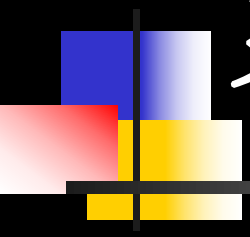
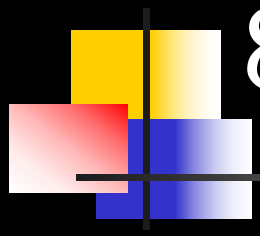
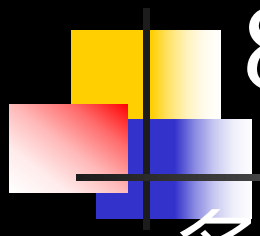


第8章 多态性-运算符重载



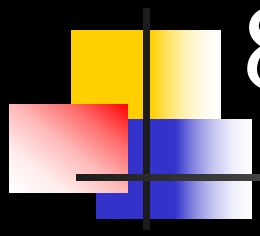
8.1 多态性概述

- 多态是一个接口具有表现多种形态的能力，一组具有相同基本语义的方法能在同一接口下为不同的对象服务。
- 按其实现的时机分为编译时多态和运行时多态两类。



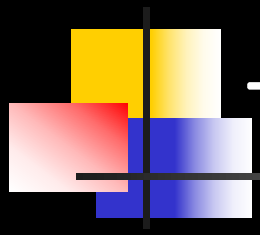
8.1.1 多态的类型

- 多态的类型
  - **函数重载多态**
  - **运算符重载多态**
  - **虚函数和抽象类**
- 根据多态性作用的时机可以分为
  - **编译时的多态**
  - **运行时的多态**



8.1.2 多态的实现

- 绑定是指将一个标识符名和一个存储地址联系在一起的过程
- 编译时的多态，绑定工作在编译连接阶段完成的情况称为静态绑定。
- 运行时的多态，绑定工作在程序运行阶段完成的情况称为动态绑定



+-运算符重载

- 运算符重载的概念
- 以成员函数的方式重载运算符的方法
- 以友元函数的方式重载运算符的方法

复数的运算

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

复数的运算

```
Complex cpx1,cpx2,cpx3
cpx1=cpx2+cpx3;
cpx1=cpx2-cpx3;
```

- 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

- 运算符重载是对已有的运算符赋予多重含义
- 实质就是函数重载

规则和限制

- 可以重载C++中除下列运算符外的所有运算符：.　.*　::　?:
- 只能重载C++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

- 重载为类的成员函数
- 重载为非成员函数

运算符函数

- **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**
- **重载为非成员函数时参数个数＝原操作数个数，且至少应该有一个自定义类型的形参。**

重载为类成员函数

- **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。

```
class Complex {
public:
   Complex(double r = 0.0, double i = 0.0)
   {real=r;imag=i; }
   Complex operator + (Complex &c2) ;
   Complex operator - (Complex &c2);
   void display() const;
private:
   double real;   //复数实部
   double imag; //复数虚部
};
```

```
Complex Complex::operator + (Complex &c2){
    //创建一个临时无名对象作为返回值
    return Complex(real+c2.real,imag+c2.imag);
}

Complex Complex::operator - (Complex &c2){
    return Complex(real - c2.real, imag - c2.imag);
}
```

```
void Complex::display() const {
   cout << "(" << real << ", " << imag << ")" << endl;
}
int main() {
   Complex c1(5, 4), c2(2, 10), c3;
   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();
   return 0;
}
```

程序输出的结果为：

c1 = (5, 4)

c2 = (2, 10)

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

c3 = c1 + c2 = (7, 14)

运算符重载为非成员函数

- 函数的形参代表依自左至右次序排列的各操作数。
- 如果在运算符的重载函数中需要操作某类对象的私有成员，可以将此函数声明为该类的友元。

运算符重载为非成员函数

- 运算符重载为非成员函数，在类中声明为友元函数
- 将+、-（双目）重载为非成员函数，并将其声明为复数类的友元，两个操作数都是复数类的常引用。

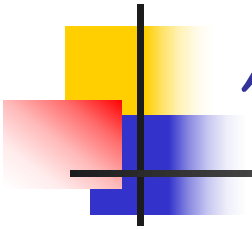
运算符重载为非成员函数

- 将<<（双目）重载为非成员函数，并将其声明为复数类的友元，它的左操作数是std::ostream引用，右操作数为复数类的常引用，返回std::ostream引用，用以支持下面形式的输出：cout << a << b;

```
#include <iostream>
using namespace std;
class Complex {      //复数类定义
public:     //外部接口
   Complex(double r = 0.0, double i = 0.0) :
   real(r), imag(i) { }      //构造函数
   friend Complex operator + (Complex &c1,
   Complex &c2);     //运算符+重载
   friend Complex operator - (Complex &c1,
   Complex &c2);     //运算符-重载
   friend ostream & operator << (ostream
   &out1, Complex &c); //运算符<<重载
private:   //私有数据成员
   double real; //复数实部
   double imag;       //复数虚部
};
```

```
Complex operator + (Complex &c1, Complex &c2) {
   return Complex(c1.real + c2.real, c1.imag+c2.imag);
}
Complex operator - (Complex &c1, Complex &c2) {
   return Complex(c1.real - c2.real, c1.imag - c2.imag);
}
ostream & operator <<(ostream &out1, Complex &c){
   out1 << "(" << c.real << ", " << c.imag << ")";
   return out1;
}
```

```
int main() {
  Complex c1(5, 4), c2(2, 10), c3;
  cout << "c1 = "<<c1;
  cout << "c2 = "<<c2;
  c3 = c1 - c2;
  cout << "c3 = c1 - c2 = "<<c3;
  c3 = c1 + c2;
  cout << "c3 = c1 + c2 = "<<c3;
  return 0;
}
```



作业