

1 ЛАБОРАТОРНАЯ РАБОТА №1

«Исследование способов анализа областей эквивалентности и построения тестовых последовательностей»

1.1 Цель работы

Исследовать способы анализа областей эквивалентности входных данных для тестирования программного обеспечения. Приобрести практические навыки составления построения тестовых последовательностей.

1.2 Вариант задания

Задача 1. Дана целочисленная прямоугольная матрица. Определить количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент.

Задача 2. Дана строка, среди символов которой есть двоеточие. Получить все символы, расположенные между первым и вторым двоеточием. Если второго двоеточия нет, то получить все символы, расположенные после единственного имеющегося двоеточия.

Задача 3. Программа, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.

1.3 Ход выполнения работы

1.3.1 В начале выполнения лабораторной работы был написан класс, который определяет количество отрицательных элементов в тех строках прямоугольной матрицы, которые содержат хотя бы один нулевой элемент. Код программы представлен в листинге 1.1.

Листинг 1.1 – Текст первого тестируемого класса

```
public class MatrixCounter
{
    public int CountNegativeNumbersInRowsWithZeros(List<List<int>>> data)
    {
        int result = 0;
        foreach (var row in data)
        {
            if (row.Contains(0))
            {
                var amount = row.Count(x => x < 0);
                result += amount;
            }
        }

        return result;
    }
}
```

Для этой программы были определены области эквивалентности, которые показаны в таблице 1.1.

Таблица 1.1 – Области эквивалентности для первого класса

Размер матрицы	Есть ноль	Есть отрицательные элементы в строке с нулём
1×1	Нет	-
1×1	Есть	-
N×M	Нет	-
N×M	Есть в одной строке	Нет
N×M	Есть в одной строке	Есть
N×M	Есть в нескольких строках	Нет
N×M	Есть в нескольких строках	Есть

Далее для этой программы по выявленным областям эквивалентности были разработаны примеры тестовых последовательностей, что показано в таблице 1.2.

Таблица 1.2 – Тестовые последовательности для первой программы

Входная матрица
1
0
[1, 2, 3] [4, 5, 6] [7, 8, 9]
[1, 5, 0, 0] [2, 3, 4, 5]
[1, 2, 3, 4] [0, 0, 0, -1] [6, 7, 8, 9] [1, 2, 3, 4]
[1, 2, 0] [4, 0, 6] [7, 0, 9]
[1, 0, 0, -1, 0] [2, 3, 4, 5, 6] [0, 0, -1, -6, 0]

Для данного класса были написаны юнит-тесты с тестовыми последовательностями определёнными ранее. Код тестов представлен в листинге 1.2. Результаты выполнения тестов представлены на рисунке 1.1.

Листинг 1.2 – Тесты первого тестируемого класса

```
public class MatrixCounterTests
{
    private readonly MatrixCounter _matrix;

    public MatrixCounterTests()
    {
        _matrix = new MatrixCounter();
    }
}
```

```

[Fact]
public void MatrixTest1()
{
    var source = new List<List<int>>
    {
        new List<int> { 1 },
    };

    var result =
_matrix.CountNegativeNumbersInRowsWithZeros(source);

    result.ShouldBeEquivalentTo(0);
}

[Fact]
public void MatrixTest2()
{
    var source = new List<List<int>>
    {
        new List<int> { 1 },
    };

    var result =
_matrix.CountNegativeNumbersInRowsWithZeros(source);

    result.ShouldBeEquivalentTo(0);
}

[Fact]
public void MatrixTest3()
{
    var source = new List<List<int>>
    {
        new List<int> { 1, 2, 3},
        new List<int> { 4, 5, 6},
        new List<int> { 7, 8, 9},
    };

```

```

        var result
        _matrix.CountNegativeNumbersInRowsWithZeros(source);

        result.ShouldBeEquivalentTo(0);
    }

    [Fact]
    public void MatrixTest4()
    {
        var source = new List<List<int>>
        {
            new List<int> { 1, 5, 0, 0},
            new List<int> { 2, 3, 4, 5},
        };

        var result
        _matrix.CountNegativeNumbersInRowsWithZeros(source);

        result.ShouldBeEquivalentTo(0);
    }

    [Fact]
    public void MatrixTest5()
    {
        var source = new List<List<int>>
        {
            new List<int> { 1, 2, 3, 4},
            new List<int> { 0, 0, 0, -1},
            new List<int> { 6, 7, 8, 9},
            new List<int> { 1, 2, 3, 4},
        };

        var result
        _matrix.CountNegativeNumbersInRowsWithZeros(source);

        result.ShouldBeEquivalentTo(1);
    }

```

```

[Fact]
public void MatrixTest6()
{
    var source = new List<List<int>>
    {
        new List<int> { 1, 2, 0},
        new List<int> { 4, 0, 6},
        new List<int> { 7, 0, 9},
    };

    var result =
    _matrix.CountNegativeNumbersInRowsWithZeros(source);

    result.ShouldBeEquivalentTo(0);
}

[Fact]
public void MatrixTest7()
{
    var source = new List<List<int>>
    {
        new List<int> { 1, 0, 0, -1, 0},
        new List<int> { 2, 3, 4, 5, 6},
        new List<int> { 0, 0, -1, -6, 0},
    };

    var result =
    _matrix.CountNegativeNumbersInRowsWithZeros(source);

    result.ShouldBeEquivalentTo(3);
}
}

```



Рисунок 1.1 – Результат выполнения юнит-тестов первого класса

1.3.2 Далее был написан класс, который получает все символы, расположенные между первым и вторым двоеточием. Если второго двоеточия нет, то получает все символы, расположенные после единственного имеющегося двоеточия. Код программы представлен в листинге 1.3.

Листинг 1.3 – Текст класса расширения строки

```
public static class StringExtension
{
    public static string GetStringBetweenColons(this string str)
    {
        var match = Regex.Match(str, ":[^:]*:");
        var result = match.Groups[0].Value.Replace(":", "");
        return result;
    }
}
```

Для этой программы были определены области эквивалентности, которые показаны в таблице 1.3.

Таблица 1.3 – Области эквивалентности для второй программы

Размер строки	Двоеточие
0	-
1	Есть

>1	Нет
>1	Одно
>1	2
>1	>2

Далее для этой программы по выявленным областям эквивалентности были разработаны примеры тестовых последовательностей, что показано в таблице 1.4.

Таблица 1.4 – Тестовые последовательности для второй программы

Входная строка	Выходная строка
a	
:	
abcde	
:kukaracha	kukaracha
:kukaracha:	kukaracha
:kukaracha:azaza:	kukaracha

Также для данного класса были написаны юнит-тесты. Код тестов представлен на листинге 1.4. На рисунке 1.2 показано правильное выполнение тестов с разработанными последовательностями.

Листинг 1.4 – Тесты класса расширения строки

```
using Zadanie;
using Shouldly;

namespace ZadanieTests;

public class StringExtensionTests
{
    [Fact]
    public void StringExtensionsTest5()
    {
        var source = ":kukaracha:";

        var result = source.GetStringBetweenColons();
```



```
        result.ShouldBeEquivalentTo("kukaracha");
    }

    [Fact]
    public void StringExtensionsTest6()
    {
        var source = ":kukaracha:azaza:";

        var result = source.GetStringBetweenColons();

        result.ShouldBeEquivalentTo("kukaracha");
    }

    [Fact]
    public void StringExtensionsTest4()
    {
        var source = ":kukaracha";

        var result = source.GetStringBetweenColons();

        result.ShouldBeEquivalentTo("kukaracha");
    }

    [Fact]
    public void StringExtensionsTest3()
    {
        var source = "abcde";

        var result = source.GetStringBetweenColons();

        result.ShouldBeEmpty();
    }

    [Fact]
    public void StringExtensionsTest2()
    {
        var source = ":";
```

```

        var result = source.GetStringBetweenColons();

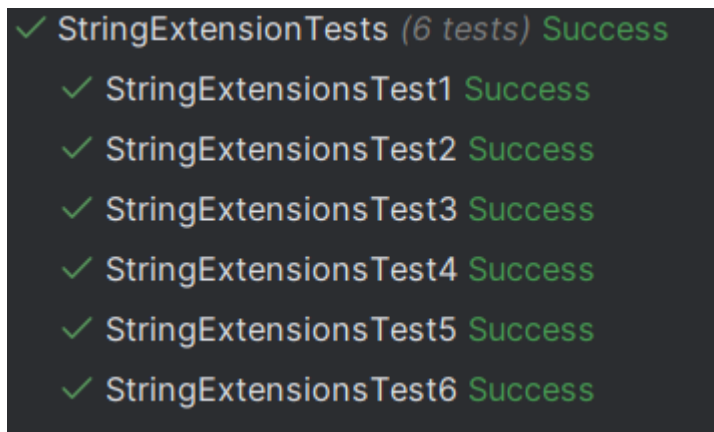
        result.ShouldBeEmpty();
    }

    [Fact]
    public void StringExtensionsTest1()
    {
        var source = "a";

        var result = source.GetStringBetweenColons();

        result.ShouldBeEmpty();
    }
}

```



```

✓ StringExtensionTests (6 tests) Success
  ✓ StringExtensionsTest1 Success
  ✓ StringExtensionsTest2 Success
  ✓ StringExtensionsTest3 Success
  ✓ StringExtensionsTest4 Success
  ✓ StringExtensionsTest5 Success
  ✓ StringExtensionsTest6 Success

```

Рисунок 1.2 – Тесты класса расширения строки

1.3.3 Последней была написана программа, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации. Код программы представлен в листинге 1.5.

Листинг 1.5 – Текст третьей программы

```

public class SadFileReader
{

```

```

        private readonly List<char> _punctuationMarks = new List<char>() {'.', '!', '?', ';', ':'};

        public void WriteToConsoleStringWithMaxPunctuationMarksCount(string
fileName)
        {
            var maxStrings = new List<string>();
            int punctuationMarksMaxAmount = 0;

            var reader = new StreamReader(fileName);
            string? line;
            while ((line = reader.ReadLine()) != null)
            {
                int punctuationMarksCount = line.Count(x =>
                _punctuationMarks.Contains(x));
                if (punctuationMarksCount > punctuationMarksMaxAmount)
                {
                    maxStrings.Clear();
                    maxStrings.Add(line);
                    punctuationMarksMaxAmount = punctuationMarksCount;
                }
                else if (punctuationMarksCount == punctuationMarksMaxAmount)
                {
                    maxStrings.Add(line);
                }
            }

            foreach (var maxString in maxStrings)
            {
                Console.WriteLine(maxString);
            }
        }
    }

```

Для этой программы были определены области эквивалентности, которые показаны в таблице 1.5.

Таблица 1.5 – Области эквивалентности для третьей программы

Количество строк	Знаки пунктуации	Количество максимальных строк
0	-	-
1	Нет	-
1	Есть	1
>1	Есть в одной строке	1
>1	Есть в нескольких строках	1
>1	Есть в нескольких строках	>1

Далее для этой программы по выявленным областям эквивалентности были разработаны примеры тестовых последовательностей, что показано в таблице 1.6.

Таблица 1.6 – Тестовые последовательности для третьей программы

Входной файл	Выходные строки
aaaaaaaaaa	aaaaaaaaaa
a45bcde,xxx	a45bcde,xxx
Aaaaaa Aaaaa, aaaaaa	Aaaaa,
Aaaaaa Aaaaa,,, Aaaaa, aaaaaa	Aaaaa,,,
Aaaaaa Aaaaa,,, Aaaaa,,, aaaaaa	Aaaaa,,, Aaaaa,,,

На рисунке 1.3 показано правильное выполнение нескольких тестовых примеров из разработанной последовательности.

```
Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile.txt

Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile1.txt
aaaaaaaaaaa

Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile2.txt
a45bcde,xxx

Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile3.txt
Aaaaa,

Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile4.txt
Aaaaa,,,

Test for: C:\Users\k_dod\repos\5Semestr\tpo\lr1\lr1\ConsoleApp1\testFile5.txt
Aaaaa,,,
Aaaaa,,,
```

Рисунок 1.3 – Результат выполнения тестов третьей программы

Выводы

В ходе лабораторной работы были исследованы способы анализа областей эквивалентности входных данных для тестирования программного обеспечения. Также были приобретены практические навыки составления построения тестовых последовательностей. В конце выполнения лабораторной работы был написан отчет.