Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №21**

Дисциплина: «информатика»

Тема: бинарные деревья

Вариант 4

Выполнил работу

студент группы РИС-20-1б

Еске Вячеслав Сергеевич

Проверила

Доцент кафедры ИТАС

Полякова О.А.

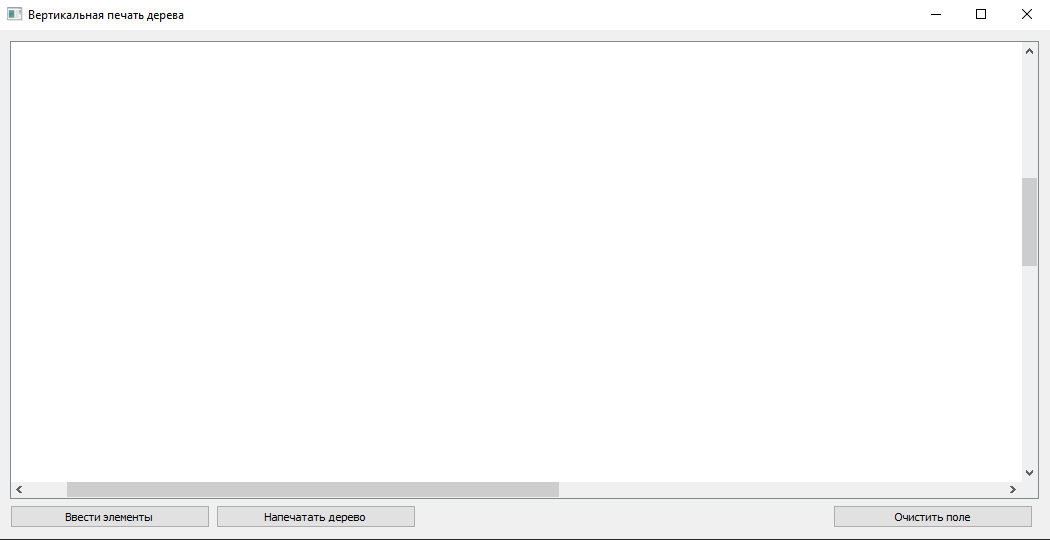
Пермь, 2021

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево.
2. Распечатать полученное дерево.
3. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
4. Преобразовать идеально сбалансированное дерево в дерево поиска.
5. Распечатать полученное дерево.
6. Тип информационного поля double. Найти минимальный элемент в дереве.

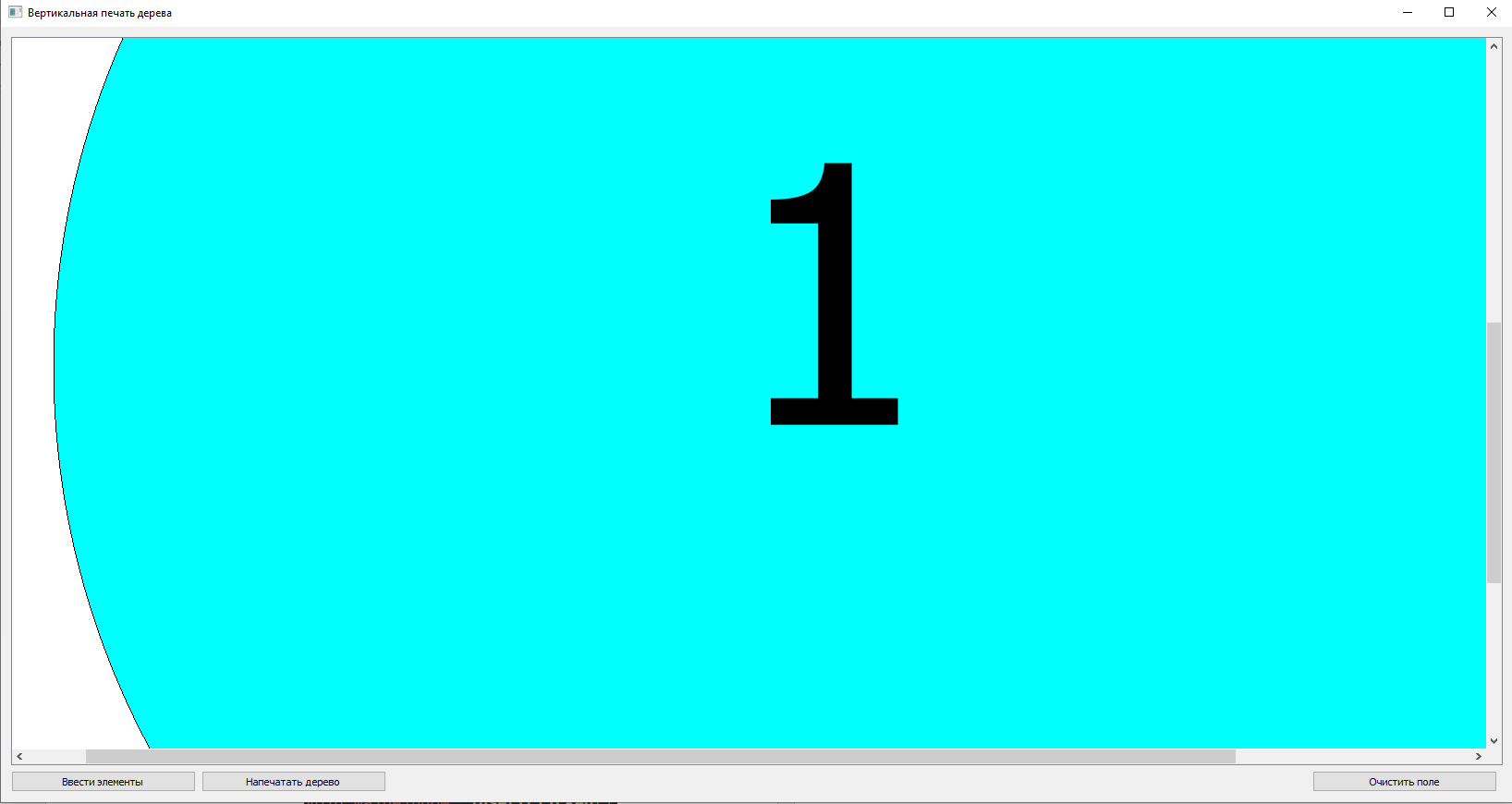
**Анализ задачи**

1. Для решения задачи необходимо…
   1. Проработать лаконичный интерфейс для пользователя;



Кнопка добавить элемент запускает программу, которая генерирует файл с заданными элементами;

При нажатии кнопки напечатать дерево, данные считываются из файла и строится дерево.



* 1. Создать класс MainWindow, в котором будут указаны все параметры, указывающие на окно программы;



* 1. Создать класс RenderArea, который будет отвечать за вывод бинарного дерева в консоль;



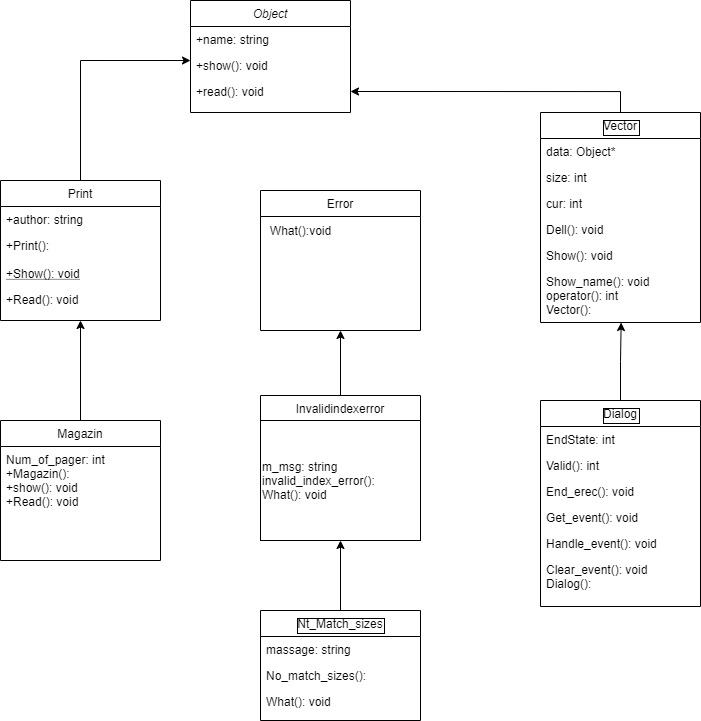
1. В программе были использованы следующие типы данных:
   1. Тип данных void для обработки нажатий в области окна;



* 1. Тип данных double для переменной, отвечающей за масштабирование;



**Диаграмма классов**



**Решение**

#include <iostream>

#include "BinaryTree.h"

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Введите количество элементов в бинарном дереве:" << endl;

do {

cout << ">";

cin >> array\_numbers\_size;

if (array\_numbers\_size < 1)

cout << "Введите положительное число" << endl;

} while (array\_numbers\_size < 1);

array\_numbers = new double[array\_numbers\_size];

BinaryTree tree;

cout << "Введите вещественные числа:\n";

tree.create\_balanced\_binary\_tree(array\_numbers\_size);

system("cls");

cout << "\t\t\t\t\t\t\t\t\t\t\tВЫВОД БИНАРНОГО ДЕРЕВА";

cout << "\n\t\t\t\t\t\t\t\t\t\t\tГлубина дерева: " << tree.depth\_tree();

tree.print\_tree();

cout << "\n\n\n\n";

system("pause");

system("cls");

BinaryTree search\_tree;

for (int i = 0; i < array\_numbers\_size; i++)

search\_tree.insert\_in\_search\_tree(array\_numbers[i]);

isBalancedSortPrinting = false;

cout << "\t\t\t\t\t\t\t\t\t\t\tВЫВОД БИНАРНОГО ДЕРЕВА ПОИСКА";

cout << "\n\t\t\t\t\t\t\t\t\t\t\tГлубина дерева: " << search\_tree.depth\_tree();

search\_tree.print\_tree();

cout << "\n\n\n\n";

system("pause");

cout << endl;

cout << "Минимальное значение элемента в дереве: " << search\_tree.find\_max() << endl;

system("pause");

system("cls");

search\_tree.print\_tree();

system("pause");

return 0;

}

#pragma once

#include <iostream>

#include <stdio.h>

#include <Windows.h>

using namespace std;

COORD position;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

double\* array\_numbers;

int array\_numbers\_size;

int index = 1;

bool isBalancedSortPrinting = true;

const int start\_x\_position = 100;

struct Node {

double data;

Node\* left, \* right;

};

class BinaryTree {

private:

Node\* root;

void delete\_tree(Node\* leaf) {

if (leaf != NULL) {

delete\_tree(leaf->left);

delete\_tree(leaf->right);

delete leaf;

}

}

bool isElementInTree(Node\* leaf, const double& element) {

if (leaf == NULL)

return false;

if (leaf->data == element)

return true;

if (element < leaf->data)

isElementInTree(leaf->left, element);

return isElementInTree(leaf->right, element);

}

void get\_subtrees\_sizes(const int size, int& left\_size, int& right\_size) {

if ((size - 1) % 2 == 0) {

left\_size = (size - 1) / 2;

right\_size = left\_size;

}

else {

left\_size = size / 2;

right\_size = (size - 1) / 2;

}

}

void create\_balanced\_binary\_tree(Node\*& leaf, int size) {

if (size == 0) {

leaf = NULL;

return;

}

cout << index + 1 << ") ";

double number;

cin >> number;

array\_numbers[index] = number;

index++;

leaf = new Node;

leaf->data = number;

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

create\_balanced\_binary\_tree(leaf->left, left\_subtree\_size);

create\_balanced\_binary\_tree(leaf->right, right\_subtree\_size);

}

int depth\_tree(Node\* leaf) {

if (leaf == NULL)

return 0;

return max(depth\_tree(leaf->right), depth\_tree(leaf->left)) + 1;

}

void print\_branch(Node\* leaf, int x, int y) {

if (leaf == NULL)

return;

position.X = x;

position.Y = y;

SetConsoleCursorPosition(hConsole, position);

cout << leaf->data;

int formula\_new\_x\_left\_leaf, formula\_new\_x\_right\_leaf;

if (isBalancedSortPrinting) {

formula\_new\_x\_left\_leaf = x - 2 \* (y + 2);

formula\_new\_x\_right\_leaf = x + 2 \* (y + 2);

}

else {

formula\_new\_x\_left\_leaf = x - 2 \* (y + 1);

formula\_new\_x\_right\_leaf = x + 2 \* y;

}

print\_branch(leaf->left, formula\_new\_x\_left\_leaf, y + 2);

print\_branch(leaf->right, formula\_new\_x\_right\_leaf, y + 2);

}

void print\_tree(Node\*& leaf) {

if (leaf == NULL)

return;

Node\* branch\_left = NULL;

Node\* branch\_right = NULL;

branch\_left = leaf->left;

branch\_right = leaf->right;

position.X = start\_x\_position;

position.Y = 3;

SetConsoleCursorPosition(hConsole, position);

cout << leaf->data;

int depthOfTheTree = depth\_tree(leaf);

int offset\_coefficient;

if (isBalancedSortPrinting)

offset\_coefficient = 6;

else

offset\_coefficient = 4;

print\_branch(branch\_left, start\_x\_position - offset\_coefficient \* depthOfTheTree, 5);

print\_branch(branch\_right, start\_x\_position + offset\_coefficient \* depthOfTheTree, 5);

}

Node\* insert\_in\_search\_tree(Node\*& leaf, const double element) {

if (isElementInTree(leaf, element)) //если элемент дублируется, то в дерево поиска мы его не вставляем

return leaf;

if (leaf == NULL) {

Node\* new\_leaf = new Node;

new\_leaf->data = element;

new\_leaf->left = NULL;

new\_leaf->right = NULL;

return new\_leaf;

}

if (element < leaf->data)

leaf->left = insert\_in\_search\_tree(leaf->left, element);

if (element > leaf->data)

leaf->right = insert\_in\_search\_tree(leaf->right, element);

return leaf;

}

double find\_max(Node\* leaf) {

system("cls");

cout << "\t\t\t\t\t\t\t\t\t\tПРОЦЕСС ПОИСКА МИНИМАЛЬНОГО ЭЛЕМЕНТА В ДЕРЕВЕ";

print\_tree(leaf);

cout << endl;

getchar();

system("cls");

if (leaf->left == NULL)

return leaf->data;

return find\_max(leaf->left);

}

public:

BinaryTree() {

root = NULL;

}

~BinaryTree() {

delete\_tree(root);

}

void create\_balanced\_binary\_tree(int size) {

double number;

cout << "1) ";

cin >> number;

root = new Node;

root->data = number;

array\_numbers[0] = number;

if ((size - 1) > 0) {

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

create\_balanced\_binary\_tree(root->left, left\_subtree\_size);

create\_balanced\_binary\_tree(root->right, right\_subtree\_size);

}

}

int depth\_tree() {

return depth\_tree(root);

}

void print\_tree() {

print\_tree(root);

}

void insert\_in\_search\_tree(const double element) {

root = insert\_in\_search\_tree(root, element);

}

double find\_max() {

return find\_max(root);

}

};

**Скриншоты**

