**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. 9382 _____ Русинов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

**Цель работы.**

Изучить нелинейную структуру — дерево, способы ее реализации и рекурсивной обработки. Получить навыки решения задач по обработке деревьев, как с использованием рекурсивных функций, так и с использованием функций не имеющих рекурсивной природы.

**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.

*Лист (концевой узел)* — узел  множество поддеревьев которого пусто.

*Упорядоченное дерево* — дерево в котором важен порядок перечисления его поддеревьев.

*Лес* – множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев.

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

**Задание.**

5д. Заданы два бинарных дерева b1 и b2 типа BT с произвольным типом элементов. Проверить:

- подобны ли они (два бинарных дерева подобны, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);
- равны ли они (два бинарных дерева равны, если они подобны и их соответствующие элементы равны);
- зеркально подобны ли они (два бинарных дерева зеркально подобны, если они оба пусты либо они оба непусты и для каждого из них левое поддерево одного подобно правому поддереву другого);
- симметричны ли они (два бинарных дерева симметричны, если они зеркально подобны и их соответствующие элементы равны).

**Описание структуры данных для реализации бинарного дерева.**

Для реализации бинарного дерева через динамическую память был создан класс Node. В нем определено 3 поля: *left* — для хранения указателя на левое поддерево, либо нулевого указателя в случае отсутствия этого поддерева; *right* — для хранения указателя на правое поддерево, либо нулевого указателя в случае его отсутствия; *value* — для хранения значения корня бинарного дерева.
Также определено 3 метода для работы с этим классом: addLeftNode - для установки левого поддерева, addRightNode – для установки правого поддерева. Также у него указаны friend NodeOperations<type> и friend printTree(Node<type>* tree, int level) – класс для проведения операций над деревьями и функция печати дерева для предоставления приватных полей и методов класса Node.

**Описание алгоритма.**

Для хранения бинарного дерева создается класс Node рекурсивной природы с использованием динамической памяти.

Дерево заполняется с помощью рекурсивной функции TreeCreator::createTree. В классе TreeCreator реализованы дополнительные статические методы для создания дерева из строки:

- Bool isAlphabetSymbol(char symbol) – проверяет, является ли символ алфавитным нижнего регистра.
- Std::string indexFormatGenerator(const int* index) – генерирует строку для форматного вывода.
- Node<char>** createTreesFromFile() – создает массив из двух деревьев из файла file.txt

Сам метод createTree работает следующим образом:

1. На вход дается строка с записью дерева в скобочном формате
2. Если встречен символ ‘(’, то найдено дерево, дальше нужно найти корень, левое поддерево и правое поддерево, затем терминальный символ ‘)’. Корень является алфавитным символом, а левое и правое поддерево может начинаться с символа ‘(’, то есть может принимать формат дерева. Так же, если нужно обозначить пустую ветку, то можно воспользоваться символом ‘/’.
3. Данная функция проходит по строке и рекурсивно формирует дерево. Если внутри дерева встречен ‘(’, то делается вызов createTree, который возвращает Node*.

Удаляется дерево с помощью рекурсивной деструктора. Выполняются команды delete left, delete right.

Все элементы бинарного дерева печатаются по уровням с помощью функции printTree(Node<char>* tree, int level). В нее передается дерево в котором элементы записаны в порядке, в котором их нужно выводить, уровни отделяются с помощью значения “ ” между ними. Сначала рекурсивно печатаются левые поддеревья, затем печатаются значения, а после рекурсивно печатаются правые поддеревья.

Для выполнения задания реализован класс NodeOperations и класс Exercise. NodeOperations – выполняет операции над двумя деревьями, он содержит методы для проверки подобности, равности, зеркальной подобности, симметричности заданных деревьев. Класс Exercise – класс для выполнения всего задания. Сначала пользователь считывает с помощью команды TreeCreator::createTreesFromFile деревья из файла, затем создает класс Exercise, передавая в аргументы массив из двух деревьев и вызывает метод executeTask(). В консоли печатается результат выполнения задания над двумя деревьями.

**Описание функций и классов.**

*template<typename type>*
*class Node;*

— класс для дерева. Конструктор принимает type value – значение корня.

- Метод addLeftNode – принимает Node* node и устанавливает в качестве левой ветки заданный Node.
- Метод addRightNode – принимает Node* node и устанавливает в качестве правой ветки заданный Node.

template<typename type>
class NodeOperations;

— класс для проведения операций над двумя деревьями. Конструктор принимает два дерева Node<type>* firstNode, secondNode.

- Метод bool areSimilar() – проверяет два дерева на подобность. Если два дерева пусты, то они подобны, если одно из них пустое, то не подобны, так мы рекурсивно проверяем areSimilar() для левых корней деревьев и правых корней и получаем результат.
- Метод bool areEqual() – принцип схож с areSimilar(), только еще проверка значений узлов.

- Метод areMirroredSimilar() – принцип схож с areSimilar(), только теперь проверяются на подобность firstNode->left и secondNode->right.
- Метод areMirroredEqual() – принцип схож с areMirroredSimilar, только теперь проверяются значения узлов на равность.

Class TreeCreator;
— класс для создания деревьев из строки скобочного формата. Также предоставляет метод для задания – считывание двух деревьев из файла.

- Static bool isAlphabetSymbol(char symbol) – проверяет, является ли символ алфавитным нижнего регистра
- Static std::string indexFormatGenerator(const int* index) – метод для генерации форматной строки.
- Static Node<char>* createTree(const std::string& string, int* index = nullptr) – метод для генерации структуры дерева из строки скобочного формата.
- Static Node<char>** createTreesFromFile() – метод для генерации двух деревьев для задания из файла file.txt. Возвращает массив из двух деревьев.

Void PrintTree(Node<char>* tree, int level);
— функция для печати дерева. Печать в формате ЛКП.

Class Exercise;
— класс для выполнения лабораторного задания. Конструктор принимает на вход массив из двух деревьев, полученный с помощью класса TreeCreator метода createTreesFromFile. У класса есть метод executeTask() – выполняет лабораторное задание.

**Тестирование.**

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

| No | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1 | (aa(bd/))<br>(aa(dba)) | ------Считывание деревьев из файла------<br>---Считывание первого дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - a<br>Завершено создание левой ветви | Дерево, для которого не выполняется ни одно свойство из лабораторной работы |

| | | | |
|---|---|---|---|
| | | Начато создание правой ветви<br>[3] Встречено дерево<br>Начато создание левой ветви<br>[5] Встречен символ - d<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[6] Встречен символ - /<br>Завершено создание левой ветви<br>[7] Найден терминальный символ ')'<br>Завершено создание левой ветви<br>[8] Найден терминальный символ ')'<br>---Считывание второго дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - a<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречено дерево<br>Начато создание левой ветви<br>[5] Встречен символ - b<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[6] Встречен символ - a<br>Завершено создание левой ветви<br>[7] Найден терминальный символ ')'<br>Завершено создание левой ветви<br>[8] Найден терминальный символ ')'<br>---Вывод первого дерева---<br>  a<br>a<br>  d<br>  b<br>---Конец вывода первого дерева---<br>---Вывод второго дерева---<br>  a<br>a<br>  b<br>  d | |

| | | | |
|---|---|---|---|
| | | a<br>---Конец вывода второго дерева---<br>---Выполнение задания---<br>Заданные деревья не подобны<br>Заданные деревья не равны<br>Заданные деревья не подобны зеркально<br>Заданные деревья несимметричны<br>---Конец--- | |
| 2 | (aa(bd/))<br>(aa(dba) | ------Считывание деревьев из файла------<br>---Считывание первого дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - a<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречено дерево<br>Начато создание левой ветви<br>[5] Встречен символ - d<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[6] Встречен символ - /<br>Завершено создание левой ветви<br>[7] Найден терминальный символ ')'<br>Завершено создание левой ветви<br>[8] Найден терминальный символ ')'<br>---Считывание второго дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - a<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречено дерево<br>Начато создание левой ветви<br>[5] Встречен символ - b<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[6] Встречен символ - a | Считывание некорректных данных |

| | | | |
|---|---|---|---|
| | | Завершено создание левой ветви<br>[7] Найден терминальный символ ')'<br>Завершено создание левой ветви<br>[8] Не найден терминальный символ ')'<br>libc++abi.dylib: terminating with uncaught exception of type std::invalid_argument: [8] Did not find the terminal symbol ')' | |
| 3 | (<br>( | libc++abi.dylib: terminating with uncaught exception of type std::invalid_argument: [1] Found root -<br>------Считывание деревьев из файла------<br>---Считывание первого дерева---<br>[0] Встречено дерево<br>[1] Встречен корень – | Считывание некорректных данных |
| 4. | (a/b)<br>(ab/) | ------Считывание деревьев из файла------<br>---Считывание первого дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - /<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречен символ - b<br>Завершено создание левой ветви<br>[4] Найден терминальный символ ')'<br>---Считывание второго дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - b<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречен символ - / | Симметричное и зеркально подобное дерево |

| | | | |
|---|---|---|---|
| | | Завершено создание левой ветви<br>[4] Найден терминальный символ ')'<br>---Вывод первого дерева---<br>a<br>  b<br>---Конец вывода первого дерева---<br>---Вывод второго дерева---<br>  b<br>a<br>---Конец вывода второго дерева---<br>---Выполнение задания---<br>Заданные деревья не подобны<br>Заданные деревья не равны<br>Заданные деревья зеркально подобны<br>Заданные деревья симметричны<br>---Конец--- | |
| 5. | (a//)<br>(a//) | ------Считывание деревьев из файла------<br>---Считывание первого дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - /<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречен символ - /<br>Завершено создание левой ветви<br>[4] Найден терминальный символ ')'<br>---Считывание второго дерева---<br>[0] Встречено дерево<br>Начато создание левой ветви<br>[2] Встречен символ - /<br>Завершено создание левой ветви<br>Начато создание правой ветви<br>[3] Встречен символ - /<br>Завершено создание левой ветви | Дерево, для которого выполняются все свойства лабораторной работы |

| | | | |
|---|---|---|---|
| | | [4] Найден терминальный символ ')'<br>---Вывод первого дерева---<br>a<br>---Конец вывода первого дерева---<br>---Вывод второго дерева---<br>a<br>---Конец вывода второго дерева---<br>---Выполнение задания---<br>Заданные деревья подобны<br>Заданные деревья равны<br>Заданные деревья зеркально подобны<br>Заданные деревья симметричны<br>---Конец--- | |

**Выводы.**

Были изучены и опробованы различные методы работы с бинарными деревьями на языке C++. Была создана программа для реализации и работы с бинарными деревьями, использующая, как рекурсивные функции, так и функции не рекурсивной природы.

# ПРИЛОЖЕНИЕ С КОДОМ

## main.cpp :

```
#include <iostream>
#include <string>
#include "fstream"

template<typename type>
class Node;

template<typename type>
class NodeOperations;


template<typename type>
class Node {
    friend NodeOperations<type>;
    friend void printTree(Node<type>* tree, int level);
    type value;
    Node* left = nullptr;
    Node* right = nullptr;

public:
    explicit Node(char value = 0) : value(value) {}
    ~Node() {
        delete left;
        delete right;
    }
    void addLeftNode(Node* node) {
        delete left;
        left = node;
    }
    void addRightNode(Node* node) {
        delete right;
        right = node;
    }
};


template<typename type>
class NodeOperations {
    Node<type>* firstNode = nullptr;
    Node<type>* secondNode = nullptr;

public:
    NodeOperations(Node<type>* firstNode, Node<type>* secondNode)
            : firstNode(firstNode), secondNode(secondNode) {}

    bool areSimilar(){ // Метод проверки подобности
        if (firstNode == nullptr && secondNode == nullptr) return true;
        if (firstNode == nullptr || secondNode == nullptr) return false;
        return NodeOperations(firstNode->left, secondNode->left).areSimi-
lar()
               && NodeOperations(firstNode->right, secondNode-
>right).areSimilar();
```

```
    }

    bool areEqual(){ // Метод проверки равности
        if (firstNode == nullptr && secondNode == nullptr) return true;
        if (firstNode == nullptr || secondNode == nullptr) return false;
        if (firstNode->value != secondNode->value) return false;
        return NodeOperations(firstNode->left, secondNode-
>left).areEqual()
               && NodeOperations(firstNode->right, secondNode-
>right).areEqual();
    }

    bool areMirroredSimilar(){ // Метод проверки зеркальной подобности
        if (firstNode == nullptr && secondNode == nullptr) return true;
        if (firstNode == nullptr || secondNode == nullptr) return false;
        return NodeOperations(firstNode->left, secondNode->right).areMir-
roredSimilar()
               && NodeOperations(firstNode->right, secondNode->left).are-
MirroredSimilar();
    }

    bool areMirroredEqual(){ // Метод проверки симметричности
        if (firstNode == nullptr && secondNode == nullptr) return true;
        if (firstNode == nullptr || secondNode == nullptr) return false;
        if (firstNode->value != secondNode->value) return false;
        return NodeOperations(firstNode->left, secondNode->right).areMir-
roredEqual()
               && NodeOperations(firstNode->right, secondNode->left).are-
MirroredEqual();
    }
};


class TreeCreator { // Класс для создания деревьев
    static bool isAlphabetSymbol(char symbol){ // Метод для проверки, яв-
ляется ли символ алфавитным нижнего регистра
        if (97 <= symbol && symbol <= 122) return true;
        return false;
    }

    static std::string indexFormatGenerator(const int* index) { // Метод
для генерации форматного вывода
        return std::string("[") + std::to_string(*index) + std::string("]
");
    }

    static Node<char>* createTree(const std::string& string, int* index =
nullptr) { // Рекурсивный метод создания дерева
        if (!index) { // Начальный вызов метода
            int startIndex = 0;
            index = &startIndex;
        }

        if (isAlphabetSymbol(string[*index]) || string[*index] == '/') {
// Если символ алфавитный, создаем узел без ветвей
            std::cout << indexFormatGenerator(index) << "Встречен символ
- " << string[*index] << std::endl;
```

```
            if (isAlphabetSymbol(string[*index])) return new
Node<char>(string[*index]);
            // если символ /, значит возвращаем нулевой указатель
            return nullptr;
        }

        else if (string[*index] == '(') { // встречено дерево
            std::cout << indexFormatGenerator(index) << "Встречено де-
рево" << std::endl;
            *index += 1;
            if (!isAlphabetSymbol(string[*index])) { // проверяем корень
                std::cout << indexFormatGenerator(index) << "Встречен ко-
рень - " << string[*index] << ", данный символ некорректен!" <<
std::endl;
                throw std::invalid_argument(indexFormatGenerator(index) +
"Found root - " + string[*index] + ", this symbol is incorrect!");
            }

            auto* node = new Node<char>(string[*index]); // создание
корня
            *index += 1;

            std::cout << "Начато создание левой ветви" << std::endl;
            Node<char>* leftBranch = createTree(string, index); // созда-
ние левой ветви
            node->addLeftNode(leftBranch);
            *index += 1;
            std::cout << "Завершено создание левой ветви" << std::endl;

            std::cout << "Начато создание правой ветви" << std::endl;
            Node<char>* rightBranch = createTree(string, index); // со-
здание правой ветви
            node->addRightNode(rightBranch);
            *index += 1;
            std::cout << "Завершено создание левой ветви" << std::endl;

            if (string[*index] != ')') { // проверка терминального сим-
вола
                std::cout << indexFormatGenerator(index) << "Не найден
терминальный символ ')'" << std::endl;
                throw std::invalid_argument(indexFormatGenerator(index) +
"Did not find the terminal symbol ')'");
            }

            std::cout << indexFormatGenerator(index) << "Найден терми-
нальный символ ')'" << std::endl;

            return node;

        } else {
            std::cout << indexFormatGenerator(index) << "Встречен некор-
ректный символ!" << std::endl;
            throw std::invalid_argument(indexFormatGenerator(index) +
std::string("Incorrect symbol") + string[*index]);
        }
    }
public:
```

```
    static Node<char>** createTreesFromFile() { // создание деревьев из
файла
        std::cout << "------Считывание деревьев из файла------" <<
std::endl;
        auto** treeArray = new Node<char>*[2];
        std::fstream treesFile("file.txt");
        if (!treesFile.is_open()) { // проверим, получилось ли открыть
файл
            std::cout << "Не удалось открыть файл file.txt" << std::endl;
            throw std::invalid_argument("Can not to open file.txt");
        }

        std::string firstTree;
        std::string secondTree;
        std::getline(treesFile, firstTree, '\n');
        std::getline(treesFile, secondTree, '\n');
        treesFile.close();
        std::cout << "---Считывание первого дерева---" << std::endl;
        treeArray[0] = createTree(firstTree); // создаем первое дерево
        std::cout << "---Считывание второго дерева---" << std::endl;
        treeArray[1] = createTree(secondTree); // создаем второе дерево
        return treeArray; // возвращаем массив из двух деревьев
    }
};


// Функция печати дерева
void printTree(Node<char>* tree, int level)
{
    if(tree)
    {
        printTree(tree->left, level + 1);
        for (int i = 0; i < level; ++i) std::cout << "   ";
        std::cout << tree->value << std::endl;
        printTree(tree->right, level + 1);
    }
}


class Exercise { // класс задания
    Node<char>** treeArray = nullptr;
public:
    explicit Exercise(Node<char>** treeArray) : treeArray(treeArray) {}
    ~Exercise(){
        delete treeArray[0];
        delete treeArray[1];
        delete [] treeArray;
    }
    void executeTask() { // метод для выполнения задания
        std::cout << "---Вывод первого дерева---" << std::endl;
        printTree(treeArray[0], 0);
        std::cout << "---Конец вывода первого дерева---" << std::endl;

        std::cout << "---Вывод второго дерева---" << std::endl;
        printTree(treeArray[1], 0);
        std::cout << "---Конец вывода второго дерева---" << std::endl;
```

```
        std::cout << "---Выполнение задания---" << std::endl;
        NodeOperations<char> operations(treeArray[0], treeArray[1]);
        if (operations.areSimilar()) std::cout << "Заданные деревья по-
добны" << std::endl;
        else std::cout << "Заданные деревья не подобны" << std::endl;

        if (operations.areEqual()) std::cout << "Заданные деревья равны"
<< std::endl;
        else std::cout << "Заданные деревья не равны" << std::endl;

        if (operations.areMirroredSimilar()) std::cout << "Заданные дере-
вья зеркально подобны" << std::endl;
        else std::cout << "Заданные деревья не подобны зеркально" <<
std::endl;

        if (operations.areMirroredEqual()) std::cout << "Заданные деревья
симметричны" << std::endl;
        else std::cout << "Заданные деревья несимметричны" << std::endl;
        std::cout << "---Конец---" << std::endl;
    }
};


int main() {
    Exercise(TreeCreator::createTreesFromFile()).executeTask();
    return 0;
}
```