МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет ИТМО

Лабораторная работа №2

«Алгоритмы с бинарными деревьями.»

Цель работы: Изучить алгоритмы работы с деревьями. Реализовать средствами ООП дерево и рекурсивные функции, выполняющие обход дерева в прямом, обратном, концевом порядке. Вычислить значение выражения, заданного деревом.

Выполнила студентка группы №K3222

Маркозубова Анастасия Кирилловна

Задание

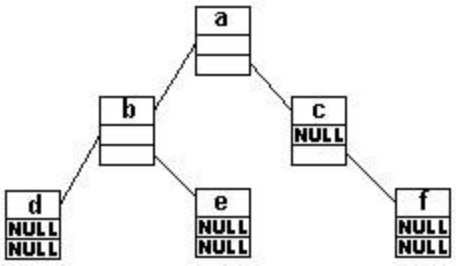
1. Реализовать средствами ООП дерево и рекурсивные функции, выполняющие обход дерева в прямом, обратном и концевом порядке.

   Выполнить расчет примера:

   Обход дерева в прямом порядке a b d e c d

   Обход дерева в обратном порядке d b e a c f

   Обход дерева в концевом порядке d e b f c a



2. Вычислить значение выражения, заданного деревом. Вычисление выполнять в порядке концевого обхода.

Задание 1. Реализовать средствами ООП дерево и рекурсивные функции, выполняющие обход дерева в прямом, обратном и концевом порядке

Tree.cs

```
using System;
using System.Collections.Generic;

namespace Lab2
{
    internal class Tree
    {
        public Tree Left;
        public Tree Right;
        public char Value;

        public Tree(char val)
        {
            Value = val;
            Left = null;
            Right = null;
        }

        public static Tree BuildTree(ref int index, List<char> nodes)
        {
            if (index >= nodes.Count || nodes[index] == '.')
            {
                index++;
                return null;
            }

            Tree newNode = new Tree(nodes[index]);
            index++;

            newNode.Left = BuildTree(ref index, nodes);
            newNode.Right = BuildTree(ref index, nodes);

            return newNode;
        }

        public static void TreeWalk_Pr(Tree node, List<char> result)
        {
            if (node == null) return;
            result.Add(node.Value);
            TreeWalk_Pr(node.Left, result);
            TreeWalk_Pr(node.Right, result);
        }

        public static void TreeWalk_Obr(Tree node, List<char> result)
        {
            if (node == null) return;
            TreeWalk_Obr(node.Left, result);
            result.Add(node.Value);
            TreeWalk_Obr(node.Right, result);
        }

        public static void TreeWalk_K(Tree node, List<char> result)
        {
            if (node == null) return;
            TreeWalk_K(node.Left, result);
            TreeWalk_K(node.Right, result);
            result.Add(node.Value);
        }

        public static void PrintTree(Tree node, string indent = "", bool last
```

```
= true)
        {
            if (node != null)
            {
                Console.Write(indent);
                Console.Write(last ? "└─" : "├─");
                Console.WriteLine(node.Value);

                indent += last ? "  " : "| ";

                PrintTree(node.Left, indent, false);
                PrintTree(node.Right, indent, true);
            }
        }
    }
}
```

Programm.cs

```
using System;
using System.Collections.Generic;

namespace Lab2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<char> chars = new List<char> { 'a', 'b', 'd', '.', '.', 'e',
'.', '.', 'c', 'f', '.', '.', 'g', '.', '.', };
            int index = 0;

            Tree root = Tree.BuildTree(ref index, chars);
            Console.WriteLine("Структура дерева:");
            Tree.PrintTree(root);

            Console.WriteLine("Прямой порядок (NLR):");
            List<char> preOrder = new List<char>();
            Tree.TreeWalk_Pr(root, preOrder);
            Console.WriteLine(string.Join(" ", preOrder));

            Console.WriteLine("Обратный порядок (LNR):");
            List<char> inOrder = new List<char>();
            Tree.TreeWalk_Obr(root, inOrder);
            Console.WriteLine(string.Join(" ", inOrder));

            Console.WriteLine("Кончиковый порядок (LRN):");
            List<char> postOrder = new List<char>();
            Tree.TreeWalk_K(root, postOrder);
            Console.WriteLine(string.Join(" ", postOrder));

            Console.ReadLine();
        }
    }
}
```

```
Структура дерева:
└─a
  ├─b
  │ ├─d
  │ └─e
  └─c
    ├─f
    └─g
Прямой порядок (NLR):
a b d e c f g
Обратный порядок (LNR):
d b e a f c g
Кончиковый порядок (LRN):
d e b f g c a
```

Задание 2. Вычислить значение выражения, заданного деревом. Вычисление выполнять в порядке концевого обхода.

Добавленный код в Tree.cs

```
public static int CalcTree(Tree node)
{
    if (char.IsDigit(node.Value)) return node.Value - '0';

    int num1 = CalcTree(node.Left);
    int num2 = CalcTree(node.Right);

    switch (node.Value)
    {
        case '+': return num1 + num2;
        case '-': return num1 - num2;
        case '*': return num1 * num2;
        case '/': return num1 / num2;
        default: return 0;
    }
}
```

Добавленный код в Program.cs

```
List<char> expressionChars = new List<char>
{
    '/', '*', '+', '2', '.', '.', '3', '.', '.', '-', '7', '.', '.', '4',
'.', '.', '3', '.', '.'
};
index = 0;
Tree expressionTree = Tree.BuildTree(ref index, expressionChars);
```

```
Дерево выражения:
└─/
  ├─*
  │ ├─+
  │ │ ├─2
  │ │ └─3
  │ └─-
  │   ├─7
  │   └─4
  └─3
Кончиковый порядок (LRN) дерева выражения:
2 3 + 7 4 - * 3 /
```

Заключение

В рамках лабораторной работы были изучены и реализованы основные алгоритмы обхода бинарного дерева: прямой (NLR), обратный (LNR) и концевой (LRN). Эти методы позволяют эффективно анализировать структуру дерева и получать данные в нужной последовательности. Дополнительно был реализован алгоритм вычисления арифметического выражения, представленного в виде дерева, с использованием концевого обхода. Работа продемонстрировала универсальность деревьев и их практическое применение в задачах обработки выражений.