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 2

Студент группы 6ИСб-1 И. Нозимзода

Преподаватель С.Ю. Александров

2018

Содержание

[Содержание 2](#_Toc533446795)

[Задания 4](#_Toc533446796)

[Введение 5](#_Toc533446797)

[1 Линейные программы 6](#_Toc533446798)

[1.1 Описание программы 6](#_Toc533446799)

[1.2 Текст программы 6](#_Toc533446800)

[1.3 Тестирование программы 7](#_Toc533446801)

[2 Разветвляющиеся вычислительные процессы 8](#_Toc533446802)

[2.1 Описание программы 8](#_Toc533446803)

[2.2 Текст программы 8](#_Toc533446804)

[2.3 Тестирование программы 10](#_Toc533446805)

[3 Организация циклов 11](#_Toc533446806)

[3.1 Описание программы 11](#_Toc533446807)

[3.2 Текст программы 11](#_Toc533446808)

[3.3 Тестирование программы 13](#_Toc533446809)

[4 Простейшие классы 14](#_Toc533446810)

[4.1 Описание программы 14](#_Toc533446811)

[4.2 Текст программы 14](#_Toc533446812)

[4.3 Тестирование программы 16](#_Toc533446813)

[5 Одномерные массивы 18](#_Toc533446814)

[5.1 Описание программы 18](#_Toc533446815)

[5.2 Текст программы 18](#_Toc533446816)

[5.3 Тестирование программы 20](#_Toc533446817)

[6 Двумерные массивы 21](#_Toc533446818)

[6.1 Описание программы 21](#_Toc533446819)

[6.2 Текст программы 21](#_Toc533446820)

[6.3 Тестирование программы 23](#_Toc533446821)

[7 Строки 24](#_Toc533446822)

[7.1 Описание программы 24](#_Toc533446823)

[7.2 Текст программы 24](#_Toc533446824)

[7.3 Тестирование программы 25](#_Toc533446825)

[8 Классы и операции 26](#_Toc533446826)

[8.1 Описание программы 26](#_Toc533446827)

[8.2 Текст программы 26](#_Toc533446828)

[8.3 Тестирование программы 29](#_Toc533446829)

[9 Наследование 31](#_Toc533446830)

[9.1 Описание программы 31](#_Toc533446831)

[9.2 Текст программы 31](#_Toc533446832)

[9.3 Тестирование программы 34](#_Toc533446833)

[10 Структуры 35](#_Toc533446834)

[10.1 Описание программы 35](#_Toc533446835)

[10.2 Текст программы 36](#_Toc533446836)

[10.3 Тестирование программы 37](#_Toc533446837)

[Заключение 38](#_Toc533446838)

[Список использованных источников 39](#_Toc533446839)

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

# Линейные программы

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

## Описание программы

*Написать программу для расчета по двум формулам*



## Текст программы

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

Листинг . – Текст файла prog1.cs

|  |
| --- |
| using System;  namespace Lab1  {  class Lab1  {  static void Main(string[] args)  {  Console.WriteLine("Введите аргумент к функции (x):");  double x = double.Parse(Console.ReadLine());  Task01\_1 t1 = new Task01\_1();  Console.WriteLine("Для аргумента x = {0} функции Z1 и Z2 равны:\n\nZ1(x) = {1}\n Z2(x) ={2}\n", x, t1.Z1(x), t1.Z2(x));  }  }  class Task01\_1  {  public double Z1(double x)  {  return Math.Pow(Math.Cos(3.0 / 8.0 \* Math.PI - x / 4.0), 2.0) -  Math.Pow(Math.Cos(11.0 / 8.0 \* Math.PI + x / 4.0), 2.0); ;  }  public double Z2(double x)  {  return Math.Sqrt(2.0) \* Math.Sin(x / 2.0) / 2; ;  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 1.1.

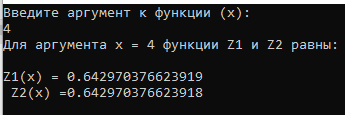


Рисунок . – Результат работы линейной программы

# Разветвляющиеся вычислительные процессы

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

## 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.*

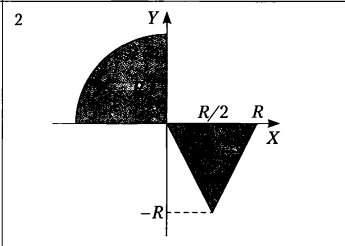


Рисунок 2.1 – График

## Текст программы

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

Листинг . – Текст файла prog2.cs

|  |
| --- |
| using System;  namespace Lab2  {  class Lab2  {  static void Main(string[] args)  {  Console.WriteLine("Задание 1.\n Введите аргумент к функции (x):");  double x = double.Parse(Console.ReadLine());  double y = double.Parse(Console.ReadLine());  Task02\_1 func = new Task02\_1();  Console.WriteLine("\n Для аргумента х = {0} функция f равна: f(x) = {1}\n\nЗадание 2.\n Введите координаты (a;b), ограничивающие область попадания:\n", x, func.F(x));  Console.WriteLine("\n Введите радиус окружности (R):\n");  double R = double.Parse(Console.ReadLine());  Console.WriteLine("\n Введите координаты точки попадания в область (х;у):\n");  Task02\_2 obl = new Task02\_2();  Console.WriteLine("\n Для координат {0} и {2} точка в область {1}", x,obl.O(x, y, R),y);  Console.ReadLine();  }  }  class Task02\_1  {  public double F(double x)  {  double y = 0;  if (x < -7 || x > 11) { /\*Console.WriteLine("Функция не определена при данном аргументе");\*/ return double.NaN; }  else  {  if (x < -7) y = 0;  if (x >= -7 && x < -3) y = 3;  if (x >= -3 && x < 3) y = -Math.Sqrt(9 - x \* x) + 3;  if (x >= 3 && x < 6) y = -2 \* x + 9;  if (x >= 6 && x < 11) y = x - 9;  if (x >= 11) y = 2;  return y;  }  }  }  class Task02\_2  {  public string O(double x, double y, double R)  {  string flag = "не попадает";  if ((((x + R) \* (x + R) + (y - R) \* (y - R)) <= (R \* R)) || ((x >= -1) && (x <= 2 \* R) && (y >= -R) && (y <= 0))) flag = "попадает";  return flag;  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 2.2.

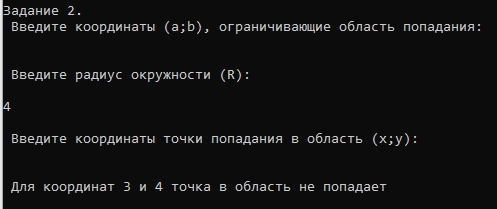


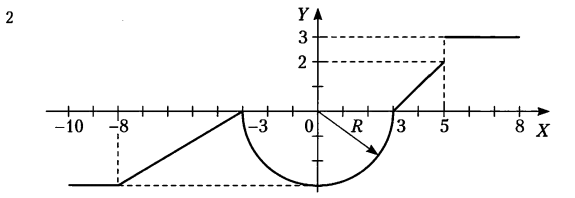
Рисунок .2 – Результат работы программы по разветвляющимся вычислительным процессам

# Организация циклов

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

## Описание программы

*Вычислить и вывести на экран значения функции, заданной* графически*, на итервале от до с шагом dx.*



## Текст программы

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

Листинг 3. – Текст файла prog3.cs

|  |
| --- |
| using System;  namespace Lab3  {  class Lab3  {  static void Main(string[] args)  {  Console.WriteLine("Задание 1.\n Введите начальное х(xn), конечное х(xk) и шаг dx:\n");  double xn = double.Parse(Console.ReadLine());  double xk = double.Parse(Console.ReadLine());  double dx = double.Parse(Console.ReadLine());  Task03\_1 func = new Task03\_1();  Console.WriteLine(" Таблица Значений Функции");  Console.WriteLine("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine("| x | f(x) |" +  "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  double x = xn;  while (x <= xk)  {  Console.WriteLine(" {0} {1} ", x, func.One(x));  x += dx;  }  Console.WriteLine("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  ///Задание 2  Console.WriteLine("\n\nЗадание 2.\n Введите координаты (x;y), ограничивающие область попадания:\n");  double X = double.Parse(Console.ReadLine());  double Y = double.Parse(Console.ReadLine());  Console.WriteLine("\n Введите радиус окружности (R):\n");  double R = double.Parse(Console.ReadLine());  Task03\_2 obl = new Task03\_2();  Console.WriteLine("\n Введите координаты точки попадания в область (х;у):\n");  for (int q = 0; q < 1; q++)  {  Console.WriteLine("{0}\n", obl.Two(X, Y, R));  }  }  }  class Task03\_1  {  public double One(double x)  {  double y = 0;  if (x < -7 || x > 11) { return double.NaN; }  else  {  if (x < -7) y = 0;  if (x >= -7 && x < -3) y = 3;  if (x >= -3 && x < 3) y = -Math.Sqrt(9 - x \* x) + 3;  if (x >= 3 && x < 6) y = -2 \* x + 9;  if (x >= 6 && x < 11) y = x - 9;  if (x >= 11) y = 2;  return y;  }  }  }  class Task03\_2  {  public string Two(double x, double y, double R)  {  string flag = "Не попадает";  if ((((x + R) \* (x + R) + (y - R) \* (y - R)) <= (R \* R))  || ((x >= -1) && (x <= 2 \* R) && (y >= -R) && (y <= 0))) flag = "Попадает";  return flag;  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 3.1.

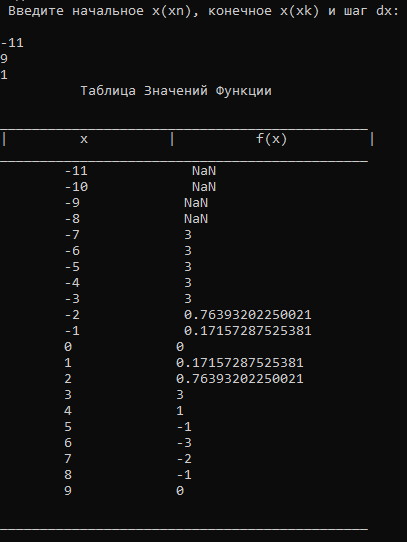


Рисунок 3.1 – Результат работы программы по организации циклов

# 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.

## Описание программы

*Описать класс, реализующий шестнадцатеричный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне.*

*Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, — и свойство, позволяющее получить его текущее состояние. При выходе за границы диапазона выбрасываются исключения.*

*Написать программу, демонстрирующую все разработанные элементы класса.*

## Текст программы

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

Листинг 4. – Текст файла prog4.cs

|  |
| --- |
| using System;  namespace Lab4  {  class Lab4  {  static void Main(string[] args)  {  try  {  Counter cn = new Counter();  while (true)  {  Console.Clear();  cn.ShowBorder();  cn.ShowCount();  Console.WriteLine("1. Установить значение");  Console.WriteLine("2. Увеличить значение счетчика");  Console.WriteLine("3. Уменьшить значение счетчика");  Console.WriteLine("4. Установить диапазон");  Console.Write("> ");  switch (Convert.ToInt32(Console.ReadLine()))  {  case 1:  {  Console.Write("Введите значение: ");  cn = new Counter(Convert.ToInt32(Console.ReadLine()));  break;  }  case 2:  {  cn.CountUp();  break;  }  case 3:  {  cn.CountDown();  break;  }  case 4:  {  Console.Write("Введите минимальное значение: ");  int minValue = Convert.ToInt32(Console.ReadLine());  Console.Write("Введите максимальное значение: ");  int maxValue = Convert.ToInt32(Console.ReadLine());  Counter.SetBorder(minValue, maxValue);  break;  }  }  }  }  catch (ArgumentException)  {  Console.WriteLine("ИСКЛЮЧЕНИЕ!!! COUNTER OVERFLOW");  }  finally  {  Console.ReadKey();  }  }  }  }  public class Counter  {  public Counter()  {  count = 250;  }  public Counter(int count)  {  if (count > BottomBorder && count <= TopBorder)  this.count = count;  else throw new ArgumentException();  }  private int count;  public static int BottomBorder = 0, TopBorder = 100;  public static void SetBorder(int i, int y)  {  if (i >= 0 && y > 1)  {  BottomBorder = i;  TopBorder = y;  }  else  {  BottomBorder = 0;  TopBorder = 100;  }  }    private int Count  {  get { return count; }  }  public void CountDown()  {  if (count > BottomBorder)  {  count--;  }  else throw new ArgumentException();  }  public void CountUp()  {  if (count < TopBorder)  {  count++;  }  else throw new ArgumentException();  }  public void ShowCount()  {  Console.WriteLine("Текущее значение счетчика (шестнадцатеричное значение): " + Convert.ToString(Count, 16));  }  public void ShowBorder()  {  Console.WriteLine("Текущий диапазон: {0} - {1}", BottomBorder, TopBorder);  }  } |

## Тестирование программы

Результат работы программы приведен на рисунках 4.1 - 4.6.

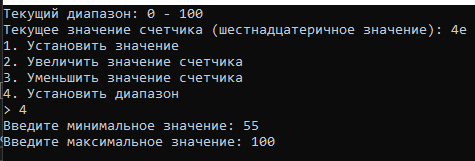


Рисунок 4.1 – Установка нового диапазона

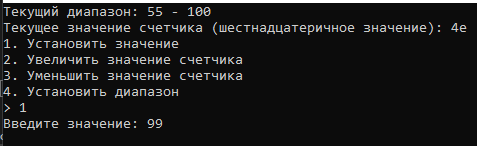


Рисунок 4.2 – Установка нового значения счетчика

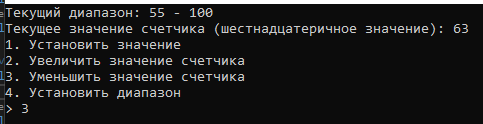


Рисунок 4.3 – Уменьшение диапазона

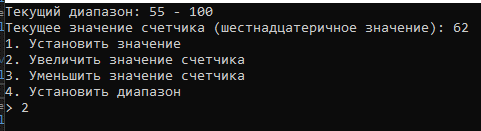


Рисунок 4.4 – Увеличение диапазона

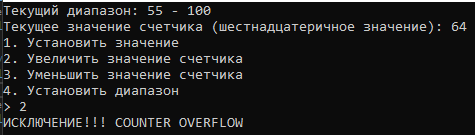


Рисунок 4.5 – Выход из диапазона

# 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.

## Описание программы

*В одномерном массиве, состоящем из n вещественных элементов вычислить:*

*1) сумму положительных элементов массива*

*2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.*

*Упорядочить элементы массива по убыванию.*

## Текст программы

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

Листинг 5. – Текст файла prog5.cs

|  |
| --- |
| using System;  namespace Lab5  {  class Lab5  {  static void Main(string[] args)  {  Console.Write("Количество элементов массива: ");  int arraySize = Convert.ToInt32(Console.ReadLine());  double[] array = new double[arraySize];  for (int i = 0; i < array.Length; i++)  {  Console.Write("Значение {0} элемента: ", i+1);  array[i] = Convert.ToDouble(Console.ReadLine()); ;  }  Console.Write("Введенный массив: ");  for (int i = 0; i < array.Length; i++)  {  Console.Write(" " + array[i]);  }  Console.WriteLine();  ///  double tempSum = 0;  for (int i = 0; i < array.Length; i++)  {  if (array[i] > 0) tempSum += array[i];  }  Console.WriteLine("Cумма положительных элементов массива: {0}", tempSum);  ///  double result = array[0];  int maxind = 0, minind = 0;  double max = array[0], min = array[0];  for (int i = 1; i < array.Length; i++)  {  if (Math.Abs(array[i]) > Math.Abs(max))  {  max = array[i];  maxind = i;  }  if (Math.Abs(array[i]) < Math.Abs(min))  {  min = array[i];  minind = i;  }  }  if (maxind > minind)  {  for (int i = minind; i < maxind; i++)  result \*= array[i];  }  else  {  for (int i = maxind; i < minind; i++)  result \*= array[i];  }  Console.WriteLine("Произведение элементов массива , расположенных между максимальным по модулю и минимальным по модулю элементами: {0}", result);  Console.ReadLine();  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 5.1.

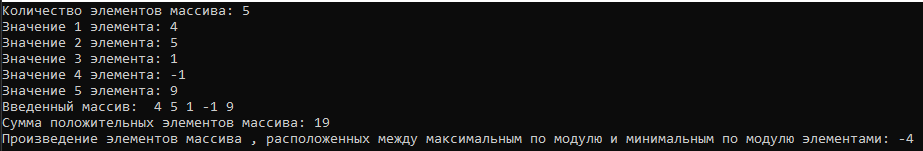


Рисунок 5.1 – Результат работы программы по одномерным массивам

# 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.

## Описание программы

*Дана целочисленная прямоугольная матрица.*

*Определить количество столбцов, не содержащих ни одного нулевого элемента.*

*Характеристикой строки целочисленной матрицы назовём сумму её положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.*

## Текст программы

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

Листинг 6. – Текст файла prog6.cs

|  |
| --- |
| using System;  namespace Lab6  {  class Lab6  {  static void Main()  {  Console.WriteLine("Создание прямоугольного массива");  Console.Write("Количество строк: ");  int rows = int.Parse(Console.ReadLine());  Console.Write("Количество столбцов: ");  int columns = int.Parse(Console.ReadLine());  Console.Clear();  int[,] matrix = new int[rows, columns];  Random random = new Random();  Console.WriteLine("Исходная матрица: ");  int[] sum = new int[rows];  for (int i = 0; i < rows; i++)  {  sum[i] = 0;  for (int j = 0; j < columns; j++)  {  matrix[i, j] = random.Next(-10, 10);  Console.Write("{0, 5}", matrix[i, j]);  if (matrix[i, j] > 0 && matrix[i, j] % 2 == 0)  sum[i] += matrix[i, j];  }  Console.WriteLine(" Сумма положительных четных: {0}", sum[i]);  }  Console.WriteLine();  int count = 0;  for (int i = 0; i < columns; i++)  {  try  {  for (int j = 0; j < rows; j++)  {  if (matrix[j, i] == 0)  throw new ArgumentException();  }  count++;  }  catch (ArgumentException) { }  }  Console.WriteLine("Количество столбцов, не содержащих ни одного нулевого элемента: {0}", count);  Console.WriteLine();  for (int i = 0; i < sum.Length - 1; i++)  {  for (int j = i + 1; j < sum.Length; j++)  if (sum[i] > sum[j])  {  int b = sum[i];  sum[i] = sum[j];  sum[j] = b;  for (int m = 0; m < columns; m++)  {  b = matrix[i, m];  matrix[i, m] = matrix[j, m];  matrix[j, m] = b;  }  }  }  Console.WriteLine("Новая матрица: ");  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < columns; j++)  Console.Write("{0, 5}", matrix[i, j]);  Console.WriteLine(" Sum: {0}", sum[i]);  }  Console.ReadLine();  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 6.1.

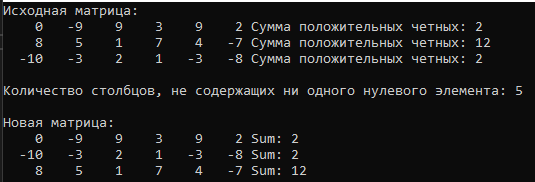


Рисунок 6.1 – Результат работы программы по двумерным массивам

# 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.

## Описание программы

*Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.*

## Текст программы

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

Листинг 7. – Текст файла prog7.cs

|  |
| --- |
| using System;  using System.IO;  namespace Lab7  {  class Lab7  {  static void Main(string[] args)  {  Console.Write("Файл: ");  string text = File.ReadAllText(Console.ReadLine());  string[] sentences = text.Split('.', '?', '!');  Console.Write("Искомое слово: ");  string word = Console.ReadLine();  foreach (string sent in sentences)  if (sent.Contains(word))  Console.WriteLine(sent);  Console.ReadLine();  }  }  } |

## Тестирование программы

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

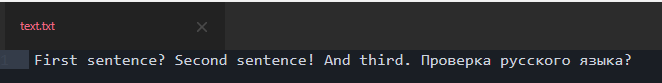


Рисунок 7.1 – Текст из файла \*txt



Рисунок 7.2 - Результат работы программы

# 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.

## Описание программы

*Описать класс для работы с одномерным массивом строк фиксированной длинны.*

*Обеспечить следующие возможности:*

*- Задание произвольных целых границ индексов при создании объекта*

*- Обращение к отдельной строке массива по индексу с контролем выхода за пределы массива*

*- Выполнение операций поэлементного сцепления двух массивов с образованием нового массива*

*- Выполнение операций слияния двух массивов с исключением повторяющихся элементов*

*- Вывод на экран элемента массива по заданному индексу и всего массива.*

## Текст программы

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

Листинг 8. – Текст файла prog8.cs

|  |
| --- |
| using System;  namespace Lab8  {  class Lab8  {  static void Main(string[] args)  {  Console.SetWindowSize(80, 50);  Random random = new Random();  Console.WriteLine("Создание первого массива с границами индексов c -2 до 3");  MyArray arr = new MyArray(-2, 3);  for (int i = arr.FirstIndex; i < arr.LastElemPos; i++)  {  arr[i] = random.Next(-5, 5).ToString();  }  Console.WriteLine("Все элементы первого массива");  for (int i = arr.FirstIndex; i < arr.LastElemPos; i++)  {  Console.WriteLine("индекс {0}: значение: {1}", i.ToString(), arr[i]);  }  Console.WriteLine();  Console.WriteLine("Создание второго массива с границами индексов c 1 до 5");  MyArray arr2 = new MyArray(1, 5);  for (int i = arr2.FirstIndex; i < arr2.LastElemPos; i++)  {  arr2[i] = random.Next(-5, 5).ToString();  }  Console.WriteLine("Все элементы второго массива");  for (int i = arr2.FirstIndex; i < arr2.LastElemPos; i++)  {  Console.WriteLine("индекс {0}: значение: {1}", i.ToString(), arr2[i]);  }  Console.WriteLine();  Console.WriteLine("Элемент первого массива с индексом -1 - {0}", arr[-1]);  Console.WriteLine("Элемент первого массива с индексом 10 - {0}", arr[-10]);  Console.WriteLine();  MyArray concatArr = MyArray.Concat(arr, arr2);  Console.WriteLine("Объединенный массив");  for (int i = concatArr.FirstIndex; i < concatArr.LastElemPos; i++)  {  Console.WriteLine("индекс {0}: значение: {1}", i.ToString(), concatArr[i]);  }  Console.WriteLine();  MyArray MergeArr = MyArray.Merge(arr, arr2);  Console.WriteLine("объединенный массив с исключением повторяющихся элементов из другого массива");  for (int i = MergeArr.FirstIndex; i < MergeArr.LastElemPos; i++)  {  Console.WriteLine("индекс {0}: значение: {1}", i.ToString(), MergeArr[i]);  }  Console.ReadLine();  }  }  // создаем наш класс - контейнер для стрингов, реализуем Icloneable,  //чтобы не портить исходные массивы при объединении  public class MyArray : ICloneable  {  private int firstIndex, lastIndex;  string[] arr; //объявляем внутренний служебный массив  public MyArray(int firstIndex, int lastIndex) // конструктор, вызывается при созданнии объекта  {  if (firstIndex >= lastIndex)  throw new ArgumentException("первый индекс должен быть меньше последнего");  this.firstIndex = firstIndex; // инициализируем границы нашего контейнера  this.lastIndex = lastIndex;  arr = new string[lastIndex - firstIndex + 1]; // инициализируем внутренний массив  }  public int FirstIndex // свойство, возвращающее индекс первого элемент нашего контейнера  {  get { return firstIndex; }  }  public int LastElemPos // свойство, возвращающее индекс последнего элемент нашего контейнера  {  get { return arr.Length + firstIndex; }  }  public string this[int pos] // для того, чтобы обращаться к нашему контейнеру синтаксисом MyArr[i], создаем индексатор  {  get  {  try  {  if (pos < firstIndex || pos > lastIndex) // проверяем границы массива, если вышли - генерируем исключение  throw new ArgumentOutOfRangeException("Вышли за границы массива :(");  else  return arr[pos - firstIndex];  }  catch (ArgumentOutOfRangeException)  {  return "ArgumentOutOfRangeException";  }  }  set  {  if (pos < firstIndex || pos > lastIndex) // проверяем границы массива, если вышли - генерируем исключение  throw new ArgumentOutOfRangeException("Вышли за границы массива :(");  else  arr[pos - firstIndex] = value;  }  }  public object Clone() // реализуем клонирование  {  MyArray copy = new MyArray(this.firstIndex, this.lastIndex);  for (int i = 0; i < copy.arr.Length; i++)  {  copy.arr[i] = this.arr[i];  }  return copy;  }  public static MyArray Concat(MyArray arr1, MyArray arr2) //объединяем массивы  {  //клонируем входящие массивы, чтобы не портить данные  MyArray temp1 = (MyArray)arr1.Clone();  MyArray temp2 = (MyArray)arr2.Clone();  int temp = temp1.arr.Length;  Array.Resize(ref temp1.arr, temp1.arr.Length + temp2.arr.Length);  temp1.lastIndex += temp2.arr.Length;  temp2.arr.CopyTo(temp1.arr, temp);  return temp1;  }  public static MyArray Merge(MyArray pArr1, MyArray pArr2)  {  MyArray arr1 = (MyArray)pArr1.Clone();  MyArray arr2 = (MyArray)pArr2.Clone();  int sizeForAdd = arr2.arr.Length;  for (int i = 0; i < arr1.arr.Length; i++)  {  for (int j = 0; j < arr2.arr.Length; j++)  {  if (arr1.arr[i] == arr2.arr[j]) // если нашли совпадения, повторяющемуся эл-ту присваиваем null  {  arr2.arr[j] = null;  sizeForAdd--;  }  }  }  int temp = arr1.arr.Length;  Array.Resize(ref arr1.arr, arr1.arr.Length + sizeForAdd);  for (int i = 0; i < arr2.arr.Length; i++)  {  if (arr2.arr[i] != null)  {  arr1.arr[temp] = arr2.arr[i];  temp++;  }  }  arr1.lastIndex += sizeForAdd;  return arr1;  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 8.

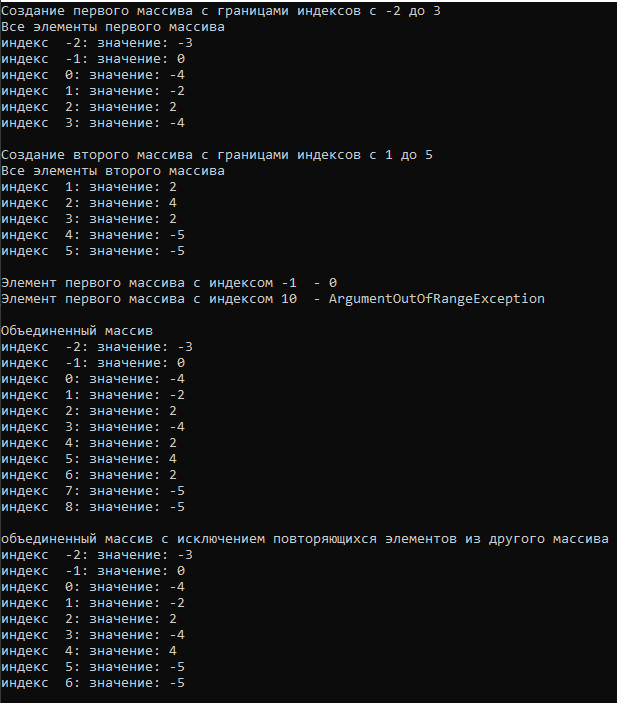


Рисунок 8 - Результат работы программы по классам и операциям

# 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.

## Описание программы

*Создать абстрактный класс Vehicle (транспортное средство). На его основе реализовать классы Plane (самолет), Саr (автомобиль) и Ship (корабль).*

*Классы должны иметь возможность задавать и получать координаты и параметры средств передвижения (цена, скорость, год выпуска и т. п.) с помощью свойств. Для самолета должна быть определена высота, для самолета и корабля — количество пассажиров, для корабля — порт приписки.*

*Динамические характеристики задать с помощью методов.*

## Текст программы

Проект состоит из 3 файлов исходного кода, которые приведены в листингах 8.1, 8.2, 8.3.

Листинг 8. – Текст файла prog9.cs

|  |
| --- |
| using System;  namespace Vehicle  {  class Lab9  {  static void Main()  {  Console.WriteLine("Создание машины");  Console.Write("Стоимость: ");  var car = new Car { Price = Convert.ToDouble(Console.ReadLine())};  Console.WriteLine("Координаты X и Y (через Enter)");  double carX = Convert.ToDouble(Console.ReadLine());  double carY = Convert.ToDouble(Console.ReadLine());  car.Coordinate(carX, carY);  Console.WriteLine();  Console.WriteLine("Характеристики машины");  Console.WriteLine("Стоимость машины {0}:", car.Price.ToString());  Console.WriteLine("Координаты машины X:{0},Y: {1}", car.CoordinateX.ToString(), car.CoordinateY.ToString());  Console.WriteLine();  Console.WriteLine("Создание самолета");  Console.Write("Стоимость: ");  var plane = new Plane { Price = Convert.ToDouble(Console.ReadLine()) };  Console.WriteLine("Координаты X и Y (через Enter)");  double planeX = Convert.ToDouble(Console.ReadLine());  double planeY = Convert.ToDouble(Console.ReadLine());  plane.Coordinate(planeX, planeY);  Console.Write("Высота самолёта: ");  plane.Height = Convert.ToDouble(Console.ReadLine());  Console.Write("Количество пассажиров: ");  plane.NumberOfPassengers = Convert.ToDouble(Console.ReadLine());  Console.WriteLine();  Console.WriteLine("Характеристики самолета");  Console.WriteLine("Стоимость самолета {0}:", plane.Price.ToString());  Console.WriteLine("Координаты самолета X:{0},Y: {1}", plane.CoordinateX.ToString(), plane.CoordinateY.ToString());  Console.WriteLine("Высота самолёта: {0}", plane.Height.ToString());  Console.WriteLine("Количество пассажиров: {0}", plane.NumberOfPassengers.ToString());  Console.WriteLine("Создание корабля");  Console.Write("Стоимость: ");  var ship = new Ship { Price = Convert.ToDouble(Console.ReadLine()) };  Console.WriteLine("Координаты X и Y (через Enter)");  double shipX = Convert.ToDouble(Console.ReadLine());  double shipY = Convert.ToDouble(Console.ReadLine());  ship.Coordinate(shipX, shipY);  Console.Write("Количество пассажиров: ");  ship.NumberOfPassengers = Convert.ToDouble(Console.ReadLine());  Console.Write("Порт: ");  ship.DestinationPort = Console.ReadLine();  Console.WriteLine();  Console.WriteLine("Характеристики корабля");  Console.WriteLine("Стоимость корабля {0}:", ship.Price.ToString());  Console.WriteLine("Координаты корабля X:{0},Y: {1}", ship.CoordinateX.ToString(), ship.CoordinateY.ToString());  Console.WriteLine("Количество пассажиров: {0}", ship.NumberOfPassengers.ToString());  Console.WriteLine("Порт: {0}", ship.DestinationPort);  Console.Read();  }  }  internal class Vehicle  {  private double coordinateX, coordinateY;  private double price, speed, year\_of\_construction;  /\* можно также применять автосвойства(тогда не нужны написанные выше поля ):  public double CoordinateX { get; set; }  \*/  public double CoordinateX  {  get { return coordinateX; }  set { coordinateX = value; }  }  public double CoordinateY  {  get { return coordinateY; }  set { coordinateY = value; }  }  public double Price  {  get { return price; }  set { price = value; }  }  public double Speed  {  get { return speed; }  set { speed = value; }  }  public double Year\_of\_construction  {  get { return year\_of\_construction; }  set { year\_of\_construction = value; }  }  //Метод для динамически меняющихся свойств  public void Coordinate(double CoordinateX, double CoordinateY)  {  this.CoordinateX = CoordinateX;  this.coordinateY = CoordinateY;  }  //Метод для динамически меняющихся свойств  public void SpeedMethod(double Speed)  {  this.Speed = Speed;  }  }  class Plane : Vehicle  {  public double Height { get; set; }  public double NumberOfPassengers { get; set; }  }  class Ship : Vehicle  {  public double NumberOfPassengers { get; set; }  public string DestinationPort { get; set; }  }  class Car : Vehicle  {  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 9.

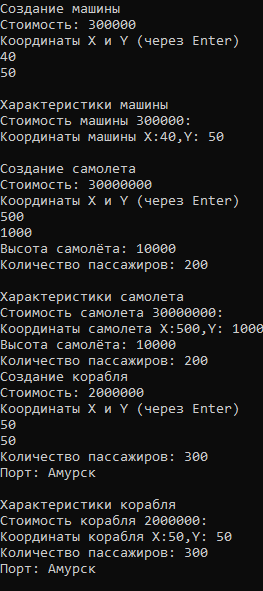


Рисунок 9 - Результат работы программы

# 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.

## Описание программы

*Описать структуру с именем STUDENT, содержащую следующие поля:*

*• фамилия и инициалы;*

*• номер группы;*

*• успеваемость (массив из пяти элементов).*

*Написать программу, выполняющую следующие действия:*

*• ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT*

*(упорядочить записи по возрастанию среднего балла);*

*• вывод на дисплей фамилий и номеров групп для всех студентов,*

*включённых в массив, если у них имеются оценки 4 и 5, если их нету вывести соответствующее сообщение.*

## Текст программы

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

Листинг 10. – Текст файла prog10.cs

|  |
| --- |
| using System;  using System.Linq;  namespace Students  {  class Lab10  {  static void Main()  {  Student[] students = new Student[4];  for (int i = 0; i < students.Length; i++)  {  Console.Write("Введите фамилию и инициалы {0}-ого студента: ", i + 1);  string name = Console.ReadLine();  Console.Write("Введите номер группы {0}-ого студента: ", i + 1);  int nomber = int.Parse(Console.ReadLine());  Console.Write("Введите, через запятую, 5 оценок {0}-ого студента: ", i + 1);  string[] marks = Console.ReadLine().Split(',');  int[] progress = new int[5];  for (int a = 0; a < 5; a++)  progress[a] = int.Parse(marks[a].ToString());  students[i] = new Student(name, nomber, progress);  }  var stud = from i in students orderby i.MediumBall() select i;  Console.WriteLine("\nУпорядоченный список по возрастанию среднего балла студентов: ");  foreach (Student student in stud)  Console.WriteLine("\n" + student.ToString());  Console.WriteLine("Студенты у которых имеются оценки 4 и 5");  int count = 0;  foreach (Student student in students)  if (student.progress.Any(x => x == 4) && student.progress.Any(x => x == 5))  {  count++;  Console.WriteLine("\n" + student.ToString());  }  if (count == 0)  Console.WriteLine("\nНет студентов с оценками 4 и 5!");  Console.ReadKey();  }  }  struct Student  {  public string Name;  public int GroupNomber;  public int[] progress;  public Student(string Name, int GroupNomber, int[] marks)  {  this.Name = Name;  this.GroupNomber = GroupNomber;  progress = marks;  }  public double MediumBall()  {  double MedBall = 0;  foreach (int i in progress)  MedBall += i;  MedBall /= progress.Length;  return MedBall;  }  public override string ToString()  {  return string.Format("ФИО: {0} \nНомер группы: {1}", Name, GroupNomber);  }  }  } |

## Тестирование программы

Результат работы программы приведен на рисунке 10.

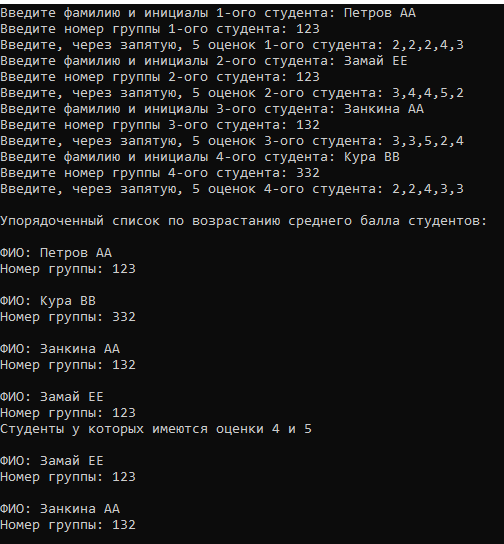


Рисунок 10 - Результат работы программы по структурам

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»).