

Министерство цифрового развития, связи и массовых коммуникаций  
Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Сибирский государственный университет  
телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа  
«Абстрактный тип комплексное число»

Выполнил:  
Студент группы ИП-911  
Мироненко К.А.  
Работу проверил:  
доцент кафедры ПМиК  
Зайцев М.Г.

Новосибирск 2022 г.

## СОДЕРЖАНИЕ

<b>1. Задание .....</b>	<b>3</b>
<b>2. Исходный код программы .....</b>	<b>13</b>
2.1. Код программы .....	13
2.2. Код тестов.....	18
<b>3. Результаты .....</b>	<b>23</b>
3.1. Пример работы программы .....	23
3.2. Результаты тестирования.....	23
<b>4. Вывод.....</b>	<b>24</b>

## 1. 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 Visual Studio.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

### Спецификация типа данных «простые дроби».

#### ADT TComplex

##### Данные

Комплексное число TComplex - это неизменяемая пара вещественных чисел, представляющие действительную и мнимую части комплексного числа ( $a + i*b$ ).

##### Операции

4. Операции могут вызываться только объектом комплексное число (тип TComplex), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется `this` «само число».

<b>КонструкторЧисло</b>	
Вход:	Пара вещественных чисел a и b.
Предусловия:	Нет.
Процесс:	Устанавливает значения a, b в поля экземпляра класса комплексное число (тип TComplex) this.a -
	действительной частью и this.b мнимая часть.  Например: <i>КонструкторЧисло</i> (6,3)= $6 + i*3$ <i>КонструкторЧисло</i> (3,0)= $3 + i*0$ <i>КонструкторЧисло</i> (0,0)= $0 + i*0$

Постусловия:	Поля объекта <code>this</code> инициализированы входными данными.
Выход:	Нет.
<b>КонструкторСтрока</b>	
Вход:	Строка <code>f</code> , представляющая комплексное число.
Предусловия:	Изображение комплексного числа во входной строке <code>f</code> должно быть представлено в заданном формате.
Процесс:	<p>Выделяет из строки <code>f = 'a + i*b'</code>, действительную часть (<code>a</code>) и комплексную часть (<code>b</code>) и преобразует их в число. Устанавливает значения <code>a</code>, <code>b</code> в поля экземпляра класса комплексное число (тип <code>TComplex</code>) <code>this.a</code> - действительной частью и <code>this.b</code> мнимая часть.</p> <p>Например:</p> <p><i>КонструкторСтрока('6+i*3') = 6+i*3</i></p> <p><i>КонструкторСтрока('0+i*3') = 0+i*3</i></p>
Постусловия:	Поля объекта <code>this</code> инициализированы входными данными.
Выход:	Нет.
<b>Копировать:</b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Создаёт и возвращает собственную копию - комплексное число (тип <code>TComplex</code> ) с действительной и мнимой частями такими же, как у самого числа <code>this</code> .

Выход:	Комплексное число (тип TComplex).  Например: $c = 6+i3$ , Копировать( $c$ ) = $6+i3$
Постусловия:	Нет.
<b>Сложить</b>	
Вход:	Комплексное число d (тип TComplex).
Предусловия:	Нет.
Процесс	Создаёт и возвращает комплексное число, полученное сложением самого числа $this = a1+i*b1$ с числом $d = a2+i*b2$ : $((a1+i*b1)+(a2+i*b2)=(a1+a2)+i*(b1+b2))$ . Например: $q = (2 +i*1)$ , $d = (2 +i*1)$ , $q.Сложить(d) = (4 +i*2)$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Умножить</b>	
Вход:	Комплексное число d (тип TComplex).
Предусловия:	Нет.
Процесс	Создаёт и возвращает комплексное число, полученное умножением самого числа $this = a1+i*b1$ на число $d = a2+i*b2$ : $((a1+i*b1)*(a2+i*b2)=(a1*a2 - b1*b2)+i*(a1*b2+ a2*b1))$ .
Выход:	Комплексное число (тип TComplex).

Постусловия:	Нет.
<b>Квадрат</b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Создаёт и возвращает комплексное число (тип TComplex), полученное умножением числа this насамого себя: $((a1+i*b1)*(a1+i*b1)=(a1*a1 - b1*b1)+i*(a1*b1+ a1*b1))$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Обратное</b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Создаёт и возвращает комплексное число (тип TComplex), полученное делением единицы на само число $1/((a1+i*b1) = a1/(a1**2 + b1**2) - i*b1/( a1**2 + b1**2 ))$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Вычесть</b>	
Вход:	Комплексное число d (тип TComplex)..
Предусловия:	Нет.

Процесс	Создаёт и возвращает комплексное число (тип TComplex), полученное вычитанием $d = a_2 + i b_2$ из самого числа $this = (a_1 + i b_1)$ : $(a_1 + i b_1) - (a_2 + i b_2) = (a_1 - a_2) + i(b_1 - b_2)$ .
	Например: $q = (2 + i * 1)$ , $d = (2 + i * 1)$ $q.$ Вычесть( $d$ ) = $(0 + i 0)$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Делить</b>	
Вход:	Комплексное число $d$ .
Предусловия:	Нет.
Процесс	Создаёт и возвращает комплексное число (тип TComplex), полученное делением самого числа $this$ на число ( $d$ ) $((a_1 + i b_1) / (a_2 + i b_2) = (a_1 * a_2 + b_1 * b_2) / (a_2^2 + b_2^2) + i(a_2 * b_1 - a_1 * b_2) / (a_2^2 + b_2^2))$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Минус</b>	
Вход:	Нет.
Предусловия:	Нет.

Процесс	Создаёт и возвращает комплексное число (тип TComplex), являющееся разностью комплексных чисел z и i самого числа this, где z – комплексное число (0+i0).
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Модуль</b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Вычисляет и возвращает модуль самого комплексного числа this. Например: $q = (2 + i*1), q.$ Модуль = $\sqrt{(2*2+1*1)}$ . $q = (i*17), q.$ Модуль = $\sqrt{(0*0+17*17)}$ .
Выход:	Вещественное число.
Постусловия:	Нет.
<b>УголРад</b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Возвращает аргумент $\varphi$ самого комплексного числа this (в радианах). $\varphi = (\arctg(b/a), a > 0; \pi/2, a = 0, b > 0; \arctg(b/a) + \pi, a < 0; -\pi/2, a = 0, b < 0)$ . <p>Например:</p> $q = (1 + i*1), q.$ УголРад = 0,79.
Выход:	Вещественное число.
Постусловия:	Нет.



<b><i>УголГрад</i></b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Возвращает аргумент самого комплексного числа this(в градусах). Например: $q = (1 + i * 1)$ , q. Град = 45.
Выход:	Вещественное число.
Постусловия:	Нет.
<b><i>Степень</i></b>	
Вход:	Целое n.
Предусловия:	Нет.
Процесс	Возвращает целую положительную степень n самого комплексного числа this. $this^n = r^n(\cos(n * \phi) + i * \sin(n * \phi))$ .
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b><i>Корень</i></b>	
Вход:	Целое n, целое i.
Предусловия:	Нет.

Процесс	Возвращает i-ый корень целой положительной степени n самого комплексного числа this. $\sqrt[n]{this} = \sqrt[n]{r} * (\cos((\phi + 2*k*\pi)/n) + i * \sin((\phi + 2*k*\pi)/n))$ При этом коэффициенту k придается последовательно n значений: $k = 0, 1, 2, \dots, n - 1$ и получают n значений корня, т.е. ровно столько, каков показатель корня.
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.
<b>Равно</b>	
Вход:	Комплексное число d.
Предусловия:	Нет.
Процесс	Сравнивает само комплексное число this с числом d. Возвращает значение True, если они - тождественные комплексные числа, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
<b>НеРавно</b>	
Вход:	Комплексное число d.
Предусловия:	Нет.
Процесс	Сравнивает само комплексное число this с числом d. Возвращает значение True, если само число $\neq d$ , - значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.

<b><i>ВзятьReЧисло</i></b>	
Вход:	Нет
Предусловия:	Нет.
Процесс	Возвращает значение действительной части самого комплексного числа this в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
<b><i>ВзятьImЧисло</i></b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Возвращает значение мнимой части самого комплексного числа this в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
<b><i>ВзятьReСтрока</i></b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Возвращает значение вещественной части самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
<b><i>ВзятьImСтрока</i></b>	
Вход:	Нет.
Предусловия:	Нет.

Процесс	Возвращает значение мнимой части самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
<b><i>ВзятьКомплексноеСтрока</i></b>	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Возвращает значение самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.

***end TComplex***

## 2. Исходный код программы

### 2.1. Код программы

#### Program.cs

```
using System;

namespace fraction
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

#### TComplex.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace complex
{
    // Обработка исключения
    public class MyException : Exception
    {
        public MyException(string str) : base(str) { }
    }
    public abstract class TComplex
    {
        private double real;
        private double imaginary;

        public double Real {
            get
            {
                return real;
            }
            set
            {
                real = value;
            }
        }
        public double Imaginary
        {
            get
            {
                return imaginary;
            }
            set
            {
                imaginary = value;
            }
        }
        public TComplex(double a, double b)
        {
            real = a;
        }
    }
}
```

```

        imaginary = b;
    }
    public TComplex(string str)
    {
        try
        {
            str = str.Replace(" ", "");
            var indexPlus = str.IndexOf("+");
            var astr = str.Substring(0, indexPlus);
            var bstr = str.Substring(indexPlus + 3);
            real = Double.Parse(astr);
            imaginary = Double.Parse(bstr);
        }
        catch
        {
            throw new MyException("Не получилось обработать строку");
        }
    }
    public TComplex Copy()
    {
        return (TComplex)this.MemberwiseClone();
    }

    public TComplex Add(TComplex b)
    {
        TComplex res = this.Copy();
        res.real += b.real;
        res.imaginary += b.imaginary;
        return res;
    }

    public TComplex Multiplication(TComplex b)
    {
        TComplex res = this.Copy();
        res.real = this.real * b.real - this.imaginary * b.imaginary;
        res.imaginary = this.real * b.imaginary + this.imaginary * b.real;
        return res;
    }

    public TComplex Square()
    {
        TComplex res = this.Copy();
        res.real = this.real * this.real - this.imaginary * this.imaginary;
        res.imaginary = this.real * this.imaginary + this.real * this.imaginary;
        return res;
    }

    public TComplex Reverse()
    {
        TComplex res = this.Copy();
        res.real = this.real / (this.real * this.real + this.imaginary *
this.imaginary);
        res.imaginary = -this.imaginary / (this.real * this.real + this.imaginary *
this.imaginary);
        return res;
    }

    public TComplex Subtract(TComplex b)
    {
        TComplex res = this.Copy();
        res.real -= b.real;
        res.imaginary -= b.imaginary;
        return res;
    }

```

```

public TComplex Divide(TComplex b)
{
    TComplex res = this.Copy();
    res.real = (this.real * b.real + this.imaginary * b.imaginary) / (b.real *
b.real + b.imaginary * b.imaginary);
    res.imaginary = (b.real * this.imaginary - this.real * b.imaginary) / (b.real
* b.real + b.imaginary * b.imaginary);
    return res;
}

public TComplex Minus()
{
    TComplex res = this.Copy();
    res.real = 0 - res.real;
    res.imaginary = 0 - res.imaginary;
    return res;
}

public double Abs()
{
    return Math.Sqrt(this.real * this.real + this.imaginary * this.imaginary);
}

public double Rad()
{
    if (this.real > 0)
        return Math.Atan(this.imaginary / this.real);

    if (this.real == 0 && this.imaginary > 0)
        return (Math.PI / 2);

    if (this.real < 0)
        return (Math.Atan(this.imaginary / this.real) + Math.PI);

    if (this.real == 0 && this.imaginary < 0)
        return (-Math.PI / 2);

    return 0;
}

public double Degree()
{
    return Rad() * 180 / Math.PI;
}

public TComplex Pow(int n)
{
    TComplex res = this.Copy();
    res.real = Math.Pow(Abs(), n) * Math.Cos(n * Rad());
    res.imaginary = Math.Pow(Abs(), n) * Math.Sin(n * Rad());
    return res;
}

public TComplex Sqrt(int powN, int rootI)
{
    if (powN == 0)
    {
        TComplex res0 = this.Copy();
        res0.real = 1;
        res0.imaginary = 0;
        return res0;
    }
}

```

```

        if (rootI == 0)
            new MyException("Деление на 0.");

        TComplex new1 = Pow(powN);

        TComplex res = this.Copy();
        res.real = Math.Pow(new1.Abs(), 1 / (double)rootI) * Math.Cos((new1.Rad() + 2
* Math.PI * rootI) / rootI);
        res.imaginary = Math.Pow(new1.Abs(), 1 / (double)rootI) *
Math.Sin((new1.Rad() + 2 * Math.PI * rootI) / rootI);

        return res;
    }

    public bool Equal(TComplex anClass)
    {
        return (this.real == anClass.real && this.imaginary == anClass.imaginary);
    }

    public bool NotEqual(TComplex anClass)
    {
        return (this.real != anClass.real || this.imaginary != anClass.imaginary);
    }

    public double GetRealNumber()
    {
        return this.real;
    }

    public double GetImaginaryNumber()
    {
        return this.imaginary;
    }

    public string GetRealString()
    {
        return this.real.ToString();
    }

    public string GetImaginaryString()
    {
        return this.imaginary.ToString();
    }

    public string GetString()
    {
        return this.real.ToString("###,###") + ' ' + (this.imaginary >= 0 ? '+' : '-')
+ " i * " + this.imaginary.ToString("###,###");
    }
}
}

```

## Complex.cs

```

using System;
using System.Collections.Generic;
using System.Text;

namespace complex
{
    public class Complex : TComplex

```



```
{
    public Complex(double a, double b) : base(a, b)
    {

    }

    public Complex(string str) : base(str)
    {

    }
}
}
```

## 2.2. Код тестов

### UnitTestComplex.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using complex;

namespace UnitTests
{
    [TestClass]
    public class UnitTestComplex
    {
        [TestMethod]
        public void TestTComplexDouble()
        {
            var testClass = new Complex(9.99, 0.44);

            Assert.AreEqual(testClass.Real, 9.99);
            Assert.AreEqual(testClass.Imaginary, 0.44);
        }

        [TestMethod]
        public void TestTComplexString()
        {
            string output = "9,99 + i * 0,44";
            var testClass = new Complex(output);

            Assert.AreEqual(testClass.Real, 9.99);
            Assert.AreEqual(testClass.Imaginary, 0.44);
        }

        [TestMethod]
        public void TestTComplexStringEx()
        {
            void Action()
            {
                string output = "9,5i +i3,1";
                new Complex(output);
            }
            Action action = new Action(Action);

            Assert.ThrowsException<MyException>(action);
        }

        [TestMethod]
        public void TestCopy()
        {
            var test1 = new Complex(9.89, 0.44);
            var test2 = test1.Copy();

            Assert.AreEqual(test1.Real, test2.Real);
            Assert.AreEqual(test1.Imaginary, test2.Imaginary);
        }

        [TestMethod]
        public void TestAdd()
        {
            var test1 = new Complex(3, 4);
            var test2 = new Complex(4, -1);
            var resulr = test1.Add(test2);

            Assert.AreEqual(resulr.Real, 7);
            Assert.AreEqual(resulr.Imaginary, 3);
        }
    }
}
```

```

[TestMethod]
public void TestMultiply()
{
    var test1 = new Complex(3, 4);
    var test2 = new Complex(4, -1);
    var resulr = test1.Multiplication(test2);

    Assert.AreEqual(resulr.Real, 16);
    Assert.AreEqual(resulr.Imaginary, 13);
}

[TestMethod]
public void TestSubtract()
{
    var test1 = new Complex(3, 4);
    var test2 = new Complex(4, -1);
    var resulr = test1.Subtract(test2);

    Assert.AreEqual(resulr.Real, -1);
    Assert.AreEqual(resulr.Imaginary, 5);
}

[TestMethod]
public void TestDivide()
{
    var test1 = new Complex(3, 4);
    var test2 = new Complex(4, -1);
    var resulr = test1.Divide(test2);

    Assert.AreEqual(resulr.Real, 0.470588, 5);
    Assert.AreEqual(resulr.Imaginary, 1.117647, 5);
}

[TestMethod]
public void TestSquare()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Square();

    Assert.AreEqual(resulr.Real, -7);
    Assert.AreEqual(resulr.Imaginary, 24);
}

[TestMethod]
public void TestReverse()
{
    var test1 = new Complex(0, -3);
    var resulr = test1.Reverse();

    Assert.AreEqual(resulr.Real, 0);
    Assert.AreEqual(resulr.Imaginary, 0.333333, 5);
}

[TestMethod]
public void TestMinus()
{
    var test1 = new Complex(0, 4);
    var resulr = test1.Minus();

    Assert.AreEqual(resulr.Real, 0);
    Assert.AreEqual(resulr.Imaginary, -4);
}

```

```

[TestMethod]
public void TestAbs()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Abs();

    Assert.AreEqual(resulr, 5);
}

[TestMethod]
public void TestRad()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Rad();

    Assert.AreEqual(0.927295, resulr, 5);
}

[TestMethod]
public void TestDegree()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Degree();

    Assert.AreEqual(53.1301, resulr, 4);
}

[TestMethod]
public void TestPow()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Pow(5);

    Assert.AreEqual(resulr.Real, -237, 4);
    Assert.AreEqual(resulr.Imaginary, -3116, 4);
}

[TestMethod]
public void TestRoot()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Sqrt(3, 4);

    Assert.AreEqual(resulr.Real, 2.567133, 5);
    Assert.AreEqual(resulr.Imaginary, 2.142468, 5);
}

[TestMethod]
public void TestRavnFalse()
{
    var test1 = new Complex(3, 4);
    var test2 = new Complex(4, -1);
    var resulr = test1.Equal(test2);

    Assert.IsFalse(resulr);
}

[TestMethod]
public void TestRavnTrue()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.Equal(test1);

    Assert.IsTrue(resulr);
}

```

```

[TestMethod]
public void TestNeRavnFalse()
{
    var test1 = new Complex(3, 4);
    var test2 = new Complex(4, -1);
    var resulr = test1.NotEqual(test2);

    Assert.IsTrue(resulr);
}
[TestMethod]
public void TestNeRavnTrue()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.NotEqual(test1);

    Assert.IsFalse(resulr);
}

[TestMethod]
public void TestGetRealNumber()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.GetRealNumber();

    Assert.AreEqual(resulr, 3);
}

[TestMethod]
public void TestGetImaginaryNumber()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.GetImaginaryNumber();

    Assert.AreEqual(resulr, 4);
}

[TestMethod]
public void TestGetRealString()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.GetRealString();

    Assert.AreEqual(resulr, "3");
}

[TestMethod]
public void TestGetImaginaryString()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.GetImaginaryString();

    Assert.AreEqual(resulr, "4");
}

[TestMethod]
public void TestGetString()
{
    var test1 = new Complex(3, 4);
    var resulr = test1.GetString();

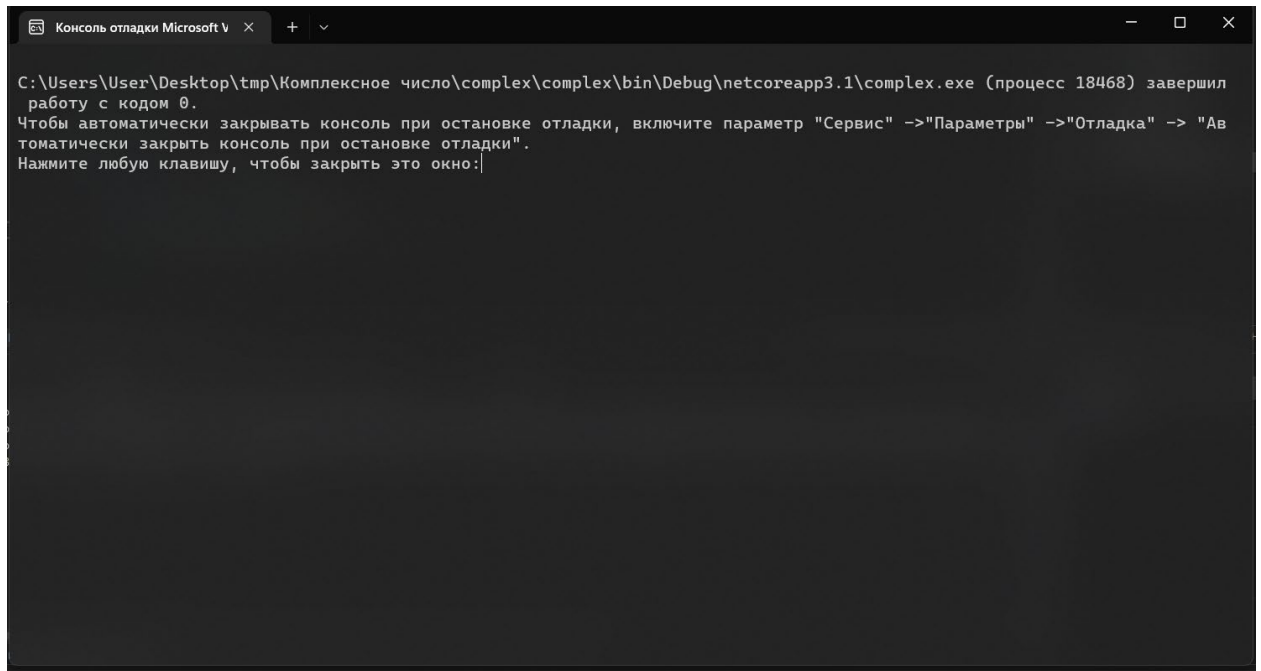
    Assert.AreEqual(resulr, "3 + i * 4");
}
}

```

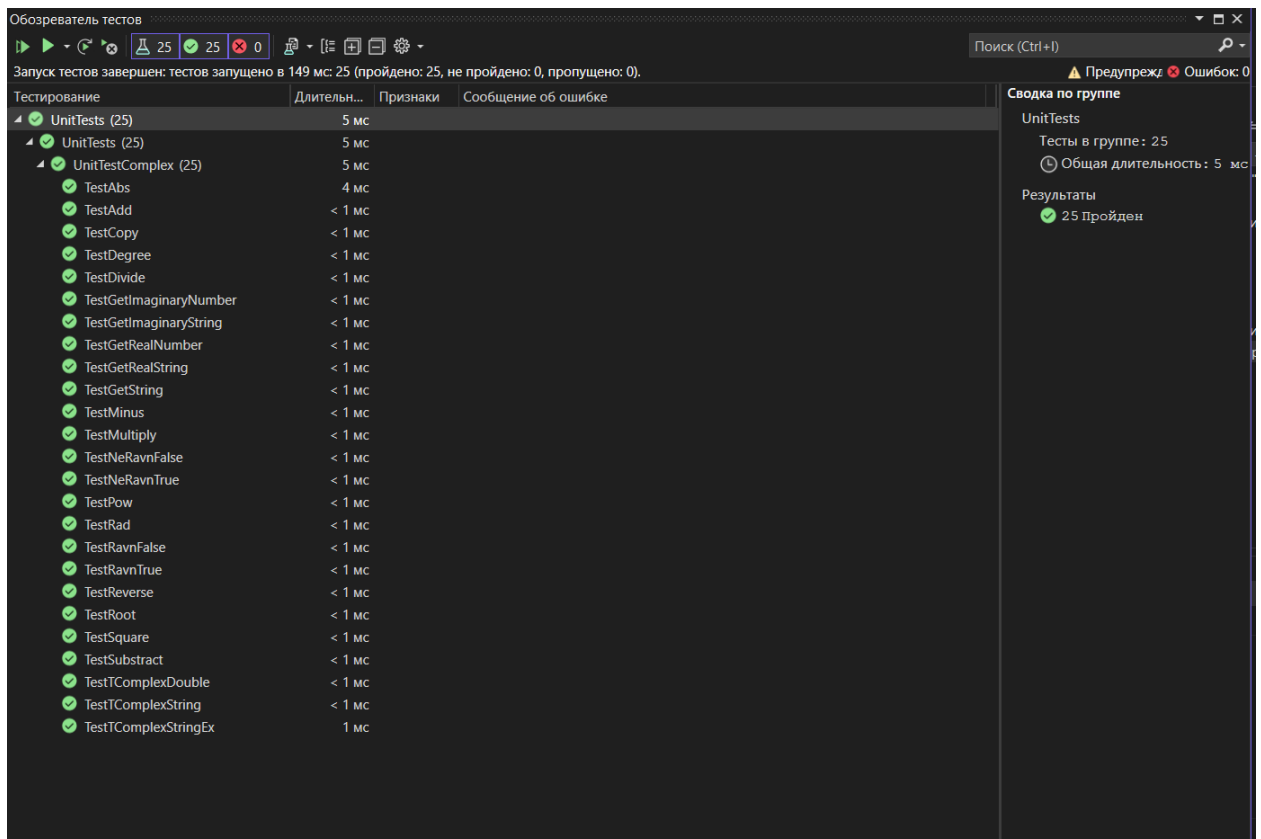
}

### 3. Результаты

#### 3.1.Пример работы программы



#### 3.2.Результаты тестирования



#### **4. Вывод**

По итогам данной лабораторной работе были сформированы практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов C# и их модульного тестирования.