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 _____ Савельев И.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с деревьями научиться рекурсивно обрабатывать их. Реализовать функции поиска листьев и их вывода на экран, поиска количества узлов на заданном уровне.

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 T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., 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.

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

Задание.

Вариант 3д

3. Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- напечатать элементы из всех листьев дерева b;

- подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня).

Выполнение работы.

Описание структуры данных использованной для дерева:

```
struct Node {
    char data;
    Node* lt=nullptr;
    Node* rt=nullptr;
};
```

Содержит в себе два указателя на левое и правое поддерево соответственно и поле data для содержимого узла.

Описание методов класса.

Node* CreateTree(string& user_input), принимает указатель на строку - user_input, возвращает указатель на структуру Node.

string ReadFile(const string& file_name), принимает указатель на строку - fileName, возвращает строку считанную из файла.

void PrintTreeLeaves(Node *tree), принимает указатель на Node - tree, печатает листья дерева.

void SearchTreeLeaves(Node *tree, int indent), принимает указатель на Node - tree и indent - количество отступов, ищет листья дерева.

int Count(Node *tree, int depth, int cur, int count, int indent), принимает указатель на структуру Node - tree, глубину на которой надо искать - depth, текущую глубину - cur, счетчик - count, глубину отступов - indent. Возвращает количество узлов.

int ScanDeep(), считывает значение введенное пользователем и возвращает его - depth.

void PrintTree(Node *tree, int indent), принимает указатель на Node - tree, количество отступов - indent. Выводит дерево в ступенчатом виде.

void Start(), запускает работу программы.

Алгоритм.

Сначала с файла считывается ввод пользователя с помощью функции ReadFile() в переменную user_input, затем эта переменная передается в функцию CreateTree(), где создается бинарное дерево и ссылка на него возвращается из функции в переменную bin_tree, после чего вызывается функция SearchTreeLeaves(), которая ищет листья дерева и выводит промежуточные результаты и функция PrintTreeLeaves(), которая выводит листья в строчку. После чего функция ScanDeep(), запрашивает у пользователя глубину на которой надо искать узлы и возвращает ее в переменную deep, наконец вызывается функция Count(), которая ищет количество узлов на заданной глубине и выводит промежуточные результаты. В конце вызывается функция Del(), которая очищает память.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 - Результаты тестирования

№ п/п	Входные данные	Выходные данные
1	(a(b(v)(s(l)))(c)) level: 0	leaves: v l c nodes: 0
2	(a(b(v)(s(l)))(c)) level: 1	leaves: v l c nodes: 1
3	(a(b(v)(s(l)))(c)) level: 2	leaves: v l c nodes: 2
4	(a(b(v)(s(l)))(c)) level: 3	leaves: v l c nodes: 2
5	(a(b(v)(s(l)))(c)) level: 4	leaves: v l c nodes: 1
6	(a(b(v)(s(l)))(c)) level: 5	leaves: v l c nodes: 0
7	(a) level: 1	leaves: a nodes: 1
8	(a(b(c(d(e(f)))))) level: 2	leaves: f nodes: 1
9	(a)	leaves: a nodes: 1
10	____	Wrong input !i
11	!@#$%	Wrong input !
12	(a (b(c(d(e(f))))))	Wrong job
13	(990(b(c(d(e(f))))))	Wrong job
14	(a(b(v)(s(l)))(c)) suafuha UFHVUF level: 2	leaves: v l c nodes: 2

Выводы.

В ходе выполнения данной лабораторной работы были изучены принципы реализации деревьев и их рекурсивная обработка в языке C++. Успешно реализованы функции поиска листьев и их вывода на экран, поиска количества узлов на заданном уровне.

Исходный код программы.

```
#include <iostream>
#include <string>
#include <fstream>
#include <iterator>
#include <sstream>

#define FOLDER "readme.txt" // Изменить в соответствии с расположением на
                                                                    // своем компьютере
using namespace std;

// Структура узла Бинарного дерева
struct Node {
	char data;
	Node* lt=nullptr; // Левое поддерево
	Node* rt=nullptr; // Правое поддерево
};

class BinTree {

private:
	Node* bin_tree=nullptr;

public:
	// Очистка выделенной паммяти
	void Del(Node *tree) {
	// Проверка наличия левого поддерева
	if(tree->lt != nullptr)
		Del(tree->lt);
	// Проверка наличия правого поддерева
	if(tree->rt != nullptr)
		Del(tree->rt);
	delete tree;
	}

	// Считывание из файла
	string ReadFile(const string& file_name) {
	ifstream f(file_name);
	stringstream ss;
	ss << f.rdbuf();
	return ss.str();
	}

	// Создание бинарного дерева
	Node* CreateTree(string& user_input) {
	if (user_input[0] != '(') {
		std::cout << "Wrong input !" << '\n';
		exit(0);
	}
	Node* fin_tree = new Node;
	int i = 1;
	fin_tree->data = user_input[i];
	i++;
	int i_lt = i;
	int i_rt = 0;
	if(user_input[i_lt] == '(') {
		int openBrackets = 1;
		int closeBrackets = 0;
		// Подсчет количества скобок в левомподдереве
		while (openBrackets != closeBrackets) {
```

```
	i++;
	if (user_input[i] == '(') {
		openBrackets++;
	}
	else if (user_input[i] == ')')  {
		closeBrackets++;
	}
	}
	// Обрезка строки
	string t = user_input.substr(i_lt, i);
	// Вызов функции для левого поддереве
	fin_tree->lt = CreateTree(t);
	i++;
	i_rt = i;
	// Если првое поддерево отсутствует
	if (user_input[i] == ')'){
	return fin_tree;
	}

	if(user_input[i] == '('){
	int openBrackets = 1;
	int closeBrackets = 0;
	// Подсчет скобок в правом поддереве
	while (openBrackets != closeBrackets) {
		i++;
		if (user_input[i] == '(') {
			openBrackets++;
		}
		else if (user_input[i] == ')') {
			closeBrackets++;
		}
	}
	// Обрезка строки
	string g = user_input.substr(i_rt, i);
	// Вызов функции для правого поддерева
	fin_tree->rt = CreateTree(g);
	}
}
return fin_tree;

}

// Вывод листьев в консоль
void PrintTreeLeaves(Node *tree) {
// Если корень
if(tree->lt == nullptr && tree->rt == nullptr ) {
	std::cout << tree->data << ' ';
	return ;
}
// Если есть потомки
if(tree->lt != nullptr) {
	PrintTreeLeaves(tree->lt);
}
// Если есть потомки
if(tree->rt != nullptr) {
	PrintTreeLeaves(tree->rt);
}
}

// Поиск листьев в дереве
void SearchTreeLeaves(Node *tree, int indent) {
// Выводим нужное количество отступов
for (int i = 0; i < indent; i++){
```

```
            std::cout << ('\t');
        }
        if(tree->lt == nullptr && tree->rt == nullptr ) {
            std::cout << "Leaf " << tree->data << '\n';
            return ;
        }
        else {
            std::cout  << tree->data << " has descendant" << '\n';
        }
        if(tree->lt != nullptr) {
            SearchTreeLeaves(tree->lt, indent + 1);
        }
        if(tree->rt != nullptr) {
            SearchTreeLeaves(tree->rt, indent + 1);
        }
        }


        // Вывод узлов в консоль
        int Count(Node *tree, int depth, int cur, int count, int indent) {
        // Выводим нужное количество отступов
        for (int i = 0; i < indent; i++){
            std::cout << ('\t');
        }
        std::cout << tree->data << " Curent depth: " << cur << '\n';
        if(cur == depth) {
            return 1;
        }
        else {

            if(tree->rt != nullptr && tree->lt != nullptr) {
            return  Count(tree->lt,  depth,  cur  +  1,  count,  indent  +  1)  +
Count(tree->rt, depth, cur + 1, count, indent + 1);
            }
            else if (tree->rt == nullptr && tree->lt != nullptr) {
            return Count(tree->lt, depth, cur + 1, count, indent + 1);
            }
            else if(tree->rt != nullptr && tree->lt == nullptr) {
            return Count(tree->rt, depth, cur + 1, count, indent + 1);
            }
            else {
            return 0;
            }

        }
        }

        // Считывание уровня введеного пользователем
        int ScanDeep() {
        int depth = 0;
        std::cout << "Enter the level at which you want to count the nodes: ";
        // Считывание с консоли
        std::cin >> depth;
        return depth;
        }

        // Вывод дерева в консоль
        void PrintTree(Node *tree, int indent) {
        // Выводим нужное количество отступов
        for (int i = 0; i < indent; i++) {
            std::cout << ('\t');
        }
        // Вывод содержимого узла
```

```
	std::cout << tree->data << '\n';
	// Если есть потомки
	if(tree->lt != nullptr) {
		PrintTree(tree->lt, indent + 1);
	}
	// Если есть потомки
	if(tree->rt != nullptr) {
		PrintTree(tree->rt, indent + 1);
	}
	}

	// Управляющий метод
	void Start(){
	int nodes_count = 0;
	string user_input = ReadFile(FOLDER);
	BinTree tree;
	// Вывод данных введеных пользователем
	std::cout << "You entered tree: " << user_input;
	// Создание бинарного дерева
	bin_tree = tree.CreateTree(user_input);
	// Вывод дерева в консоль
	std::cout << '\n';
	std::cout << "Your tree in the form of a ledge list" << '\n';
	std::cout << '\n';
	tree.PrintTree(bin_tree, 0);
	std::cout << '\n';
	// Поиск листьев
	std::cout << "Search leaves" << '\n' << '\n';
	tree.SearchTreeLeaves(bin_tree, 0);
	// Вывод листьев в консоль
	std::cout << '\n' << "Tree leaves: ";
	tree.PrintTreeLeaves(bin_tree);
	std::cout << "\n\n";
	// Считывание данных введеных пользователем
	int deep = ScanDeep();
	std::cout << '\n';
	std::cout << "Search nodes" << '\n';
	std::cout << '\n';
	// Подсчет количества Узлов
	nodes_count = tree.Count(bin_tree, deep, 1, nodes_count, 0);
	std::cout << "\n" << "Nodes in this level: " << nodes_count << '\n';
	// Освобождение памяти
	tree.Del(bin_tree);
	}

};


int main() {
	BinTree tree;
	tree.Start();
	return 0;
}
```