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 _____ Куршев Е.О

Преподаватель _____ Филатов Ар.Ю

Санкт-Петербург

2020

Цель работы.

Создание базовых функций для работы с иерархическим списком, а также создание рекурсивной функции, просматривает весь иерархический список на всех уровнях вложения. Работа подразумевает дальнейшее тестирование программы.

Задание

11 вариант через указатели.

С помощью построения дерева-формулы t преобразовать заданную формулу f из инфиксной формы в префиксную (перечисление узлов t в порядке КЛП) и в постфиксную (перечисление в порядке ЛПК). Если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t.

формула::= терминал | формула знак формула

знак::= + | - | *

терминал::= 0 | 1 | ... | 9 | a | b | ... | z

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 >= 0 попарно не пересекающихся множествах T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm 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. Уровень узла определяется рекурсивно следующим образом:

1) корень имеет уровень 1;

2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня.

Говорят, что каждый корень является отцом корней своих поддеревьев и что последние являются сыновьями своего отца и братьями между собой.

Говорят также, что узел n – предок узла m (а узел m – потомок узла n), если n – либо отец m, либо отец некоторого предка m.

Если в определении дерева существен порядок перечисления поддеревьев T1, T2, ..., Tm , то дерево называют упорядоченным и говорят о «первом» (T1), «втором» (T2) и т. д. поддеревьях данного корня.

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт б в определении дерева можно было бы сформулировать так: узлы дерева, за исключением корня, образуют лес.

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

В ходе выполнения работы были реализованы следующие функции и структуры:

1. class BinTree// класс, описывающий элемент дерева;
2. BinTree* Get_left()// получаем указатель на левое поддерево
3. BinTree* Get_right()// получаем указатель на правое поддерево
4. void Create_right()// создаём правое поддерево
5. void Create_left()// создаём левое поддерево
6. char Get_value() const// получаем значение элемента
7. void Set_value(char val)// меняем значение элемента
8. void add(char data, BinTree *p)// добавляем элемент
9. void read_tree(string& line, int start, int end, BinTree* tree)// чтение дерева

10. void postfix(BinTree *p, stack<char> &st)//формируем постфиксную запись
11. void prefix(BinTree *p, stack<char> &st)// формируем префиксную запись
12. int res(BinTree* tree, bool& err)// вычисляем значение выражения

Выводы.

Был реализован класс для работы с бинарным деревом. Так же были реализованы функции для работы с бинарным деревом. Было проведено тестирование программы. В реализации некорректными считаются те данные, которые не содержат скобочного представления дерева. Все подобные строки не будут считываться и программа перейдет к следующей строке файла.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <stack>

using namespace std;

class BinTree{
private:
    char value;
    BinTree* left;
    BinTree* right;

public:
    BinTree(){
        left = nullptr;
        right = nullptr;
    }

    BinTree* Get_left(){
        return this->left;
    }

    BinTree* Get_right(){
        return this->right;
    }

    void Create_right(){
        this->right = new BinTree;
    }
```

```
        void Create_left(){
            this->left = new BinTree;
        }

        char Get_value() const{
            return this->value;
        }

        void Set_value(char val){
            this->value = val;
        }

        void add(char data, BinTree *p){
            p->Set_value(data);
            p->Create_left();
            p->Create_right();
        }
    };

    void read_tree(string& line, int start, int end, BinTree*
tree){
        int def = 0;
        if (start == end)
            tree->add(line[start], tree);
        else{
            for (int i = start; i <= end; i++){
                if (line[i] == '(')
                    def++;
                if (line[i] == ')')
                    def--;
                if (def == 1 && (line[i] == '+' || line[i] == '-'
|| line[i] == '*')){
                    tree->add(line[i], tree);
                    read_tree(line, start+1, i-1, tree-
>Get_right());
```

```
                    read_tree(line, i+1, end-1, tree->Get_left());
                }
            }
        }
    }

    void postfix(BinTree *p, stack<char> &st){
        if (p->Get_left() || p->Get_right()){
            st.push(p->Get_value());
            postfix(p->Get_left(), st);
            postfix(p->Get_right(), st);
        }
    }

    void prefix(BinTree *p, stack<char> &st){
        if (p->Get_left() || p->Get_right()){
            prefix(p->Get_left(), st);
            prefix(p->Get_right(), st);
            st.push(p->Get_value());
        }
    }

    int res(BinTree* tree, bool& err){
        if (tree->Get_value() == '+')
            return res(tree->Get_left(), err) + res(tree-
>Get_right(), err);
        if (tree->Get_value() == '-')
            return res(tree->Get_right(), err) - res(tree-
>Get_left(), err);
        if (tree->Get_value() == '*')
            return res(tree->Get_left(), err) * res(tree-
>Get_right(), err);
        if (isdigit(tree->Get_value()))
            return tree->Get_value() - '0';
        else{
```

```
            err = true;
            return 0;
        }
    }

    int main(){
        string path;
        cout << "Enter the file name:\n";
        getline(cin,path);
        ifstream input(path);
        if(!input)
            cout << "Wrong file name!\n";
        else{
            cout << '\n';
            string line;
            while(getline(input, line)){
                stack <char> s1, s2;
                auto *tree = new BinTree;
                read_tree(line, 0, line.length() - 1, tree);
                cout << "input: " << line << '\n';
                cout << "prefix: ";
                prefix(tree, s1);
                while(!s1.empty()){
                    cout << s1.top();
                    s1.pop();
                }
                cout << '\n';
                cout << "postfix: ";
                postfix(tree, s2);
                while(!s2.empty()){
                    cout << s2.top();
                    s2.pop();
                }
                cout << '\n';
                bool error = false;
```

```
            int r = res(tree, error);
            if (!error)
                cout << "res = " << r;
            cout << "\n\n";
        }
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Входной файл: test.txt

```
((1+2)*(3+4))
(a*(b*(c*(d+e))))
(1+a*(2+d*(f-c)))
```

Результаты работы программы:

```
input: ((1+2)*(3+4))
prefix: *+12+34
postfix: 12+34+*
res = 21

input: (a*(b*(c*(d+e))))
prefix: *a*b*c+de
postfix: abcde+***


input: (1+a*(2+d*(f-c)))
prefix: **-fc
postfix: fc-**
```