**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: Бинарное дерево**

Студент гр. 9382 _____ Демин В.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с базовыми функциями обработки бинарных деревьев на языке программирования C++.

**Задание.**

**Вариант 3в.**

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).

**Основные теоретические сведения.**

Традиционно иерархические списки представляют или графически или в виде скобочной записи. На рисунке 1 приведен пример графического изображения списка на базе вектора. Соответствующая этому изображению сокращенная скобочная запись – это (a (b c) d e).

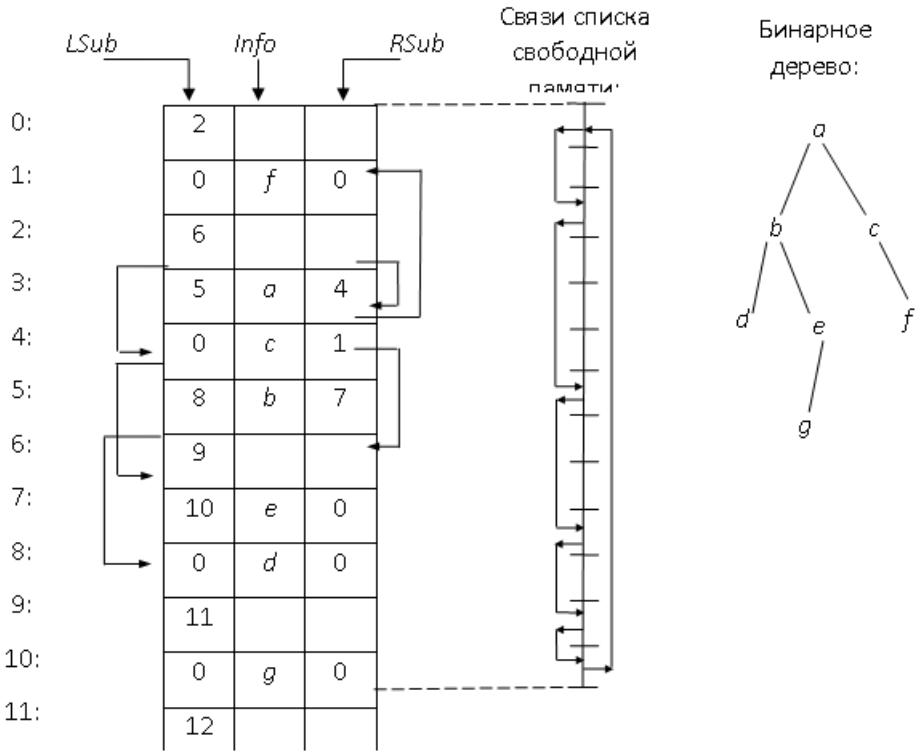


рис. 1

**Алгоритм.**

1. Чтобы напечать листья дерева, необходимо обойти узлы дерева, и вывести те, у которых ветви имеют нулевой указатель. Обход дерева прямой, то есть сначала работаем с корнем узла, а далее обходим другие деревья рекурсивно.

2. Чтобы найти количество узлов на одном заданном уровне, нужно также обойти дерево и сравнивать уровень рекурсии с заданным уровнем. Если они равны, то мы не заходим в рекурсию глубже, а обходим другие ветви.

**Функции и структуры данных.**

Class treeNode – класс узла дерева, хранит в себе корень и ветви дерева.

Class binaryTree – класс бинарного дерева который хранит в себе корень всего дерева, и методы обработки.

Методы класса binaryTree:

- isNull- проверяет пустый ли узел
- root – получает корень узла
- cons – соединяет узлы в новый узел
- writeNide – рекурсивная функция вывод элементов дерева
- writeLeaflet – рекурсивная функция вывода листьев дерева
- writeLevel – рекурсивная функция, считающая количество узлов одного уровня
- writeTree – интерфейс для вызова рекурсивной функции вывода элементов дерева
- enterTree – рекурсивная функция ввода дерева из консоли
- enterTreeFromFile - рекурсивная функция ввода дерева из файла
- writeWithLevel - вывод количества узлов одного уровня

Функция enter и writeResult необходимы для пользователя, которой выберет какой ввод и вывод ему нужен: файл или консоль.

**Выводы.**

В данной задаче были изучены основные методы обработки бинарного дерева.

**Тестирование.**

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | abd/g///ce//fi//jk///<br>5 | Tree elements: a b d g c e f i j k<br>Tree leaves:g e i k<br>Tree elements with levels 5 : 1 | |
| 2. | a//<br>2 | Tree elements: a<br>Tree leaves:a<br>Count tree elements with level 2 :0 | |
| 3. | ab///<br>2 | Tree elements: a b<br>Tree leaves:b<br>Count tree elements with level 2 :1 | |
| 4. | ab//c//<br>2 | Tree elements: a b c<br>Tree leaves:b c<br>Count tree elements with level 2 :2 | |
| 5. | a/b/c/d/e/f/ | Tree elements: a b c d e f<br>Tree leaves:f<br>Count tree elements with level 6 :1 | |
| 6. | abcdefg//////// | Tree elements: a b c d e f g<br>Tree leaves: g<br>Count tree elements with level 2 :1 | |

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
//
// Created by vikto on 04.11.2020.
//
#include <iostream>
#include <fstream>

using namespace std;
template<typename T>
class binaryTree;
//класс узла дерева
template<typename T>
class treeNode {
public:
    ~treeNode() {
    }

    treeNode() {
        nullBool = true;
        leftIndex = -1;
        rightIndex = -1;
    }

    bool nullBool;
    T root;//корень
    int leftIndex;//левая ветвь
    int rightIndex;//правая ветвь
};

//класс для вывода промежуточных значений в файл
class logger {
    string str;
    std::FILE *file;
    treeNode<char>* BT;

public:
```

```
        char root(int bt) {
            return (BT + bt)->root;
        }
        //конструктор класса длв вывода промежуточных значений
        logger() : file(std::fopen("intermeiate.txt", "w+")) {
            if (!file)
                throw std::runtime_error("file open failure");
        }

        //закрытие файла
        ~logger() {
            cout<<    "\nIntermeiate    values    write    to    file
\"intermeiate.txt\"";
            if (std::fclose(file) != 0) {
            }
        };
        bool isNull(int bt) {
            if(bt<0){
                return true;
            }
            return (BT + bt)->nullBool;
        }

        //рекурсивная функция для вывода промежуточных деревьев
        void logTree(int tree, int level) {
            if (!isNull(tree)) {
                logTree(BT[tree].leftIndex, level + 1);
                for (int i = 0; i < level; i++) fputs("   ", file);
                if            (isNull(BT[tree].leftIndex)            &&
isNull(BT[tree].rightIndex)) {
                    str = "(" + std::string(1, BT[tree].root) + ")\n";
                    fputs(str.c_str(), file);
                } else {
                    str = string(1,  BT[tree].root) + "\n";
                    fputs(str.c_str(), file);
                }
                logTree(BT[tree].rightIndex, level + 1);
            }
```

```
        }

        //вызов рекурсивной функции
        void logWriteTree(int tree, int level) {
            if (!isNull(tree)) {
                fputs("-----------------------------------------
\n", file);
                logTree(tree, level);
            }
        }

        //начало для файла промежуточных значений
        void logStartEnter() {
            fputs("StartEnter\n", file);
            fputs("Leaves in brackets\n", file);
        }
        void setBT(treeNode<char>* bt){
            BT=bt;
        }
    };

    static logger *log = new logger();
    template<typename T>
    class binaryTree {
        ~binaryTree() {
            delete BT;
        }

        //проверка пустое ли дерево
        bool isNull(int bt) {
            if(bt<0){
                return true;
            }
            return (BT + bt)->nullBool;
        }

        //получение корня узла дерева
        T root(int bt) {
```

```
        return (BT + bt)->root;
    }

    //создание узла дерева
    int cons(T elem, int left, int right) {
        treeNode<T> *a = &BT[countTree];
        countTree++;
        a->root = elem;
        a->leftIndex = left;
        a->rightIndex = right;
        a->nullBool = false;
        return countTree - 1;
    }

    //рекурсивная функции вывода элементов дерева
    void writeNode(int bt, ofstream &file) {
        if (file.is_open()) {
            if (!isNull(bt)) {
                file << root(bt) << " ";
                writeNode(BT[bt].leftIndex, file);
                writeNode(BT[bt].rightIndex, file);
            }
        } else {
            if (!isNull(bt)) {
                cout << root(bt) << " ";
                writeNode(BT[bt].leftIndex, file);
                writeNode(BT[bt].rightIndex, file);
            }
        }
    }

    //рекурсивная функция вывод листьев дерева
    void writeLeaflet(int bt, ofstream &file) {
        if (file.is_open()) {
            if (!isNull(bt)) {
                if (!isNull(BT[bt].leftIndex) &&
!isNull(BT[bt].leftIndex)) {
                    file << root(bt) << " ";
```

```
                    }
                    writeLeaflet(BT[bt].leftIndex, file);
                    writeLeaflet(BT[bt].rightIndex, file);
                }
            } else {
                if (!isNull(bt)) {
                    if           (!isNull(BT[bt].leftIndex)           &&
!isNull(BT[bt].leftIndex)) {
                        cout << root(bt) << " ";
                    }
                    writeLeaflet(BT[bt].leftIndex, file);
                    writeLeaflet(BT[bt].rightIndex, file);
                }
            }

        }

        //рекурсивная функция вывода узлов с уровнем
        void writeLevel(int bt, int level, int n, int *count) {
            if (!isNull(bt)) {
                if (n == level) {
                    (*count)++;
                    return;
                }
                writeLevel(BT[bt].leftIndex, level + 1, n, count);
                writeLevel((BT[bt].rightIndex), level + 1, n, count);
            }
        }

    public:
        treeNode<T> *BT;
        int countTree;
        int mem;
        int rootTree;

        //создание пустого дерева
        binaryTree() : countTree(0), mem(0), rootTree(0) {
            this->BT = new treeNode<T>[100];
```

```
        this->mem = 100;
        log->setBT(BT);
    }



    //вызов рекурсивной функции для вывода элементов дерева
    void writeTree(ofstream &file) {
        writeNode(this->rootTree, file);
    }

    //вызов рекурсивной функции для вывода листьев дерева
    void writeLeaves(ofstream &file) {
        writeLeaflet(this->rootTree, file);
    }


//рекурсивная функция ввода дерева из консоли
    int enterTree() {
//        log->logWriteTree(BT, 1);
        char ch;
        int l;
        int r;
        cin >> ch;

        if (ch == '/') return -1;
        else {
            l = enterTree();
            log->logWriteTree(l, 0);
            r = enterTree();
             log->logWriteTree(r, 0);
            return cons(ch, l, r);
        }
    }

//рекурсивная функция ввода дерева из файла
    int enterTreeFromFile(ifstream *file) {
        char ch;
```

```
        int l;
        int r;
        *file >> ch;
        if (ch == '/') return -1;
        else {
            l = enterTreeFromFile(file);
            log->logWriteTree(l, 0);
            r = enterTreeFromFile(file);
            log->logWriteTree(r, 0);
            return cons(ch, l, r);;
        }
    }

//вывод количества узлов одного уровня
    void writeWithLevel(ofstream &file, int n) {
        if (file.is_open()) {
            int count = 0;
            writeLevel(this->rootTree, 1, n, &count);
            file << count;

        } else {
            int count = 0;
            writeLevel(this->rootTree, 1, n, &count);
            cout << count;
        }

    }

};

//интерфейс
void enter(binaryTree<char> *a) {
    int type = 0;
    cout << "Enter from console -1\n";
    cout << "Enter from file -2\n";
    cin >> type;
    ifstream file;
    string name;
```

```
    log->logStartEnter();
    switch (type) {
        case 2:
            cout << "Enter file name\n";
            cin >> name;
            file.open(name);
            if (file.is_open()) {
                a->rootTree = a->enterTreeFromFile(&file);
                 log->logWriteTree(a->rootTree, 0);
                file.close();
            } else cout << "Unable to open file";
            break;
        case 1:
            a->rootTree = a->enterTree();
             log->logWriteTree(a->rootTree, 0);
            break;
        default:
            cout << "incorrect type\n";
    }
}

void writeResult(binaryTree<char> *a) {
    int type = 0;
    int n = 0;
    cout << "Print to console -1\n";
    cout << "Print to file -2\n";
    cin >> type;
    cout << "At what level to count the number of tree nodes?\n";
    cin >> n;
    ofstream file;
    string name_f;
    switch (type) {
        case 1: {
            cout << "Tree elements: ";
            a->writeTree(file);
            cout << "\n";
            cout << "Tree leaves:";
            a->writeLeaves(file);
```

```
                cout << "\n";
                cout << "Count tree elements with level " << n << " :";

                a->writeWithLevel(file, n);
            }
                break;
            case 2:
                cout << "Enter file name\n";
                cin >> name_f;
                file.open(name_f);
                if (file.is_open()) {
                    file << "Tree elements: ";
                    a->writeTree(file);
                    file << "\n";
                    file << "Tree leaves:";
                    a->writeLeaves(file);
                    file << "\n";
                    file << "Tree elements with levels: " << n << " :";
                    a->writeWithLevel(file, n);
                    cout << "the final result is written to the file
\"" + name_f << "\"" << endl;
                } else cout << "Unable to open file";
                break;
            default:
                cout << "error: selection of output type";
                break;
        }

    }

    int main() {
        binaryTree<char> *a = new binaryTree<char>();
        enter(a);
        writeResult(a);
        delete log;
        return 0;
    }
```