CHƯƠNG 7. KHUÔN HÌNH (TEMPLATE)

KHUÔN HÌNH - TEMPLATE LÀ GÌ?

Ví dụ: Xây dựng hàm tìm max của hai số bất kỳ

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

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

```
1. void Swap( int &x, int &y)
2. {
3.     int Temp = x;
4.     x = y;
5.     y = Temp;
6. }
```

➔ Trong C++ có hỗ trợ cho chúng ta giải quyết vấn đề trên.

2

KHUÔN HÌNH - TEMPLATE LÀ GÌ?

- "**Template**" là từ khóa trong C++
- Là một kiểu dữ liệu trừu tượng, đặc trưng cho các kiểu dữ liệu cơ bản.
- "**Template**" là từ khóa báo cho trình biên dịch rằng đoạn mã sau đây định nghĩa cho nhiều kiểu dữ liệu và mã nguồn của nó sẽ được compile sinh ra tương ứng cho từng kiểu dữ liệu trong quá trình biên dịch.

3

KHUÔN HÌNH - TEMPLATE LÀ GÌ?

Có 2 loại "template" cơ bản:

- **Function template**: là một khuôn mẫu hàm, cho phép định nghĩa các hàm tổng quát thao tác cho nhiều kiểu dữ liệu.
- **Class template**: là một khuôn mẫu lớp, cho phép định nghĩa các lớp tổng quát cho nhiều kiểu dữ liệu.

4

Khuôn hình hàm - Function template

Cú pháp:

```
template <class kieu_tham_chieu>
kieu_tra_ve ten_ham (danh sach tham so)
{
   // phần thân hàm
}
```

5

Khuôn hình hàm - Function template

Ví dụ:

```
1. template <class Type>
2. void Swap( Type &x, Type &y)
3. {
4.     Type Temp = x;
5.     x = y;
6.     y = Temp;
7. }
8.
9. void main()
10. {
11.     int x = 5, y = 10;
12.     float a = 5.5f, b = 3.0f;
13.
14.     Swap(x, y);
15.     Swap(a, b);
16. }
```

Trong ví dụ trên:

- ✓ "**Type**" chỉ là một tên riêng thể hiện cho một kiểu dữ liệu tổng quát.
- ✓ "**class**" ta có thể thay thế bằng "typename", ở đây nó không có sự khác biệt.

6

Khuôn hình hàm - Function template

Vậy cách thức nó hoạt động ra sao??

- Ta hãy xét hoạt động của trình biên dịch khi gặp lời gọi hàm Swap(x, y) với tham số truyền vào là kiểu int,
- Trước hết khi gặp lời gọi hàm Swap(x, y) thì trình biên dịch tìm xem có hàm Swap() nào đã được khai báo với tham số kiểu int hay chưa, nếu có thì sẽ liên kết với hàm đó, nếu chưa nhưng lại tìm thấy từ khóa "template" với hàm Swap() được truyền vào 2 tham số cùng kiểu với nhau (lúc này là kiểu Type), trình biên dịch chỉ cần kiểm tra xem lời gọi hàm Swap(x, y) có 2 tham số có cùng kiểu dữ liệu với nhau hay không(trong ví dụ trên là 2 tham số kiểu int -> cùng kiểu dữ liệu), nếu cùng kiểu thì trình biên dịch lại kiểm tra xem hàm Swap() với 2 tham số kiểu int đã được sinh ra trước đó hay chưa, nếu có thì lời gọi hàm sẽ liên kết với hàm Swap() đã được sinh ra, nếu chưa thì khi đó trình biên dịch sẽ sinh ra hàm Swap(int &x, int &y), tương tự với Swap(a, b) kiểu float cũng tương tự như vậy. Vì vậy trong quá trình biên dịch sẽ sinh ra 2 hàm Swap() cho 2 loại kiểu dữ liệu trên.

7

Khuôn hình hàm - Function template

- Ngoài ra ta có thể dùng prototype cho nguyên mẫu hàm giống như ta làm cho các hàm thông thường

```
1. template <class T> void Swap( T &x, T &y);
2.
3. void main()
4. {
5.     int x = 3, y = 4;
6.     float a = 1.2f, b = 2.2f;
7.
8.     Swap(x, y);
9.     Swap(a, b);
10. }
11.
12. template <class T> void Swap( T &x, T &y)
13. {
14.     T z = x;
15.     x = y;
16.     y = z;
17. }
```

8

Khuôn hình hàm - Function template

- Trong ví dụ trên ta chỉ hoán ví giữa 2 tham số cùng kiểu, vậy với khác kiểu thì làm như thế nào?? Ta chỉ cần khai báo thêm một kiểu dữ liệu tổng quát.

```
1. template <class T, class X>
2. T Sum( T x, X y)
3. {
4.     T sum = x + y;
5.     return sum;
6. }
7.
8. template <class T, class X>
9. X Sum( T x, X y, T z)
10. {
11.    X sum = x + y + z;
12.    return sum;
13. }
```

- Lưu ý: muốn nạp chồng hàm thì các tham số truyền vào ở các hàm phải khác nhau

9

Khuôn hình hàm - Function template

Ví dụ về Function Template

- Xây dựng khuôn hình hàm Swap cho phép tráo đổi giá trị 2 biến số truyền vào
- Xây dựng khuôn hình hàm Sort dùng để sắp xếp gía trị của một mảng trong đó có sử dụng khuôn hình hàm Swap
- Xây dựng khuôn hình hàm Print để in giá trị của một mảng ra màn hình.
- Xây dựng lớp Phanso để sử dụng các khuôn hình hàm trên.

10

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
#include <iostream.h>
#include <conio.h>
class phanso
{
  private:
    int ts,ms;
  public:
    phanso();
    phanso(phanso & p);
    bool operator<(phanso & p);
    friend ostream& operator<<(ostream &mh,phanso &p);
    friend istream& operator>>(istream &bp,phanso &p);
};
```

11

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
phanso::phanso()
{
  ts=0;
  ms=1;
}
phanso::phanso(phanso & p)
{
  ts=p.ts;
  ms=p.ms;
}
bool phanso::operator<(phanso &p)
{
  return (ts*p.ms<ms*p.ts);
}
ostream&operator<<(ostream &mh,phanso &p)
{
  mh<<p.ts<<"/"<<p.ms;
  return mh;
}
```

12

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
istream&operator>>(istream &bp,phanso &p)
{
  cout<<" Nhap tu so :";
  bp>>p.ts;
  cout<<" Nhap mau so :";
  bp>>p.ms;
}
template<class T>
void swap(T &a, T &b)
{
  T tg;
  tg=a;
  a=b;
  b=tg;
}
```

13

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
template<class T>
void sort(T *a,int n)
{
  for(int i=0;i<n-1;i++)
    for(int j=i+1;j<n;j++)
       if(a[i]<a[j])
            swap(a[i],a[j]);
}
template<class T>
void print(T *a,int n)
{
  for(int i=0;i<n;i++)
   cout<<a[i]<<"   ";
  cout<<endl;
}
```

14

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
void main()
{
  int a[10],n;
  phanso b[10];
  cout<<" Nhap n "<<endl;
  cin>>n;
  cout<<" Nhap cac phan tu cho mang a "<<endl;
  for(int i=0;i<n;i++)
    {
      cout<<"a["<<(i+1)<<"]=";
      cin>>a[i];
    }
  cout<<" Nhap cac phan tu cho mang b"<<endl;
  for(int i=0;i<n;i++)
  {
     cin>>b[i];
  }
```

15

Khuôn hình hàm - Function template

Ví dụ về Function Template

```
cout<<" Cac phantu cua mang a truoc khi sap xep la:"<<endl;
print(a,n);
sort(a,n);
cout<<" Cac phantu cua mang a sau khi sap xep la:"<<endl;
print(a,n);
cout<<" Cac phantu cua mang b truoc khi sap xep la:"<<endl;
print(b,n);
sort(b,n);
cout<<" Cac phantu cua mang b sau khi sap xep la:"<<endl;
print(b,n);
getch();
}
```

16

Khuôn hình lớp - Class template

Ví dụ: Chương trình sử dụng khuôn hình (template) để thực hiện việc tìm phân số nhỏ nhất, sau đó đưa kết quả lên màn hình.

```
#include "iostream.h"
#include "conio.h"
template <class T> T min(T a, T b){
  if (a<b) return a;
  else return b;
}

class PS{
  int t,m;
public:
  PS(int t1=0,int m1=0){t=t1; m=m1;}
  void display(){
    cout<<"Phan so:"<<endl;
    cout<<t<<"/"<<m;
  }
friend int operator<(PS,PS);
};
```

17

Khuôn hình lớp - Class template

```
int operator<(PS p1,PS p2){
  return (p1.t/p1.m<p2.t/p2.m);
}
void main(){
 PS p1(1,2),p2(3,3);
 min(p1,p2).display();
getch();
}
```

18

Khuôn hình lớp

Cũng giống như khuôn hình hàm, chỉ cần định nghĩa khuôn hình lớp một lần rồi sau đó có thể áp dụng chúng với các kiểu dữ liệu khác nhau để được các thể hiện lớp khác nhau.

Tạo một khuôn hình lớp

Ví dụ xây dựng lớp diem:

```
class  diem
{        int  x , y;
    public:
        diem (int  xd = 0, int  yd = 0);
        void   hienthi ( );
}
```

Nếu muốn tọa độ điểm có kiểu dữ liệu khác (long, float, double…) thì phải định nghĩa một lớp khác bằng cách thay int bằng từ khóa tương ứng với kiểu dữ liệu mong muốn.

C++ cho phép định nghĩa một khuôn hình lớp và sau đó áp dụng khuôn hình lớp này với các kiểu dữ liệu khác nhau để thu được các lớp thể hiện như mong muốn:

```
template  <class  T> class  diem
{     T    x , y;
       public:
      diem ( T  xd = 0, T  yd = 0);
      void   hienthi ( );
}
```

Định nghĩa khuôn hình lớp:

```
template <class T1, class T2...> class <tên khuôn hình lớp>
{
        <khai báo các thành phần>
};
```

Định nghĩa hàm thành phần: có hai cách:

- Định nghĩa bên trong khai báo của khuôn hình lớp: giống như hàm thông thường.
- Định nghĩa bên ngoài khai báo: phải "nhắc lại" các tham số kiểu của khuôn hình lớp:

VD: template <class T> void diem<T>::hienthi( )
{ … }

VD:

```
template <classT> class diem
{       T x, y;
    public:
        diem(T xd = 0, T yd = 0)
                { x = xd ; y = yd; }
        void hienthi();
};
// Định nghĩa hàm thành phần ở bên ngoài khuôn hình lớp
template <class T> void diem<T>::hienthi()
{ cout<<"Diem ("<< x << " ," << y << ")\n"; }
```

- Sử dụng khuôn hình lớp:

Mỗi giá trị của tham số kiểu, chương trình dịch sẽ phát sinh ra một lớp cụ thể (lớp thể hiện của khuôn hình lớp), dạng:

<tên khuôn hình> <kiểu>

- Khai báo đối tượng lớp:

<tên khuôn hình><kiểu> <tên đối tượng> ;

Ví dụ:

diem <int> a; diem <float> b; diem <long> c;

Dòng trên khai báo đối tượng a có hai thành phần tọa độ kiểu int, đối tượng b có hai thành phần tọa độ kiểu float, đối tượng c có hai thành phần tọa độ kiểu long.

Còn diem <int>, hoặc diem <float>, hoặc diem <long> là các lớp thể hiện của khuôn hình lớp diem.

Bài tập chương 7

1/ Xây dựng hàm template tính tổng của một dãy các đối tượng của lớp, sau đó áp dụng tính tổng một dãy số nguyên, một dãy phân số. Chú ý xây dựng lớp phân số có hàm chồng toán tử +.

2/ Xây dựng hàm template sắp xếp tăng dần một dãy các đối tượng của lớp, sau đó áp dụng sắp tăng dần một dãy số nguyên, một dãy phân số. Chú ý xây dựng lớp phân số có hàm chồng toán tử so sánh >

3/ Xây dựng lớp template ngăn xếp *stack* . Áp dụng để khai báo một stack chứa các số nguyên và sử dụng stack này để đổi số từ hệ đếm 10 sang hệ đếm 2, hoặc hệ 8, hoặc hệ 16. Áp dụng lớp template stack để khai báo một stack chứa các ký tự, ứng dụng giải bài toán so khớp các dấu ngoặc đơn.