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: Бинарные деревья

Студентка гр. 7382 _____ Давкаева В.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Вариант 2(д)

Для заданного бинарного дерева b типа BT с произвольным типом элементов:
а) определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;
б) вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла;
в) напечатать элементы из всех листьев дерева b;
г) подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня)

Содержательная постановка задачи.

На входе программа получает последовательность символов из файла “input.txt”. Для решения поставленных задач используются рекурсивные процедуры.

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом. Так определенное бинарное дерево не является частным случаем дерева.

Определим скобочное представление бинарного дерева (БД):

- < БД > ::= < пусто > | < непустое БД >,
- < пусто > ::= /,
- < непустое БД > ::= (< корень > < БД > < БД >).
- Здесь пустое дерево имеет специальное обозначение - /.

Исходные данные представляют собой строку, содержащую элементы бинарного дерева. Результат работы программы представляет собой числа – значения требуемых выражений.

Спецификация программы

Место и форма представления входных данных: файл

Входные данные : строка

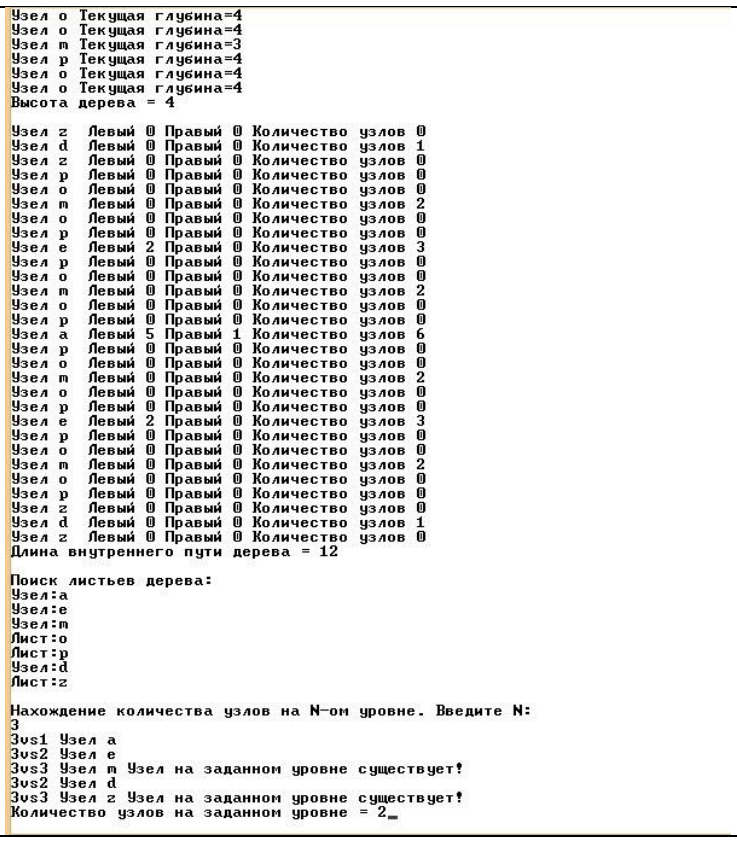
Выходные данные: На экран выводится информация о заданном бинарном дереве

Таблица 1 - Описание функций

Имя функции	Назначение
`binTree enterBT();`	Считывает бинарное дерево из файла .
`int getMaxDepth(binTree q, int depth)`	Считает глубину дерева .Выводит текущую глубину на экран. На вход подается узел бинарного дерева
`void displayBT(binTree b, int n`	Функция вывода бинарного дерева в консоль.
`unInt lenBT(binTree b);`	Считает внутреннюю длину бинарного дерева. На вход подается указатель на текущий узел дерева
`unInt NumOfLvl(binTree b, int a, int c = 1)`	Функция проверяющая существует ли узел на заданном уровне
`void levelBT(binTree b);`	Считает количество уровней бинарного дерева.

ТЕСТИРОВАНИЕ

№ опыта	Входные данные	Исходные данные
1	(a(e(m(o//)(p//))(/)) (d()/(z//)))	Бинарное дерево (повернутое): a d z e m p o Вычисление высоты дерева: Узел a Текущая глубина=1 Узел d Текущая глубина=2 Узел z Текущая глубина=3 Узел z Текущая глубина=3 Узел e Текущая глубина=2 Узел m Текущая глубина=3 Узел p Текущая глубина=4 Узел o Текущая глубина=4 Узел o Текущая глубина=4 Узел m Текущая глубина=3 Узел p Текущая глубина=4 Узел o Текущая глубина=4 Узел o Текущая глубина=4 Узел e Текущая глубина=2 Узел m Текущая глубина=3 Узел p Текущая глубина=4 Узел o Текущая глубина=4 Узел o Текущая глубина=4 Узел m Текущая глубина=3 Узел p Текущая глубина=4 Узел o Текущая глубина=4 Узел o Текущая глубина=4 Высота дерева = 4 Узел z Левый 0 Правый 0 Количество узлов 0 Узел d Левый 0 Правый 0 Количество узлов 1 Узел z Левый 0 Правый 0 Количество узлов 0 Узел p Левый 0 Правый 0 Количество узлов 0 Узел o Левый 0 Правый 0 Количество узлов 0 Узел m Левый 0 Правый 0 Количество узлов 2 Узел o Левый 0 Правый 0 Количество узлов 0 Узел p Левый 0 Правый 0 Количество узлов 0 Узел e Левый 2 Правый 0 Количество узлов 3 Узел p Левый 0 Правый 0 Количество узлов 0 Узел o Левый 0 Правый 0 Количество узлов 0 Узел m Левый 0 Правый 0 Количество узлов 2 Узел o Левый 0 Правый 0 Количество узлов 0 Узел p Левый 0 Правый 0 Количество узлов 0 Узел a Левый 5 Правый 1 Количество узлов 6 Узел p Левый 0 Правый 0 Количество узлов 0 Узел o Левый 0 Правый 0 Количество узлов 0 Узел m Левый 0 Правый 0 Количество узлов 2 Узел o Левый 0 Правый 0 Количество узлов 0 Узел p Левый 0 Правый 0 Количество узлов 0 Узел e Левый 2 Правый 0 Количество узлов 3 Узел p Левый 0 Правый 0 Количество узлов 0 Узел o Левый 0 Правый 0 Количество узлов 0 Узел m Левый 0 Правый 0 Количество узлов 2 Узел o Левый 0 Правый 0 Количество узлов 0 Узел p Левый 0 Правый 0 Количество узлов 0

		
2	(/)	**Пустое БД**
3	(a(e(m//())(/))(d()/(z//)))	Высота БД 3 Внутренний путь БД 6 Уровень 3 Количество узлов на данном уровне 2
4	(a//)	Высота БД 1 Внутренний путь 0 Уровень 1 Количество узлов на данном уровне 1
5	(a(o//)(i//))	Высота БД 2 Внутренний путь 2 Уровень 2 Количество узлов на данном уровне 2

Рассмотрим работу алгоритма на примере теста 5

1. Программа считывает бинарное дерево и запускает функцию displayBT, которая строит изображение данного дерева.
   1.1 displayBT для заданного дерева:
   Сначала выводится корень a;
   Затем проверяются левое и правое поддерево. За счет использовани рекурсии выводятся значения узлов и листьев, при этом делаются отступы для удобства понимания.
2. Далее с помощью функции getMaxGepth находится высота дерева. С помощью рекурсии и функции max алгоритм проходит по всем узлам, и

выдает самую большую найденную глубину. В данном случае функция возвращает 2, так как o и i находятся на уровне 2.

3. Потом находится длина внутреннего пути дерева с помощью функции lenBT и sizeBT. Функция рекурсивно проходится по всем узлом и, используя счетчик вычисляет длину пути. В данном случае алгоритм заходит в корень a, а от него в правое поддерево o и левое поддерево i, в каждом случае вызывается функция sizeBT, в которой прибавляется единица. В результате возвращается значение 3.
4. Следующий шаг-поиск листьев дерева, осуществляемый через функцию levelBT. На примере заданного дерева, программа заходит в корень a, проверят наличие правого и левого поддеревьев, так как указатели не пусты переходим к проверке этих поддеревьев. У них в свою очередь отсутствуют правое и левое поддерево, и они выводятся на экран как листья.
5. Находим количество узлов на заданном уровне. Функция работает рекурсивно, проходя все уровни до заданного, и в момент когда мы доходим до нужного уровня на консоль выводим информацию о существовании узла на заданном уровне.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

Btree.h

```
// интерфейс АТД "Бинарное дерево"(в процедурно-модульной парадигме)
namespace binTree_modul
{

    //---------------------------------------
    typedef char base;
    struct node {
        base info;
        node *lt;
        node *rt;
        // constructor
        node() { lt = NULL; rt = NULL; }
    };

    typedef node *binTree; // "представитель" бинарного дерева
    binTree Create(void);
    bool isNull(binTree);
    bool RooBT(binTree);
    base RootBT(binTree); // для непустого бин.дерева
    binTree Left(binTree);// для непустого бин.дерева
    binTree Right(binTree);// для непустого бин.дерева
    binTree ConsBT(const base &x, binTree &lst, binTree &rst);
    void destroy(binTree&);

} // end of namespace binTree_modul

/*
1) RootBT: NonNullBT -> α;
2) Left: NonNullBT -> BT;
3) Right: NonNullBT -> BT;
4) ConsBT: α * BT * BT -> NonNullBT;
5) Null: BT -> Boolean;
6) Create:  -> BT
*/
```

bt_func.cpp

```
// Implementation - Реализация АТД "Бинарное дерево" (в процедурно-модульной парадигме)
#include <iostream>
#include <cstdlib>
#include "Btree.h"
using namespace std;

namespace binTree_modul
{

    //---------------------------------------
    binTree Create()
    {
        return NULL;
    }
    //---------------------------------------
    bool isNull(binTree b)
    {
        return (b == NULL);
    }
    //---------------------------------------
    base RootBT(binTree b)              // для непустого бин.дерева
```

```
    {
        if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
        else return b->info;
    }
    bool RooBT(binTree b)
    {
        if (b == NULL) return false;
        else return true;
    }
    //-------------------------------------
    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
```

Исходный код.cpp

```
// Пример работы с АТД "Бинарное дерево" (в процедурно-модульной парадигме)
#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include "Btree.h"
#include <windows.h>
#include <conio.h>
#include <algorithm>

using namespace std;
using namespace binTree_modul;

typedef unsigned int unInt;

binTree enterBT();
```

```
void displayBT(binTree b, int n);
unInt sizeBT(binTree b);
unInt lenBT(binTree b);
unInt NumOfLvl(binTree b, int a, int c = 1);
int getMaxDepth(binTree b, int depth = 1);
void levelBT(binTree b);
ifstream infile("KLP.txt");

int main()
{
    binTree b;
    SetConsoleCP(1251);                  // для вывода кирилицы
    SetConsoleOutputCP(1251);     // для вывода кирилицы

    b = enterBT();
    int depth = 1;
    if (isNull(b)) cout << "Пустое БД" << endl;
    else
    {
        cout << "Бинарное дерево (повернутое): " << endl;
        displayBT(b, 1);
        cout << "\nВычисление высоты дерева: " << endl;
        int a = getMaxDepth(b) - 1;
        cout << "Высота дерева = " << a << "\n\n";
        cout << "Длина внутреннего пути дерева = " << lenBT(b) << endl;
        cout << "\nПоиск листьев дерева:" << endl; levelBT(b);
        cout << "\nНахождение количества узлов на N-ом уровне. Введите N:" << endl;
        int num;
        cin >> num;
        cout << "Количество узлов на заданном уровне = " << NumOfLvl(b, num);

        destroy(b);

    }
    _getch();
}

//---------------------------------------
binTree enterBT()
{
    char ch;
    binTree p, q;
    infile >> ch;
    while (ch == '('||ch==')') infile >> ch;
    if (ch == '/'){ return NULL; }
    else { p = enterBT(); q = enterBT();
    return ConsBT(ch, p, q); }
}

int getMaxDepth(binTree q, int depth)
{

    if (q != NULL)cout << "Узел " << q->info << " Текущая глубина=" << depth << endl;
    if (q == NULL){ return depth; }
    else return max(getMaxDepth(q->rt, depth + 1), getMaxDepth(q->lt, depth + 1));
}


//---------------------------------------
void displayBT(binTree b, int n)
{      // n - уровень узла
    if (b != NULL) {
        cout << ' ' << RootBT(b);
        if (!isNull(Right(b))) { displayBT(Right(b), n + 1); }
```

```
        else cout << endl; // вниз
        if (!isNull(Left(b))) {
            for (int i = 1; i <= n; i++) cout << "   "; // вправо
            displayBT(Left(b), n + 1);
        }
    }
    else {};
}

void levelBT(binTree b)
{
    if (!isNull(b))
    {
        if (!(isNull(b->lt) && isNull(b->rt)))
            cout << "Узел:" << b->info << endl;
        levelBT(b->lt);
        if (isNull(b->lt) && isNull(b->rt))
             cout <<"Лист:"<< b->info<<endl;

        levelBT(b->rt);

    }
}

unInt NumOfLvl(binTree b, int lvl, int lvl_now)
{
    if (b != NULL&&lvl >= lvl_now)cout << lvl << "vs" << lvl_now << " Узел " << b->info;
    {
        if (lvl == lvl_now) cout << " Узел на заданном уровне существует!\n";
        else cout << "\n";
    }
    if (lvl == lvl_now) return 1;
    else if(lvl>=lvl_now)return(((b->lt) ? NumOfLvl(b->lt, lvl, lvl_now + 1) : 0) + ((b->rt)
? NumOfLvl(b->rt, lvl, lvl_now + 1) : 0));
}

int len = 0;
unInt lenBT(binTree b)
{
    if(!isNull(b))cout << "Узел " << b->info << " " << " Левый " << lenBT(Left(b)) << "
Правый " << lenBT(Right(b)) << " Количество узлов " << sizeBT(b) - 1 << endl;
    if (isNull(b)) { return 0; }
    else
    {
        return lenBT(Left(b)) + lenBT(Right(b)) + sizeBT(b) - 1;
    }
}
//---------------------------------------
unInt sizeBT(binTree b)
{
    if (isNull(b)) return 0;
    else return sizeBT(Left(b)) + sizeBT(Right(b)) + 1;
}
```