

**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕУЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | Информационные системы и технологии |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по практической работе №3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | Лабораторная работа №4 «Массивы и строки». | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | **Разработка модулей программного обеспечения для компьютерных систем** |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Бедретдинов Ильсаф |  | Не требуется |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | Дкип-206Прог |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | **Сибирев Иван Валерьевич** |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2024 г.**

Вариант 4 Лаба 5

A screenshot of a computer

AI-generated content may be incorrect.

using System;

public class Address

{

// Поля класса

private string \_country;

private string \_region;

private string \_city;

private string \_street;

private string \_houseNumber;

private string \_apartmentNumber;

private string \_postalCode;

// Конструктор класса

public Address(string country, string region, string city, string street, string houseNumber, string apartmentNumber, string postalCode)

{

Country = country;

Region = region;

City = city;

Street = street;

HouseNumber = houseNumber;

ApartmentNumber = apartmentNumber;

PostalCode = postalCode;

}

// Свойства для доступа к полям с проверкой допустимости значений

public string Country

{

get { return \_country; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Страна не может быть пустой.");

}

\_country = value;

}

}

public string Region

{

get { return \_region; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Регион не может быть пустым.");

}

\_region = value;

}

}

public string City

{

get { return \_city; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Город не может быть пустым.");

}

\_city = value;

}

}

public string Street

{

get { return \_street; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Улица не может быть пустой.");

}

\_street = value;

}

}

public string HouseNumber

{

get { return \_houseNumber; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Номер дома не может быть пустым.");

}

\_houseNumber = value;

}

}

public string ApartmentNumber

{

get { return \_apartmentNumber; }

set

{

if (string.IsNullOrEmpty(value))

{

\_apartmentNumber = ""; // Квартира может отсутствовать, допустимо пустую строку.

return;

}

\_apartmentNumber = value;

}

}

public string PostalCode

{

get { return \_postalCode; }

set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentException("Почтовый индекс не может быть пустым.");

}

// Проверка на соответствие шаблону (можно настроить под конкретную страну)

// Например, для России - 6 цифр

if (!System.Text.RegularExpressions.Regex.IsMatch(value, @"^\d{6}$"))

{

throw new ArgumentException("Некорректный формат почтового индекса (требуется 6 цифр).");

}

\_postalCode = value;

}

}

// Метод для представления адреса в виде строки

public override string ToString()

{

return $"{Country}, {Region}, {City}, ул. {Street}, д. {HouseNumber}, кв. {ApartmentNumber}, {PostalCode}";

}

}

public class Program

{

public static void Main(string[] args)

{

try

{

// Создание объекта Address

Address address = new Address(

"Россия",

"Московская область",

"Москва",

"Тверская",

"10",

"123",

"123456"

);

// Вывод адреса

Console.WriteLine("Исходный адрес: " + address);

// Изменение отдельных частей адреса

address.Street = "Арбат";

address.HouseNumber = "15";

Console.WriteLine("Измененный адрес: " + address);

// Проверка обработки исключений

try

{

address.PostalCode = "123"; // Некорректный индекс

}

catch (ArgumentException ex)

{

Console.WriteLine("Ошибка при изменении почтового индекса: " + ex.Message);

}

try

{

address.Country = ""; // Пустая страна

}

catch (ArgumentException ex)

{

Console.WriteLine("Ошибка при изменении страны: " + ex.Message);

}

// Проверка допустимости пустой квартиры

address.ApartmentNumber = "";

Console.WriteLine("Адрес с пустой квартирой: " + address);

}

catch (ArgumentException ex)

{

Console.WriteLine("Произошла ошибка: " + ex.Message);

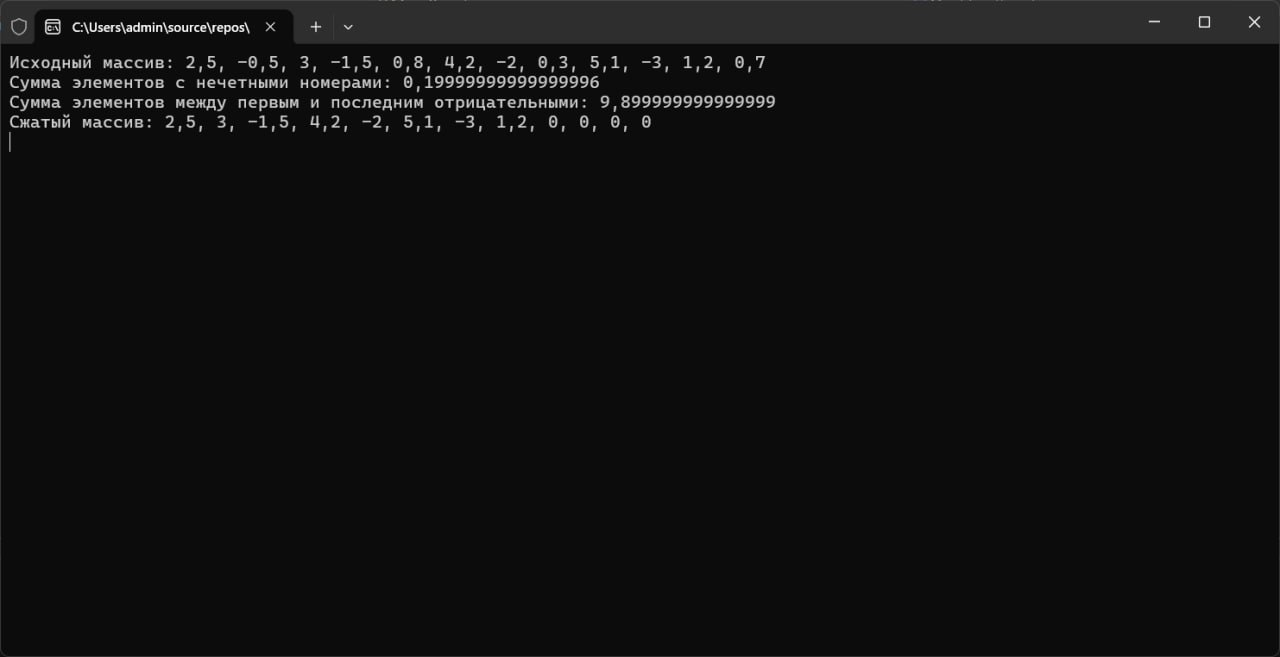
}

Console.ReadKey();

}

}

Вариант 4 Лаба 6



using System;

using System.Linq;

class Program

{

static void Main()

{

try

{

// Ввод размеров матрицы

Console.Write("Введите количество строк: ");

int rows = int.Parse(Console.ReadLine());

Console.Write("Введите количество столбцов: ");

int cols = int.Parse(Console.ReadLine());

// Инициализация матрицы

int[][] matrix = new int[rows][];

for (int i = 0; i < rows; i++)

{

matrix[i] = new int[cols];

for (int j = 0; j < cols; j++)

{

Console.Write($"Элемент [{i}][{j}]: ");

matrix[i][j] = int.Parse(Console.ReadLine());

}

}

// Функция для подсчета максимального количества одинаковых элементов в строке

Func<int[], int> countMaxDuplicates = row =>

{

var counts = new System.Collections.Generic.Dictionary<int, int>();

foreach (int num in row)

{

if (counts.ContainsKey(num))

counts[num]++;

else

counts[num] = 1;

}

return counts.Values.Max();

};

// Сортировка строк по возрастанию количества одинаковых элементов

Array.Sort(matrix, (row1, row2) => countMaxDuplicates(row1).CompareTo(countMaxDuplicates(row2)));

// Вывод отсортированной матрицы

Console.WriteLine("\nОтсортированная матрица:");

for (int i = 0; i < rows; i++)

{

Console.WriteLine(string.Join(" ", matrix[i]));

}

// Поиск первого столбца без отрицательных элементов

int firstNonNegativeColumn = -1;

for (int col = 0; col < cols; col++)

{

bool hasNegative = false;

for (int row = 0; row < rows; row++)

{

if (matrix[row][col] < 0)

{

hasNegative = true;

break;

}

}

if (!hasNegative)

{

firstNonNegativeColumn = col + 1; // нумерация с 1

break;

}

}

if (firstNonNegativeColumn != -1)

{

Console.WriteLine($"\nНомер первого столбца без отрицательных элементов: {firstNonNegativeColumn}");

}

else

{

Console.WriteLine("\nСтолбцов без отрицательных элементов нет.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

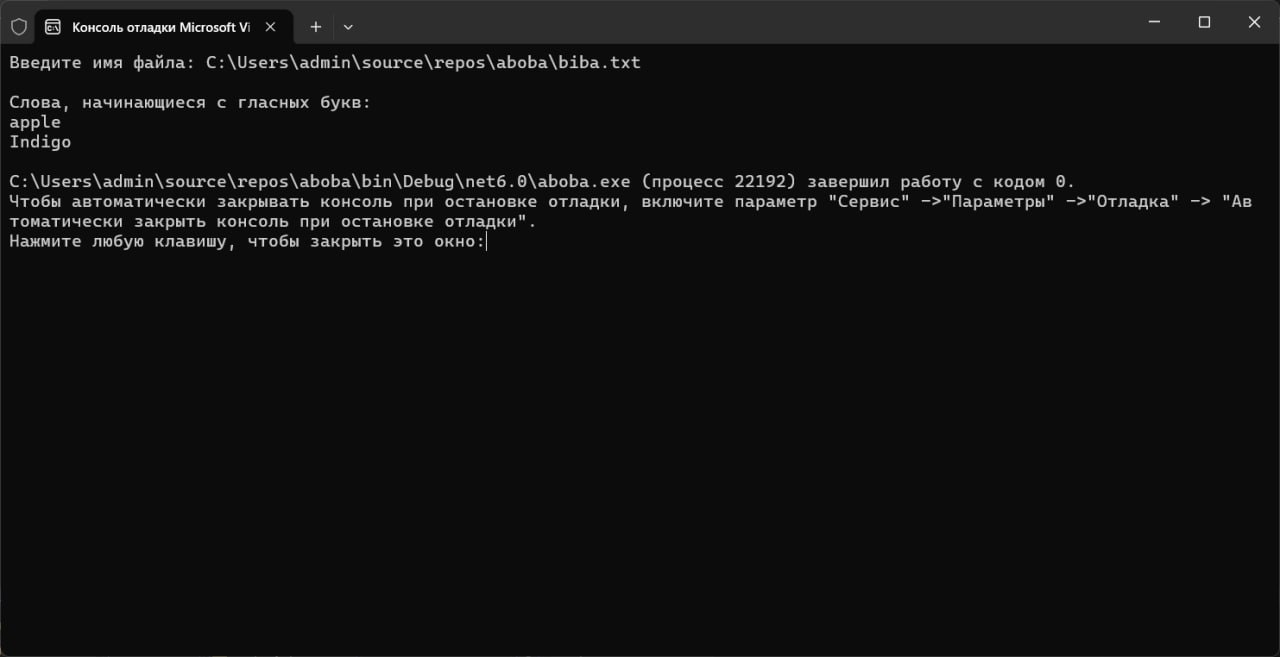
}

}

Лабораторная работа 7. Строки

Теоретический материал: глава 6, раздел «Символы и строки»

Вариант 4



using System;

using System.Collections.Generic;

using System.IO;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

try

{

Console.Write("Введите имя файла: ");

string fileName = Console.ReadLine();

string text = File.ReadAllText(fileName);

// Регулярное выражение для поиска слов, начинающихся с гласных (регистронезависимо)

Regex regex = new Regex(@"\b[aeiouAEIOU]\w+\b", RegexOptions.IgnoreCase);

MatchCollection matches = regex.Matches(text);

List<string> vowelWords = new List<string>();

foreach (Match match in matches)

{

string word = match.Value.Trim(); //trim для удаления лишних пробелов в начале и конце

vowelWords.Add(word);

}

Console.WriteLine("\nСлова, начинающиеся с гласных букв:");

foreach (string word in vowelWords)

{

Console.WriteLine(word);

}

}

catch (Exception ex)

{

Console.WriteLine("Ошибка: " + ex.Message);

}

}

}