**19 ДОКУМЕНТИРОВАНИЕ В ФОРМАТЕ XML**

Задание 1. Процедуры и функции – методы класса, выполнить XML документирование кода.

Листинг программы:

class Program

{

/// <summary>

/// Вычисление функции

/// </summary>

/// <param name="arg"></param>

/// <returns></returns>

static int func(int arg)

{ return (arg / 10) % 10; }

/// <summary>

/// Метод Main

/// </summary>

/// <param name="args"></param>

static void Main(string[] args)

{

int a = 123456, b = 345678, c = 987694, result;

result = func(a) + func(b) - func(c);

Console.Write($" Сумма вторых справа цифр = { result } ");

Console.ReadKey();

}

}

Анализ результатов:



Рисунок 19.1 – Результат работы программы

Источник: собственная разработка

Задание 2. Делегаты, выполнить XML документирование кода.

Листинг программы:

namespace zad2

{

/// <summary>

/// создание Делегата

/// </summary>

/// <param name="a"></param>

/// <returns></returns>

public delegate string Stroka(string a);

class Program

{

static void Main()

{

Console.WriteLine("Введите действие: \n1 Удалить первое слово \n2 Замена всех букв о на ОК \n3 Удаление точек ");

string a = Console.ReadLine();

Console.WriteLine("Введите строку: ");

string x = Console.ReadLine();

switch (a)

{

case "1":

Stroka calc = (c) => c;

string del = x.Remove(0, x.IndexOf(' ') + 1);

Console.WriteLine("Ответ:{0}", del);

break;

case "2":

Stroka calc1 = (c) => c;

string del1 = x.Replace("о", "ОК");

Console.WriteLine("Ответ:{0}", del1);

break;

case "3":

Stroka calc2 = (c) => c;

string zamena = x.Replace(".", "");

Console.WriteLine("Ответ:{0}", zamena);

break;

}

Console.ReadKey();

}

}

}

Анализ результатов:

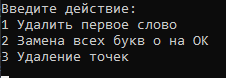


Рисунок 19.2 – Результат работы программы

Источник: собственная разработка

Задание 3. Работа с файлами, выполнить XML документирование кода.

Листинг программы:

static void Main()

{

Console.WriteLine("Вывод текста из файла");

Print();

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

var filePath = @"first.txt";

var filePath1 = @"second.txt";

int count = File.ReadAllLines(filePath).Length;

Console.WriteLine($"Количество строк: {count}");

var numbers = SymCount(filePath);

int i = 1;

foreach (var num in numbers)

{

Console.WriteLine($"Кол-во символов в {i} строке: {num.ToString()}");

i++;

}

Console.WriteLine("\nУдаление последней строки и запись результата в другой файл");

DeleteStringAndWrite(filePath, filePath1);

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

S1toS2(filePath, 1, 3);

LongString(filePath);

FirstLetterLine(filePath, 'в');

ReverseFile(filePath);

Console.ReadKey();

}

/// <summary>

/// Создание нового файла

/// </summary>

/// <param name="path"></param>

private static void ReverseFile(string path)

{

string pathoutput = @"Reverse.txt";

File.WriteAllLines(pathoutput, File.ReadAllLines(path).Reverse());

}

/// <summary>

/// возврат текста по первой букве

/// </summary>

/// <param name="path"></param>

/// <param name="firstChar"></param>

/// <returns></returns>

private static string FirstLetterLine(string path, char firstChar)

{

return Convert.ToString(File.ReadAllLines(path).Where(s => s[0] == firstChar));

}

/// <summary>

/// вывод длинной строки

/// </summary>

/// <param name="path"></param>

/// <returns></returns>

private static string LongString(string path)

{

return Convert.ToString(File.ReadAllLines(path).Where(s => s.Length == File.ReadAllLines(path).Max(m => m.Length)).First());

}

/// <summary>

/// вывод строк по номеру

/// </summary>

/// <param name="filePath"></param>

/// <param name="fromS1"></param>

/// <param name="toS2"></param>

/// <returns></returns>

private static string S1toS2(string filePath, int fromS1, int toS2)

{

return Convert.ToString(File.ReadAllLines(filePath).Skip(fromS1).Take(File.ReadAllLines(filePath).Length - toS2));

}

/// <summary>

/// удаление строки из первого файла и занесение остального текста в другой файл

/// </summary>

/// <param name="filePath"></param>

/// <param name="filePath1"></param>

private static void DeleteStringAndWrite(string filePath, string filePath1)

{

string[] readText = File.ReadAllLines(filePath);

string[] writeText = new string[readText.Length - 1];

Array.Copy(readText, 0, writeText, 0, readText.Length - 1);

File.WriteAllLines(filePath1, writeText);

}

/// <summary>

/// вывод строки с самым большим количеством символов

/// </summary>

/// <param name="filePath"></param>

/// <returns></returns>

private static List<int> SymCount(string filePath)

{

var countsCharInLines = new List<int>();

using (var reader = new StreamReader(filePath, detectEncodingFromByteOrderMarks: true))

{

var line = String.Empty;

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

{

var count = line.Length;

countsCharInLines.Add(count + 1);

}

}

return countsCharInLines;

}

/// <summary>

/// вывод текста

/// </summary>

private static void Print()

{

try

{

using (StreamReader sr = new StreamReader("first.txt"))

{

String line = sr.ReadToEnd();

Console.WriteLine(line);

sr.Close();

}

}

catch (Exception e)

{

Console.WriteLine("Исключение: " + e.Message);

}

}

Анализ результатов:

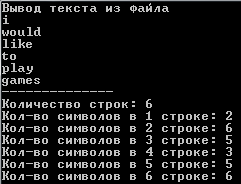


Рисунок 19.3 – Результат работы программы

Источник: собственная разработка