C++语言程序设计

第八章　多态性

本章主要内容

- 多态性
- 运算符重载
- 虚函数
- 纯虚函数
- 抽象类

多态性的概念

- 多态性是面向对象程序设计的重要特征之一。
- 多态性是指发出同样的消息被不同类型的对象接收时有可能导致完全不同的行为。（不同对象收到相同消息时产生不同的动作.）
- 多态的实现：
  - 函数重载
  - 运算符重载
  - 虚函数

编译时的多态

运行时的多态

问题举例——复数的运算

运算符重载

```
class complex          //复数类声明
{
public:
    complex(double r=0.0,double i=0.0)  //构造函数
    {  real=r;  imag=i;  }
    void display();     //显示复数的值
private:
    double real;
    double imag;
};
```

问题举例——复数的运算

运算符重载

用“+”、“-”能够实现复数的加减运算吗？

实现复数加减运算的方法

——重载“+”、“-”运算符

运算符重载的实质

运算符重载

- 运算符重载是对已有的运算符赋予多重含义
- 必要性
  - C++中预定义的运算符其运算对象只能是基本数据类型，而不适用于用户自定义类型（如类）
- 实现机制
  - 将指定的运算表达式转化为对运算符函数的调用，运算对象转化为运算符函数的实参。
  - 编译系统对重载运算符的选择，遵循函数重载的选择原则。

规则和限制

运算符重载

- 可以重载C++中除下列运算符外的所有运算符：

  .　　.*　　::　　?:

- 只能重载C++语言中已有的运算符，不可臆造新的。
- 不改变原运算符的优先级和结合性。
- 不能改变操作数个数。
- 经重载的运算符，其操作数中至少应该有一个是自定义类型。

两种形式

运算符重载

- 重载为类成员函数。
- 重载为非成员函数（通常为友元函数）。

运算符函数

运算符重载

- 声明形式

```
函数类型  operator 运算符（形参）
{
        ……
}
```

- 重载为类成员函数时
  参数个数=原操作数个数-1 （后置++、--除外）
- 重载为友元函数时　参数个数=原操作数个数，且至少应该有一个自定义类型的形参。

运算符重载

用成员函数重载运算符的语法形式：

1）在类的定义体内声明运算符函数

函数类型　operator 运算符（形参表）

2）定义运算符号函数

函数类型　operator 运算符（参数表）

```
{
        //定义的操作
}
```

3）重载运算符的使用：

运算符的定义后是为了方便使用，定义后的重载运算符使用起来就像原来原来运算符一样方便。

运算符成员函数的设计

运算符重载

- 双目运算符 B
  - 如果要重载 B 为类成员函数，使之能够实现表达式 oprd1 B oprd2，其中 oprd1 为A类对象，则 B 应被重载为 A 类的成员函数，形参类型应该是 oprd2 所属的类型。
  - 经重载后，表达式 oprd1 B oprd2 相当于 oprd1.operator B(oprd2)

例 8-1

运算符重载

将“+”、“-”运算重载为复数类的成员函数。

- 规则：
  - 实部和虚部分别相加减。
- 操作数：
  - 两个操作数都是复数类的对象。

```
#include<iostream>
using namespace std;
class complex    //复数类声明
{
public:     //外部接口
   complex(double r=0.0,double
   i=0.0){real=r;imag=i;}

   //构造函数
   complex operator + (complex c2); //+重载为成员函数
   complex operator - (complex c2); //-重载为成员函数
   void display();      //输出复数
private:    //私有数据成员
   double real;   //复数实部
   double imag;   //复数虚部
};
```

```
complex complex::
  operator +(complex c2) //重载函数实
  现
{
  complex c;
  c.real=c2.real+real;
  c.imag=c2.imag+imag;
  return complex(c.real,c.imag);
}
```

```
complex complex::
  operator -(complex c2)  //重载函数实
  现
{
  complex c;
  c.real=real-c2.real;
  c.imag=imag-c2.imag;
  return complex(c.real,c.imag);
}
```

```
void complex::display()
{   cout<<"("<<real<<","<<imag<<")"<<endl; }

int main()      //主函数
{ complex c1(5,4),c2(2,10),c3;  //声明复数类的
  对象
  cout<<"c1="; c1.display();
  cout<<"c2="; c2.display();
  c3=c1-c2;       //使用重载运算符完成复数减法
  cout<<"c3=c1-c2=";
  c3.display();
  c3=c1+c2;       //使用重载运算符完成复数加法
  cout<<"c3=c1+c2=";
  c3.display();
}
```

程序输出的结果为：

c1=(5,4)

c2=(2,10)

c3=c1-c2=(3,-6)

c3=c1+c2=(7,14)

运算符成员函数的设计

运算符重载

- 前置单目运算符 U
  - 如果要重载 U 为类成员函数，使之能够实现表达式 U oprd，其中 oprd 为A类对象，则 U 应被重载为 A 类的成员函数，无形参。
  - 经重载后，
    表达式 U oprd 相当于 oprd.operator U ()

运算符成员函数的设计

运算符重载

- 后置单目运算符 ++和--
  - 如果要重载 ++或--为类成员函数，使之能够实现表达式 oprd++ 或 oprd-- ，其中 oprd 为A类对象，则 ++或-- 应被重载为 A 类的成员函数，且具有一个 int 类型形参。
  - 经重载后，表达式 oprd++ 相当于 oprd.operator ++(0)

例8-2

运算符重载

- 运算符前置++和后置++重载为时钟类的成员函数。
- 前置单目运算符，重载函数没有形参，对于后置单目运算符，重载函数需要有一个整型形参。
- 操作数是时钟类的对象。
- 实现时间增加1秒钟。

```
//8_2.cpp
#include<iostream>
using namespace std;
class Clock  //时钟类声明
{
  public:       //外部接口
     Clock(int NewH=0, int NewM=0, int NewS=0);
     void ShowTime();
     Clock&  operator ++();  //前置单目运算符重载
     Clock operator ++(int);  //后置单目运算符重载
  private:      //私有数据成员
     int Hour,Minute,Second;
};
```

```
Clock& Clock::operator ++() //前置单目运算符重载函数
{ Second++;
  if(Second>=60)
  {   Second=Second-60;
      Minute++;
      if(Minute>=60)
      {
          Minute=Minute-60;
          Hour++;
          Hour=Hour%24;
      }
  }
    return *this;
}
```

```
//后置单目运算符重载
Clock Clock::operator ++(int)
{    //注意形参表中的整型参数
   Clock old=*this;
   ++(*this);
   return old;
}
```