МИНИСТЕРСТВО ОБРАЗОВАНИЯ ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ДОНЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Отчет по лабораторной работе №9

Курс «Основы программирования»

Вариант №2

Выполнил: студент Бережецкий К.Т.

ИВТ-2, 1 курс

Донецк, 2024 г.

**СОДЕРЖАНИЕ**

[КОНТРОЛЬНЫЕ ВОПРОСЫ 3](#_Toc197640513)

[ПРАКТИЧЕСКИЕ ЗАДАНИЯ 7](#_Toc197640514)

[Задание №1 7](#_Toc197640515)

[Задание №2 9](#_Toc197640516)

[Задание №3 10](#_Toc197640517)

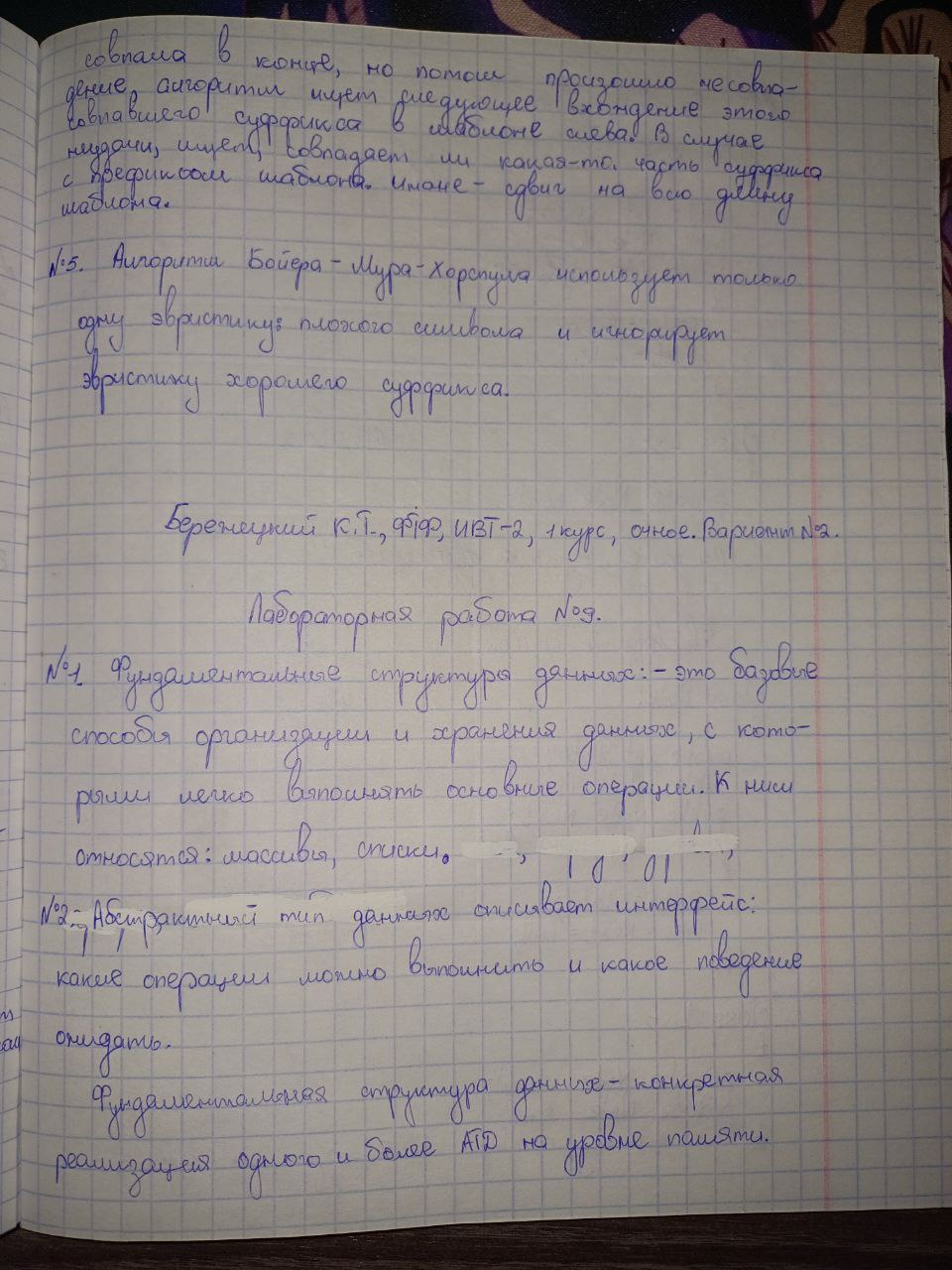
[Задание №4 11](#_Toc197640518)

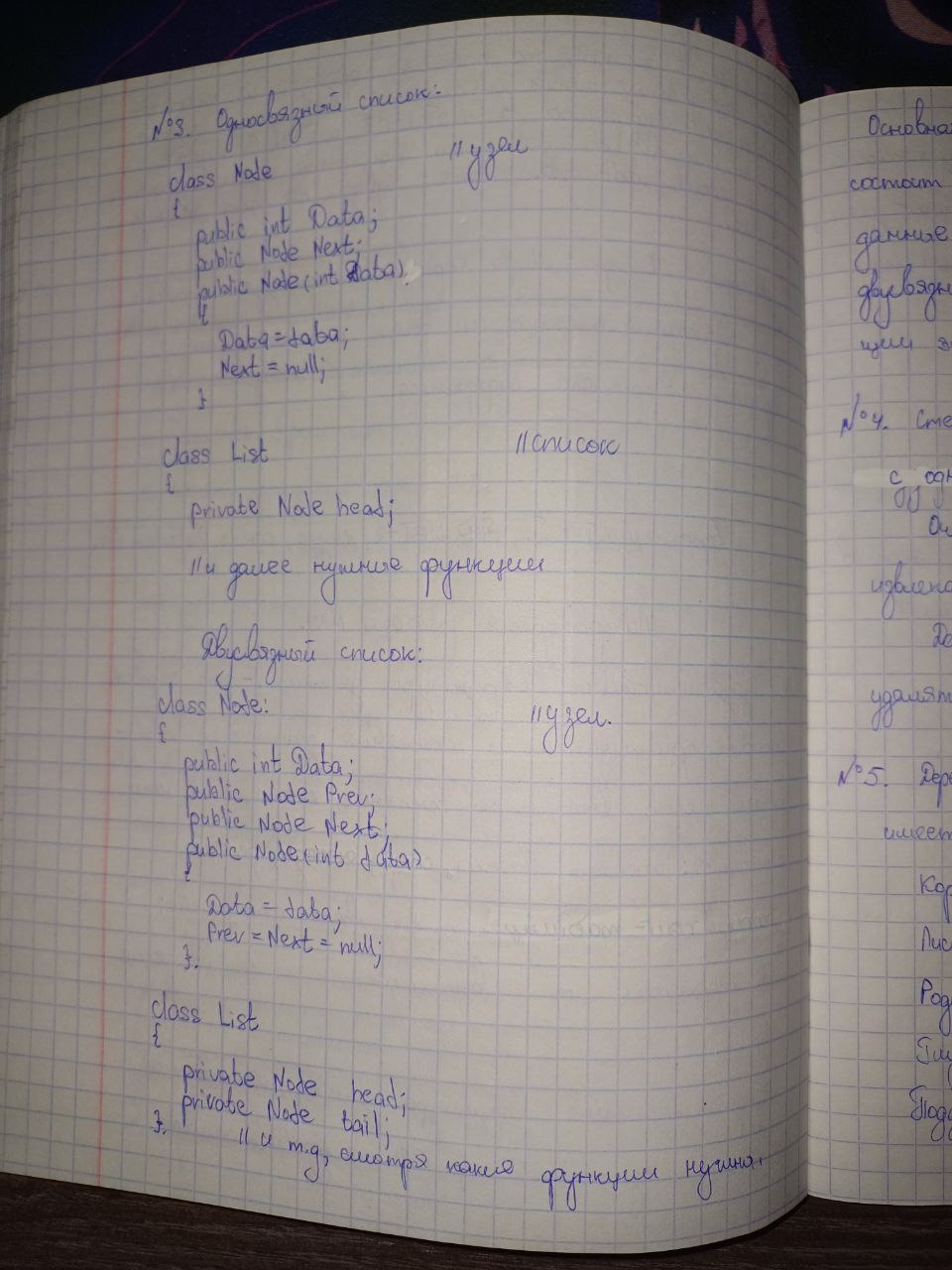
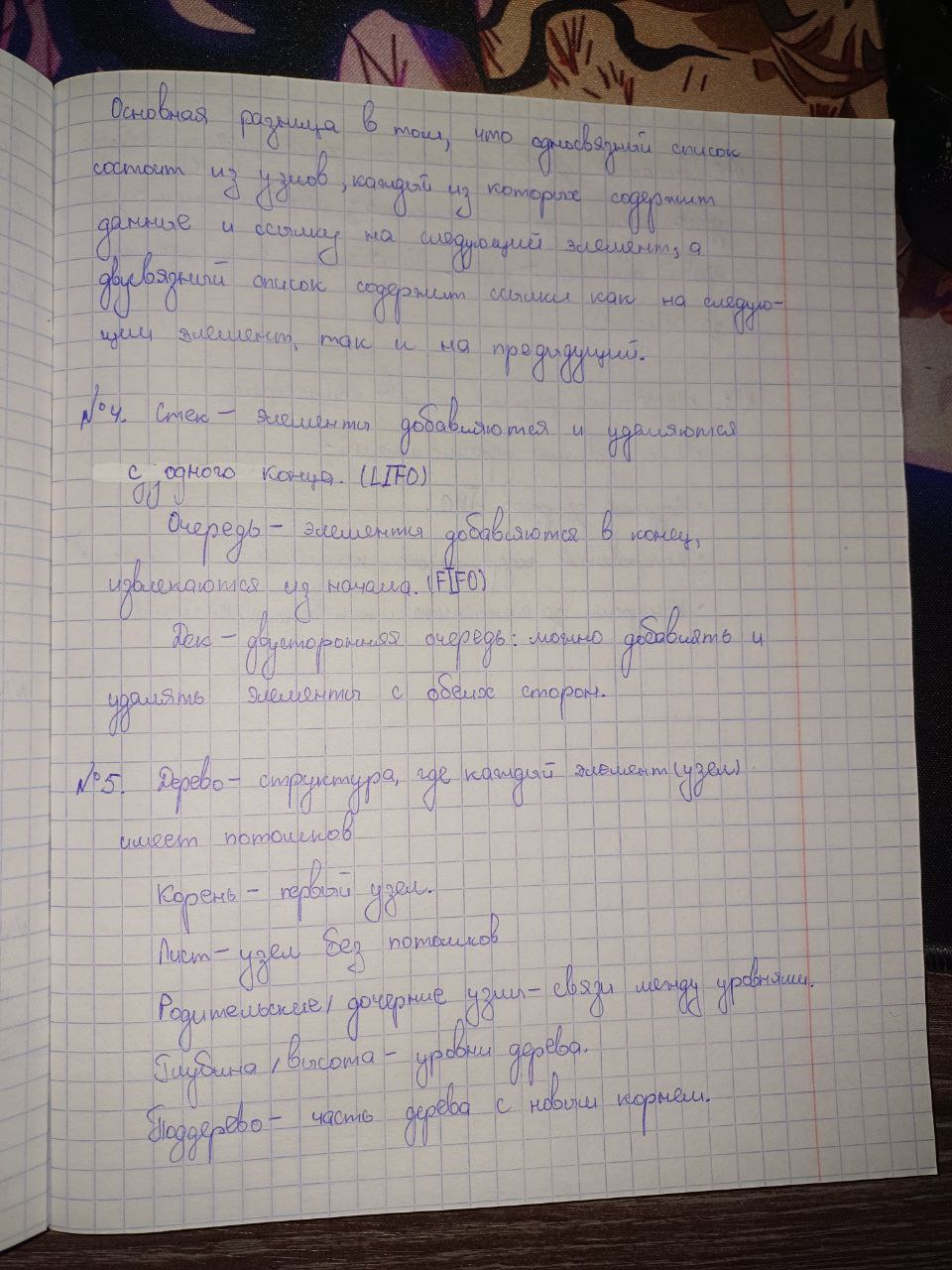
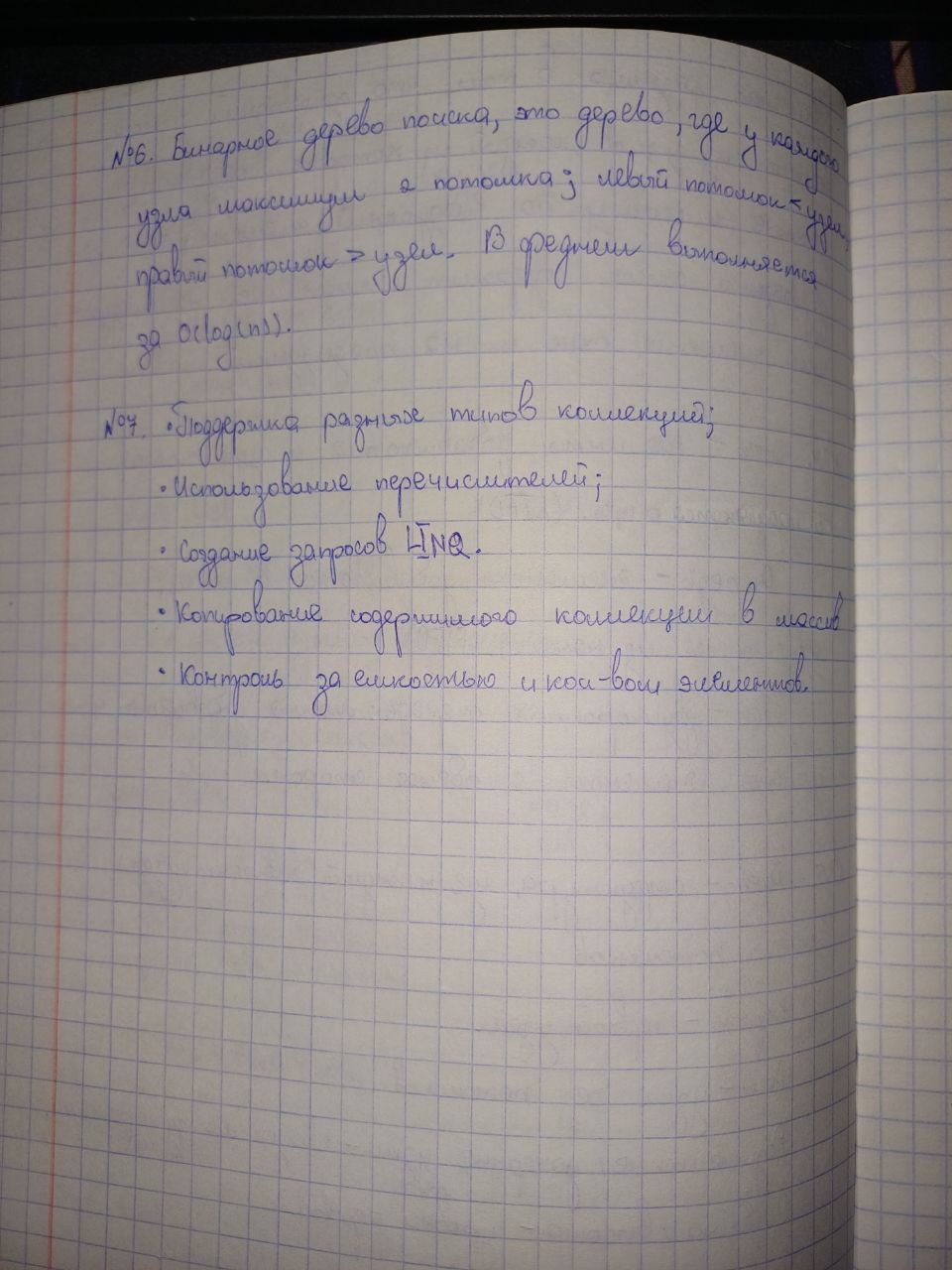
[Задание №5 12](#_Toc197640519)

[Задание №6 14](#_Toc197640520)

[Задание №7 16](#_Toc197640521)

# КОНТРОЛЬНЫЕ ВОПРОСЫ



# ПРАКТИЧЕСКИЕ ЗАДАНИЯ

## Задание №1

Хранение в виде двусвязного списка реализовано следующим образом:  
struct Country // структура данных о стране

{

public string Name;

public string Capital;

public int Population;

public string GovernmentForm;

}

struct Logs // структура для лога

{

public string Operation;

public DateTime Timestamp;

public string Details;

}

// реализация двусвязного списка для стран

class CountryNode

{

public Country Data;

public CountryNode Next;

public CountryNode Prev;

}

class CountryList

{

private CountryNode head;

private CountryNode tail;

public int Count { get; private set; }

// добавление в конец списка

public void Add(Country country)

{

CountryNode node = new CountryNode { Data = country };

if (head == null)

{

head = tail = node;

}

else

{

tail.Next = node;

node.Prev = tail;

tail = node;

}

Count++;

}

// получение страны по индексу

public Country Get(int index)

{

if (index < 0 || index >= Count)

throw new IndexOutOfRangeException();

CountryNode current = head;

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

{

current = current.Next;

}

return current.Data;

}

// удаление по индексу

public void RemoveAt(int index)

{

if (index < 0 || index >= Count)

return;

CountryNode current = head;

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

current = current.Next;

if (current.Prev != null)

current.Prev.Next = current.Next;

else

head = current.Next;

if (current.Next != null)

current.Next.Prev = current.Prev;

else

tail = current.Prev;

Count--;

}

// обновление по индексу

public void UpdateAt(int index, Country country)

{

if (index < 0 || index >= Count)

return;

CountryNode current = head;

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

current = current.Next;

current.Data = country;

}

// перечисление всех стран

public List<Country> GetAll()

{

List<Country> result = new List<Country>(); // создаём список для хранения стран

CountryNode current = head; // начинаем с головы списка

while (current != null) // пока не дошли до конца

{

result.Add(current.Data); // добавляем страну в список

current = current.Next; // переходим к следующему элементу

}

return result; // возвращаем весь список

}

// сортировка методом выбора по возрастанию населения

public void SortByPopulation()

{

if (Count < 2)

return;

for (CountryNode i = head; i != null; i = i.Next)

{

CountryNode min = i;

for (CountryNode j = i.Next; j != null; j = j.Next)

{

if (j.Data.Population < min.Data.Population)

min = j;

}

if (min != i)

{

// меняем данные

Country temp = i.Data;

i.Data = min.Data;

min.Data = temp;

}

}

}

}

Хранение с использованием коллекций .NET реализовано следующим образом:  
struct Country // структура данных о стране

{

public string Name;

public string Capital;

public int Population;

public string GovernmentForm;

}

struct Logs // структура для лога

{

public string Operation;

public DateTime Timestamp;

public string Details;

}

// Используем List для хранения данных о странах

static List<Country> countries = new List<Country>();

static List<Logs> logs = new List<Logs>();

## Задание №2

За обработку стека отвечает следующий код:  
static bool Check(string expression)

{

Stack<char> stack = new Stack<char>(); // стек для хранения открывающих скобок

foreach (char c in expression)

{

if (c == '(') // если открывающая скобка, добавляем в стек

{

stack.Push(c);

}

else if (c == ')') // если закрывающая скобка

{

if (stack.Count == 0) // если стек пуст, то скобки не сбалансированы

{

return false;

}

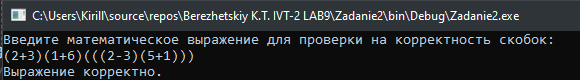
stack.Pop(); // убираем последнюю открывающую скобку из стека

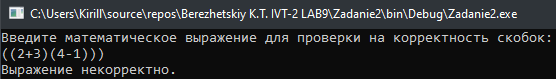
}

}

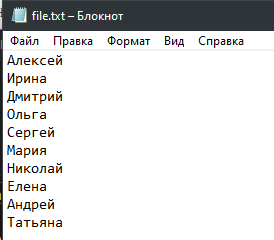
// если стек пуст, то все скобки сбалансированы

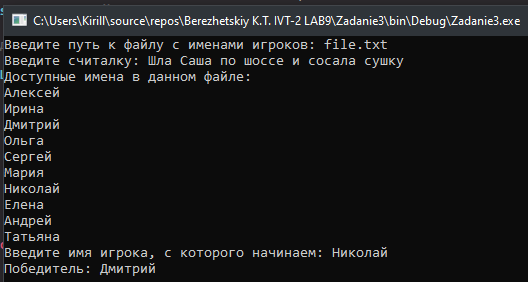
return stack.Count == 0;

}  
Т.е. если количество открывающих и закрывающих скобок равно нулю в стеке после всех условий, то строка введена верно. Пример:   




## Задание №3



Имеем такой файл с такими именами. Проверим как работает:  
  
Как видим, код выполнен верно. Имя «Дмитрий» выпало как раз на последнее слово данной считалочки, которую мы ввели.

Циклический связанный список:  
// Метод для создания циклического связного списка

static Player circularList(List<string> playerNames)

{

if (playerNames.Count == 0) return null;

Player first = new Player { Name = playerNames[0] };

Player current = first;

// Создаем список и связываем участников

for (int i = 1; i < playerNames.Count; i++)

{

current.Next = new Player { Name = playerNames[i] };

current = current.Next;

}

// Циклическое связывание

current.Next = first;

return first;

}

Когда мы доходим до конца списка, то он возвращается в начало и продолжает идти дальше. Для решения без доп. структур можно было бы использовать обычный массив, записав туда список игроков, и двигаться по индексу игроков, увеличивая его на каждом шаге и брать остаток от деления на длину массива, чтобы обеспечить "круговое" поведение

## Задание №4

Тут пришлось сохранить результаты вывода в отдельный текстовый файл, т.к. в консоли не помещались все комбинации из-за их большого количества  
const int maxN = 50000;

const int maxCube = 100; // ограничим кубы, чтобы не выходили далеко за 50000

var combinations = new Dictionary<int, int>();

for (int x = 0; x <= maxCube; x++)

{

int x3 = x \* x \* x;

for (int y = 0; y <= maxCube; y++)

{

int y3 = y \* y \* y;

for (int z = 0; z <= maxCube; z++)

{

int z3 = z \* z \* z;

int sum = x3 + y3 + z3;

if (sum > maxN)

continue;

if (combinations.ContainsKey(sum))

combinations[sum]++;

else

combinations[sum] = 1;

}

}

}

using (StreamWriter writer = new StreamWriter("результаты.txt"))

{

foreach (var pair in combinations)

{

if (pair.Value >= 3)

{

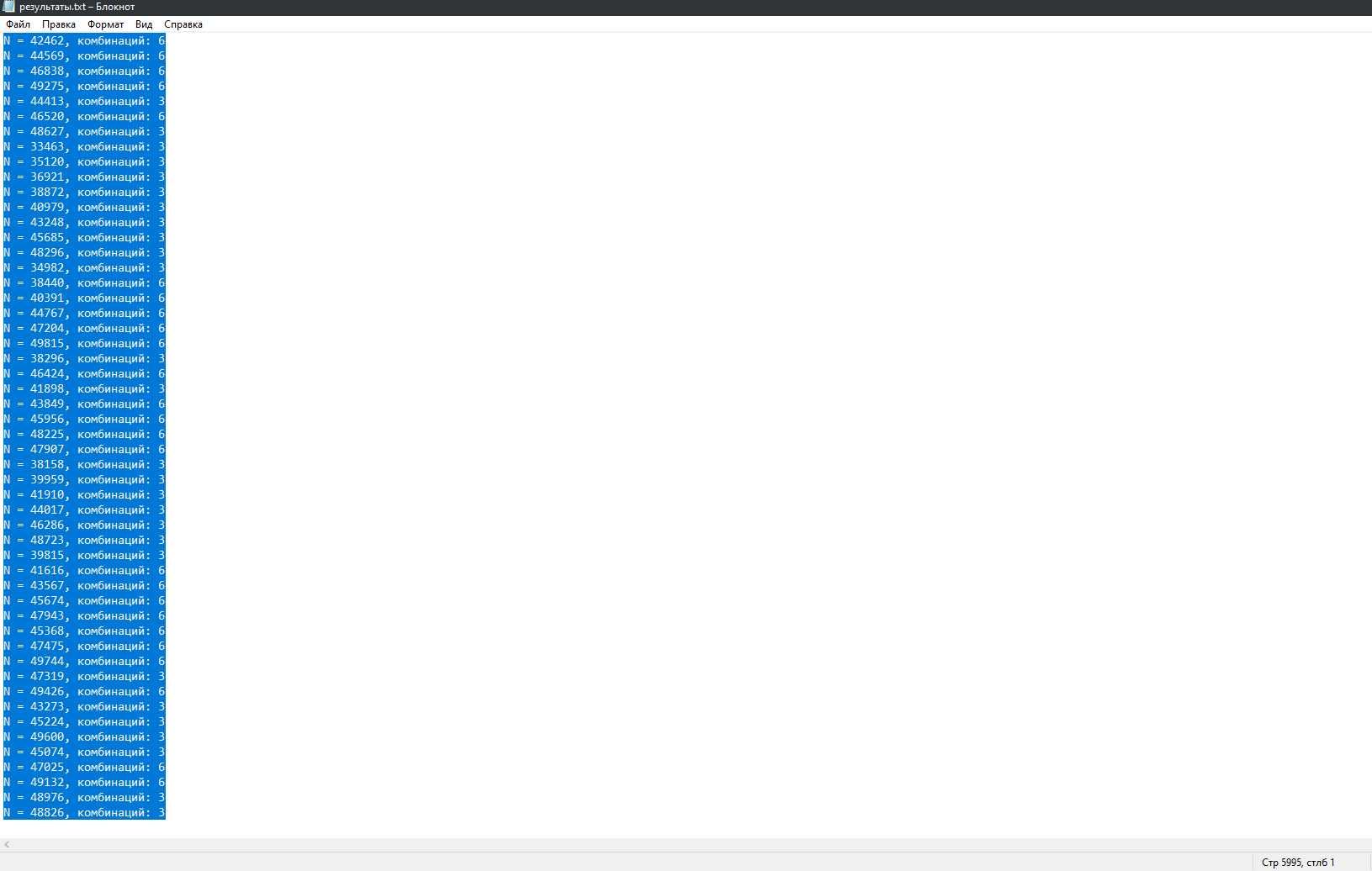
writer.WriteLine($"N = {pair.Key}, комбинаций: {pair.Value}");

}

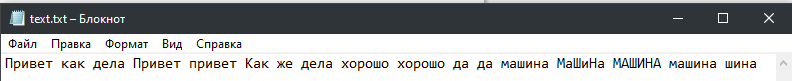
}

}

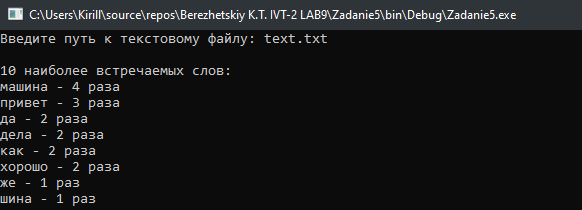
Создается словарь, где записывается ключ (число, сумма кубов) и значение (кол-во комбинаций). Тройной цикл перебирает все возможные комбинации и считает сумму кубов

Выходит вот такой вот спиоск из 5995 строк:  


## Задание №5

Имеем такой текстовый файл:  


Вывод будет таким:



Реализовано в таком формате:  
 Console.Write("Введите путь к текстовому файлу: ");

string filePath = Console.ReadLine();

if (!File.Exists(filePath))

{

Console.WriteLine("Файл не найден.");

return;

}

string text = File.ReadAllText(filePath);

char[] separators = {' '};

string[] words = text.ToLower().Split(separators, StringSplitOptions.RemoveEmptyEntries);

Dictionary<string, int> wordCount = new Dictionary<string, int>();

foreach (string word in words)

{

if (wordCount.ContainsKey(word))

{

wordCount[word] = wordCount[word] + 1;

}

else

{

wordCount[word] = 1;

}

}

// Переводим словарь в список для ручной сортировки

List<KeyValuePair<string, int>> list = new List<KeyValuePair<string, int>>();

foreach (KeyValuePair<string, int> pair in wordCount)

{

list.Add(pair);

}

// Сортируем список: сначала по убыванию количества, потом по алфавиту

for (int i = 0; i < list.Count - 1; i++)

{

for (int j = i + 1; j < list.Count; j++)

{

if (list[j].Value > list[i].Value ||

(list[j].Value == list[i].Value && string.Compare(list[j].Key, list[i].Key) < 0))

{

// меняем местами

KeyValuePair<string, int> temp = list[i];

list[i] = list[j];

list[j] = temp;

}

}

}

Console.WriteLine("\n10 наиболее встречаемых слов:");

for (int i = 0; i < 10 && i < list.Count; i++)

{

string word = list[i].Key;

int count = list[i].Value;

string form = GetForm(count);

Console.WriteLine(word + " — " + count + " " + form);

}

Console.ReadLine();

}

static string GetForm(int n)

{

if (n % 10 == 1 && n % 100 != 11)

{

return "раз";

}

else if (n % 10 >= 2 && n % 10 <= 4 && (n % 100 < 10 || n % 100 >= 20))

{

return "раза";

}

else

{

return "раз";

}

}

Тут создается словарь, где ключом будет слово, а значением – количество его вхождений. В цикле перебираются слова и проверяется, есть ли там уже слово. Если найдено – то добавляем 1, если нет – количество его останется равным 1. Далее создается список из элементов словаря, и позже сортируется с помощью сортировки пузырьком

## Задание №6

Класс с узлами дерева:  
// класс узла дерева, хранит команды, счёт и потомков

class Node

{

public string Team1;

public string Team2;

public int Score1;

public int Score2;

public Node Left;

public Node Right;

}

Код, отвечающий за генерацию дерева турнира:

// список команд, 16 штук

string[] teams = {

"BRA", "ARG", "FRA", "COL", "CHI", "URU", "GER", "NIG",

"CRC", "MEX", "NED", "GRE", "BEL", "SWI", "USA", "ALG"

};

// очередь для построения дерева матчей

Queue<Node> queue = new Queue<Node>();

// 1/16 финала — пары команд играют друг с другом

for (int i = 0; i < teams.Length; i += 2)

{

Node node = new Node();

node.Team1 = teams[i];

node.Team2 = teams[i + 1];

PlayMatch(node); // играем матч

queue.Enqueue(node); // добавляем результат в очередь

}

// сборка турнирного дерева из победителей

while (queue.Count > 1)

{

Node left = queue.Dequeue(); // берём левый матч

Node right = queue.Dequeue(); // берём правый матч

Node parent = new Node(); // создаём родительский матч

parent.Left = left; // привязываем левый матч

parent.Right = right; // привязываем правый матч

parent.Team1 = GetWinner(left); // первая команда — победитель из левого

parent.Team2 = GetWinner(right); // вторая — из правого

PlayMatch(parent); // играем новый матч

queue.Enqueue(parent); // добавляем в очередь

}

// финальный матч — корень дерева

Node root = queue.Dequeue();

Console.WriteLine("Результаты турнира:\n");

PrintTree(root, 0); // выводим дерево

Console.ReadLine(); // ждём, чтобы окно не закрылось сразу

}

// функция, играющая матч — генерирует счёт

static void PlayMatch(Node match)

{

match.Score1 = rnd.Next(0, 5); // случайный счёт от 0 до 4

match.Score2 = rnd.Next(0, 5);

while (match.Score1 == match.Score2) // ничья не допускается

{

match.Score1 = rnd.Next(0, 5);

match.Score2 = rnd.Next(0, 5);

}

}

// возвращает победителя матча

static string GetWinner(Node match)

{

return match.Score1 > match.Score2 ? match.Team1 : match.Team2;

}

// рекурсивный вывод дерева турнира

static void PrintTree(Node node, int level)

{

if (node == null)

return;

string indent = new string(' ', level \* 4); // отступ зависит от уровня

Console.WriteLine($"{indent}{node.Team1} - {node.Team2} : {node.Score1} - {node.Score2}");

PrintTree(node.Left, level + 1); // выводим левую ветку

PrintTree(node.Right, level + 1); // потом правую

}

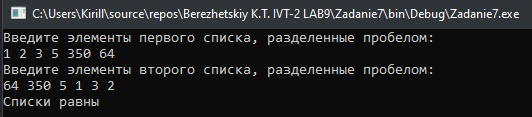
Изначально создается очередь для хранения матчей. Для каждой пары создается узел дерева, в котором задаются команды и генерируется результат матча с помощью функции PlayMatch, и результаты добавляются в очередь. Пока в очереди больше одного элемента, из очереди извлекаются пары матчей (победители предыдущих этапов), для которых создается новый матч, где играют победители этих матчей. И когда очередь содержит только один элемент, это финальный матч турнира, который становится корнем дерева

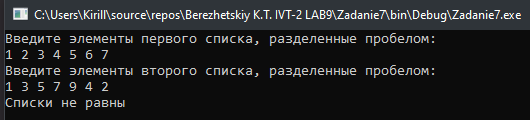
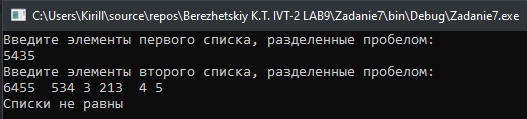




## Задание №7

В моем варианте:  
написать программу, которая проверяет равенство двух списков. Списки считать равными, если они содержат одинаковые элементы в одинаковом количестве (порядок не важен).





После того, как пользователь введет строки, они преобразуются в списки:  
Console.WriteLine("Введите элементы первого списка, разделенные пробелом:");

string input1 = Console.ReadLine();

List<int> list1 = ConvertToList(input1);

// Ввод второго списка

Console.WriteLine("Введите элементы второго списка, разделенные пробелом:");

string input2 = Console.ReadLine();

List<int> list2 = ConvertToList(input2);

Сама функция ConvertToList:  
static List<int> ConvertToList(string input)

{

string[] elements = input.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

List<int> list = new List<int>();

foreach (var element in elements)

{

if (int.TryParse(element, out int num))

{

list.Add(num);

}

else

{

Console.WriteLine($"Некорректное значение: {element}. Пропускаем.");

}

}

return list;

}  
Функция принимает строку с числами, разделенными пробелами, и преобразует её в список целых чисел. Сначала строка разбивается на массив строк с помощью Split. Затем каждое значение из массива пытается быть преобразованным в число. Если преобразование успешно, число добавляется в список. Если преобразование не удается, выводится сообщение об ошибке и элемент пропускается.

А с помощью функции Check:  
static bool Check(List<int> a, List<int> b)

{

if (a.Count != b.Count)

{

return false;

}

var dictA = new Dictionary<int, int>();

var dictB = new Dictionary<int, int>();

// считаем количество каждого элемента

foreach (int num in a)

{

if (dictA.ContainsKey(num))

{

dictA[num]++;

}

else

{

dictA[num] = 1;

}

}

foreach (int num in b)

{

if (dictB.ContainsKey(num))

{

dictB[num]++;

}

else

{

dictB[num] = 1;

}

}

// сравниваем словари

foreach (var pair in dictA)

{

if (!dictB.ContainsKey(pair.Key) || dictB[pair.Key] != pair.Value)

{

return false;

}

}

return true;

}

сначала создаются два словаря (dictA и dictB), которые будут хранить количество каждого числа в списках. Для каждого элемента списка a и b увеличиваем количество этого элемента в соответствующем словаре. После этого сравниваются оба словаря: если для любого ключа в одном словаре нет такого же ключа в другом или количество значений не совпадает, возвращается false, иначе — true. И затем проверяются эти самые два списка:  
bool equal = Check(list1, list2);

if (equal)

{

Console.WriteLine("Списки равны");

}

else

{

Console.WriteLine("Списки не равны");

}