Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «МОП ЭВМ»

КОНТРОЛЬНАЯ РАБОТА

по дисциплине «Технологии программирования»

Вариант 9

Студент группы 6ИСб-1 И. Нозимзода

Преподаватель С.Ю. Александров

2018

Содержание

Содержание 2

Задания 4

Введение 5

1 Линейные программы 6

1.1 Описание программы 6

1.2 Текст программы 6

1.3 Тестирование программы 7

2 Разветвляющиеся вычислительные процессы 8

2.1 Описание программы 8

2.2 Текст программы 8

2.3 Тестирование программы 9

3 Организация циклов 10

3.1 Описание программы 10

3.2 Текст программы 10

3.3 Тестирование программы 11

4 Простейшие классы 12

4.1 Описание программы 12

4.2 Текст программы 12

4.3 Тестирование программы 15

5 Одномерные массивы 16

5.1 Описание программы 16

5.2 Текст программы 16

5.3 Тестирование программы 18

6 Двумерные массивы 19
6.1 Описание программы 19
6.2 Текст программы 19
6.3 Тестирование программы 21
7 Строки 22
7.1 Описание программы 22
7.2 Текст программы 22
7.3 Тестирование программы 23
8 Классы и операции 24
8.1 Описание программы 24
8.2 Текст программы 24
8.3 Тестирование программы 30
9 Наследование 31
9.1 Описание программы 31
9.2 Текст программы 32
9.1 Тестирование программы 35
10 Структуры 36
10.1 Описание программы 36
10.2 Текст программы 37
10.3 Тестирование программы 38
Заключение 39
Список использованных источников 40

Задания

1 Линейные программы.

2 Разветвляющиеся вычислительные процессы.

3 Организация циклов.

4 Простейшие классы.

5 Одномерные массивы.

6 Двумерные массивы.

7 Строки.

8 Классы и операции.

9 Наследование.

10 Структуры.

Введение

Язык C# как средство обучения программированию обладает рядом несомненных достоинств. Он хорошо организован, строг, большинство его конструкций логичны и удобны. Развитые средства диагностики и редактирования кода делают процесс программирования приятным и эффективным.

Немаловажно, что C# является не учебным, а профессиональным языком, предназначенным для решения широкого спектра задач, и в первую очередь - в быстро развивающейся области создания распределенных приложений.

1 Линейные программы

Линейной называется программа, все операторы которой выполняются последовательно в том порядке, в котором они записаны.

1.1 Описание программы

Написать программу для расчета по двум формулам.

$$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2; \qquad z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cdot\cos(\alpha+\beta).$$

1.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 1.1.

Листинг 1.1 – Текст файла prog1.cs

```
using System;

namespace Lab1
{
    class Lab1
    {
        static void Main(string[] args)
        {
            Console.Write("Введите число A: ");
            double a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Введите число B: ");
            double b = Convert.ToDouble(Console.ReadLine());

            double z1 = Math.Pow(Math.Cos(a) - Math.Cos(b), 2) - Math.Pow(Math.Sin(a) -
Math.Sin(b), 2);
            double z2 = -4 * Math.Pow(Math.Sin((a - b) / 2.0), 2) * Math.Cos(a + b);

            Console.WriteLine("z1 = " + z1);
            Console.WriteLine("z2 = " + z2);

            Console.Read();
        }
    }
}
```

1.3 Тестирование программы

Результат работы программы приведен на рисунке 1.1.

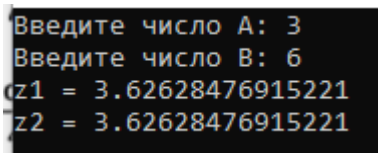

Рисунок 1.1 – Результат работы линейной программы

2 Разветвляющиеся вычислительные процессы

Разветвляющиеся вычислительные процессы – это вычислительные процессы, в которых предусмотрено разветвление выполняемой последовательности действий в зависимости от результата проверки какого-либо условия.

2.1 Описание программы

Написать программу, которая определяет, попадает ли точка с заданными координатами в область, закрашенную на рисунке серым цветом (рисунок 2.1). Результат вывести в виде текстового сообщения.

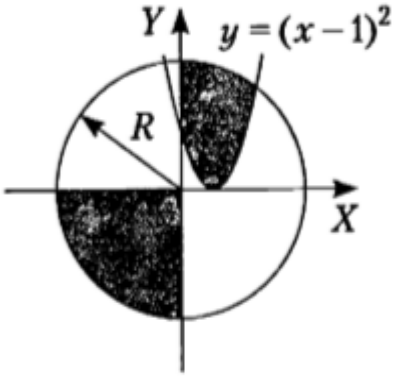

Рисунок 2.1 – График

2.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 2.1.

Листинг 2.1 – Текст файла prog2.cs

```
using System;

namespace Lab2
{
    class Lab2
    {
        class Task02_2
        {
            public string O(double x, double y, double r)
            {
                string flag = "не попадает";
                if (((y < 0) && (x < 0) && (x * x + y * y < r * r)) || (y > 0) && (x > 0)
&& (Math.Pow(x - 1, 2)  < y)) flag = "попадает";
                return flag;
```

```
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Расположение точки");
            while (true)
            {
                Console.Write("Введите X: ");
                double x = double.Parse(Console.ReadLine());
                Console.Write("Введите Y: ");
                double y = double.Parse(Console.ReadLine());
                Console.Write("Введите радиус окружности (R):");
                double R = double.Parse(Console.ReadLine());
                Task02_2 obl = new Task02_2();
                Console.WriteLine("Точка {0}", obl.O(x, y, R));
            }
        }
    }
}
```

2.3 Тестирование программы

Результат работы программы приведен на рисунке 2.2.

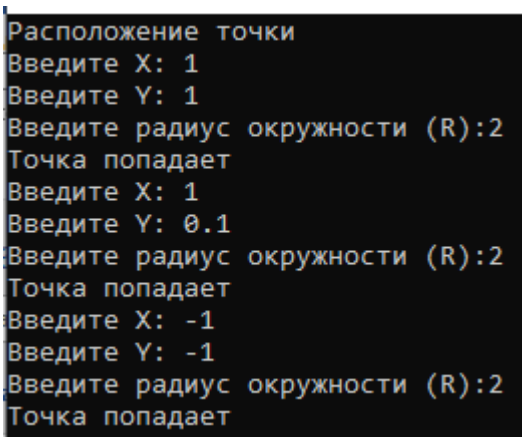

Рисунок 2.2 – Результат работы программы по разветвляющимся вычислительным процессам

3 Организация циклов

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз.

3.1 Описание программы

Вычислить и вывести на экран значения функции, заданной с помощью ряда Тейлора, на итервале от $x_{нач}$ *до* $x_{кон}$ *с шагом dx с точностью e.*

$$\operatorname{arctg} x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots, \quad |x| < 1.$$

3.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 3.1.

Листинг 3.1 – Текст файла prog3.cs

```
using System;

namespace Lab3
{
    class Lab3
    {
        static void Main()
        {
            double eps, x, xlim, dx;
            do
            {
                Console.WriteLine("Введите X0 : |X0| < 1");
            } while (!Double.TryParse(Console.ReadLine(), out x) || Math.Abs(x) >= 1);
            do
            {
                Console.WriteLine("Введите X : X0 < X < 1");
            } while (!Double.TryParse(Console.ReadLine(), out xlim) || xlim <= x);
            do
            {
                Console.WriteLine("Введите dx");
            } while (!Double.TryParse(Console.ReadLine(), out dx) || dx > Math.Abs(1 -
x));
            do
            {
                Console.WriteLine("Введите 0 < eps < 1");
            } while (!Double.TryParse(Console.ReadLine(), out eps) || eps >= 1 || eps <=
0);
            for (; x < xlim; x += dx)
            {
                double a = x;
```

```
                double s = 0;
                int n = 1;
                int iter = 0;
                for (; Math.Abs(Math.Atan(x) - s) > eps && iter < 1000; n++, iter++)
                {
                    if((x % 2) == 0)
                    {
                        s = s - a;
                    }
                    else
                    {
                        s = s + a;
                    }
                    a = (2 * n + 1) / (2 * n + 1);
                }
                if (s == 0) s = x;
                if (iter > 1000)
                {
                    Console.WriteLine("x = " + x + " ряд расходится (>1000 итераций)");
                    break;
                }

                Console.WriteLine("x = {0:f} сумма {1} вычислена с точностью {2} за {3}
итераци" + ((n % 10 == 1 && n / 10 != 1) ? "ю" : (n % 10 > 1 && n % 10 < 5 && n / 10 != 1)
? "и" : "й"), x, s, eps, n);
            }
            Console.ReadKey();
        }
    }
}
```

3.3 Тестирование программы

Результат работы программы приведен на рисунке 3.1.

```
Введите X0 : |X0| < 1
0.1
Введите X : X0 < X < 1
0.9
Введите dx
0.2
Введите 0 < eps < 1
0.01
x = 0.10 сумма 0.1 вычислена с точностью 0.01 за 2 итерации
x = 0.30 сумма 0.3 вычислена с точностью 0.01 за 2 итерации
x = 0.50 сумма 999.5 вычислена с точностью 0.01 за 1001 итерацию
x = 0.70 сумма 999.7 вычислена с точностью 0.01 за 1001 итерацию
x = 0.90 сумма 999.9 вычислена с точностью 0.01 за 1001 итерацию
```

Рисунок 3.1 – Результат работы программы по организации циклов

4 Простейшие классы

Класс является типом данных, определяемым пользователем. Он должен представлять собой одну логическую сущность, например, являться моделью реального объекта или процесса. Элементами класса являются данные и функции, предназначенные для их обработки.

4.1 Описание программы

Составить описание класса для представления времени. Предусмотреть возможности установки времени и изменения его отдельных полей (час, минута, секунда) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения. Создать методы изменения времени на заданное количество часов, минут и секунд. Написать программу, демонстрирующую все разработанные элементы класса.

4.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 4.1.

Листинг 4.1 – Текст файла prog4.cs

```
using System;

namespace Lab4
{
    class Lab4
    {
        class Time
        {
            DateTime time;

            public Time(int h, int m, int s)
            {
                if (!DateTime.TryParse(string.Format("{0}:{1}:{2}", h, m, s), out time))
throw new ArgumentException();
            }

            public void AddHour(int hour)
            {
                if (hour > 23 || hour < 0)
                    throw new HoursExcept();
                time = time.AddHours(hour);
            }

            public void AddMinutes(int minutes)
```

```
            {
                if (minutes > 59 || minutes < 0)
                    throw new MinutesExcept();
                time = time.AddMinutes(minutes);
            }

            public void AddSecond(int second)
            {
                if (second > 59 || second < 0)
                    throw new SecondsExcept();
                time = time.AddSeconds(second);
            }

            public void ShowInterface()
            {
                Console.WriteLine("1. Добавить часы");
                Console.WriteLine("2. Добавить минуты");
                Console.WriteLine("3. Добавить секунды");
            }

            public override string ToString()
            {
                return time.ToLongTimeString();
            }
        }

        class TimeExcepts : Exception
        {
            public override string Message
            {
                get
                {
                    return "Ошибка изменения значений времени.";
                }
            }
        }
        class HoursExcept : TimeExcepts
        {
            public override string Message
            {
                get
                {
                    return base.Message + " Вы ввели недопустимое значение для переменной
Часы.";
                }
            }
        }
        class MinutesExcept : TimeExcepts
        {
            public override string Message
            {
                get
                {
                    return base.Message + " Вы ввели недопустимое значение для переменной
Минуты.";
                }
            }
        }
        class SecondsExcept : TimeExcepts
        {
            public override string Message
            {
                get
                {
```

```
                    return base.Message + " Вы ввели недопустимое значение для переменной
Секунды.";
                }
            }
        }

        static void Main(string[] args)
        {
            Console.Write("Установка начального времени\n");
            Console.Write("Введите часы: ");
            int tempHour = Convert.ToInt16(Console.ReadLine());
            Console.Write("Введите минуты: ");
            int tempMinut = Convert.ToInt16(Console.ReadLine());
            Console.Write("Введите секунды: ");
            int tempSecond = Convert.ToInt16(Console.ReadLine());
            Time t = new Time(tempHour, tempMinut, tempSecond);

            while (true)
            {
                Console.WriteLine("Текущее время " + t.ToString());
                t.ShowInterface();
                try
                {
                    Console.Write("Введите номер пункта меню ");
                    switch (Convert.ToInt16(Console.ReadLine()))
                    {
                        case 1:
                            Console.Write("Количество часов: ");
                            t.AddHour(Convert.ToInt16(Console.ReadLine()));
                            break;
                        case 2:
                            Console.Write("Количество минут: ");
                            t.AddMinutes(Convert.ToInt16(Console.ReadLine()));
                            break;
                        case 3:
                            Console.Write("Количество секунд: ");
                            t.AddSecond(Convert.ToInt16(Console.ReadLine()));
                            break;
                        default:
                            break;
                    }
                }
                catch (TimeExcepts e)
                {
                    Console.WriteLine(e.Message);
                }
            }
        }
    }
}
```

4.3 Тестирование программы

Результат работы программы приведен на рисунке 4.1.

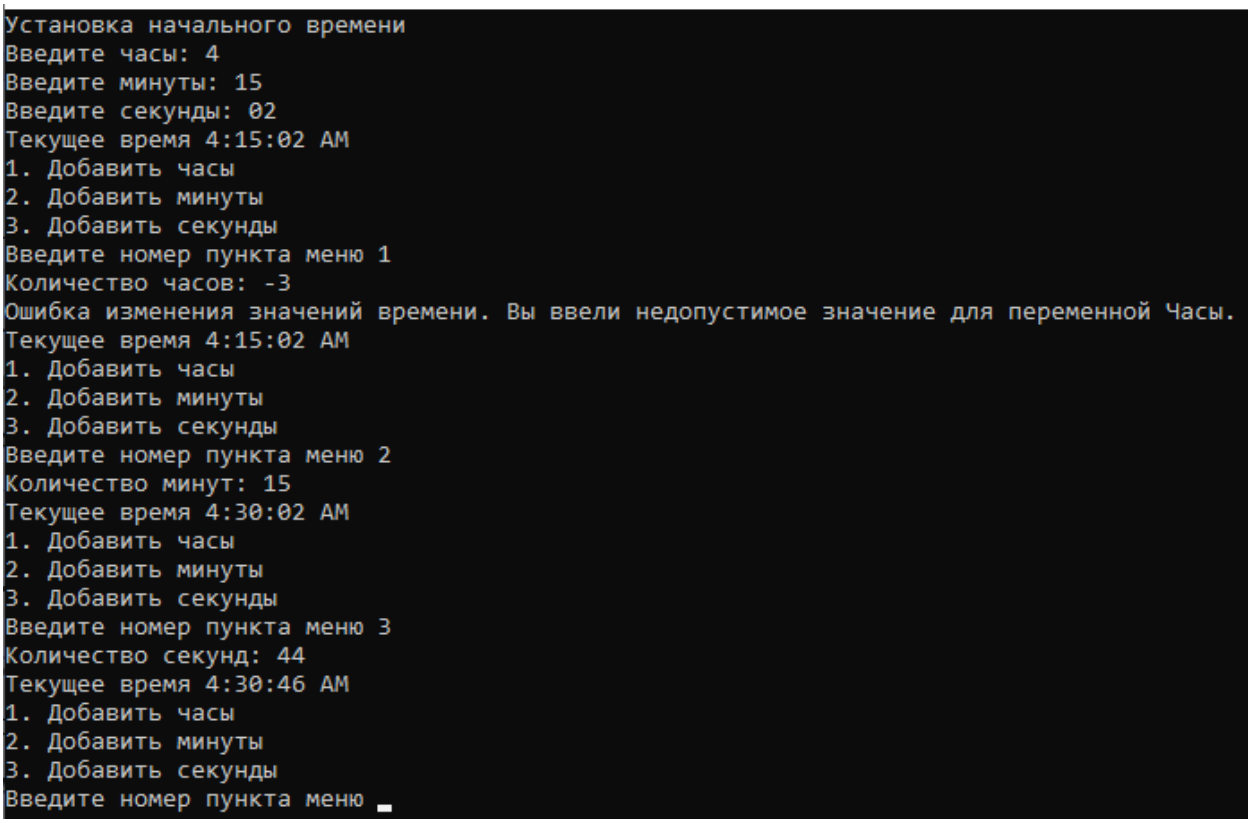

Рисунок 4.1 – Результат работы программы по простейшим классам

5 Одномерные массивы

До настоящего момента использовали в программах простые переменные. При этом каждой области памяти, выделенной для хранения одной величины, соответствует своё имя. Если переменных много, программа, предназначенная для их обработки, получается длинной и однообразной. Поэтому в любом процедурном языке есть понятие массива – ограниченной совокупности однотипных величин.

Элементы массива имеют одно и то же имя, а различаются порядковым номером (индексом). Это позволяет компактно записывать множество операций с помощью циклов.

5.1 Описание программы

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) максимальный по модулю элемент массива;

2) сумму элементов массива, расположенных между первым и вторым положительными элементами.

Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

5.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 5.1.

Листинг 5.1 – Текст файла prog5.cs

```
using System;

namespace Lab5
{
    class Lab5
    {
        static void Main(string[] args)
        {
            Console.Write("Введите количество элементов массива: ");
            int n = Convert.ToInt16(Console.ReadLine());
            int[] array = new int[n];
            Random rand = new Random();
            for (int i = 0; i < n; i++)
```

```
            {
                array[i] = rand.Next(0, 10) - rand.Next(0, 10);
                Console.Write(array[i].ToString() + " ");
            }
            Console.WriteLine();
            int max = Math.Abs(array[0]), sum = 0;
            bool one = false, two = false;
            for (int i = 0; i < n; i++)
            {
                if (max < Math.Abs(array[i]))
                    max = Math.Abs(array[i]);
                if (one && !two)
                    sum += array[i];
                if (array[i] > 0 && !two && !one)
                    one = true;
                else if (array[i] > 0 && one && !two)
                {
                    sum -= array[i];
                    two = true;
                }
            }
            Console.WriteLine("Максимальный по модулю элемент массива: " + max);
            Console.WriteLine("Сумма элементов массива, расположенных между первым и вторым
положительными элементами: " + sum);

            Console.WriteLine("Преобразование массива таким образом, чтобы элементы, равные
нулю, располагались после всех остальных...");

            for (int i = array.Length-1; i > 0; i--)
            {
                for (int j = i; j > 0; j--)
                {
                    if (array[j] == 0)
                    {
                        int temp = array[i];
                        array[i] = array[j];
                        array[j] = temp;
                        break;
                    }
                }
            }

            Console.Write("Отсортированный массив: ");
            for (int i = 0; i < array.Length; i++)
                Console.Write(array[i] + " ");

            Console.ReadKey();
        }
    }
}
```

5.3 Тестирование программы

Результат работы программы приведен на рисунке 5.1.

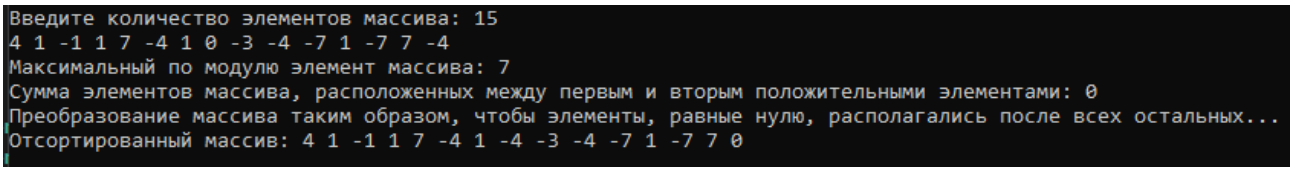

Рисунок 5.1 – Результат работы программы по одномерным массивам

6 Двумерные массивы

Двумерный массив - это одномерный массив, элементами которого являются одномерные массивы. Другими словами, это набор однотипных данных, имеющий общее имя, доступ к элементам которого осуществляется по двум индексам.

6.1 Описание программы

Соседями элемента в матрице Aij назовем элементы Akl где $i-1 \le k \le i+1$, $j-1 \le l \le j+1, (k,l) \ne (i,j)$. *Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 10 x 10.*

В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.

6.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 6.1.

Листинг 6.1 – Текст файла prog6.cs

```
using System;

namespace Lab6
{
    class Lab6
    {
        static void Main(string[] args)
        {
            Random ranf = new Random();
            double[,] mas = new double[10, 10];

            Console.WriteLine("Исходная матрица");
            // заполнение массива случайными числами в пределах 10-100
            for (int i = 0; i < mas.GetLength(0); i++)
            {
                for (int j = 0; j < mas.GetLength(1); j++)
                {
                    mas[i, j] = ranf.Next(10, 100);
                    Console.Write(" {0}", mas[i, j]);
```

```
                }
                Console.WriteLine();
            }

            double[,] newmas = new double[mas.GetLength(0), mas.GetLength(1)];
            double sum = 0;


            //  создание нового массива "newmas", с исходного массива
            //  "mas" путем "сглаживаниия"
            Console.WriteLine("--------------------------------");
            Console.WriteLine("Новая матрица, созданная с исходного массива путём
\"Сглаживания\"");
            for (int i = 0; i < mas.GetLength(0); i++)
            {
                for (int j = 0; j < mas.GetLength(1); j++)
                {
                    if (j == 0 || j == 9)
                        newmas[i, j] = mas[i, j];

                    else newmas[i, j] = (mas[i, j - 1] + mas[i, j + 1]) / 2;

                    Console.Write(" {0}", newmas[i, j]);
                }
                Console.WriteLine();
            }
            // поиск элементов под главной диагональю матрицы и
            // подсчета их суммы
            Console.WriteLine("--------------------------------");
            for (int i = 1; i < mas.GetLength(0); i++)
            {
                for (int j = 1; j < mas.GetLength(1); j++)
                {
                    if (i == j)
                        sum += Math.Abs(newmas[i, j - 1]);
                }
            }
            Console.WriteLine("Сумма модулей элементов, расположенных ниже главной диагонали
в \"сглаженной\" матрице = {0}", sum);
            Console.ReadLine();
        }
    }
}
```

6.3 Тестирование программы

Результат работы программы приведен на рисунке 6.1.

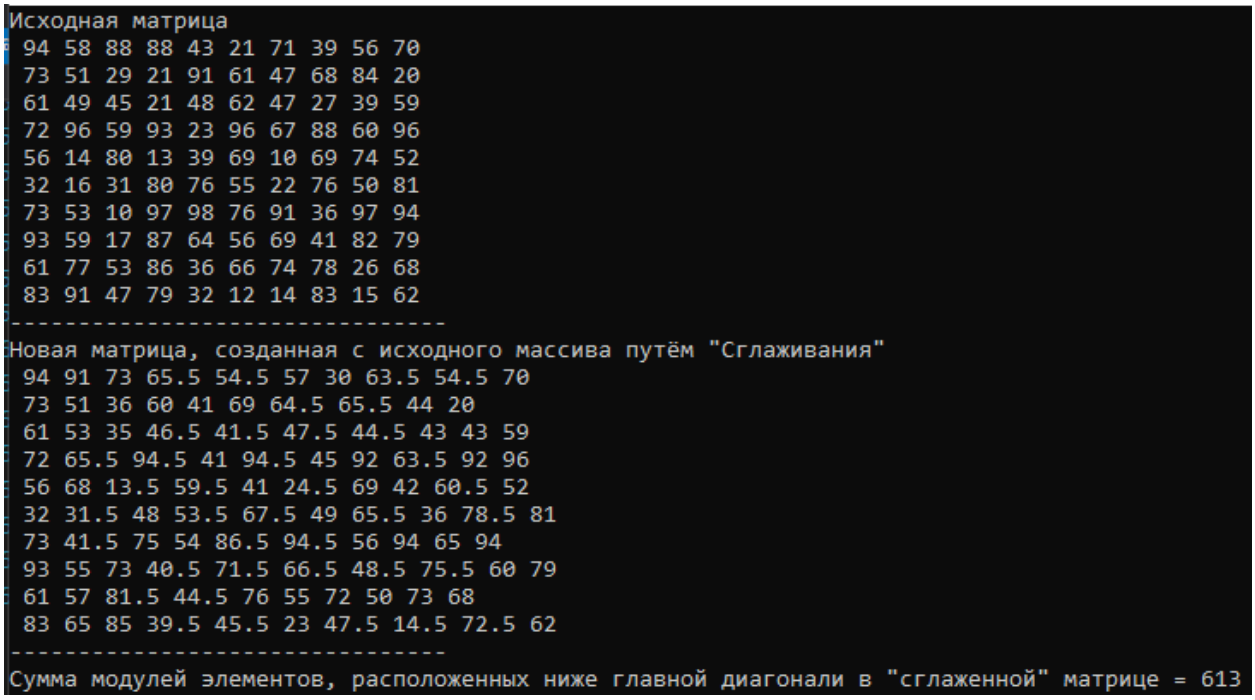

Рисунок 6.1 – Результат работы программы по двумерным массивам

7 Строки

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый класс System.String библиотеки .NET.

Несмотря на то что строки являются ссылочным типом данных, на равенство и неравенство проверяются не ссылки, а значения строк. Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.

7.1 Описание программы

Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

7.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 7.1.

Листинг 7.1 – Текст файла prog7.cs

```
using System;
using System.IO;

namespace Lab7
{
    class Lab7
    {
        static void Main(string[] args)
        {
            Console.Write("Введите путь к файлу (перетащите файл в консоль для
автоматического определения пути): ");
            string[] text = File.ReadAllText(Console.ReadLine()).Split('.', '?', '!');
            Console.Write("Количество слов: ");
            int Count = int.Parse(Console.ReadLine());
            for (int i = 0; i < text.Length; i++)
            {
                text[i] = text[i].Trim();
                string[] words = text[i].Split(' ');
                if (words.Length == Count)
                    Console.WriteLine(text[i]);
            }
            Console.ReadKey(true);
        }
    }
}
```

7.3 Тестирование программы

Результат работы программы при считывании текста из файла *txt (рисунок 7.1) приведён на рисунке 7.2.

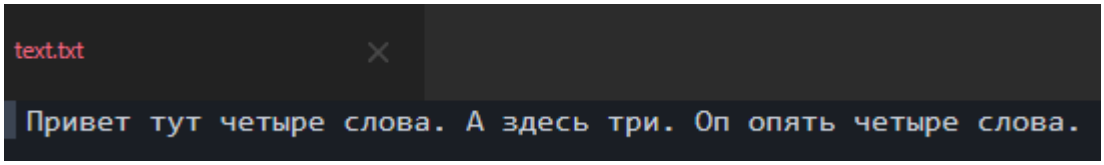

Рисунок 7.1 – Текст из файла *txt

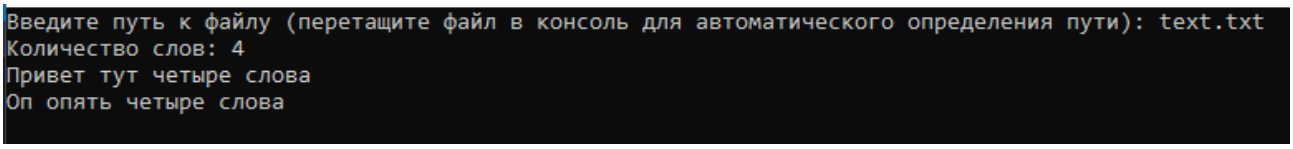

Рисунок 7.2 - Результат работы программы

8 Классы и операции

C# позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные функции. Это даёт возможность применять экземпляры собственных типов данных в составе выражений таким же образом, как стандартных. Определение собственных операций класса часто называют перегрузкой операций.

В C# существуют три вида операций класса: унарные, бинарные и операции преобразования типа.

8.1 Описание программы

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен релизовать следующие операции над матрицами:

- сложение, вычитание (как с другой матрицей, так и с числом);

- комбинированные операции присваивания (+=, -=);

- операции сравнения на равенство / неравенство;

- операции вычисления обратной и транспонированной матрицы;

- доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса.

8.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 8.1.

Листинг 8.1 – Текст файла prog8.cs

```
using System;

namespace Lab8
{
    class Lab8
    {
        public class Matrix
        {
            double[,] matrix;
            int Row = 0, Col = 0;
```

```
            //матрица N x M
            public Matrix(int row, int col)
            {
                matrix = new double[row, col];
                Row = row; Col = col;
            }

            //копия матрицы
            public Matrix(Matrix prMatrix)
            {
                this.matrix = new double[prMatrix.matrix.GetLength(0),
prMatrix.matrix.GetLength(1)];
                Row = prMatrix.matrix.GetLength(0); Col = prMatrix.matrix.GetLength(1);
                for (int i = 0; i < Row; i++)
                    for (int j = 0; j < Col; j++)
                    {
                        matrix[i, j] = prMatrix.matrix[i, j];
                    }
            }

            //квадратная матрица
            public Matrix(int N)
            {
                matrix = new double[N, N];
                Row = Col = N;
            }

            //перегрузка индексатора, чтобы обратится к
            //элементу матрицы как к элементу двумерного массива
            public double this[int i, int j]
            {
                get { return matrix[i, j]; }
                set { matrix[i, j] = value; }
            }

            //перегружаем сложение матриц
            public static Matrix operator +(Matrix first, Matrix second)
            {
                Matrix mat = new Matrix(first.Row, first.Col);
                for (int i = 0; i < first.Row; i++)
                    for (int j = 0; j < first.Col; j++)
                        mat[i, j] = first[i, j] + second[i, j];
                return mat;
            }

            //перегружаем вычитание матриц
            public static Matrix operator -(Matrix first, Matrix second)
            {
                Matrix mat = new Matrix(first.Row, first.Col);
                for (int i = 0; i < first.Row; i++)
                    for (int j = 0; j < first.Col; j++)
                        mat[i, j] = first[i, j] - second[i, j];
                return mat;
            }

            // перегрузка сравнения матриц
            public static bool operator ==(Matrix first, Matrix second)
            {
                for (int i = 0; i < first.Row; i++)
                {
                    for (int j = 0; j < second.Col; j++)
                    {
                        if (first[i, j] != second[i, j])
                            return false;
```

```
                }
            }
            return true;
        }
        public static bool operator !=(Matrix first, Matrix second)
        {
            for (int i = 0; i < first.Row; i++)
            {
                for (int j = 0; j < second.Col; j++)
                {
                    if (first[i, j] != first[i, j])
                        return true;
                }
            }
            return false;
        }

        //умножение на число
        public static Matrix operator *(Matrix m, int t)
        {
            Matrix mat = new Matrix(m.Row, m.Col);
            for (int i = 0; i < m.Row; i++)
                for (int j = 0; j < m.Col; j++)
                    mat[i, j] = m[i, j] * t;
            return mat;
        }

        //распечатать матрицу
        public void PrintMatrix()
        {
            for (int i = 0; i < this.Row; i++)
            {
                for (int j = 0; j < this.Col; j++)
                    Console.Write("{0}  ", this[i, j]);
                Console.Write("\n");
            }

        }

        //произведение матриц
        public static Matrix operator *(Matrix first, Matrix second)
        {
            Matrix matr = new Matrix(first.Row, first.Col);
            for (int i = 0; i < first.Row; i++)
            {
                for (int j = 0; j < second.Col; j++)
                {
                    double sum = 0;
                    for (int r = 0; r < first.Col; r++)
                        sum += first[i, r] * second[r, j];
                    matr[i, j] = sum;
                }
            }
            return matr;
        }

        //возведение в степень
        public static Matrix operator ^(Matrix first, int pow)
        {
            Matrix matr = new Matrix(first.Row, first.Col);
            matr = first;
            for (int z = 1; z < pow; z++)
            {
                Matrix bufer = new Matrix(first.Row, first.Col);
```

```
                    for (int i = 0; i < first.Row; i++)
                    {
                        for (int j = 0; j < first.Row; j++)
                        {
                            double sum = 0;
                            for (int r = 0; r < first.Row; r++)
                                sum += matr[i, r] * first[r, j];
                            bufer[i, j] = sum;
                        }
                    }
                    matr = bufer;
                }
                return matr;
            }
            // транспонирование матрицы
            public void Transp()
            {
                double tmp;
                for (int i = 0; i < matrix.GetLength(0); i++)
                {
                    for (int j = 0; j < i; j++)
                    {
                        tmp = matrix[i, j];
                        matrix[i, j] = matrix[j, i];
                        matrix[j, i] = tmp;
                    }
                }
            }

            // вычисление обратной матрицы
            public static Matrix Inverse(Matrix mA, uint round = 0)
            {
                Matrix tempMatrix = new Matrix(mA); //Делаем копию исходной матрицы
                double determinant = DetRec(mA.matrix); //Находим детерминант

                if (determinant == 0) return tempMatrix; //Если определитель == 0 - матрица
вырожденная

                for (int i = 0; i < mA.matrix.GetLength(0); i++)
                {
                    for (int t = 0; t < mA.matrix.GetLength(1); t++)
                    {
                        Matrix tmp = Exclude(mA, i, t);  //получаем матрицу без строки i и
столбца t
                                                         //(1 / determinant) *
Determinant(tmp) - формула пределения элемента обратной матрицы
                        tempMatrix.matrix[t, i] = round == 0 ? (1 / determinant) *
Math.Pow(-1, i + t) * DetRec(mA.matrix) : Math.Round(((1 / determinant) * Math.Pow(-1, i +
t) * DetRec(mA.matrix)), (int)round, MidpointRounding.ToEven);
                    }
                }
                return tempMatrix;
            }

            private static Matrix Exclude(Matrix mA, int row, int col)
            {
                Matrix temp = new Matrix(mA.Row - 1, mA.Col - 1);

                bool setMinusRow = false;
                bool setMinusCol = false;
                for (int i = 0; i < temp.matrix.GetLength(0); i++)
                {
                    if (i == row)
                    {
```

```
                        setMinusRow = true;
                        continue;
                    }
                    for (int j = 0; j < temp.matrix.GetLength(1); j++)
                    {
                        if (j == col)
                        {
                            setMinusCol = true;
                            continue;
                        }

                        if (setMinusCol == true && setMinusRow == true)
                        {
                            temp.matrix[i, j] = mA.matrix[i, j];
                        }
                        else if (setMinusCol == false && setMinusRow == true)
                        {
                            temp.matrix[i, j] = mA.matrix[i, j];
                        }
                        else if (setMinusCol == true && setMinusRow == false)
                        {
                            temp.matrix[i,j] = mA.matrix[i, j];
                        } else temp.matrix[i, j] = mA.matrix[i, j];
                    }
                }
                return mA;
            }

            private static double DetRec(double[,] matrix)
            {
                if (matrix.Length == 4)
                {
                    return matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0];
                }
                double sign = 1, result = 0;
                for (int i = 0; i < matrix.GetLength(1); i++)
                {
                    double[,] minor = GetMinor(matrix, i);
                    result += sign * matrix[0, i] * DetRec(minor);
                    sign = -sign;
                }
                return result;
            }

            private static double[,] GetMinor(double[,] matrix, int n)
            {
                double[,] result = new double[matrix.GetLength(0) - 1, matrix.GetLength(0) -
1];
                for (int i = 1; i < matrix.GetLength(0); i++)
                {
                    for (int j = 0, col = 0; j < matrix.GetLength(1); j++)
                    {
                        if (j == n)
                            continue;
                        result[i - 1, col] = matrix[i, j];
                        col++;
                    }
                }
                return result;
            }
        }

        static void Main(string[] args)
        {
```

```
            //размерность
            int N = 3;
            //степень
            int pow = 3;

            Random rand = new Random();
            Matrix first = new Matrix(N);
            Matrix second = new Matrix(N);

            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                {
                    first[i, j] = rand.Next(1, 4);
                    second[i, j] = rand.Next(1, 4);
                }

            Console.WriteLine("Первая матрица:");
            first.PrintMatrix();
            Console.WriteLine("\nВторая матрица:");
            second.PrintMatrix();
            Console.WriteLine("\nСумма матриц:");
            (first + second).PrintMatrix();
            Console.WriteLine("\nРазница матриц:");
            (first - second).PrintMatrix();
            Console.WriteLine("\nКомбинированная операция присваивания второй матрице к
первой");
            second += first;
            Console.WriteLine("Первая матрица:");
            first.PrintMatrix();
            Console.WriteLine("\nВторая матрица:");
            second.PrintMatrix();
            Console.WriteLine("\nСравнение матриц:");
            if (first == second)
            {
                Console.WriteLine("Матрицы равны");
            }
            else Console.WriteLine("Матрицы не равны");
            Console.WriteLine("\nПолучение элемента первой матрицы по индексу [1][1]:");
            Console.WriteLine(first[1, 1]);
            Console.WriteLine("\nТранспонирование второй матрицы:");
            second.Transp();
            second.PrintMatrix();
            Console.WriteLine("\nНахождение обратной матрицы из второй матрицы:");
            second = Matrix.Inverse(second);
            second.PrintMatrix();
            Console.ReadKey();
        }
    }
}
```

8.3 Тестирование программы

Результат работы программы приведен на рисунке 8.1.

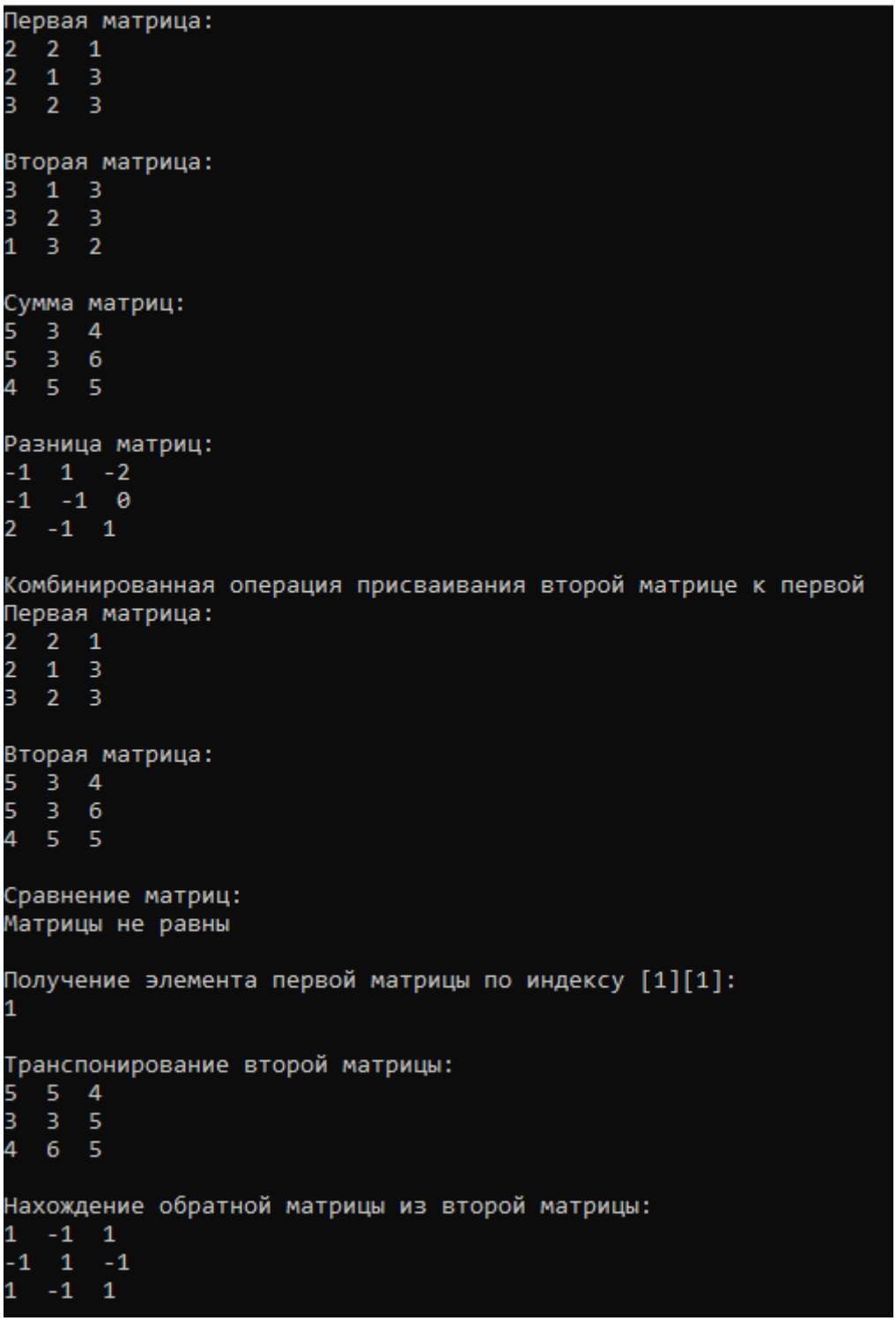

Рисунок 8.1 - Результат работы программы по классам и операциям

9 Наследование

Управлять большим количеством разрозненных классов довольно сложно. С этой проблемой можно справиться путём упорядочивания и ранжирования классов, то есть объединяя общие для нескольких классов свойства в одном классе и используя его в качестве базового.

Эту возможность предоставляет механизм наследования, который является мощнейшим инструментом ООП. Он позволяет строить иерархии, в которых классы-потомки получают свойства классов-предков и могут дополнять их или заменять.

9.1 Описание программы

Описать базовый класс «Строка».

Обязательные поля класса:

- поле для хранения символов строки;
- значение типа word для хранения длины строки в байтах.

Реализовать обязательные методы следующего назначения:

- конструктор без параметров;
- конструктор,принимающий в качестве параметра символ;
- конструктор,принимающий в качестве параметра строковый литерал;
- метод получения длины строки;
- метод очистки строки (сделать строку пустой);

Описать производный от класса «Строка» класс «Битовая_Строка». Строки данного класса могут содержать только символы '0' или '1'. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, класс «Битовая_строка» принимает нулевое значение. Содержимое данных строк рассматривается как двоичное число. Отрицательные числа хранятся в дополнительном коде.

Для класса «Битовая_строка» описать следующие методы:

- конструктор, принимающий в качестве параметра строковый литерал;

- деструктор;

- изменение знака на противоположный (перевод числа в дополнительный код);

- присваивание;

- вычисление арифметической суммы строк;

- проверка на равенство;

В случае необходимости более коротка битовая строка расширяется влево знаковым разрядом.

9.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 8.1.

Листинг 8.1 – Текст файла prog9.cs

```
using System;
namespace Lab9
{
    class Lab9
    {
        private class String
        {
            private int _length;
            private string _str;

            // конструктор без параметров
            public String()
            {
            }

            // конструктор, принимающий в качестве параметра строковый литерал
            public String(string str)
            {
                _str = str;
                _length = str.Length;
            }

            // конструктор, принимающий в качестве параметра символ;
            public String(char ch)
            {
                _str = Convert.ToString(ch);
                _length = 1;
            }

            // Метод возвращающий длинну строки.
            public int GetLength()
            {
                return _length;
            }

            // Метод очищающий стоку.
```

```
            public void Clear()
            {
                _str = "";
                _length = 0;
            }

            public override string ToString()
            {
                return _str;
            }
        }

        private class BitString : String
        {
            public bool znak;
            private string _str;
            private int _length;

            // конструктор, принимающий в качестве параметра строковый литерал
            public BitString(string str)
            {
                    _str = str;
                    _length = str.Length;
            }
            // деструктор
            ~BitString()
            {
            }

            public override string ToString()
            {
                return _str;
            }

            public static BitString operator +(BitString m1, BitString m2)
            {
                BitString str = new BitString("0000000000000000000");
                char[] a = str._str.ToCharArray();
                for (int i = m1._str.Length - 1; i >= 0; i--)
                    a[i] = Convert.ToString(Convert.ToInt32(Convert.ToString(m1._str[i])) +
Convert.ToInt32(Convert.ToString(m2._str[i])))[0];
                for (int i = m1._str.Length - 1; i > 0; i--)
                {

                    if (a[i] == '2')
                    {
                        a[i - 1] = Convert.ToString(Convert.ToInt32(Convert.ToString(a[i -
1])) + 1)[0];
                        a[i] = '0';
                    }
                    if (a[i] == '3')
                    {
                        a[i - 1] = Convert.ToString(Convert.ToInt32(Convert.ToString(a[i -
1])) + 1)[0];
                        a[i] = '1';
                    }
                }
                string g = "";
                for (int i = 0; i < a.Length; i++)
                    g += a[i];
                str._str = g;
                return str;
            }
            public static bool operator ==(BitString m1, BitString m2)
```

```
        {
            bool x;
            if (m1._str == m2._str)
                x = true;
            else x = false;
            return x;
        }
        public static bool operator !=(BitString m1, BitString m2)
        {
            bool x;
            if (m1._str != m2._str)
                x = true;
            else x = false;
            return x;
        }
        public static BitString Dop_kod(BitString m1)
        {
            char[] a = m1._str.ToCharArray();
            if (m1.znak == false)
            {
                for (int i = a.Length - 1; i >= 0; i--)
                {
                    if (a[i] == '0')
                        a[i] = '1';
                    else
                        a[i] = '0';
                }
                a[0] = Convert.ToChar(Convert.ToInt32(a[a.Length - 1]) + 1);
                for (int i = a.Length - 1; i >= 0; i--)
                {
                    if (a[i] == '2')
                    {
                        a[i - 1] = Convert.ToChar(Convert.ToInt32(a[i]) + 1);
                        a[i] = '0';
                    }
                    if (a[i] == '3')
                    {
                        a[i - 1] = Convert.ToChar(Convert.ToInt32(a[i]) + 1);
                        a[i] = '1';
                    }
                }


                m1.znak = true;
            }
            string g = "";
            for (int i = 0; i < a.Length; i++)
                g += a[i];
            m1._str = g;
            return m1;
        }
        public static BitString Prisvaivanie(string str)
        {
            BitString m1 = new BitString(str);
            return m1;
        }
    }

    static void Main(string[] args)
    {
        BitString m1, m2, m3;
        m1 = BitString.Prisvaivanie("0000000000000110001");
        Console.WriteLine("Битовая строка 1: " + m1.ToString());
        m2 = BitString.Prisvaivanie("0000000000000011001");
```

```
            Console.WriteLine("Битовая строка 2: " + m2.ToString());

            Console.Write("\n");
            if (m2 == m1)
                Console.WriteLine("Строки 1 и 2 равны");
            else
                Console.WriteLine("Строки 1 и 2 не равны");

            Console.Write("\n");
            Console.WriteLine("Строка 1 меняет знак на противоположный...");
            m1 = BitString.Dop_kod(m1);
            Console.WriteLine("Строка 1: " + m1.ToString());

            Console.Write("\n");
            m3 = m1 + m2;
            Console.WriteLine("Складываем 1 и 2 битовые стоки... Ответ: " + m3.ToString());

            Console.ReadLine();
        }
    }
}
```

9.1 Тестирование программы

Результат работы программы приведен на рисунке 9.1.

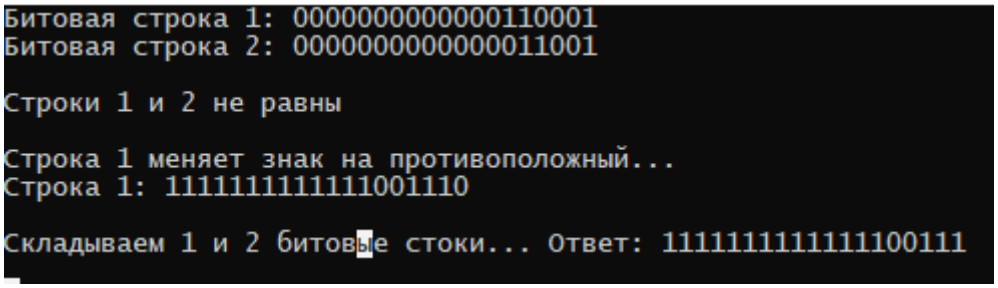

Рисунок 9.1 - Результат работы программы по классу «Битовая строка»

10 Структуры

Структура – тип данных, аналогичный классу, но имеющий ряд важных отличий от него:

- структура является значимым, а не ссылочным типом данных, то есть экземпляр структуры хранит значения своих элементов, а не ссылки на них, и располагается в стеке, а не в хипе;
- структура не может участвовать в иерархиях наследования, она может только реализовывать интерфейсы;
- в структуре запрещено определять конструктор по умолчанию, поскольку он определен неявно и присваивает всем её элементам значения по умолчанию;
- в структуре запрещено определять деструкторы, поскольку это бессмысленно.

10.1 Описание программы

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;

- номер поезда;

- время отправления.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN (записи должны быть упорядочены по номерам поездов);

- вывод на экран информации о поезде, номер которого введён с клавиаткры (если таких поездов нет, вывести соответствующее сообщение).

10.2 Текст программы

Проект состоит из одного файла исходного кода, который приведен в листинге 10.1.

Листинг 10.1 – Текст файла prog10.cs

```
using System;
using System.Collections.Generic;

namespace Lab10
{
    public struct Train
    {
        string name;
        public int number;
        public string date;

        public Train(string name, int number, string date)
        {
            this.name = name;
            this.number = number;
            this.date = date;
        }

        public static int Compare(Train first, Train second)
        {
            return first.number.CompareTo(second.number);
        }

        public override string ToString()
        {
            return String.Format("Название пункта назначения " + this.name + " \nНомер
поезда " + number + " \nВремя отправления " + date);
        }
    }

    class Lab10
    {
        static void Main()
        {
            List<Train> nL = new List<Train>();

            for (int i = 0; i < 3; i++)
            {
                Console.Write("Введите название пункта назначения: ");
                string tempName = Console.ReadLine();
                Console.Write("Введите номер поезда: ");
                int tempNumber = Convert.ToInt32(Console.ReadLine());
                Console.Write("Введите время отправления: ");
                string tempTime = Console.ReadLine();
                nL.Add(new Train(tempName, tempNumber, tempTime));
                Console.WriteLine();
            }

            foreach (Train c in nL)
            {
                Console.WriteLine(c.ToString());
            }

            Console.WriteLine("\nСортируем по номерам поездов");
```

```
            nL.Sort(Train.Compare);

            foreach (Train c in nL)
            {
                Console.WriteLine(c.ToString());
            }

            Console.WriteLine("Вывод на экран информации о поездах, номер которого введен с
клавиатуры");
            Console.Write("Введите номер поезда ");
            int number = int.Parse(Console.ReadLine());
            foreach (var item in nL)
            {
                if (item.number == number)
                    Console.WriteLine(item.ToString());
            }
            Console.ReadLine();
        }
    }
}
```

10.3 Тестирование программы

Результат работы программы приведен на рисунке 10.1.

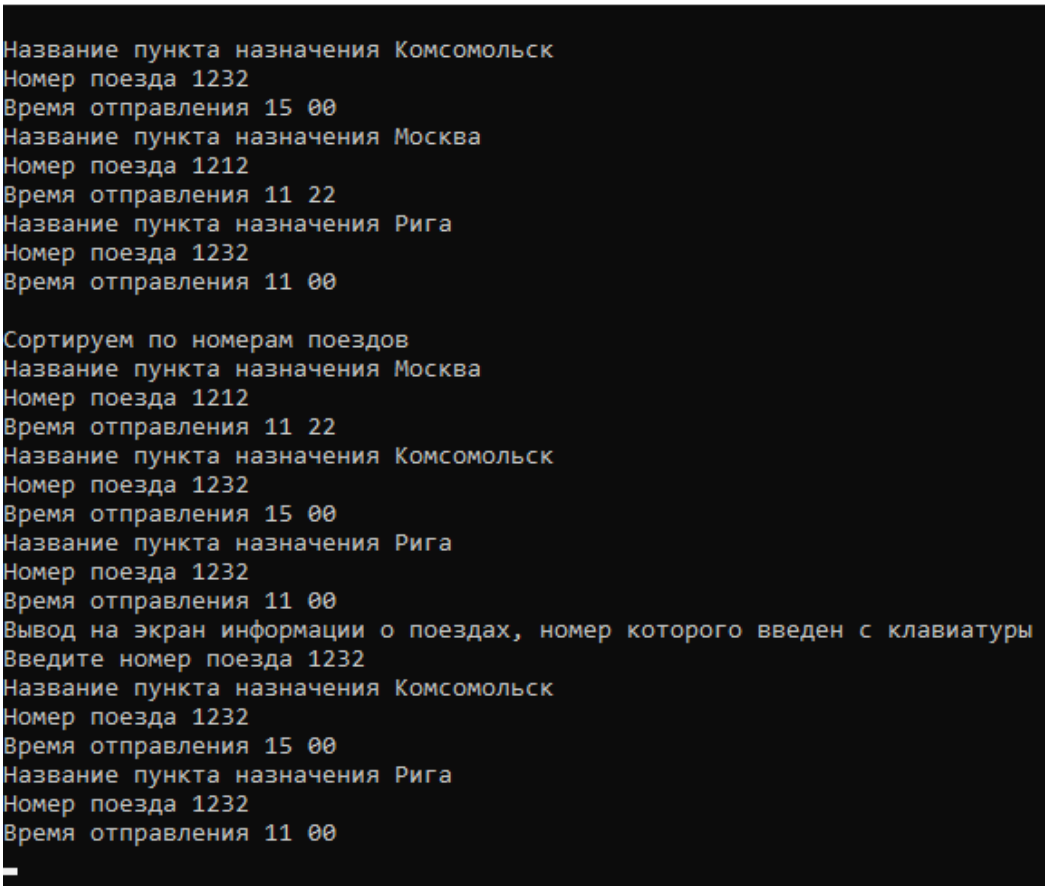

Рисунок 10.1 - Результат работы программы по структурам

Заключение

В ходе изучения дисциплины «Технологии программирования» по изучению языка программирования C# были рассмотрены такие темы как:

1 Линейные программы.

2 Разветвляющиеся вычислительные процессы.

3 Организация циклов.

4 Простейшие классы.

5 Одномерные массивы.

6 Двумерные массивы.

7 Строки.

8 Классы и операции.

9 Наследование.

10 Структуры.

Полученные навыки и знания будут использоваться в дальнейших проектах.

Список использованных источников

1 Павловская Т. А., C# Программирование на языке высокого уровня: Практикум. — СПб.: Питер, 2009. — 432 с.: ил. — (Серия «Учебное пособие»).