



第 8 章 (2) 运算符重载

郑 莉 清华大学

教材：C++语言程序设计（第5版） 郑莉 清华大学出版社

复习题：下列语句中，正确的纯虚函数声明是

- ☐ A `virtual int vf(int);`
- ☐ B `void vf(int)=0;`
- ☒ C `virtual void vf()=0;`
- ☐ D `virtual void vf(int) { }`

提交

预习题：下列关于运算符重载的描述中，正确的是

- ☐ A 可以改变参与运算的操作数个数
- ☐ B 可以改变运算符原来的优先级
- ☐ C 可以改变运算符原来的结合性
- ☒ D 不能改变原运算符的优先级与结合性

提交

运算符重载概述

思考

- 有如下复数类定义：

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

- 能对复数类对象使用+、-运算符吗？

运算符重载的意义

- 运算符重载是对已有的运算符赋予多重含义，使同一个运算符作用于不同类型的数据时导致不同的行为。
- 针对自定义的类，可以对原有运算符进行重载。
- 例如：
 - 使复数类的对象可以用 “+” 运算符实现加法；
 - 是时钟类对象可以用 “++” 运算符实现时间增加1秒。

运算符重载的规定

- C++ 几乎可以重载全部的运算符，而且只能够重载C++中已经有的。
 - 不能重载的运算符： `"."` 、 `"*"` 、 `"::"` 、 `"?:"`
- 重载之后运算符的优先级和结合性都不会改变。
- 可以重载为类的非静态成员函数。
- 可以重载为非成员函数（必要时可以声明为友元）。

运算符重载为成员函数

例8-1 复数类加减法运算重载为成员函数

- 要求:
 - 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。

例8-1 复数类加减法运算重载为成员函数

```
#include <iostream>
using namespace std;
class Complex {
public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) { }
    //运算符+重载成员函数
    Complex operator + (const Complex &c2) const;
    //运算符-重载成员函数
    Complex operator - (const Complex &c2) const;
    void display() const;      //输出复数
private:
    double real; //复数实部
    double imag;  //复数虚部
};
```



例8-1 复数类加减法运算重载为成员函数

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

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

例8-1 复数类加减法运算重载为成员函数

```
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;  
}
```

输出的结果为：

c1 = (5, 4)

c2 = (2, 10)

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

c3 = c1 + c2 = (7, 14)



重载为类成员的运算符函数定义形式

函数类型 operator 运算符 (形参)

{

.....

}

参数个数=原操作数个数-1 (后置++、--除外)

双目运算符重载规则

- 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)`

思考：

- 单目运算符前置语后置++、--如何区分呢？

例8-2重载前置++和后置++为时钟类成员函数

- 前置单目运算符，重载函数没有形参
- 后置++运算符，重载函数需要有一个int形参
- 操作数是时钟类的对象。
- 实现时间增加1秒钟。

例8-2重载前置++和后置++为时钟类成员函数

```
#include <iostream>
using namespace std;
class Clock { //时钟类定义
public:
    Clock(int hour = 0, int minute = 0, int second = 0);
    void showTime() const;
    //前置单目运算符重载
    Clock& operator ++ ();
    //后置单目运算符重载
    Clock operator ++ (int);
private:
    int hour, minute, second;
};
```



例8-2重载前置++和后置++为时钟类成员函数

```
Clock::Clock(int hour, int minute, int second) {  
    if (0 <= hour && hour < 24 && 0 <= minute &&  
        minute < 60  
            && 0 <= second && second < 60) {  
        this->hour = hour;  
        this->minute = minute;  
        this->second = second;  
    } else  
        cout << "Time error!" << endl;  
}  
void Clock::showTime() const { //显示时间  
    cout << hour << ":" << minute << ":" << second <<  
endl;  
}
```



例8-2重载前置++和后置++为时钟类成员函数

```
Clock & Clock::operator ++ () {  
    second++;  
    if (second >= 60) {  
        second -= 60; minute++;  
        if (minute >= 60) {  
            minute -= 60; hour = (hour + 1) % 24;  
        }  
    }  
    return *this;  
}
```

```
Clock Clock::operator ++ (int) {  
    //注意形参表中的整型参数  
    Clock old = *this;  
    ++(*this);    //调用前置 “++” 运算符  
    return old;  
}
```



例8-2重载前置++和后置++为时钟类成员函数

```
int main() {  
    Clock myClock(23, 59, 59);  
    cout << "First time output: ";  
    myClock.showTime();  
    cout << "Show myClock++:  ";  
    (myClock++).showTime();  
    cout << "Show ++myClock:  ";  
    (++myClock).showTime();  
    return 0;  
}
```

运行结果:

First time output: 23:59:59

Show myClock++: 23:59:59

Show ++myClock: 0:0:1



前置单目运算符重载规则

- 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()

后置单目运算符 ++ 和 -- 重载规则

- 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)`

运算符重载为非成员函数

有些运算符不能重载为成员函数，例如二元运算符的左操作数不是对象，或者是不能由我们重载运算符的对象

例8-3

重载Complex的加减法和 “<<” 运算符为非成员函数

- 将+、-（双目）重载为非成员函数，并将其声明为复数类的友元，两个操作数都是复数类的常引用。
- 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;
```

该输出调用的是：

```
operator << (operator << (cout, a), b);
```


例8-3 重载复数的加减法和 “<<” 运算符为非成员函数

```
//8_3.cpp
#include <iostream>
using namespace std;

class Complex {
public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) { }
    friend Complex operator+(const Complex &c1, const Complex &c2);
    friend Complex operator-(const Complex &c1, const Complex &c2);
    friend ostream & operator<<(ostream &out, const Complex &c);
private:
    double real; //复数实部
    double imag; //复数虚部
};
```



例8-3 重载复数的加减法和 "<<" 运算符为非成员函数

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



例8-3 重载复数的加减法和 "<<" 运算符为非成员函数

```
int main() {  
    Complex c1(5, 4), c2(2, 10), c3;  
    cout << "c1 = " << c1 << endl;  
    cout << "c2 = " << c2 << endl;  
    c3 = c1 - c2; //使用重载运算符完成复数减法  
    cout << "c3 = c1 - c2 = " << c3 << endl;  
    c3 = c1 + c2;      //使用重载运算符完成复数加法  
    cout << "c3 = c1 + c2 = " << c3 << endl;  
    return 0;  
}
```

运算符重载为非成员函数的规则

- 函数的形参代表依自左至右次序排列的各操作数。
- 重载为非成员函数时
 - 参数个数=原操作数个数（后置++、--除外）
 - 至少应该有一个自定义类型的参数。
- 后置单目运算符 ++和--的重载函数，形参列表中要增加一个int，但不必写形参名。
- 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的规则

- 双目运算符 B重载后,
表达式 `opr1 B opr2`
等同于 `operator B(opr1,opr2)`
- 前置单目运算符 B重载后,
表达式 `B oprd`
等同于 `operator B(oprd)`
- 后置单目运算符 ++和--重载后,
表达式 `opr B`
等同于 `operator B(oprd,0)`

小结

- 主要内容
 - 多态性的概念、运算符重载、虚函数、纯虚函数、抽象类、override 和 final
- 达到的目标
 - 掌握运算符重载原理和方法
 - 理解动态多态性的原理，掌握通过虚函数实现的多态性的方法
 - 掌握纯虚函数和抽象类的概念和设计方法