**西南大学计算机与信息科学学院**

**《C++》之**

**实验003： 复数**

**实验基本信息：**

未命名实验类型： □验证性 □设计性 □综合性

理论教师： 代立云 实验指导教师：代立云

学生姓名: 宋行健 学号: 222018321062006

班级： 18级软件工程一班

时间： 2019.6.1

**实验过程：**

1. 实验代码

#include <iostream>

using namespace std;

class Complex

{

// 默认为私有变量private

double real;

double imaginary;

public:

//默认构造函数将复数初始化为0+0i

Complex()

{

real = 0;

imaginary = 0;

}

//将对象的成员变量设置为任何值

void set(double re,double imag)

{

imaginary = imag;

real = re;

}

//将对象初始化为实变量+0\*i

void set(double re)

{

imaginary = 0;

real = re;

}

// 调用实部

double get\_Real(void)

{

return real;

}

// 调用虚部

double get\_Imaginary(void)

{

return imaginary;

}

// 重载 + 运算符，用于把两个 Complex 对象相加

Complex operator+(const Complex& data)

{

Complex complex\_num;

complex\_num.real = this->real + data.real;

complex\_num.imaginary = this->imaginary + data.imaginary;

return complex\_num;

}

// 重载 - 运算符，用于把两个 Complex 对象相减

Complex operator-(const Complex& data)

{

Complex complex\_num;

complex\_num.real = this->real - data.real;

complex\_num.imaginary = this->imaginary - data.imaginary;

return complex\_num;

}

// 重载 \* 运算符，用于把两个 Complex 对象相乘

Complex operator\*(const Complex& data)

{

Complex complex\_num;

complex\_num.real = this->real \* data.real - this->imaginary \* data.imaginary;

complex\_num.imaginary = this->imaginary \* data.real + this->real \* data.imaginary;

return complex\_num;

}

// 重载 == 运算符，用于比较两个 Complex 对象

bool operator ==(const Complex& data)

{

//Complex complex\_num; 【注意：关系运算符的重载，这里不需要定义结果变量】

if (this->real == data.real && this->imaginary == data.imaginary)

{

return true;

}

else

{

return false;

}

}

// 重载 >> 运算符，用于比较两个 Complex 对象

bool operator >>(const Complex& data)

{

if (this->real > data.real)

{

return true;

}

else if (this->real == data.real)

{

if (this->imaginary > data.imaginary)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

// 重载 << 运算符，用于比较两个 Complex 对象

bool operator <<(const Complex& data)

{

if (this->real < data.real)

{

return true;

}

else if (this->real == data.real)

{

if (this->imaginary < data.imaginary)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

// 展示所输入的复数

void show()

{

cout << "You Input:\t" << get\_Real() << " + " << get\_Imaginary() << " \* i" << endl;

}

};

int main()

{

Complex i;

i.set(0, 1); //初始化i为0+1i

Complex complex\_1; // 声明 Complex\_1，类型为 Complex

Complex complex\_2; // 声明 Complex\_2，类型为 Complex

Complex complex\_result; // 声明 Complex\_result，类型为 Complex

double real, imaginary; // 声明double类型的复数的两个组成部分

// Complex\_1 赋值为纯实数类型（只有一个参数，为实部）

cout << "Please intput the second complex number (One Parameter):" << endl;

cin >> real;

complex\_1.set(real); // 形成调用Complex类型私有变量的端口

complex\_1.show(); // 展示所输入的虚数

// Complex\_2 赋值为复数数类型（有两个参数，一个为实部，另一个为虚部）

cout << "\nPlease intput the first complex number (Two Parameters):" << endl;

cin >> real >> imaginary;

complex\_2.set(real, imaginary); // 形成调用Complex类型私有变量的端口

complex\_2.show(); // 展示所输入的虚数

// 把两个对象相加

complex\_result = complex\_1 + complex\_2;

cout << "\n加法：" << endl;

cout << "("<<complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) + (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i ) == (" << complex\_result.get\_Real() << " + " << complex\_result.get\_Imaginary() << " \* i )" << endl;

// 把两个对象相减

complex\_result = complex\_1 - complex\_2;

cout << "\n减法：" << endl;

cout << "(" << complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) - (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i ) == (" << complex\_result.get\_Real() << " + " << complex\_result.get\_Imaginary() << " \* i )" << endl;

// 把两个对象相乘

cout << "\n乘法：" << endl;

complex\_result = complex\_1 \* complex\_2;

cout << "(" << complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) \* (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i ) == (" << complex\_result.get\_Real() << " + " << complex\_result.get\_Imaginary() << " \* i )" << endl;

// 把两个对象相比较

cout << "\n比大小：" << endl;

if (complex\_1 << complex\_2)

{

cout << "(" << complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) < (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i )" << endl;

}

else if (complex\_1 >> complex\_2)

{

cout << "(" << complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) > (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i )" << endl;

}

else

{

cout << "(" << complex\_1.get\_Real() << " + " << complex\_1.get\_Imaginary() << " \* i ) == (" << complex\_2.get\_Real() << " + " << complex\_2.get\_Imaginary() << " \* i )" << endl;

}

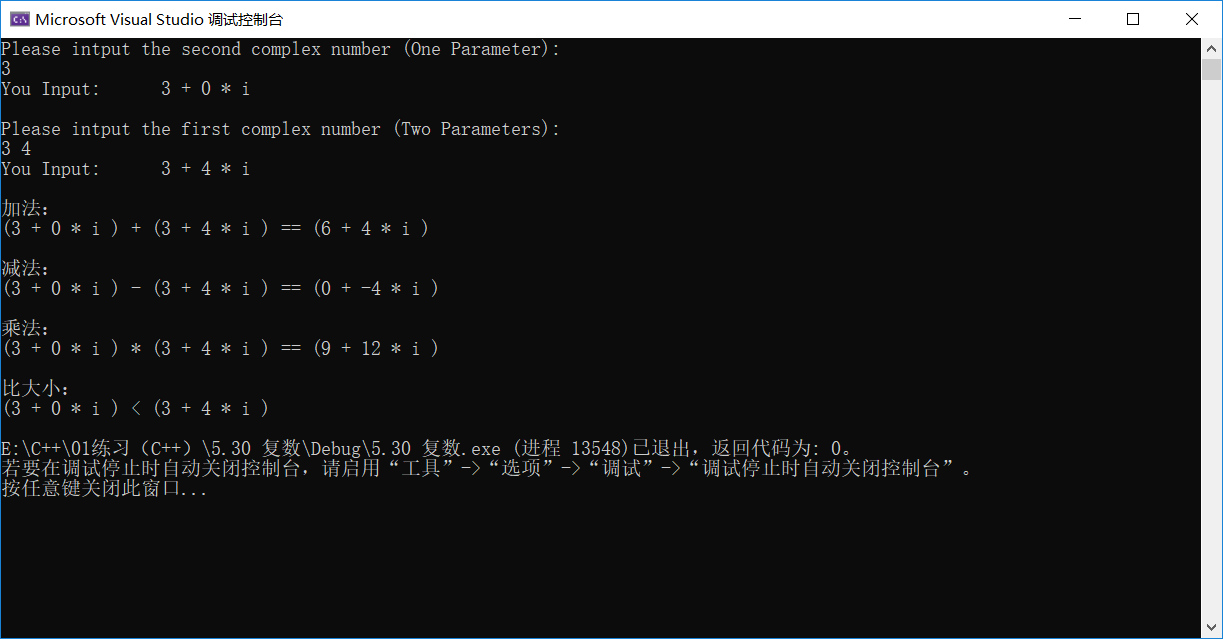
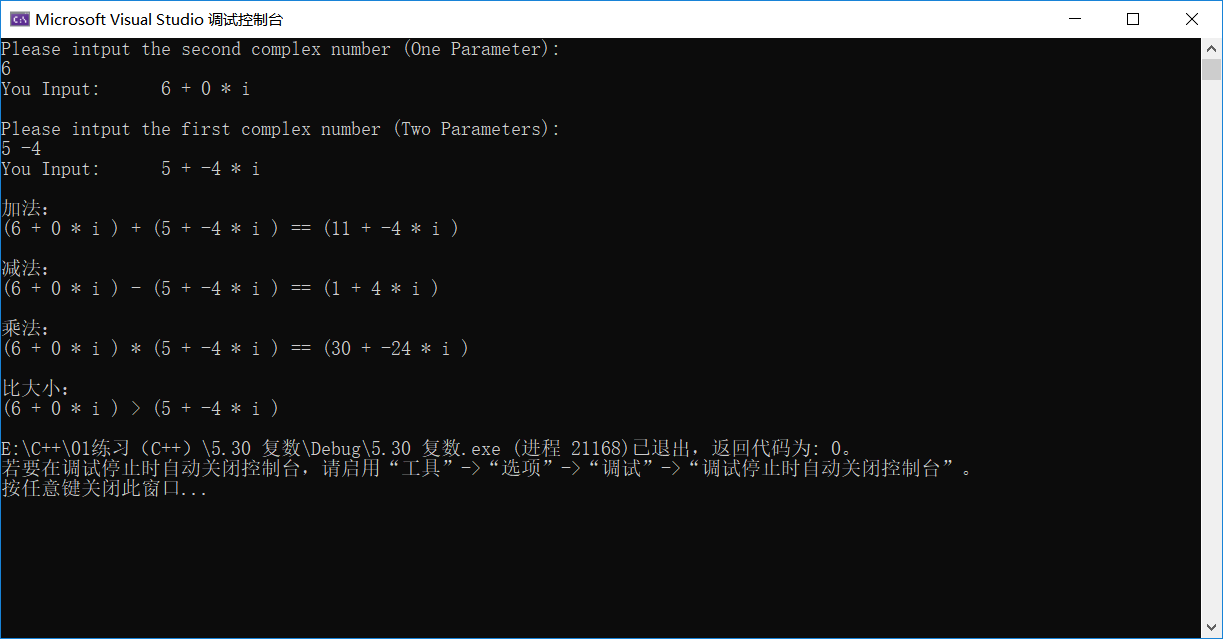
return 0;

}

1. 实验感想

通过本次实验，我更加熟练掌握了类中的默认构造函数和函数的重构。同时对类的公有函数和私有函数有了更加深刻的理解。在类的外部是不能调用他的私有函数的，所以要对类的私有写一个对外部的端口，例如： double get\_Real(void) { return real; }。通过这些端口实现了对类私有变量的调用，同时又保证了其私有变量不随意改变。

其中我认为比较困难的问题是关系运算符的重构。它与操作运算符不同的是它的返回值是bool类型的，并且不需要定义结果返回变量。另外，二元操作符的重构所需要的参数只有一个，前面的参数内容在调用函数的时候自动导入。

1. 实验截图