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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 9303 _____ Низовцов Р. С.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием «бинарное дерево», изучить его особенности и реализовать программу, решающую поставленную задачу с помощью бинарного дерева.

Задание.

Вариант 16(д)

Упростить дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам (f+0), (0+f), (f-0), (f*1), (1*f), на поддеревья, соответствующие формуле f; поддеревья, соответствующие формулам (f*0), (0*f) и (f – f), – на узел с 0; поддеревья, соответствующие формулам (f_1+(0–f_2)) и ((0–f_2)+f_1), – на поддеревья, соответствующие формуле (f_1–f_2); поддеревья, соответствующие формуле (f_1–(0–f_2)), – на поддеревья, соответствующие формуле (f_1+f_2). Предусмотреть возможность упрощения в несколько этапов.

Основные теоретические положения.

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения *меньше*.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных. Однако это касается реализации, а не природы двоичного дерева поиска.

Выполнение работы.

Для выполнения программы были реализованы функции read, simplify, write.

В функции int main() запрашивается путь к файлу с данными для обработки, проверяется её корректность. Потом запускается цикл и работает, пока getLine не вернет false. После этого информирует об окончании работы, закрывает все файлы.

В функции void read(string, int, int) проводится посимвольное чтение из line до конца строки. Переменные типа int контролирует корректный проход по строке. Из них создается иерархический список.

В функции bool simplify() проводится упрощение бинарного дерева: сначала проверяются выражения со входом в выражение одной из веток, потом проверяются выражения без дополнительных вхождений. Работа продолжается, пока функция не вернет false.

В функции void write(ofstream&) проводится вывод дерева в файл, параллельно выводя результат в консоль.

Выводы.

Ознакомился с понятием «бинарное дерево», изучил его особенности и реализовал программу, решающую поставленную задачу с помощью бинарного дерева.

Была реализована программа, включающая в себя функцию simplify для обработки элементов, хранящихся в дереве. Программа выполняет чтение, запись результата в файл, а также производит проверку и обработку считанного текста.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <stack>
#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]);
                cur++;

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

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

        } else {
            return;
        }
    }

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

    template<typename U>
    void change(Node<T>** root, U value){
        Node<T>::setVal(root, value);
        delete (*root)->left;
        (*root)->left = nullptr;
        delete (*root)->right;
        (*root)->right = nullptr;
    }

    bool InOrderEzyTraversal(Node<T>** node){
        if (*node){
            bool answer = false;
            answer |= InOrderEzyTraversal(&((*node)->left));

            if(((*node)->value == '-') && ((*node)->right) && ((*node)-
>right->value == '-') && ((*node)->right->left) && ((*node)->right->left-
>value == '0') && ((*node)->right->right) && (isalnum((*node)->right-
>right->value))){
                Node<T>::setVal(node, '+');
                change(&((*node)->right), (*node)->right->right->value);
                answer = true;
            }
            else if(((*node)->value == '+') && ((*node)->right) &&
((*node)->right->value == '-') && ((*node)->right->left) && ((*node)-
>right->left->value == '0') && ((*node)->right->right) &&
(isalnum((*node)->right->right->value))){
                Node<T>::setVal(node, '-');
                change(&((*node)->right), (*node)->right->right->value);
                answer = true;
            }
```

```
            else if(((*node)->value == '-') && ((*node)->left) &&
((*node)->left->value == '-') && ((*node)->left->left) && ((*node)->left-
>left->value == '0') && ((*node)->left->right) && (isalnum((*node)->left-
>right->value))){
                Node<T>::setVal(node, '+');
                change(&((*node)->left), (*node)->left->right->value);
                answer = true;
            }
            else if(((*node)->value == '+') && ((*node)->left) &&
((*node)->left->value == '-') && ((*node)->left->left) && ((*node)->left-
>left->value == '0') && ((*node)->left->right) && (isalnum((*node)->left-
>right->value))){
                Node<T>::setVal(node, '-');
                change(&((*node)->left), (*node)->left->right->value);
                answer = true;
            }

            if(((*node)->value == '-') && ((*node)->right) &&
(isalnum((*node)->right->value)) && ((*node)->left) && (isalnum((*node)-
>left->value)) && ((*node)->left->value == (*node)->right->value)){
                change(node, '0');
                answer = true;
            }
            else if(((*node)->value == '-') && ((*node)->right) &&
(isalnum((*node)->right->value)) && ((*node)->left) && (isalnum((*node)-
>left->value)) && ((*node)->right->value == '0')){
                change(node, (*node)->left->value);
                answer = true;
            }
            else if(((*node)->value == '+') && ((*node)->right) &&
((*node)->right->value == '0') && ((*node)->left) && (isalnum((*node)-
>left->value))){
                change(node, (*node)->left->value);
                answer = true;
            }
            else if(((*node)->value == '+') && ((*node)->left) &&
((*node)->left->value == '0') && ((*node)->right) && (isalnum((*node)-
>right->value))){
                change(node, (*node)->right->value);
                answer = true;
            }
            else if(((*node)->value == '*') && ((*node)->right) &&
((*node)->right->value == '1') && ((*node)->left) && (isalnum((*node)-
>left->value))){
                change(node, (*node)->left->value);
                answer = true;
            }
            else if(((*node)->value == '*') && ((*node)->left) &&
((*node)->left->value == '1') && ((*node)->right) && (isalnum((*node)-
>right->value))){
                change(node, (*node)->right->value);
                answer = true;
            }
            else if(((*node)->value == '*') && ((*node)->right) &&
(isalnum((*node)->right->value)) && ((*node)->left) && (isalnum((*node)-
>left->value)) && ((*node)->left->value == '0' || (*node)->right->value
== '0')){
                change(node, '0');
```

```
            answer = true;
        }

        answer |= InOrderEzyTraversal(&((*node)->right));
        return answer;
    }
    else{
        return false;
    }
}

bool ezyTraverse(){
    return InOrderEzyTraversal(&(this->data));
}

void PrintInOrderTraversal(Node<T>* node, int k, ofstream &outFile){
    if (node){
        if (k == 1){
            cout << "(";
            outFile << "(";
        }
        PrintInOrderTraversal(node->left, 1, outFile);

        cout << node->value ;
        outFile << node->value;

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

void print(ofstream &outFile){
    PrintInOrderTraversal(this->data, 0, outFile);
    cout << endl;
    outFile << endl;
}

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;
    cin >> file_name;
    ifstream inFile(file_name);
    if (!inFile.is_open()){
        cout << "Permission denied or wrong path";
        return 0;
    }
    cout << "Result was saved in result.txt" << endl;
    ofstream outFile("/home/rostislav/result.txt");
    string line;
    while(getline(inFile, line)){
        int len = line.length();
        int cur = 0;
        BinTree<char> bt1;
        bt1.read(line, cur, len);
        cout << line << " -> ";
        while(bt1.ezyTraverse());
        bt1.print(outFile);
        bt1.clean();
        }
    cout << "End of work" << endl;
    inFile.close();
    outFile.close();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 — Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарий
1.			Программа работает корректно
2.	(f-0) ((f-0)*(1-0))	f f	Программа работает корректно
3.	(((f-0)-0)*(1-(f-f)))	f	Программа работает корректно
4.	(1-(0-f)) ((0-d)-(0-0))	(1+f) d	Программа работает корректно