Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики»

Лабораторная работа №2

«Абстрактный тип данных комплексное число»

Выполнил: студент IV курса
ИВТ, гр. ИП-713
Михеев Н.А.
Проверил: ассистент кафедры
ПМиК
Агалаков А.А.

Новосибирск, 2020 г.

Цель

Сформировать практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов C++. Синтаксис классов: инкапсуляция, простые свойства.

Задание

1. Реализовать абстрактный тип данных «комплексное число», используя класс C++, в соответствии с приведенной ниже спецификацией.

2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Реализация

В ходе выполнения задания был реализован класс «комплексное число» - TComplex, который хранит в себе два приватных поля типа double – re и im – действительная и мнимая части числа. Были реализованы сопутствующие конструкторы, методы класса и перегрузки операторов в соответствии с заданием.

Далее о каждом из объектов класса подробнее:

TComplex() – базовый конструктор класса, формирует начальное число «0+i*0»;
TComplex(double a, double b) – конструктор комплексного числа из двух целых чисел;
TComplex(std::string num) – конструктор комплексного числа из строки;
TComplex copy() – метод копирования одного комплексного числа в другое, вызывается как a.copy(b);
TComplex add(TComplex d) – метод суммирования одного комплексного числа с другим, вызывается как a.add(b);
TComplex multiply(TComplex d) – метод умножения одного комплексного числа на другое;
TComplex square() – метод возведения комплексного числа в квадрат;
TComplex reverse() – метод получения обратного комплексного числа, получается путем деления единицы на число;
TComplex subtract(TComplex d) – метод вычитания одного комплексного числа из другого;
TComplex divide(TComplex d) – метод деления одного комплексного числа на другое;
TComplex operator-() – перегрузка оператора «минус»;
double abs() – метод получения абсолютного значения комплексного числа;

double angleRad() – метод получения угла в радианах по данному комплексному числу;

double angleDegree() – метод получения угла в градусах по данному комплексному числу;

TComplex pow(int n) – метод возведения комплексного числа в данную степень n;

TComplex root(int n, int i) – метод получения i-го кореня целой положительной степени n самого комплексного числа;

bool operator==(TComplex d) – перегрузка булейвого оператора «==», необходимого для сравнения двух комплексных чисел;

bool operator!=(TComplex d) - перегрузка булейвого оператора «!=», необходимого для сравнения двух комплексных чисел;

double getReNum() – метод, возвращающий числовое значение действительной части комплексного числа;

double getImNum() – метод, возвращающий числовое значение мнимой части комплексного числа;

std::string getReStr() – метод, возвращающий строковое значение действительной части комплексного числа;

std::string getImStr() – метод, возвращающий строковое значение мнимой части комплексного числа;

std::string getComplexStr() – метод, возвращающий строковое значение всего комплексного числа;

Рис.1 – Результат проверки работоспособности программы.

Так же были реализованы тесты для всех методов и перегрузок.

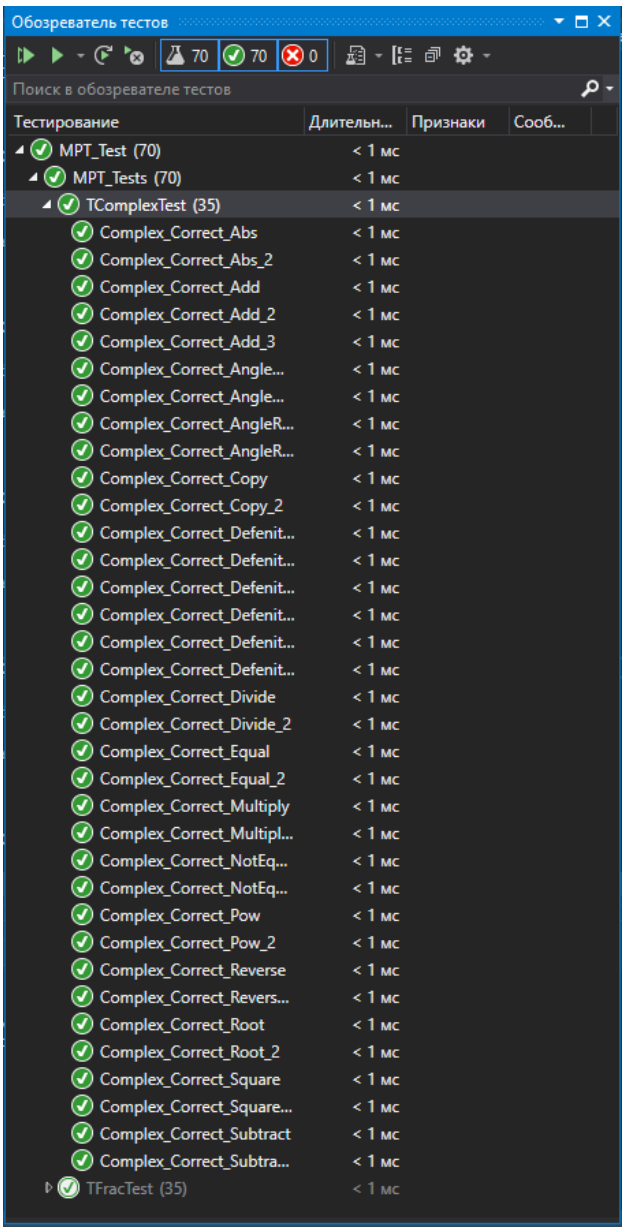

Рис.2 – демонстрация результатов проведенного тестирования

Вывод

В результате работы были сформированы практические навыки реализации абстрактных типов данных, реализован такой тип данных «комплексное число», разработаны неоходимые методы и перегрузки в соответствии с приведенной в задании спецификации. Написаны тесты для каждой операции, используя средства модульного тестирования Visual Studio.

Листинг программы:

TComplex.h:

```
#pragma once
#define _USE_MATH_DEFINES
#include <math.h>
#include <string>
#include <sstream>

class TComplex
{
private:
	double re;
	double im;
public:
	TComplex();
	TComplex(double a, double b);
	TComplex(std::string num);

	TComplex copy();
	TComplex add(TComplex d);
	TComplex multiply(TComplex d);
	TComplex square();
	TComplex reverse();
	TComplex subtract(TComplex d);
	TComplex divide(TComplex d);
	TComplex operator-();
	double abs();
	double angleRad();
	double angleDegree();
	TComplex pow(int n);
	TComplex root(int n, int i);
	bool operator==(TComplex d);
	bool operator!=(TComplex d);
	double getReNum();
	double getImNum();
	std::string getReStr();
	std::string getImStr();
	std::string getComplexStr();
};
```

TComplex.cpp:

```
#include "TComplex.h"

TComplex::TComplex()
{
	re = 0;
	im = 0;
}

TComplex::TComplex(double a, double b) : re(a), im(b) {}

TComplex::TComplex(std::string num)
{
	size_t index = num.find("+i*");
	int sign = 1;
	if (index == std::string::npos)
	{
		index = num.find("-i*");
		sign = -1;
	}
	re = std::stod(num.substr(0, index));
```

```
	im = std::stod(num.substr(index + 3)) * sign;
}

TComplex TComplex::copy()
{
	return { re, im };
}

TComplex TComplex::add(TComplex d)
{
	return { re + d.re, im + d.im };
}

TComplex TComplex::multiply(TComplex d)
{
	return { (re * d.re - im * d.im), (re * d.im + im * d.re) };
}

TComplex TComplex::square()
{
	return { (re * re - im * im), (re * im + re * im) };
}

TComplex TComplex::reverse()
{
	return { re / (re * re + im * im), -(im / (re * re + im * im)) };
}

TComplex TComplex::subtract(TComplex d)
{
	return { re - d.re, im - d.im };
}

TComplex TComplex::divide(TComplex d)
{
	return { ((re * d.re + im * d.im) / (d.re * d.re + d.im * d.im)),
	((d.re * im - re * d.im) / (d.re * d.re + d.im * d.im)) };
}

TComplex TComplex::operator-()
{
	return TComplex(0 - re, 0 - im);
}

double TComplex::abs()
{
	return sqrt(re * re + im * im);
}

double TComplex::angleRad()
{
	if (re == 0 && im == 0)
		return 0;
	if (re == 0 && im > 0)
		return M_PI_2;
	if (re == 0 && im < 0)
		return -M_PI_2;
	if (re > 0)
		return atan(im / re);
	if (re < 0)
		return atan(im / re) + M_PI;

	return -1;
}
```

```
double TComplex::angleDegree()
{
	return (angleRad() * 180) / M_PI;
}

TComplex TComplex::pow(int n)
{
	return { std::pow(abs(), n) * cos(n * angleRad()), std::pow(abs(), n) * sin(n *
angleRad()) };
}

TComplex TComplex::root(int n, int i)
{
	if (i >= n)
		return { 0, 0 };

	return { (std::pow(abs(), 1.0 / (double)n) * (cos((angleRad() + 2 * i * M_PI) /
(double)n))),
		(std::pow(abs(), 1.0 / (double)n) * (sin((angleRad() + 2 * i * M_PI) /
(double)n))) };
}

bool TComplex::operator==(TComplex d)
{
	return (re == d.re && im == d.im);
}

bool TComplex::operator!=(TComplex d)
{
	return !(*this == d);
}

double TComplex::getReNum()
{
	return re;
}

double TComplex::getImNum()
{
	return im;
}

std::string TComplex::getReStr()
{
	return std::to_string(re);
}

std::string TComplex::getImStr()
{
	return std::to_string(im);
}

std::string TComplex::getComplexStr()
{
	std::string symbol = "+";

	if (im < 0)
	{
		symbol = "-";
		im = -im;
	}

	std::ostringstream str;
	str << re << symbol << "i*" << im;
```

```
	return str.str();
}
```

TComplex_Test.cpp:

```
#include "pch.h"
#include "CppUnitTest.h"

#include "../Lab1/TComplex.h"
#include "../Lab1/TComplex.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
	TEST_CLASS(TComplexTest)
	{
	public:

		TEST_METHOD(Complex_Correct_Defenition)
		{
			std::string expect = "2+i*2";
			TComplex test(2, 2);
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Defenition_2)
		{
			std::string expect = "2-i*2";
			TComplex test(2, -2);
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Defenition_3)
		{
			std::string expect = "-2-i*2";
			TComplex test("-2-i*2");
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Defenition_4)
		{
			std::string expect = "3.15-i*2";
			TComplex test("3.15-i*2");
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Defenition_5)
		{
			std::string expect = "235.265+i*742.763";
			TComplex test(235.265, 742.763);
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Defenition_6)
		{
			std::string expect = "235.265-i*742.763";
			TComplex test(235.265, -742.763);
			Assert::AreEqual(expect, test.getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Copy)
		{
			std::string expect = "235.265+i*742.763";
			TComplex test(235.265, 742.763);
			Assert::AreEqual(expect, test.copy().getComplexStr());
```

```
		}

		TEST_METHOD(Complex_Correct_Copy_2)
		{
			std::string expect = "23.26+i*74.76";
			TComplex test("23.26+i*74.76");
			Assert::AreEqual(expect, test.copy().getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Add)
		{
			std::string expect = "5+i*7";
			TComplex test(3, 6);
			Assert::AreEqual(expect, test.add(TComplex(2, 1)).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Add_2)
		{
			std::string expect = "5+i*7";
			TComplex test("3+i*6");
			Assert::AreEqual(expect, test.add(TComplex(2, 1)).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Add_3)
		{
			std::string expect = "5+i*7";
			TComplex test(3, 6);
			Assert::AreEqual(expect,
test.add(TComplex("2+i*1")).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Subtract)
		{
			std::string expect = "1+i*5";
			TComplex test(3, 6);
			Assert::AreEqual(expect, test.subtract(TComplex(2,
1)).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Subtract_2)
		{
			std::string expect = "1+i*5";
			TComplex test(3, 6);
			Assert::AreEqual(expect,
test.subtract(TComplex("2+i*1")).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Multiply)
		{
			std::string expect = "0+i*15";
			TComplex test(3, 6);
			Assert::AreEqual(expect, test.multiply(TComplex(2,
1)).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Multiply_2)
		{
			std::string expect = "0+i*15";
			TComplex test(3, 6);
			Assert::AreEqual(expect,
test.multiply(TComplex("2+i*1")).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Divide)
		{
```

```
			std::string expect = "2.4+i*1.8";
			TComplex test(3, 6);
			Assert::AreEqual(expect, test.divide(TComplex(2,
1)).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Divide_2)
		{
			std::string expect = "2.4+i*1.8";
			TComplex test(3, 6);
			Assert::AreEqual(expect,
test.divide(TComplex("2+i*1")).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Abs)
		{
			double expect = 5;
			TComplex test(3, 4);
			Assert::AreEqual(expect, test.abs());
		}

		TEST_METHOD(Complex_Correct_Abs_2)
		{
			double expect = 5;
			TComplex test(0, -5);
			Assert::AreEqual(expect, test.abs());
		}

		TEST_METHOD(Complex_Correct_AngleRad)
		{
			double expect = 0;
			TComplex test(0, 0);
			Assert::AreEqual(expect, test.angleRad());
		}

		TEST_METHOD(Complex_Correct_AngleRad_2)
		{
			double expect = M_PI_2;
			TComplex test(0, 6);
			Assert::AreEqual(expect, test.angleRad());
		}

		TEST_METHOD(Complex_Correct_AngleDegree)
		{
			double expect = 90;
			TComplex test(0, 6);
			Assert::AreEqual(expect, test.angleDegree());
		}

		TEST_METHOD(Complex_Correct_AngleDegree_2)
		{
			double expect = 0;
			TComplex test(0, 0);
			Assert::AreEqual(expect, test.angleDegree());
		}

		TEST_METHOD(Complex_Correct_Pow)
		{
			std::string expect = "-16+i*30";
			TComplex test(3, 5);
			Assert::AreEqual(expect, test.pow(2).getComplexStr());
		}

		TEST_METHOD(Complex_Correct_Pow_2)
		{
```

```
	std::string expect = "-0.0138408-i*0.0259516";
	TComplex test(3, 5);
	Assert::AreEqual(expect, test.pow(-2).getComplexStr());
}

TEST_METHOD(Complex_Correct_Root)
{
	TComplex test(3, 7);
	Assert::IsTrue(test.root(3, 1).getReNum() - -1.556006 < 0.001);
	Assert::IsTrue(test.root(3, 1).getImNum() - 1.2040441 < 0.001);
}

TEST_METHOD(Complex_Correct_Root_2)
{
	std::string expect = "0+i*0";
	TComplex test(3, 5);
	Assert::AreEqual(expect, test.root(2, 10).getComplexStr());
}

TEST_METHOD(Complex_Correct_Reverse)
{
	std::string expect = "0.1-i*0.1";
	TComplex test(5, 5);
	Assert::AreEqual(expect, test.reverse().getComplexStr());
}

TEST_METHOD(Complex_Correct_Reverse_2)
{
	std::string expect = "0.25+i*0.25";
	TComplex test(2, -2);
	Assert::AreEqual(expect, test.reverse().getComplexStr());
}

TEST_METHOD(Complex_Correct_Square)
{
	std::string expect = "-5+i*12";
	TComplex test(2, 3);
	Assert::AreEqual(expect, test.square().getComplexStr());
}

TEST_METHOD(Complex_Correct_Square_2)
{
	std::string expect = "9+i*40";
	TComplex test(5, 4);
	Assert::AreEqual(expect, test.square().getComplexStr());
}

TEST_METHOD(Complex_Correct_Equal)
{
	TComplex test(5, 4);
	TComplex test2(5, 4);
	Assert::IsTrue(test == test2);
}

TEST_METHOD(Complex_Correct_Equal_2)
{
	TComplex test(5, 4);
	TComplex test2(4, 5);
	Assert::IsFalse(test == test2);
}

TEST_METHOD(Complex_Correct_NotEqual)
{
	TComplex test(5, 4);
	TComplex test2(4, 5);
```

```
			Assert::IsTrue(test != test2);
		}

		TEST_METHOD(Complex_Correct_NotEqual_2)
		{
			TComplex test(5, 4);
			TComplex test2(5, 4);
			Assert::IsFalse(test != test2);
		}
	};
}
```