**Лабоpатоpная pабота №2**

**Бинарные деревья**

Написать функцию формирования бинарного дерева, состоящего из целых чисел. Для представления дерева использовать динамические структуры данных. Количество элементов дерева, а также его вид задаются случайным образом. Произвести вывод элементов дерева тремя видами обхода. Используя информацию о выведенном дереве изобразить структуру одного из построенных деревьев в отчете. Выполнить по вариантам следующие задания:

Вариант 12

12. Написать рекурсивную функцию, которая находит элемент дерева, встречающийся чаще всего.

Код:

# include <iostream>

# include <fstream>

# include <string>

# include <time.h>

using namespace std;

struct Node {

int data;

Node\* left, \* right;

};

void add(int x, Node\*& node) {

if (node == NULL) {

node = new Node;

node->data = x;

node->left = NULL;

node->right = NULL;

}

else {

if (x < node->data) {

add(x, node->left);

}

else {

add(x, node->right);

}

}

}

void createTree(Node\*& root, int& n) {

srand(time(NULL));

int capacity = rand() % 20;

for (int i = 0; i < capacity; ++i) {

int value = rand() % 10;

add(value, root);

if (value > n) n = value;

}

}

// Обход дерева прямым способом

void prePrint(Node\* node) {

if (node != NULL) {

cout << node->data << " ";

prePrint(node->left);

prePrint(node->right);

}

}

// Обход дерева симметричным способом

void inPrint(Node\* node) {

if (node != NULL) {

inPrint(node->left);

cout << node->data << " ";

inPrint(node->right);

}

}

// Обход дерева обратным способом

void postPrint(Node\* node) {

if (node != NULL) {

postPrint(node->left);

postPrint(node->right);

cout << node->data << " ";

}

}

void Search(Node\* node, int& max, int& count, int arr[]) {

if (node != NULL) {

arr[node->data]++;

if (arr[node->data] > count) {

count = arr[node->data];

max = node->data;

}

Search(node->left, max, count, arr);

Search(node->right, max, count, arr);

}

}

int main() {

srand(time(NULL));

int n = -1;

Node\* root = NULL;

createTree(root, n);

cout << n << endl;

int\* arr = new int[n+1];

int max = -1, count = 0;

for (int i = 0; i <= n; i++)

arr[i] = 0;

prePrint(root);

cout << endl;

inPrint(root);

cout << endl;

postPrint(root);

cout << endl;

Search(root, max, count, arr);

cout << max;

cout << endl;

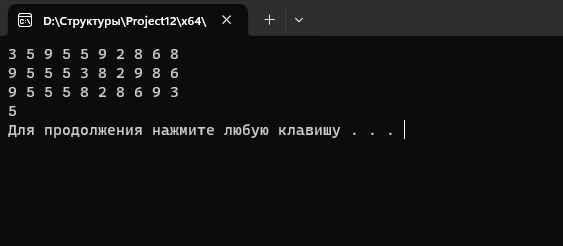
delete[] arr;

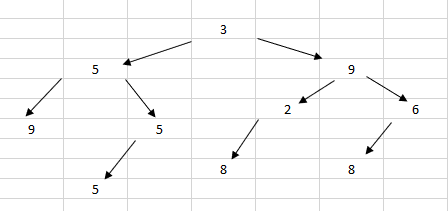
system("pause");

return 0;

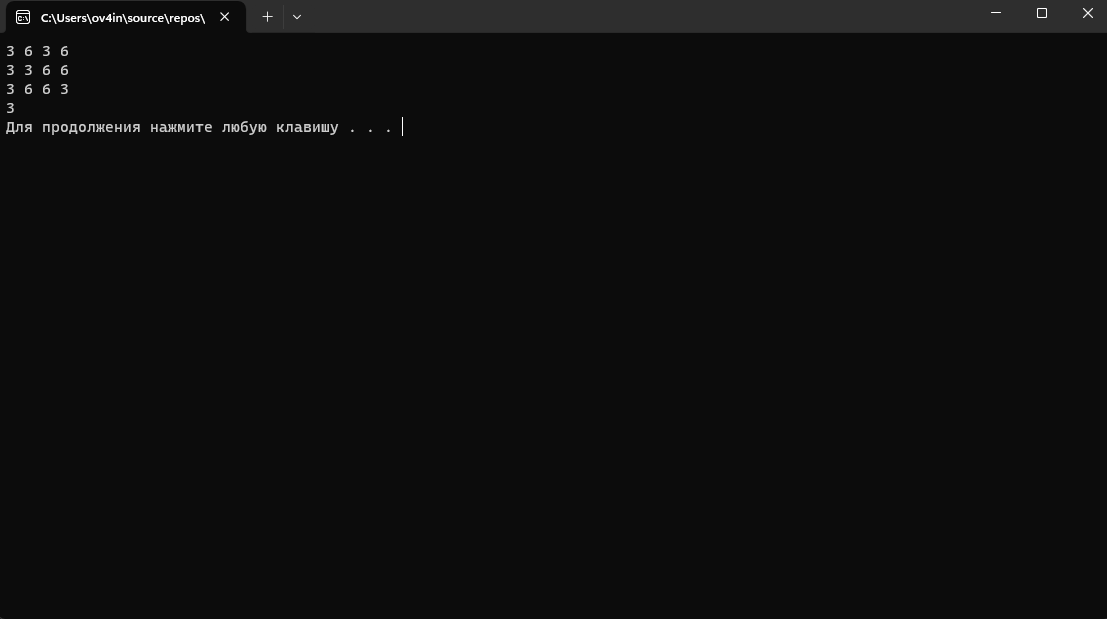
}Тесты:

1)

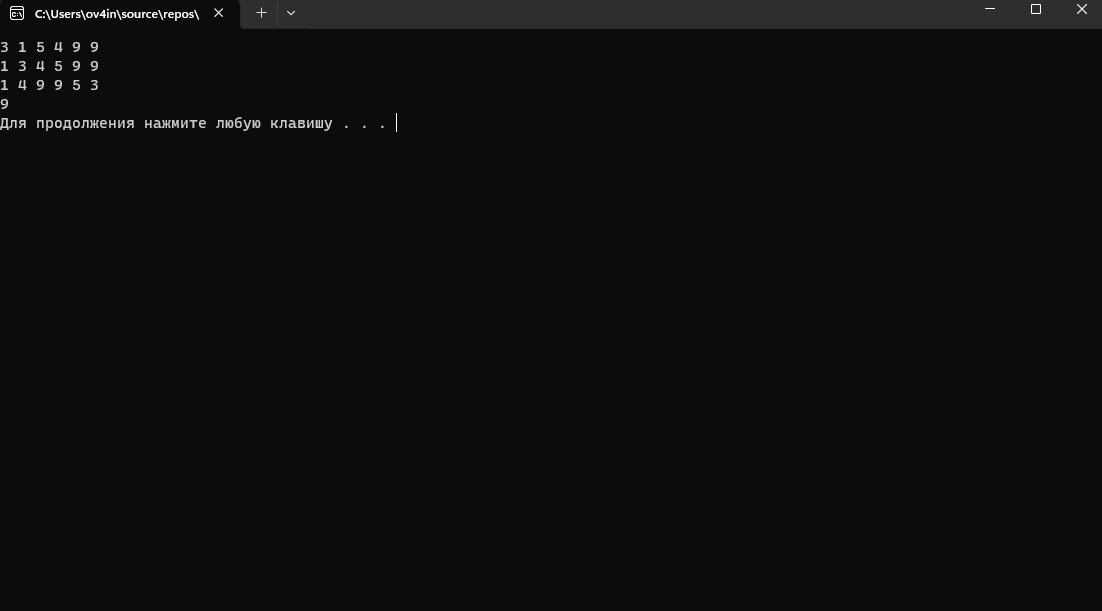




2)



3)



Алгоритм:

1. Определение структуры **Node** и функций для бинарного дерева.
2. Объявление массива **arr[n]**, который является массивом счётчиков для всех возможных элементов (размер массива равен максимально возможному значению бинарного дерева).
3. Объявление и инициализация переменных **max** и **count** типа **int**.
4. Заполнение массива **arr[n]** нулями.
5. Объявление корня бинарного дерева **root**.
6. Инициализация дерева с помощью функции **createTree.**
7. Вывод элементов дерева прямым обходом с помощью функции **prePrint.**
8. Вывод элементов дерева симметричным обходом с помощью функции **inPrint.**
9. Вывод элементов дерева обратным обходом с помощью функции **postPrint.**
10. Поиск элемент дерева, встречающийся чаще всего с помощью функции **Search.**
11. Вывод встречающегося чаще всего элемента **max**.
12. Завершение программы