

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема. Нелинейные структуры данных. Бинарное дерево.

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Основин А.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc149353504)

[2 Тестовый пример 4](#_Toc149353505)

[3 Результаты тестирования 5](#_Toc149353506)

[3.1 Определение среднего арифметического всех узлов дерева, используя алгоритм обхода в «ширину» 5](#_Toc149353507)

[3.2 Операция определения количества узлов в дереве 5](#_Toc149353508)

[3.3 Операция удаления крайнего левого листа из дерева 6](#_Toc149353509)

[3.4 Операция определения уровня узла с заданным значением 7](#_Toc149353510)

[4 Прототипы функций 9](#_Toc149353511)

[5 Код основной программы 13](#_Toc149353512)

[6 ВЫВОД 16](#_Toc149353513)

# Постановка задачи

Индивидуальный вариант №17.

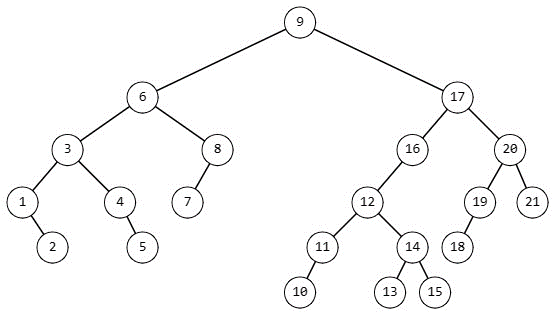
Вид дерева: бинарное дерево поиска (БДП).

Реализовать операции общие для вариантов с 16 по 20:

* Создать бинарное дерево поиска, информационная часть узла – целое число. Для создания БДП необходимо реализовать операцию вставки нового значения в БДП и использовать ее при создании дерева;
* Отобразить дерево на экране, повернув его справа налево;
* Определить среднее арифметическое всех узлов дерева, используя алгоритм обхода в «ширину»;
* Определить количество узлов в дереве;
* Удалить самый левый лист дерева;
* Определить уровень, на котором находится заданное значение.

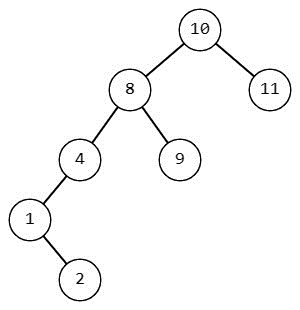
# Тестовый пример

При создании дерева путём добавления узлов в следующем порядке: 9, 17, 6, 8, 3, 20, 16, 19, 18, 21, 12, 11, 10, 14, 7, 4, 1, 2, 13, 15, 5, должно получиться дерево, которое изображено на Рисунке 1.



**Рисунок 1 – Бинарное дерево поиска**

На Рисунке 2 представлено бинарное дерево, построенное по последовательно добавленным в него узлам: 10, 8, 4, 1, 9, 2, 11.

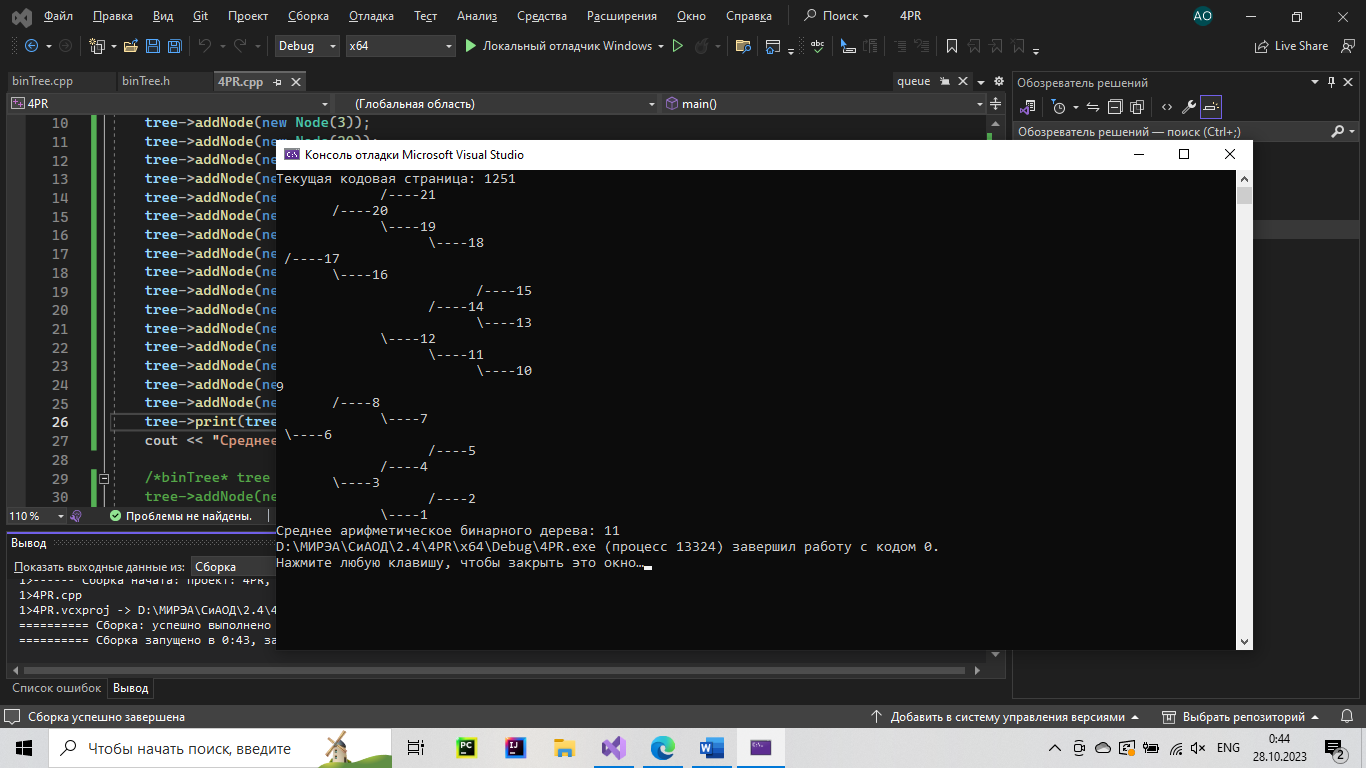


**Рисунок 2 – Бинарное дерево поиска**

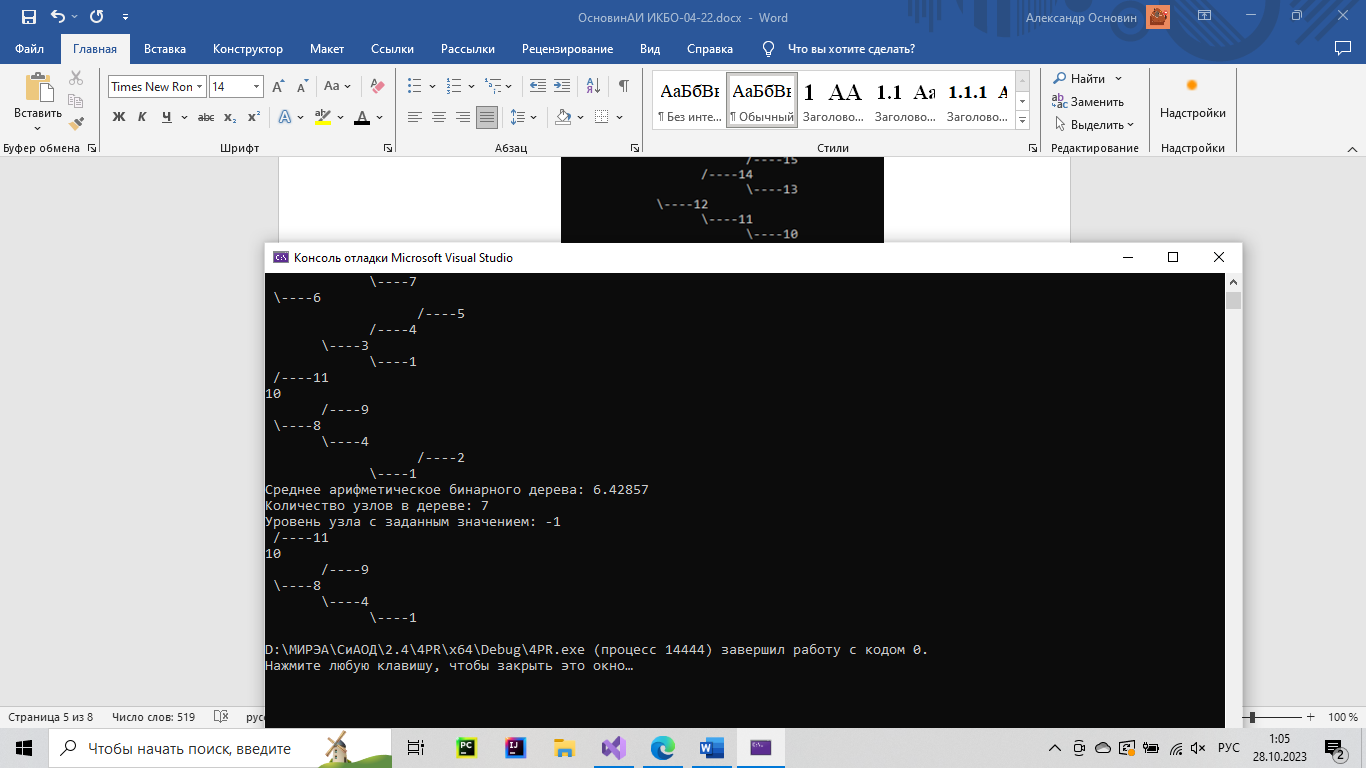
# Результаты тестирования

## Определение среднего арифметического всех узлов дерева, используя алгоритм обхода в «ширину»

На Рисунках 3 – 4 представлены результаты тестирования функции определения среднего арифметического всех узлов дерева для первого и второго тестового примера соответственно.



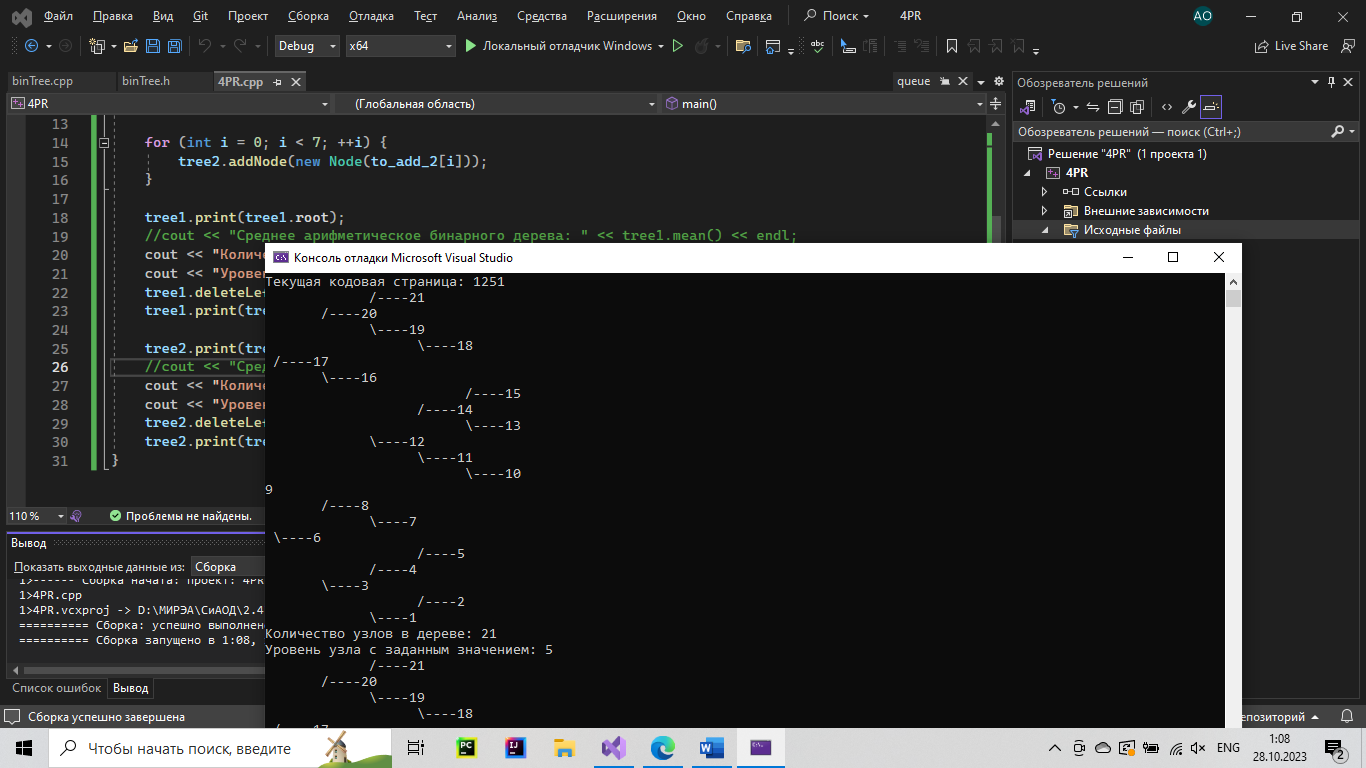
**Рисунок 3 – Тестирование функции определения среднего арифметического всех узлов дерева**



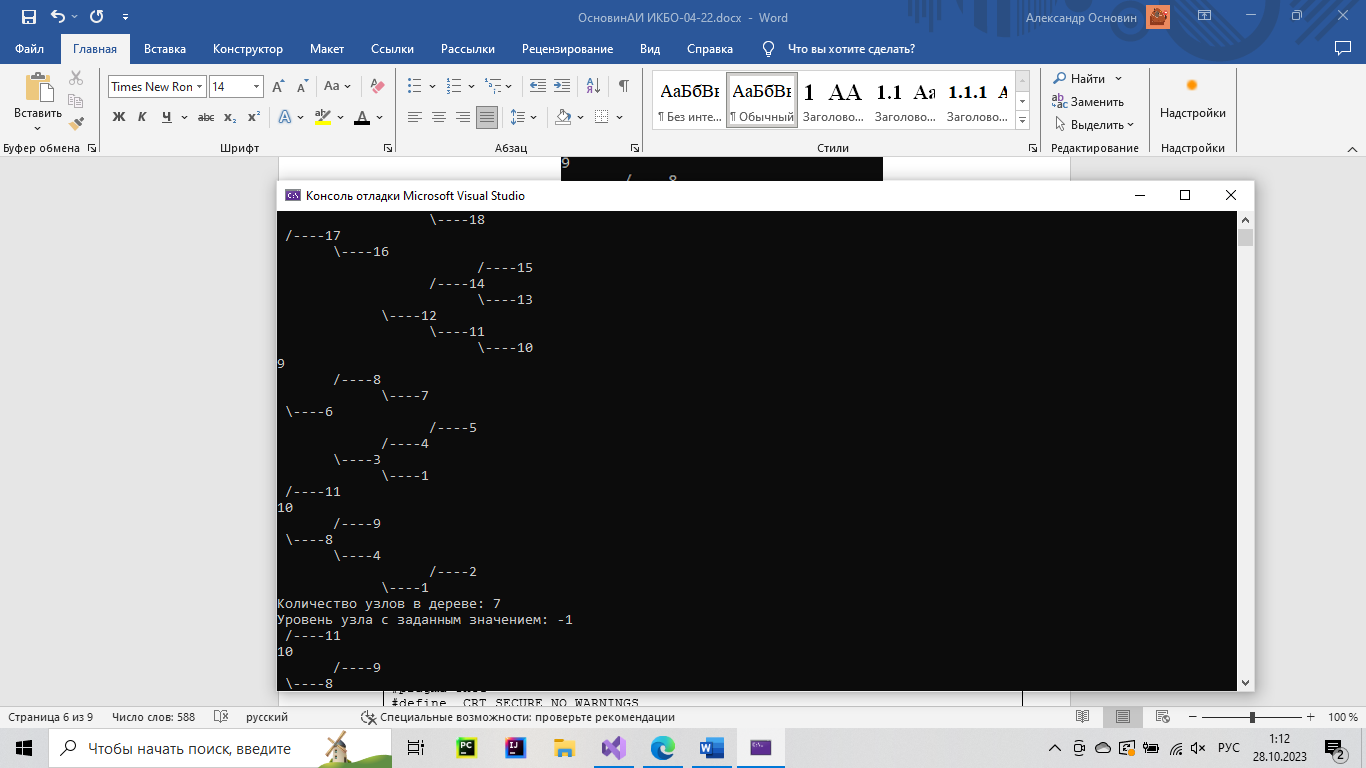
**Рисунок 4 – Тестирование функции определения среднего арифметического всех узлов дерева**

## Операция определения количества узлов в дереве

На Рисунках 5 – 6 представлены результаты тестирования функции определения количества узлов в дереве для первого и второго тестового примера соответственно.



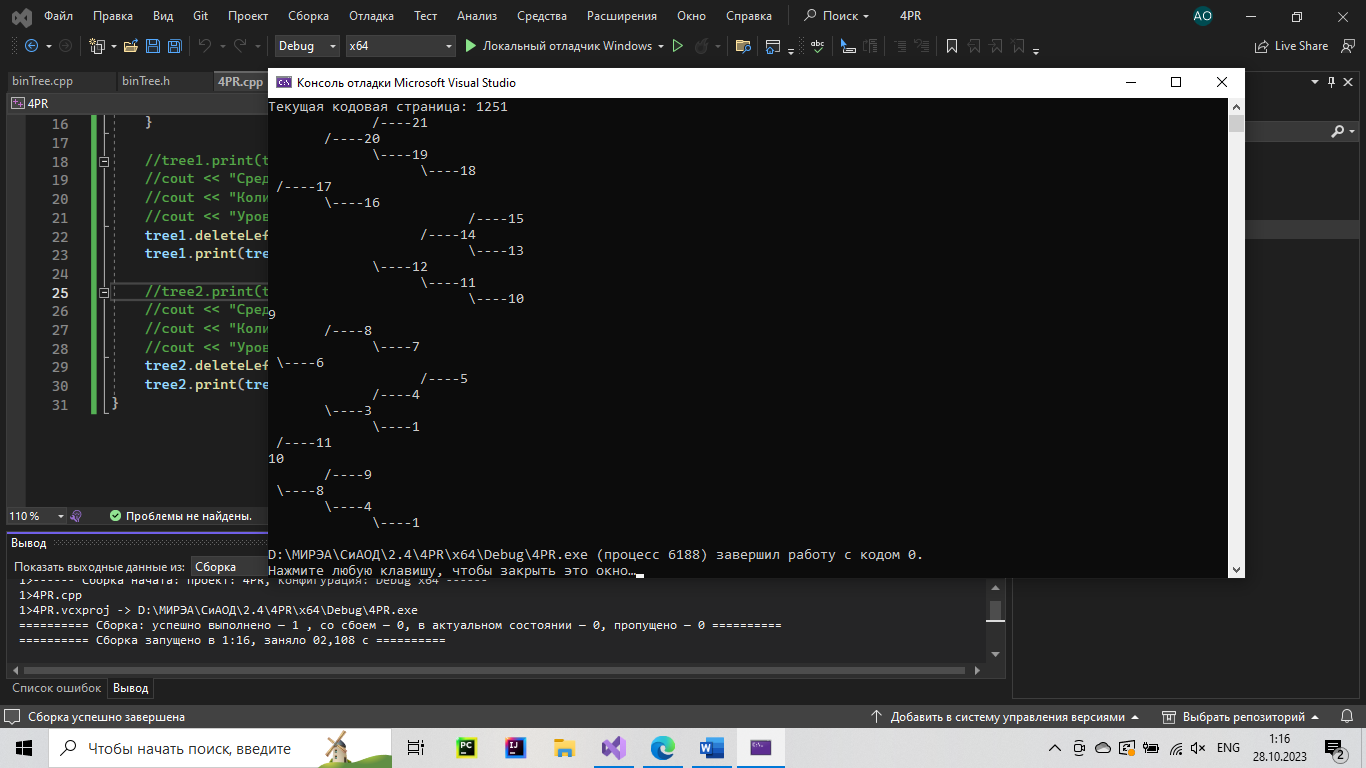
**Рисунок 5 – Тестирование функции определения количества узлов в дереве**



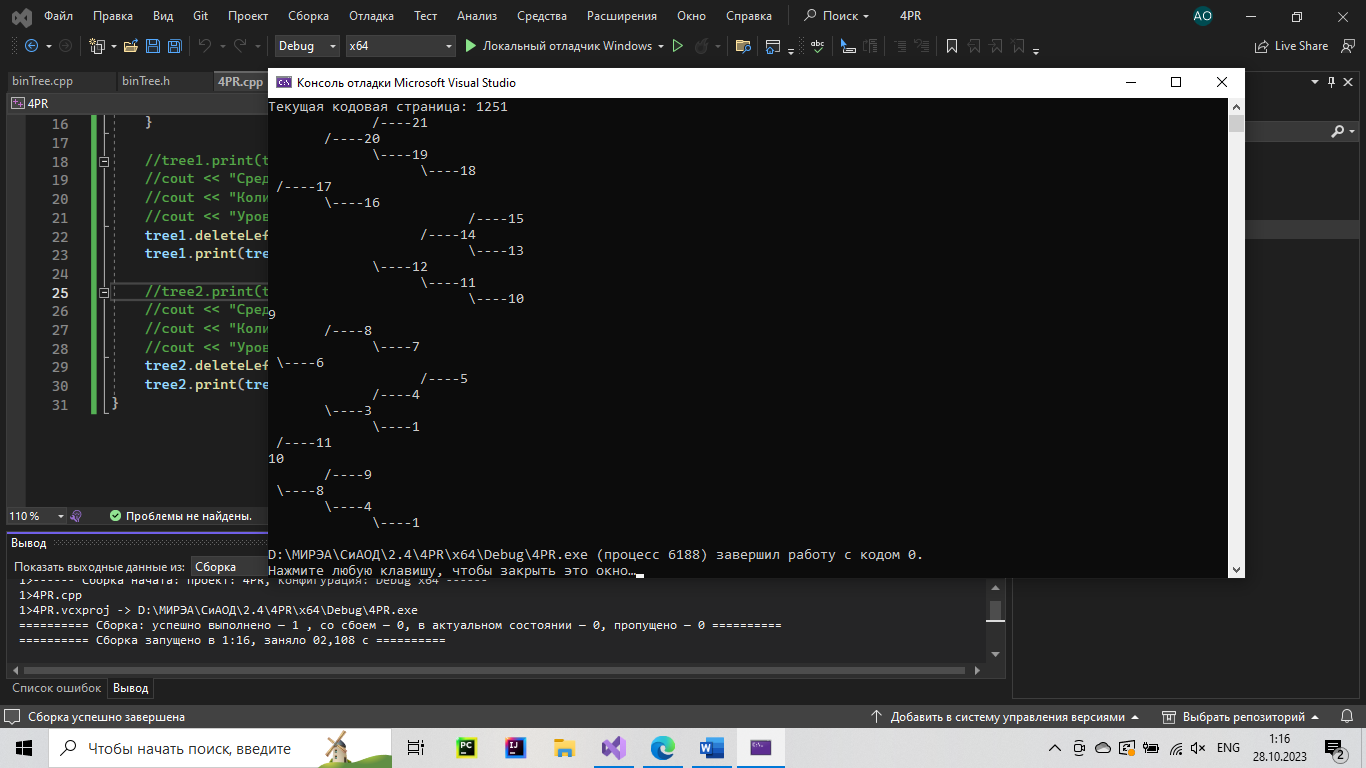
**Рисунок 6 – Тестирование функции определения количества узлов в дереве**

## Операция удаления крайнего левого листа из дерева

На Рисунках 7 – 8 представлены результаты тестирования функции удаления крайнего левого листа из дерева для первого и второго тестового примера соответственно.



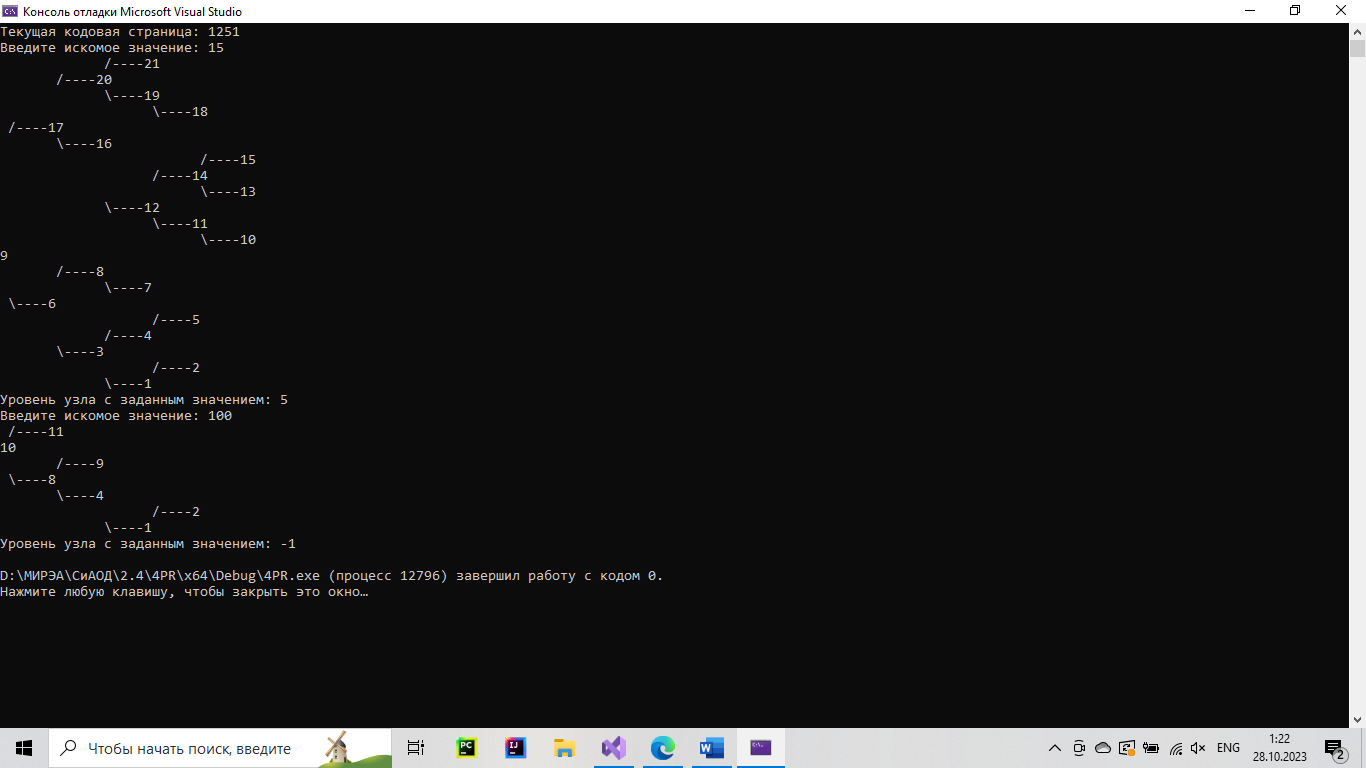
**Рисунок 7 – Тестирование функции удаления крайнего левого листа из дерева**



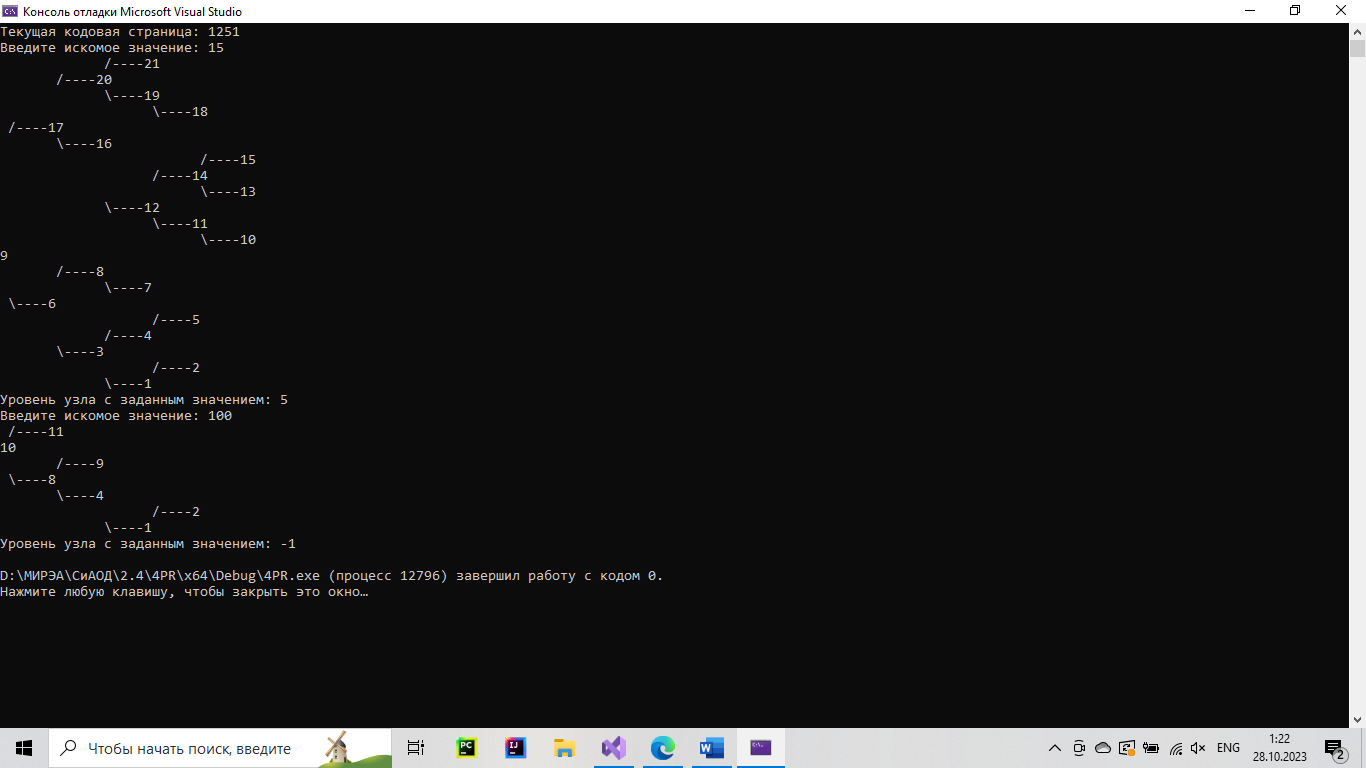
**Рисунок 8 – Тестирование функции удаления крайнего левого листа из дерева**

## Операция определения уровня узла с заданным значением

На Рисунках 9 – 10 представлены результаты тестирования функции определения уровня узла с заданным значением для первого и второго тестового примера соответственно (результат второго тестирования – «-1» - означает, что в БДП отсутствует узел с заданным значением).



**Рисунок 9 – Тестирование функции определения уровня узла с заданным значением**



**Рисунок 10 – Тестирование функции определения уровня узла с заданным значением**

# Прототипы функций

В Листинге 1 представлен заголовочный файл с прототипами методов, реализующих операции варианта, а также структурой узла – элемента, который является единицей бинарного дерева.

*Листинг 1 – binTree.h*

#pragma once

#include <iostream>

#include <cmath>

#include <queue>

using namespace std;

struct Node {

int value;

Node\* left;

Node\* right;

Node(int \_value) : value(\_value), left(nullptr), right(nullptr) {};

};

struct binTree {

Node\* root;

binTree() : root(nullptr) {};

binTree(Node\* \_root) : root(\_root) {};

bool addNode(Node\* new\_node);

static void print(Node\* current, char from = ' ', size\_t level = 0);

double mean();

static size\_t count(Node\* current);

void deleteLeft();

int getLevel(Node\* current, int value, int level = -1);

};

1. Операция вставки нового значения в БДП.

* addNode(Node\* new\_node) – реализует операцию вставки узла new\_node в БДП, если узла с таким же значением не существует; для этого проходит по узлам бинарного дерева до первого свободного места, сравнивая значение new\_node со значением текущего узла, переходит влево если значение new\_node меньше, чем значение текущего узла, иначе – вправо.
* Предусловие: указатель на узел Node new\_node – узел, содержащий значений, которое необходимо вставить в БДП.
* Постусловие: в БДП при условии соблюдения уникальности значений узлов вставлен узел new\_node; возвращаемое значение – булева переменная, истина, если узел добавлен, ложь, если нарушено условие уникальности.

1. Операция вывода БДП, перевернутого на левый бок, в консоль.

* print(Node\* current, char from = ' ', size\_t level = 0) – реализует операцию вывода БДП в консоль. Обходит дерево в глубину симметрично и выводит узлы, форматируя значение выводимых строк для наглядности.
* Предусловие: указатель на узел Node current – узел, с которого будет выведено дерево (можно вывести поддерево), необходим для рекурсивного вызова функции; символ from – используется при форматировании строки, указывает при выводе в консоль на левый/правый подузел текущего узла, беззнаковое целое level – показывает глубину текущего узла в БДП, необходимо для форматирования.
* Постусловие: в консоль выведено БДП; возвращаемое значение отсутствует.

1. Операция определения среднего арифметического всех узлов дерева с использованием алгоритма обхода в «ширину».

* mean() – реализует операцию определения среднего арифметического всех узлов дерева с использованием алгоритма обхода в «ширину». Сначала добавляет в очередь корневой узел БДП, затем поочередно достаёт первый узел из очереди, удаляя его из очереди, и добавляет в очередь правый и левый зависимые узлы текущего. Для текущего узла увеличивается значение счётчика и суммы значений узлов.
* Предусловие: функция вызывается из основной функции программы, передаваемые параметры отсутствуют.
* Постусловие: возвращаемое значение – среднее арифметическое значений узлов БДП.

1. Определение количества узлов в дереве.

* count(Node\* current) – реализует операцию определения количества узлов в дереве (или поддереве), корень которого – current. Возвращает 0, если текущий параметр – пустой указатель, иначе рекурсивно вызывает себя для правого поддерева текущего узла и для левого поддерева текущего узла, складывая эти значения и прибавляя к ним единицу.
* Предусловие: указатель на узел Node current – корень дерева, для которого необходимо подсчитать количество узлов.
* Постусловие: возвращаемое значение – беззнаковое целое, количество узлов в дереве.

1. Удаление крайнего левого листа дерева

* deleteLeft() – реализует операцию удаления крайнего левого листа из дерева. Проходит по БДП, всегда выбирая левый узел в качестве следующей точки маршрута при условии существования левого узла, иначе выбирает правый узел; останавливается на элементе, который является родителем самого левого листа БДП, стирает для него указатель на дочерний узел, который является крайним левым листом дерева, и освобождает память, выделенную под самый левый лист дерева.
* Предусловие: функция вызывается из основной функции программы, передаваемые параметры отсутствуют.
* Постусловие: из БДП удалён крайний левый лист, возвращаемое значение отсутствует.

1. Определение уровня, на котором находится заданное значение.

* getLevel(Node\* current, int value, int level = -1) – реализует операцию определения уровня, на котором находится заданное значение. Рекурсивно вызывает себя для левого и правого поддерева.
* Предусловие: указатель на узел Node current – корень дерева, в котором необходимо узнать уровень узла; целочисленное value – заданное значение искомого узла; целочисленное level – текущий уровень в БДП, необходимо для рекурсивного вызова функции.
* Постусловие: возвращаемое значение – целое число, номер уровня, на котором находится узел с заданным значением, начиная с 0, если такой узел существует, иначе – «-1».

# Код основной программы

В Листинге 2 представлена реализация методов класса бинарного дерева.

*Листинг 2 – binTree.cpp*

#include "binTree.h"

bool binTree::addNode(Node\* new\_node) {

if (root == nullptr) {

root = new\_node;

return true;

}

bool status = true;

Node\* current = root;

while (status && current->left != new\_node && current->right != new\_node) {

if (new\_node->value < current->value) {

if (current->left != nullptr) {

current = current->left;

}

else {

current->left = new\_node;

}

}

else if (new\_node->value > current->value) {

if (current->right != nullptr) {

current = current->right;

}

else {

current->right = new\_node;

}

}

else {

status = false;

}

}

return status;

}

void binTree::print(Node\* current, char from, size\_t level) {

if (current != nullptr) {

print(current->right, '/', level + 1);

cout << (level > 0 ? (level > 1 ? string((level - 1) \* 6, ' ') : "") + " " + from + "----" : "");

cout << current->value << endl;

print(current->left, '\\', level + 1);

}

}

double binTree::mean() {

size\_t count = 0, sum = 0;

queue <Node\*> tree;

tree.push(root);

while (!tree.empty()) {

Node\* current = tree.front();

tree.pop();

if (current != nullptr) {

tree.push(current->left);

tree.push(current->right);

count++;

sum += current->value;

}

}

return (double)sum / ((count != 0) ? count : 1);

}

size\_t binTree::count(Node\* current) {

if (current == nullptr) {

return 0;

}

return 1 + count(current->left) + count(current->right);

}

void binTree::deleteLeft() {

if (root == nullptr) {

return;

}

Node\* current = root, \* previous = root;

while (current->left != nullptr || current->right != nullptr) {

previous = current;

if (current->left != nullptr) {

current = current->left;

}

else {

current = current->right;

}

}

if (previous->left != nullptr) {

previous->left = nullptr;

}

else {

previous->right = nullptr;

}

delete current;

}

int binTree::getLevel(Node\* current, int value, int level) {

if (current == nullptr) {

return -1;

}

if (current->value == value) {

return ++level;

}

int left = getLevel(current->left, value, level + 1);

if (left != -1) {

return left;

}

return getLevel(current->right, value, level + 1);

}

В Листинге 3 представлена главная функция программы – main().

*Листинг 3 – Функция main()*

#include "binTree.h"

int main() {

system("chcp 1251");

int to\_add\_1[21] = {9, 17, 6, 8, 3, 20, 16, 19, 18, 21, 12, 11, 10, 14, 7, 4, 1, 2, 13, 15, 5};

int to\_add\_2[7] = { 10, 8, 4, 1, 9, 2, 11 };

binTree tree1 = binTree();

binTree tree2 = binTree();

for (int i = 0; i < 21; ++i) {

tree1.addNode(new Node(to\_add\_1[i]));

}

for (int i = 0; i < 7; ++i) {

tree2.addNode(new Node(to\_add\_2[i]));

}

cout << "Введите искомое значение: ";

int a;

cin >> a;

tree1.print(tree1.root);

//cout << "Среднее арифметическое бинарного дерева: " << tree1.mean() << endl;

//cout << "Количество узлов в дереве: " << tree1.count(tree1.root) << endl;

cout << "Уровень узла с заданным значением: " << tree1.getLevel(tree1.root, a) << endl;

//tree1.deleteLeft();

//tree1.print(tree1.root);

cout << "Введите искомое значение: ";

int b; // 15

cin >> b; // 100

tree2.print(tree2.root);

//cout << "Среднее арифметическое бинарного дерева: " << tree2.mean() << endl;

//cout << "Количество узлов в дереве: " << tree2.count(tree2.root) << endl;

cout << "Уровень узла с заданным значением: " << tree2.getLevel(tree2.root, b) << endl;

//tree2.deleteLeft();

//tree2.print(tree2.root);

}

# ВЫВОД

В ходе выполнения данной практической работы были получены навыки разработки операций над структурой данных бинарное дерево. Также были изучены алгоритмы обхода дерева в ширину и в глубину: прямым, симметричным и обратным способом.