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

# ОТЧЕТ

**по лабораторной работе №3**

**по дисциплине «АиСД»**

**Тема: Бинарные деревья**

Студент гр. 9303 _____ Емельянов С.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

**Цель работы.**

Создание программы для вывода бинарного дерева в ширину с помощью очереди.

**Задание.**

Вариант 6 (д)

Задано бинарное дерево b типа BT с произвольным типом элементов.

Используя очередь, напечатать все элементы дерева b по уровням: сначала – из корня дерева, затем (слева направо) – из узлов, сыновних по отношению к корню, затем (также слева направо) – из узлов, сыновних по отношению к этим узлам, и т. д.

**Основные теоретические положения.**

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка.

Описать такую структуру можно следующим образом:

```
struct point
{
 int data;//информационное поле
 point *left;//адрес левого поддерева
 point *right;//адрес правого поддерева
};
```

- Начальный узел называется корнем дерева. Узел, не имеющий поддеревьев, называется листом. Исходящие узлы называются предками, входящие — потомками. Высота дерева определяется количеством уровней, на которых располагаются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется деревом поиска. Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу,

двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле. Такой поиск гораздо эффективнее поиска по списку, поскольку время поиска определяется высотой дерева, а она пропорциональна двоичному логарифму количества узлов.

В идеально сбалансированном дереве количество узлов справа и слева отличается не более чем на единицу.

Линейный список можно представить как вырожденное бинарное дерево, в котором каждый узел имеет не более одной ссылки. Для списка среднее время поиска равно половине длины списка.

Деревья и списки являются рекурсивными структурами, т. к. каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

- либо пустой структурой;
- либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

**Выполнение работы.**

Для решения поставленной задачи была разработана программа в среде Visual Studio Code на языке C++.

Для реализации бинарного дерева был написан класс BinTree.

BinTree *lSub – поле хранящее ссылку на левый объект класса BinTree.

BinTree *rSub – поле хранящее ссылку на правый объект класса BinTree.

string info – поле хранящее содержание узла дерева.

BinTree* GetrSub() – метод, возвращающий указатель на правый объект.

BinTree* GetlSub() – метод, возвращающий указатель на левый объект.

BinTree* SetlSub(BinTree* lSub) – метод, изменяющий левый указатель.

BinTree* SetrSub(BinTree* rSub) – метод, изменяющий правый указатель.

Для реализации очереди и прохода бинарного дерева в ширину, был реализован класс stack.

Основные методы это pop() и get() – позволяют взять первый элемент в очереди. push(T x) – позволяет добавить элемент в конец очереди.

Поля int counter и int begin хранят размер очереди и начало очереди.

void ShowBinThreeDir (BinTree* head, stack<BinTree>& st, ofstream& fout) – функция, которая выводит бинарное дерево в ширину ступенчатым способом.

Исходный код программы представлен в приложении А. Результаты тестирования приведены в файле result1.txt , который прилагается к отчёту.

**Выводы.**

Были изучены теоретические материалы по бинарным деревьям, были разработаны алгоритмы по выводу и обработке бинарных деревьев. Был разработан класс stack для вывода бинарного дерева в ширину.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <fstream>
#include <unistd.h>
using namespace std;

template <class T>
class stack
{
private:
    T* Buffer;
    int size;
    void addSize(){
        size += 50;
        T *tmp = new T [size];
        for (int i = 0; i < counter; i++)
        {
            tmp[i] = Buffer[i];
        }
        delete Buffer;
        Buffer = tmp;
    }
public:
    int counter;
    int begin;
    stack(){
        size = 50;
        counter = 0;
        begin = 0;
        Buffer = new T [size];
    }
    void pop(){
```

```
        begin++;

    }
    T* getBuffer(int n){
        return &Buffer[n];
    }

    T get(){
        return Buffer[begin];
    }
    void push(T x){
        if( counter >= size-1 )
            addSize();

        Buffer[counter] = x;
        counter++;
    }
    bool isEmpty(){
        return (this->counter == this->begin);
    }
};

class BinTree
{

public:
    BinTree *lSub;
    BinTree *rSub;
    string info;
    int level;
    BinTree* GetrSub(){
        return rSub;
    }
    BinTree* GetlSub(){
```

```
            return lSub;
        }
        BinTree* SetlSub(BinTree* lSub){
            this->lSub = lSub;
        }
        BinTree* SetrSub(BinTree* rSub){
            this->rSub = rSub;
        }
        BinTree(){
            info = "*";
            lSub = nullptr;
            rSub = nullptr;
        }
    };

    void CreatBinTree(BinTree* b, string& str, int& n, int level){
        if (n >= str.length() ) return;
        if (str[n] == '/'){
            n++;
            b->level = level;
            return;
        }
        else{
            b->info = str[n];
            b->level = level;
            n++;
            b->lSub = new BinTree;
            CreatBinTree(b->lSub,str,n, level + 1);
            b->rSub = new BinTree;
            CreatBinTree(b->rSub,str,n, level + 1);
        }
    }
```

```
    void ShowBinThreeDir (BinTree* head, stack<BinTree>& st,
ofstream& fout){
        st.push(*head);
        while(!st.isEmpty()){
            if (st.get().info != "*"){
                if (head->lSub!=nullptr){
                    st.push(*st.get().lSub);
                }
                if (head->rSub!=nullptr){
                    st.push(*st.get().rSub);
                }
            }
            for(int i = 0; i < st.get().level; i++){
                fout<<"    ";
            }
            fout<<st.get().info<<"\n";
            st.pop();

        }
        return;
    }

    int main(){
        //BinTree b;
        ifstream fin;
        ofstream fout;
        int n = 0;
        fin.open("KLP.txt");
        fout.open("result1.txt", ios_base::app);
        string str ;
        while(!fin.eof()){
            BinTree b;
            stack<BinTree> st;
            n = 0;
```

```
            getline(fin,str);
                    fout<<  "\nИсходные  данные:  "  <<  str<<"\
n"<<"Результат:\n";
            CreatBinTree(&b,str,n, 0);
            ShowBinThreeDir(&b, st, fout);
        }
        fin.close();
        fout.close();
        return 0;
    }
```