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

ОТЧЕТ

по практической работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Определения дерева, леса, бинарного дерева.
Скобочное представление

Студент гр. 8304 _____ Кирьянов Д.И.
Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Вариант 6-д

Цель работы

Изучить основные принципы работы с деревьями и лесами и их принципы обработки

Задание

Для заданного леса с произвольным типом элементов:

- получить естественное представление леса бинарным деревом;
- вывести изображение леса и бинарного дерева;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Описание алгоритма

1. Открывается файл, либо считывается с консоли строка.
2. При помощи функции check происходит проверка на корректность подаваемых значений. Если данные введены неверно, то программа сообщает об этом и завершается.
3. Вызывается метод input класса Forest, который выделяет подстроку с деревом из строки с лесом при помощи итератора. Затем из подстрок с деревом формирует деревья и записывает их в vector.
4. Вызывается метод printforest класса Forest, который при помощи очереди выводит лес в ширину. В очередь сначала заносится элемент, затем он выводится, заносятся его дети, после чего удаляется при помощи pop. Завершает работу, когда очередь станет пустой.
5. Вызывается метод createbintree класса Forest, который при помощи рекурсии преобразовывает лес в бинарное дерево и записывает его.
6. Вызывается метод printbintree класса Forest, который выводит запись бинарного дерева, при этом если у узла нет продолжения, то ставится “#”.
7. Вызывается метод print2D класса Forest, который выводит графическую запись бинарного дерева.

Описание основных функций

1. bool check(std::string& array)

Проверка на корректность подаваемых значений.

2. void input(std::string& array)

Метод класса Forest, который динамически записывает лес в памяти.

3. tree* createtree(std::string& array, std::string::iterator start, std::string::iterator end)

Метод, который записывает в памяти одно дерево.

4. void printforest(std::vector <tree*> forest)

Метод, который выводит лес в ширину.

5. void createbintree(std::vector <tree*> forest, bintree* mybin, long unsigned int index)

Метод, который преобразует лес в бинарное дерево и сохраняет его в памяти.

6. void printtree(bintree* mybin)

Метод, который выводит запись бинарного дерева.

7. void print2D(bintree* root)

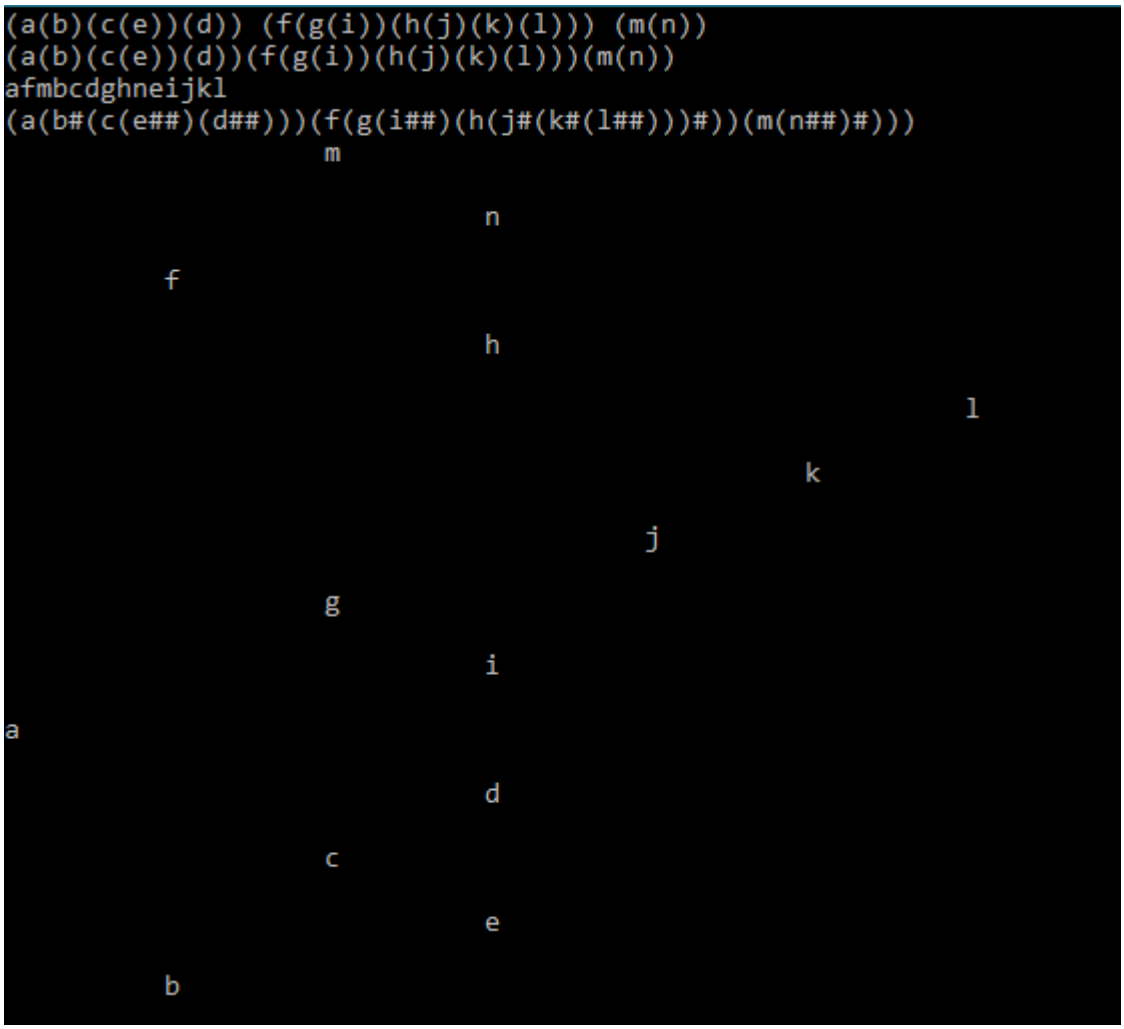
Метод, для графического вывода бинарного дерева.

Вывод.

Была реализована программа, позволяющая строить бинарные деревья по заданной форме, были получены навыки работы с деревьями.

ПРИЛОЖЕНИЕ А

Тестирование программы



ПРИЛОЖЕНИЕ Б

Файл lab4.cpp

```
#include <iostream>
#include <string>
#include <stack>
#include <fstream>
#include <vector>
#include <algorithm>
#include <queue>
#define COUNT 10

std::string::iterator find_end(std::string& str, std::string::iterator start){
	int cntr = 0;
	while (start < str.end()){
		if (*start == '(')
			++cntr;
		if (*start == ')')
			--cntr;
		if (!cntr)
			return start;
		++start;
	}
	return start;
}

template <typename type>
class Forest {
public:
	struct bintree {
		type value;
		bintree* right = nullptr;
		bintree* left = nullptr;
		~bintree() {
			if (right)
				delete right;
			if (left)
				delete left;
		}
	};
	struct tree {
		type value;
		std::vector <tree*> branches;
		~tree() {
			branches.erase(branches.begin(), branches.end());
		}
	};
	std::vector <tree*> forest;
	Forest() = default;
	Forest(const Forest<type>& copy) {
		for (int i = 0; i < (copy.forest).size(); ++i) {
			forest.push_back(copy.forest[i]);
		}

	}
	void input(std::string& array) {
		std::string::iterator start, end;
		start = array.begin();
		end = start;
		while (end != array.end() - 1) {
			end = find_end(array, start);
			forest.push_back(createtree(array, start, end));
			if (end != array.end() - 1)
```

```
                        start = end + 1;
                }
        }

        tree* createtree(std::string& array, std::string::iterator start, std::string::iterator end) {
                tree* res = new tree;
                res->value = type(*(++start));
                ++start;
                while (start <= end) {
                        if (*start == ')')
                                return res;
                        if (*start == '(')
                                res->branches.push_back(createtree(array, start, find_end(array, start)));
                        start = find_end(array, start);
                        if (start != array.end())
                                ++start;
                }
                return res;
        }
        void printforest(std::vector <tree*> forest) {
                std::queue<tree*> qu;
                for (long unsigned int i = 0; i < forest.size(); i++) {
                        qu.push(forest[i]);
                }
                while (!qu.empty()) {
                        std::cout << (qu.front())->value;
                        for (long unsigned int i = 0; i < ((qu.front())->branches).size(); i++) {
                                qu.push((qu.front())->branches[i]);
                        }
                        qu.pop();
                }
        }
        bintree* mybin = new bintree;
        void createbintree(std::vector <tree*> forest, bintree* mybin, long unsigned int index) {
                mybin->value = forest[index]->value;
                if ((forest[index]->branches).size() != 0) {
                        mybin->left = new bintree;
                        createbintree(forest[index]->branches, mybin->left, 0);
                }

                if (index != forest.size()-1) {
                        mybin->right = new bintree;
                        createbintree(forest, mybin->right, index+1);
                }
        }
        void printtree(bintree* mybin) {
                if (!mybin) {
                        std::cout << '#';
                        return;
                }
                std::cout << '(';
                std::cout << mybin->value;
                printtree(mybin->left);
                printtree(mybin->right);
                std::cout << ')';
        }
        void print2DUtil(bintree* root, int space){
                if (root == nullptr)
                        return;
                space += COUNT;
                print2DUtil(root->right, space);
```

```
		std::cout << std::endl;
		for (int i = COUNT; i < space; i++)
			std::cout << " ";
		std::cout << root->value << "\n";
		print2DUtil(root->left, space);
	}
	void print2D(bintree* root){
		print2DUtil(root, 0);
	}

	~Forest() {
		delete mybin;
		forest.erase(forest.begin(), forest.end());
	}
};

bool check(std::string& array) {
	std::stack <char> Stack;
	for (unsigned int i = 0; i < array.length(); ++i) {
		if (array[i] == '(') {
			Stack.push(array[i]);
			if ((array[i + 1] == '(') || (array[i + 1] == ')'))
				return false;
		}
		else if (array[i] == ')') {
			if (Stack.empty()) {
				return false;
			}
			Stack.pop();
		}
		else {
			if (!Stack.empty()) {
				if (array[i - 1] != '(')
					return false;
				if ((array[i + 1] != '(') && (array[i + 1] != ')'))
					return false;
			}
			else
				if (array[i] != ' ')
					return false;
		}
	}
	if (!Stack.empty()) {
		return false;
	}
	return true;
}

int main(int argc, char* argv[]) {

	std::string array;
	if (argc == 1) {
		std::getline(std::cin, array);
		array.erase(std::remove(array.begin(), array.end(), ' '), array.end());
		if (!check(array)) {
			std::cout << "Wrong input" << std::endl;
			return 0;
		}
		std::cout << array << "\n";
		Forest<char>  myForest;
		myForest.input(array);
		myForest.printforest(myForest.forest);
		std::cout << std::endl;
		myForest.createbintree(myForest.forest, myForest.mybin, 0);
		myForest.printtree(myForest.mybin);
```

```
		myForest.print2D(myForest.mybin);
	}
	else {
		std::ifstream in(argv[1]);
		if (!in.is_open()) {
			std::cout << "Can't open file" << std::endl;
			return 0;
		}
		while (std::getline(in, array)) {
			if (!check(array)) {
				std::cout << "Wrong input" << std::endl;
				continue;
			}
			array.erase(std::remove(array.begin(), array.end(), ' '), array.end());
			std::cout << array << "\n";
			Forest<char>  myForest;
			myForest.input(array);
			myForest.printforest(myForest.forest);
			std::cout << std::endl;
			myForest.createbintree(myForest.forest, myForest.mybin, 0);
			myForest.printtree(myForest.mybin);
			myForest.print2D(myForest.mybin);
		}
		in.close();
	}
	return 0;
}
```