**Отчет по лабораторной работе № 6 по курсу С#**

5

(количество листов)

Студент группы ИУ5-32:

Платонов Артемий

Подпись Дата 06.12.2017

Руководитель: Гапанюк Ю.Е

Подпись

Дата

Москва, 2017

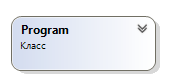
***Задание:***

Разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера.

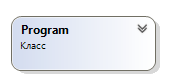
1. Программа должна быть разработана в виде библиотеки классов на языке C#.
2. Использовать самый простой вариант алгоритма без оптимизации.
3. Дополнительно возможно реализовать вычисление расстояния Дамерау-Левенштейна (с учетом перестановок соседних символов).
4. Модифицировать предыдущую лабораторную работу, вместо поиска подстроки используется вычисление расстояния Левенштейна.
5. Предусмотреть отдельное поле ввода для максимального расстояния. Если расстояние Левенштейна между двумя строками больше максимального, то строки считаются несовпадающими и не выводятся в список результатов.

***Диаграмма классов:***

Lab61:



Lab62:



***Текст программы:***

Lab61:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab61

{

class Program

{

delegate double Average(int p1, int p2, int p3);

static double Arif(int a1, int a2, int a3)

{

return (a1 + a2 + a3) / 3;

}

static void PrintAverage(int i1, int i2, int i3, Average AvParam)

{

Console.WriteLine("среднее от {0}, {1}, {2} = {3}", i1, i2, i3, AvParam(i1, i2, i3));

}

static void PrintAverageFunc(int i1, int i2, int i3, Func<int,int,int,double> AvParam)

{

Console.WriteLine("среднее от {0}, {1}, {2} = {3}", i1, i2, i3, AvParam(i1, i2, i3));

}

static void Main()

{

Console.WriteLine("Делегат Average\nделегат с готовым методом: ср. ариф.");

PrintAverage(1, 2, 3, Arif);

Console.WriteLine("делегат с лямбда-выражением: ср. геом.");

PrintAverage(1, 8, 27, (x, y, z) => Math.Pow(x\*y\*z, 1.0 / 3.0));

Console.WriteLine("Делегат Func<>\nделегат с готовым методом: ср. ариф.");

PrintAverage(1, 2, 3, Arif);

Console.WriteLine("делегат с лямбда-выражением: ср. геом.");

PrintAverage(1, 8, 27, (x, y, z) => Math.Pow(x \* y \* z, 1.0 / 3.0));

}

}

}

Lab62:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Lab62

{

class Program

{

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam) { Description = DescriptionParam; }

public string Description { get; set; }

}

public class ForInspection

{

public ForInspection() { }

public ForInspection(int i) { }

public ForInspection(string str, double d) { }

public int Plus(int x, int y) { return x + y; }

public int Minus(int x, int y) { return x - y; }

public double Average(int x, int y) { return (double)(x + y) / 2; }

[NewAttribute("Описание для property1")]

public string property1

{

get { return \_property1; }

set { \_property1 = value; }

}

private string \_property1;

public int property2 { get; set; }

[NewAttribute(Description = "Описание для property3")]

public double property3 { get; private set; }

public int field1;

public float field2;

}

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

var isAttribute = checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void Main()

{

Type t = typeof(ForInspection);

Console.WriteLine("Тип " + t.FullName + " унаследован от " + t.BaseType.FullName);

Console.WriteLine("Пространство имен " + t.Namespace);

Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("\nПоля данных (public):");

foreach (var x in t.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства, помеченные атрибутом:");

foreach (var x in t.GetProperties())

{

object attrObj;

if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))

{

NewAttribute attr = attrObj as NewAttribute;

Console.WriteLine(x.Name + " - " + attr.Description);

}

}

Console.WriteLine("\nВызов метода:");

ForInspection fi = (ForInspection)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

object[] parameters = new object[] { 3, 2 };

object Result = t.InvokeMember("Average", BindingFlags.InvokeMethod, null, fi, parameters);

Console.WriteLine("Plus(3,2)={0}", Result);

Console.ReadLine();

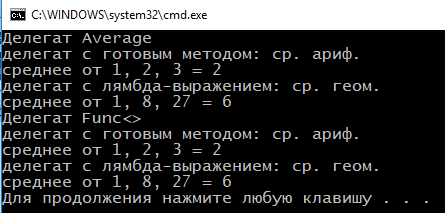
}

}

}

***Скриншоты:***

Lab61:



Lab62:

