面向对象程序设计
420420

类模板

- 1、类模板的定义

例如有如下类：

```
class   Compare_int
{
        public  :
                Compare(int   a,int  b){x=a;y=b;}//构造函数
                float  max(){return   (x>y)?x:y;}//求最大值
                float  min(){return   (x<y)?x:y;}//求最小值
        private  :
                int   x,y;
}
```

及如下类：

```
class   Compare_double
{
        public  :
                Compare(double  a,double   b){x=a;y=b;}//构造函数
                float  max(){return   (x>y)?x:y;}//求最大值
                float  min(){return   (x<y)?x:y;}//求最小值
        private  :
                double   x,y;
}
```

- 如何避免大量重复工作？

41.1 类模板的定义

- 就像可以定义函数模板一样，也可以定义类模板。其定义的一般形式为：

```
template  <模板形参表>
class  类模板名  {//类体
        成员列表
};
```

- 类模板必须以关键字template开头，后接模板形参表。模板形参表是用一对尖括号< >括住的一个或多个模板形参的列表，不允许为空，形参之间以逗号分隔。其一般形式为：

```
<class  类型参数1,  class 类型参数2, ..... >
```

41.1 类模板的定义

- 模板形参表用于表示可以在类定义中使用的数据类型。类型形参跟在关键字class或typename之后定义，如class T是名为T的类型形参，在这里class和typename没有区别。一般地，类模板习惯用class，函数模板习惯用typename。

- 除了模板形参列表外，类模板的定义与类定义相似。类模板可以定义数据成员和成员函数，也可以使用访问标号控制对成员的访问，还可以定义构造函数和析构函数等等。在类和类成员的定义中，可以使用模板形参作为类型或值的占位符，在使用类时再提供那些类型或值的具体信息。

41.1 类模板的定义

- 由于类模板包含类型参数，因此又称为参数化的类。如果说类是对象的抽象，对象是类的实例，则类模板是类的抽象，类是类模板的实例。利用类模板可以建立支持各种数据类型的类。

- 例如定义一个类模板表示平面上的点：

```
template <class T> //类模板定义
class Point { //Point不是类名是模板名
public:
        Point() : x(0), y(0) { } //默认构造函数
        Point(const T a, const T b) : x(a), y(b) { } //带参数构造函数
        void Set(const T a, const T b);          //用于给坐标重新设值
        void Display()
       { cout<<"Display: "<<"x="<<x<<",y="<<y<<endl; }
private:
        T x, y;       //私有数据成员，坐标值
};
```

- 如果在类模板外部定义成员函数，形式为：

```
template  <模板形参表>
返回类型  类名<类型参数表>::函数名(形式参数列表)
{
        函数体
}
```

- 例如：

```
template  <class  T>
void  Point<T>::Set(const  T  a,  const  T  b)
{
        x=a  ,  y=b;
}
```

- 用类模板定义对象时，必须为模板形参显式指定类型实参，一般形式为：

```
类模板名<类型实参表> 对象名列表;
类模板名<类型实参表> 对象名1(实参列表1), 对象名2(实参列表2),.....;
```

- 例如：

```
Point <int> a, b; //定义类模板对象，调用默认构造函数
Point <double> m(1,2), n(3,4); //定义类模板对象 ，调用带参数构造函数
```

- 模板形参表还可以是非类型形参，其形式与函数形参表相似。例如：

```
template <class T, int N>
class Sequence { //Sequence类模板
public:
        void Set(int i, T value);
        T Get(int i) { return array[i]; }
private:
        T array[N]; //私有数据成员，一个长度为N的T类型数组
};
template <class T, int N>
void Sequence<T,N>::Set(int i, T value) //类外定义Set成员函数
{       array[i]=value; }
```

- 当定义类模板对象时，必须为每个非类型形参提供常量表达式以供使用。

以Sequence类模板为例：

```
Sequence   <int,5>   a;              //提供类型和常量表达式
Sequence   <double,5>   b;   //提供类型和常量表达式
for(i=0;i<5;i++)   a.Set(i,i);                    //给a对象的数组成员赋值
for(i=0;i<5;i++)   cout<<a.Get(i)<<“   ”;   //输出a对象数组成员的值
```

- 模板形参还可以设置默认值。例如：

```
template <class T=char , int N=10> //类模板定义
class Sequence {…};
```

- 则对象定义时可以有以下形式：

```
Sequence <> m; //对象m使用默认类型char和使用默认值10
Sequence <double> n; //提供类型和使用默认值10
Sequence <int,100> k; //提供类型和常量表达式
```

- 标准C++为此提供了关键字export，其作用与extern相似，例如：

```
extern int n; //声明整型n，变量定义在另一个编译单元
extern struct Point p; //实例化结构体对象p，结构体定义在另一
个编译单元
extern class Stack s;  //实例化类对象s，类定义在另一个编译单元


export template<class T>class A<int> a; //实例化类模板对
象a，类模板定义在另一个编译单元
export template<class T>void f (T& t);//实例化函数模板f，
函数模板定义在另一个编译单元
```

- 这样，函数模板或类模板的实例化与定义体可以不必放在同一个编译单元了。

- 类模板在表示数组、向量、列表、队列、栈、矩阵等数据结构时，显得特别重要，因为这些数据结构的表示和算法的选择不受其所包含的元素的类型的影响。

类模板

- 2、类模板应用举例

泛型编程

- 面向过程PO（procedure oriented）、面向对象OO（object oriented）、泛型编程GP（generic programming）是三种重用的编程方法。

- 早期的C++语言泛型编程思想仅仅体现于简单的模板技术，之后的标准模板库STL（standard template library）是泛型编程思想的实际体现和具体实现。

- 面向过程是通过将代码段封装在一个函数中，通过函数调用来实现目标代码的重用。
- 面向对象是通过类的继承来实现对象目标代码的重用。
- 如果需要编写一个可用于不同数据类型的算法，可以采用的方法有：
- ①面向过程的方法：对源代码进行复制和修改，生成不同数据类型版本的算法函数，调用时需要对数据类型进行人工的判断；
- ②面向对象的方法：在一个类中，编写多个同名函数，它们的算法一致，但是所处理数据的类型不同，当然函数的参数类型也不同，通过函数重载来自动调用对应数据类型版本的函数。

- 显然，以上两种方法都需编写多个相同算法的不同函数，不能做到代码真正的重用。它们二者之间的主要差别，只是调用方便与否。如果采用泛型编程，就可以实现源代码级的重用。

41.2 类模板的应用举例

- 泛型（generic）是一种允许一个数据取不同类型的技术，与操作对象数据类型独立的算法称为泛型算法，独立于任何特定类型的编程方法称为泛型编程。

- 模板是泛型编程的基础。泛型编程关注于产生通用的软件组件，让这些组件在不同的应用场合都能很容易地重用。在C++中，类模板和函数模板是进行泛型编程极为有效的机制。

【例41.1】类模板应用举例

```
#include <iostream>
using namespace std;
template <class T> //声明一个模板，虚拟类型名为T
class Compare //类模板名为Compare
{
public :
        Compare(T a,T b){x=a;y=b;}     //构造函数
        T max(){return (x>y)?x:y;}
        T min(){return (x<y)?x:y;}
private :
        T x,y;     //T类型的私有数据成员
};
int main() {
     Compare<int> a(4,7);
     cout<<"a：max="<<a.max()<<",min="<<a.min()<<endl;
    Compare<double> b(8.34, 5.55);
    cout<<“b：max="<<b.max()<<",min="<<b.min()<<endl;
    return 0;
}
```

41.2 类模板的应用举例

【例41.2】定义数组类模板，实现不同类型数组的输入输出。

```
#include <iostream>
using namespace std;
template <class T> //声明一个模板，虚拟类型名为T
class Array          //类模板名为Array
{      public:
              Array(int c){arr = new T[c];}  //构造函数
              T& operator[](int i){return arr[i];} //对[]进行重载
       private:
              T *arr;  //私有数据成员是T类型的指针
};
int main()
{      int i;
       Array<int> array(5);  //定义整形数组对象array，长度为5
       for (i=0; i<5;i++) cin>>array[i];//输入数组各个元素的值
       for (i = 0; i < 5; i++)//输出数组各个元素的值
              cout <<"array["<<i<<"]:"<<array[i]<<"      ";
       cout<<endl;
       return 0;
}
```