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 данных простая дробь»

Выполнил:

Студент группы ИП-911

Мироненко К.А.

Работу проверил:

доцент кафедры ПМиК

Зайцев М.Г.

**СОДЕРЖАНИЕ**

[**1.** **Задание** 3](#_Toc126767043)

[**2.** **Исходный код программы** 11](#_Toc126767044)

[2.1. Код программы 11](#_Toc126767045)

[2.2. Код тестов 18](#_Toc126767046)

[**3.** **Результаты** 22](#_Toc126767047)

[3.1. Пример работы программы 22](#_Toc126767048)

[3.2. Результаты тестирования 22](#_Toc126767049)

[**4.** **Вывод** 23](#_Toc126767050)

1. **Задание**
2. Реализовать абстрактный тип данных «простая дробь», используя класс С++ в соответствии с приведенной ниже спецификацией.
3. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования Visual Studio. Тестирование осуществляйте по критерию команд С0 .
4. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

**Спецификация типа данных «простые дроби».**

**ADT** TFrac

**Данные**

Простая дробь (тип TFrac) - это пара целых чисел: числитель и знаменатель (a/b).

Простые дроби не изменяемые.

**Операции**

Операции могут вызываться только объектом простая дробь (тип TFrac), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется «сама дробь» this.

|  |  |
| --- | --- |
| ***Конструктор*** |  |
| Вход: | Пара целых чисел a и b. |
| Предусловия: | b не равно 0. В противном случае возбуждается  исключение. |
| Процесс: | Инициализирует поля простой дроби this: числитель значением a, знаменатель - b. В случае необходимости дробь предварительно сокращается.  Например:  *Конструктор*(6,3) = (2/1)  *Конструктор*(0,3) = (0/3). |
| Выход: | Нет. |
| Постусловия: | Поля объекта проинициализированы начальными  значениями. |
|  | |
| ***Конструктор*** |  |
| Вход: | Строковое представление простой дроби . Например:  “7/9”. |
| Предусловия: | b не равно 0. В противном случае возбуждается  исключение. |
| Процесс: | Инициализирует поля простой дроби this строкой f  =”a/b”. Числитель значением a, знаменатель - b. В случае необходимости дробь предварительно сокращается.  Например:  *Конструктор*(‘6/3’) = 2/1  *Конструктор* (‘0/3’) = 0/3 |
| Выход: | Нет. |
| Постусловия: | Поля объекта проинициализированы начальными  значениями. |
|  | |
| **Копировать:** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт копию самой дроби this с числителем, и знаменателем такими же, как у самой дроби. |
| Выход: | Простая дробь (тип TFrac).  Например:  c = 2/1, Копировать(c) = 2/1 |
| Постусловия: | Нет. |
|  | |
| ***Сложить*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает простую дробь (тип TFrac), полученную сложением самой дроби this = a1/b1 с d  =a2/b2: ((a1/b1)+(a2/b2)=(a1\*b2 + a2\*b1)/( b1\* b2)).  Например: q = 1/2, d = -3/4  q.Сложить(d) = -1/4. |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет. |
|  | |
| ***Умножить*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Нет. |
| Процесс: | Создаёт простую дробь (тип TFrac), полученную  умножением самой дроби this = a1/b1 на d = a2/b2 ((a1/b1)\*(a2/b2)=(a1\* a2)/( b1\* b2)). |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет. |
|  | |
| ***Вычесть*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает простую дробь (тип TFrac), полученную вычитанием d = a2/b2 из самой дроби this  = a1/b1: ((a1/b1)-(a2/b2)=(a1\* b2-a2\*b1)/(b1\*b2)).  Например: q = (1/2), d = (1/2)  q.Вычесть(d) = (0/1). |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет |
|  | |
| ***Делить*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Числитель числа d не равно 0. |
| Процесс: | Создаёт и возвращает простую дробь (тип TFrac), полученное делением самой дроби this = a1/b1 на дробь  d = a2/b2: ((a1/b1)/(a2/b2)=(a1\* b2)/( a2\*b1)). |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет. |
|  | |
| ***Квадрат*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает простую дробь (тип TFrac), полученную умножением самой дроби this на себя:  ((a/b)\*(a/b)=(a\* a)/( b\* b)). |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет. |
|  | |
| ***Обратное*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает простую дробь (тип TFrac), полученное делением единицы на саму дробь this:  1/((a/b) = b/a. |
| Выход: | Простая дробь (тип TFrac) |
| Постусловия: | Нет. |
|  | |
| ***Минус*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт простую дробь, являющуюся разностью  простых дробей z и this, где z - простая дробь (0/1. |
| Выход: | Простая дробь (тип TFrac). |
| Постусловия: | Нет. |
|  | |
| ***Равно*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Нет |
| Процесс: | Сравнивает саму простую дробь this и d. Возвращает  значение True, если this и d - тождественные простые дроби, и значение False - в противном случае. |
| Выход: | Булевское значение. |
| Постусловия: | Нет. |
|  | |
| ***Больше*** |  |
| Вход: | Простая дробь d (тип TFrac). |
| Предусловия: | Нет. |
| Процесс: | Сравнивает саму простую дробь this и d. Возвращает значение True, если this > d, - значение False - в  противном случае. |
| Выход: | Булевское значение. |
| Постусловия: | Нет. |
|  | |
| ***ВзятьЧислительЧисло*** |  |
| Вход: |  |
| Предусловия: | Нет. |
| Процесс: | Возвращает значение числителя дроби this в числовом  формате. |
| Выход: | Вещественное значение. |
| Постусловия: | Нет. |
|  | |
| ***ВзятьЗнаменательЧисло*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Возвращает значение знаменателя дроби this в  числовом формате. |
| Выход: | Вещественное значение. |
| Постусловия: | Нет. |
|  | |
| ***ВзятьЧислительСтрока*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Возвращает значение числителя дроби this в строковом  формате. |
| Выход: | Строка. |
| Постусловия: | Нет. |
|  | |
| ***ВзятьЗнаменательСтрока*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Возвращает значение знаменателя дроби this в  строковом формате. |
| Выход: | Строка. |
| Постусловия: | Нет. |
|  | |
| ***ВзятьДробьСтрока*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Возвращает значение простой дроби this, в строковом  формате. |
| Выход: | Строка. |
| Постусловия: | Нет. |
|  |  |

**end** TFrac

1. **Исходный код программы**
   1. **Код программы**

**Program.cs**

using System;

namespace fraction

{

class Program

{

static void Main(string[] args)

{

TFrac frac = new Frac(100, 5);

Console.WriteLine($"{frac.Numerator} {frac.Denominator}");

}

}

}

**TFrac.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace fraction

{

// Обработка исключения

public class MyException : Exception

{

public MyException(string str) : base(str) { }

}

public abstract class TFrac

{

private int numerator;

private int denominator;

/// Числитель

public int Numerator

{

get

{

return numerator;

}

set

{

numerator = value;

}

}

/// Знаменатель

public int Denominator

{

get

{

return denominator;

}

set

{

denominator = value;

}

}

public TFrac()

{

Numerator = 0;

Denominator = 1;

}

public TFrac(int a, int b)

{

if (b == 0)

{

throw new MyException("Деление на ноль невозможно!");

}

Numerator = a;

Denominator = b;

Norm(this);

}

public TFrac(string str)

{

var index = str.IndexOf("/");

if (index < 0)

{

throw new MyException("Строка пуста!");

}

var num = str.Substring(0, index);

var den = str.Substring(index + 1);

var numInt = Convert.ToInt32(num);

var denInt = Convert.ToInt32(den);

if (denInt == 0)

{

throw new MyException("Деление на ноль невозможно!");

}

Numerator = numInt;

Denominator = denInt;

Norm(this);

}

public TFrac Copy()

{

return (TFrac)this.MemberwiseClone();

}

/// Сумма

public TFrac Add(TFrac b)

{

TFrac res = b.Copy();

if (this.Denominator == b.Denominator)

{

res.denominator = this.Denominator;

res.numerator = this.Numerator + b.Numerator;

}

else

{

int nok = NOK(Convert.ToInt32(this.Denominator), Convert.ToInt32(b.Denominator));

res.denominator = nok;

res.numerator = this.Numerator \* (nok / this.Denominator) + b.Numerator \* (nok / b.Denominator);

}

return Norm(res);

}

/// Разность

public TFrac Difference(TFrac B)

{

//if (A.Numerator == 0) return Multiplication(Norm(B), new TFrac(-1, 1));

if (B.Numerator == 0) return Norm(this);

TFrac res = this.Copy();

TFrac a = Norm(this), b = Norm(B);

if (a.Denominator == b.Denominator)

{

res.Denominator = a.Denominator;

res.Numerator = a.Numerator - b.Numerator;

}

else

{

int nok = NOK(Convert.ToInt32(a.Denominator), Convert.ToInt32(b.Denominator));

res.Denominator = nok;

res.Numerator = a.Numerator \* (nok / a.Denominator) - b.Numerator \* (nok / b.Denominator);

}

return Norm(res);

}

/// Произведение

public TFrac Multiplication(TFrac b)

{

TFrac res = this.Copy();

res.Denominator = this.Denominator \* b.Denominator;

res.Numerator = this.Numerator \* b.Numerator;

return res;

}

/// Деление

public TFrac Division(TFrac b)

{

TFrac res = this.Copy();

res.Denominator = this.Denominator \* b.Numerator;

res.Numerator = this.Numerator \* b.Denominator;

return Norm(res);

}

/// Квадрат

public TFrac Square()

{

return this.Multiplication(this);

}

/// Обратное

public TFrac Reverse()

{

TFrac res = this.Copy();

res.Denominator = this.Numerator;

res.Numerator = this.Denominator;

return res;

}

/// Минус

public TFrac Minus()

{

TFrac res = this.Copy();

res.Denominator = this.Denominator;

res.Numerator = 0 - this.Numerator;

return res;

}

/// Равно

public bool Equal(TFrac b)

{

/\*

TFrac res = this.Difference(b);

if (res.Numerator == 0)

{

return true;

}

return false;

\*/

if ((b.Numerator == this.Numerator) && (this.Denominator == b.Denominator))

{

return true;

}

else return false;

}

/// Больше

public bool More(TFrac d)

{

TFrac otv = this.Difference(d);

if ((otv.Numerator > 0 && otv.Denominator > 0)

|| (otv.Numerator < 0 && otv.Denominator < 0))

{

return true;

}

return false;

}

/// ВзятьЧислительЧисло

public int GetNumeratorNumber()

{

return numerator;

}

/// ВзятьЗнаменательЧисло

public int GetDenominatorNumber()

{

return denominator;

}

/// ВзятьЧислительСтрока

public string GetNumeratorString()

{

return numerator.ToString();

}

/// ВзятьЗнаменательСтрока

public string GetDenominatorString()

{

return denominator.ToString();

}

/// ВзятьДробьСтрока

public string GetString()

{

return numerator + "/" + denominator;

}

private int NOK(int a, int b) {

return (a \* b) / Gcd(a, b);

}

private int Gcd(int a, int b) {

return a != 0 ? Gcd(b % a, a) : b;

}

public int NOD(List<int> list)

{

if (list.Count == 0) return 0;

int i;

int gcd = list[0];

for (i = 0; i < list.Count - 1; i++)

gcd = NOD(gcd, list[i + 1]);

return gcd;

}

static int NOD(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

private TFrac Norm(TFrac SimpleFractions)

{

TFrac fractions = SimpleFractions;

if (fractions.Numerator == 0) { fractions.Denominator = 1; return fractions; }

fractions = Reduction(fractions);

if (NOD(new List<int> { fractions.Numerator, fractions.Denominator }) != 0)

{

int nod = NOD(new List<int> { fractions.Numerator, fractions.Denominator });

fractions.Numerator /= nod;

fractions.Denominator /= nod;

}

if (fractions.Denominator < 0)

{

fractions.Numerator \*= -1;

fractions.Denominator \*= -1;

}

return fractions;

}

public TFrac Reduction(TFrac SimpleFractions)

{

TFrac a = SimpleFractions;

if ((SimpleFractions.Numerator >= 0 && SimpleFractions.Denominator < 0) || (SimpleFractions.Numerator < 0 && SimpleFractions.Denominator < 0))

{

SimpleFractions.Numerator \*= -1;

SimpleFractions.Denominator \*= -1;

}

var nod = NOD(new List<int> { a.Numerator, a.Denominator });

if (nod != 1)

{

a.Denominator /= nod;

a.Numerator /= nod;

}

return a;

}

}

}

**Frac.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace fraction

{

public class Frac : TFrac

{

public Frac(int a, int b) : base(a, b)

{

}

public Frac(string str) : base(str)

{

}

public Frac() : base()

{

}

public static Frac operator +(Frac a, Frac b)

{

return (Frac)a.Add(b);

}

public static Frac operator \*(Frac a, Frac b)

{

return (Frac)a.Multiplication(b);

}

public static Frac operator -(Frac a, Frac b)

{

return (Frac)a.Difference(b);

}

public static Frac operator /(Frac a, Frac b)

{

return (Frac)a.Division(b);

}

public static Frac operator /(int a, Frac b)

{

return (Frac)(new Frac(a, 1)).Division(b);

}

public override bool Equals(object obj)

{

Frac frac = (Frac)obj;

if ((frac.Numerator == this.Numerator) && (this.Denominator == frac.Denominator))

{

return true;

}

else return false;

}

public static bool operator ==(Frac a, Frac b)

{

return (a.Numerator == b.Numerator) && (a.Denominator == b.Denominator);

}

public static bool operator !=(Frac a, Frac b)

{

return (a.Numerator != b.Numerator) || (a.Denominator != b.Denominator);

}

public static bool operator >(Frac a, Frac b)

{

return ((double)a.Numerator / (double)a.Denominator) > ((double)b.Numerator / (double)b.Denominator);

}

public static bool operator <(Frac a, Frac b)

{

return ((double)a.Numerator / (double)a.Denominator) < ((double)b.Numerator / (double)b.Denominator);

}

public override int GetHashCode()

{

return this.Numerator.GetHashCode() + this.Denominator.GetHashCode();

}

public override string ToString()

{

return GetString();

}

}

}

* 1. **Код тестов**

**UnitTest1.cs**

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using fraction;

namespace UniyTestProject

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestTFracInt()

{

TFrac frc = new Frac(10, 5);

Assert.AreEqual(2, frc.Numerator);

Assert.AreEqual(1, frc.Denominator);

}

[TestMethod]

public void TestTFracInt2()

{

void Action()

{

new Frac(1, 0);

}

Action action = new Action(Action);

Assert.ThrowsException<MyException>(action);

}

[TestMethod]

public void TestTFracStr1()

{

var frc = new Frac("99/88");

Assert.AreEqual(9, frc.Numerator);

Assert.AreEqual(8, frc.Denominator);

}

[TestMethod]

public void TestTFracStr2()

{

void Action()

{

new Frac("1/0");

}

Action action = new Action(Action);

Assert.ThrowsException<MyException>(action);

}

[TestMethod]

public void TestCopy()

{

var a = new Frac(10, 5);

var fCopy = a.Copy();

Assert.AreEqual(a.Numerator, fCopy.Numerator);

Assert.AreEqual(a.Denominator, fCopy.Denominator);

}

[TestMethod]

public void TestAdd()

{

var a = new Frac(1, 2);

var b = new Frac(1, 3);

var res = a + b;

Assert.AreEqual(5, res.Numerator);

Assert.AreEqual(6, res.Denominator);

}

[TestMethod]

public void TestDifference()

{

var a = new Frac(1, 2);

var b = new Frac(1, 3);

var res = a - b;

Assert.AreEqual(1, res.Numerator);

Assert.AreEqual(6, res.Denominator);

}

[TestMethod]

public void TestMultiplication()

{

var a = new Frac(11, 2);

var b = new Frac(13, 7);

var res = a \* b;

Assert.AreEqual(143, res.Numerator);

Assert.AreEqual(14, res.Denominator);

}

[TestMethod]

public void TestDivision()

{

var a = new Frac(1, 2);

var b = new Frac(2, 4);

var res = a / b;

Assert.AreEqual(1, res.Numerator);

Assert.AreEqual(1, res.Denominator);

}

[TestMethod]

public void TestSquare()

{

var a = new Frac(3, 2);

var res = a.Square();

Assert.AreEqual(9, res.Numerator);

Assert.AreEqual(4, res.Denominator);

}

[TestMethod]

public void TestReverse()

{

var a = new Frac(3, 2);

var res = a.Reverse();

Assert.AreEqual(2, res.Numerator);

Assert.AreEqual(3, res.Denominator);

}

[TestMethod]

public void TestMinus()

{

var a = new Frac(3, 2);

var res = a.Minus();

Assert.AreEqual(-3, res.Numerator);

Assert.AreEqual(2, res.Denominator);

}

[TestMethod]

public void TestRavn()

{

var a = new Frac(1, 2);

var b = new Frac(1, 2);

var res = a == b;

Assert.IsTrue(res);

}

[TestMethod]

public void TestMore()

{

var a = new Frac(2, 3);

var b = new Frac(1, 2);

var res = a > b;

Assert.IsTrue(res);

}

[TestMethod]

public void TestGetNumeratorNumber()

{

var a = new Frac(2, 3);

var res = a.GetNumeratorNumber();

Assert.AreEqual(2, res);

}

[TestMethod]

public void TestGetDenominatorNumber()

{

var a = new Frac(2, 3);

var res = a.GetDenominatorNumber();

Assert.AreEqual(3, res);

}

[TestMethod]

public void TestGetNumeratorString()

{

var a = new Frac(2, 3);

var res = a.GetNumeratorString();

Assert.AreEqual("2", res);

}

[TestMethod]

public void TestGetDenominatorString()

{

var a = new Frac(2, 3);

var res = a.GetDenominatorString();

Assert.AreEqual("3", res);

}

[TestMethod]

public void TestGetString()

{

var a = new Frac(7, 10);

var res = a.GetString();

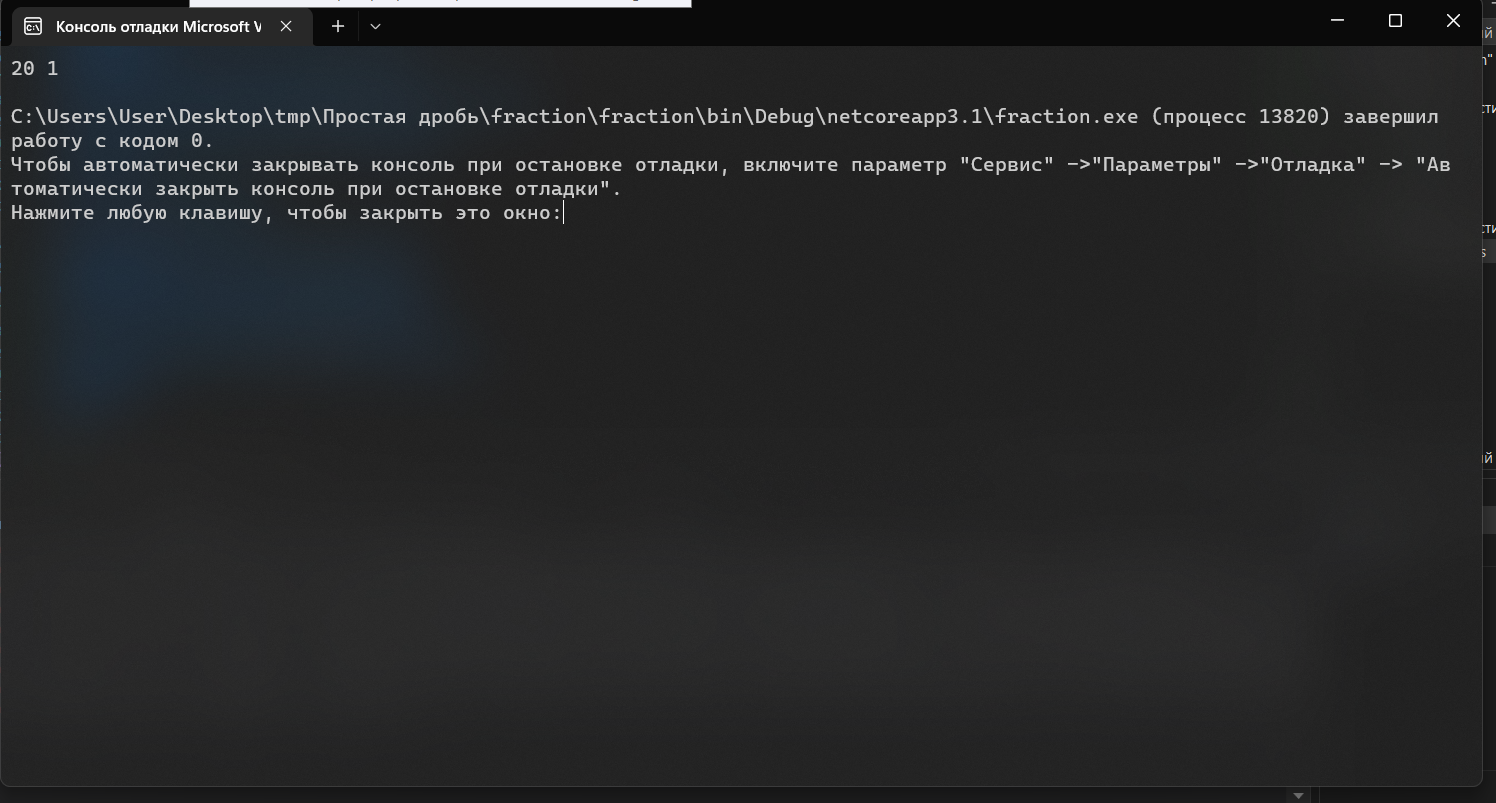
Assert.AreEqual("7/10", res);

}

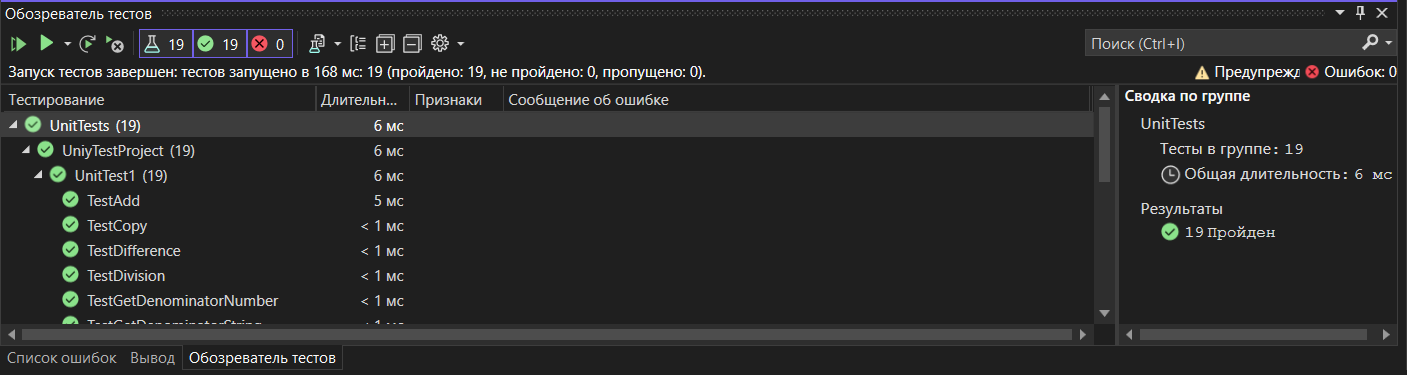
}

}

1. **Результаты** 
   1. **Пример работы программы**



* 1. **Результаты тестирования**



1. **Вывод**

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