类属机制——模板

多态性

- 含义：某一论域的一个元素存在多种解释
- 多态的表现
  - （1）一名多用
    - 重载（Overloading）
      - 函数名重载
      - 操作符重载
    - 覆盖（Override）：派生类可以对基类的虚函数进行重定义
  - （2）类属性（Genericity）
    - 类属函数：一个函数能对多种类型的参数进行操作
    - 类属类型：一个类型可以描述多种类型的数据

多态性的例子

- 例1：函数名重载：给多个不同的函数取相同名字的语言机制

```
int fun(int x, int y)
 {  return x-y;
 }
 double fun(double x, double y)
 { return x+y;
 }
```

以下调用返回的结果是：

```
cout<<fun(9,6);
cout<<fun('g','c');
cout<<fun(9.3,6);
```

- 注意：如果两个函数的参数类型和个数相同，只有返回值类型不同，这时不能对他们重载函数名

多态性的例子

- 例2：操作符重载

  有以下复数类定义

```
class Complex       //复数类定义
{public:
    Complex(double r=0.0,double i=0.0)
    { real=r; imag=i;
    }
       ......
  private:
    double real;
    double imag;
};
```

如何用+号实现两个复数相加？

```
Complex a(1.0,2.0),b(3.0,4.0),c;
c = a + b;
```

方法1：作为成员函数重载

```
class Complex
{ public:
    Complex operator + (const Complex& x) const
    {       Complex temp;
            temp.real = real+x.real;
            temp.imag = imag+x.imag;
            return temp;
    }
    ......
};
.....
Complex a(1.0,2.0),b(3.0,4.0),c;
c = a + b;
```

- 方法2：作为全局（友元）函数重载

```
class Complex
{ ......
   friend Complex operator + (const Complex& c1,
                              const Complex& c2);
};
Complex operator + (const Complex& c1,
                    const Complex& c2)
{  Complex temp;
   temp.real = c1.real + c2.real;
   temp.imag = c1.imag + c2.imag;
   return temp;
}
……
Complex a(1.0,2.0),b(3.0,4.0),c;
c = a + b;
```

多态性的例子

- 例3：虚函数

```
class A
{  public:
        virtual void f()
          {cout<< “A” }
        void g();
};
class B: public A
{  public:
        void f()
          {cout<< “B” }
        void g();
};
```

```
.....
void h(A& x)
{
        x.f();
}
以下调用输出什么？
A a;
B b;
h(a);  //调用A::f还是B::f?
h(b); //调用A::f还是B::f?
```

答：**h(a)输出A，h(b)输出B**

- 类属函数

  例4：如何定义一个sort（）函数，既可以对整数进行排序，也可以对实数进行排序？

- 类属类

  例：如何定义一个Stack类，既可以表示整数的栈，又可以表示实数的栈？

类属（泛型）程序设计

- 在程序设计中，经常需要用到一些功能完全相同的程序实体，但它们所涉及的数据的类型不同。

- 例如，对不同元素类型的数组进行排序的函数：

```
void int_sort(int x[],int num);
void double_sort(double x[],int num);
void A_sort(A x[],int num);
```

这三个函数的实现基本是一样的。

- 再例如，元素类型不同的栈类

```
class IntStack
{       int buf[100];
  public:
        bool push(int);
        bool pop(int&);
};
```

```
class AStack
{               A buf[100];
        public:
                bool push(A);
                bool pop(A&);
};
```

```
class DoubleStack
{               double buf[100];
        public:
                bool push(double);
                bool pop(double&);
};
```

这三个类的实现基本也是一样的。

类属（泛型）程序设计（续）

- 对于前面的三个排序函数和三个类，如果能分别只用一个函数和一个类来描述它们将会大大简化程序设计的工作。

- 在程序设计中，一个程序实体能对多种类型的数据进行操作或描述的特性称为类属性。

- 具有类属性的程序实体通常有：
  - 类属函数
  - 类属类

- 基于具有类属性的程序实体进行程序设计的技术称为：类属程序设计（或泛型程序设计，Generic Programming）。

类属函数

- 类属函数是指一个函数能对不同类型的参数完成相同的操作。

- C++提供了两种实现类属函数的机制：
  - 采用通用指针类型的参数
  - 函数模板

函数模板

- 在C++中，实现类属函数的一种办法是用函数模板。
  方法：将通用的类型在模板中说明

```
template <class T>
void sort(T elements[], unsigned int count)
{ /*
  1、取第i个元素
  elements[i]
  2、比较第i个和第j个元素的大小
  elements[i] < elements [j]
  3、交换第i个和第j个元素
  T temp=elements [i];
  elements[i] = elements [j];
  elements[j] = temp;
  */
}
```

- 函数模板是指带有类型参数的函数定义，其格式如下：

```
template <class T1, class T2, ...>
<返回值类型> <函数名>(<参数表>)
{ ......
}
```

其中，

- T1、T2等是函数模板的类型参数
- <返回值类型> 、<参数表>中的参数类型以及函数体中的局部变量的类型可以是：T1、T2等
- 使用函数模板定义的函数时需要提供与T1、T2相对应的具体类型。

函数模板的使用——实例化

- 函数模板定义了一系列重载的函数。要使用函数模板所定义的函数（称为模板函数），首先必须要对函数模板进行实例化（生成具体的函数）。
- 实例化方法1：隐式实例化

  函数模板的实例化通常是隐式的，由编译程序根据实参的类型自动地把函数模板实例化为具体的函数，这种确定函数模板实例的过程叫做模板实参推演（template argument deduction）。

例如：

```
template <class T> void sort(T elements[], unsigned int count) { … }

int a[100];
sort(a,100);  //调用void sort(int elements[], unsigned int count)

double b[200];
sort(b,200);  //调用void sort(double elements[], unsigned int count)

A c[300];
sort(c,300);  //调用void sort(A elements[], unsigned int count)
```

- 实例化方法2：显式实例化
  - 有时，编译程序无法根据调用时的实参类型来确定所调用的模板函数，这时，需要在程序中显式地实例化函数模板。
  - 显式实例化时，在调用模板函数时需要向函数模板提供实参。

例如：

```
template <class T> void sort(T elements[], unsigned int count) { … }

int a[100];
sort<int>(a,100);

double b[200];
sort<double>(b,200);

A c[300];
sort<A>(c,300);
```

- 除了类型参数外，函数模板也可以带有非类型参数。例如：

```
template <class T, int size> //size为一个int型的普通参数。
void f(T a)
{  T temp[size];
   ......
}
......
f<int,10>(1);  //调用模板函数f(int a)，该函数体中的size为10。
```

思考题

- 如何写一个max(a,b)函数，既可以求两个整数的最大值，也可以求两个实数的最大值？

类模板

- 如果一个类的成员的类型可变，则该类称为类属类。类属类一般用类模板实现。

  类属类的实现方法：将类中的通用数据在模板中说明

```
template <class T>  class Stack
{  T buffer[100];
   int top;
  public:
   Stack() { top = -1; }
   bool push(const T &x);
   bool pop(T &x);
};
template <class T> bool Stack <T>::push(const T &x)
  { ..... }          //定义push函数
template <class T> bool Stack <T>::pop(T &x)
  { ..... }          //定义pop函数
```

- 类模板的格式为：

```
template <class T1,class T2,...>
  class <类名>
  { <类成员说明>
  }
```

  - 其中，T1、T2等为类模板的类型参数，在类成员的说明中可以用T1、T2等来说明它们的类型。

类模板的使用——实例化

- 类模板定义了若干个类，在使用这些类之前，编译程序将会对类模板进行实例化。类模板的实例化需要在程序中显式地指出。例如：

```
Stack<int> st1; //创建一个元素为int型的栈对象。
int x;
st1.push(10); st1.pop(x);

Stack<double> st2; //创建一个元素为double型的栈对象。
double y;
st2.push(1.2); st2.pop(y);

Stack<A> st3; //创建一个元素为A类型的栈对象（A为定义的一个类）。
A a,b;
st3.push(a); st3.pop(b);
```

带非类型参数的类模板

- 除了类型参数外，类模板也可以包括非类型参数。 例如：

```
template <class T, int size>
class Stack
{         T buffer[size];
          int top;
  public:
          Stack() { top = -1; }
          bool push(const T &x);
          bool pop(T &x);
};
template <class T,int size>
bool Stack <T,size>::push(const T &x) { ..... }
template <class T, int size>
bool Stack <T,size>::pop(T &x) { ..... }
......
Stack<int,100> st1; //st1为元素个数最多为100的int型栈
Stack<int,200> st2; //st2为元素个数最多为200的int型栈
```

C++标准模板库（STL）

在C++标准库中，除了从C标准库保留下来的一些功能外，所提供的功能大都以模板形式给出，这些模板构成了C++的标准模板库（Standard Template Library，简称STL）。

STL主要包含：

- 常用的算法（函数）模板（如：排序函数模板、查找函数模板等）
- 容器类模板（如：集合类模板、栈类模板等）
- 迭代器类模板，实现了抽象的指针功能，用于对容器中的数据元素进行遍历和访问。

主要内容

- 多态性的含义和表现
- 类属（泛型）的基本概念
- 函数模板
- 类模板
- C++标准模板库简介