\n";
  for(int i=0;i<5;i++) {
    cout<<"a["<<i<<"]= ";
    cin>>fr(a,i);
    }
  cout<<"Mang a sau khi nhap\n";
  for(i=0;i<5;i++)
    cout<<a[i]<<" ";
  cout<<"\n";
  getch();
  }

int &fr(int *d,int i) {
  return d[i];
  }
```

```
Nhap gia tri cho mang a:
a[0]= 6
a[1]= 4
a[2]= 3
a[3]= 5
a[4]= 6
Mang a sau khi nhap
6 4 3 5 6
```

Bạn đọc có thể xem thêm phần "Định nghĩa chồng toán tử" để thấy được lợi ích của vấn đề trả về tham chiếu cho hàm.

6. ĐỊNH NGHĨA CHỒNG HÀM (Overloading functions)

C++ cho phép sử dụng một tên cho nhiều hàm khác nhau ta gọi đó là sự "chồng hàm". Trong trường hợp đó, các hàm sẽ khác nhau ở giá trị trả về và danh sách kiểu các tham số. Chẳng hạn chúng ta muốn định nghĩa các hàm trả về số nhỏ nhất trong:

1. hai số nguyên
2. hai số thực
3. hai ký tự
4. ba số nguyên
5. một dãy các số nguyên ...

Dĩ nhiên có thể tìm cho mỗi hàm như vậy một tên phân. Lợi dụng khả năng "định nghĩa chồng hàm" của C++, chúng ta có thể viết các hàm như sau:

Ví dụ 2.12

```
#include <iostream.h>
//Hàm nguyên mẫu
int min(int, int); //Hàm 1
double min(double, double);//Hàm 2
char min(char, char);//Hàm 3
int min(int, int, int);//Hàm 4
int min(int, int *);//Hàm 5
main() {
int n=10, p =12, q = -12;
double x = 2.3, y = -1.2;
char c = 'A', d= 'Q';
int td[7] = {1,3,4,-2,0,23,9};
cout<<"min (n,p) : "<<min(n,p)<<"\n"; //Hàm 1
cout<<"min (n,p,q) : "<<min(n,p,q)<<"\n"; //Hàm 4
cout<<"min (c,d) : "<<min(c,d)<<"\n"; //Hàm 3
cout<<"min (x,y) : "<<min(n,p)<<"\n"; //Hàm 2
cout<<"min (td) : "<<min(7,td)<<"\n"; //Hàm 5
cout<<"min (n,x) : "<<min(n,x)<<"\n"; //Hàm 2
//cout<<"min (n,p,x) : "<<min(n,p,x)<<"\n"; //Lỗi
}
int min(int a, int b) {
  return (a> b? a: b);
}
int min(int a, int b, int c) {
```

```
  return (min(min(a,b),c));
}
double min(double a, double b) {
  return (a> b? a: b);
}
char min(char a, char b) {
  return (a> b? a: b);
}
int min(int n, int *t) {
int res = t[0];
for (int i=1; i<n; i++)
   res = min(res,t[i]);
return res;
}
```

Nhận xét

1. Một hàm có thể gọi đến hàm cùng tên với nó (ví dụ như hàm 4,5 gọi hàm 1).
2. Trong trường hợp có các hàm trùng tên trong chương trình, việc xác định hàm nào được gọi do chương trình dịch đảm nhiệm và tuân theo các nguyên tắc sau:

Trường hợp các hàm có một tham số

Chương trình dịch tìm kiếm "sự tương ứng nhiều nhất" có thể được; có các mức độ tương ứng như sau (theo độ ưu tiên giảm dần):

a) Tương ứng thật sự: ta phân biệt các kiểu dữ liệu cơ sở khác nhau đồng thời lưu ý đến cả dấu.

b) Tương ứng dữ liệu số nhưng có sự chuyển đổi kiểu dữ liệu tự động ("numeric promotion"): **char** và **short** -->**int**; **float** -->**int**.

c) Các chuyển đổi kiểu chuẩn được C và C++ chấp nhận.

d) Các chuyển đổi kiểu do người sử dụng định nghĩa.

Quá trình tìm kiếm bắt đầu từ mức cao nhất và dừng lại ở mức đầu tiên cho phép tìm thấy sự phù hợp. Nếu có nhiều hàm phù hợp ở cùng một mức, chương trình dịch đưa ra thông báo lỗi do không biết chọn hàm nào giữa các hàm phù hợp.

Trường hợp các hàm có nhiều tham số

Ý tưởng chung là phải tìm một hàm phù hợp nhất so với tất cả những hàm còn lại. Để đạt mục đích này, chương trình dịch chọn cho mỗi tham số các hàm phù hợp (ở tất cả các mức độ). Trong số các hàm được lựa chọn, chương trình dịch chọn ra (nếu tồn tại và tồn tại duy nhất) hàm sao cho đối với mỗi đối số nó đạt được sự phù hợp hơn cả so với các hàm khác.

Trong trường hợp vẫn có nhiều hàm thoả mãn, lỗi biên dịch xảy ra do chương trình dịch không biết chọn hàm nào trong số các hàm thỏa mãn. Đặc biệt lưu ý khi sử dụng định nghĩa chồng hàm cùng với việc khai báo các hàm với tham số có giá trị ngầm định sẽ được trình bày trong mục tiếp theo.

7. THAM SỐ NGẦM ĐỊNH TRONG LỜI GỌI HÀM

Ta xét ví dụ sau:

Ví dụ 2.13

```
#include <iostream.h>
void main() {
int n=10,p=20;
void fct(int, int = 12) ;//khai báo hàm với một giá trị ngầm định
fct(n,p); //lời gọi thông thường, có hai tham số
fct(n); //lời gọi chỉ với một tham số
//fct() sẽ không được chấp nhận
}
//khai báo bình thường
void fct(int a, int b) {
cout <<"tham so thu nhat : " <<a <<"\n";
cout<<"tham so thu hai    : "<<b<<"\n";
}
```

```
tham so thu nhat : 10
tham so thu hai  : 20
tham so thu nhat : 10
tham so thu hai  : 12
```

Trong khai báo của `fct()` bên trong hàm `main()` :

```
void fct(int,int =12);
```

khai báo

```
int = 12
```

chỉ ra rằng trong trường hợp vắng mặt tham số thứ hai ở lời gọi hàm fct() thì tham số hình thức tương ứng sẽ được gán giá trị ngầm định 12.

Lời gọi

```
fct();
```

không được chấp nhận bởi vì không có giá trị ngầm định cho tham số thứ nhất.

Ví dụ 2.14

```
#include <iostream.h>
void main(){
int n=10,p=20;
void fct(int = 0, int = 12);//khai báo hàm với hai tham số có giá trị ngầm định
fct(n,p); //lời gọi thông thường, có hai tham số
fct(n);   //lời gọi chỉ với một tham số
fct() ;   //fct() đã được chấp nhận
}
void fct(int a, int b)  //khai báo bình thường
{
cout<<"tham so thu nhat : " <<a <<"\n";
cout<<"tham so thu hai   : "<<b<<"\n";
}
```

```
tham so thu nhat : 10
tham so thu hai   : 20
tham so thu nhat : 10
tham so thu hai   : 12
tham so thu nhat : 0
tham so thu hai   : 12
```

Chú ý

1. Các tham số với giá trị ngầm định phải được đặt ở cuối trong danh sách các tham số của hàm để tránh nhầm lẫn các giá trị.
2. Các giá trị ngầm định của tham số được khai báo khi sử dụng chứ không phải trong phần định nghĩa hàm. Ví dụ sau đây gây ra lỗi biên dịch:

Ví dụ 2.15

```
#include <conio.h>
#include <iostream.h>
void f();
void main() {
  clrscr();
  int n=10,p=20;
  void fct(int =0,int =12);
  cout<<"Goi fct trong main\n";
  fct(n,p);
  fct(n);
  fct();
  getch();
  }
void fct(int a=10,int b=100) {
  cout<<"Tham so thu nhat : "<<a<<"\n";
  cout<<"Tham so thu hai  : "<<b<<"\n";
  }
```

3. Nếu muốn khai báo giá trị ngầm định cho một tham số biến trỏ, thì phải chú ý viết * và = cách xa nhau ít nhất một dấu cách.
4. Các giá trị ngầm định có thể là một biểu thức bất kỳ (không nhất thiết chỉ là biểu thức hằng), có giá trị được tính tại thời điểm khai báo:

```
float x;
int n;
void fct(float = n*2+1.5);
```

5. Chồng hàm và gọi hàm với tham số có giá trị ngầm định có thể sẽ dẫn đến lỗi biên dịch khi chương trình dịch không xác định được hàm phù hợp.

Xét ví dụ sau:

Ví dụ 2.16

```
#include <iostream.h>
void fct(int, int=10);
void fct(int);
```

```
void main() {
  int n=10, p=20;
  fct(n,p);//OK
  fct(n);//ERROR
  }
```

8. BỔ SUNG THÊM CÁC TOÁN TỬ QUẢN LÝ BỘ NHỚ ĐỘNG: new và delete

8.1 Toán tử cấp phát bộ nhớ động new

Với khai báo

```
int * adr;
```

chỉ thị

```
ad = new int;
```

cho phép cấp phát một vùng nhớ cần thiết cho một phần tử có kiểu int và gán cho adr địa chỉ tương ứng. Lệnh tương đương trong C:

```
ad =(int *) malloc(sizeof(int));
```

Với khai báo

```
char *adc;
```

chỉ thị

```
adc =new char [100];
```

cho phép cấp phát vùng nhớ đủ cho một bảng chứa 100 ký tự và đặt địa chỉ đầu của vùng nhớ cho biến adc. Lệnh tương ứng trong C như sau:

```
adc =(char *)malloc(100);
```

Hai cách sử dụng **new** như sau:

Dạng 1

```
new type;
```

giá trị trả về là

(i) một con trỏ đến vị trí tương ứng khi cấp phát thành công

(ii) NULL trong trường hợp trái lại.

Dạng 2

```
new type[n];
```

trong đó n là một biểu thức nguyên không âm nào đó, khi đó toán tử **new** xin cấp phát vùng nhớ đủ để chứa n thành phần kiểu type và trả lại con trỏ đến đầu vùng nhớ đó nếu như cấp phát thành công.

8.2 Toán tử giải phóng vùng nhớ động *delete*

Một vùng nhớ động được cấp phát bởi **new** phải được giải phóng bằng **delete** mà không thể dùng free được, chẳng hạn:

```
delete adr;
delete adc;
```

Ví dụ 2.17

Cấp phát bộ nhớ động cho mảng hai chiều

```
#include<iostream.h>
void Nhap(int **mat);//nhập ma trận hai chiều
void In(int **mat);//In ma trận hai chiều
void main() {
  int **mat;
  int i;
  /*cấp phát mảng 10 con trỏ nguyên*/
  mat = new int *[10];
  for(i=0; i<10; i++)
    /*mỗi con trỏ nguyên xác định vùng nhớ 10 số nguyên*/
    mat[i] = new int [10];
   /*Nhập số liệu cho mảng vừa được cấp phát*/
   cout<<"Nhap so lieu cho matran 10*10\n";
   Nhap(mat);
   /*In ma trận*/
   cout<<"Ma tran vua nhap \n";
   In(mat);
   /*Giải phóng bộ nhớ*/
```

```
   for(i=0;i<10;i++)
     delete mat[i];
   delete mat;
  }
void Nhap(int ** mat) {
  int i,j;
  for(i=0; i<10;i++)
    for(j=0; j<10;j++) {
      cout<<"Thanh phan thu ["<<i<<"]["<<j<<"]= ";
      cin>>mat[i][j];
      }
  }
void In(int ** mat) {
  int i,j;
  for(i=0; i<10;i++) {
    for(j=0; j<10;j++)
      cout<<mat[i][j]<<" ";
    cout<<"\n";
      }
  }
```

Ví dụ 2.18

Quản lý tràn bộ nhớ set_new_handler

```
#include <iostream>
main()
{
void outof();
set_new_handler(&outof);
long taille;
int *adr;
int nbloc;
cout<<"Kich thuoc can nhap? ";
```

```
cin >>taille;
for(nbloc=1;;nbloc++)
  {
   adr =new int [taille];
   cout <<"Cap phat bloc so : "<<nbloc<<"\n";
  }
}
void outof()//hàm được gọi khi thiếu bộ nhớ
{
cout <<"Het bo nho -Ket thuc \n";
exit(1);
}
```

9. TÓM TẮT

9.1 Ghi nhớ

C++ là một sự mở rộng của C(superset), do đó có thể sử dụng một chương trình biên dịch C++ để dịch và thực hiện các chương trình nguồn viết bằng C.

C yêu cầu các chú thích nằm giữa /* và */. C++ còn cho phép tạo một chú thích bắt đầu bằng "//" cho đến hết dòng.

C++ cho phép khai báo khá tuỳ ý. Thậm chí có thể khai báo biến trong phần khởi tạo của câu lệnh lập for.

C++ cho phép truyền tham số cho hàm bằng tham chiếu. Điều này tương tự như truyền tham biến cho chương trình con trong ngôn ngữ PASCAL. Trong lời gọi hàm ta dùng tên biến và biến đó sẽ được truyền cho hàm qua tham chiếu. Điều đó cho phép thao tác trực tiếp trên biến được truyền chứ không phải gián tiếp qua biến trỏ.

Toán tử **new** và **delete** trong C++ được dùng để quản lý bộ nhớ động thay vì các hàm cấp phát động của C.

C++ cho phép người viết chương trình mô tả các giá trị ngầm định cho các tham số của hàm, nhờ đó hàm có thể được gọi với một danh sách các tham số không đầy đủ.

Toán tử "**::**" cho phép truy nhập biến toàn cục khi đồng thời sử dụng biến cục bộ và toàn cục trùng tên.

Có thể định nghĩa các hàm cùng tên với các tham số khác nhau. Hai hàm cùng tên sẽ được phân biệt nhờ giá trị trả về và danh sách kiểu các tham số.