Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «Современные платформы программирования»

Выполнил:

Студент 3 курса

Группы ПО-4

Иваненко И. Л.

Проверил:

Монтик Н. С.

Брест 2022

Вариант 10

Цель работы: освоить приемы тестирования кода на примере использования библиотеки NUnit

# Задание 1

## Текст задания:

Создать новй класс Sum и протестировать его метод

### Класс SumUtil:

namespace Lab5;  
  
public static class SumUtil  
{  
 public static long GetSum(params long[] values)  
 {  
 return values.Sum();  
 }  
}

### Код тестирующего метода:

[Test]  
public void Sum\_SumCorrect()  
{  
 long val1 = 3;  
 long val2 = 5;  
 long val3 = -6;  
 Assert.AreEqual(SumUtil.GetSum(val1, val2, val3), val1 + val2 + val3);  
}

### Результат выполнения теста:

Graphical user interface, text, application

Description automatically generated

# Задание 2

## Текст задания

Создайте класс StringUtils, в котором будут находится реализуемые функции. Напишите тесты для реализуемых функций.

## StringUtils:

namespace Lab5;  
  
public static class StringUtils  
{  
 public static string Repeat(string pattern, int repeats)  
 {  
 if (repeats < 0)  
 {  
 throw new ArgumentOutOfRangeException(nameof(repeats), "Repeats count cannot be less than zero");  
 }  
   
 var result = "";  
 for (var i = 0; i < repeats; i++)  
 {  
 result += pattern;  
 }  
  
 return result;  
 }  
  
 public static string Repeat(string pattern, char separator, int repeats)  
 {  
 if (repeats < 0)  
 {  
 throw new ArgumentOutOfRangeException(nameof(repeats), "Repeats count cannot be less than zero");  
 }  
  
 var result = "";  
 for (var i = 0; i < repeats; i++)  
 {  
 result += $"{pattern}{separator}";  
 }  
  
 return result.TrimEnd(separator);  
 }  
  
 public static string Keep(string str, string pattern)  
 {  
 if (pattern == "\*")  
 {  
 return str;  
 }  
  
 foreach (var c in str.Where(c => !pattern.Contains(c)))  
 {  
 str = str.Replace(c.ToString(), "");  
 }  
   
 return str;  
 }  
  
 public static string Loose(string str, string pattern)  
 {  
 return pattern == "\*"   
 ? str   
 : pattern.Aggregate(str, (current, c) => current.Replace(c.ToString(), ""));  
 }  
  
 public static int IndexOfDifference(string str1, string str2)  
 {  
 if (str1 == str2)  
 {  
 return -1;  
 }  
  
 int iterationsCount = str1.Length > str2.Length ? str2.Length : str1.Length;  
  
 for (var i = 0; i < iterationsCount; i++)  
 {  
 if (str1[i] != str2[i])  
 {  
 return i;  
 }  
 }  
  
 return iterationsCount;  
 }  
  
 public static string Common(string str1, string str2)  
 {  
 string smallest = str1.Length <= str2.Length ? str1 : str2;  
 string largest = str1.Length > str2.Length ? str1 : str2;  
 return GetCommon(smallest, largest);  
 }  
  
 private static string GetCommon(string smallest, string largest)  
 {  
 if (largest.Contains(smallest))  
 {  
 return smallest;  
 }  
   
 if (smallest.Length == 2)  
 {  
 foreach (char c in smallest)  
 {  
 if (largest.Contains(c))  
 {  
 return c.ToString();  
 }  
 }  
  
 return "";  
 }  
  
 if (largest.Contains(smallest[..^1]))  
 {  
 return smallest[..^1];  
 }  
  
 if (largest.Contains(smallest[1..]))  
 {  
 return smallest;  
 }  
  
 return GetCommon(smallest[1..^1], largest);  
 }  
  
 public static string SubstringBetween(string str, string open, string close)  
 {  
 if (!str.Contains(open) || !str.Contains(close))  
 {  
 return "";  
 }  
  
 int startIndex = str.IndexOf(open, StringComparison.Ordinal) + open.Length;  
 int endIndex = str.IndexOf(close, StringComparison.Ordinal);  
 return str.Substring(startIndex, endIndex - startIndex);  
 }  
  
 public static int LevenshteinDistance(string str1, string str2)  
 {  
 if (str1 == str2)  
 {  
 return 0;  
 }  
  
 int levenshteinDistance = Math.Abs(str1.Length - str2.Length);  
 string smallest = str1.Length >= str2.Length ? str2 : str1;  
 for (var i = 0; i < smallest.Length; i++)  
 {  
 if (str1[i] != str2[i])  
 {  
 levenshteinDistance++;  
 }  
 }  
  
 return levenshteinDistance;  
 }  
  
 public static int HamingDistance(string str1, string str2)  
 {  
 if (str1.Length != str2.Length)  
 {  
 throw new ArgumentException("Length of str1 and str2 should be equal");  
 }  
  
 return str1.Where((t, i) => t != str2[i]).Count();  
 }  
}

## UtilsTests:

using System;  
using Lab5;  
using NUnit.Framework;  
  
namespace Tests;  
  
[TestFixture]  
public class UtilsTests  
{  
 [Test]  
 [TestCase("abc", 3, "abcabcabc")]  
 [TestCase("aa", 1, "aa")]  
 [TestCase("a", 0, "")]  
 public void Repeat\_NoSeparator\_CorrectResult(string pattern, int repeatsCount, string expectedResult)  
 {  
 Assert.AreEqual(StringUtils.Repeat(pattern, repeatsCount), expectedResult);  
 }  
  
 [Test]  
 public void Repeat\_NoSeparator\_RepeatsLessThanZero\_ThrowsException()  
 {  
 var str = "abc";  
 int repeatsCount = -3;  
 Assert.Throws<ArgumentOutOfRangeException>(() => StringUtils.Repeat(str, repeatsCount));  
 }  
  
 [Test]  
 [TestCase("abcdef", "abc", "abc")]  
 [TestCase("abcccdef", "abc", "abccc")]  
 [TestCase("abcdef", "lll", "")]  
 public void Keep\_Success(string str, string pattern, string expectedResult)  
 {  
 Assert.AreEqual(StringUtils.Keep(str, pattern), expectedResult);  
 }  
  
 [Test]  
 [TestCase("abcdef", "abc", "def")]  
 [TestCase("abcccdef", "abc", "def")]  
 [TestCase("abcdef", "lll", "abcdef")]  
 public void Loose\_Success(string str, string pattern, string expectedResult)  
 {  
 Assert.AreEqual(StringUtils.Loose(str, pattern), expectedResult);  
 }  
  
 [Test]  
 [TestCase("abcdef", "abckpr", 3)]  
 [TestCase("abc", "abc", -1)]  
 [TestCase("abc", "def", 0)]  
 public void IndexOfDifference\_IndexCorrect(string str1, string str2, int expectedIndex)  
 {  
 Assert.AreEqual(StringUtils.IndexOfDifference(str1, str2), expectedIndex);  
 }  
  
 [Test]  
 [TestCase("abcdef", "kkcdkk", "cd")]  
 [TestCase("abcdef", "", "")]  
 [TestCase("aaaa", "aabb", "aa")]  
 [TestCase("bbbb", "aabb", "bb")]  
 public void Common\_ReturnValueCorrect(string str1, string str2, string expected)  
 {  
 Assert.AreEqual(StringUtils.Common(str1, str2), expected);  
 }  
  
 [Test]  
 [TestCase("[abcdef]", "[", "]", "abcdef")]  
 [TestCase("[abcdef]", "[a", "d", "bc")]  
 [TestCase("[abcdef]", "x", "y", "")]  
 public void SubstringBetween\_ReturnValueCorrect(string str, string open, string close, string expected)  
 {  
 Assert.AreEqual(StringUtils.SubstringBetween(str, open, close), expected);  
 }  
  
 [Test]  
 [TestCase("abcdef", "vvvdef", 3)]  
 [TestCase("abcdef", "", 6)]  
 [TestCase("vvvvvv", "abc", 6)]  
 [TestCase("vvvvv", "vvvvv", 0)]  
 public void LevenshteinDistance\_ReturnValueCorrect(string str1, string str2, int expectedDistance)  
 {  
 Assert.AreEqual(StringUtils.LevenshteinDistance(str1, str2), expectedDistance);  
 }  
  
 [Test]  
 [TestCase("abcdef", "abcdef", 0)]  
 [TestCase("abcdef", "abcxxx", 3)]  
 public void HamingDistance\_ReturnValueCorrect(string str1, string str2, int expectedDistance)  
 {  
 Assert.AreEqual(StringUtils.HamingDistance(str1, str2), expectedDistance);  
 }  
  
 [Test]  
 public void HamingDistance\_StringLengthNotEqual\_ThrowsException()  
 {  
 var str1 = "aaa";  
 var str2 = "aaaa";  
  
 Assert.Throws<ArgumentException>(() => StringUtils.HamingDistance(str1, str2));  
 }  
  
 [Test]  
 public void Sum\_SumCorrect()  
 {  
 long val1 = 3;  
 long val2 = 5;  
 long val3 = -6;  
 Assert.AreEqual(SumUtil.GetSum(val1, val2, val3), val1 + val2 + val3);  
 }  
}

## Результат выполнения тестов:

Text

Description automatically generated

# Вывод

В ходе выполнения данной лабораторной работы были приобретены практические навыки создания тестов с помощью библиотеки NUnit.