第13周 运算符重载

什么是运算符重载

- 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。

运算符重载规则

- C++ 几乎可以重载全部的运算符，但**只能够重载C++中已经有的运算符**
- 重载之后运算符的优先级和结合性都不会改变，操作数个数也不会改变
- 运算符重载是针对新类型数据的实际需要，对原有运算符进行适当的改造
- 两种重载方式：重载为类成员函数和重载为友元函数
- 重载运算符的函数不能有默认的参数，否则就改变了运算符参数的个数

运算符重载规则

- C++中可以被重载的操作符：

+	-	*	/	%	^	&	|	~
!	,	=	<	>	<=	>=	++	--
<<	>>	==	!=	&&	||	+=	-=	/=
%=	^=	&=	|=	*=	<<=?	>>=	[]	0
->	->*	new	new[]	delete	delete[]			

- C++中不能被重载的操作符：
  - “.”、“.*”、“::”、“? :”
- C++要求，重载时，赋值"="、下标"[]"、调用"()"和成员访问箭头"->"操作符必须被重载为类的成员函数

运算符重载为成员函数

- 声明形式

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

运算符前后可以有空格，也可以没有空格

- 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

运算符重载为成员函数

- 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)

这是理解和应用函数重载的关键

例：复数类加减法运算符重载

- 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。

例：复数类加减法运算符重载

```
#ifndef Complex_hpp
#define Complex_hpp
class Complex       //复数类定义
{
public:     //外部接口
    //构造函数
     Complex(double r = 0.0, double i = 0.0);
     //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;
     //输出复数
     void display() const;
private:     //私有数据成员
     //复数实部
     double m_real;
     //复数虚部
     double m_imag;
};

#endif /* Complex_hpp */
```

+、-运算符：
返回值为什么是Complex？
为什么有const后缀？

例：复数类加减法运算符重载

```
#include "Complex.hpp"
#include <iostream>
#include <iomanip>
using namespace std;
//构造函数
Complex::Complex(double r, double i) {
    m_real = r;
    m_imag = i;
}
//运算符+重载成员函数
Complex Complex::operator + (const Complex& c2) const{
    return Complex(m_real - c2.m_real, m_imag - c2.m_imag);
}
//运算符-重载成员函数
Complex Complex::operator - (const Complex& 2) const {
    return Complex(m_real - c2.m_real, m_imag - c2.m_imag);
}
//输出复数
void Complex::display() const {
    cout << m_real << setiosflags(ios::showpos) << m_imag << "i" << endl;
}
```

运算符重载为成员函数

- **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()

运算符重载为成员函数

后置单目运算U

- 如果要重载 U为类成员函数，使之能够实现表达式 oprd ，其中 oprd 为A类对象，则 U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：前、后自增运算符重载为成员函数

- 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

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
- 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

不是形参所属类的成员函数，
但具有访问其非公有成员的特权

运算符重载为非成员函数

- 双目运算符 B重载后，

  表达式oprd1 B oprd2

  等同于operator B(oprd1,oprd2)

- 前置单目运算符 U重载后，

  表达式 U oprd

  等同于operator U(oprd)

- 后置单目运算U重载后，

  表达式 oprd U

  等同于operator U(oprd,0)

以友元函数形式重载Complex的加减法运算和<<”运算符

- 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);
```

以友元函数形式重载Complex的加减法运算和<<”运算符

```
#ifndef Complex_hpp
#define Complex_hpp
#include <iostream>
using namespace std;
class Complex       //复数类定义
{
public:     //外部接口
    //构造函数
    Complex(double r = 0.0, double i = 0.0);
    //运算符+重载为友元函数
    friend Complex operator + (const Complex& c1, const Complex& c2);
    //运算符-重载友元函数
    friend Complex operator - (const Complex& c1, const Complex& c2);
    //输出运算符<<
    friend ostream& operator << (ostream& out, const Complex& c);
private:     //私有数据成员
    //复数实部
    double m_real;
    //复数虚部
    double m_imag;
};
#endif /* Complex_hpp */
```

以友元函数形式重载Complex的加减法运算和<<”运算符

```
#include "Complex.hpp"
#include <iostream>
#include <iomanip>
using namespace std;
//构造函数
Complex::Complex(double r, double i) {
    m_real = r;
    m_imag = i;
}

Complex operator + (const Complex& c1, const Complex& c2){
    return Complex(c1.m_real - c2.m_real, c1.m_imag - c2.m_imag);
}

Complex operator - (const Complex& c1, const Complex& c2){
    return Complex(c1.m_real - c2.m_real, c1.m_imag - c2.m_imag);
}

ostream& operator << (ostream& out, const Complex& c){
    out << c.m_real << setiosflags(ios::showpos) << c.m_imag << "i";
    return out;
}
```