



|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 18040400054 | **姓名** | **周师扬** |
| **班级** | **1803054** | **任课教师** | **张淑平** |
| **实验名称** | 类与操作符重载 | | |
| **实验学期** | **2019 – 2020 学年第2学期** | | |
| **实验日期** | 选择日期 | **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、操作符重载机制来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：[MacOS]

开发工具：[clion]

# 实验内容

## **题目一：Complete and test class Complex of §11.3 (Page267-272)**

## **题目二：定义类Vec4及操作**

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

## **题目三：定义指针类**

Complete class ***Ptr\_to\_T*** from §11.11 and test it. To be complete, ***Ptr\_to\_T*** must have at least the operators \*, ->, =, ++, and -– define. Do not cause a run-time error until a wild pointer is actually dereferenced.

## **题目五：定义基本类型的包装类**

# 数据结构与算法设计

## **题目一：Complete and test class Complex of §11.3 (Page267-272)**

**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，Complex类。

**关键数据结构设计：**

class Complex {  
public:  
 Complex(double r = 0, double i = 0) : re(r), im(i) {};  
  
 ~Complex() {};  
  
 Complex(const Complex &c) : re(c.re), im(c.im) {};  
  
 Complex &operator+=(Complex a);  
  
 Complex &operator-=(Complex a);  
  
 Complex &operator+=(double b);  
  
 Complex &operator-=(double b);  
  
 Complex operator-() {  
 return Complex(-re, -im);  
 }  
  
 friend ostream &operator<<(ostream &out, Complex &object) {  
 out << setiosflags(ios::showpos) << object.re << setiosflags(ios::showpos) << object.im << 'i' << endl;  
 return out;  
 }  
  
private:  
 double re, im;  
};

**算法1.1 int main()**

作 用：主控函数，也实现对题目所需其他内容的测试。

参 数：无参数。

返回值：总是返回0。

计算过程：

1. 测试构造函数
2. 测试重载运算符

**算法1.2 Complex &Complex::operator+=(Complex a)（其余重载不再赘述）**

作 用：重载+=运算

参 数：Complex a

返回值：Complex &

计算过程：

1. This->re += a.re
2. This->im += a.im
3. 返回引用

## **题目二：定义类Vec4及其操作**

**模块结构及文件组织设计：**

模块1：主控模块：仅包含文件main.cpp，定义了main()函数，用于测试所编写的头文件。

模块2：头文件模块：包含Vec4.hpp和Vec4.cpp

内部封装了对于Vec4的向量操作。

**关键数据结构设计：**

class Vec4 {  
private:  
 float floats[4] = {0};  
public:  
 Vec4();  
  
 Vec4(const Vec4 &);  
  
 explicit Vec4(float \*);  
  
public:  
 Vec4 &operator=(Vec4);  
  
 friend Vec4 &operator+(Vec4, float);  
  
 friend Vec4 &operator+(Vec4, Vec4);  
  
 friend Vec4 &operator-(Vec4, float);  
  
 friend Vec4 &operator-(Vec4, Vec4);  
  
 friend Vec4 &operator\*(Vec4, float);  
  
 friend Vec4 &operator\*(Vec4, Vec4);  
  
 friend Vec4 &operator/(Vec4, float);  
  
 friend Vec4 &operator/(Vec4, Vec4);  
  
 Vec4 &operator+=(float);  
  
 Vec4 &operator+=(Vec4);  
  
 Vec4 &operator-=(float);  
  
 Vec4 &operator-=(Vec4);  
  
 Vec4 &operator\*=(float);  
  
 Vec4 &operator\*=(Vec4);  
  
 Vec4 &operator/=(float);  
  
 Vec4 &operator/=(Vec4);  
  
 float &operator[](int);  
  
 friend istream &operator>>(istream &, Vec4 &);  
  
 friend ostream &operator<<(ostream &, Vec4 &);

};

**算法2.1 int main()**

作 用： 测试编写的头文件

参 数： 无参数

返回值： 总是返回0

计算过程：

1. 申请Vec4类型的变量a，b
2. 测试重载输入流
3. 测试重载运算符
4. 输出验证

**算法2.2 Vec4 &operator+()（其余重载不再赘述）**

作 用： 重载+运算符

参 数： **Vec4 a, float det**

返回值： **Vec4 &**

计算过程：

1. 使用遍历使所有的向量+=det
2. 返回修改后的Vec4

**算法2.3 float &Vec4::operator[]()**

作 用： 重载**[]**运算符

参 数： **int a**

返回值： **float &**

计算过程：

1. 返回**Vec4**中**floats**的对应的数组值。

## **题目三：定义指针类**

**模块结构及文件组织设计：**

模块1:主控模块：仅包含文件main.cpp

定义了struct T，类模版Ptr\_to\_T，以及mian()函数。

**关键数据结构设计：**

struct T {  
 M m;  
  
 T() {};  
  
 T(M v) { m = v; }  
};  
  
template<class M>  
class Ptr\_to\_T {  
 T<M> \*p;  
 T<M> \*array;  
 int size;  
 int index = 0;  
public:  
 Ptr\_to\_T(T<M> \*, T<M> \*, int);  
  
 T<M> \*operator->() { return p; }  
  
 T<M> &operator\*() {  
 return \*p;  
 }  
  
 T<M> &operator[](int i) { return p[i]; }  
  
 void operator--() {  
 this->p--;  
 this->index--;  
 if (index < 0) {  
 throw 1;  
 }  
 }  
  
 void operator++() {  
 this->p++;  
 this->index++;  
 if (index > size - 1) {  
 throw 2;  
 }  
 }  
};

**算法3.1 int main()**

作 用： 测试编写的模版函数

参 数： 无参数

返回值： 始终返回0；

计算过程：

1. 申请int类型的模版类型T
2. 测试编写的函数，进行输出。

**算法3.2 Ptr\_to\_T<M>::Ptr\_to\_T()**

作 用： 构造

参 数： **T<M> \*a, T<M> \*b, int Size**

返回值： 无返回值

计算过程：

1. P = a ; array = b ; size = Size;
2. 使用地址差/每个数据长度来定位指针（index = (a - b) / sizeof(b)）;
3. 构造完毕。

## **题目五：定义基本类型的包装类**

**模块结构及文件组织设计：**

模块1:主控模块：仅包含文件main.cpp

模块2:头文件模块：包含文件Interger.h

**关键数据结构设计：**

class Interger {  
 int ii;  
public:  
 Interger(int v = 0) : ii(v) {}; //构造函数  
 Interger(const char \*c\_str) {  
 ii = atoi(c\_str);  
 }; //构造函数  
 Interger(const Interger &rhs) {  
 ii = rhs.ii;  
 } //复制操作  
 Interger &operator=(const Interger &rhs) {  
 this->ii = rhs.ii;  
 return \*this;  
 } //复制操作  
 Interger &operator=(int n) {  
 this->ii = n;  
 return \*this;  
 }  
  
 std::string toString() const {  
 string c\_str;  
 stringstream ss;  
 ss << this->ii;  
 ss >> c\_str;  
 return c\_str;  
 } //转换字符串函数  
  
 int value() const {  
 return int(this->ii);  
 } //获取值函数  
  
 operator int() const {  
 return int(this->ii);  
 } //int类型转换  
  
 operator double() const {  
 return double(this->ii);  
 } //double类型转换  
  
 operator char() const {  
 return char(this->ii);  
 } //char类型转换  
  
 int cmp(const Interger &rhs) {  
 if (this->ii < rhs.ii) {  
 return -1;  
 } else if (this->ii = rhs.ii) {  
 return 0;  
 } else {  
 return 1;  
 }  
 } //cmp比较函数  
  
 friend bool operator<(Interger a, const Interger &b) {  
 return a.cmp(b) == -1;  
 } //重载二元运算符6种  
  
 friend bool operator<=(Interger a, const Interger &b) {  
 return a.cmp(b) != 1;  
 }  
  
 friend bool operator>(Interger a, const Interger &b) {  
 return a.cmp(b) == 1;  
 }  
  
 friend bool operator>=(Interger a, const Interger &b) {  
 return a.cmp(b) != -1;  
 }  
  
 friend bool operator==(Interger a, const Interger &b) {  
 return a.cmp(b) == 0;  
 }  
  
 friend bool operator!=(Interger a, const Interger &b) {  
 return a.cmp(b) != 0;  
 }  
  
 Interger &operator+=(const Interger &a) {  
 ii += a.ii;  
 return \*this;  
 } //重载一元运算符8种  
  
 Interger &operator-=(const Interger &a) {  
 ii -= a.ii;  
 return \*this;  
 }  
  
 Interger &operator\*=(const Interger &a) {  
 ii \*= a.ii;  
 return \*this;  
 }  
  
 Interger &operator/=(const Interger &a) {  
 ii /= a.ii;  
 return \*this;  
 }  
  
 Interger &operator+=(const int &a) {  
 ii += a;  
 return \*this;  
 }  
  
 Interger &operator-=(const int &a) {  
 ii -= a;  
 return \*this;  
 }  
  
 Interger &operator\*=(const int &a) {  
 ii \*= a;  
 return \*this;  
 }  
  
 Interger &operator/=(const int &a) {  
 ii /= a;  
 return \*this;  
 }  
  
 friend Interger &operator+(Interger a, const Interger &b) {  
 a.ii += b.ii;  
 return a;  
 } //重载二元运算符4种  
  
 friend Interger &operator-(Interger a, const Interger &b) {  
 a.ii -= b.ii;  
 return a;  
 }  
  
 friend Interger &operator\*(Interger a, const Interger &b) {  
 a.ii \*= b.ii;  
 return a;  
 }  
  
 friend Interger &operator/(Interger a, const Interger &b) {  
 a.ii /= b.ii;  
 return a;  
 }  
  
 friend istream &operator>>(istream &in, Interger &object) {  
 in >> object.ii;  
 return in;  
 } //重载输入输出流函数  
  
 friend ostream &operator<<(ostream &out, Interger &object) {  
 out << object.ii << endl;  
 return out;  
 }  
};

# 测试用例与测试结果

## **题目一：Complete and test class Complex of §11.3 (Page267-272)**

**测试用例：**

Complex a = Complex(2.1, 5.2);  
Complex b(5);  
Complex c = a + 2.3;  
Complex d = 2 + b;  
Complex e = -b - c;  
cout << a << b << c << d << e;

**输出结果：**

**+2.1+5.2i**

**+5+0i**

**+4.4+5.2i**

**+7+0i**

**-9.4-5.2i**

## **题目二：定义Vec4类及其操作**

**测试用例：**

Vec4 a, b;  
cin >> a >> b;  
a += b;  
cout << a;  
a = b + 1;  
a -= 2.0;  
a = b \* 2;  
a = a \* b;  
a /= b;  
cout << a;  
a[0] = 3.1415926E+5;  
cout << a[0] << " " << a[1] << " " << a[2] << " " << a[3] << endl;

**1 2 3 4 1 2 3 4**

**输出结果：**

**2 4 6 8**

**2 4 6 8**

**314159 4 6 8**

## **题目三：定义指针类**

**测试用例：**

T<int> v[200];  
Ptr\_to\_T<int> P(&v[0], v, 200);  
try {  
 --P;  
 T<int> a = \*P;  
}  
catch (int) {  
 cerr << "Overflow!" << endl;  
}  
\*P = T<int>(0);  
P = Ptr\_to\_T<int>(&v[10], v, 200);  
P->m = 10;

**输出结果：**

**Overflow!**

## **题目五：定义基本类型的包装类**

**测试用例：**

Interger n1("2");  
cout << n1;  
Interger n2(3);  
cout << n2;  
n1 += n2;  
cout << n1;  
string c\_str = n1.toString();  
for (char i : c\_str) {  
 cout << i;  
}  
cout << endl;

**输出结果：**

**2**

**3**

**5**

**5**

# 实验总结

本次实验一共包括5个上机题

其中有一道题是在前面的选座完成，这次实验完成了其余的四道题。

题目一：对复数的操作

这道题建立在之前实现过的复数操作上，在其中新学到了如何多次重载函数，以及如何输出带符号的整数。

题目二：定义Vec4类及其操作

这道题中仍然是重载，在其中使用了[]的重载，可以使Vec4像数组一样使用

题目三：定义指针类

这道题仍然是重载，在其中使用了地址的方式去定位指针，是一种比较新奇的想法

题目五：封装基本类型类

这道题是对重载运算符的一个集大成的题，在其中涉及到了各种常用的重载运算符的操作，使得封装后的类能够像基本类型一样的使用。

总结：这次实验不仅让我深刻的理解了重载运算符的操作，而且也学习到了很多新的想法，知道了出现各种错误如何去更改，期待最后一次实验。