Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Тульский государственный университет»

КАФЕДРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

**СИЛЬНОВЕТВЯЩИЕСЯ ДЕРЕВЬЯ**

отчет о   
лабораторной работе №9

по дисциплине

*ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ*

***ВАРИАНТ 7***

|  |  |  |
| --- | --- | --- |
| Выполнила: | ст. гр. 230711 | Павлова В.С. |
| Проверил: | асс. каф. ИБ | Курбаков М.Ю. |

Тула, 2022 г.

# **ЦЕЛЬ И ЗАДАЧА РАБОТЫ**

**Цель:** ознакомиться с понятием сильноветвящихся деревьев, изучение методов их представления, создания, работы, изменения вниз-направленных сильноветвящихся деревьев.

**Задача:** в данной работе требуется написать программу, демонстрирующую использование изученных принципов.

# **ЗАДАНИЕ НА РАБОТУ**

Дано выражение ((1 – P)\*A+B/(C – D)). Представить его в виде дерева.

# **СХЕМА ПРОГРАММЫ**

Схема алгоритма программы представлена на рисунке 1.



Рисунок 1 – Схема алгоритма программы

Схема алгоритмов подпрограмм представлены на рисунке 2.

|  |  |
| --- | --- |
|  |  |
| Рисунок 2. а – Схема алгоритма подпрограммы создания дерева | Рисунок 2.б – Схема алгоритма подпрограммы создания постфиксной записи |

# **ТЕКСТ ПРОГРАММЫ**

Текст программы на языке программирования С++ для представления выражения ((1 – P)\*A + B/(C – D)) в виде дерева представлен в листинге 1.

## **Листинг 1. Текст программы**

#include <iostream>

#include <string>

#include <stack>

using namespace std;

struct Node //cтруктура данных для хранения узла бинарного дерева

## **Листинг 1. Текст программы (продолжение)**

{

char data;

Node\* left, \* right;

Node(char newData)

{

data = newData;

left = right = nullptr;

};

Node(char newData, Node\* newLeft, Node\* newRight)

{

data = newData;

left = newLeft;

right = newRight;

};

};

bool isOperator(char c)

{

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

bool isBracket(char c)

{

return (c == '(' || (c == ')'));

}

void printTree(string prefix, const Node\* node, bool isLeft)

{

if (node != nullptr)

{

cout << prefix;

cout << (isLeft ? "|--" : "L--");

cout << node->data << "\n";

printTree(prefix + (isLeft ? "| " : " "), node->right, true);

printTree(prefix + (isLeft ? "| " : " "), node->left, false);

}

}

void printTree(const Node\* node)

{

printTree("", node, false);

}

Node\* makeTree(string postfix)

{

stack <Node\*> s;

for (char c: postfix)

{

if (isOperator(c))

{

Node\* x = s.top();

s.pop();

Node\* y = s.top();

s.pop();

## **Листинг 1. Текст программы (продолжение)**

//построить новое бинарное дерево, корнем которого является оператор

//левый и правый дочерние элементы указывают на `y` и `x` соответственно

Node\* node = new Node(c, y, x);

s.push(node); //поместить текущий узел в стек

}

else

s.push(new Node(c)); //создать новый узел бинарного дерева

//дерева, корень – данный операнд

}

return s.top();

}

string makePostfix(string s) //образовать постфиксную запись из данной

{

stack <char> stackOps;

string postfix = "";

Node\* a, \* b;

Node\* c = nullptr;

for (int i = 0; i < s.length(); i++)

{

if ((!isBracket(s[i])) && (!isOperator(s[i])))

postfix += s[i];

if (isOperator(s[i]))

stackOps.push(s[i]);

if (s[i] == ')')

{

postfix += stackOps.top();

stackOps.pop();

}

}

return postfix;

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

string s = "(((1-P)\*A)+(B/(C-D)))";

string postfix = makePostfix(s);

Node\* tree = makeTree(postfix);

printTree(tree);

return 0;

}

# **ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ**

Данная программа предназначена для того, чтобы из выражения ((1 – P)\*A + B/(C – D)) получать бинарное дерево. Вводить в программу ничего не требуется, поскольку выражение уже задано. Программа производит необходимые вычисления и выводит полученное дерево.

# **ИНСТРУКЦИЯ ПРОГРАММИСТА**

Структуры данных, используемые в программе, приведены в таблицах 1-2.

Таблица 1 – Структуры данных в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип (класс)** | **Предназначение** |
| s | string | Данное выражение |
| postfix | string | Постфиксная запись выражения |
| tree | Node\* | Полученное дерево |

Для хранения узла бинарного дерева была создана вспомогательная структура Node, описание которой представлено в таблице 2.

Таблица 2 – Описание разработанной структуры Node

|  |  |
| --- | --- |
| struct Node | |
| **Поля/свойства (элементы данных) структуры** | |
| Название и тип | Описание |
| char data | Значение, которое хранится в узле |
| Node\* left | Ссылка на левое поддерево |
| Node\* right | Ссылка на правое поддерево |

# **ДЕМОНСТРАЦИОННЫЙ ПРИМЕР**

Дано выражение ((1 – P)\*A + B/(C – D)). Необходимо представить его в виде дерева. Запишем в виде таблицы алгоритм для преобразования инфиксной записи в постфиксную, с помощью которой в дальнейшем будет образовано дерево.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Шаг | Элемент | Действие | Стек | Вывод |
| 1 | 1 | print |  | 1 |
| 2 | – | push | – | 1 |
| 3 | P | print | – | 1P |
| 4 | ) | pop |  | 1P – |
| 5 | \* | push | \* | 1P – |
| 6 | A | print | \* | 1P –A |
| 7 | ) | pop |  | 1P –A\* |
| 8 | + | push | + | 1P –A\*B |
| 9 | B | print | + | 1P –A\*B |
| 10 | / | push | + / | 1P –A\*BC |
| 11 | C | print | + / | 1P –A\*BC |
| 12 | – | push | + / – | 1P –A\*BC |
| 13 | D | print | + / – | 1P –A\*BCD |
| 14 | ) | pop |  | 1P –A\*BCD–/+ |

Из полученной инфиксной записи «1P–A\*BCD–/+» можно построить бинарное дерево следующим образом: оператор «+» имеет два операнда, и первый из них – это результат деления «/» между B и разностью С–D, а второй – результат умножения «\*» A на разность (1–Р). Следовательно «+» будет корнем, из которого исходят две ветви, содержащие в себе результаты операций «\*» и «/». Аналогично получаются и другие поддеревья. Таким образом, дерево для исходного выражения имеет следующий вид (рисунок 3):



Рисунок 3 – Полученное двоичное дерево

Результат работы программы (рисунок 4) для данного набора входных данных соответствует полученному теоретически.

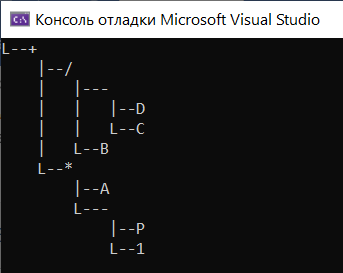


Рисунок 4 – Результат работы программы

# **ВЫВОДЫ**

В ходе данной лабораторной работы был изучен принцип работы с деревьями. Для демонстрации полученных знаний была написана программа, создающая дерево на основе входного выражения. По результатам проверки программы можно сделать вывод о том, что она работает корректно.