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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Программирование алгоритмов с бинарными деревьями
Вариант 1-д

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Постановка задачи.

1. Задано бинарное дерево *b* типа *BT* с типом элементов *Elem*. Для введенной пользователем величины *E* (**var** *E*: *Elem*):

а) определить, входит ли элемент *E* в дерево *b*;

б) определить число вхождений элемента *E* в дерево *b*;

в) найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E* (если *E* не входит в *b*, за ответ принять –1).

Описание алгоритма.

Программа посимвольно считывает каждый элемент. С помощью функции `binTree enterBT(char c, int curr, ifstream& f, int* check, int* openingBracket, int* endingBracket);` создается дерево на основе введенных данных, а также проверяет, бинарное ли это дерево с помощью переменных openingBracket и endingBracket. Далее, с помощью функции `void outBT(binTree b, char elem, int *check, int *count, int length, int* min);` идет подсчет количества запрашиваемых элементов в дерево и длина минимального пути от корня до элемента.

Функция enterBT.

На вход функции подается *c* - введенный элемент, *curr* - длина от корня до текущего элемента, *f* - файл, из которого считываются данные, *check* - переменная для проверки разных случаев, *openingBracket* - переменная для подсчета открывающихся скобок, *endingBracket* - для подсчета закрывающихся. Если считывается '(', то check = 2, если считывается ')', то check = 0, если считывается пробел, то check = 1. Далее, если считывается буква, то идет составление дерева. Если для левой ветки функция считала '(', то функция заново считывает элемент - если считалась буква, то она заносится в левую

ветку, если нет - то левая ветка будет пустой. Если функция считала ')', то элемент считывается еще раз и заносится в правую ветку (левая ветка при считанном символе ')' в любом случае будет пустой).

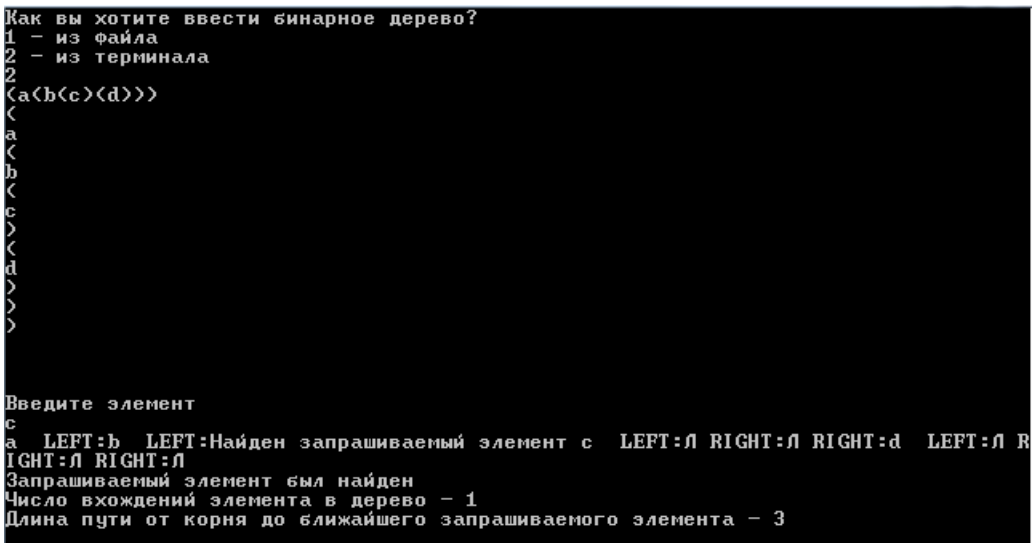
Функция outBT.

На вход функции подается дерево *b*, запрашиваемый элемент *elem*, *check* - переменная, хранящая в себе 1, если элемент был найден и 0 если не был. *count* - счетчик количества найденных элементов, *length* - длина от корня до каждого найденного элемента, *min* - минимальная длина *length*. Работа функции: если корень дерева не пустой, то функция проверяет, тот ли это элемент, который нам нужно найти. Если да, то счетчик count увеличивается на 1, в min заносится длина length, если она был меньше min. То же самое рекурсивно повторяется для левой ветки и правой, при этом length увеличивается на 1.

Спецификация программы.

Программа написана на языке C++. Входные данные подаются либо из терминала, либо из файла. Результат программы выводится на экран.

Тестирование.



Input	Output
(a) ELEM: a	Найден; 1; 1;
() ELEM: c	Не найден

(a(b)(1)) ELEM: a	Это не бинарное дерево
(a(b(e)(d))(e)) ELEM: e	Найден; 2; 2
a(b (c))(d)) ELEM: d	Найден; 1; 2

Вывод.

В ходе выполнения лабораторной работы были приобретены навыки работы с бинарными деревьями, изучены обходы бинарных деревьев и реализовано бинарное дерево на языке C++.

Приложение А.

Содержимое файла header.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <windows.h>
#include <cctype>
#include <string>

using namespace std;

namespace binTree_modul
{

	//-------------------------------------
	typedef char base;

	struct node {
		base info;		//элемент
		node* lt;		//указатель на левую ветку
		node* rt;		//указатель на правую ветку
		node() {
			info = NULL;
			lt = NULL;
			rt = NULL;
		}
	};

	typedef node* binTree; // "представитель" бинарного дерева

	binTree Create(void);
	base RootBT(binTree); // функция, возвращающая корень дерева
	binTree Left(binTree);// функция, возвращающая левую ветку дерева
	binTree Right(binTree);// функция, возвращающая правую ветку дерева
	binTree ConsBT(const base& x, binTree& lst, binTree& rst);  //функция сборки
бинарного дерева
	void destroy(binTree&);   //удаление бинарного дерева

} // end of namespace binTree_modul

namespace binTree_modul
{

	//-------------------------------------
	binTree Create()
	{
		return NULL;
	}
	//-------------------------------------
	base RootBT(binTree b)                    // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
		else return b->info;
	}
	//-------------------------------------
	binTree Left(binTree b)           // для непустого бин.дерева
	{
		if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
		else return b->lt;
	}
	//-------------------------------------
```

```
binTree Right(binTree b)          // для непустого бин.дерева
{
	if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }
	else return b->rt;
}
//-------------------------------------
binTree ConsBT(const base & x, binTree & lst, binTree & rst)
{
	binTree p;
	p = new node;
	if (p != NULL) {
		p->info = x;
		p->lt = lst;
		p->rt = rst;
		return p;
	}
	else { cerr << "Memory not enough\n"; exit(1); }
}
//-------------------------------------
void destroy(binTree & b)
{
	if (b != NULL) {
		destroy(b->lt);
		destroy(b->rt);
		delete b;
		b = NULL;
	}
}

} // end of namespace h_list
```

Приложение Б.

Содержимое файла code.cpp

```
#include "header.h"

using namespace std;
using namespace binTree_modul;

typedef unsigned int unInt;

binTree enterBT(char c, int curr, ifstream& f, int* check, int* openingBracket, int*
endingBracket);       //функция для ввода бинарного дерева
void outBT(binTree b, char elem, int *check, int *count, int length, int* min);
//функция длв вывода элементов дерева и для подсчета нужных сведений о запрашиваемом
элементе
char reading(ifstream &f);        //функция, возвращающая вводимый с терминала/файла элемент

int main()
{
	binTree b;
	int how;
	int curr = 0;
	string s;
	ifstream in;
	char elem;
	char c = NULL;
	int check = 1;
	int checkForElem = 0;
	int count = 0;        //счетчик найденных элементов
	int length = 1;        //длина до каждого найденного элемента
	int min = 100000;    //для расчета минимального пути
	int openingBracket = 0;
	int endingBracket = 0;

	/*SetConsoleCP(1251);                    // для вывода кирилицы
	SetConsoleOutputCP(1251);  // для вывода кирилицы*/
	setlocale(LC_ALL, "RUS");

	cout << "Как вы хотите ввести бинарное дерево?\n1 - из файла\n2 - из терминала\n";
	cin >> how;
	if (how != 1 && how != 2) {
		cout << "Нужно ввести 1 или 2!";
		return 0;
	}

	if (how == 1) {
		string nameOfFile;

		cout << "Доступные файлы для выбора:\n";
		WIN32_FIND_DATA FindFileData;
		HANDLE hf;
		hf = FindFirstFile(".\\*.txt", &FindFileData);
		if (hf != INVALID_HANDLE_VALUE) {
			do {
				cout << FindFileData.cFileName << endl;
			} while (FindNextFile(hf, &FindFileData) != 0);
			FindClose(hf);
		}

		cout << "Введите название файла\n";
		cin >> nameOfFile;
		in.open(nameOfFile);
```

```
		if (!in.is_open()) {
			cout << "Неверное название файла!\n";
			return 0;
		}

		getline(in, s);
		cout << "Строка в файле:\n" << s << endl;
		in.close();
		in.open(nameOfFile);
	}

	cin.sync();
	cin.get();

	b = enterBT(c, curr, in, &check, &openingBracket, &endingBracket);
	cout << endl;

	if (openingBracket != endingBracket || check == 4) {
		cout << "Это не бинарное дерево!" << endl;
		return 0;
	}

	cout << "Введите элемент\n";
	cin >> elem;
	if (!isalpha(elem)) {
		cout << "Неверно введен элемент или это не бинарное дерево\n";
		return 0;
	}

	outBT(b, elem, &checkForElem, &count, length, &min);
	if (checkForElem) {
		cout << "\nЗапрашиваемый элемент был найден\n";
		cout << "Число вхождений элемента в дерево - " << count << endl;
		cout << "Длина пути от корня до ближайшего запрашиваемого элемента - " << min
<< endl;
	}
	else {
		cout << "\nЗапрашиваемый элемент не был найден\n";
		return 0;
	}
	cout << endl;

	destroy(b);
	cout << endl;
	return (0);
}

binTree enterBT(char c, int curr, ifstream& f, int* check, int* openingBracket, int*
endingBracket) {		//функция для ввода бинарного дерева
	binTree p, q;

	if (c == EOF || c == '\n') return NULL;
	c = reading(f);

	if (curr == 0) {
		if (c == '(') {
			(*openingBracket)++;
			c = reading(f);
			if (c == ')') {
				(*endingBracket)++;
				return NULL;
			}
		}
		else {
```

```
                *check = 4;
                return NULL;
        }
    }

    else if (c == ')') {
        *check = 0;
        (*endingBracket)++;
        return NULL;
    }

    else if (isspace(c)) {
        *check = 1;
        return NULL;
    }

    else if (c == '(') {
        (*openingBracket)++;
        *check = 2;
        return NULL;
    }

    if (isalpha(c)) {
        p = enterBT(c, curr + 1, f, check, openingBracket, endingBracket);

        if (*check == 2) {                          //Если открывающаяся скобка
            p = enterBT(c, curr + 1, f, check, openingBracket, endingBracket);
            if (*check != 5) {
                *check = 3;
            }
            else q = NULL;
        }
        else if (*check == 1) {                     //Если пробел
            p = enterBT(c, curr + 1, f, check, openingBracket, endingBracket);
        }

        if (*check == 0) {                           //Если закрывающаяся скобка
            binTree tmp = NULL;
            if (curr != 0) {
                tmp = enterBT(c, curr + 1, f, check, openingBracket,
endingBracket);
            }
            if (*check == 2) q = NULL;
            else if (*check == 0) {
                q = tmp;
                *check = 5;
            }
        }
        else if (*check != 5) {
            q = enterBT(c, curr + 1, f, check, openingBracket, endingBracket);
            if (*check == 0) q = enterBT(c, curr + 1, f, check, openingBracket,
endingBracket);
            if (*check == 2) q = enterBT(c, curr + 1, f, check, openingBracket,
endingBracket);
            else if (*check == 5) enterBT(c, curr + 1, f, check, openingBracket,
endingBracket);
        }
        else if (*check == 5) {
            *check = 3;
            q = NULL;
        }

        return ConsBT(c, p, q);
    }
```

```
	return 0;
}

void outBT(binTree b, char elem, int *check, int *count, int length, int *min) {
//функция вывода запрашиваемых сведений для элемента
	if (b != NULL) {
		if (b->info == elem) {				//если найден запрашиваемый элемент
			if (length < *min) {
				*min = length;
			}
			(*count)++;
			*check = 1;
			cout << "Найден запрашиваемый элемент " << RootBT(b) << " ";
		}
		else cout << RootBT(b) << " ";
		cout << " LEFT:";
		outBT(Left(b), elem, check, count, length+1, min);
		cout << " RIGHT:";
		outBT(Right(b), elem, check, count, length+1, min);
	}
	else cout << "Л";
}

char reading(ifstream& f) {
	char c = NULL;
	if (f.is_open()) c = f.get();
	else c = cin.get();
	cout << c << endl;
	return c;
}
```