**Отчет по ПР8.2**

**Лопаткин Сергей ИСП-41**

**Цель работы**

Научиться определять необходимое количество тестов.

**Задание:** Написать программу, найти минимальный набор тестов для программы своего варианта.

1. Дан одномерный массив *А*1, *А*2, …, *А*10 целых чисел. Получить наименьшее среди *А*1+*А*6, *А*2+*А*7, …, *А*5+*А*10.

**Теоретические сведения**

**Определение**

*Цикломатическая сложность* ⁠— предложенная Томасом Мак­‑Кейбом метрика программного обеспечения, позволяющая оценить сложность (англ. *complexity*) программы как количество *линейно независимых путей выполнения* в исходном коде. Значение цикломатической сложности иногда называют *числом Мак­‑Кейба*. Его можно вычислить для произвольного фрагмента исходного текста, в т. ч. для подпрограммы (процедуры, функции, метода) [1].

**Значение для тестирования**

Цикломатическая сложность обычно учитывается при тестировании базового пути. *Тестирование базового пути*, или *структурированное тестирование* (англ. *strucrured testing*), ⁠— также разработанный Т. Мак­‑Кейбом метод проектирования контрольных примеров, предназначенных для тестирования программы методом белого ящика.

Тестирование базового пути гарантирует полное *тестовое покрытие* линейно независимых путей выполнения, то есть по крайней мере однократное выполнение каждого выражения в исходном коде, с использованием минимально достаточного количества тестовых вариантов.

**Вычисление цикломатической сложности**

Алгоритм вычисления цикломатической сложности требует проанализировать *граф потока управления* (англ. *control­‑flow graph*, *CFG*), для которого характерны следующие элементы:

‑ вершины (узлы) ⁠— обозначают базовые блоки, последовательно выполняемые предложения;

‑ дуги (ориентированные рёбра) ⁠— отражают поток управления и, следовательно, соединяют вершины.

Вершины делятся на предикатные и операторные.

*Предикатная вершина* является началом двух и только двух дуг, так как обозначает конструкцию, определяющую последовательность выполнения в программе. Другими словами, предикатная вершина обозначает условие, используемое в условном операторе (if), операторе ветвления (case…of, switch…case и т. п.), операторе цикла (for, while и т. п.) и т. д. Оператор ветвления обозначается последовательностью предикатных вершин, а оператор цикла ⁠— предикатной вершиной, которая является не только началом двух дуг, но и концом как минимум одной дуги.

*Операторная вершина* не является началом ни одной дуги (см. ниже) или является началом только одной дуги, так как обозначает такой программный блок, который является последовательностью безусловных предложений; такая последовательность может содержать операторы присваивания, ввода, вывода и т. п., но не операторы условия.

Важное примечание: если подпрограмма пуста, то вход в неё обозначается операторной вершиной, которая не является концом ни одной дуги. В остальных случаях вход не обозначается.

Вершина, обозначающая выход из программы, является концом как минимум одной и как максимум бесконечного множества дуг, но, конечно же, не является началом ни одной дуги.

**Формулы**

Цикломатическая сложность C(G) вычисляется тремя способами:

1. подсчёт количества регионов R (под регионом понимается область, ограниченная со всех сторон узлами и дугами; рабочая область должна засчитываться за отдельный регион!);
2. по формуле C(G)=E−V+2, где E ⁠— количество дуг, V ⁠— количество узлов;
3. по формуле C(G)=Vp+1, где Vp ⁠— количество предикатных узлов.

Оценка *Complexity* на веб­‑сайте [http://www.lizard.ws](http://www.lizard.ws/) равна C(G).

Все способы дают одинаковые результаты, поэтому их можно (и нужно!) использовать для проверки.

Сравнить результаты не получится. Только ручной расчет. Подробности смотреть на рисунке 1.

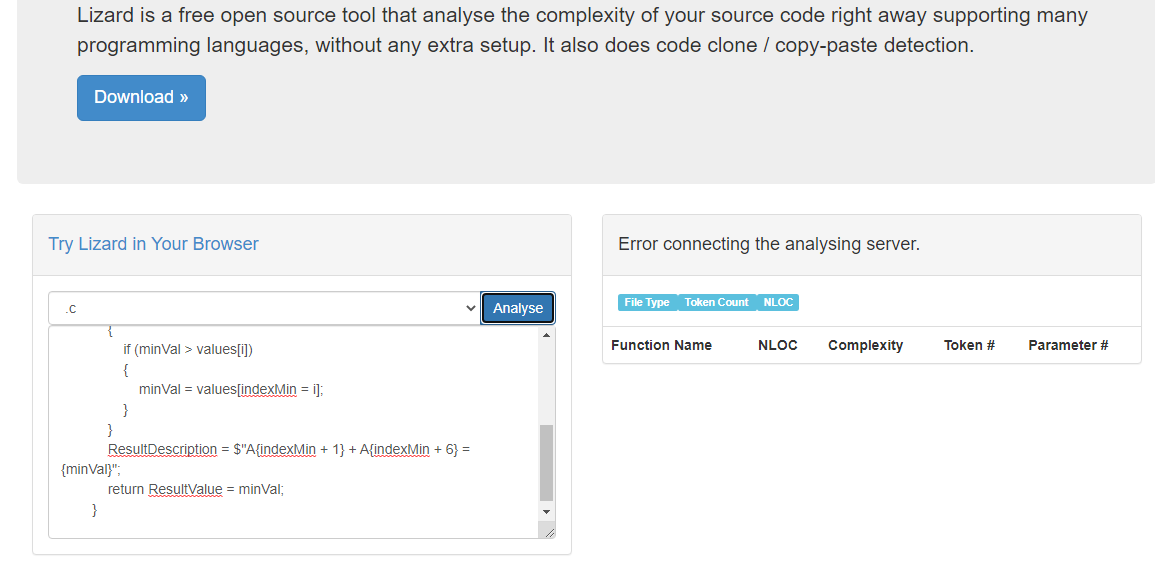


Рисунок 1 – Ошибка аналитического сервера

Взяты 3 метода с именами GetLess, CreateArray, FillArray (листинги имеются ниже).

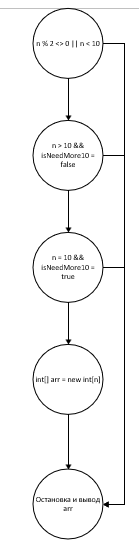


Рисунок 2 – Граф потока управления для CreateArray

R = 4 – количество регионов. E = 7 – количество дуг. V = 5 – количество узлов. Vp = 3 – предикатные узлы

* 1. С(G) = R = 4
  2. C(G) = E – V + 2 = 7 – 5 + 2 = 4
  3. C(G) = Vp + 1 = 3 + 1 = 4

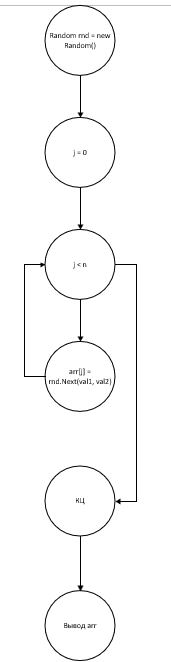


Рисунок 3 – Граф потока управления FillArray

R = 2 – количество регионов. E = 6 – количество дуг. V = 6 – количество узлов. Vp = 1 – предикатные узлы

* 1. С(G) = R = 2
  2. C(G) = E – V + 2 = 6 – 6 + 2 = 2
  3. C(G) = Vp + 1 = 1 + 1 = 2

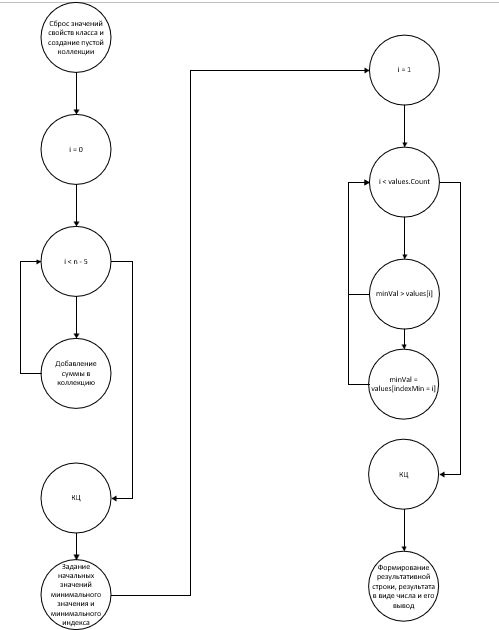


Рисунок 4 – Граф потока управления GetLess

R = 4 – количество регионов. E = 14 – количество дуг. V = 12 – количество узлов. Vp = 3 – предикатные узлы

* 1. С(G) = R = 4
  2. C(G) = E – V + 2 = 14 – 12 + 2 = 4
  3. C(G) = Vp + 1 = 3 + 1 = 4

Было разработано 12 UnitTests. С учетом усовершенствования метода, было необходимо проверить условия, добавленные для этого.

Таблица 1. Юнит-тесты.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Что проверяется | Начальные значения | Ожидаемый результат | Фактический результат | Успешность |
| 1 | Правильность заполнения массива разработанным методом | Расширение массива (больше 10) = false;  Размер = 10 | Массив нулевых значений | Массив нулевых значений | Не успешно |
| 2 | Безошибочное заполнение случайными значениями массива | Массив нулевых значений  Диапазон: от 0 до 16 | Массив случайных значений в указанном диапазоне | Массив случайных значений в указанном диапазоне | Не успешно |
| 3 | Правильное формирование таблицы в DataGrid | Массив нулевых значений  Диапазон: от 0 до 16 | Значения таблицы равны тем, что занесены в массиве | Значения таблицы равны тем, что занесены в массиве | Не успешно |
| 4 | Получение неискаженных значений из DataTable в массив | Заполненная таблица значениями | Значения таблицы равны тем, что занесены в массиве | Значения таблицы равны тем, что занесены в массиве | Не успешно |
| 5 | Проверка правильного нахождения значения минимального А(n) + A(n+5) | localArray[0] = 15;  localArray[1] = 16;  localArray[2] = 17;  localArray[3] = 18;  localArray[4] = 15;  localArray[5] = 16;  localArray[6] = 17;  localArray[7] = 18;  localArray[8] = -5;  localArray[9] = 0; | Вывод минимального значения равного А4+А9(или localArray[3] + localArray[8]) в сформированной строке свойства ResultDescription | Вывод минимального значения равного А4+А9(или localArray[3] + localArray[8]) в сформированной строке свойства ResultDescription | Не успешно |
| 6 | Проверка правильного нахождения значения минимального А(n) + A(n+5) (используется массив нулевых значений) | Массив нулевых значений | Вывод минимального значения в виде А1+А6 | Вывод минимального значения в виде А1+А6 | Не успешно |
| 7 | Проверка правильной смены значения свойств класса LessFinder(результативные) | Аналогично тесту 5 | Вывод минимального значения равного А4+А9(или localArray[3] + localArray[8]) в сформированной строке свойства ResultDescription | Вывод минимального значения равного А4+А9(или localArray[3] + localArray[8]) в сформированной строке свойства ResultDescription | Не успешно |
| 8 | Проверка условия  n > 10 И isNeedMore10 = false | n = 14  isNeedMore10 = false | Перехват ошибки | Перехват ошибки | Не успешно |
| 9 | Проверка условия  n = 10 И isNeedMore10 = true | n = 10  isNeedMore10 = true | Перехват ошибки | Перехват ошибки | Не успешно |
| 10 | Проверка условия  n > 10 И isNeedMore10 = true | n = 14  isNeedMore10 = true | Создание массива нулевых значений | Создание массива нулевых значений | Не успешно |
| 11 | Проверка условия  n % 2 <> 0 Или n < 10) | n = 11  isNeedMore10 = true | Перехват ошибки | Перехват ошибки | Не успешно |
| 12 | Проверка условия  n % 2 <> 0 Или n < 10) | n = 9  isNeedMore10 = false | Перехват ошибки | Перехват ошибки | Не успешно |

**Листинг тестов (класса FirstUnitTest)**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using CreatingArray;

using \_02\_01xPW8\_1xArray;

using System.Diagnostics;

namespace UnitTestForLess

{

[TestClass]

public class FirstUnitTest

{

[TestMethod]

public void \_\_1TestArrayCreate()

{

int[] arr = ArrayCreator.CreateArray(10, false);

TraceTransfer.ToTrace(arr, true);

}

[TestMethod]

public void \_\_2TestArrayFill()

{

localArray = ArrayCreator.FillArray(ArrayCreator.CreateArray(10, false), new ArrayCreator.Range(0, 16));

TraceTransfer.ToTrace(localArray, true);

}

static int[] localArray;

static VisualArray visualArr = new VisualArray();

[TestMethod]

public void \_\_3TestArrayToTable()

{

TraceTransfer.DataGridTraceView(visualArr.CreateTable(localArray), true);

}

[TestMethod]

public void \_\_4TestArrayFromTable()

{

TraceTransfer.DataGridTraceView(visualArr.CreateTable(localArray), true);

TraceTransfer.ToTrace(visualArr.GetOneArray(), true);

}

[TestMethod]

public void \_\_5TestLessFinderWithFill()//Должно быть A(3)4+A(8)9

{

FindLessWithManualValues();

}

[TestMethod]

public void \_\_6TestLessFinderWith0()//Должно быть A(0)1+A(5)6

{

FindLessWith0();

}

[TestMethod]

public void \_\_7FindLessCheckPropertiesClear()

{

FindLessWithManualValues();

FindLessWith0();

FindLessWithManualValues();

}

[TestMethod]

public void \_\_8Check10RestrictionWhenMore10AndFalse()

{

bool ifError = false;

try

{

localArray = ArrayCreator.CreateArray(14, false);

ifError = true;

throw new Exception();

}

catch

{

if (ifError) throw new Exception();

Debug.WriteLine("Переловил исключение. Условие работает исправно!");

}

}

[TestMethod]

public void \_\_9Check10RestrictionWhenNotMore10AndTrue()

{

bool ifError = false;

try

{

localArray = ArrayCreator.CreateArray(10, true);

ifError = true;

throw new Exception();

}

catch

{

if (ifError) throw new Exception();

Debug.WriteLine("Переловил исключение. Условие работает исправно!");

}

}

[TestMethod]

public void \_10Check10RestrictionWhenMore10AndTrue()

{

localArray = ArrayCreator.CreateArray(14, true);

Debug.WriteLine("Условие работает исправно!");

}

[TestMethod]

public void \_11CheckPercent2()

{

bool ifError = false;

try

{

localArray = ArrayCreator.CreateArray(11, true);

ifError = true;

throw new Exception();

}

catch

{

if (ifError) throw new Exception();

Debug.WriteLine("Переловил исключение. Условие работает исправно!");

}

}

[TestMethod]

public void \_12CheckLess10Size()

{

bool ifError = false;

try

{

localArray = ArrayCreator.CreateArray(9, false);

ifError = true;

throw new Exception();

}

catch

{

if (ifError) throw new Exception();

Debug.WriteLine("Переловил исключение. Условие работает исправно!");

}

}

private void FindLessWith0()

{

localArray = ArrayCreator.CreateArray(10, false);

TraceTransfer.ToTrace(localArray, true);

Assert.IsTrue(/\*Будет 0\*/localArray[3] + localArray[8] == LessFinder.GetLess(localArray)/\*Должно быть тоже 0\*/);

TraceTransfer.IsDefaultToReadme = false;

TraceTransfer.ToTrace(LessFinder.ResultDescription);

}

private void FindLessWithManualValues()

{

localArray[0] = 15;

localArray[1] = 16;

localArray[2] = 17;

localArray[3] = 18;

localArray[4] = 15;

localArray[5] = 16;

localArray[6] = 17;

localArray[7] = 18;

localArray[8] = -5;

localArray[9] = 0;

TraceTransfer.ToTrace(localArray, true);

Assert.IsTrue(/\*Будет 13\*/localArray[3] + localArray[8] == LessFinder.GetLess(localArray)/\*Должно быть тоже 13\*/);

TraceTransfer.IsDefaultToReadme = false;

TraceTransfer.ToTrace(LessFinder.ResultDescription);

}

}

}

**Листинг класса, используемого для расширенной работы с трассировкой (класс TraceTransfer)**

using System.Data;

using System.Diagnostics;

using System.IO;

namespace UnitTestForLess

{//Задачи на следующую пару:--------------------------------------------

//Добавить учет true/false для отдельного вывода таблицы

//Добавить свойство для добавления таблицы перед записью в файл

//Оформить ведение таблицы через unit тесты (для примера)

public static class TraceTransfer

{

/// <summary>

/// Счетчик для учета ведения порядкового номера теста

/// </summary>

public static int Counter = 0;

/// <summary>

/// Накопитель информации для вывода результатов в файл (может быть README)

/// </summary>

public static string AllDebug = "";

/// <summary>

/// Используется для добавления информации из трассировки в файл автоматически.

/// Если true - автоматическое занесение в файл.

/// </summary>

public static bool IsDefaultToReadme { get; set; }

/// <summary>

/// Фиксирование в трассировке необходимой информации по проводимому тесту

/// </summary>

public static void ToTrace(int[] arr, bool isOnlyTrace)//isOnlyTrace используется как идентификатор сохранения трассировки в файл

{

if (!isOnlyTrace) StartTest();

TraceAndSave("Представление одномерного массива для целочисленных значений:\n\n");

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

{

TraceAndSave($"{arr[i]} ");

}

TraceAndSave("\n\n");

if (!isOnlyTrace) EndTest();

if (!isOnlyTrace) if (IsDefaultToReadme) ToReadme();

}

/// <summary>

/// Фиксирование в трассировке необходимой информации по проводимому тесту

/// </summary>

public static void ToTrace(double[] arr, bool isOnlyTrace)

{

if (!isOnlyTrace) StartTest();

TraceAndSave("Представление одномерного массива:\n\n");

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

{

TraceAndSave($"{arr[i]} ");

}

TraceAndSave("\n\n");

if (!isOnlyTrace) EndTest();

if (!isOnlyTrace) if (IsDefaultToReadme) ToReadme();

}

/// <summary>

/// Фиксирование в трассировке необходимой информации по проводимому тесту

/// </summary>

public static void ToTrace(double[,] arr, bool isOnlyTrace)

{

if (!isOnlyTrace) StartTest();

TraceAndSave("Представление двумерного массива:\n\n");

for (int i = 0; i < arr.GetLength(0); i++)

{

for (int j = 0; j < arr.GetLength(1); j++)

{

TraceAndSave($"{arr[i, j]} ");

}

TraceAndSave("\n\n");

}

TraceAndSave("\n\n");

if (!isOnlyTrace) EndTest();

if (!isOnlyTrace) if (IsDefaultToReadme) ToReadme();

}

/// <summary>

/// Фиксирование в трассировке необходимой информации по проводимому тесту (чаще, можно использовать для классов)

/// </summary>

/// <param name="allAboutSomething"></param>

public static void ToTrace(string allAboutSomething)

{

StartTest();

TraceAndSave(allAboutSomething + "\n\n");//Для конца строки нужно всегда указывать \n

EndTest();

if (IsDefaultToReadme) ToReadme();

}

/// <summary>

/// Используется для сохранения из AllDebug информации в файл, с указанием пути вручную в коде

/// </summary>

public static void ToReadme()

{

#region Поставить нужный путь

StreamWriter writer = new StreamWriter(@"E:\Совершенствование класса TraceTransfer\PW8P1-V11-Transport\PW8P1-V11-Transport\README.md");//<--

#endregion

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

{

writer.Write(AllDebug[i]);

}

writer.Close();

}

/// <summary>

/// Используется для объявления начала теста

/// </summary>

private static void StartTest()

{

AllDebug += $"Test {++Counter}:\n\n";

Debug.WriteLine("");

}

/// <summary>

/// Используется для объявления окончания теста

/// </summary>

private static void EndTest()

{

AllDebug += $"Test {Counter} окончен\n\n";

Debug.WriteLine("");

}

/// <summary>

/// Используется для сохранения информации в трассировку теста и в переменную сохранения любой строчной информации

/// без объявления о начале или завершении теста

/// </summary>

/// <param name="something"></param>

private static void TraceAndSave(string something)

{

AllDebug += something;

Debug.Write(something);

}

/// <summary>

/// Учет дополнительной информации (используется для unit test'a чаще всего)

/// </summary>

/// <param name="something"></param>

/// <param name="isAdvancedForTest"></param>

public static void TraceAndSave(string something, bool isAdvancedForTest)

{

if (isAdvancedForTest)

{

TraceAndSave("Дополнительно:\n\n" + something + "\n\n");

}

else TraceAndSave(something);

}

/// <summary>

/// Добавление заголовка в README (чаще используется на старте работы)

/// </summary>

/// <param name="header"></param>

public static void GiveHeaderToReadme(string header)

{

AllDebug += "# " + header;

AllDebug += "\n\n";

}

/// <summary>

/// Добавление номеров строк последовательности к строкам

/// </summary>

/// <param name="info"></param>

public static void GiveNumberStrings(string[] info)

{

for (int i = 1; i <= info.Length; i++)

{

AllDebug += i + "." + " " + info.GetValue(i - 1);

AllDebug += "\n";

}

AllDebug += "\n\n";

}

/// <summary>

/// Добавление точек беред каждой строкой (для README)

/// </summary>

/// <param name="info"></param>

public static void GivePointStrings(string[] info)

{

for (int i = 1; i <= info.Length; i++)

{

AllDebug += "\*" + " " + info.GetValue(i - 1);

AllDebug += "\n";

}

AllDebug += "\n\n";

}

/// <summary>

/// Создание таблицы для README

/// </summary>

/// <param name="headers"></param>

/// <param name="values"></param>

public static void GiveStringsToTable(string[] headers, string[] values)

{

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

{

AllDebug += headers.GetValue(i) + "|";

}

AllDebug += "\n";

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

{

AllDebug += "-|";

}

AllDebug += "\n";

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

{

AllDebug += values.GetValue(i) + "|";

if ((i + 1) % headers.Length == 0 && i != 0) AllDebug += "\n";

}

AllDebug += "\n\n";

}

/// <summary>

/// Добавление горизонтального разделителя для заголовка

/// </summary>

public static void GiveHorizontalSeparator()

{

AllDebug += "----------------------------------";

AllDebug += "\n\n";//В конце всегда 2 \n для отделение пустым параграфом

}

//Раскомментировать, если потребуется, так как необходимо для элемента DataTable добавить ссылку в тестовый проект на PresentationFramework.dll

public static void DataGridTraceView(DataTable table, bool isOnlyTrace)

{

if (!isOnlyTrace) StartTest();

TraceAndSave("DataTable представление:\n\n");

for (int i = 0; i < table.Rows.Count; i++)

{

var row = table.Rows[i];

for (int j = 0; j < table.Columns.Count; j++)

{

TraceAndSave($"{row[j]} ");

}

TraceAndSave("\n\n");

}

TraceAndSave("\n\n");

if (!isOnlyTrace) EndTest();

if (!isOnlyTrace) if (IsDefaultToReadme) ToReadme();

}

}

}

**Листинг класса VisualArray**

using System;

using System.Data;

namespace CreatingArray

{

public class VisualArray

{

public DataTable CurrentTable { get; private set; } = new DataTable();

public DataTable CreateTable(int[] arr)

{

CurrentTable = new DataTable();

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

{

CurrentTable.Columns.Add("Col\_" + (i + 1), typeof(string));

}

var row = CurrentTable.NewRow();

for (int j = 0; j < arr.Length; j++)

{

row[j] = arr[j];

}

CurrentTable.Rows.Add(row);

return CurrentTable;

}

public DataTable CreateTable(double[,] arr)

{

CurrentTable = new DataTable();

for (int i = 0; i < arr.GetLength(1); i++)

{

CurrentTable.Columns.Add("Col\_" + (i + 1), typeof(string));

}

for (int i = 0; i < arr.GetLength(0); i++)

{

var row = CurrentTable.NewRow();

for (int j = 0; j < arr.GetLength(1); j++)

{

row[j] = arr[i, j];

}

CurrentTable.Rows.Add(row);

}

return CurrentTable;

}

public double[,] GetArray()

{

var arr = new double[CurrentTable.Rows.Count, CurrentTable.Columns.Count];

for (int i = 0; i < CurrentTable.Rows.Count; i++)

{

var row = CurrentTable.Rows[i];

for (int j = 0; j < CurrentTable.Columns.Count; j++)

{

arr[i, j] = Convert.ToDouble(row[j]);

}

}

return arr;

}

public int[] GetOneArray()

{

var arr = new int[CurrentTable.Columns.Count];

var row = CurrentTable.Rows[0];

for (int j = 0; j < CurrentTable.Columns.Count; j++)

{

arr[j] = Convert.ToInt32(row[j]);

}

return arr;

}

public void EditCell(int i, int j, double value)

{

var row = CurrentTable.Rows[i];//ПРОВЕРИТЬ АКТУАЛЬНОСТЬ РАБОТЫ!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

row[j] = value;

}

public void ClearTable()

{

CurrentTable = new DataTable();

}

}

}

**Листинг класса ArrayCreator**

using System;

namespace CreatingArray

{

public static class ArrayCreator

{

public struct Range

{

private int \_secondValue;

public Range(int firstValue, int secondValue)

{

FirstValue = firstValue;

\_secondValue = secondValue;

}

public int FirstValue { get; set; }

public int SecondValue

{

get => \_secondValue;

set

{

if (FirstValue > value) throw new Exception("Второе значение диапазона больше или равно первому!");

\_secondValue = value;

}

}

}

/// <summary>

/// Создает двумерный массив данных, заполненный нулями.

/// </summary>

/// <param name="n"></param>

/// <param name="m"></param>

/// <returns></returns>

public static double[,] CreateArray(int n, int m)

{

double[,] arr = new double[n, m];

return arr;

}

public static int[] CreateArray(int n, bool isNeedMore10)

{

if (n % 2 != 0 || n < 10) throw new Exception();

if (n > 10 && !isNeedMore10) throw new Exception();

if (n == 10 && isNeedMore10) throw new Exception();

int[] arr = new int[n];

return arr;

}

public static double[,] CreateArray(int n, int m, Range range)

{

var arr = CreateArray(n, m);

Random rnd = new Random();

for (int i = 0; i < arr.GetLength(0); i++)

{

for (int j = 0; j < arr.GetLength(1); j++)

{

arr[i, j] = rnd.Next(range.FirstValue, range.SecondValue);

}

}

return arr;

}

public static double[,] FillArray(double[,] arr, Range range)

{

Random rnd = new Random();

for (int i = 0; i < arr.GetLength(0); i++)

{

for (int j = 0; j < arr.GetLength(1); j++)

{

arr[i, j] = rnd.Next(range.FirstValue, range.SecondValue);

}

}

return arr;

}

public static int[] FillArray(int[] arr, Range range)

{

Random rnd = new Random();

for (int j = 0; j < arr.Length; j++)

{

arr[j] = rnd.Next(range.FirstValue, range.SecondValue);

}

return arr;

}

}

}

**Листинг задачного класса по варианту (класс LessFinder)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_02\_01xPW8\_1xArray

{

public static class LessFinder

{

public static string ResultDescription { get; set; }

public static int ResultValue { get; set; }

public static int GetLess(int[] arr)

{

ResultValue = 0;

ResultDescription = "";

List<int> values = new List<int>();

for (int i = 0; i < arr.Length - 5; i++)

{

values.Add(arr[i] + arr[i + 5]);

}

int indexMin = 0, minVal = values[0];

for (int i = 1; i < values.Count; i++)

{

if (minVal > values[i])

{

minVal = values[indexMin = i];

}

}

ResultDescription = $"A{indexMin + 1} + A{indexMin + 6} = {minVal}";

return ResultValue = minVal;

}

}

}