C++程序设计(II)

浙江工业大学计算机学院

1 模板函数

2 模板类

模版函数

- 回顾—重载机制实现的求绝对值

虽然名字可以相同，但是写三遍一样的代码也是负担！！！

```
int abs(int x)
{cout<<“Using integer version of abs().\n”;
  return (x>0?x:-x);
}

long abs(long x)
{cout<<“Using long version of abs().\n”;
  return (x>0?x:-x);
}

double abs(double x)
{cout<<“Using double version of abs().\n”;
  return (x>0?x:-x);
}
```

```
#include <iostream.h>
int main()
{
  cout<<abs(-5)<<endl;
  cout<<abs(-5L)<<endl;
  cout<<abs(3.14)<<endl;
}
```

❖ 模版函数

- C++允许将类型作为参数，定义一系列相关重载函数的模式。数据类型作为参数与函数的参数与普通的函数参数不同，这些类属参数必须用模板来定义。

```
template<class Type>
Type abs(Type x)
{cout<<“Using double version of abs().\n”;
  return (x>0?x:-x);
}
```

```
#include <iostream.h>
int main()
{
  cout<<abs(-5)<<endl;
  cout<<abs(-5L)<<endl;
  cout<<abs(3.14)<<endl;
}
```

- 当程序中使用模板函数（类属函数）时，编译程序将根据函数调用时的实际数据类型产生相应的函数。

❖ 模版函数

- 定义语法

```
template <模板参数表>
返回值类型  函数名(函数参数列表)
{
  //模板函数体
}
```

- 模板参数可以由typename或者class关键字给出。

```
template <typename A>
……
```

等价于

```
template<class A>
……
```

- 可有多个模板参数，用逗号分隔。

add模板可以定义为：

```
template <class T>
void add(T a[],T b[],int size){
      for(int i=0;i<size;i++) b[i]+=a[i];
}
```

- 其中，“< >”括起部分就是模板的形参表，T是一个虚拟类型参数。注意，可以用多个虚拟参数构成模板形参表。
- 不但普通函数可以声明为函数模板，类的成员函数也可以声明为函数模板。

❖ 模版函数

- 使用注意：

1)由template给出的每一个形式类属参数都必须出现在函数参数表中。

```
template<class Type>    //错误例1
Type*  func()
{ …… }

template<class Type>    //错误例2
 void  func()
 { Type obj;
 …… }
```

❖ 模版函数

- 使用注意：

2)模板函数须实例化后才能真正使用。实例化是由编译程序根据函数调用的实际参数类型自动完成的。

3) 调用模板函数时，必须保证函数的实际类型与形式类属参数完全匹配，因为编译程序在实例化过程中不作任何类型的转换。如

```
template<class T>
T  max(T x,T y)
{  return x>y?x:y;}
```

调用1:　max(2,3); //V

调用2:　max(3.5,3.14); //V

调用3:　max(3,3.14);　//X

❖ 模版函数

- 使用注意：

4) 模板函数与其他同名函数（重载关系）的调用规则：

- 如果某个函数的参数类型与函数调用的实际参数完全匹配，则调用该函数；→
- 如果能从模板函数中实例化一个函数实例，而其参数类型与函数调用的实际参数完全匹配，则调用该函数；→
- 对函数调用的实际参数作隐式类型转换后与非模板函数作匹配，找到匹配的函数则则调用它；→
- 提示语法错误。

❖模版函数

- 使用注意4)例程：

```
#include <iostream.h>
//模板函数
template<class T>
void swap(T& x, T& y)
{ T temp;
  temp=x;
  x=y;
  y=temp;
  cout<<"调用模板函数"<<endl;
  return;
}
```

```
void swap(int& x,int& y) //重载函数
{ int temp;
  temp=x;
  x=y;
  y=temp;
  cout<<"调用特殊重载函数"<<endl;
  return;      }
```

```
int main()
{ int i=10,j=20;
   float x=1.44,y=3.14;
   char a='x',b='y';
   swap(i,j);
   swap(x,y);
   swap(a,b);
   swap(i,b);
   return 0;
}
```

❖ 问题：分析程序输出结果。

```
#include <iostream.h>
#include<string.h>
template<class stype> void bubble(stype *item, int count);
void main()
{int nums1[]={4,7,2,9,3,7,6,1};
 bubble(nums1,8);
 cout<<"The sorted numbers are ";
 for(int i(0);i<8;i++)
      cout<<nums1[i]<<" ";
 cout<<endl;
 double nums2[]={2.3,5.3,6.7,3.9,7.2,1.5};
 bubble(nums2,6);
 cout<<"The sorted numbers are ";
 for(i=0;i<6;i++)
      cout<<nums2[i]<<" ";cout<<endl;}
```

```
The sorted numbers are 1 2 3 4 6 7 7 9
The sorted numbers are 1.5 2.3 3.9 5.3 6.7 7.2
```

❖ 模版类

- 使用模板定义的类称为模板类(template class)、类属类(generic class)或者参数化类(parameterized class)。
- 模板类不是真正的类类型：模板类仅描述了一组类型的通用样板，由于其所处理对象的数据类型未确定，程序员不可用模板类直接创建对象实例。
- 模板类的实例化：模板类必须经过实例化后才能成为可创建对象实例的类类型。用具体的数据类型替代模板类的类型参数。
- C++中的模板类是无约束模板类：实例化时实际的参数可以是任何类型，各个类型不要求有共同的祖先类。

模版类--语法

- template<class 模板参数1，class 模板参数2，…>

class 类名{

……

};

- 在模板外对成员函数的声明格式是：

template <模板参数表>

<返回类型> <类名> <模板参数表>::<函数名>(<函数参数表>) <函数体>

- 用类模板定义对象的格式是：

<类名><<模板实参表>><对象名>(<构造函数实参表>);

```
class  Compare_int
{
public:
  Compare_int(int a,int b)
  {x=a;y=b;}
  int max( ) {return(x>y)?x:y;}
  int min( ) {return(x<y)?x:y;}
private:
   int x,y;
};
```

```
class Compare_float
{
public:
  Compare_float(float a,float b)
  {x=a;y=b;}
  float max( ) {return(x>y)?x:y;}
  float min( ) {return(x<y)?x:y;}
private:
  float x,y;
};
```

❖模版类

- 例：声明一个类模
  数和字符的比较，

```
template<class T>
class  Compare
{
public:
  Compare (T a, T b)
  {x=a;y=b;}
  T max( ) {return(x>y)?x:y;}
  T min( ) {return(x<y)?x:y;}
private:
   T x,y;
};
```

```
template<class T>
class  Compare
{public:
  Compare (T a, T b) ;
  T max( ) ;
  T min( ) ;
private:
   T x,y;     };

template<class T>
Compare<T>::Compare(T a,T b):x(a),y(b){}
template<class T>
T Compare<T>::max()
{return(x>y)?x:y;}
template<class T>
T Compare<T>::min()
{return(x<y)?x:y;}
```

❖模版类

- 例：声明一个类模板，利用它分别实现两个整数、浮点数和字符的比较，求出大数和小数。

```
#include <iostream.h>
#include “compare.h”
int main( )
{Compare<int> cmp1(3,7); //用于两个整数的比较
cout<<cmp1.max( )<<"：整数比较最大"<<endl;
cout<<cmp1.min()<<"：整数比较最小"<<endl;
Compare<float> cmp2(45.78,93.6); //用于两个浮点数的比较
cout<<cmp2.max( )<<"：浮点比较最大"<<endl;
cout<<cmp2.min( )<<"：浮点比较最小"<<endl;
Compare<char> cmp3('a','A'); //用于两个字符的比较
cout<<cmp3.max( )<<"：字符最大"<<endl;
cout<<cmp3.min( )<<"：字符最小"<<endl;
return 0;
} //测试主程
```

```
#include<iostream>
using namespace std;
//---------------------------------
template<class T>
T f(T* a, T* b, int n){
  T s = (T)0;
  for(int i=0; i<n; i++)
    s += a[i]*b[i];
  return s;
}//---------------------------------
int main(){
  double c[4] = { 1.1, 2.2, 3.3, 4.4 };
  double d[4] = { 10, 0, 100, 0 };
  cout<<f(c, d, 4)<<endl;
}//=================================
```

3 4 1

练习与作业

编写一个**print**函数模板，它能接收一个数组的引用，能处理任意大小、任意元素类型的数组。

C++程序设计（II）

浙江工业大学计算机学院