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

Лабораторная работа №2

«Модульное тестирование библиотеки классов на C# средствами Visual Studio»

Вариант №4

Выполнил: студент 4 курса группы ИП-111

Кузьменок Денис Витальевич

Проверил преподаватель: Зайцев Михаил Георгиевич

Новосибирск, 2024 г.

**Цель:**

Сформировать практические навыки разработки модульных тестов для

библиотек классов C# и выполнения модульного тестирования с помощью средств автоматизации Visual Studio.

**Задание:**

Разработайте на языке С# класс, содержащий функции в соответствии с вариантом задания. Разработайте тестовые наборы данных для тестирования функций класса, по критерию С1. Протестируйте созданный класс с помощью средств автоматизации модульного тестирования Visual Studio. Проанализируйте результаты выполненных тестов по объёму покрытия тестируемого кода. Напишите отчёт о результатах проделанной работы.

1. Поиск максимума из трех чисел.
2. Функция получает двумерный массив вещественных переменных A. Отыскивает и возвращает произведение значений компонентов массива, у которых сумма значений индексов – чётная.
3. Функция получает двумерный массив вещественных переменных A. Отыскивает и возвращает минимальное значение компонентов массива, лежащих на и ниже главной диагонали.

**Реализация**

В ходе выполнения лабораторной работы мною была реализована библиотека классов C# LibraryLab2. Внутри данной библиотеки присутствует класс MyClass с методами в соответствии с вариантом задания, а именно:

public static int FindMax(int a, int b, int c) – Метод принимает три целочисленных параметра. Среди них находит максимальное значение и возвращает его.

public static double ProductOfElementsWithEvenSumOfIndices(double[,] A) – Метод принимает матрицу, заполненную данными типа чисел с плавающей точкой. В методе предусмотрено две генерации исключения:

1. Если была передана матрица, не содержащая в себе ни одного значения;
2. Если была передана матрица, у которой одно из измерений не было инициализировано.

Во вложенном цикле проходится по каждому элементу массива, и если индекс у этого элемента четный, то производится операция умножения. Таким образом, метод возвращает произведение всех элементов, у которых сумма индексов i и j является четной.

public static double MinValueOnAndBelowDiagonal(double[,] A) – Метод принимает матрицу, заполненную данными типа чисел с плавающей точкой. В методе предусмотрено три генерации исключения:

1. Если была передана матрица, не содержащая в себе ни одного значения;
2. Если была передана матрица, у которой одно из измерений не было инициализировано;
3. Т.к. по заданию нужно найти число в матрице, умеющую главную диагональ, то в случае, если передана матрица размера NxM, где N≠M, то генерируется исключение, потому что для данного пункта необходима квадратная матрица.

Находит и возвращает минимальное число в матрице, находящееся на или ниже главной диагонали.

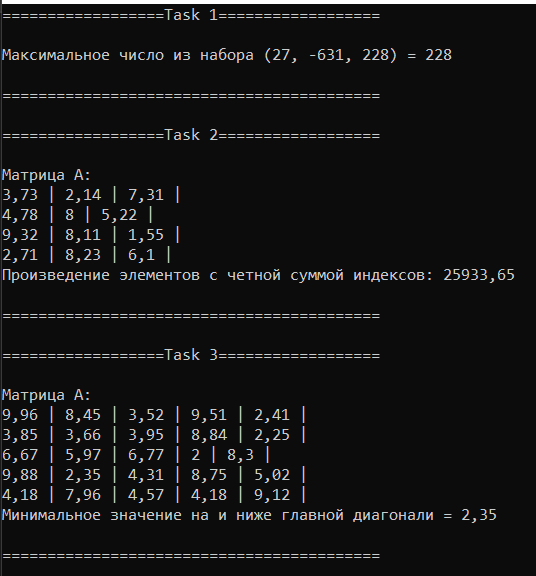
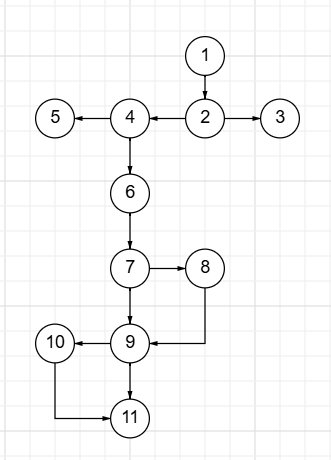


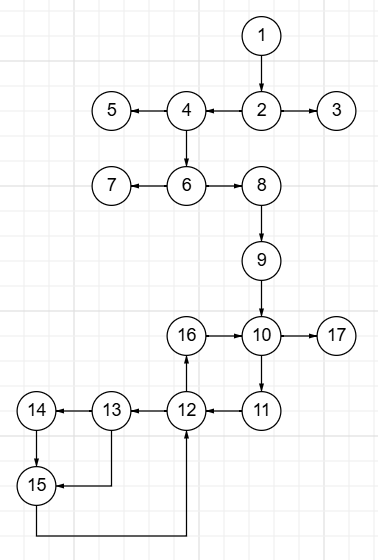
Рис. 1 – Результат запуска программы

В результате написания методов были созданы управляющие графы для каждого из них:

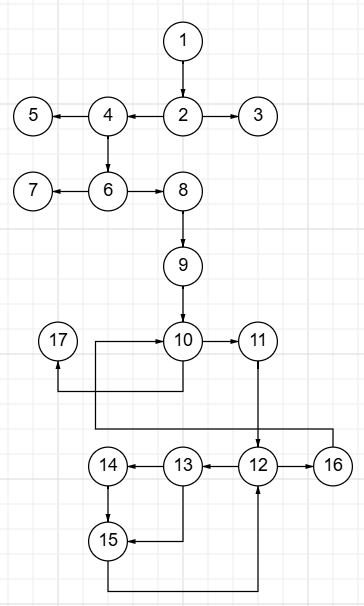
Для public static int FindMax(int a, int b, int c):



Для public static double ProductOfElementsWithEvenSumOfIndices(double[,] A):



Для public static double MinValueOnAndBelowDiagonal(double[,] A):



1. **FindMax**
2. **Корректные данные:**

* Входные данные: a = 5, b = -10, c = 2
* Ожидаемый результат: максимальное значение – a = 5

1. **Три одинаковых числа:**

* Входные данные: a = 10, b = 10, c = 10
* Ожидаемый результат: возвращение любого из трех значений (в моем случае возвращается переменная a)

1. **Все числа равны нулю:**

* Входные данные: a = 0, b = 0, c = 0
* Ожидаемый результат: возвращение 0

1. **ProductOfElementsWithEvenSumOfIndices**
2. **Корректные данные:**

* Входные данные: A = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
* Ожидаемый результат: нахождение произведения элементов матрицы, сумма индексов которых – четное число – 20.0

1. **Матрица пустая:**

* Входные данные: A = [[]]
* Ожидаемый результат: генерация исключения invalid\_exception\_2

1. **Матрица, у которой инициализировано лишь одно измерение:**

* Входные данные: A = [[1.0, null], [3.0, null], [5.0, null]]
* Ожидаемый результат: генерация исключения invalid\_exception\_3

1. **Матрица, в которой содержится лишь один элемент:**

* Входные данные: A = [[1.0]]
* Ожидаемый результат: возвращение этого элемента – 1.0

1. **MinValueOnAndBelowDiagonal**
2. **Корректные данные:**

* Входные данные: A = [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]
* Ожидаемый результат: нахождение минимального элемента на и ниже главной диагонали – 1.0

1. **Матрица пустая:**

* Входные данные: A = [[]]
* Ожидаемый результат: генерация исключения invalid\_exception\_2

1. **Матрица, у которой инициализировано лишь одно измерение:**

* Входные данные: A = [[1.0, null, null], [4.0, null, null], [7.0, null, null]]
* Ожидаемый результат: генерация исключения invalid\_exception\_3.

1. **Матрица, у которой длина первого измерения не равна длине второго измерения (матрица не является квадратной):**

* Входные данные: A = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
* Ожидаемый результат: генерация исключения invalid\_exception\_1.

Тестирование по структуре критериев C1 для каждого из трех методов предполагает:

1. Метод FindMax:

* Можно выделить следующие ветви: (1, 2), (2, 3), (2, 4), (4, 5), (4, 6, 7), (7, 8, 9), (7, 9), (9, 10, 11), (9, 11). Необходимо будет реализовать два тестовых метода (X, Y) = {(a = 0, b = 0, c = 0, “0”), (a = 10, b = 10, c = 10, “10”)}, чтобы покрыть первые четыре ветви. Для покрытия остальных ветвей требуется реализовать три тестовых метода (X, Y, Z) = {(a = 3, b = 2, c = 1, “3”), (a = 1, b = 3, c = 2, “3”), (a = 1, b = 2, c = 3, “3”)}.

1. Метод ProductOfElementsWithEvenSumOfIndices:

* Можно выделить следующие ветви: (1, 2), (2, 3), (2, 4), (4, 5), (4, 6), (6, 7), (6, 8, 9, 10), (10, 17), (10, 11, 12), (12, 16, 10), (12, 13), (13, 14, 15, 12), (13, 15, 12). Необходимо будет реализовать три тестовых метода (X, Y, Z) = {(A = [[]], invalid\_exception\_2), (A = [[1.0, null], [3.0, null]], invalid\_exception\_3), (A = [[1.0]], “1.0”)}, чтобы покрыть первые шесть ветвей. Для покрытия остальных ветвей требуется реализовать лишь один тестовый метод (X) = {(A = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]], “20.0”)}.

1. Метод MinValueOnAndBelowDiagonal:

* Можно выделить следующие ветви: (1, 2), (2, 3), (2, 4), (4, 5), (4, 6), (6, 7), (6, 8, 9, 10), (10, 17), (10, 11, 12), (12, 16, 10), (12, 13), (13, 14, 15, 12), (13, 15, 12). Необходимо будет реализовать три тестовых метода (X, Y, Z) = {(A = [[]], invalid\_exception\_2), (A = [[1.0, null], [3.0, null]], invalid\_exception\_3), (A = [[1.0]], “1.0”)}, чтобы покрыть первые шесть ветвей. Для покрытия остальных ветвей требуется реализовать лишь один тестовый метод (X) = {(A = [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]], “1.0”)}.

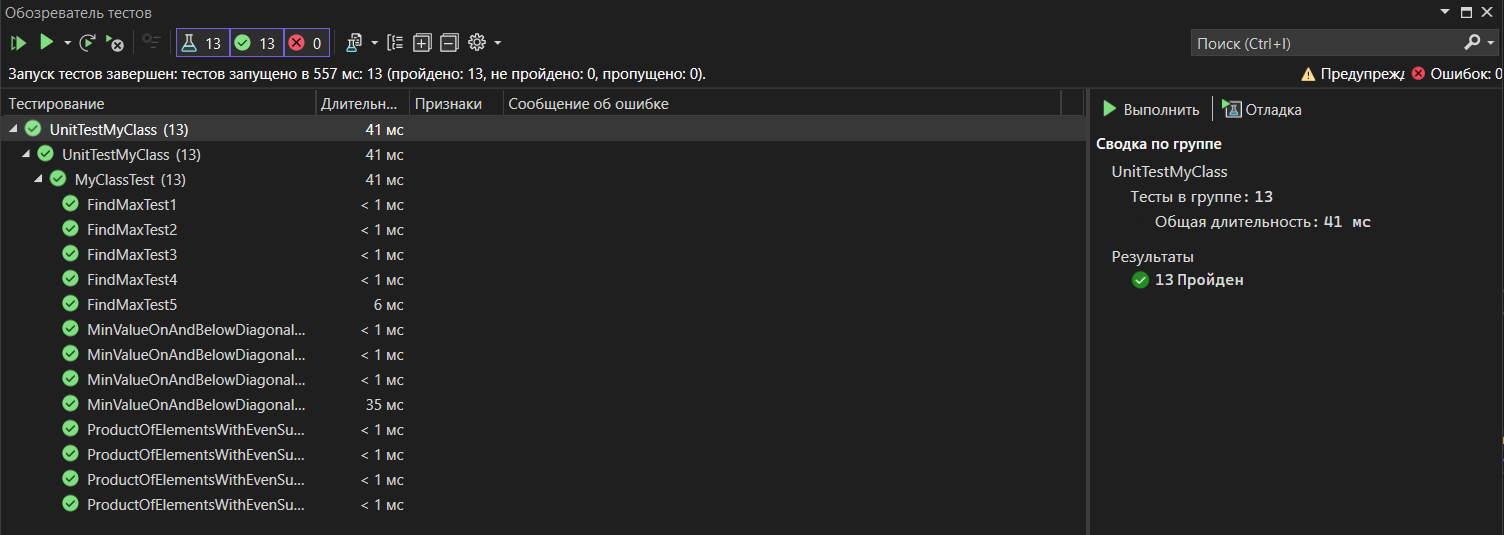


Рис. 2 – Результат выполнения модульных тестов.

**Вывод:**

В ходе выполнения данной работы была сформирована практическая база по разработке модульных тестов для библиотек классов C# с использованием средств автоматизации Visual Studio.

В результате проведенной работы были достигнуты следующие навыки:

1. Разработана библиотека классов C# для решения поставленной задачи;
2. Написан набор модульных тестов для проверки правильности работы библиотеки;
3. Настроен процесс автоматического выполнения модульных тестов в Visual Studio на основе критерия C1.

**Листинг программы:**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLab2;

namespace MyProjectLab2

{

class Program

{

const int MAX = 10;

const int MIN = 1;

static void Main(string[] args)

{

Console.WriteLine("==================Task 1==================\n");

int[] numbers = { 27, -631, 228, };

int max = MyClass.FindMax(numbers[0], numbers[1], numbers[2]);

Console.WriteLine($"Максимальное число из набора ({numbers[0]}, {numbers[1]}, {numbers[2]}) = {max}\n");

Console.WriteLine("==========================================\n");

Random rnd = new Random();

int rows = rnd.Next(2, 6);

int cols = rnd.Next(2, 6);

double[,] matrix\_secondTask = new double[rows, cols];

for(int i = 0; i < rows; i++)

{

for(int j = 0; j < cols; j++)

{

matrix\_secondTask[i, j] = Math.Round(((rnd.NextDouble() \* (MAX - MIN)) + MIN), 2);

}

}

Console.WriteLine("==================Task 2==================\n");

Console.WriteLine("Матрица A: ");

for(int i = 0; i < matrix\_secondTask.GetLength(0); i++)

{

for(int j = 0; j < matrix\_secondTask.GetLength(1); j++)

{

Console.Write(matrix\_secondTask[i, j] + " | ");

}

Console.WriteLine();

}

double product = MyClass.ProductOfElementsWithEvenSumOfIndices(matrix\_secondTask);

try

{

Console.WriteLine("Произведение элементов с четной суммой индексов: " + Math.Round(product, 2) + '\n');

}

catch (Exception ex)

{

Console.WriteLine("Ошибка: " + ex.Message);

}

Console.WriteLine("==========================================\n");

int size = rnd.Next(2, 6);

double[,] matrix\_thirdTask = new double[size, size];

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

matrix\_thirdTask[i, j] = Math.Round(((rnd.NextDouble() \* (MAX - MIN)) + MIN), 2);

}

}

Console.WriteLine("==================Task 3==================\n");

Console.WriteLine("Матрица A: ");

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

Console.Write(matrix\_thirdTask[i, j] + " | ");

}

Console.WriteLine();

}

double min = MyClass.MinValueOnAndBelowDiagonal(matrix\_thirdTask);

try

{

Console.WriteLine("Минимальное значение на и ниже главной диагонали = " + min + '\n');

}

catch(Exception ex)

{

Console.WriteLine("Ошибка: " + ex.Message);

}

Console.WriteLine("==========================================\n");

}

}

}

**MyClass.cs**

using System;

using System.Collections.Generic;

using System.Data;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LibraryLab2

{

public class MyClass

{

static string s1 = "Матрица должна быть квадратной!";

static string s2 = "Матрица не может быть null!";

static string s3 = "Матрица не может быть пустой!";

public class invalid\_exception\_1 : ArgumentException

{

public invalid\_exception\_1(string message) : base(message) { }

}

public class invalid\_exception\_2 : ArgumentException

{

public invalid\_exception\_2(string message) : base(message) { }

}

public class invalid\_exception\_3 : ArgumentException

{

public invalid\_exception\_3(string message) : base(message) { }

}

public static int FindMax(int a, int b, int c)

{

if (a == 0 && b == 0 && c == 0) return 0;

if (a == b && b == c) return a;

int max = a;

if(b > max)

{

max = b;

}

if(c > max)

{

max = c;

}

return max;

}

public static double ProductOfElementsWithEvenSumOfIndices(double[,] A)

{

if(A == null) throw new invalid\_exception\_2(s2);

if (A.GetLength(0) == 0 || A.GetLength(1) == 0) throw new invalid\_exception\_3(s3);

if (A.GetLength(0) == 1 && A.GetLength(1) == 1) return A[0, 0];

double product = 1.0;

for(int i = 0; i < A.GetLength(0); i++)

{

for(int j = 0; j < A.GetLength(1); j++)

{

if ((i + j) % 2 == 0)

{

product \*= A[i, j];

}

}

}

return product;

}

public static double MinValueOnAndBelowDiagonal(double[,] A)

{

if (A == null) throw new invalid\_exception\_2(s2);

if (A.GetLength(0) == 0 || A.GetLength(1) == 0) throw new invalid\_exception\_3(s3);

if (A.GetLength(0) != A.GetLength(1)) throw new invalid\_exception\_1(s1);

double min = double.MaxValue;

for(int i = 0; i < A.GetLength(0); i++)

{

for(int j = 0; j <= i; j++)

{

if (A[i, j] < min)

{

min = A[i, j];

}

}

}

return min;

}

}

}

**MyClassTest.cs**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using LibraryLab2;

namespace UnitTestMyClass

{

[TestClass]

public class MyClassTest

{

[TestMethod]

public void FindMaxTest1()

{

//arrange

int first = 3;

int second = 2;

int third = 1;

int expected = 3;

//act

int actual = MyClass.FindMax(first, second, third);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindMaxTest2()

{

//arrange

int first = 1;

int second = 3;

int third = 2;

int expected = 3;

//act

int actual = MyClass.FindMax(first, second, third);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindMaxTest3()

{

//arrange

int first = 1;

int second = 2;

int third = 3;

int expected = 3;

//act

int actual = MyClass.FindMax(first, second, third);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindMaxTest4()

{

//arrange

int first = 0;

int second = 0;

int third = 0;

int expected = 0;

//act

int actual = MyClass.FindMax(first, second, third);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindMaxTest5()

{

//arrange

int first = 10;

int second = 10;

int third = 10;

int expected = 10;

//act

int actual = MyClass.FindMax(first, second, third);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void ProductOfElementsWithEvenSumOfIndicesTest1()

{

//arrange

double[,] matrix = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0} };

double expected = 20.0;

//act

double actual = MyClass.ProductOfElementsWithEvenSumOfIndices(matrix);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void ProductOfElementsWithEvenSumOfIndicesTest2()

{

//arrange

double[,] matrix = { { 4.0} };

double expected = 4.0;

//act

double actual = MyClass.ProductOfElementsWithEvenSumOfIndices(matrix);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyClass.invalid\_exception\_2))]

public void ProductOfElementsWithEvenSumOfIndicesTest3()

{

//arrange

double[,] matrix = null;

//act

double actual = MyClass.ProductOfElementsWithEvenSumOfIndices(matrix);

}

[TestMethod]

[ExpectedException(typeof(MyClass.invalid\_exception\_3))]

public void ProductOfElementsWithEvenSumOfIndicesTest4()

{

//arrange

int row = 3, col = 0;

double[,] matrix = new double[row, col];

//act

double actual = MyClass.ProductOfElementsWithEvenSumOfIndices(matrix);

}

[TestMethod]

public void MinValueOnAndBelowDiagonalTest1()

{

//arrange

double[,] matrix = { { 43.0, 63.0, 87.0 }, { 1.0, 0.5, 0.4 }, { 115.0, -62.0, 193.0 } };

double expected = -62.0;

//act

double actual = MyClass.MinValueOnAndBelowDiagonal(matrix);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyClass.invalid\_exception\_1))]

public void MinValueOnAndBelowDiagonalTest2()

{

//arrange

double[,] matrix = { { 43.0, 63.0}, { 1.0, 0.5}, { 115.0, -62.0} };

//act

double actual = MyClass.MinValueOnAndBelowDiagonal(matrix);

}

[TestMethod]

[ExpectedException(typeof(MyClass.invalid\_exception\_2))]

public void MinValueOnAndBelowDiagonalTest3()

{

//arrange

double[,] matrix = null;

//act

double actual = MyClass.MinValueOnAndBelowDiagonal(matrix);

}

[TestMethod]

[ExpectedException(typeof(MyClass.invalid\_exception\_3))]

public void MinValueOnAndBelowDiagonalTest4()

{

//arrange

int rows = 0, cols = 3;

double[,] matrix = new double[rows, cols];

//act

double actual = MyClass.MinValueOnAndBelowDiagonal(matrix);

}

}

}