|  |
| --- |
| ФГБОУ ВО ЮГОРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ |
| ИНЖЕНЕРНАЯ ШКОЛА ЦИФРОВЫХ ТЕХНОЛОГИЙ |

ОТЧЕТ

по лабораторной работе №1

Тема:

«Модульное тестирование»

По дисциплине:

# «Тестирование и отладка программного обеспечения»

Выполнил: студент группы

1521б

Максимчук Михаил Юрьевич

Проверил: старший преподаватель

Усманов Руслан Талгатович

г. Ханты-Мансийск, 2024 г.

# Цель работы: понять как пишутся модульные тесты и получить навыки проведения модульного тестирования на языке программирования С#.

# Ход работы

1)Были написаны все необходимые функции (Приложение 1).

2)Были написаны тесты (Приложение 2).

3)Все тесты были успешно пройдены (Рис.1).

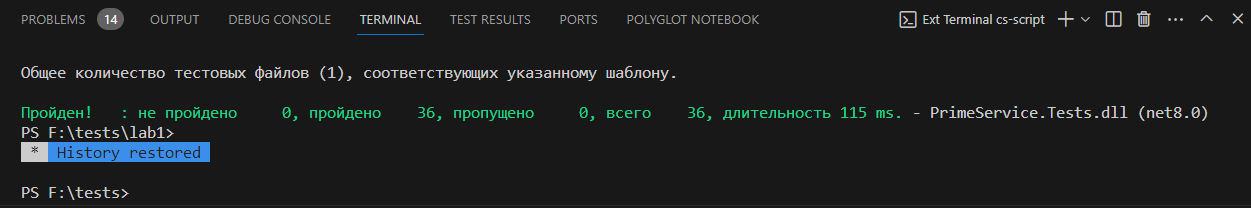


Рисунок 1 – Тестирование.

**Заключение**:

В ходе выполнения лабораторной работы я научился проводить модульное тестирование.

**Приложение 1.** Файл Program.cs

using System;

using System.Text;

namespace Prime.Services

{

    public class Test

    {

        public int[] Bubble(int[] array)

        {

            if (array.Length == 0)

            {

                return [-1];

            }

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

            {

                for (int j = 0; j < array.Length - 1 - i; j++)

                {

                    if (array[j] > array[j + 1])

                    {

                        int a = array[j];

                        array[j] = array[j + 1];

                        array[j + 1] = a;

                    }

                }

            }

            return array;

        }

        public bool Palindrome(string str)

        {

            if (str.Length == 0)

            {

                return false;

            }

            for (int i = 0, j = str.Length - 1; i < j; i++, j--)

                if (str[i] != str[j])

                    return false;

            return true;

        }

        public int Factorial(int number)

        {

            if (number < 0 || number >= 13)

            {

                return -1;

            }

            int result = 1;

            for (int i = number; i > 0; i--)

            {

                result \*= i;

            }

            return result;

        }

        public int Fibonachi(int number)

        {

            if (number < 0 || number > 47)

            {

                return -1;

            }

            if (number == 0 || number == 1) return number;

            return Fibonachi(number - 1) + Fibonachi(number - 2);

        }

        public bool Substring(string str, string sub)

        {

            if (sub.Length > str.Length)

            {

                return false;

            }

            else if (str.Length == 0 && sub.Length == 0)

            {

                return true;

            }

            else if (str.Length == 0 || sub.Length == 0)

            {

                return false;

            }

            return str.Contains(sub);

        }

        public bool PrimeNumber(int number)

        {

            if (number < 0)

            {

                return false;

            }

            for (int i = 2; i < Math.Round(Math.Sqrt(number)); i++)

                if (number % i == 0)

                    return false;

            return true;

        }

        public int ReverseNumber(int number)

        {

            string str\_number = number.ToString();

            if (str\_number.Length == 1)

            {

                return number;

            }

            string reverse = new string(str\_number.Reverse().ToArray());

            char minus = reverse[reverse.Length - 1];

            if (minus == '-')

            {

                string res = minus + reverse.Substring(0, reverse.Length - 1);

                if (long.Parse(res) <= 2147483647 && long.Parse(res) >= -2147483647)

                {

                    return int.Parse(res);

                }

            }

            if (long.Parse(reverse) <= 2147483647 && long.Parse(reverse) >= -2147483647)

            {

                return int.Parse(reverse);

            }

            return 0;

        }

        public string RomanNumber(int number)

        {

            if (number < 0 || number > 3999)

                return "error";

            if (number == 0) return "N";

            int[] numbers\_dict = new int[] { 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };

            string[] roman\_dict = new string[] { "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" };

            StringBuilder result = new StringBuilder();

            for (int i = 0; i < numbers\_dict.Length; i++)

            {

                while (number >= numbers\_dict[i])

                {

                    number -= numbers\_dict[i];

                    result.Append(roman\_dict[i]);

                }

            }

            return result.ToString();

        }

    }

    public class Program

    {

        public static void Main()

        {

        }

    }

}

**Приложение 2.** ФайлUnitTest1.cs

using NUnit.Framework.Interfaces;

using Prime.Services;

namespace PrimeService.Tests;

    public class Tests

    {

        private Test obj;

        [SetUp]

        public void Setup()

        {

            obj = new Test();

        }

        [TestCase(new int[] { 2, 0, 1, 6 }, new int[] { 0, 1, 2, 6 })]

        [TestCase(new int[] { -10, 6, -1, 2 }, new int[] { -10, -1, 2, 6 })]

        [TestCase(new int[] { -9, 0, 4, 2 }, new int[] { -9, 0, 2, 4 })]

        [TestCase(new int[] { }, new int[] { -1 })]

        public void Bubble(int[] array, int[] res)

        {

            Assert.AreEqual(res, obj.Bubble(array));

        }

        [TestCase("1221", true)]

        [TestCase("1212", false)]

        [TestCase("1", true)]

        [TestCase("", false)]

        public void Palindrome(string str, bool res)

        {

            Assert.AreEqual(res, obj.Palindrome(str));

        }

        [TestCase(-12, -1)]

        [TestCase(1000, -1)]

        [TestCase(4, 24)]

        [TestCase(6, 720)]

        public void Factorial(int number, int res)

        {

            Assert.AreEqual(res, obj.Factorial(number));

        }

        [TestCase(0, 0)]

        [TestCase(-12, -1)]

        [TestCase(1000, -1)]

        [TestCase(3, 2)]

        public void Fibonachi(int number, int res)

        {

            Assert.AreEqual(res, obj.Fibonachi(number));

        }

        [TestCase("", "", true)]

        [TestCase("qwer1ty", "1", true)]

        [TestCase("", "a", false)]

        [TestCase("qwerty", "", false)]

        public void Substring(string str, string sub, bool res)

        {

            Assert.AreEqual(res, obj.Substring(str, sub));

        }

        [TestCase(-12, false)]

        [TestCase(12, false)]

        [TestCase(3, true)]

        [TestCase(100, false)]

        public void PrimeNumber(int number, bool res)

        {

            Assert.AreEqual(res, obj.PrimeNumber(number));

        }

        [TestCase(-12, -21)]

        [TestCase(-1, -1)]

        [TestCase(1000, 1)]

        [TestCase(2147483647, 0)]

        [TestCase(0, 0)]

        public void ReverseNumber(int number, int res)

        {

            Assert.AreEqual(res, obj.ReverseNumber(number));

        }

        [TestCase(1, "I")]

        [TestCase(2, "II")]

        [TestCase(100, "C")]

        [TestCase(1010, "MX")]

        [TestCase(0, "N")]

        [TestCase(-1000, "error")]

        [TestCase(4000, "error")]

        public void RomanNumber(int n, string res)

        {

            Assert.AreEqual(res, obj.RomanNumber(n));

        }

    }

(P.s. я забыл подкрепить отчет раньше, хотя лабу сдал давно)