**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине: «Объектно-ориентированное программирование»

на тему: «Особенности разработки и верификации в объектно-ориентированном языке»

Выполнил: студент гр. ИТП-21

Король В.Н.

Принял: преподаватель-стажер

Гуменников Е.Д.

Гомель 2022

**Цель работы:** изучить методы перегрузки на языке программирования *C*#. Научится создавать внутри классов операторы. Изучить работу модульных тестов в *Visual Studio*.

**Задание**

1. Необходимо разработать тип и выполнить для него перегрузку операций,

согласно варианта (таблица 2)

2. Тип должен быть размещён в библиотеке классов.

3. Весь код должен быть снабжён элементами документирования

4. Разработать не менее 15 модульных тестов для тестирования созданного класса

5. Создать консольное приложение с интерфейсом пользователя

6. Подключить созданную библиотеку в консольное приложение

7. Сгенерировать XML-файл с документацией по проекту

8. Верифицировать консольное приложение

**Вариант 14**

Реализовать тип, для выполнения операций над одномерным массивом. Реализовать операции сложения двух массивов, в результате которой элементы второго массива добавляются в конец первого массива, и умножения/деления на число, в результате которой все элементы умножаются/делятся на число. Предусмотреть возможность выполнения операций.

**Ход Работы**

Для работы с массива был создан класс *OperationOnArray* где параметром передается массив чисел типа *double*. После этого был создан метод *ShowArray* который выводит массив чисел на экран. Пример работы метода *ShowArray* указан на рисунке 1.

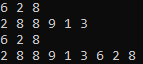


Рисунок 1 - Пример работы метода *ShowInfo*

Для сложения двух массив был создан оператор + который возвращает объект класса *OperationOnArray* и в качестве параметров принимает два объекта класса *OperationOnArray.* Внутри оператора создается временный массив *tmp* куда записывается первый и второй переданные массивы и возвращается массив *tmp* который передается в объект класса внутри оператора. Пример работы оператора + указан на рисунке 2.

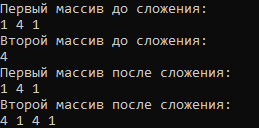


Рисунок 2 - Пример работы оператора +

Для умножения каждого элемента массива на число был создан оператор \* который принимаете объект класса *OperationOnArray* и число типа *double.* Внутри оператора \* каждый элемент массива умножается на заданное число и возвращается измененный массив. Пример работы оператора \* указан на рисунке 3.

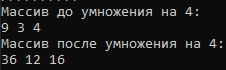


Рисунок 3 - Пример работы оператора \*

Оператор / работает аналогично оператору \* за исключением того что внутри происходит проверка на то чтобы переданное параметром число не равнялось 0. Пример работы оператора / указан на рисунке 4.

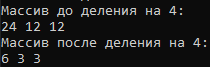


Рисунок 4 - Пример работы оператора /

Для проверки класса *OperationOnArray* были созданы модульные тесты которые передают некоторые заданные данные и сравнивают их с результатами методов. При помощи класса *Assert* можно проверить соответствие этих данные. Все модульные тесты указаны в листинге *OperationsOnArraysTests*. Для их создания были созданы операторы == и != в которые передаются два объекта и возвращает значение *bool.* Пример работы модульных тестов указан на рисунке 5.

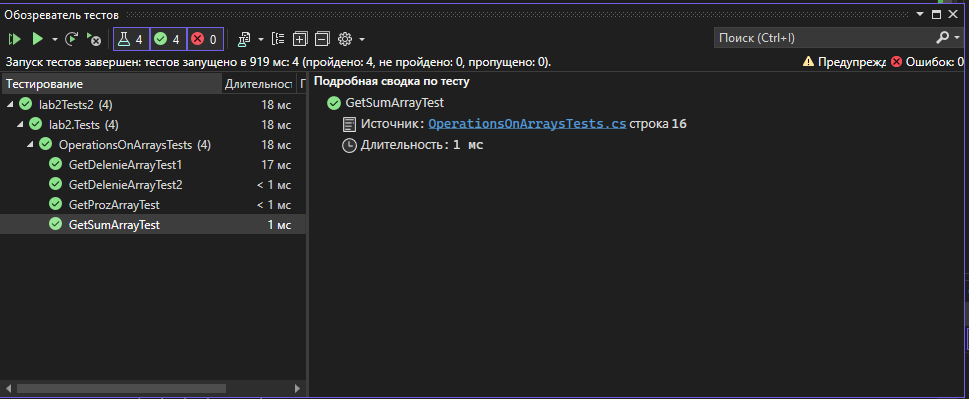


Рисунок 5 - Пример работы модульных тестов

**Вывод:** в ходе выполнения лабораторной работы были методы перегрузки на языка программирования *C*#. Научился создавать операторы. Научился делать модульные тесты в *Visual Studio*.

**Листинг класса OperationOnArray**

using System;

using System.Collections.Generic;

using System.Dynamic;

using System.Linq;

using System.Runtime.InteropServices.WindowsRuntime;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

public class OperationsOnArrays

{

public double[] Array;

public OperationsOnArrays(double[] array)

{

Array = array;

}

public OperationsOnArrays()

{

Random random = new Random();

double[] array = new double[random.Next(1, 10)];

for (int i = 0; i < array.Length; i++)

{

array[i] = random.Next(1, 10);

}

Array = array;

}

public void ShowArray()

{

foreach (var i in Array)

{

Console.Write($"{i} ");

}

Console.WriteLine();

}

public static OperationsOnArrays operator +(OperationsOnArrays array1, OperationsOnArrays array2)

{

OperationsOnArrays tmp = new OperationsOnArrays();

int kol = 0;

tmp.Array = new double[array1.Array.Length + array2.Array.Length];

for (int i = 0; i < array2.Array.Length; i++, kol++)

{

tmp.Array[kol] = array2.Array[i];

}

for (int i = 0; i < array1.Array.Length; i++, kol++)

{

tmp.Array[kol] = array1.Array[i];

}

return tmp;

}

public static OperationsOnArrays operator \*(OperationsOnArrays array, double num)

{

for (int i = 0; i < array.Array.Length; i++)

{

array.Array[i] \*= num;

}

return array;

}

public static OperationsOnArrays operator /(OperationsOnArrays array, double num)

{

if (num != 0)

{

for (int i = 0; i < array.Array.Length; i++)

{

array.Array[i] /= num;

}

}

return array;

}

public static bool operator ==(OperationsOnArrays array1, OperationsOnArrays array2)

{

for (int i = 0; i < array1.Array.Length; i++)

{

if (array1.Array[i] != array2.Array[i])

{

return false;

}

}

return true;

}

public static bool operator !=(OperationsOnArrays array1, OperationsOnArrays array2)

{

for (int i = 0; i < array1.Array.Length; i++)

{

if (array1.Array[i] != array2.Array[i])

{

return true;

}

}

return false;

}

}

}

**Листинг класса OperationsOnArraysTests**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using lab2;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Cryptography;

namespace lab2.Tests

{

[TestClass()]

public class OperationsOnArraysTests

{

[TestMethod()]

public void GetSumArrayTest()

{

double[] array1 = { -3, -2, -1, 0, 1, 2, 3, 4, 5 };

double[] array2 = { 1,2, 3, 4, 5, 6, 7};

double[] expect = { -3, -2, -1, 0, 1, 2, 3, 4, 5 , 1, 2, 3, 4, 5, 6, 7 };

OperationsOnArrays Expect = new OperationsOnArrays(expect);

OperationsOnArrays Array1 = new OperationsOnArrays(array1);

OperationsOnArrays Array2 = new OperationsOnArrays(array2);

OperationsOnArrays Array3 = new OperationsOnArrays();

Array3 = Array2 + Array1;

Assert.IsTrue(Array3 == Expect);

}

[TestMethod()]

public void GetProzArrayTest()

{

double[] expect = {2, 4, 6, 8, 10};

OperationsOnArrays Expect = new OperationsOnArrays(expect);

OperationsOnArrays array = new OperationsOnArrays(new double[] {1, 2, 3, 4, 5 });

array = array \* 2;

Assert.IsTrue(array == Expect);

}

[TestMethod()]

public void GetDelenieArrayTest1()

{

double[] expect = { 2, 4, 6, 8, 10 };

OperationsOnArrays Expect = new OperationsOnArrays(expect);

OperationsOnArrays array = new OperationsOnArrays(new double[] { 4, 8, 12, 16, 20 });

array = array / 2;

Assert.IsTrue(array == Expect);

}

[TestMethod()]

public void GetDelenieArrayTest2()

{

double[] expect = { 2, 4, 6, 8, 10 };

OperationsOnArrays Expect = new OperationsOnArrays(expect);

OperationsOnArrays array = new OperationsOnArrays(new double[] { 2, 4, 6, 8, 10 });

array = array / 0;

Assert.IsTrue(array == Expect);

}

}

}