МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 9303 _____ Махаличев Н.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятиями «дерево» и «бинарное дерево», его особенностями и свойствами. Реализовать программу для работы с бинарным деревом согласно требованиям задания на языке C++.

Задание.

Вариант 13 (д).

- для заданной формулы *f* построить дерево-формулу *t*;

- для заданного дерева-формулы *t* напечатать соответствующую формулу *f*;

- построить дерево-формулу *t* из строки, задающей формулу в постфиксной форме (перечисление узлов *t* в порядке ЛПК);

- упростить дерево-формулу *t*, выполнив в нем все операции вычитания, в которых уменьшаемое и вычитаемое – цифры. Результат вычитания – цифра или формула вида (*0* – цифра).

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., T_m называются поддеревьями данного дерева.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом.

Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом

Выполнение работы.

Создан класс бинарного дерева BinTree, содержащий методы и поля для работы с вводимыми данными. Класс BinTree содержит следующие поля:

- BinTree *left – левое поддерево текущего узла;
- BinTree *right – правое поддерево текущего узла;
- char element – значение текущего узла;
- string operands – строка, содержащая допустимые значения узлов дерева;
- string operators – строка, содержащая допустимые операторы дерева;
- string numbers – строка, содержащая все цифры.

Также в классе BinTree определены следующие методы:

- bool IsNull(BinTree *tree) – проверка на существование дерева;
- BinTree *Left(BinTree *tree) – возвращает указатель на левое поддерево текущего узла;
- BinTree *Right(BinTree *tree) – возвращает указатель на правое поддерево текущего узла;
- char RootElement(BinTree *tree) – возвращает значение текущего узла;
- void MakeElement(BinTree *tree, char element, BinTree *left, BinTree *right) – создание узла;
- void FunctionRead(BinTree *tree, string &line, int from, int to) – создание бинарного дерева из инфиксного представления;
- void ReadTree(BinTree *tree, string &line) – создание бинарного дерева из префиксного представления;
- void Destroy(BinTree *tree) – удаление и очистка памяти созданного дерева

- void Display(BinTree *tree, int n) – данный метод выводит хранящееся в памяти бинарное дерево;
- void PostfixPrint(BinTree *tree) – вывод хранящегося в памяти дерева в терминал в постфиксной форме
- void FunctionPrint(BinTree *tree, string &res) – представляет бинарное дерево в инфиксной форме
- void Diff(BinTree *tree) – метод, выполняющий вычитание правого элемента из левого, если они – цифры. Если в результате вычислений получается положительное число, оно записывается в узел оператора, а левое и правое поддерево удаляется. Если получилось отрицательное число, то в значение левого узла записывается 0, а правого полученная разность.

Определены функции string PostfixToPrefix(string postfix), преобразующая введённое постфиксное представление дерева в префиксное, и string ClearSymbols(string line), очищающая строку от пробелов и знаков табуляции.

Полученный программный код см. в приложении А. Тестирование программы представлено в приложении Б.

Выводы.

В процессе выполнения лабораторной работы было изучены понятия «дерево» и «бинарное дерево», их свойства и приёмы.

Была разработана программа, создающая бинарное дерево из введённой постфиксной или инфиксной формулы, и выполняющая его обработку согласно заданию.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include "../Headers/BinTree.h"
#include <algorithm>
using namespace std;

string PostfixToPrefix(string postfix){
    string stack[50];
    int count = 0;
    string answer = "";
    string operators = "+-*";
    for (int i = 0; i < postfix.length(); i++){
        if (operators.find(postfix[i]) != -1){
            if (count >= 1){
                string op1 = stack[--count];
                stack[count--] = "";
                string op2 = stack[count];
                stack[count++] = postfix[i] + op2 + op1;
            } else {
                cerr << "Error: Wrong size" << endl;
                exit(1);
            }
        } else {
            stack[count++] = postfix[i];
        }
    }
    for (int i = 0; i < count; i++){
        answer += stack[i];
    }
    return answer;
}

string ClearSymbols(string line){
    for (int i = 0; i < line.length(); i++){
        if ((line[i] == ' ') || (line[i] == '\t')) {
            line.erase(i, 1);
            i--;
```

```
        }
    }
    return line;
}


int main(){
    string input;
    ifstream inputfile;
    ofstream outputfile;
    inputfile.open("input.txt");
    outputfile.open("output.txt", std::fstream::out | std::fstream::app);
    if (inputfile.is_open() && outputfile.is_open()){
        while(getline(inputfile, input)){
            cout << "Checking - " << input << endl;
            BinTree *bintree = new BinTree;
            if (input[0] == '('){
                string line = ClearSymbols(input);
                cout << line << endl;
                bintree->FunctionRead(bintree, line, 0, line.length()-1);
            } else {
                string line = PostfixToPrefix(input);
                reverse(line.begin(), line.end());
                bintree->ReadTree(bintree, line);
            }
            bintree->Display(bintree, 0);
            bintree->Diff(bintree);
            cout << "Postfix output - ";
            bintree->PostfixPrint(bintree);
            cout << endl;
            string res;
            cout << "Infix output - ";
            bintree->FunctionPrint(bintree, res);
            cout << res << endl;
            outputfile << res << endl;
            bintree->Destroy(bintree);
        }
        inputfile.close();
```

```
        outputfile.close();
    } else {
        cout << "Can't open file" << endl;
    }
    return 0;
}
```

Файл BinTree.h

```
#ifndef BINTREE_H
#define BINTREE_H

#include <iostream>
#include <fstream>

using namespace std;

class BinTree{
public:
    BinTree();
    BinTree *left;
    BinTree *right;
    char element;
    bool IsNull(BinTree *tree);
    BinTree *Left(BinTree *tree);
    BinTree *Right(BinTree *tree);
    char RootElement(BinTree *tree);
    void MakeElement(BinTree *tree, char element, BinTree *left, BinTree
*right);
    void FunctionRead(BinTree *tree, string &line, int from, int to);
    void ReadTree(BinTree *tree, string &line);
    void Destroy(BinTree *tree);
    void Display(BinTree *tree, int n);
    void PostfixPrint(BinTree *tree);
    void FunctionPrint(BinTree *tree, string &res);
    void Diff(BinTree *tree);
private:
    string operands = "abcdefghijklmnopqrstuvwxyz01234567890";
    string operators = "+-*";
    string numbers = "0123456789";
};

#endif
```

Файл BinTree.cpp

```
#include "../Headers/BinTree.h"


BinTree::BinTree(): left(nullptr), right(nullptr){};


bool BinTree::IsNull(BinTree *tree){
```

```
    return tree == nullptr;
}

BinTree *BinTree::Left(BinTree *tree){
    if (tree == nullptr){
        cerr << "Error: Left(nullptr)\n";
        exit(1);
    }
    return tree->left;
}

BinTree *BinTree::Right(BinTree *tree){
    if (tree == nullptr){
        cerr << "Error: Right(nullptr)\n";
        exit(1);
    }
    return tree->right;
}

char BinTree::RootElement(BinTree *tree){
    if (tree == nullptr){
        cerr << "Error: Root(nullptr)\n";
        exit(1);
    }
    return tree->element;
}

void BinTree::MakeElement(BinTree *tree, char element, BinTree *left,
BinTree *right){
    tree->element = element;
    tree->left = left;
    tree->right = right;
}

void BinTree::FunctionRead(BinTree *tree, string &line, int from, int
to){
    int deep = 0;
    if (from == to){
```

```
        if ((operands.find(line[from]) != -1)){
            MakeElement(tree, line[from], nullptr, nullptr);
        } else {
            cerr << "Error: Wrong element - " << line[from] << endl;
            exit(1);
        }
        return;
    }
    for (int i = from; i<=to; i++){
        if (line[i] == '(')
            deep++;
        if (line[i] == ')')
            deep--;
        if ((operators.find(line[i]) != -1) && (deep == 1)){
            if ((line[i-1] == '(') || (line[i+1] == ')')){
                cerr << "Error: Wrong function" << endl;
                exit(1);
            }
            BinTree *p = new BinTree;
            BinTree *q = new BinTree;
            FunctionRead(p, line, from+1, i-1);
            FunctionRead(q, line, i+1, to-1);
            MakeElement(tree, line[i], p, q);
        }
    }
}

void BinTree::ReadTree(BinTree *tree, string &line){
    if (line.length()){
        char element;
        element = line[line.length()-1];
        line.pop_back();
        if (operators.find(element) != -1){
            BinTree *p = new BinTree;
            BinTree *q = new BinTree;
            ReadTree(p, line);
            ReadTree(q, line);
            MakeElement(tree, element, p, q);
```

```
        } else {
            if (operands.find(element) != -1){
                tree->element = element;
            } else {
                cerr << "Error: Wrong element - " << element << endl;
                exit(1);
            }
        }
    } else {
        cerr << "Error: Wrong size" << endl;
        exit(1);
    }
}

void BinTree::Destroy(BinTree *tree){
    if (!IsNull(Left(tree))){
        Destroy(tree->left);
    }
    if (!IsNull(Right(tree))){
        Destroy(tree->right);
    }
    delete tree;
}

void BinTree::Display(BinTree *tree, int n){
    if (tree != nullptr){
        for (int i = 0; i < n; i++){
            cout << "  ";
        }
        cout << RootElement(tree) << endl;
        if (!IsNull(Right(tree))){
            Display(Right(tree), n+1);
        }
        if (!IsNull(Left(tree))){
            Display(Left(tree), n+1);
        }
    }
}
```

```
void BinTree::PostfixPrint(BinTree *tree){
    if (!IsNull(tree)){
        PostfixPrint(Left(tree));
        PostfixPrint(Right(tree));
        cout << RootElement(tree);
    }
}

void BinTree::FunctionPrint(BinTree *tree, string &res){
    if (Left(tree) && Right(tree)){
        res.push_back('(');
        FunctionPrint(Left(tree), res);
        res.push_back(tree->element);
        FunctionPrint(Right(tree), res);
        res.push_back(')');
    } else {
        res.push_back(tree->element);
    }
}

void BinTree::Diff(BinTree *tree){
    if (!IsNull(tree)){
        Diff(Left(tree));
        Diff(Right(tree));
        if (RootElement(tree) == '-') {
            if   ((numbers.find(RootElement(Right(tree)))   !=   -1)   &&
(numbers.find(RootElement(Left(tree))) != -1)){
                int diff, op1, op2;
                op1 = (int)RootElement(Left(tree)) - '0';
                op2 = (int)RootElement(Right(tree)) - '0';
                cout << "Calculating: " << RootElement(Left(tree)) << '-'
<< RootElement(Right(tree)) << endl;
                if (op1 >= op2){
                    diff = op1-op2;
                    tree->element = diff + '0';
                    delete Left(tree);
                    delete Right(tree);
```

```
                    tree->left = nullptr;
                    tree->right = nullptr;
                } else {
                    Left(tree)->element = '0';
                    Right(tree)->element = (op2-op1) + '0';
                }

            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	~	Error: Wrong element - ~	Получен ожидаемый вывод (проверка на недопустимые символы)
2	91-21-+-	Error: Wrong size	Получен ожидаемый вывод (лишний оператор).
3	91-21-+	Postfix output - 81+ Infix output - (8+1)	Получен ожидаемый вывод (постфиксный ввод).
4	((((((((((9-1)-(5-9))-1)-2)-1)-1)-(9*1))-(1*2))+2)-5)	Postfix output - 804--1-2-1-1-91*-12*-2+5- Infix output - ((((((((((8-(0-4))-1)-2)-1)-1)-(9*1))-(1*2))+2)-5)	Получен ожидаемый вывод (инфиксный ввод, положительный и отрицательный результат вычитания).
5	31-97--	Postfix output - 0 Infix output - 0	Получен ожидаемый вывод (рекурсивное вычитание).
6	31-94--	Postfix output - 03- Infix output - (0-3)	Получен ожидаемый вывод (рекурсивное вычитание с отрицательным результатом).