

**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **«СИНЕРГИЯ»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | 09.02.07 Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

.

**Отчет по лабораторной работе № 4**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **на тему** | |  | Массивы и строки | | | | |
|  | | | | | |  | (наименование темы) |
|  |  | |  | | | | |
| **по дисциплине** | | | |  | Разработка программных модулей | | |
|  | | | | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Чилингиров Нури Эскендерович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП-112 Прог |  |
|  |  |  |  |

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

**Москва, 2025**

**7** Вариант

Написать программу формирования ОПЗ и расчета полученного выражения. Разработать удобный интерфейс ввода исходных данных и вывода результатов. Работу программы проверить на конкретном примере

Листинг кода: using System;

using System.Collections.Generic;

using System.Globalization;

class Program

{

static void Main()

{

Console.WriteLine("Программа преобразования выражения в обратную польскую запись (ОПЗ) и его вычисления");

Console.WriteLine("Пример выражения: a\*(b-c)/(d+e)");

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

// Тестовые данные

Dictionary<string, double> variables = new Dictionary<string, double>

{

{"a", 1.6},

{"b", 4.9},

{"c", 5.7},

{"d", 0.8},

{"e", 2.3}

};

// Вывод тестовых данных

Console.WriteLine("\nТестовые значения переменных:");

foreach (var kvp in variables)

{

Console.WriteLine($"{kvp.Key} = {kvp.Value}");

}

string expression = "a\*(b-c)/(d+e)";

Console.WriteLine($"\nИсходное выражение: {expression}");

// Преобразование в ОПЗ

List<string> rpn = ConvertToRPN(expression);

Console.WriteLine("\nОбратная польская запись: " + string.Join(" ", rpn));

// Вычисление выражения

double result = CalculateRPN(rpn, variables);

Console.WriteLine($"\nРезультат вычисления: {result:F3}");

// Проверка ожидаемого результата

Console.WriteLine($"Ожидаемый результат: -0.413");

}

// Преобразование выражения в ОПЗ с использованием алгоритма сортировочной станции

static List<string> ConvertToRPN(string expression)

{

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

Stack<string> operators = new Stack<string>();

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

{

char c = expression[i];

if (char.IsWhiteSpace(c)) continue;

if (char.IsLetter(c))

{

// Считываем имя переменной (может быть длиннее одного символа)

string variable = c.ToString();

while (i + 1 < expression.Length && char.IsLetterOrDigit(expression[i + 1]))

{

variable += expression[++i];

}

output.Add(variable);

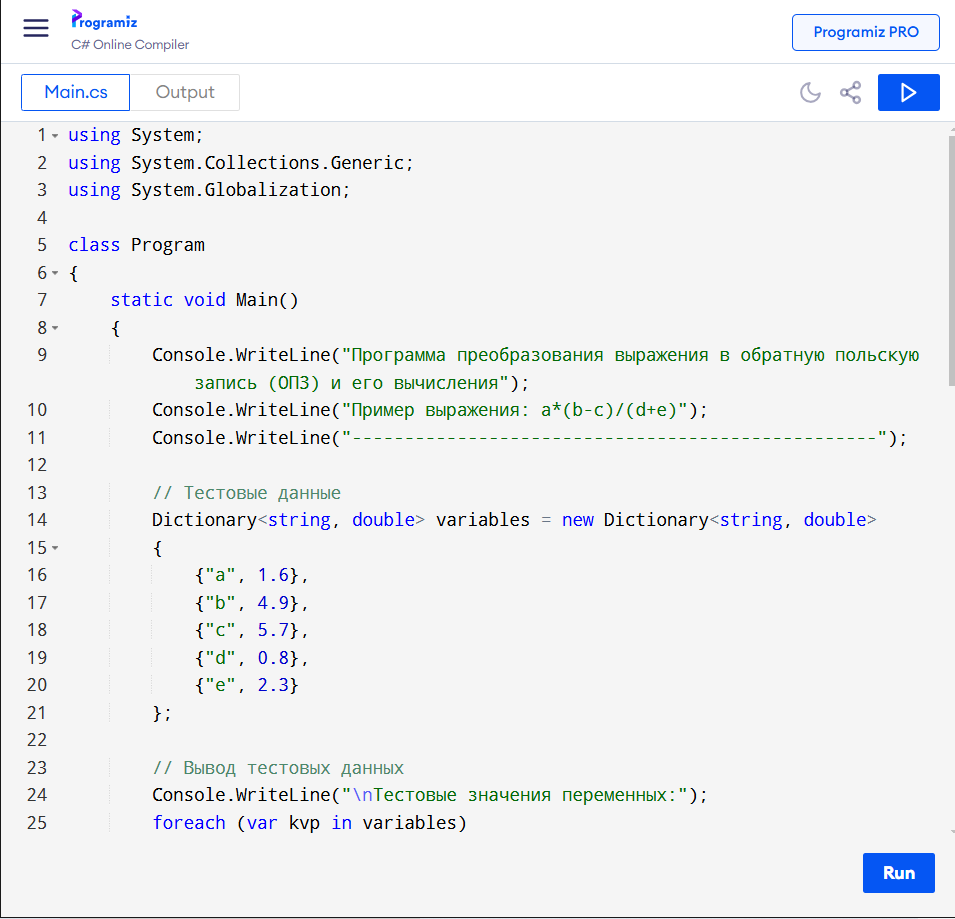
}

else if (char.IsDigit(c) || c == '.')

{

// С

Скрин кода:



2. Определить количество символов во всех строках дерева.

Листинг кода:

using System;

using System.Collections.Generic;

// Класс для представления узла дерева

class TreeNode

{

public string Value { get; set; }

public List<TreeNode> Children { get; set; }

public TreeNode(string value)

{

Value = value;

Children = new List<TreeNode>();

}

// Метод для добавления дочернего узла

public void AddChild(TreeNode child)

{

Children.Add(child);

}

}

class Program

{

static void Main()

{

Console.WriteLine("Программа для подсчета количества символов во всех строках дерева");

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

// Создаем тестовое дерево

TreeNode root = new TreeNode("Корень");

TreeNode child1 = new TreeNode("Ветка 1");

child1.AddChild(new TreeNode("Лист 1.1"));

child1.AddChild(new TreeNode("Лист 1.2"));

TreeNode child2 = new TreeNode("Ветка 2");

child2.AddChild(new TreeNode("Лист 2.1"));

child2.AddChild(new TreeNode("Лист 2.2"));

child2.AddChild(new TreeNode("Лист 2.3"));

root.AddChild(child1);

root.AddChild(child2);

// Вычисляем общее количество символов

int totalChars = CountCharactersInTree(root);

Console.WriteLine($"\nОбщее количество символов во всех строках дерева: {totalChars}");

// Дополнительный вывод: количество символов в каждом узле

Console.WriteLine("\nКоличество символов по узлам:");

PrintNodeCharacters(root, 0);

}

// Рекурсивный метод для подсчета символов во всем дереве

static int CountCharactersInTree(TreeNode node)

{

if (node == null) return 0;

int count = node.Value.Length;

foreach (var child in node.Children)

{

count += CountCharactersInTree(child);

}

return count;

}

// Вспомогательный метод для вывода информации о каждом узле

static void PrintNodeCharacters(TreeNode node, int level)

{

if (node == null) return;

string indent = new string(' ', level \* 2);

Console.WriteLine($"{indent}{node.Value} - {node.Value.Length} символов");

foreach (var child in node.Children)

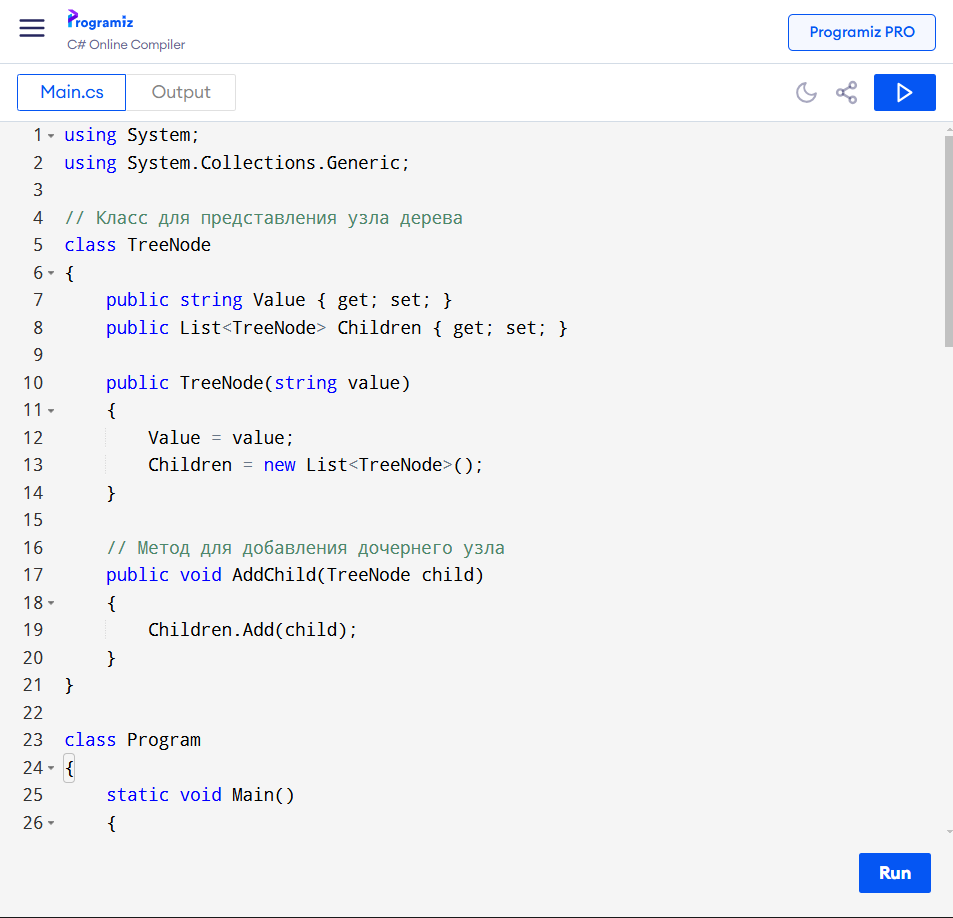
{

PrintNodeCharacters(child, level + 1);

}

}

}

Скрин кода: 

3. В вариантах, использующих итерационные методы, построить график зависимости количества итераций it, необходимых для достижения заданной точности, от параметра релаксации ω и установить, при каком значении ω число итераций минимально. Параметр релаксации менять от 0.2 до 2 с шагом 0.2.

Листинг кода:

using System;

using System.Collections.Generic;

using System.Linq;

class RelaxationMethodAnalysis

{

static void Main()

{

Console.WriteLine("Анализ зависимости количества итераций от параметра релаксации ω");

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

// Параметры задачи (из примера)

double[,] matrix = { {7, -4}, {-2.75, 7} };

double[] vector = { 3, 5 };

double[] initialGuess = { 0, 0 };

double tolerance = 1e-6;

int maxIterations = 1000;

// Диапазон параметра релаксации

double[] omegaValues = Enumerable.Range(1, 10).Select(i => i \* 0.2).ToArray();

// Список для хранения результатов

List<(double omega, int iterations)> results = new List<(double, int)>();

// Анализ для каждого значения ω

foreach (double omega in omegaValues)

{

int iterations = SolveWithRelaxation(matrix, vector, initialGuess, omega, tolerance, maxIterations);

results.Add((omega, iterations));

Console.WriteLine($"ω = {omega:F1}, итераций = {iterations}");

}

// Находим оптимальное значение ω

var optimal = results.OrderBy(r => r.iterations).First();

Console.WriteLine($"\nОптимальное значение ω = {optimal.omega:F1} (минимальное количество итераций = {optimal.iterations})");

// Генерация данных для построения графика

Console.WriteLine("\nДанные для построения графика:");

Console.WriteLine("ω\tИтерации");

foreach (var result in results)

{

Console.WriteLine($"{result.omega:F1}\t{result.iterations}");

}

}

static int SolveWithRelaxation(double[,] A, double[] b, double[] x0, double omega, double tolerance, int maxIterations)

{

int n = b.Length;

double[] x = new double[n];

Array.Copy(x0, x, n);

int iterations = 0;

while (iterations < maxIterations)

{

double maxDiff = 0;

double[] xNew = new double[n];

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

{

double sum = b[i];

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

{

if (j != i) sum -= A[i, j] \* x[j];

}

xNew[i] = (1 - omega) \* x[i] + omega \* (sum / A[i, i]);

maxDiff = Math.Max(maxDiff, Math.Abs(xNew[i] - x[i]));

}

x = xNew;

iterations++;

if (maxDiff < tolerance) break;

}

return iterations;

}

}

Скрин кода: 