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: Деревья

Студент гр. 9382 _____ Дерюгин Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научится создавать бинарные деревья через динамическую память, освоить применение классов, научится работать с шаблонами.

Теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m □ 0 попарно не пересекающихся множествах T 1 , T 2 , ..., T m , каждое из которых, в свою очередь, является деревом. Деревья T 1 , T 2 , ..., T m называются поддеревьями данного дерева.

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт б в определении дерева можно было бы сформулировать так: *узлы дерева, за исключением корня, образуют лес.*

Задание.

Вариант 10д

Рассматриваются бинарные деревья с элементами типа *Elem* (в качестве *Elem* использовать *char*). Заданы перечисления узлов некоторого дерева *b* в порядке ЛКП и ЛПК. Требуется:

- восстановить дерево *b* и вывести его изображение;
- перечислить узлы дерева *b* в порядке КЛП.

Описание алгоритма.

На вход подается два выражения, которые являются записями одного и того же бинарного дерево(ЛКП, ЛПК). В записи ЛПК последний символ является корнем первого уровня. Находим этот же корень в записи ЛКП и делим ЛКП на 2 части: все символы, которые находились левее корня, входят в левое поддерево данного дерева, а символы, которые находятся правее данного корня, входя в правое поддерево данного дерева. Таким образом мы

вытаскиваем левое и правое поддерево. С данными деревьями проделываем тоже самое, до тих пор, пока не окончатся символы.

Структура бинарного дерева.

```
struct Node {
    Elem data;// root data
    Node* left;//left subtree
    Node* right;// right subtree
```

Data - символ, который является корнем данного дерева.

Left - левое поддерево

Right - правое поддерево

Описание функций и остальных структур.

enum Errors - перечисление, в котором хранятся ошибки, которые могут произойти при работе программы

Void errors(Errors error) - функция обработки ошибок, на вход передается перечисление, ничего не возвращает. Выводит в консоль ошибку.

Class BinaryTree - класс бинарного дерева. Содержит Структуру Node, которая описана выше, и методы для работы с деревом.

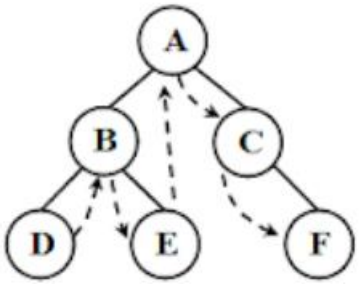
Node* createBinaryTree(string lkp, string lpk) - рекурсивная функция. На вход подаются 2 строки, которые являются ЛКП и ЛПК представлением бинарного дерева. Создает новое бинарное дерево. Возвращает это дерево.

void printIntermediateResult(Node *tree) - на вход подается дерево, выводит промежуточные значения, а именно корень дерева и левое и правое поддерево. Ничего не возвращает.

void printKLP(Node* binTree) - рекурсивная функция, которая выводит дерево в КЛП. На вход подается дерево, ничего не возвращает.

void printTree(Node* binTree, int space) - рекурсивная функция, которая выводит графическое представление бинарного дерева. На вход подаются дерево и отступ. Ничего не возвращает.

Пример работы программы.



Результат:

#

f

#

c

#

a

#

e

#

b

#

d

#

Тестирование.

№	Входные данные	Вызодные данные	Комментарии
1	dbeacf debfca	You entered: LKP: dbeacf; LPK: debfca Image of subtree: b / \ d e	

		---------- Image of subtree: c / \ # f ---------- Image of subtree: a / \ b c / \ \ ---------- KLP is: abdecf Tree: # f # c # a # e # b # d # 	
2	abcdefghi acedbhigf	You entered: LKP: abcdefghi; LPK: acedbhigf Image of subtree:	Действительно, первый символ должен быть «(«

		d / \ c e ---------- Image of subtree: b / \ a d / \ ---------- Image of subtree: i / \ h # ---------- Image of subtree: g / \ # i / ---------- Image of subtree: f / \ b g / \ \ ---------- KLP is: fbadcegih	
3	abcdefghi acedbhigffsdf	You entered: LKP: abcdefghi; LPK: acedbhigffsdf Incorrect data	Обходы не являются одним и тем же деревом

4		You entered: LKP: ; LPK: Incorrect data	Ввод пустой строки.

Выводы.

В ходе данной работы было реализовано бинарное дерево на основе рекурсии и построено бинарное дерево на основе ЛКП и ЛПК обходов.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include <iostream>
#include <fstream>

using namespace std;
//enum for errors
enum Errors {
    CANNOT_OPEN_FILE,
    INCORRECT_DATA
};

void error(Errors error) {
    if (error == CANNOT_OPEN_FILE) cout<<"Cannot open file\n";
    if (error == INCORRECT_DATA) cout<<"Incorrect data\n";
}// end error func

template <typename Elem>
class BinaryTree {
    struct Node {
        Elem data;// root data
        Node* left;//left subtree
        Node* right;// right subtree
    };

    Node* createBinaryTree(string lkp, string lpk) {
        // if hasn't right leaf
        if (lkp.size() == 0 && lpk.size() == 0) return nullptr;
        //create new leaf
        if (lkp == lpk && lkp.size() == 1) {
            Node* tree = new Node;
            tree->data = lpk[0];
            tree->left = nullptr;
            tree->right = nullptr;
```

```
        return tree;
    }
    Node* tree = new Node;// create tree
    int indexOfRoot;// index of current root
    tree->data = lpk[lpk.size() - 1];// last symbol is root
    indexOfRoot = lkp.find(tree->data);
    if (indexOfRoot < 0) {
        error(INCORRECT_DATA);
        exit(1);
    }
    // if hasn't right leaf
    if (indexOfRoot == 0) tree->left = nullptr;
    else tree->left = createBinaryTree(lkp.substr(0, indexOfRoot), lpk.substr(0, indexOfRoot));
    tree->right = createBinaryTree(lkp.substr(indexOfRoot + 1, lkp.size() - indexOfRoot - 1),
lpk.substr(indexOfRoot, lkp.size() - indexOfRoot - 1));
    printIntermediateResult(tree);
    return tree;
}

void printIntermediateResult(Node *tree) {
    //print tree and left and right subtree
    cout<<"Image of subtree:\n";
    cout<<"     "<<tree->data<<endl;
    cout<<"    /";
    cout<<" \\"<<endl;
    cout<<"   ";
    if (tree->left) cout<<tree->left->data;
    else cout<<"#";
    cout<<"   ";
    if (tree->right) cout<<tree->right->data;
    else cout<<"#";
    cout<<endl;
    if (tree->left && tree->left->left) cout<<"  /";
    else cout<<"  ";
    if (tree->left&& tree->left->right) cout<<"  \\";
```

```
        else cout<<"  ";
        if (tree->right &&tree->right->left) cout<<"  /";
        else cout<<"  ";
        if (tree->right && tree->right->right) cout<<" \\";
        else cout<<"  ";
        cout<<endl<<"-----------------"<<endl;
    }
public:
    Node* tree;
    BinaryTree(string lkp, string lpk) {
        cout<<"You entered:\nLKP: "<<lkp<<";\nLPK: "<<lpk<<endl;
        //if lkp size != lpk size
        if (lkp.size() != lpk.size()) {
            error(INCORRECT_DATA);
            exit(1);
        }
        int indexOfRoot;// index of current root
        tree = new Node;// create tree
        tree->data = lpk[lpk.size() - 1];//last symbol is root
        indexOfRoot = lkp.find(tree->data);// find index of root
        if (indexOfRoot < 0) {
            error(INCORRECT_DATA);
            exit(1);
        }
        // create left subtree
        if (indexOfRoot == 0) tree->left = nullptr;
        else tree->left = createBinaryTree(lkp.substr(0, indexOfRoot), lpk.substr(0, indexOfRoot));
        //create right subtree
        tree->right = createBinaryTree(lkp.substr(indexOfRoot + 1, lkp.size() - indexOfRoot - 1),
lpk.substr(indexOfRoot, lkp.size() - indexOfRoot - 1));
        printIntermediateResult(tree);
    }

    void printKLP(Node* binTree) {
        //print root-left-right tree
```

```
        cout<<binTree->data;
        if (binTree->left) printKLP(binTree->left);
        if (binTree->right) printKLP(binTree->right);
    }

    void printTree(Node* binTree, int space) {
        if (!binTree) {
            for(int i = 0; i < space; i++) cout<<"\t";
            cout<<"#"<<endl;
            return;
        }
        printTree(binTree->right,space+1);
        for(int i = 0 ; i < space; i++) cout<<"\t";
        cout<<binTree->data<<endl;
        printTree(binTree->left,space+1);

    }

};


int main() {
    string path = "input.txt";// path to input file
    int typeOfInput;// 1 if console
    string lkp, lpk;
    cout<<"Enter '1' if you wanna write down binary tree in console otherwise write down any letter or number:\n";
    cin>>typeOfInput;
    //input from console
    if (typeOfInput == 1){
        cout<<"Enter binary tree(LKP):\n";
        cin>>lkp;
        cout<<"Enter binary tree(LPK):\n";
        cin>>lpk;
    }
```

```
    else {
        //open file
        ifstream fin;
        fin.open(path);
        //if cannot open file
        if (!fin.is_open()) {
            error(CANNOT_OPEN_FILE);
            exit(1);
        }
        //reading file line by line
        getline(fin, lkp);
        getline(fin, lpk);
        fin.close();//close file
    }
    //print result
    BinaryTree<char> binTree(lkp, lpk);
    cout<<"KLP is: ";
    binTree.printKLP(binTree.tree);
    cout<<endl;
    cout<<"Tree:";
    //print tree
    binTree.printTree(binTree.tree, 0);
    return 0;
}
```