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

ОТЧЕТ

по практической работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Программирование алгоритмов с бинарными деревьями

Студент гр. 7382 _____ Находько А.Ю.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Познакомиться с нелинейной структурой данных бинарное дерево. Бинарное дерево используется при решении задач кодирования и поиска. Познакомится со способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Задание.

Вариант 8д. Рассматриваются бинарные деревья с элементами типа char. Заданы перечисления узлов некоторого дерева b в порядке ЛКП и ЛПК. Требуется:

а) восстановить дерево b и вывести его изображение;

б) перечислить узлы дерева b в порядке КЛП.

Пояснение задачи.

Требуется чтобы программа сопоставляя ЛКП и ЛПК записи дерева, определяла его вид и восстанавливала его, корректно выводила изображение. После построения дерева правильно записывала КЛП запись узлов дерева.

Основные теоретические положения.

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями.

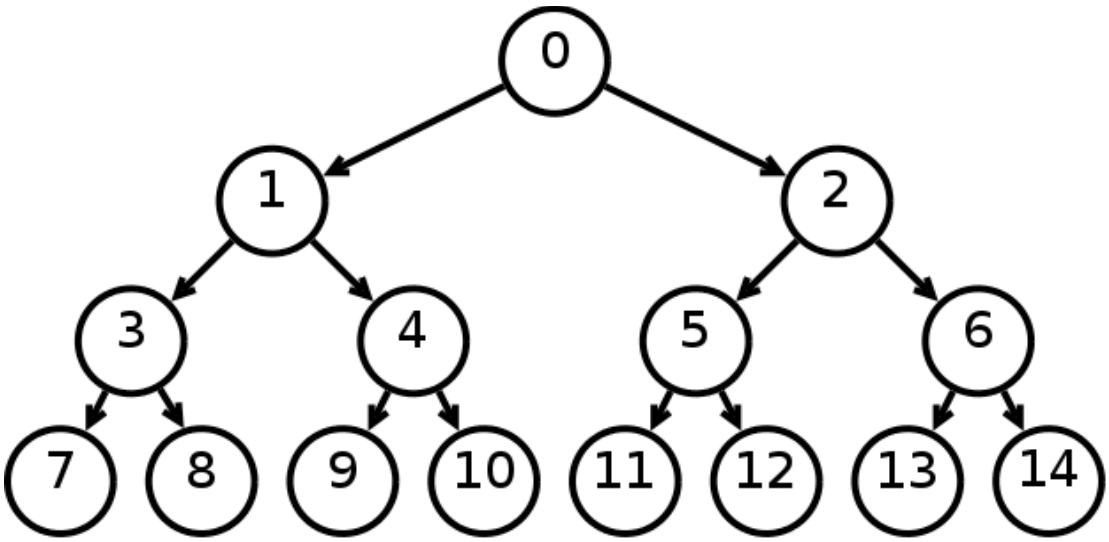


Рисунок 1. Бинарное дерево

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

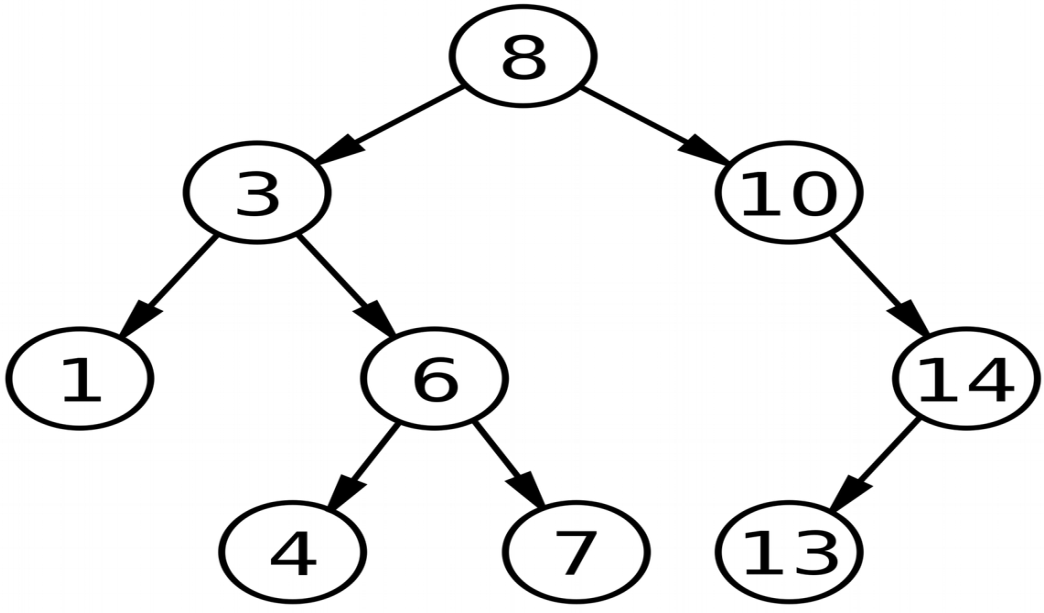


Рисунок 2. Бинарное дерево поиска

Операции обхода дерева - прямой (КЛП),обратный (ЛКП), концевой (ЛПК).

Выполнение работы.

1) Заводим переменные типа string под ЛКП, ЛПК записи узлов и для копирования содержимого потока.
2) Заводим переменную count для подсчёта ЛКП и ЛПК записей.
3) С помощью функции getline помещаем содержимое файла потока в строку s_count. Произведём подсчёт строк в файле и запишем их число в переменную count. Если count число чётное, значит каждой ЛКП записи соответствует ЛПК запись. Иначе программа не будет выполняться.
4) С помощью функции seekg передвинем указатель на начало входного потока.
5) Заведём цикл for с начальным значением i=0, шагом i++, условие окончание: когда все записи узлов будут проанализированы.
6) С помощью функции getline произведём извлечение ЛКП и ЛПК записей из потока, если их длины не раны – значит они не соответствуют друг другу.
7) Произведём вызов рекурсивной функции recovery, передавая в неё записи узлов и местоположение корня с наименьшим уровнем.
8) В функции recovery, если: запись узлов пуста, местоположение корня с наименьшим уровнем число отрицательное - выход из функции.
9) Заводим переменную флаг, которая хранит индекс x в ЛКП записи.
10) Если записи состоят из одного одинакового элемента, значит это дерево из одного узла – возвращаем его в main.
11) Создаём элемент типа binTree, который и будет являться результатом восстановления дерева из ЛКП и ЛПК записей.
12) С помощью функции substr возвращаем flag символов из ЛКП записи узлов начиная с 0. Также возвращаем flag символов из ЛПК записи узлов начиная с 0. И присваиваем x_r значение flag-1.
13) Прямой рекурсией вызываем функцию recovery с вновь полученными параметрами для нахождения индекса левого узла.

14) С помощью функции substr возвращаем flag+1 символов из ЛКП записи. Также возвращаем x-1 символов из ЛПК записи узлов начиная с flag. И присваиваем x_r значение x-1-flag.
15) Прямой рекурсией вызываем функцию recovery с вновь полученными параметрами для нахождения индекса правого узла.
16) Когда дерево будет восстановлено – возвращаем его.
17) Произведём вывод изображения дерева в консоль, выведем КЛП запись узлов дерева используя функцию printKLP.

Описание функций.

DisplayBT – функция, которая принимает дерево и по его представлению строит изображение и выводит его на консоль.

RootBT – функция взятия корня поддерева.

isNull – функция проверки на нулевой узел.

Right – функция взятия индекса правого узла.

Left – функция взятия индекса левого узла.

printKLP – функция, которая принимает дерево, и возвращает КЛП запись этого дерева.

recovery – рекурсивная функция восстановления дерева по ЛПК и ЛКП записям узлов исходного дерева. В качестве аргументов функция получает ЛКП и ЛПК записи узлов дерева, x – местоположение корня с наименьшим уровнем.

node – структура дерева, которая содержит индексы левого и правого узла.

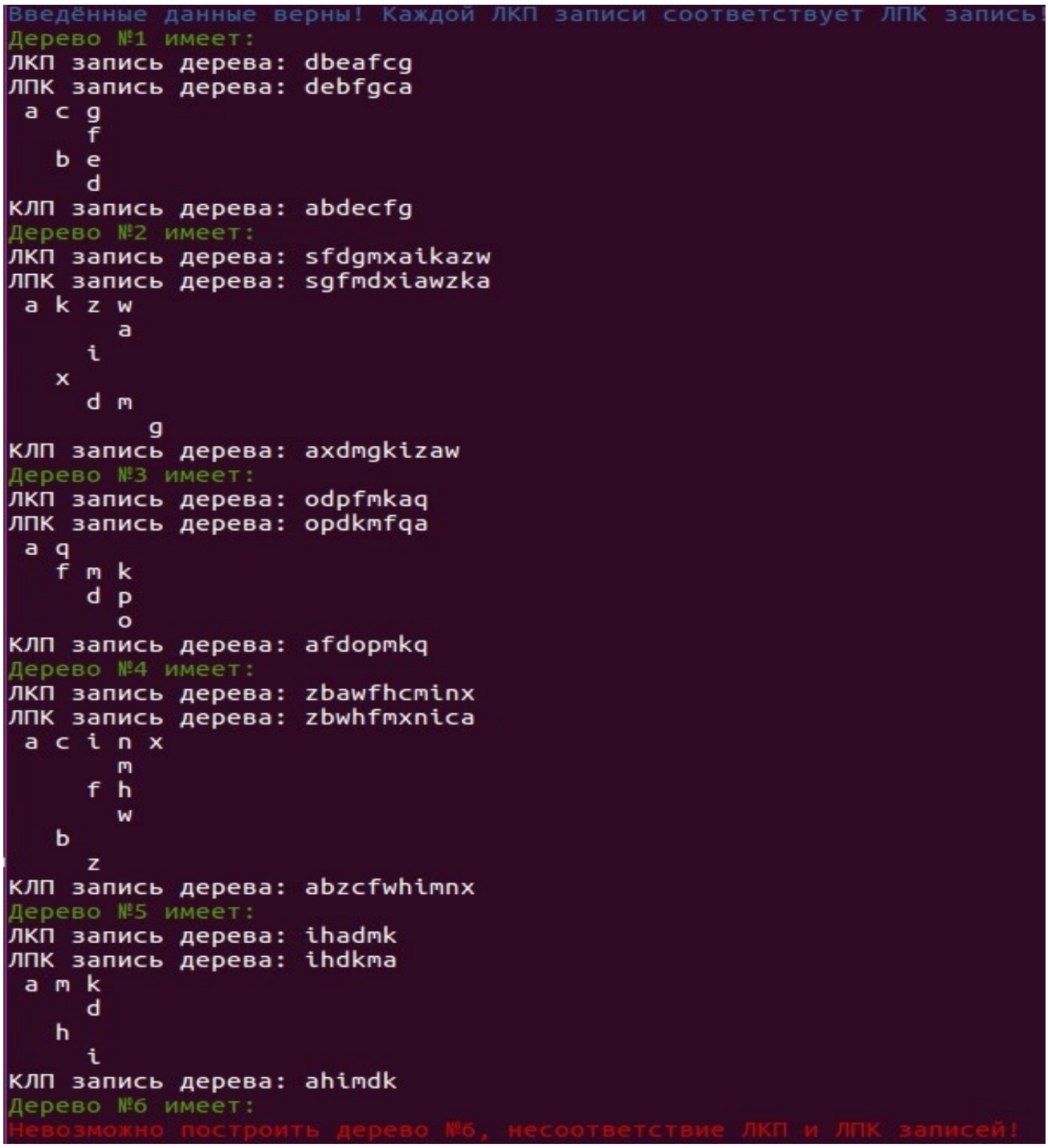
Тестирование.

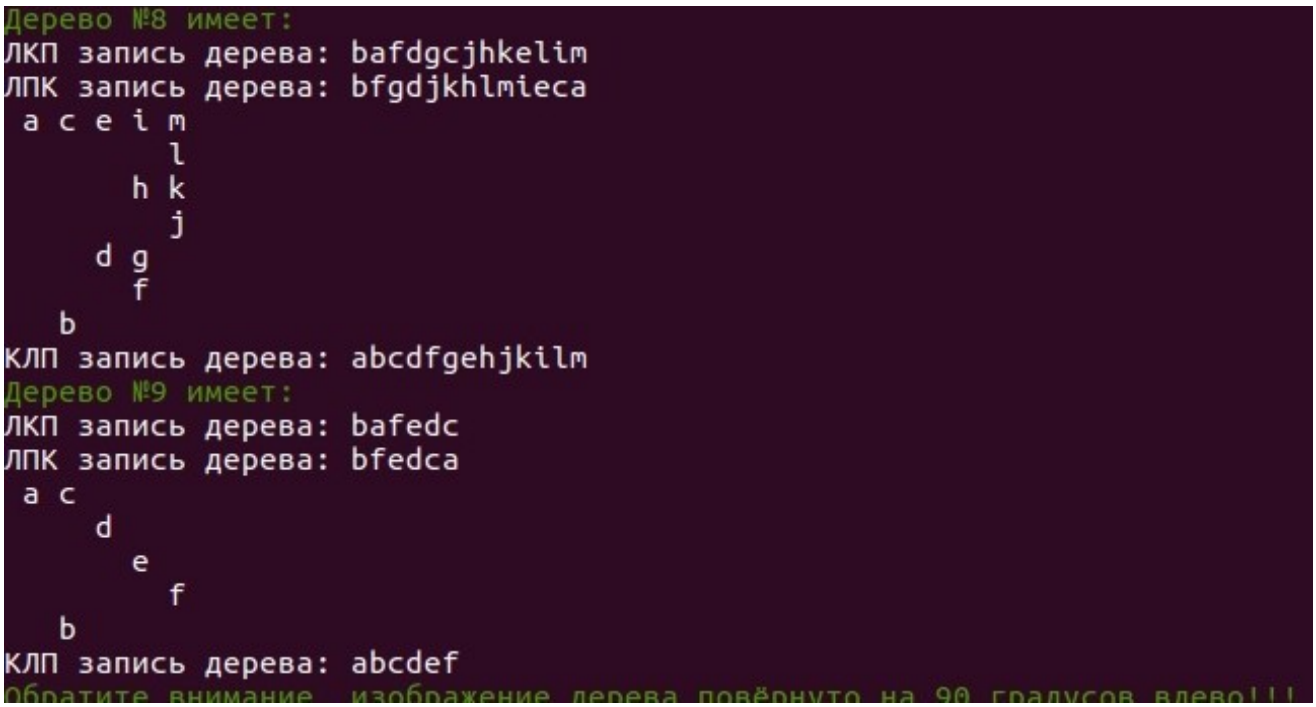
Таблица тестирования программы.

№ теста	**Входные данные:**	**Результат:**
1	dbeafcg debfgca	Рисунок дерева
2	sfdgmxaikazw sgfmdxiawzka	Рисунок дерева

3	odpfmkaq opdkmfqa	Рисунок дерева
4	zbawfhcminx zbwhfmxnica	Рисунок дерева
5	ihadmk ihdkma	Рисунок дерева
6	ruhwuowrohewue wefihewfhoohewfohiewfoewohew	Несоответствие ЛКП И ЛПК запсей
7	hdibgekalfmcngo hidgkeblmfnogca	Рисунок дерева
8	bafdgcjhkelim bfgdjkhlmieca	Рисунок дерева
9	bafedc bfedca	Рисунок дерева

Иллюстрации работы тестирования:


Введённые данные верны! Каждой ЛКП записи соответствует ЛПК запись!
Дерево №1 имеет:
ЛКП запись дерева: dbeafcg
ЛПК запись дерева: debfgca
a c g
f
b e
d
КЛП запись дерева: abdecfg
Дерево №2 имеет:
ЛКП запись дерева: sfdgmxaikazw
ЛПК запись дерева: sgfmdxiawzka
a k z w
a
i
x
d m
g
КЛП запись дерева: axdmgkizaw
Дерево №3 имеет:
ЛКП запись дерева: odpfmkaq
ЛПК запись дерева: opdkmfqa
a q
f m k
d p
o
КЛП запись дерева: afdopmkq
Дерево №4 имеет:
ЛКП запись дерева: zbawfhcminx
ЛПК запись дерева: zbwhfmxnica
a c i n x
m
f h
w
b
z
КЛП запись дерева: abzcfwhimnx
Дерево №5 имеет:
ЛКП запись дерева: ihadmk
ЛПК запись дерева: ihdkma
a m k
d
h
i
КЛП запись дерева: ahimdk
Дерево №6 имеет:
Невозможно построить дерево №6, несоответствие ЛКП и ЛПК записей!



Описание алгоритма

На примере теста №5.

1) Заводим переменные под ЛКП, ЛПК записи узлов, строку для записи потока, переменную count – в которую будем записывать количество строк.

2) Копируем содержимое файла потока в строку s_count. Найдём в нём количество строк. В данном случае count=2.

3) count – число чётное, следовательно выполнено соответствие ЛПК и ЛКП записей.

4) С помощью функции seekg возвращаем указатель на начало потока.

5) Заводим цикл for с начальным значением i=0, шагом i++, условие окончание: когда записи узлов будут проанализированы.

6) С помощью функции getline произведём извлечение ЛКП и ЛПК записей из потока, их длины не раны – значит они не соответствуют друг другу.

7) Произведём вызов рекурсивной функции recovery, передавая в неё записи узлов и местоположение корня с наименьшим уровнем. В данном случае ЛКП=”ihadmk”, ЛПК=”ihdkma”, x=5.

8) В функции recovery: запись узлов не пуста, местоположение корня с наименьшим уровнем число положительное.
9) Заводим переменную flag, которая хранит индекс x в ЛКП записи. flag=2.
10) Создаём элемент типа binTree, который и будет являться результатом восстановления дерева из ЛКП и ЛПК записей.
11) С помощью функции substr возвращаем flag символов из ЛКП записи узлов начиная с 0. lkp_r=”ih”. Также возвращаем flag символов из ЛПК записи узлов начиная с 0. lpk_r=”ih” И присваиваем x_r значение flag-1. x_r=1.
12) Прямой рекурсией вызываем функцию recovery с вновь полученными параметрами для нахождения индекса левого узла.
13) Краткое содержание следующего вызова: 2)r->info='h', lpk_r=’i’, lkp_r=’i’, x_r=0.
14) При последующем вызове ЛКП запись состоит из одного элемента, значит это – лист. И его родитель h, который исходит из корня дерева a.
15) Начнём поиск правого поддерева. При следующем вызове значения параметров:4)lkp_r=”dmk”, lpk_r=”dkm”, x_r=2.
16) При последующих вызовах окажется, что d является листом и его родителем является m. Также и k.
17) После выполнения данных шагов получится восстановленное дерево. Для которого выведем его изображение на консоль под углом 90 градусов. И выведем его КЛП запись: ahimdk.

Выводы.

В результате выполнения лабораторной работы ознакомился с нелинейной структурой данных бинарное дерево. Получил основные сведения о бинарных деревьях. Мною были изучены базовые функции обработки бинарных деревьев.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

BinTree.h

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>

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);
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&);
}
```

BinTree.cpp

```
#include <iostream>
#include <cstdlib>
#include "BinTree.h"
using namespace std ;

namespace binTree_modul
{
binTree Create()
{ return NULL;
}
```

```
bool isNull(binTree b)
{ return (b == NULL);
}

base RootBT (binTree b) // .
{ if (b == NULL) { cerr « "Error: RootBT(null) \n"; exit(1); }
else return b->info;
}

binTree Left (binTree b) // .
{ if (b == NULL) { cerr « "Error: Left(null) \n"; exit(1); }
else return b ->lt;
}

binTree Right (binTree b) // .
{ 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;
}
}


}
```

main.cpp

```
#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include "BinTree.h"
#include <string>
#include <stdio.h>
#define RED "\033[0;31m"
#define COL "\033[0;34m"
#define COL2 "\033[0;32m"
#define NONE "\033[0m"
using namespace std;
using namespace binTree_modul;

void displayBT(binTree b, int n) //функция вывода дерева
{ // 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 printKLP (binTree b) //функция вывода узлов дерева в порядке КЛП
{ if (!isNull(b))
{
cout « RootBT(b);
printKLP (Left(b));
printKLP (Right(b));
}
}

binTree recovery(string &lkp, string &lpk, int x) { //Рекурсивная функция,
которая принимает записи узлов в порядке ЛКП и ЛПК, позицию корня и
```

```
возвращает построенное по ним дерево
if(lkp == "")
return NULL;
if (x < 0) //Если номер текущего элемента отрицательный
return NULL;
if(lkp.length()==1) //Условие, когда длинна ЛКП записи равна 1
{
binTree p = new node;
p->info = lkp[0];
return p; //Тогда дерево состоит из одного узла
}
int flag = lkp.find(lpk[x]); //Заводим переменную flag, которая хранит
номер текущего элемента x из записи ЛПК в ЛКП записи
if(flag < 0) //Если номер окажется отрицательным
return NULL;
binTree r = new node; //дерево, которое будем возвращать, если
соблюдены все условия выше
r->info = lpk[x];
string lkp_r=lkp.substr(0, flag); //функция substr возвращает flag символов
из ЛКП записи узлов начиная с 0
string lpk_r=lpk.substr(0, flag); //функция substr возвращает flag символов
из ЛПК записи узлов начиная с 0
int x_r=flag-1;
r->lt=recovery(lkp_r, lpk_r, x_r); //прямой рекурсией вызываем функцию
recovery с полученными параметрами

return r; //Возвращаем полученное дерево
}

int main() {
string str_lkp, str_lpk, s_count; //Строки для ЛКП и ЛПК записи узлов
ifstream fin("input.txt"); //Файл потока
if(!fin)
{
cout « "Невозможно открыть файл!\n";
return 0;
}
int count=0; //Счётчик символов "/n"
getline(fin, s_count, '\0'); //Считываем всё содержимое потока в строку
s_count
for(int i=0; s_count[i]!='\0'; i++){ //Подсчитаем количество символов
переноса строки
if (s_count[i]=='\n'){
count++;
}
}
if(count%2==0) //Если число чётное - выполнено соответствие,
продолжаем выполнение программы
printf("%sВведённые данные верны! Каждой ЛКП записи соответствует
ЛПК запись!%s\n", COL, NONE);
else{ //Иначе выход из программы
```

```
printf("%sНеверно введены данные. Несоответствие ЛКП и ЛПК строк.
%s\n", RED, NONE);
return 0;
}
fin.seekg(0); //Передвигаем каретку на начало потока
for(int i=0; i<count/2; i++){ //Цикл, пока не будут просмотрены все записи
printf("%sДерево №%d имеет:%s\n", COL2, i+1, NONE);
getline(fin, str_lkp); //Отделяем функцией getline из файла input.txt -
строчную ЛКП запись узлов
getline(fin, str_lpk); //Отделяем функцией getline из файла input.txt -
строчную ЛПК запись узлов
if(str_lkp.length()!=str_lpk.length())
{
printf("%sНевозможно построить дерево №%d, несоответствие ЛКП и ЛПК
записей!%s\n", RED, i+1, NONE);
continue;
}
else
{
cout « "ЛКП запись дерева: ";
cout « str_lkp « endl;
cout « "ЛПК запись дерева: ";
cout « str_lpk « endl;
binTree Tree = recovery(str_lkp, str_lpk, str_lpk.length()-1); //Передаём
полученные параметры в функцию построения дерева
displayBT(Tree, 1); //Выведем дерево на консоль под углом 90 градусов
cout « "КЛП запись дерева: ";
printKLP(Tree); //Выведем КЛП запись файла на консоль
cout « endl;
}
}
printf("%sОбратите внимание, изображение дерева повёрнуто на 90
градусов влево!!!%s\n", COL2, NONE);
return 0;
}
```