



|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 20009200713 | **姓名** | 曾凡浩 |
| **班级** | **2003052** | **任课教师** | **张淑平** |
| **实验名称** | 第5次实验 | | |
| **实验学期** | **2021 – 2022 学年第2学期** | | |
| **实验日期** | 2022年5月30日 | **实验地点** |  |
| **报告成绩** |  | | |

西安电子科技大学 计算机科学与技术学院

# 实验目的

通过使用类、操作符重载机制设计并实现一些程序，了解并熟悉使得自定义类型更便于使用的手段，加深对基于类的操作符重载机制的理解，进一步提高面向对象编程能力。

# 实验环境

操作系统：Window 10

开发工具：Visual Studio2020

# 实验内容

## Complete and test class Complex of §11.3

## Define a type Vec4 as a vector of four floats. Define operator[] for Vec4. Define operators +, -, \*, /, =, +=, -=, \*=, and /= for combinations of vectors and floating-point numbers

# 数据结构与算法说明

## Complete and test class Complex of §11.3

**模块结构及文件组织设计：**

模块1：主控模块，仅包括文件main.cpp ，定义了 main()函数。

模块2：复数模块，包括Complex.cpp和Complex.h

Complex.h: 定义复数的成员和各种运算符的重载以及构造函数

Complex.cpp: 各种接口的实现

**关键数据结构设计：**

class Complex

{

private:

double real, imagine;

public:

Complex();

Complex(double);

Complex(double ,double);

Complex operator+(double);

Complex operator-(double);

Complex operator\*(double);

Complex operator+(const Complex&);

Complex operator-(const Complex&);

Complex operator\*(const Complex&);

void operator=(const Complex&);

void operator=(double);

bool operator==(const Complex &);

friend ostream& operator<<(ostream&, Complex&);

friend Complex operator-(Complex &);

friend Complex operator+(double, Complex&);

friend Complex operator\*(double, Complex&);

friend Complex operator-(double, Complex&);

friend istream& operator>>(istream&, Complex&);

};

Complex operator+(double,Complex&);

Complex operator-(double, Complex);

Complex operator-(Complex&);

Complex operator\*(double, Complex);

ostream& operator<<(ostream &,Complex&);

## **Define a type Vec4 as a vector of four floats. Define operator[] for Vec4. Define operators +, -, \*, /, =, +=, -=, \*=, and /= for combinations of vectors and floating-point numbers**

**模块结构及文件组织设计：**

模块1：main函数作为测试使用

模块2：包含Vec4.h和Vec4.cpp

**关键数据结构设计：**

数据结构：

class Vet4

{

private:

float f1, f2, f3, f4;

public:

Vet4();

Vet4(float, float, float, float);

Vet4 operator+(const Vet4&);

Vet4 operator-(const Vet4&);

Vet4 operator\*(const Vet4&);

Vet4 operator/(const Vet4&);

Vet4& operator+=(const Vet4&);

Vet4& operator-=(const Vet4&);

Vet4& operator\*=(const Vet4&);

Vet4& operator/=(const Vet4&);

Vet4 operator=(const Vet4&);

Vet4 operator=(float);

float operator[](int);

friend ostream& operator<<(ostream&, Vet4&);

};

ostream& operator<<(ostream&, Vet4&);

# 测试用例与测试结果

## Complete and test class Complex of §11.3

|  |  |  |
| --- | --- | --- |
| 序号 | 测试数据 | 输出 |
| 1 | Complex a = Complex(1, 2);  Complex b = 3;  Complex c = a + 2.3;  Complex d = 2 + b;  Complex e = -b - c;  Complex f = c \* 2 \* c;  cout << "a:" << a << endl;  cout << "b:" << b << endl;  cout << "c:" << c << endl;  cout << "d:" << d << endl;  cout << "e:" << e << endl;  cout << "f:" << f << endl;  if (a == b) {  cout << "a equal b" << endl;  }  else cout << "a not equal b" << endl;  Complex g(1,2);  cout << "g:" << g << endl;  if (a == g) {  cout << "a equal g" << endl;  }  else cout << "a not equal g" << endl; | a:1+2i  b:3+0i  c:3.3+2i  d:5+0i  e:-6.3-2i  f :21.78+8i  a not equal b  g:1+2i  a equal g |

## **Define a type Vec4 as a vector of four floats. Define operator[] for Vec4. Define operators +, -, \*, /, =, +=, -=, \*=, and /= for combinations of vectors and floating-point numbers**

|  |  |  |
| --- | --- | --- |
| 序号 | 测试数据 | 输出 |
| 1 | Vet4 v1(1, 2, 3, 4);  Vet4 v2(0.1, 0.2, 0.3, 0.4);  Vet4 v;  cout << "v1:" << v1<<endl;  cout << "v2:" << v2<<endl;  v = v1 + v2;  cout <<"(v1+v2):" << v<<endl;  v = v1 - v2;  cout << "(v1-v2):" << v << endl;  v = v1 \* v2;  cout << "(v1\*v2):" << v << endl;  v = v1 / v2;  cout << "(v1/v2):" << v << endl;  v = v1;  v += v2;  cout << "(v1+=v2):" << v << endl;  v -= v2;  cout << "(v1-=v2):" << v << endl;  v \*= v2;  cout << "(v1\*=v2):" << v << endl;  v /= v2;  cout << "(v1/=v2):" << v << endl;  cout << "v1[0]=" << v1[0] << endl;  cout << "v1[1]=" << v1[1] << endl;  cout << "v1[2]=" << v1[2] << endl;  cout << "v1[3]=" << v1[3] << endl;  cout << "v1[4]=" << v1[4] << endl; | v1:f1=1 f2=2 f3=3 f4=4  v2:f1=0.1 f2=0.2 f3=0.3 f4=0.4  (v1+v2):f1=1.1 f2=2.2 f3=3.3 f4=4.4  (v1-v2):f1=0.9 f2=1.8 f3=2.7 f4=3.6  (v1\*v2):f1=0.1 f2=0.4 f3=0.9 f4=1.6  (v1/v2):f1=10 f2=10 f3=10 f4=10  (v1+=v2):f1=1.1 f2=2.2 f3=3.3 f4=4.4  (v1-=v2):f1=1 f2=2 f3=3 f4=4  (v1\*=v2):f1=0.1 f2=0.4 f3=0.9 f4=1.6  (v1/=v2):f1=1 f2=2 f3=3 f4=4  v1[0]=1  v1[1]=2  v1[2]=3  v1[3]=4  biggest index is 3 !!! v1[4]=3.40282e+38 |

# 实验总结

在本次实验当中基本理解了什么是操作符的重载，在开始的时候也会有着疑惑，为什么要进行操作符的重载，难道说库函数里面提供的操作符难道不能够满足我们的需要吗？其实这是因为之前都没怎么使用自定义类型，都是用的基本数据类型，对于基本数据类型来说，基础的操作符已经足够使用了，但是对与自定义类型来说基本操作符必然不能很好的满足用户的需要。比如说输出用的<<操作符，其右值只能是基本数据类型，但若是想输出自定义的类型怎么办？用对象+ “.”+类的数据成员？但是类的数据成员又是private，在类的外面是不可访问的必然会报错，那么这个时候我们要是想输出我们的自定义类型的相应数据就必须重载运算符<<。重载运算符是不能修改其本身的结构特征的，比如说“+”运算符来说，他必须是有两个操作数，不能说在重载之后只有一个或者多个参数，这肯定也是不行的。重载操作符的函数是放在成员函数还是非成员函数呢？这其实是一个很重要的问题，当重载函数作为成员函数的时候其首个参数会默认为\*this，即一定是我们的自定义类型，我们只需要将另一个参数传入。但这又有一个问题，那如果我们要实现一个基本类型+自定义类型怎么办？这个时候就只能将重载函数作为非成员函数，只有这样才能显示传入各个参数（需要几个参数就传几个）。这个时候无疑又产生了一个新的问题，那就是当我们定义为非成员函数的时候就无法直接访问类中定义的private数据成员了，这个时候也就引入了friend函数，我们可以声明非成员函数作为类的友元，这样在友元函数中就可以直接访问数据成员了。但这其实也是会带来一个问题，我们把数据成员定义为private的原因就是不让非成员函数访问数据成员这样就可以做到封装，但无疑引入友元函数就在一定程度上破坏了类的封装特性，所以友元函数是宁缺勿泛的。