// File Name: Complex.cpp

// Author: 傅淏

// Student Number: 3016218083

// Email: 907571579@qq.com

// Assignment: 3

// Version: 1

// Description: 运用类的思想，通过运算符重载来实现对复数的一些操作

// Future Improvements: 1、缩短编程时间，熟练编程。2、注意加强注释的使用及简洁。3、尽量使用最简便的算法。

#ifndef \_\_Complex\_\_

#define \_\_Complex\_\_

#include <iostream>

class Complex{

public://public部分为对外接口，不可增加、修改或删除

Complex();//a＝0，b＝0

Complex(const double a, const double b);

Complex(const double a);//b＝0

~Complex();

const double getRealPart() const;//返回实部

const double getImaginaryPart() const;//返回虚部

Complex add(const Complex sc) const;//加法，sc作为第二操作数

Complex subtract(const Complex sc) const;//减法，sc作为第二操作数

Complex multiply(const Complex sc) const;//乘法，sc作为第二操作数

Complex divide(const Complex sc) const;//除法，sc作为除数

double abs() const;//返回绝对值

std::string toString() const;//转为字符串a＋bi，如果虚部为0，只显示a

Complex& operator+=(const Complex& sc);

Complex& operator-=(const Complex& sc);

Complex& operator\*=(const Complex& sc);

Complex& operator/=(const Complex& sc);

double& operator[](const int index);

const double& operator[](const int index) const;

Complex operator+() const;//对数据无实质改变

Complex operator-() const;//实部虚部取反

Complex& operator++();//对实部前置＋＋

Complex operator++(int);//对实部后置＋＋

Complex& operator--();//对实部前置－－

Complex operator--(int);//对实部后置－－

private://private部分为内部实现，可任意增加、修改或删除

double a;//实部

double b;//虚部

};

//全局变量和函数部分，不可增加、修改或删除

std::ostream& operator<<(std::ostream& ostr, const Complex& sc);//输出a+bi，支持文件输出

std::istream& operator>>(std::istream& ostr, Complex& sc);//输入实部虚部到sc，支持文件输入

Complex operator+(const Complex& sc1, const Complex& sc2);//加法

Complex operator-(const Complex& sc1, const Complex& sc2);//减法

Complex operator\*(const Complex& sc1, const Complex& sc2);//乘法

Complex operator/(const Complex& sc1, const Complex& sc2);//除法

#endif /\* defined(\_\_Complex\_\_) \*/

#include <iostream>

#include <sstream>

#include <string>

#include <iomanip>

#include <cmath>

#include <cstdlib>

#include "Complex.h"

using namespace std;

// Constructor

Complex::Complex ()

{

a = 0;

b = 0;

}

Complex::Complex (const double a , const double b)

{

this -> a = a;

this -> b = b;

}

Complex::Complex (const double a)

{

this -> a = a;

b = 0;

}

// Destructor

Complex::~Complex()

{

a = b = 0;

}

// Get the real part and the imaginary part

const double Complex::getRealPart() const

{

return this -> a;

}

const double Complex::getImaginaryPart() const

{

return this -> b;

}

// Four fundamental rules and absolute value of the operator

Complex Complex::add (const Complex sc) const

{

double newRealPart = a + sc.a;

double newImaginaryPart = b + sc.b;

return Complex(newRealPart, newImaginaryPart);

}

Complex Complex::subtract (const Complex sc) const

{

double newRealPart = a - sc.a;

double newImaginaryPart = b - sc.b;

return Complex(newRealPart, newImaginaryPart);

}

Complex Complex::multiply (const Complex sc) const

{

double newRealPart = a \* sc.a - b \* sc.b;

double newImaginaryPart = b \* sc.a + a \* sc.b;

return Complex(newRealPart, newImaginaryPart);

}

Complex Complex::divide (const Complex sc) const

{

double newRealPart = (a \* sc.a + b \* sc.b) / (sc.a \* sc.a + sc.b \* sc.b);

double newImaginaryPart = (b \* sc.a - a \* sc.b) / (sc.a \* sc.a + sc.b \* sc.b);

return Complex(newRealPart, newImaginaryPart);

}

double Complex::abs() const

{

double length = sqrt(a \* a + b \* b);

return length;

}

// Switch to the string

string Complex::toString() const

{

stringstream temp;

temp << a;

string result = temp.str();

if (b != 0)

{

temp.str("");

temp << b;

if (b > 0)

{

result = result + "+" + temp.str() + 'i';

}

else

{

result = result + temp.str() + 'i';

}

}

return result;

}

// Overload logogram operator

Complex& Complex::operator += (const Complex& sc)

{

\*this = add(sc);

return \*this;

}

Complex& Complex::operator -= (const Complex& sc)

{

\*this = subtract(sc);

return \*this;

}

Complex& Complex::operator \*= (const Complex& sc)

{

\*this = multiply(sc);

return \*this;

}

Complex& Complex::operator /= (const Complex& sc)

{

\*this = divide(sc);

return \*this;

}

// Overload [] operator

double& Complex::operator [] (const int index)

{

if (index == 0)

{

return a;

}

else

{

return b;

}

}

const double& Complex::operator [] (const int index) const

{

if (index == 0)

{

return a;

}

else

{

return b;

}

}

// Overload unary operator

Complex Complex::operator + () const

{

return \*this;

}

Complex Complex::operator - () const

{

return Complex(-a , -b);

}

// Overload prepose and postpose operator

Complex& Complex::operator ++ ()

{

a++;

return \*this;

}

Complex Complex::operator ++ (int star)

{

Complex temp (a , b);

a ++;

return temp;

}

Complex& Complex::operator -- ()

{

a--;

return \*this;

}

Complex Complex::operator -- (int star)

{

Complex temp (a , b);

a --;

return temp;

}

// Overload input and output operators

std::ostream& operator << (std::ostream& ostr, const Complex& sc)

{

ostr << sc.toString();

return ostr;

}

std::istream& operator >> (std::istream& ostr, Complex& sc)

{

ostr >> sc[0] >> sc[1];

return ostr;

}

// Overload non-member unary operator

Complex operator + (const Complex& sc1 , const Complex& sc2)

{

return sc1.add(sc2);

}

Complex operator - (const Complex& sc1 , const Complex& sc2)

{

return sc1.subtract(sc2);

}

Complex operator \* (const Complex& sc1 , const Complex& sc2)

{

return sc1.multiply(sc2);

}

Complex operator / (const Complex& sc1 , const Complex& sc2)

{

return sc1.divide(sc2);

}