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 _____ Микулик Д.П.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Создание базового функционала для работы с бинарным деревом, а ткаже написания функции-преобразования для бинарного дерева-формулы.

Задание.

Вариант 14 (ссылочная реализация)

Преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам $(f_1*(f_2+f_3))$, $((f_1+f_2)*f_3)$ на поддеревья, соответствующие формулам $((f_1*f_2)+(f_1*f_3))$, $((f_1*f_3)+(f_2*f_3))$.

Основные теоретические положения.

Дадим формальное определение дерева.

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 $T_1, T_2, \ldots, T_m$, каждое из которых, в свою очередь, является деревом. Деревья $T_1, T_2, \ldots, T_m$ называются поддеревьями данного дерева. При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Выполнение работы.

Для представления бинарного дерева (БД) в памяти были реализованы следующие шаблонные классы:

BinTree – базовый класс, который представляет собой бинарное дерево. Содержит указатель на класс Node, а также все основные методы для работы с бинарным деревом. Данный класс является шаблонным классом.

Node – базовый класс узла, который содержит в себе поля: U value (информация типа U, содержащаяся в узле), Node* left (указатель на левое поддерево), Node* right (указатель на правое поддерево), а также метод SetVal(Node<U>** node, U value) – позволяет установить значение в текущий узел, если узла не существует, то создаёт его.

Для работы с деревом для класса BinTree были реализованы следующие методы:

void readTree() – считывает строку-формулу, преобразует ее в дерево.
void read() – вызывает метод readTree().
bool isFormula() – true, если поддерево имеет вид искомой формулы.
void change() – преобразует поддерево-формулу к требуемому виду.
void InOrderTraversal() – проходит дерево в порядке ЛКП, проверяет на наличие поддеревьев-формул искомого вида, преобразовывает их.
void traverse() – вызывает функцию обхода для преобразования.
void PrintInOrderTraversal() – проходит дерево в порядке ЛКП, печатает содержимое.
void print() – вызывает функцию обхода для печати.
void CleanInOrder() – обходит дерево в порядке ЛПК, очищает память, выделенную под дерево.
void clean() – вызывает метод обхода для очистки.

Исходный код программы представлен в приложении А. Результаты тестирования включены в приложение Б

Выводы.

Был реализован набор классов, позволяющий работать с бинарным деревом, который обеспечивает базовый функционал для работы: чтение, вывод, обработка в соответствии с заданием бинарного дерева-формулы. Также было проведено тестирование программы. Следует отметить, в данном случае некорректными считаются те данные, которые не содержать искомых поддеревьев-формул.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <stack>
#include <string>
#include <ctype.h>
#include <fstream>

using namespace std;

template<typename T>
class BinTree{
protected:
    template<typename U>
    class Node{
    public:
        U value;
        Node<U>* left = nullptr;
        Node<U>* right = nullptr;

        static void setVal(Node<U>** node, U value){
            if(*node){
                (*node)->value = value;
            }
            else {
                *node = new Node<U>;
                (*node)->value = value;
            }
        }
    };

    Node<T>* data = nullptr;

public:
    BinTree(){}

      friend ostream& operator<<(ostream& out, const BinTree<T>&
bt){

        stack<BinTree::Node<T>*> st;
        if(bt.data){
            st.push(bt.data);
        }

        while(!st.empty()){
            auto node = st.top();
            st.pop();
            out << '{' << node->value << '}';
            if(node->right)
                st.push(node->right);
            if(node->left)
                st.push(node->left);
```

```
            }
            return out;

        }


         void readTree(string& line, int& cur, int& len, Node<T>**
node){
            if(cur < len){
                Node<T>::setVal(node, 0);
                if(isalnum(line[cur])){
                    Node<T>::setVal(node, line[cur]);
                    //cout << line[cur];
                    cur++;

                } else if (line[cur] == '(') {
                    //cout << line[cur];
                    cur++;
                    readTree(line, cur, len, &((*node)->left));

                    //cout << line[cur];
                    Node<T>::setVal(node, line[cur]);
                    cur++;
                    readTree(line, cur, len, &((*node)->right));

                    //cout <<line[cur];
                    cur++;
                }

            } else {
                return;
            }
        }

        void read(string& line, int& cur, int& len){
            readTree(line, cur, len, &data);
        }

        bool isFormula(Node<T>** node){
            Node<T>* root = *node;
                  if (root->left->value == '+' && root->right &&
isalnum(root->right->value) ){
                Node<T>* lElem = root->left;
                if(lElem->left && lElem->right){
                    return true;
                }
                else{
                    return false;
                }
            }
                 else if(root->left && root->right->value == '+' &&
isalnum(root->left->value)){
                Node<T>* rElem = root->right;
                if(rElem->left && rElem->right){
                    return true;
```

```
                }
                else{
                    return false;
                }

            }
            else{
                return false;
            }
        }

        void change(Node<T>** root){
            Node<T>::setVal(root, '+');
            if((*root)->left->value == '+'){
                   Node<T>::setVal(&((*root)->right->right), (*root)-
>right->value);
                Node<T>::setVal(&((*root)->left), '*');
                    Node<T>::setVal(&((*root)->right->left), (*root)-
>left->right->value);
                Node<T>::setVal(&((*root)->right), '*');
                    Node<T>::setVal(&((*root)->left->right), (*root)-
>right->right->value);
            }
            else{
                     Node<T>::setVal(&((*root)->left->left), (*root)-
>left->value);
                Node<T>::setVal(&((*root)->left), '*');
                    Node<T>::setVal(&((*root)->left->right), (*root)-
>right->left->value);
                    Node<T>::setVal(&((*root)->right->left), (*root)-
>left->left->value);
                Node<T>::setVal(&((*root)->right), '*');

            }

        }

        void InOrderTraversal(Node<T>** node){
            if (*node){
                InOrderTraversal(&((*node)->left));
                            if(((*node)->value  ==  '*')  &&  (this-
>isFormula(node))){
                    change(node);
                }
                InOrderTraversal(&((*node)->right));
            }
            else{
                return;
            }
        }

        void traverse(){
            InOrderTraversal(&(this->data));
        }

        void PrintInOrderTraversal(Node<T>* node, int k){
```

```
            if (node){
                if (k == 1){
                    cout << "(";
                }
                PrintInOrderTraversal(node->left, 1);

                cout << node->value ;

                PrintInOrderTraversal(node->right, 2);
                if (k == 2){
                    cout << ")";
                }
            }
            else{
                return;
            }
        }

        void print(){
            PrintInOrderTraversal(this->data, 0);
            cout << "\n";
        }

        void CleanInOrder(Node<T>* node){
            if (node){
                CleanInOrder(node->left);
                CleanInOrder(node->right);
                delete node;

            }
            else{
                return;
            }
        }

        void clean(){
            CleanInOrder(this->data);
        }
    };

    int main(){
        string file_name;
        cout << "Enter the name of an input file: " << endl;
        getline(cin, file_name);
        ifstream input(file_name);
        if (!input){
               cout << "You haven't entered correct input file." <<
endl;
        }
        else{
            cout << "Binary tree traverse:" << endl;
            string line;
            while(getline(input, line)){
                int len = line.length();
                int cur = 0;
                BinTree<char> bt1;
```

```
            bt1.read(line, cur, len);
            bt1.print();
            bt1.traverse();
            bt1.print();
            bt1.clean();

        }
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Файл со входными данными: test.txt

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	test.txt ((a*(b+(c*(d*(e+f))))) +(1*2))	(((a*b)+(a*(c*((d*e)+ (d*f)))))+(1*2))	Программа работает корректно.
2.	test.txt (a*(b+c))	((a*b)+(a*c))	Программа работает корректно.
3.	test.txt ((b+c)*a)	((b*a)+(c*a))	Программа работает корректно.
4.	test.txt ((a*(b+c))+((a+b)*c))	(((a*b)+(a*c))+((a*c)+ (b*c)))	Программа работает корректно.
5.	test.txt ((a+b)*(c+d))	((a+b)*(c+d))	Программа работает корректно.
6. (тест на некорректных данных)	test.txt ((a+b)+(c+d))	((a+b)+(c+d))	Программа работает корректно.
7. (тест на некорректном имени файла)	asdasd	You haven't entered correct input file.	Программа работает корректно.