东 北 大 学 秦 皇 岛 分 校 计 算 中 心

**实 验 报 告**

|  |  |  |  |
| --- | --- | --- | --- |
| 实验题目：运算符重载 | | | |
| 专业：自动化类 | 班级：2301班 | 学号 ：202313653 | 姓名：焦一晟 |
| 实验日期：2024/4/8 | 机器号： | 实验得分： | 指导教师签字： |

**一、实验目的：**

1. 了解运算符重载的作用

2. 理解运算符重载的概念

3. 掌握运算符重载的几种常用方法

4. 理解转换构造函数的作用

5. 掌握转换构造函数和类型转换函数的使用**二、实验内容(习题)：**

结合 课上练习1 和课上练习2，定义复数类（Complex）、设计main实现测试类功能

类定义包含：

（1）         构造函数（可带默认值）

（2）         析构函数（判断对象生成情况）

（3）         重载加/减法+/-运算符函数（可增加\*、/）

（4）         赋值运算符=、复合赋值运算符+=函数（可增加-=、\*=）

（5）         <<输出操作符函数（可增加输入>>）

实现基本要求：

（1）         复数与复数之间的+、-、+=运算

（2）         复数输出<<

请分析并说明以下情况：（在程序开始位置注释说明）

（1）         在程序中系统创建了哪些对象？可通过调用多少次析构函数分析

（2）         函数重载中哪些引用&是可以去掉的？如果去掉系统又会增加创建哪些对象？

（3）         <<运算符重载可以设为成员函数吗？友元函数?有何区别？哪种更合适？

（4）         运算符重载函数的返回类型是Ostream &目的？

解答：

/\*

（1）程序中系统创建了三个对象：c1、c2和c3。

析构函数会在程序结束时被调用，因为这些对象是在栈上创建的，它们的作用域是main函数内

当main函数结束时，这些对象就会被销毁，因此析构函数会被调用3次。

（2）在函数重载中，重载运算符+、-、+=、<<都可以不使用引用&。

但为了提高效率和避免不必要的复制操作，通常应该使用引用&。

如果去掉引用&，系统会增加创建临时对象。

（3）<<运算符重载可以设为成员函数，但不建议。

因为它需要访问私有成员变量，所以一般情况下应该将其定义为友元函数。

友元函数可以访问类的私有成员变量，而成员函数不需要显式传递对象，但只能访问其所属类的成员。

因此，对于重载输出操作符<<来说，更合适的做法是将其定义为友元函数。

（4）运算符重载函数的返回类型是Ostream &，目的是为了支持链式调用。

这样可以在同一行代码中连续使用输出操作符<<，

例如：cout<<c1<<c2;返回类型为Ostream &允许将一个流插入到另一个流中，从而实现链式操作。

\*/

#include<iostream>

using namespace std;

class Complex{

private:

double real,imag;

public:

Complex(double r=0.0,double i=0.0):real(r),imag(i){}

~Complex(){

cout<<"实部="<<real<<"虚部="<<imag<<endl;

}

Complex operator +(const Complex &c){

return Complex(real+c.real,imag+c.imag);

}

Complex operator -(const Complex &c){

return Complex(real-c.real,imag-c.imag);

}

Complex operator \*(const Complex &c){

return Complex(real\*c.real-imag\*c.imag,real\*c.imag+imag\*c.real);

}

Complex operator =(const Complex &c){

if(this!=&c) {

real=c.real;

imag=c.imag;

}

return \*this;

}

Complex operator +=(const Complex &c){

return Complex(real+c.real,imag+c.imag);

}

Complex operator \*=(const Complex &c){

return Complex(real\*c.real-imag\*c.imag,real\*c.imag+imag\*c.real);

}

friend ostream& operator <<(ostream& os,const Complex &complex){

if(complex.imag==0){

os<<complex.real;

}

else if(complex.imag<0){

os<<complex.real<<"-"<<-complex.imag<<"i";

}

else os<<complex.real<<"+"<<complex.imag<<"i";

return os;

}

};

int main(){

Complex c1(1.68,1.28);

Complex c2(1.38,1.58);

Complex c3;

cout<<"c1="<<c1<<endl;

cout<<"c2="<<c2<<endl;

cout<<"c3="<<c3<<endl;

c3=c1+c2;

cout<<"c1+c2="<<c3<<endl;

c3=c1-c2;

cout<<"c1-c2="<<c3<<endl;

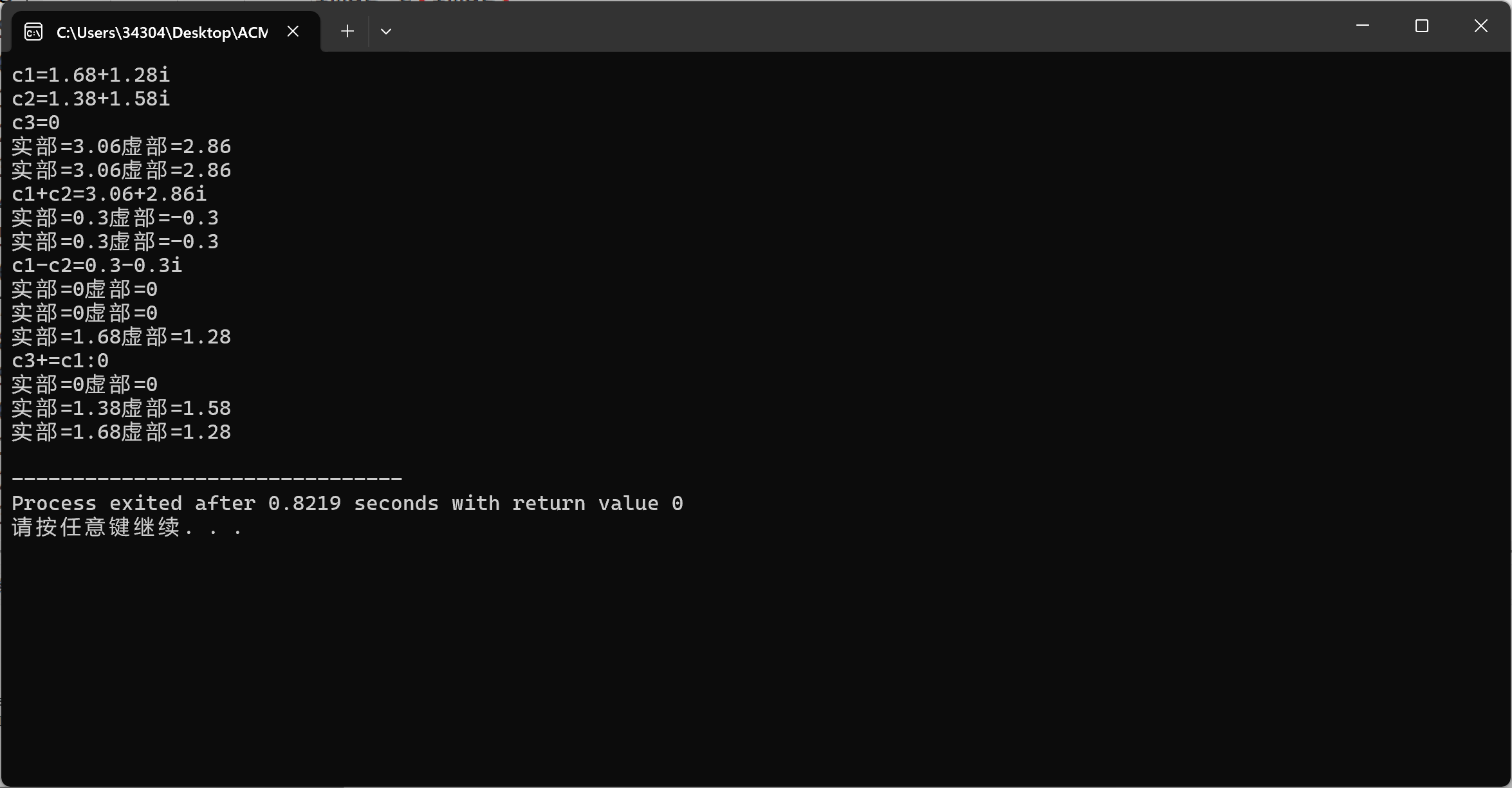
c3=(0,0);

c3+=c1;

cout<<"c3+=c1:"<<c3<<endl;

return 0;

}



**三、总结：实验操作中遇到的问题、心得体会等**

多多调试，掌握过程。