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.

Считываются две строки из файла, в этих двух строках должны находиться два дерева в скобочном формате. Пример: (a//)

- a – корень
- / - левое поддерево
- / - правое поддерево

Считывание происходит по принципу КЛП. Реализация считывания выполнена рекурсивно. Если встречен терминальный символ ‘(’, тогда проверяется следующий символ, он будет являться корнем. Далее считываются левое и правое поддеревья соответственно. Каждый раз, когда встречается терминальный символ ‘(’, вызывается функция, которая возвращает поддерево. Если на предполагаемом месте поддерева встречается алфавитный символ, то возвращается поддерево, где корень – символ, а его поддеревья пустые. После проверки поддеревьев, проверяется наличие терминального символа ‘)’.

Проверка подобности двух деревьев выполняется следующим образом:

1) Если первое дерево пустое, и второе дерево пустое, то деревья подобны. Вернуть истину.
2) Если какое-то из деревьев пустое, а другое непустое, то деревья не подобны. Вернуть ложь.
3) Если алгоритм дошел до этого пункта, то нужно проверить на подобность левые поддеревья на подобность, правые поддеревья на подобность, и вернуть логическое И между этими двумя результатами.

Проверка равности двух деревьев выполняется следующим образом:
Алгоритм схож в первом и втором пунктах проверки подобности двух деревьев, единственное отличие лишь в третьем пункте. Там необходимо проверить также на равность корни деревьев. Если корни не равны, то деревья не равны друг другу.

Проверка зеркальной подобности двух деревьев выполняется следующим образом:
Алгоритм схож в первом и втором пунктах проверки подобности двух деревьев, единственное отличие лишь в третьем пункте. Там проверяется на зеркальную подобность левое поддерево и правое поддерево деревьев, и правое поддерево с левым поддеревом деревьев. Применяется логическое И между двумя результатами, и возвращается полученное значение.

Проверка симметричности двух деревьев выполняется следующим образом:
Алгоритм схож в первом и втором пунктах проверки зеркальной подобности двух деревьев, единственное отличие лишь в третьем пункте. Необходимо проверить также значения корней. Если они не равны, то деревья не симметричны.

Все приведенные алгоритмы выполняются рекурсивно.

Описание функций и классов.

template<typename type>
class Node; — класс для дерева.

- explicit Node(char value = 0) - конструктор
- void addLeftNode(Node* node) – устанавливает левое поддерево
- void addRightNode(Node* node) – устанавливает правое поддерево

template<typename type>
class NodeOperations; — класс для проведения операций над двумя деревьями.

- NodeOperations(Node<type>* firstNode, Node<type>* secondNode) – конструктор
- bool areSimilar() – метод, проверяющий два дерева на подобность
- bool areEqual() – метод, проверяющий два дерева на равность

- bool areMirroredSimilar() – метод, проверяющий два дерева на зеркальную подобность
- bool areMirroredEqual() – метод, проверяющий два дерева на симметричность.

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 символ, является ли он алфавитным нижнего регистра. Принимает символ.
- static std::string indexFormatGenerator(const int* index) – возвращает строку с индексом для форматного вывода. Принимает текущий индекс строки, над которым проводится проверка.
- static Node<char>* createTree(const std::string&, int* index = nullptr) – генерирует дерево из заданной в скобочном формате строки. Принимает ссылку на константную строку (в ней записано дерево в скобочном формате), также принимает текущий индекс (для рекурсии).
- static Node<char>** createTreesFromFile() – считывает из файла две строки, которые являются деревьями, записанные в скобочном формате. Генерирует экземпляры классов дерева из них и возвращает массив из двух элементов.
- static Node<char>** createTreesFromConsole() – считывает две строки из консоли, которые являются деревьями, записанные в скобочном формате. Генерирует экземпляры классов дерева из них и возвращает массив из двух элементов.
- static Node<char>** createTrees(const std::string& first, const std::string& second) – метод для генерации деревьев из двух строк. Создан, чтобы избежать дублирования кода для методов createTreesFromFile и createTreesFromConsole. Принимает на вход две строки, из которых генерирует экземпляры деревьев, и возвращает массив из двух экземпляров.

Void PrintTree(Node<char>* tree, int level); — функция для печати дерева. Печать в формате ЛКП. Принимает дерево, которое необходимо распечатать, а также уровень глубины, на котором сейчас находится функция.

Class Exercise;

— класс для выполнения лабораторного задания. Конструктор принимает на вход массив из двух деревьев, полученный с помощью класса TreeCreator метода createTrees. 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/)) (aa(dba))	------Считывание деревьев из файла------ ---Считывание первого дерева--- [0] Встречено дерево Начато создание левой ветви [2] Встречен символ - a Завершено создание левой ветви Начато создание правой ветви [3] Встречено дерево Начато создание левой ветви [5] Встречен символ - d Завершено создание левой ветви Начато создание правой ветви [6] Встречен символ - / Завершено создание левой ветви [7] Найден терминальный символ ')' Завершено создание левой ветви [8] Найден терминальный символ ')' ---Считывание второго дерева--- [0] Встречено дерево Начато создание левой ветви [2] Встречен символ - a Завершено создание левой ветви Начато создание правой ветви [3] Встречено дерево Начато создание левой ветви [5] Встречен символ - b Завершено создание левой ветви Начато создание правой ветви [6] Встречен символ - a	Дерево, для которого не выполняется ни одно свойство из лабораторной работы

		Завершено создание левой ветви
		[7] Найден терминальный символ ')'
		Завершено создание левой ветви
		[8] Найден терминальный символ ')'
		---Вывод первого дерева---
		  a
		a
		  d
		  b
		---Конец вывода первого дерева---
		---Вывод второго дерева---
		  a
		a
		  b
		  d
		  a
		---Конец вывода второго дерева---
		---Выполнение задания---
		Заданные деревья не подобны
		Заданные деревья не равны
		Заданные деревья не подобны зеркально
		Заданные деревья несимметричны
		---Конец---	

2	(aa(bd/)) (aa(dba)	------Считывание деревьев из файла------ ---Считывание первого дерева--- [0] Встречено дерево Начато создание левой ветви [2] Встречен символ - a Завершено создание левой ветви Начато создание правой ветви [3] Встречено дерево Начато создание левой ветви [5] Встречен символ - d Завершено создание левой ветви Начато создание правой ветви [6] Встречен символ - / Завершено создание левой ветви	Считывание некорректных данных

		[7] Найден терминальный символ ')' Завершено создание левой ветви [8] Найден терминальный символ ')' ---Считывание второго дерева--- [0] Встречено дерево Начато создание левой ветви [2] Встречен символ - a Завершено создание левой ветви Начато создание правой ветви [3] Встречено дерево Начато создание левой ветви [5] Встречен символ - b Завершено создание левой ветви Начато создание правой ветви [6] Встречен символ - a Завершено создание левой ветви [7] Найден терминальный символ ')' Завершено создание левой ветви [8] Не найден терминальный символ ')' libc++abi.dylib: terminating with uncaught exception of type std::invalid_argument: [8] Did not find the terminal symbol ')'	
3	((	libc++abi.dylib: terminating with uncaught exception of type std::invalid_argument: [1] Found root - ------Считывание деревьев из файла------ ---Считывание первого дерева--- [0] Встречено дерево [1] Встречен корень –	Считывание некорректных данных
4.	(a/b) (ab/)	------Считывание деревьев из файла------ ---Считывание первого дерева--	Симметричное и зеркально подобное дерево

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

		[2] Встречен символ - / Завершено создание левой ветви Начато создание правой ветви [3] Встречен символ - / Завершено создание левой ветви [4] Найден терминальный символ ')' ---Считывание второго дерева--- [0] Встречено дерево Начато создание левой ветви [2] Встречен символ - / Завершено создание левой ветви Начато создание правой ветви [3] Встречен символ - / Завершено создание левой ветви [4] Найден терминальный символ ')' ---Вывод первого дерева--- a ---Конец вывода первого дерева--- ---Вывод второго дерева--- a ---Конец вывода второго дерева--- ---Выполнение задания--- Заданные деревья подобны Заданные деревья равны Заданные деревья зеркально подобны Заданные деревья симметричны ---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;
        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();
        return createTrees(firstTree, secondTree);
    }

    static Node<char>** createTreesFromConsole() { // создание дерева из
консоли
        std::cout << "------Считывание деревьев из консоли------" <<
std::endl;
        std::cout << "Введите первое дерево - ";
        std::string first;
        std::cin >> first;
        std::cout << "Введите второе дерево - ";
        std::string second;
        std::cin >> second;
        return createTrees(first, second);
    }

    static Node<char>** createTrees(const std::string& first, const
std::string& second) {
        auto** treeArray = new Node<char>*[2];
        std::cout << "---Считывание первого дерева---" << std::endl;
        treeArray[0] = createTree(first); // создаем первое дерево
        std::cout << "---Считывание второго дерева---" << std::endl;
        treeArray[1] = createTree(second); // 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() {
    int value = 0;
    std::cout << "0 - ввод из консоли, 1 - ввод из файла file.txt" <<
std::endl;
    std::cout << "Введите значение: ";
    std::cin >> value;
    if (value) {
        Exercise(TreeCreator::createTreesFromFile()).executeTask();
    } else {
        Exercise(TreeCreator::createTreesFromConsole()).executeTask();
    }
    return 0;
}
```