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

Современные технологии программирования

Практическая работа №8

«Параметризованный абстрактный тип данных «Процессор»»

Выполнил: студент 4 курса

группы ИП-111 Кузьменок Денис Витальевич

Проверил преподаватель: Зайцев Михаил Георгиевич

Новосибирск, 2024 г.

# Цель

Сформировать практические навыки: реализации параметризованного абстрактного типа данных с помощью шаблона классов С++.

# Задание

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «Процессор», используя шаблон классов С++.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

**Спецификация типа данных “Процессор”.**

## ADT TProc

**Данные**

Процессор (тип TProc) выполняет двухоперандные операции TOprtn = (None, Add, Sub, Mul, Dvd) и однооперандные операции - функции TFunc = (Rev, Sqr) над значениями типа T. Левый операнд и результат операции хранится в поле Lop\_Res, правый - в поле Rop. Оба поля имеют тип T. Процессор может находиться в состояниях: «операция установлена» - поле Operation не равно None (значение типа TOprtn) или в состоянии «операция не установлена» - поле Operation = None. Значения типа TProc - изменяемые. Они изменяются операциями: «Сброс операции» (OprtnClear), «Выполнить операцию» (OprtnRun), «Вычислить функцию» (FuncRun), «Установить операцию» (OprtnSet), «Установить левый операнд» (Lop\_Res\_Set), «Установить правый операнд» (Rop\_Set), «Сброс калькулятора» (ReSet). На значениях типа T должны быть определены указанные выше операции и функции.

# Реализация:

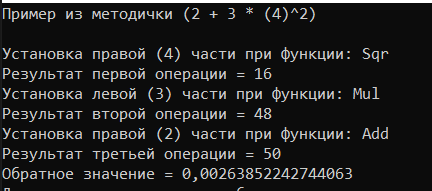


Рис. 1 – Результат проверки работоспособности программы.

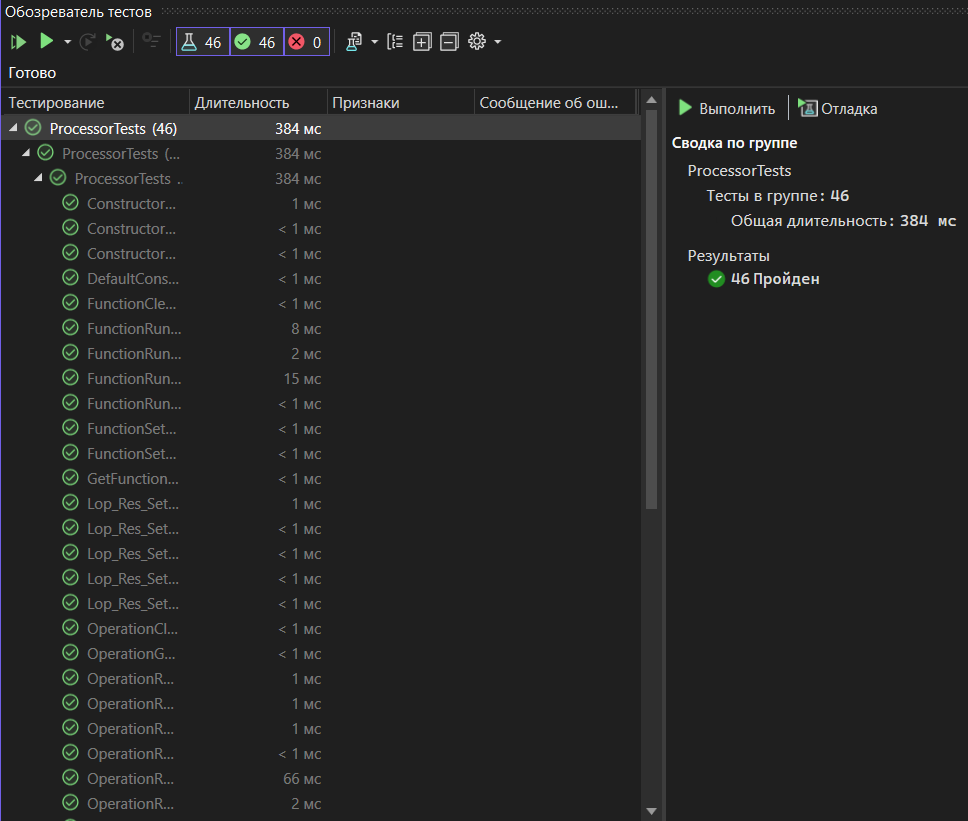


Рис. 2 – Результат выполнения модульных тестов.

## DefaultConstructor

* + **Что проверяет:** создание объекта типа Troc и задание значений по умолчанию.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
  + **Ожидаемое значение:** EOperation.None, EFunction.None, Lop\_Res = 0, Rop = 0.

## ConstructorWithValues\_int

* + **Что проверяет:** создание объекта типа Troc и задание значений целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(5, 10) — создаётся объект класса TProc.
  + **Ожидаемое значение:** EOperation.None, EFunction.None, Lop\_Res = 10, Rop = 5.

## ConstructorWithValues\_float

* + **Что проверяет:** создание объекта типа Troc и задание значений числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(-4.73f, 0.43f) — создаётся объект класса TProc.
  + **Ожидаемое значение:** EOperation.None, EFunction.None, Lop\_Res = -4.73f, Rop = 0.43f.

## ConstructorWithValues\_double

* + **Что проверяет:** создание объекта типа Troc и задание значений числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(0.0005, -94.9263) — создаётся объект класса TProc.
  + **Ожидаемое значение:** EOperation.None, EFunction.None, Lop\_Res = 0.0005, Rop = -94.9263.

## OperationClear\_ResetsOperation\_Int

* + **Что проверяет:** установка операции и её очищение.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(2);
    - proc.OperationClear()
  + **Ожидаемое значение:** EOperation.None.

## OperationRun\_Add\_Int

* + **Что проверяет:** корректность выполнения операции сложения с целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(5, 3) — создаётся объект класса TProc.
    - proc.OperationSet(1);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 8 — результат сложения двух целых чисел.

## OperationRun\_Add\_Float

* + **Что проверяет:** корректность выполнения операции сложения с числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(5.72f, 3.20f) — создаётся объект класса TProc.
    - proc.OperationSet(1);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 8.92f — результат сложения двух дробных чисел.

## OperationRun\_Add\_Double

* + **Что проверяет:** корректность выполнения операции сложения с числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(8.32, -6.99) — создаётся объект класса TProc.
    - proc.OperationSet(1);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 1.33 — результат сложения двух дробных чисел.

## OperationRun\_Sub\_Int

* + **Что проверяет:** корректность выполнения операции вычитания с целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(5, 3) — создаётся объект класса TProc.
    - proc.OperationSet(2);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 2 — результат вычитания двух целых чисел.

## OperationRun\_Sub\_Float

* + **Что проверяет:** корректность выполнения операции вычитания с числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(9.11f, -7.091f) — создаётся объект класса TProc.
    - proc.OperationSet(2);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 16.201f — результат вычитания двух дробных чисел.

## OperationRun\_Sub\_Double

* + **Что проверяет:** корректность выполнения операции вычитания с числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(-24.85, 22.15) — создаётся объект класса TProc.
    - proc.OperationSet(2);
    - proc.OperationRun()

## Ожидаемое значения:

* + - -47 — результат вычитания двух дробных чисел.

## OperationRun\_Mul\_Int

* + **Что проверяет:** корректность выполнения операции умножения с целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(62, 4) — создаётся объект класса TProc.
    - proc.OperationSet(3);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 248 — результат произведения двух целых чисел.

## OperationRun\_Mul\_Float

* + **Что проверяет:** корректность выполнения операции умножения с числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(0.1f, -62.23f) — создаётся объект класса TProc.
    - proc.OperationSet(3);
    - proc.OperationRun()

## Ожидаемое значения:

* + - -6.223f — результат произведения двух дробных чисел.

## OperationRun\_Mul\_Double

* + **Что проверяет:** корректность выполнения операции умножения с числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(0.001, 723.6813) — создаётся объект класса TProc.
    - proc.OperationSet(3);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 0.7236813 — результат умножения двух дробных чисел.

## OperationRun\_Div\_Int

* + **Что проверяет:** корректность выполнения операции деления с целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(10, 2) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 5 — результат деления двух целых чисел.

## OperationRun\_Div\_Float

* + **Что проверяет:** корректность выполнения операции деления с числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(73.5f, 4.25f) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - 17.2941176f — результат деления двух дробных чисел.

## OperationRun\_Div\_Double

* + **Что проверяет:** корректность выполнения операции деления с числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(-14.5, 2.5) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - -5.8 — результат умножения двух дробных чисел.

## OperationRun\_Div\_ByZero\_Int

* + **Что проверяет:** корректность выполнения операции деления с целыми числами.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(10, 0) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - Исключение DivideByZeroException.

## OperationRun\_Div\_ByZero\_Float

* + **Что проверяет:** корректность выполнения операции деления с числами с плавающей запятой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(-72.01f, 0.0f) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - Исключение DivideByZeroException.

## OperationRun\_Div\_ByZero\_Double

* + **Что проверяет:** корректность выполнения операции деления с числами с плавающей запятой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(0.001, 0.0) — создаётся объект класса TProc.
    - proc.OperationSet(4);
    - proc.OperationRun()

## Ожидаемое значения:

* + - Исключение DivideByZeroException.

## OperationSet\_ValidOperation

* + **Что проверяет:** корректность установления операции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(1);

## Ожидаемое значения:

* + - true – т.к. такая операция возможна (Add).

## OperationSet\_InvalidOperation\_ThrowException

* + **Что проверяет:** корректность установления операции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(10);

## Ожидаемое значения:

* + - Исключение ArgumentException.

## OperationGet\_ReturnsCorrectOperation

* + **Что проверяет:** корректность возврата операции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(2);

## Ожидаемое значения:

* + - EOperation.Sub.

## FunctionClear\_ResetsFunction

* + **Что проверяет:** установка функции и её очищение.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.FunctionSet(1);
    - proc.FunctionClear()
  + **Ожидаемое значение:** EFunction.None.

## FunctionRun\_Rev\_Int\_Valid

* + **Что проверяет:** корректность выполнения поиска обратного числа к текущему целому.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(5, 2) — создаётся объект класса TProc.
    - proc.FunctionSet(1);
    - proc.FunctionRun()
  + **Ожидаемое значение:** 0 – т.к. при работе с целыми числами дробная часть игнорируется.

## FunctionRun\_Rev\_Float\_Valid

* + **Что проверяет:** корректность выполнения поиска обратного числа к текущему числу с плавающей точкой.

## Входные значения:

* + - TProc<float> proc = new TProc<float>() — создаётся объект класса TProc.
    - proc.Rop\_Set(-2.5f)
    - proc.FunctionSet(1);
    - proc.FunctionRun()
  + **Ожидаемое значение:** -0.4f – обратное число к числу -2.5f.

## FunctionRun\_Rev\_Double\_Valid

* + **Что проверяет:** корректность выполнения поиска обратного числа к текущему числу с плавающей точкой.

## Входные значения:

* + - TProc<double> proc = new TProc<double>() — создаётся объект класса TProc.
    - proc.Rop\_Set(100.0)
    - proc.FunctionSet(1);
    - proc.FunctionRun()
  + **Ожидаемое значение:** 0.01 – обратное число к числу 100.0.

## FunctionRun\_Rev\_Int\_Zero\_ThrowsException

* + **Что проверяет:** корректность выполнения поиска обратного числа к текущему целому.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(5, 0) — создаётся объект класса TProc.
    - proc.FunctionSet(1);
    - proc.FunctionRun()
  + **Ожидаемое значение:** исключение DivideByZeroException.

## FunctionSet\_ValidFunction

* + **Что проверяет:** корректность установления функции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(0);

## Ожидаемое значения:

* + - true – т.к. такая операция возможна (None).

## FunctionSet\_InvalidFunction

* + **Что проверяет:** корректность установления функции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.OperationSet(3);

## Ожидаемое значения:

* + - Исключение ArgumentException.

## GetFunction\_ReturnsCorrectFunction

* + **Что проверяет:** корректность возврата функции.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - proc.FunctionSet(2);

## Ожидаемое значения:

* + - EFunction.Sqr.

## Lop\_Res\_Set\_SetsCorrectValues\_Int

* + **Что проверяет:** установление левого операнда (операнда результата).

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - int newValue = 10;
    - proc.Lop\_Res\_Set(newValue);

## Ожидаемое значения:

* + - Установление левого операнда (операнда результата) целым числом.

## Lop\_Res\_Set\_SetsCorrectValues\_Float

* + **Что проверяет:** установление левого операнда (операнда результата).

## Входные значения:

* + - TProc<float> proc = new TProc<float>() — создаётся объект класса TProc.
    - float newValue = -9.05f;
    - proc.Lop\_Res\_Set(newValue);

## Ожидаемое значения:

* + - Установление левого операнда (операнда результата) числом с плавающей точкой.

## Lop\_Res\_Set\_SetsCorrectValues\_Double

* + **Что проверяет:** установление левого операнда (операнда результата).

## Входные значения:

* + - TProc<double> proc = new TProc<double>() — создаётся объект класса TProc.
    - double newValue = 15.82;
    - proc.Lop\_Res\_Set(newValue);

## Ожидаемое значения:

* + - Установление левого операнда (операнда результата) числом с плавающей точкой.

## Lop\_Res\_Set\_SetsCorrectValues\_Char

* + **Что проверяет:** установление левого операнда (операнда результата).

## Входные значения:

* + - TProc<char> proc = new TProc<char>() — создаётся объект класса TProc.
    - char newValue = ‘a’;
    - proc.Lop\_Res\_Set(newValue);

## Ожидаемое значения:

* + - Установление левого операнда (операнда результата) символом.

## Lop\_Res\_Set\_SetsCorrectValues\_String

* + **Что проверяет:** установление левого операнда (операнда результата).

## Входные значения:

* + - TProc<string> proc = new TProc<string>() — создаётся объект класса TProc.
    - string newValue = “hello”;
    - proc.Lop\_Res\_Set(newValue);

## Ожидаемое значения:

* + - Установление левого операнда (операнда результата) строкой.

## Rop\_Set\_SetsCorrectValues\_Int

* + **Что проверяет:** установление правого операнда.

## Входные значения:

* + - TProc<int> proc = new TProc<int>() — создаётся объект класса TProc.
    - int newValue = 5;
    - proc.Rop\_Set(newValue);

## Ожидаемое значения:

* + - Установление правого операнда целым числом.

## Lop\_Res\_Set\_SetsCorrectValues\_Float

* + **Что проверяет:** установление правого операнда.

## Входные значения:

* + - TProc<float> proc = new TProc<float>() — создаётся объект класса TProc.
    - float newValue = -11.5f;
    - proc.Rop\_Set(newValue);

## Ожидаемое значения:

* + - Установление правого операнда числом с плавающей точкой.

## Rop\_Set\_SetsCorrectValues\_Double

* + **Что проверяет:** установление правого операнда.

## Входные значения:

* + - TProc<double> proc = new TProc<double>() — создаётся объект класса TProc.
    - double newValue = 0.0234;
    - proc.Rop\_Set(newValue);

## Ожидаемое значения:

* + - Установление правого операнда числом с плавающей точкой.

## Rop\_Set\_SetsCorrectValues\_Char

* + **Что проверяет:** установление правого операнда.

## Входные значения:

* + - TProc<char> proc = new TProc<char>() — создаётся объект класса TProc.
    - char newValue = ‘m’;
    - proc.Rop\_Set(newValue);

## Ожидаемое значения:

* + - Установление правого операнда символом.

## Rop\_Set\_SetsCorrectValues\_String

* + **Что проверяет:** установление правого операнда.

## Входные значения:

* + - TProc<string> proc = new TProc<string>() — создаётся объект класса TProc.
    - string newValue = “byebye”;
    - proc.Rop\_Set(newValue);

## Ожидаемое значения:

* + - Установление правого операнда строкой.

## ReSetByDefault\_Int

* + **Что проверяет:** установление левого (результирующего) и правого операндов значениями по умолчанию.

## Входные значения:

* + - TProc<int> proc = new TProc<int>(4, 10) — создаётся объект класса TProc.
    - proc.ReSet();

## Ожидаемое значения:

* + - (0, 0) – значения int по умолчанию.

## ReSetByDefault\_Float

* + **Что проверяет:** установление левого (результирующего) и правого операндов значениями по умолчанию.

## Входные значения:

* + - TProc<float> proc = new TProc<float>(-25.5f, 4.7f) — создаётся объект класса TProc.
    - proc.ReSet();

## Ожидаемое значения:

* + - (0.0f, 0.0f) – значения float по умолчанию.

## ReSetByDefault\_Double

* + **Что проверяет:** установление левого (результирующего) и правого операндов значениями по умолчанию.

## Входные значения:

* + - TProc<double> proc = new TProc<double>(55.67, -23.974) — создаётся объект класса TProc.
    - proc.ReSet();

## Ожидаемое значения:

* + - (0.0, 0.0) – значения double по умолчанию.

## ReSetByDefault\_Char

* + **Что проверяет:** установление левого (результирующего) и правого операндов значениями по умолчанию.

## Входные значения:

* + - TProc<char> proc = new TProc<char>(‘a’, ‘b’) — создаётся объект класса TProc.
    - proc.ReSet();

## Ожидаемое значения:

* + - (‘\0’, ‘\0’) – значения char по умолчанию.

## ReSetByDefault\_String

* + **Что проверяет:** установление левого (результирующего) и правого операндов значениями по умолчанию.

## Входные значения:

* + - TProc<string> proc = new TProc<string>(“hello”, “byebye”) — создаётся объект класса TProc.
    - proc.ReSet();

## Ожидаемое значения:

* + - (null, null) – значения string по умолчанию.

# Вывод

В результате работы над лабораторной работой были сформированы практические навыки реализации параметризованного абстрактного типа данных с помощью шаблона классов С#, разработки функций классов на языке С#, разработка модульных тестов для тестирования функций классов и выполнения модульного тестирования на языке С# с помощью средств автоматизации Visual Studio.

# Листинг программы:

**Program.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab8**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

**string** info = "";

Console.WriteLine("Пример из методички (2 + 3 \* (4)^2)\n");

TProc<**int**> proc = **new** TProc<**int**>();

TProc<**double**> test = **new** TProc<**double**>();

proc.Rop\_Set(**4**);

info = proc.RetRop();

proc.FunctionSet(**2**);

Console.WriteLine($"Установка правой ({info}) части при функции: {proc.GetFunction()}");

proc.FunctionRun();

info = proc.RetRop();

Console.WriteLine($"Результат первой операции = {info}");

proc.Lop\_Res\_Set(**3**);

info = proc.RetLop\_Res();

proc.OperationSet(**3**);

Console.WriteLine($"Установка левой ({info}) части при функции: {proc.OperationGet()}");

proc.OperationRun();

info = proc.RetLop\_Res();

Console.WriteLine($"Результат второй операции = {info}");

proc.Rop\_Set(**2**);

info = proc.RetRop();

proc.OperationSet(**1**);

Console.WriteLine($"Установка правой ({info}) части при функции: {proc.OperationGet()}");

proc.OperationRun();

info = proc.RetLop\_Res();

Console.WriteLine($"Результат третьей операции = {info}");

test.Rop\_Set(**379.00**);

test.FunctionSet(**1**);

test.FunctionRun();

**double** result = test.RetTRop();

Console.WriteLine($"Обратное значение = {result}");

TProc<**string**> a = **new** TProc<**string**>("hello", "byebye");

a.ReSet();

}

}

}

**TProc.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab8**

{

**public** **class** **TProc**<T>

{

**public** EOperation Operation;

**public** EFunction Function;

**public** T Lop\_Res, Rop;

**public** **TProc**()

{

Operation = EOperation.None;

Function = EFunction.None;

Lop\_Res = **default**(T);

Rop = **default**(T);

}

**public** **TProc**(T lop\_res, T rop)

{

Operation = EOperation.None;

Function = EFunction.None;

Lop\_Res = lop\_res;

Rop = rop;

}

**public** **void** **OperationClear**()

{

Operation = EOperation.None;

}

**public** **void** **OperationRun**()

{

**switch** (Operation)

{

**case** EOperation.Add:

Lop\_Res = (**dynamic**)Lop\_Res + (**dynamic**)Rop;

**break**;

**case** EOperation.Sub:

Lop\_Res = (**dynamic**)Lop\_Res - (**dynamic**)Rop;

**break**;

**case** EOperation.Mul:

Lop\_Res = (**dynamic**)Lop\_Res \* (**dynamic**)Rop;

**break**;

**case** EOperation.Div:

**if** (Rop.Equals(**default**(T))) **throw** **new** DivideByZeroException("Деление на ноль не допускается!");

Lop\_Res = (**dynamic**)Lop\_Res / (**dynamic**)Rop;

**break**;

}

}

**public** **void** **OperationSet**(**int** newOperation)

{

**if**(newOperation >= **0** && newOperation <= **4**)

Operation = (EOperation)newOperation;

**else** **throw** **new** **ArgumentException**("Нет такой команды!");

}

**public** EOperation **OperationGet**()

{

**return** **this**.Operation;

}

**public** **void** **FunctionClear**()

{

Function = EFunction.None;

}

**public** **void** **FunctionRun**()

{

**switch**(Function)

{

**case** EFunction.Rev:

**if** (Rop.Equals(**default**(T)))

**throw** **new** **DivideByZeroException**("Нельзя делить на ноль!");

**if** (Rop.GetType().GetMethod("Rev")?.Invoke(Rop, **null**) == **null**)

{

Rop = **1** / (**dynamic**)Rop;

}

**else** Rop = (T)Rop.GetType().GetMethod("Rev")?.Invoke(Rop, **null**);

**break**;

**case** EFunction.Sqr:

**if** (Rop.GetType().GetMethod("Sqr")?.Invoke(Rop, **null**) == **null**)

{

Rop = (**dynamic**)Rop \* (**dynamic**)Rop;

}

**else** Rop = (T)Rop.GetType().GetMethod("Sqr")?.Invoke(Rop, **null**);

**break**;

}

}

**public** **void** **FunctionSet**(**int** newFunction)

{

**if**(newFunction >= **0** && newFunction <= **2**)

Function = (EFunction)newFunction;

**else** **throw** **new** **ArgumentException**("Нет такой функции!");

}

**public** EFunction **GetFunction**()

{

**return** **this**.Function;

}

**public** **void** **Lop\_Res\_Set**(T newLopres)

{

Lop\_Res = newLopres;

}

**public** **void** **Rop\_Set**(T newRopres)

{

Rop = newRopres;

}

**public** **void** **ReSet**()

{

Lop\_Res = **default**(T);

Rop = **default**(T);

}

**public** **string** **RetLop\_Res**()

{

**if** (Lop\_Res == **null**)

**return** **null**;

**object** str = Lop\_Res.GetType().GetMethod("Show")?.Invoke(Lop\_Res, **null**) ?? Lop\_Res;

**return** str.ToString();

}

**public** T **RetTLop\_Res**()

{

**object** str = Lop\_Res.GetType().GetMethod("Copy")?.Invoke(**null**, **new** **object**[] { Lop\_Res }) ?? Lop\_Res;

**return** (T)str;

}

**public** **string** **RetRop**()

{

**if** (Rop == **null**)

**return** **null**;

**object** str = Rop.GetType().GetMethod("Show")?.Invoke(Rop, **null**) ?? Rop;

**return** str.ToString();

}

**public** T **RetTRop**()

{

**object** str = Rop.GetType().GetMethod("Copy")?.Invoke(**null**, **new** **object**[] { Rop }) ?? Rop;

**return** (T)str;

}

**public** **override** **bool** **Equals**(**object** obj)

{

**if** (((**this**.Lop\_Res.Equals(((TProc<T>)obj).Lop\_Res))) && (**this**.Rop.Equals(((TProc<T>)obj).Rop)))

{

**return** **true**;

}

**else** **return** **false**;

}

}

}

**Enum.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab8**

{

**public** **enum** EOperation

{

None,

Add,

Sub,

Mul,

Div

}

**public** **enum** EFunction

{

None,

Rev,

Sqr

}

}

**UnitTests1.cs**

**using** **Microsoft.VisualStudio.TestTools.UnitTesting**;

**using** **System**;

**using** **lab8**;

**namespace** **ProcessorTests**

{

[TestClass]

**public** **class** **ProcessorTests**

{

[TestMethod]

**public** **void** **DefaultConstructor**()

{

TProc<**int**> proc = **new** TProc<**int**>();

Assert.AreEqual(EOperation.None, proc.Operation);

Assert.AreEqual(EFunction.None, proc.Function);

Assert.AreEqual(**0**, proc.Lop\_Res);

Assert.AreEqual(**0**, proc.Rop);

}

[TestMethod]

**public** **void** **ConstructorWithValues\_int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**5**, **10**);

Assert.AreEqual(EOperation.None, proc.Operation);

Assert.AreEqual(EFunction.None, proc.Function);

Assert.AreEqual(proc.RetTLop\_Res(), proc.Lop\_Res);

Assert.AreEqual(proc.RetTRop(), proc.Rop);

}

[TestMethod]

**public** **void** **ConstructorWithValues\_float**()

{

TProc<**float**> proc = **new** TProc<**float**>(-**4.73f**, **0.43f**);

Assert.AreEqual(EOperation.None, proc.Operation);

Assert.AreEqual(EFunction.None, proc.Function);

Assert.AreEqual(proc.RetTLop\_Res(), proc.Lop\_Res);

Assert.AreEqual(proc.RetTRop(), proc.Rop);

}

[TestMethod]

**public** **void** **ConstructorWithValues\_double**()

{

TProc<**double**> proc = **new** TProc<**double**>(**0.0005**, -**94.9263**);

Assert.AreEqual(EOperation.None, proc.Operation);

Assert.AreEqual(EFunction.None, proc.Function);

Assert.AreEqual(proc.RetTLop\_Res(), proc.Lop\_Res);

Assert.AreEqual(proc.RetTRop(), proc.Rop);

}

[TestMethod]

**public** **void** **OperationClear\_ResetsOperation\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.OperationSet(**2**);

proc.OperationClear();

Assert.AreEqual(EOperation.None, proc.OperationGet());

}

[TestMethod]

**public** **void** **OperationRun\_Add\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**5**, **3**);

proc.OperationSet(**1**);

proc.OperationRun();

**int** actual = **8**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Add\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(**5.72f**, **3.20f**);

proc.OperationSet(**1**);

proc.OperationRun();

**float** actual = **8.92f**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Add\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(**8.32**, -**6.99**);

proc.OperationSet(**1**);

proc.OperationRun();

**double** actual = **1.33**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Sub\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**5**, **3**);

proc.OperationSet(**2**);

proc.OperationRun();

**int** actual = **2**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Sub\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(**9.11f**, -**7.091f**);

proc.OperationSet(**2**);

proc.OperationRun();

**float** actual = **16.201f**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Sub\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(-**24.85**, **22.15**);

proc.OperationSet(**2**);

proc.OperationRun();

**double** actual = -**47**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Mul\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**62**, **4**);

proc.OperationSet(**3**);

proc.OperationRun();

**int** actual = **248**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Mul\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(**0.1f**, -**62.23f**);

proc.OperationSet(**3**);

proc.OperationRun();

**float** actual = -**6.223f**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Mul\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(**0.001**, **723.6813**);

proc.OperationSet(**3**);

proc.OperationRun();

**double** actual = **0.7236813**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Div\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**10**, **2**);

proc.OperationSet(**4**);

proc.OperationRun();

**int** actual = **5**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Div\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(**73.5f**, **4.25f**);

proc.OperationSet(**4**);

proc.OperationRun();

**float** actual = **17.2941176f**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **OperationRun\_Div\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(-**14.5**, **2.5**);

proc.OperationSet(**4**);

proc.OperationRun();

**double** actual = -**5.8**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

[ExpectedException(typeof(DivideByZeroException))]

**public** **void** **OperationRun\_Div\_ByZero\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**10**, **0**);

proc.OperationSet(**4**);

proc.OperationRun();

}

[TestMethod]

[ExpectedException(typeof(DivideByZeroException))]

**public** **void** **OperationRun\_Div\_ByZero\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(-**72.01f**, **0.0f**);

proc.OperationSet(**4**);

proc.OperationRun();

}

[TestMethod]

[ExpectedException(typeof(DivideByZeroException))]

**public** **void** **OperationRun\_Div\_ByZero\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(**0.001**, **0**);

proc.OperationSet(**4**);

proc.OperationRun();

}

[TestMethod]

**public** **void** **OperationSet\_ValidOperation**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.OperationSet(**1**);

Assert.AreEqual(proc.OperationGet(), EOperation.Add);

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

**public** **void** **OperationSet\_InvalidOperation\_ThrowsException**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.OperationSet(**10**);

}

[TestMethod]

**public** **void** **OperationGet\_ReturnsCorrectOperation**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.OperationSet(**2**);

Assert.AreEqual(EOperation.Sub, proc.OperationGet());

}

[TestMethod]

**public** **void** **FunctionClear\_ResetsFunction**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.FunctionSet(**1**);

proc.FunctionClear();

Assert.AreEqual(proc.GetFunction(), EFunction.None);

}

[TestMethod]

**public** **void** **FunctionRun\_Rev\_Int\_Valid**()

{

TProc<**int**> proc = **new** TProc<**int**>(**5**, **2**);

proc.FunctionSet(**1**);

proc.FunctionRun();

**string** actual = "0";

Assert.AreEqual(proc.RetRop(), actual);

}

[TestMethod]

**public** **void** **FunctionRun\_Rev\_Float\_Valid**()

{

TProc<**float**> proc = **new** TProc<**float**>();

proc.Rop\_Set(-**2.5f**);

proc.FunctionSet(**1**);

proc.FunctionRun();

**float** actual = -**0.4f**;

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **FunctionRun\_Rev\_Double\_Valid**()

{

TProc<**double**> proc = **new** TProc<**double**>();

proc.Rop\_Set(**100.0**);

proc.FunctionSet(**1**);

proc.FunctionRun();

**double** actual = **0.01**;

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

[ExpectedException(typeof(DivideByZeroException))]

**public** **void** **FunctionRun\_Rev\_Int\_Zero\_ThrowsException**()

{

TProc<**int**> proc = **new** TProc<**int**>(**5**, **0**);

proc.FunctionSet(**1**);

proc.FunctionRun();

}

[TestMethod]

**public** **void** **FunctionSet\_ValidFunction**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.FunctionSet(**0**);

Assert.AreEqual(proc.GetFunction(), EFunction.None);

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

**public** **void** **FunctionSet\_InvalidFunction**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.FunctionSet(**3**);

}

[TestMethod]

**public** **void** **GetFunction\_ReturnsCorrectFunction**()

{

TProc<**int**> proc = **new** TProc<**int**>();

proc.FunctionSet(**2**);

Assert.AreEqual(EFunction.Sqr, proc.GetFunction());

}

[TestMethod]

**public** **void** **Lop\_Res\_Set\_SetsCorrectValue\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>();

**int** newValue = **10**;

proc.Lop\_Res\_Set(newValue);

**int** actual = **10**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **Lop\_Res\_Set\_SetsCorrectValue\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>();

**float** newValue = -**9.05f**;

proc.Lop\_Res\_Set(newValue);

**float** actual = -**9.05f**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **Lop\_Res\_Set\_SetsCorrectValue\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>();

**double** newValue = **15.82**;

proc.Lop\_Res\_Set(newValue);

**double** actual = **15.82**;

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **Lop\_Res\_Set\_SetsCorrectValue\_Char**()

{

TProc<**char**> proc = **new** TProc<**char**>();

**char** newValue = 'a';

proc.Lop\_Res\_Set(newValue);

**char** actual = 'a';

Assert.AreEqual(proc.RetTLop\_Res(), actual);

}

[TestMethod]

**public** **void** **Lop\_Res\_Set\_SetsCorrectValue\_String**()

{

TProc<**string**> proc = **new** TProc<**string**>();

**string** newValue = "hello";

proc.Lop\_Res\_Set(newValue);

**string** actual = "hello";

Assert.AreEqual(proc.RetLop\_Res(), actual);

}

[TestMethod]

**public** **void** **Rop\_Set\_SetsCorrectValue\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>();

**int** newValue = **5**;

proc.Rop\_Set(newValue);

**int** actual = **5**;

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **Rop\_Set\_SetsCorrectValue\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>();

**float** newValue = -**11.5f**;

proc.Rop\_Set(newValue);

**float** actual = -**11.5f**;

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **Rop\_Set\_SetsCorrectValue\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>();

**double** newValue = **0.0234**;

proc.Rop\_Set(newValue);

**double** actual = **0.0234**;

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **Rop\_Set\_SetsCorrectValue\_Char**()

{

TProc<**char**> proc = **new** TProc<**char**>();

**char** newValue = 'm';

proc.Rop\_Set(newValue);

**char** actual = 'm';

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **Rop\_Set\_SetsCorrectValue\_String**()

{

TProc<**string**> proc = **new** TProc<**string**>();

**string** newValue = "byebye";

proc.Rop\_Set(newValue);

**string** actual = "byebye";

Assert.AreEqual(proc.RetTRop(), actual);

}

[TestMethod]

**public** **void** **ReSetByDefault\_Int**()

{

TProc<**int**> proc = **new** TProc<**int**>(**4**, **10**);

proc.ReSet();

**int** actaul = **0**;

Assert.AreEqual(proc.RetTLop\_Res(), actaul);

Assert.AreEqual(proc.RetTRop(), actaul);

}

[TestMethod]

**public** **void** **ReSetByDefault\_Float**()

{

TProc<**float**> proc = **new** TProc<**float**>(-**25.5f**, **4.7f**);

proc.ReSet();

**float** actaul = **0.0f**;

Assert.AreEqual(proc.RetTLop\_Res(), actaul);

Assert.AreEqual(proc.RetTRop(), actaul);

}

[TestMethod]

**public** **void** **ReSetByDefault\_Double**()

{

TProc<**double**> proc = **new** TProc<**double**>(**55.67**, -**23.974**);

proc.ReSet();

**double** actaul = **0.0**;

Assert.AreEqual(proc.RetTLop\_Res(), actaul);

Assert.AreEqual(proc.RetTRop(), actaul);

}

[TestMethod]

**public** **void** **ReSetByDefault\_Char**()

{

TProc<**char**> proc = **new** TProc<**char**>('a', 'b');

proc.ReSet();

**char** actaul = '\0';

Assert.AreEqual(proc.RetTLop\_Res(), actaul);

Assert.AreEqual(proc.RetTRop(), actaul);

}

[TestMethod]

**public** **void** **ReSetByDefault\_String**()

{

TProc<**string**> proc = **new** TProc<**string**>("hello", "byebye");

proc.ReSet();

**string** actaul = **null**;

Assert.AreEqual(proc.RetLop\_Res(), actaul);

Assert.AreEqual(proc.RetRop(), actaul);

}

}

}