**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9383 |  | Рыжих Р.В. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2020

**Цель работы.**

На практике изучить бинарные деревья, а также лес деревьев, реализовав программу при помощи языка С++.

**Задание.**

**Вариант 8 (указатели)**

(Обратная задача.) Для заданного бинарного дерева с произвольным типом элементов:

* получить лес, естественно представленный этим бинарным деревом;
* вывести изображение бинарного дерева и леса;
* перечислить элементы леса в горизонтальном порядке (в ширину).

**Основные теоретические положения.**

*Дерево* – конечно множество Т, состоящее из одного или более узлов, так что

а) имеется один специально обозначенный узел, называемый корнем

данного дерева;

б) остальные узлы (исключая корень) содержатся в m >= 0 попарно не

пересекающихся множествах Т1,Т2,..., Тm , каждое из которых, в свою

очередь, является деревом. Деревья Т1,Т2,...,Тm 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*.

**Выполнение работы:**

В данной задаче требуется считать бинарное дерево и получить лес, естественно представленный этим деревом.

*Class Tree*

*Tree* – класс бинарного дерева. В данном классе содержится переменная info типа char, которая хранит корень бинарного дерева, а также lt и rt – указатели на левое и правое поддерево. Также присутствуют методы доступа к приватным полям класса и методы их изменения.

*Функция bool isNull()*

Данная функция проверяет, является ли переданное бинарное дерево пустым.

*Функция char root()*

Данная функция возвращает корень бинарного дерева.

*Функция BinTree left()*

Функция возвращает левое поддерево бинарного дерева.

*Функция BinTree right()*

Функция возвращает левое поддерево бинарного дерева.

*Функция BinTree cons()*

Данная функция – конструктор. При создании нового бинарного дерева требуется выделение памяти. Если памяти нет, то вызывается соответствующая ошибка. Если «хвост» не атом, то для его присоединения к «голове» требуется создать новый узел (элемент), головная ссылка которого будет ссылкой на «голову» этого «хвоста», а хвостовая часть элемента ссылкой на его «хвост». В ином случае бинарное дерево инициализируется при помощи входных данных.

*Функция readTree()*

Данная функция предназначена для рекурсивного считывания бинарного дерева.

*Функция printBinTree()*

Данная функция рекурсивно печатает введенное бинарное дерево, если оно не пустое.

*Функция startOut()*

Данная функция начинает печать дерева. Если дерево пустое, то на экран выводится, что нечего печатать. В ином случае, функция запускает функцию out()

*Функция out()*

Принимает на вход дерево и пустую строку (изначально), которая увеличивается с увеличением глубины рекурсии. Сначала печатается правое поддерево, затем корень, затем левое поддерево (правое и левое поддеревья также выводятся с помощью ф-ии out()).

*Функция destroy()*

Функция удаляет бинарное дерево.

*Функция que()*

Данная функция реализует обход леса в ширину. Узлы бинарного дерева проходят слева направо, уровень за уровнем от корня вниз. Этот горизонтальный обход использует очередь.

Функция printForest()

В данной функции создаются два дерева, каждый из которых принимает значение левого и правого поддерева соответственно. Далее вызываются функции startOut() и que() для каждого дерева, чтобы вывести изображение леса, а также напечатать перечисление элементов леса в горизонтальном порядке (в ширину).

*Функция main()*

В данной функции происходит считывание бинарного дерева. Вызываются функции для печати введенного бинарного дерева, вывода изображения этого дерева и леса, состоящего из его поддеревьев, а также печать элементов леса в горизонтальном порядке. В завершении программы память очищается.

**Пример работы программы.**

На рисунках 1-5 представлены примеры работы программы с различными исходными данными.

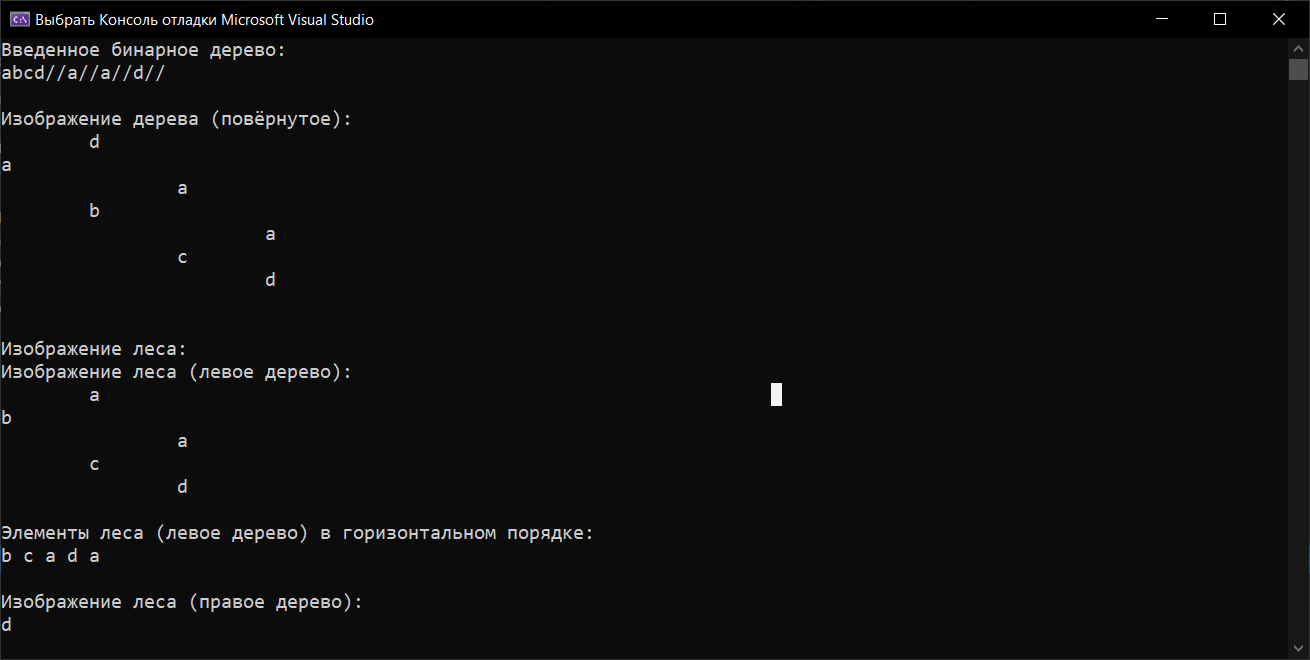


Рисунок 1 - Пример работы с входными данными № 1

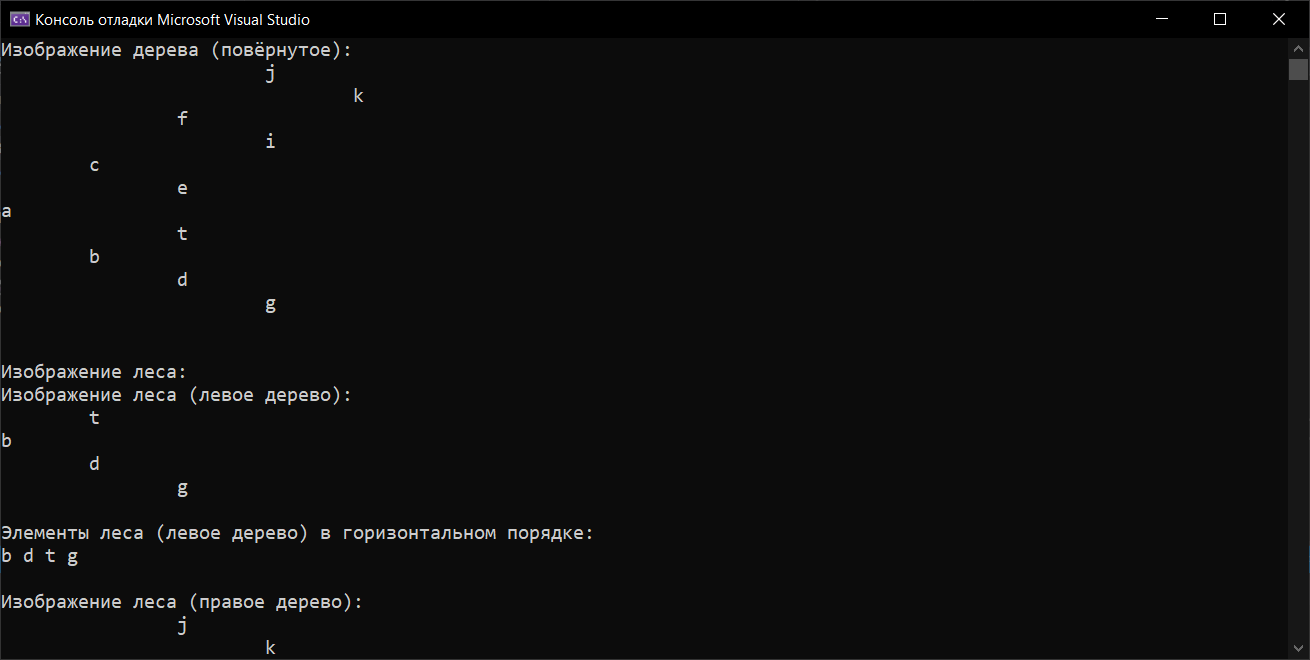


Рисунок 2 - Пример работы с входными данными № 2

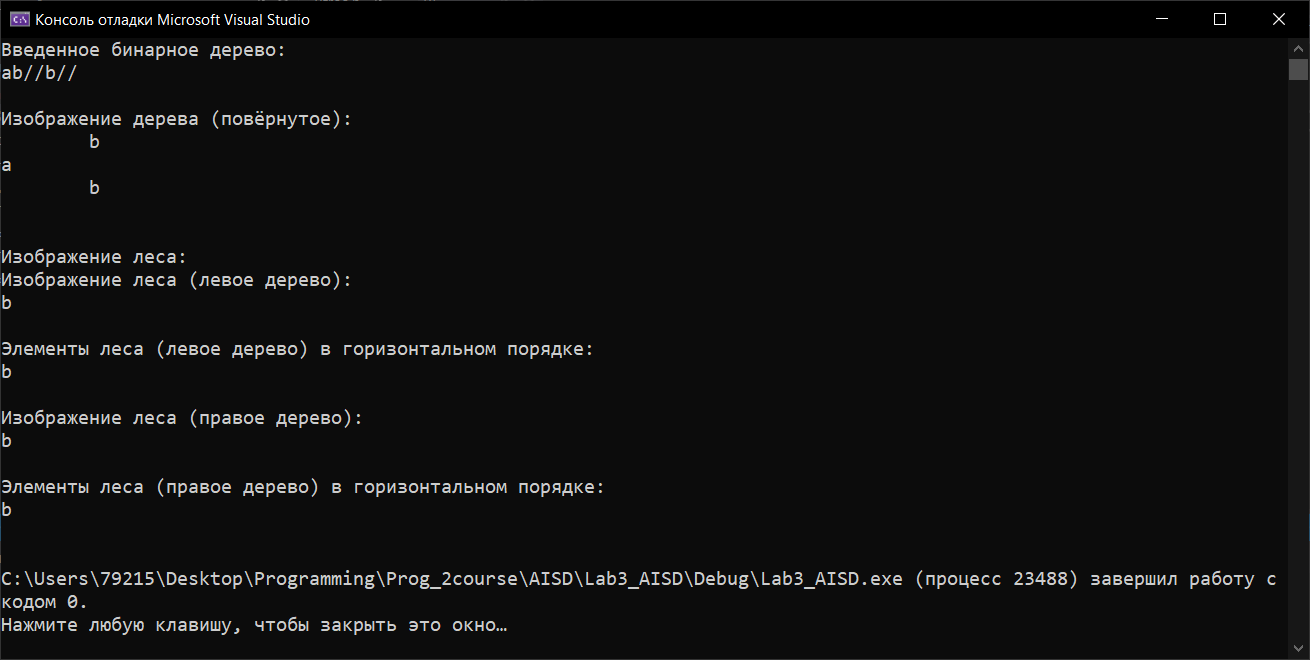


Рисунок 3 - Пример работы с входными данными № 3

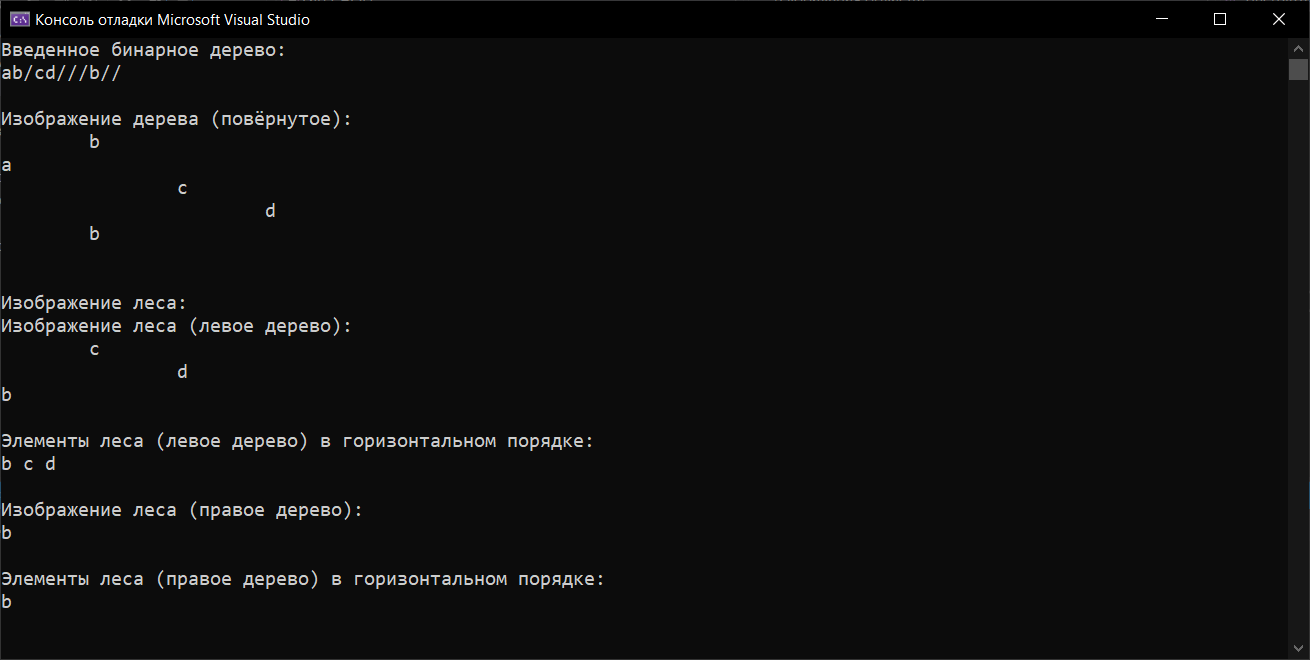


Рисунок 4 - Пример работы с входными данными № 4

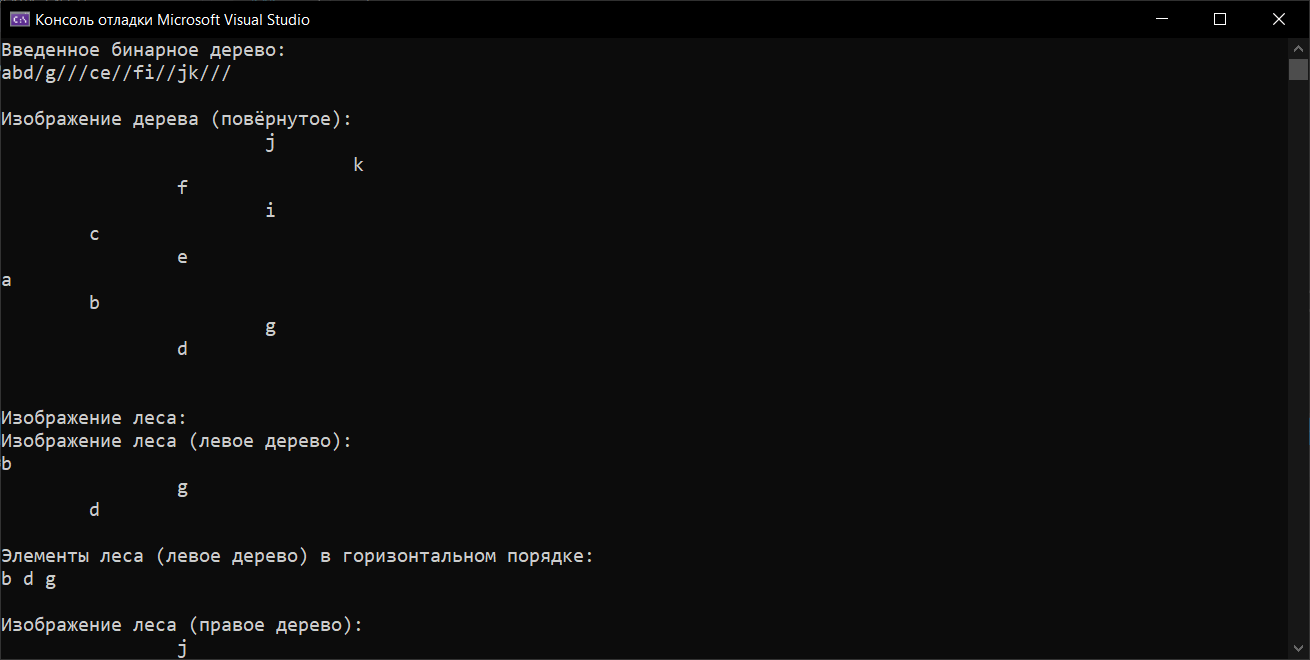


Рисунок 5 - Пример работы с входными данными № 5

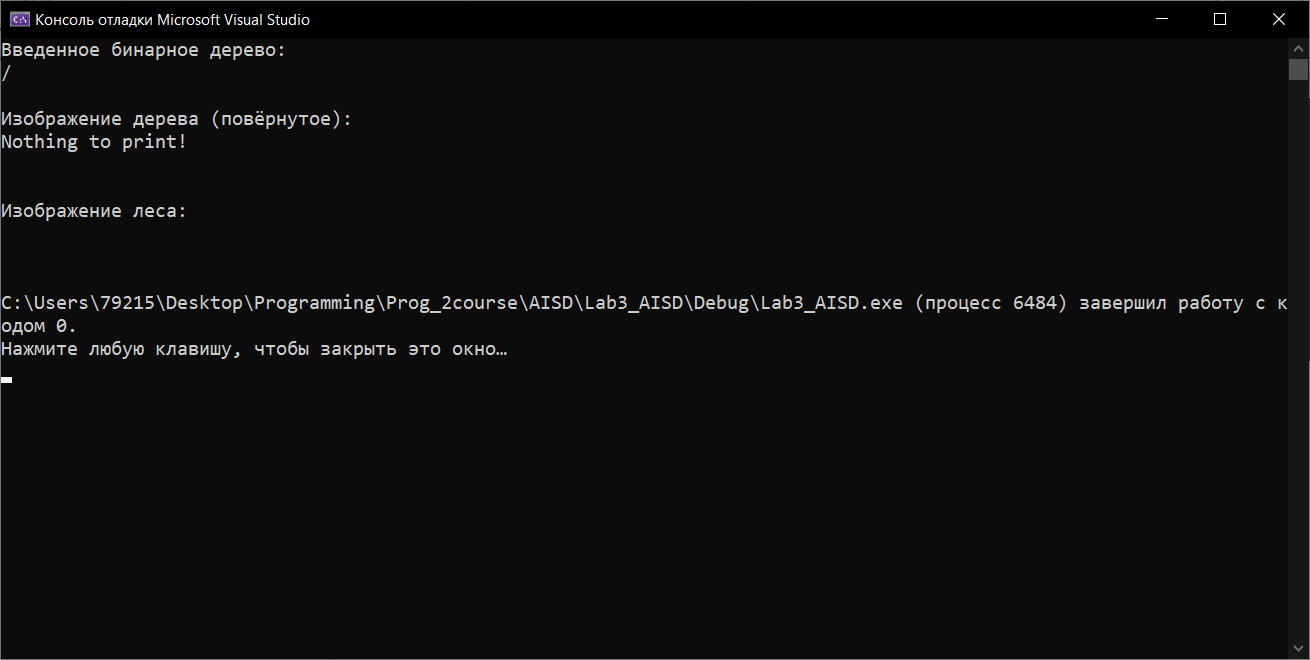


Рисунок 6 - Пример работы с входными данными № 3

**Графическое представление.**

abd/g///ce//fi//jk///

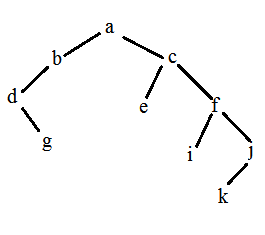


Рисунок 6 – Графическое представление бинарного дерева abd/g///ce//fi//jk///

Разработанный программный код см. в приложении А.

**Выводы.**

Были изучены бинарные деревья и лес, а так же реализована программа при помощи языка С++. Изучены основные понятия и приемы рекурсивного программирования. В ходе разработки программы были написаны основные функции для работы с бинарными деревьями, был получен лес из данного бинарного дерева, а также были перечислены элементы леса в горизонтальном порядке.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл tree.h

#ifndef TREE\_H

#define TREE\_H

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <string>

class Tree

{

public:

char getInfo()

{

return info;

}

Tree\* getLeft()

{

return lt;

}

Tree\* getRight()

{

return rt;

}

void setInfo(char data)

{

info = data;

}

void setLeft(Tree\* left)

{

lt = left;

}

void setRight(Tree\* right)

{

rt = right;

}

Tree()

{

info = '\0';

lt = nullptr;

rt = nullptr;

}

private:

char info;

Tree\* lt;

Tree\* rt;

};

typedef Tree\* BinTree; //представитель бинарного дерева

BinTree create(void);

bool isNull(BinTree);

char root(BinTree);

BinTree left(BinTree);

BinTree right(BinTree);

BinTree cons(const char& x, BinTree& lst, BinTree& rst);

void destroy(const BinTree&);

BinTree getForest(BinTree);

void printBinTree(BinTree);

BinTree readTree();

int level = 0; //уровень бинарного дерева

int level1 = 0; //уровень левого дерева

int level2 = 0; //уровень правого дерева

void printTree(BinTree, int level);

void que(BinTree);

void printForest(BinTree);

void out(BinTree, std::string);

void startOut(BinTree);

#endif

Файл tree.cpp

#include "Btree.h"

#include <queue>

//#include <Windows.h>

using namespace std;

ifstream infile("KLP.txt");

BinTree create() {

return nullptr;

}

bool isNull(BinTree b) {

return (b == nullptr);

}

char root(BinTree b) {

if (b == nullptr) {

cout << "Ошибка! Бинарное дерево пустое.\n";

exit(1);

}

else

return b->getInfo();

}

BinTree left(BinTree b) {

if (b == nullptr) {

cout << "Ошибка! Бинарное дерево пустое.\n";

exit(1);

}

else

return b->getLeft();

}

BinTree right(BinTree b) {

if (b == nullptr) {

cout << "Ошибка! Бинарное дерево пустое.\n";

exit(1);

}

else

return b->getRight();

}

BinTree cons(const char& x, BinTree& lst, BinTree& rst) { //порождает бинарное дерево из заданного узла и двух бинарных деревьев

BinTree p;

p = new Tree;

if (p != NULL) {

p->setInfo(x);

p->setLeft(lst);

p->setRight(rst);

return p;

}

else {

cout << "Нехватка памяти.\n";

exit(1);

}

}

void destroy(const BinTree& b) {

if (b != nullptr) {

destroy(b->getLeft());

destroy(b->getRight());

delete b;

}

}

void printForest(BinTree b) { // №1 получение леса

BinTree p = b->getLeft();

BinTree q = b->getRight();

cout << "Изображение леса (левое дерево):\n";

if (p) {

//cout << ' ' << root(p);

if (!isNull(p))

{

startOut(p);

}

else

cout << "\n";

}

cout << "\nЭлементы леса (левое дерево) в горизонтальном порядке:\n";

que(p);

cout << "\n\n";

cout << "Изображение леса (правое дерево):\n";

if (q) {

//cout << ' ' << root(q);

if (!isNull(q))

{

startOut(q);

}

else

cout << "\n";

}

cout << "\nЭлементы леса (правое дерево) в горизонтальном порядке:\n";

que(q);

}

BinTree readTree() {

char ch;

BinTree p, q;

infile >> ch;

if (ch == '/')

return NULL;

else {

p = readTree();

q = readTree();

return cons(ch, p, q);

}

}

void printBinTree(BinTree b) { //вывод введенного дерева

if (b != NULL) {

cout << root(b);

printBinTree(left(b));

printBinTree(right(b));

}

else

cout << '/';

}

void printTree(BinTree b, int level) { //изображение дерева

if (b) {

cout << ' ' << root(b);

if (!isNull(right(b)))

printTree(right(b), level + 1);

else cout << "\n";

if (!isNull(left(b))) {

for (int i = 1; i <= level; i++)

cout << " ";

printTree(left(b), level + 1);

}

}

}

void que(BinTree b) { //обход в ширину

queue <BinTree> Q;

BinTree p;

Q.push(b);

while (!Q.empty()) {

p = Q.front();

Q.pop();

cout << root(p) << ' ';

if (left(p) != NULL)

Q.push(left(p));

if (right(p) != NULL)

Q.push(right(p));

}

}

void startOut(BinTree b) {

if (!b) {

cout << "Nothing to print!\n";

return;

}

out(b, "");

}

void out(BinTree b, string str) {

if ((b->getLeft() == NULL) && (b->getRight() == NULL))

{

cout << str << b->getInfo() << endl;

}

else {

string str1 = str;

//cout << str << b->getInfo() << endl;

if (b->getRight() != NULL) {

str1 += " ";

out(b->getRight(), str1);

}

cout << str << b->getInfo() << endl;

if (b->getLeft() != NULL) {

str += " ";

out(b->getLeft(), str);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

BinTree b;

b = readTree();

cout << "Введенное бинарное дерево:\n";

printBinTree(b);

cout << "\n\n";

cout << "Изображение дерева (повёрнутое):\n"; //№2 вывести изображение бинарного дерева

startOut(b);

//startOut(b->getLeft());

//startOut(b->getRight());

//PrintTree(b, level);

cout << "\n\n";

cout << "Изображение леса:\n";

if (b)

printForest(b);

cout << "\n\n";

destroy(b);

return (0);

}