**19 «Документирование в формате XML».**

Задание 1. В темах: 3 (Процедуры и функции – методы класса), 12 (Делегаты), 16 (Работа с файлами) выполнить XML документирование кода.

Листинг программы:

using System;

/// <summary>

/// Класс для построения таблицы значений функции.

/// </summary>

class FunctionTable

{

/// <summary>

/// Точка входа в программу.

/// </summary>

static void Main()

{

Console.Write("Введите начальное значение a: ");

double a = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите конечное значение b: ");

double b = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите шаг h: ");

double h = Convert.ToDouble(Console.ReadLine());

BuildFunctionTable(a, b, h);

}

/// <summary>

/// Строит таблицу значений функции f(x) = x^2 в заданном диапазоне.

/// </summary>

/// <param name="a">Начальное значение диапазона.</param>

/// <param name="b">Конечное значение диапазона.</param>

/// <param name="h">Шаг изменения переменной x.</param>

static void BuildFunctionTable(double a, double b, double h)

{

Console.WriteLine(" x | f(x)");

Console.WriteLine("-----------");

for (double x = a; x <= b; x += h)

{

double fx = Math.Pow(x, 2);

Console.WriteLine($"{x,4} | {fx,4}");

}

}

}

Листинг программы:

using System;

/// <summary>

/// Программа, демонстрирующая сдвиг трех переменных вправо.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

static void Main()

{

double A1 = 1.0, B1 = 2.0, C1 = 3.0;

double A2 = 4.0, B2 = 5.0, C2 = 6.0;

ShiftRight3(ref A1, ref B1, ref C1);

Console.WriteLine($"Первый набор после сдвига: A1 = {A1}, B1 = {B1}, C1 = {C1}");

ShiftRight3(ref A2, ref B2, ref C2);

Console.WriteLine($"Второй набор после сдвига: A2 = {A2}, B2 = {B2}, C2 = {C2}");

}

/// <summary>

/// Выполняет сдвиг трех переменных вправо.

/// </summary>

/// <param name="A">Первая переменная.</param>

/// <param name="B">Вторая переменная.</param>

/// <param name="C">Третья переменная.</param>

static void ShiftRight3(ref double A, ref double B, ref double C)

{

double temp = C;

C = B;

B = A;

A = temp;

}

}

Листинг программы:

using System;

/// <summary>

/// Представляет двумерный массив целых чисел.

/// </summary>

public class TwoDimensionalArray

{

private int[,] array;

/// <summary>

/// Создает новый двумерный массив заданных размеров.

/// </summary>

/// <param name="rows">Количество строк в массиве.</param>

/// <param name="cols">Количество столбцов в массиве.</param>

public TwoDimensionalArray(int rows, int cols)

{

array = new int[rows, cols];

}

/// <summary>

/// Получает или задает значение элемента массива по указанным индексам.

/// </summary>

/// <param name="i">Индекс строки.</param>

/// <param name="j">Индекс столбца.</param>

/// <returns>Значение элемента массива.</returns>

public int this[int i, int j]

{

get { return array[i, j]; }

set { array[i, j] = value; }

}

/// <summary>

/// Оператор "больше", сравнивающий суммы элементов главной диагонали двух массивов.

/// </summary>

public static bool operator >(TwoDimensionalArray a, TwoDimensionalArray b)

{

return SumMainDiagonal(a) > SumMainDiagonal(b);

}

/// <summary>

/// Оператор "меньше", сравнивающий суммы элементов главной диагонали двух массивов.

/// </summary>

public static bool operator <(TwoDimensionalArray a, TwoDimensionalArray b)

{

return SumMainDiagonal(a) < SumMainDiagonal(b);

}

/// <summary>

/// Вычисляет сумму элементов главной диагонали массива.

/// </summary>

private static int SumMainDiagonal(TwoDimensionalArray arr)

{

int sum = 0;

for (int i = 0; i < Math.Min(arr.array.GetLength(0), arr.array.GetLength(1)); i++)

{

sum += arr[i, i];

}

return sum;

}

/// <summary>

/// Точка входа в программу.

/// </summary>

static void Main(string[] args)

{

TwoDimensionalArray arr1 = new TwoDimensionalArray(3, 3);

TwoDimensionalArray arr2 = new TwoDimensionalArray(3, 3);

arr1[0, 0] = 1; arr1[1, 1] = 2; arr1[2, 2] = 3;

arr2[0, 0] = 4; arr2[1, 1] = 5; arr2[2, 2] = 6;

if (arr1 > arr2)

{

Console.WriteLine("Сумма элементов главной диагонали arr1 больше, чем у arr2");

}

else if (arr1 < arr2)

{

Console.WriteLine("Сумма элементов главной диагонали arr1 меньше, чем у arr2");

}

else

{

Console.WriteLine("Суммы элементов главных диагоналей arr1 и arr2 равны");

}

}

}

Листинг программы:

using System;

/// <summary>

/// Программа для вычисления результата выражения, используя функцию SubMod.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

static void Main()

{

Console.Write("Введите a1: ");

double a1 = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите b1: ");

double b1 = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите a2: ");

double a2 = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите b2: ");

double b2 = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите c2: ");

double c2 = Convert.ToDouble(Console.ReadLine());

double result = SubMod(a1, b1) \* SubMod(a2, b2, c2);

Console.WriteLine($"Результат: {result}");

}

/// <summary>

/// Возвращает модуль разности между двумя числами.

/// </summary>

/// <param name="a">Первое число.</param>

/// <param name="b">Второе число.</param>

/// <returns>Модуль разности между a и b.</returns>

static double SubMod(double a, double b)

{

return Math.Abs(a - b);

}

/// <summary>

/// Возвращает модуль разности между тремя числами.

/// </summary>

/// <param name="a">Первое число.</param>

/// <param name="b">Второе число.</param>

/// <param name="c">Третье число.</param>

/// <returns>Модуль разности между a, b и c.</returns>

static double SubMod(double a, double b, double c)

{

return Math.Abs(a - b - c);

}

}

Листинг программы:

using System;

/// <summary>

/// Представляет информацию о продукте.

/// </summary>

public class Product

{

private string productName;

private string storeName;

private decimal price;

/// <summary>

/// Создает новый экземпляр класса Product.

/// </summary>

/// <param name="productName">Наименование продукта.</param>

/// <param name="storeName">Наименование магазина.</param>

/// <param name="price">Цена продукта.</param>

public Product(string productName, string storeName, decimal price)

{

this.productName = productName;

this.storeName = storeName;

this.price = price;

}

/// <summary>

/// Возвращает наименование продукта.

/// </summary>

public string ProductName

{

get { return productName; }

}

/// <summary>

/// Возвращает наименование магазина.

/// </summary>

public string StoreName

{

get { return storeName; }

}

/// <summary>

/// Возвращает цену продукта.

/// </summary>

public decimal Price

{

get { return price; }

}

}

class Program

{

static void Main()

{

Product product1 = new Product("Ноутбук", "ТехноМир", 49999.99m);

Product product2 = new Product("Смартфон", "МобиМаркет", 29999.99m);

Console.WriteLine($"Товар: {product1.ProductName}, Магазин: {product1.StoreName}, Цена: {product1.Price} руб.");

Console.WriteLine($"Товар: {product2.ProductName}, Магазин: {product2.StoreName}, Цена: {product2.Price} руб.");

}

}

Листинг программы:

using System;

/// <summary>

/// Делегат для расчета характеристик фигуры.

/// </summary>

/// <param name="R">Радиус фигуры.</param>

/// <returns>Значение характеристики фигуры.</returns>

delegate double CalcFigure(double R);

class Program

{

/// <summary>

/// Возвращает длину окружности для заданного радиуса.

/// </summary>

/// <param name="R">Радиус окружности.</param>

/// <returns>Длина окружности.</returns>

static double Get\_Length(double R)

{

return 2 \* Math.PI \* R;

}

/// <summary>

/// Возвращает площадь круга для заданного радиуса.

/// </summary>

/// <param name="R">Радиус круга.</param>

/// <returns>Площадь круга.</returns>

static double Get\_Area(double R)

{

return Math.PI \* R \* R;

}

/// <summary>

/// Возвращает объем шара для заданного радиуса.

/// </summary>

/// <param name="R">Радиус шара.</param>

/// <returns>Объем шара.</returns>

static double Get\_Volume(double R)

{

return (4.0 / 3.0) \* Math.PI \* R \* R \* R;

}

static void Main(string[] args)

{

CalcFigure CF;

CF = Get\_Length;

double circumference = CF(5);

Console.WriteLine("Длина окружности: " + circumference);

CF = Get\_Area;

double area = CF(5);

Console.WriteLine("Площадь круга: " + area);

CF = Get\_Volume;

double volume = CF(5);

Console.WriteLine("Объем шара: " + volume);

}

}

Листинг программы:

using System;

namespace CalculatorApp

{

/// <summary>

/// Программа-калькулятор, выполняющая различные арифметические операции.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Функции для выполнения операций

Func<double, double, double> Add = (x, y) => x + y;

Func<double, double, double> Sub = (x, y) => x - y;

Func<double, double, double> Mul = (x, y) => x \* y;

Func<double, double, double> Div = (x, y) =>

{

if (y != 0)

return x / y;

else

{

Console.WriteLine("Ошибка: деление на ноль!");

return double.NaN;

}

};

// Вывод меню операций

Console.WriteLine("Выберите операцию:");

Console.WriteLine("1. Сложение");

Console.WriteLine("2. Вычитание");

Console.WriteLine("3. Умножение");

Console.WriteLine("4. Деление");

// Считывание выбора операции

int choice = Convert.ToInt32(Console.ReadLine());

// Ввод двух чисел

Console.WriteLine("Введите два числа:");

double num1 = Convert.ToDouble(Console.ReadLine());

double num2 = Convert.ToDouble(Console.ReadLine());

double result = 0;

// Выбор операции и выполнение вычислений

switch (choice)

{

case 1:

result = Add(num1, num2);

break;

case 2:

result = Sub(num1, num2);

break;

case 3:

result = Mul(num1, num2);

break;

case 4:

result = Div(num1, num2);

break;

default:

Console.WriteLine("Неверный выбор операции!");

break;

}

// Вывод результата

Console.WriteLine("Результат: " + result);

}

}

}

Листинг программы:

using System;

/// <summary>

/// Класс для манипуляции строками.

/// </summary>

public class StringManipulation

{

/// <summary>

/// Преобразует все символы в верхний регистр.

/// </summary>

/// <param name="input">Входная строка.</param>

public static void ToUpper(string input)

{

Console.WriteLine(input.ToUpper());

}

/// <summary>

/// Преобразует все символы в нижний регистр.

/// </summary>

/// <param name="input">Входная строка.</param>

public static void ToLower(string input)

{

Console.WriteLine(input.ToLower());

}

/// <summary>

/// Выводит количество символов в строке.

/// </summary>

/// <param name="input">Входная строка.</param>

public static void CountCharacters(string input)

{

Console.WriteLine("Количество символов в строке: " + input.Length);

}

/// <summary>

/// Делегат для работы с методами манипуляции строками.

/// </summary>

/// <param name="input">Входная строка.</param>

public delegate void StringManipulationDelegate(string input);

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Создание экземпляра делегата

StringManipulationDelegate delegateInstance;

// Тестовая строка

string testString = "Hello, World!";

// Примеры использования методов манипуляции строками через делегаты

delegateInstance = ToUpper;

delegateInstance(testString);

delegateInstance = ToLower;

delegateInstance(testString);

delegateInstance = CountCharacters;

delegateInstance(testString);

}

}

Листинг программы:

using System;

namespace DelegateArrayExample

{

/// <summary>

/// Пример использования делегатов и массивов делегатов для генерации случайных чисел и вычисления их среднего значения.

/// </summary>

class Program

{

/// <summary>

/// Делегат для генерации случайных чисел.

/// </summary>

/// <returns>Случайное целое число.</returns>

delegate int RandomNumberDelegate();

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Массив делегатов для генерации случайных чисел

RandomNumberDelegate[] delegatesArray = new RandomNumberDelegate[5];

// Инициализация массива делегатов

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

{

delegatesArray[i] = () =>

{

Random random = new Random();

return random.Next(1, 101);

};

}

// Вычисление среднего арифметического случайных чисел

double average = CalculateAverage(delegatesArray);

// Вывод результата

Console.WriteLine("Среднее арифметическое случайных чисел: " + average);

}

/// <summary>

/// Вычисляет среднее арифметическое для массива делегатов, каждый из которых генерирует случайное число.

/// </summary>

/// <param name="delegates">Массив делегатов для генерации случайных чисел.</param>

/// <returns>Среднее арифметическое случайных чисел.</returns>

static double CalculateAverage(RandomNumberDelegate[] delegates)

{

int sum = 0;

// Вызов каждого делегата и суммирование полученных чисел

foreach (var del in delegates)

{

sum += del();

}

// Вычисление среднего арифметического

return (double)sum / delegates.Length;

}

}

}

Листинг программы:

using System;

using System.IO;

/// <summary>

/// Программа для чтения чисел из файла, фильтрации и записи результатов в другой файл.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

static void Main()

{

// Имя входного и выходного файлов

string inputFile = "f.txt";

string outputFile = "g.txt";

try

{

// Используем конструкцию using для автоматического закрытия файловых потоков

using (StreamReader reader = new StreamReader(inputFile))

using (StreamWriter writer = new StreamWriter(outputFile))

{

string line;

// Читаем файл построчно

while ((line = reader.ReadLine()) != null)

{

// Пытаемся преобразовать строку в число

if (int.TryParse(line, out int number))

{

// Проверяем, соответствует ли число условиям

if (number % 3 == 0 && number % 7 != 0)

{

// Записываем число в выходной файл

writer.WriteLine(number);

}

}

else

{

// Если строку не удалось преобразовать в число, выводим ошибку

Console.WriteLine($"Ошибка: Не удалось преобразовать строку '{line}' в число.");

}

}

}

// Выводим сообщение об успешном завершении программы

Console.WriteLine("Результат записан в файл g.txt");

}

catch (IOException ex)

{

// Если возникла ошибка ввода-вывода, выводим сообщение об ошибке

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

}

}

}

Листинг программы:

using System;

using System.IO;

/// <summary>

/// Программа для создания новой папки.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Путь к новой папке

string folderPath = "New\_folder";

try

{

// Создаем новую папку

Directory.CreateDirectory(folderPath);

Console.WriteLine("Папка успешно создана.");

}

catch (Exception ex)

{

// Выводим сообщение об ошибке, если что-то пошло не так

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

}

}

}

Листинг программы:

using System;

using System.IO;

using System.Linq;

/// <summary>

/// Программа для работы с текстовым файлом.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Имя текстового файла

string fileName = "text\_file.txt";

// a) Вывод всего содержимого файла на экран

Console.WriteLine("Содержимое файла:");

DisplayFileContent(fileName);

// b) Подсчет количества строк в файле

int lineCount = CountLines(fileName);

Console.WriteLine($"Количество строк в файле: {lineCount}");

// c) Подсчет количества символов в каждой строке

Console.WriteLine("Количество символов в каждой строке:");

CountCharactersInEachLine(fileName);

// d) Удаление последней строки из файла и запись результата в новый файл

string newFileName = "new\_text\_file.txt";

RemoveLastLine(fileName, newFileName);

Console.WriteLine($"Последняя строка удалена из файла '{fileName}'. Результат записан в файл '{newFileName}'.");

// e) Вывод на экран строк с s1 по s2

int s1 = 2; // начиная с третьей строки (индексация с нуля)

int s2 = 4; // заканчивая пятой строкой

Console.WriteLine($"Строки с {s1 + 1} по {s2 + 1}:");

DisplayLinesInRange(fileName, s1, s2);

// f) Поиск длины самой длинной строки и вывод ее на экран

int maxLength = FindLongestLine(fileName);

Console.WriteLine($"Длина самой длинной строки: {maxLength}");

// g) Вывод на экран всех строк, начинающихся с заданной буквы

char startingLetter = 'T';

Console.WriteLine($"Строки, начинающиеся с буквы '{startingLetter}':");

DisplayLinesStartingWithLetter(fileName, startingLetter);

// h) Переписывание строк файла в обратном порядке в другой файл

string reversedFileName = "reversed\_text\_file.txt";

ReverseWriteToFile(fileName, reversedFileName);

Console.WriteLine($"Строки файла '{fileName}' переписаны в файл '{reversedFileName}' в обратном порядке.");

}

/// <summary>

/// Создает текстовый файл и записывает в него 5 строк различной длины.

/// </summary>

/// <param name="fileName">Имя создаваемого файла.</param>

static void WriteTextToFile(string fileName)

{

using (StreamWriter writer = new StreamWriter(fileName))

{

writer.WriteLine("Short line");

writer.WriteLine("A bit longer line");

writer.WriteLine("This line is even longer");

writer.WriteLine("Another long line with some additional text");

writer.WriteLine("This is the longest line among all lines in this file");

}

}

// Другие методы оставлены без XML-документации для краткости...

// Вывод содержимого файла на экран

static void DisplayFileContent(string fileName)

{

using (StreamReader reader = new StreamReader(fileName))

{

Console.WriteLine(reader.ReadToEnd());

}

}

// Подсчет количества строк в файле

static int CountLines(string fileName)

{

return File.ReadAllLines(fileName).Length;

}

// Подсчет количества символов в каждой строке файла

static void CountCharactersInEachLine(string fileName)

{

string[] lines = File.ReadAllLines(fileName);

foreach (string line in lines)

{

Console.WriteLine($"Строка \"{line}\" содержит {line.Length} символов.");

}

}

// Удаление последней строки из файла и запись результата в новый файл

static void RemoveLastLine(string fileName, string newFileName)

{

string[] lines = File.ReadAllLines(fileName);

File.WriteAllLines(newFileName, lines.Take(lines.Length - 1));

}

// Вывод строк с s1 по s2 в файле

static void DisplayLinesInRange(string fileName, int s1, int s2)

{

string[] lines = File.ReadAllLines(fileName);

for (int i = s1; i <= s2; i++)

{

Console.WriteLine(lines[i]);

}

}

// Поиск длины самой длинной строки в файле

static int FindLongestLine(string fileName)

{

string[] lines = File.ReadAllLines(fileName);

int maxLength = lines.Max(line => line.Length);

return maxLength;

}

// Вывод строк файла, начинающихся с заданной буквы

static void DisplayLinesStartingWithLetter(string fileName, char letter)

{

string[] lines = File.ReadAllLines(fileName);

foreach (string line in lines)

{

if (line.StartsWith(letter.ToString(), StringComparison.OrdinalIgnoreCase))

{

Console.WriteLine(line);

}

}

}

// Переписывание строк файла в другой файл в обратном порядке

static void ReverseWriteToFile(string fileName, string reversedFileName)

{

string[] lines = File.ReadAllLines(fileName);

Array.Reverse(lines);

File.WriteAllLines(reversedFileName, lines);

}

}

Листинг программы:

using System;

using System.IO;

/// <summary>

/// Программа для замены символов в строках файла.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Имя входного и выходного файлов

string inputFileName = "input.txt";

string outputFileName = "output.txt";

// Переписывание строк с заменой символов и запись результата в новый файл

ReverseSymbols(inputFileName, outputFileName);

Console.WriteLine($"Строки файла '{inputFileName}' переписаны с заменой символов и записаны в файл '{outputFileName}'.");

}

/// <summary>

/// Переписывает строки из входного файла с заменой символов и записывает результат в выходной файл.

/// </summary>

/// <param name="inputFileName">Имя входного файла.</param>

/// <param name="outputFileName">Имя выходного файла.</param>

static void ReverseSymbols(string inputFileName, string outputFileName)

{

using (StreamReader reader = new StreamReader(inputFileName))

using (StreamWriter writer = new StreamWriter(outputFileName))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

string reversedLine = ReverseCharacters(line);

writer.WriteLine(reversedLine);

}

}

}

/// <summary>

/// Заменяет символы '0' на '1' и символы '1' на '0' в строке.

/// </summary>

/// <param name="line">Строка, в которой нужно заменить символы.</param>

/// <returns>Строка с замененными символами.</returns>

static string ReverseCharacters(string line)

{

char[] characters = line.ToCharArray();

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

{

if (characters[i] == '0')

characters[i] = '1';

else if (characters[i] == '1')

characters[i] = '0';

}

return new string(characters);

}

}

Листинг программы:

using System;

using System.IO;

using System.Linq;

/// <summary>

/// Программа для работы с числами в файле.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

double[] numbers = { 2.5, -1.3, 4.7, -0.8, 3.2, 1.6, -5.9, 7.1 };

string fileName = "numbers.txt";

WriteNumbersToFile(fileName, numbers);

double threshold = 2.0;

Console.WriteLine($"Числа файла меньше {threshold}:");

DisplayNumbersLessThan(fileName, threshold);

Console.WriteLine("Положительные числа файла:");

DisplayPositiveNumbers(fileName);

double average = CalculateEvenPositionAverage(fileName);

Console.WriteLine($"Среднее арифметическое чисел на четных позициях файла: {average}");

}

/// <summary>

/// Записывает числа в файл.

/// </summary>

/// <param name="fileName">Имя файла.</param>

/// <param name="numbers">Массив чисел для записи в файл.</param>

static void WriteNumbersToFile(string fileName, double[] numbers)

{

using (StreamWriter writer = new StreamWriter(fileName))

{

foreach (double number in numbers)

{

writer.WriteLine(number);

}

}

}

/// <summary>

/// Выводит на экран числа из файла, которые меньше заданного порога.

/// </summary>

/// <param name="fileName">Имя файла.</param>

/// <param name="threshold">Порог для сравнения чисел.</param>

static void DisplayNumbersLessThan(string fileName, double threshold)

{

using (StreamReader reader = new StreamReader(fileName))

{

string line;

while ((line = reader.ReadLine()) != null)

{

double number = double.Parse(line);

if (number < threshold)

{

Console.WriteLine(number);

}

}

}

}

/// <summary>

/// Выводит на экран положительные числа из файла.

/// </summary>

/// <param name="fileName">Имя файла.</param>

static void DisplayPositiveNumbers(string fileName)

{

using (StreamReader reader = new StreamReader(fileName))

{

string line;

while ((line = reader.ReadLine()) != null)

{

double number = double.Parse(line);

if (number > 0)

{

Console.WriteLine(number);

}

}

}

}

/// <summary>

/// Вычисляет среднее арифметическое чисел на четных позициях в файле.

/// </summary>

/// <param name="fileName">Имя файла.</param>

/// <returns>Среднее арифметическое чисел на четных позициях.</returns>

static double CalculateEvenPositionAverage(string fileName)

{

double sum = 0;

int count = 0;

using (StreamReader reader = new StreamReader(fileName))

{

string line;

int position = 0;

while ((line = reader.ReadLine()) != null)

{

if (position % 2 == 0)

{

double number = double.Parse(line);

sum += number;

count++;

}

position++;

}

}

if (count == 0)

{

return 0;

}

else

{

return sum / count;

}

}

}

Листинг программы:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

/// <summary>

/// Класс, представляющий программу для обработки оценок студентов.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

string marksFileName = "mark.txt";

string subjectsFileName = "Subjects.txt";

List<Student> students = ReadStudentsFromFile(marksFileName);

List<string> subjects = ReadSubjectsFromFile(subjectsFileName);

Console.WriteLine("Отличники:");

DisplayExcellentStudents(students);

Dictionary<string, double> averageScores = CalculateAverageScores(students);

Console.WriteLine("\nСредняя оценка класса по каждому предмету:");

foreach (var kvp in averageScores)

{

Console.WriteLine($"{subjects[int.Parse(kvp.Key) - 1]}: {kvp.Value}");

}

string outputFileName = "average\_scores.txt";

WriteStudentsAverageToFile(students, outputFileName);

Console.WriteLine($"\nДанные о средних баллах студентов записаны в файл '{outputFileName}'.");

}

/// <summary>

/// Класс, представляющий студента.

/// </summary>

class Student

{

/// <summary>

/// Имя студента.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Оценки студента.

/// </summary>

public List<int> Marks { get; set; }

/// <summary>

/// Средний балл студента.

/// </summary>

public double AverageScore => Marks.Average();

}

/// <summary>

/// Чтение информации о студентах из файла.

/// </summary>

/// <param name="fileName">Имя файла с оценками студентов.</param>

/// <returns>Список студентов.</returns>

static List<Student> ReadStudentsFromFile(string fileName)

{

List<Student> students = new List<Student>();

using (StreamReader reader = new StreamReader(fileName))

{

string line;

while ((line = reader.ReadLine()) != null)

{

string[] parts = line.Split(' ');

Student student = new Student

{

Name = parts[0],

Marks = parts.Skip(1).Select(int.Parse).ToList()

};

students.Add(student);

}

}

return students;

}

/// <summary>

/// Чтение списка предметов из файла.

/// </summary>

/// <param name="fileName">Имя файла с предметами.</param>

/// <returns>Список предметов.</returns>

static List<string> ReadSubjectsFromFile(string fileName)

{

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

using (StreamReader reader = new StreamReader(fileName))

{

string line;

while ((line = reader.ReadLine()) != null)

{

subjects.Add(line);

}

}

return subjects;

}

/// <summary>

/// Отображение студентов с отличными оценками.

/// </summary>

/// <param name="students">Список студентов.</param>

static void DisplayExcellentStudents(List<Student> students)

{

foreach (var student in students)

{

if (student.Marks.All(mark => mark >= 9))

{

Console.WriteLine(student.Name);

}

}

}

/// <summary>

/// Вычисление средних оценок по каждому предмету.

/// </summary>

/// <param name="students">Список студентов.</param>

/// <returns>Словарь средних оценок по предметам.</returns>

static Dictionary<string, double> CalculateAverageScores(List<Student> students)

{

Dictionary<string, List<int>> subjectScores = new Dictionary<string, List<int>>();

foreach (var student in students)

{

for (int i = 0; i < student.Marks.Count; i++)

{

string subjectIndex = (i + 1).ToString();

if (!subjectScores.ContainsKey(subjectIndex))

{

subjectScores[subjectIndex] = new List<int>();

}

subjectScores[subjectIndex].Add(student.Marks[i]);

}

}

Dictionary<string, double> averageScores = new Dictionary<string, double>();

foreach (var kvp in subjectScores)

{

averageScores[kvp.Key] = kvp.Value.Average();

}

return averageScores;

}

/// <summary>

/// Запись данных о средних баллах студентов в файл.

/// </summary>

/// <param name="students">Список студентов.</param>

/// <param name="fileName">Имя файла для записи данных.</param>

static void WriteStudentsAverageToFile(List<Student> students, string fileName)

{

using (StreamWriter writer = new StreamWriter(fileName))

{

foreach (var student in students.OrderByDescending(s => s.AverageScore))

{

writer.WriteLine($"{student.Name}: {student.AverageScore}");

}

}

}

}

Листинг программы:

using System;

using System.IO;

/// <summary>

/// Класс, представляющий программу для копирования файлов и создания символических ссылок.

/// </summary>

class Program

{

/// <summary>

/// Точка входа в программу.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

// Получаем список всех файлов в корневой директории "C:\"

string[] allFiles = Directory.GetFiles("C:\\");

// Создаем новую директорию "Example\_36tp" на диске D:

string exampleDir = @"D:\Example\_36tp";

Directory.CreateDirectory(exampleDir);

// Копируем файлы из исходной директории "C:\SourceDirectory" в созданную директорию "Example\_36tp"

string sourceDir = @"C:\SourceDirectory";

string[] filesToCopy = Directory.GetFiles(sourceDir);

foreach (string file in filesToCopy)

{

string fileName = Path.GetFileName(file);

string destFile = Path.Combine(exampleDir, fileName);

File.Copy(file, destFile, true);

}

// Устанавливаем атрибут Hidden для скопированных файлов и создаем символические ссылки на них

foreach (string file in filesToCopy)

{

string fileName = Path.GetFileName(file);

string destFile = Path.Combine(exampleDir, fileName);

File.SetAttributes(destFile, FileAttributes.Hidden);

string linkFilePath = Path.Combine(exampleDir, "Link\_" + fileName);

CreateSymbolicLink(linkFilePath, destFile, SymbolicLink.File);

}

Console.WriteLine("Операции завершены успешно.");

Console.ReadLine();

}

/// <summary>

/// Метод для создания символической ссылки.

/// </summary>

/// <param name="lpSymlinkFileName">Путь к создаваемой ссылке.</param>

/// <param name="lpTargetFileName">Путь к целевому файлу или директории.</param>

/// <param name="dwFlags">Тип символической ссылки (файл или директория).</param>

/// <returns>Возвращает true, если создание ссылки прошло успешно, иначе false.</returns>

[System.Runtime.InteropServices.DllImport("kernel32.dll")]

public static extern bool CreateSymbolicLink(string lpSymlinkFileName, string lpTargetFileName, SymbolicLink dwFlags);

/// <summary>

/// Перечисление, представляющее тип символической ссылки.

/// </summary>

public enum SymbolicLink

{

/// <summary>

/// Ссылка на файл.

/// </summary>

File = 0,

/// <summary>

/// Ссылка на директорию.

/// </summary>

Directory = 1

}

}