**ЛАБОРАТОРНАЯ РАБОТА №9**

**Тема:** Абстрактные типы данных (АТД). Коллекции в .NET.

**Цель:** Научиться программировать ключевые АТД с помощью массивов, связных списков и деревьев, научиться работать с коллекциями .NET.

**Ход работы**

1. Модифицировать код проекта из лабораторной работы №5 дважды: в первом проекте данные предметной области хранить и обрабатывать в виде двусвязного списка (структуру реализовать самостоятельно), во втором – использовать коллекцию .NET List.

**С использованием двухсвязного списка**

Этот вариант позволит более эффективно добавлять, удалять и изменять элементы, не сдвигая данные, как в массиве. Такой способ также обеспечивает удобный доступ к элементам вперёд и назад, что упрощает операции сортировки.

Код:

using System;

class Program

{

    // Структура для хранения информации о транспорте

    public struct Transport

    {

        public string Type;

        public string RouteNumber;

        public double Length;

        public int Time;

    }

    // Структура для записи логов действий пользователя

    struct LogEntry

    {

        public string Action;

        public string Details;

        public DateTime Timestamp;

    }

    // Узел двусвязного списка для хранения информации о транспорте

    public class TransportNode

    {

        public Transport Data;

        public TransportNode Next;    // Ссылка на следующий узел

        public TransportNode Prev;    // Ссылка на предыдущий узел

        // Конструктор

        public TransportNode(Transport data)

        {

            Data = data;

            Next = null;

            Prev = null;

        }

    }

    // Двусвязный список для хранения информации о транспорте

    public class TransportList

    {

        public TransportNode Head;   // Голова списка

        public TransportNode Tail;   // Хвост списка

        public int Count;            // Количество элементов в списке

        // Конструктор

        public TransportList()

        {

            Head = null;

            Tail = null;

            Count = 0;

        }

        // Метод для добавления записи в список

        public void Add(Transport transport)

        {

            TransportNode newNode = new TransportNode(transport);

            if (Tail == null)  // Если список пуст

            {

                Head = newNode;

                Tail = newNode;

            }

            else

            {

                Tail.Next = newNode;

                newNode.Prev = Tail;

                Tail = newNode;

            }

            Count++;

        }

        // Метод для удаления записи по индексу

        public void Delete(int index)

        {

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

                return;

            TransportNode current = Head;

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

            {

                current = current.Next;

            }

            if (current.Prev != null)

                current.Prev.Next = current.Next;

            if (current.Next != null)

                current.Next.Prev = current.Prev;

            if (current == Head)

                Head = current.Next;

            if (current == Tail)

                Tail = current.Prev;

            Count--;

        }

        // Метод для сортировки списка по выбранному критерию

        public void Sort(int sortChoice)

        {

            if (Head == null)

                return;

            // Алгоритм сортировки выбором (для упрощения)

            for (TransportNode i = Head; i != null; i = i.Next)

            {

                TransportNode minNode = i;

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

                {

                    bool condition = false;

                    if (sortChoice == 1 && j.Data.Length < minNode.Data.Length)

                    {

                        condition = true;

                    }

                    else if (sortChoice == 2 && j.Data.Time < minNode.Data.Time)

                    {

                        condition = true;

                    }

                    if (condition)

                    {

                        minNode = j;

                    }

                }

                if (minNode != i)

                {

                    // Меняем данные

                    Transport temp = i.Data;

                    i.Data = minNode.Data;

                    minNode.Data = temp;

                }

            }

        }

        // Метод для получения записи по индексу

        public Transport Get(int index)

        {

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

                return default(Transport);

            TransportNode current = Head;

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

            {

                current = current.Next;

            }

            return current.Data;

        }

        // Метод для обновления записи по индексу

        public void Update(int index, Transport updatedTransport)

        {

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

                return;

            TransportNode current = Head;

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

            {

                current = current.Next;

            }

            current.Data = updatedTransport;

        }

    }

    static void Main()

    {

        TransportList transports = new TransportList();  // Двусвязный список

        LogEntry[] logs = new LogEntry[50];

        int logCount = 0;

        DateTime lastActionTime = DateTime.Now;

        TimeSpan maxIdleTime = TimeSpan.Zero;

        // Чтение данных из файла при запуске программы

        LoadData(ref transports, ref logs, ref logCount);

        while (true)

        {

            Console.WriteLine("\nДоступные действия:");

            Console.WriteLine("1 – Показать таблицу");

            Console.WriteLine("2 – Отсортировать таблицу");

            Console.WriteLine("3 – Добавить новую запись");

            Console.WriteLine("4 – Удалить запись");

            Console.WriteLine("5 – Обновить запись");

            Console.WriteLine("6 – Найти записи");

            Console.WriteLine("7 – Показать лог действий");

            Console.WriteLine("8 – Завершить работу");

            Console.Write("Введите номер действия: ");

            int choice = Convert.ToInt32(Console.ReadLine());

            // Время простоя

            TimeSpan idleTime = DateTime.Now - lastActionTime;

            if (idleTime > maxIdleTime)

                maxIdleTime = idleTime;

            // Время последнего действия

            lastActionTime = DateTime.Now;

            switch (choice)

            {

                case 1:

                    ViewTable(transports);

                    break;

                case 2:

                    Console.WriteLine("\nВыберите столбец для сортировки:");

                    Console.WriteLine("1 – По длине маршрута");

                    Console.WriteLine("2 – По времени маршрута");

                    int sortChoice = Convert.ToInt32(Console.ReadLine());

                    transports.Sort(sortChoice);

                    break;

                case 3:

                    AddRecord(ref transports, ref logs, ref logCount);

                    break;

                case 4:

                    DeleteRecord(ref transports, ref logs, ref logCount);

                    break;

                case 5:

                    UpdateRecord(ref transports, ref logs, ref logCount);

                    break;

                case 6:

                    SearchRecords(transports);

                    break;

                case 7:

                    ViewLog(logs, logCount, maxIdleTime);

                    break;

                case 8:

                    Console.WriteLine("Завершение работы программы.");

                    // Сохранение данных в файл

                    SaveData(transports, logs, logCount);

                    return;

                default:

                    Console.WriteLine("Неверный ввод. Пожалуйста, выберите пункт из списка.");

                    break;

            }

        }

    }

    // Метод для записи данных в файл

    static void SaveData(TransportList transports, LogEntry[] logs, int logCount)

    {

        using (BinaryWriter writer = new BinaryWriter(File.Open("lab.dat", FileMode.Create)))

        {

            // Записываем количество элементов в списке и в логе

            writer.Write(transports.Count);

            writer.Write(logCount);

            // Записываем данные о транспорте

            TransportNode current = transports.Head;

            while (current != null)

            {

                writer.Write(current.Data.Type);

                writer.Write(current.Data.RouteNumber);

                writer.Write(current.Data.Length);

                writer.Write(current.Data.Time);

                current = current.Next;

            }

            // Записываем данные о логах

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

            {

                writer.Write(logs[i].Action);

                writer.Write(logs[i].Details);

                writer.Write(logs[i].Timestamp.ToString());

            }

        }

    }

    // Метод для чтения данных из файла

    static void LoadData(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        if (File.Exists("lab.dat"))

        {

            using (BinaryReader reader = new BinaryReader(File.Open("lab.dat", FileMode.Open)))

            {

                int transportCount = reader.ReadInt32();

                logCount = reader.ReadInt32();

                // Загружаем данные о транспорте

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

                {

                    Transport newTransport = new Transport

                    {

                        Type = reader.ReadString(),

                        RouteNumber = reader.ReadString(),

                        Length = reader.ReadDouble(),

                        Time = reader.ReadInt32()

                    };

                    transports.Add(newTransport);

                }

                // Загружаем данные о логах

                logs = new LogEntry[logCount];

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

                {

                    logs[i].Action = reader.ReadString();

                    logs[i].Details = reader.ReadString();

                    logs[i].Timestamp = DateTime.Parse(reader.ReadString());

                }

            }

        }

    }

    // Метод для отображения таблицы

    static void ViewTable(TransportList transports)

    {

        Console.WriteLine("\nСписок транспорта");

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

        Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта",

            "Маршрут", "Дистанция (км)", "Длительность (мин)");

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

        TransportNode current = transports.Head;

        while (current != null)

        {

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", current.Data.Type,

                current.Data.RouteNumber, current.Data.Length, current.Data.Time);

            current = current.Next;

        }

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

    }

    // Метод для добавления записи

    static void AddRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Transport newTransport = new Transport();

        string[] validTypes = { "Тр", "А", "М" };

        bool validType = false;

        while (!validType)

        {

            Console.Write("Введите тип транспорта (Тр - трамвай, А - автобус, М - метро): ");

            newTransport.Type = Console.ReadLine().Trim();

            if (Array.Exists(validTypes, type => type == newTransport.Type))

            {

                validType = true;

            }

            else

            {

                Console.WriteLine("Неверный тип транспорта. Пожалуйста, выберите " +

                    "один из предложенных.");

            }

        }

        Console.Write("Введите номер маршрута: ");

        newTransport.RouteNumber = Console.ReadLine();

        Console.Write("Введите длину маршрута (в км): ");

        newTransport.Length = Convert.ToDouble(Console.ReadLine());

        Console.Write("Введите длительность маршрута (в минутах): ");

        newTransport.Time = Convert.ToInt32(Console.ReadLine());

        transports.Add(newTransport);

        AddLog(ref logs, ref logCount, "Добавление", $"Добавлен маршрут {newTransport.RouteNumber}");

    }

    // Метод для удаления записи

    static void DeleteRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Console.Write("Введите номер записи для удаления: ");

        int index = Convert.ToInt32(Console.ReadLine()) - 1;

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

        {

            Console.WriteLine("Некорректный номер записи.");

            return;

        }

        Transport deletedTransport = transports.Get(index);

        transports.Delete(index);

        AddLog(ref logs, ref logCount, "Удаление", $"Удален маршрут {deletedTransport.RouteNumber}");

    }

    // Метод для обновления записи

    static void UpdateRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Console.Write("Введите номер записи для обновления: ");

        int index = Convert.ToInt32(Console.ReadLine()) - 1;

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

        {

            Console.WriteLine("Ошибка: некорректный номер записи.");

            return;

        }

        Transport updatedTransport;

        Console.Write("Введите новый тип транспорта: ");

        updatedTransport.Type = Console.ReadLine();

        Console.Write("Введите новый номер маршрута: ");

        updatedTransport.RouteNumber = Console.ReadLine();

        Console.Write("Введите новую длину маршрута (в км): ");

        updatedTransport.Length = Convert.ToDouble(Console.ReadLine());

        Console.Write("Введите новую длительность маршрута (в минутах): ");

        updatedTransport.Time = Convert.ToInt32(Console.ReadLine());

        transports.Update(index, updatedTransport);

        AddLog(ref logs, ref logCount, "Обновление", $"Обновлен маршрут {updatedTransport.RouteNumber}");

    }

    // Метод для поиска записей

    static void SearchRecords(TransportList transports)

    {

        Console.WriteLine("Выберите фильтр поиска:");

        Console.WriteLine("1 – Поиск по длине маршрута");

        Console.WriteLine("2 – Поиск по времени маршрута");

        Console.Write("Введите номер фильтра: ");

        int filterChoice = Convert.ToInt32(Console.ReadLine());

        if (filterChoice == 1)

        {

            Console.Write("Введите минимальную длину маршрута (в км): ");

            double minLength = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("\nРезультаты поиска по длине маршрута:");

            Console.WriteLine("-------------------------------------------------------------" +

                "-----------------------");

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта", "Маршрут",

                "Дистанция (км)", "Длительность (мин)");

            Console.WriteLine("--------------------------------------------------------------" +

                "----------------------");

            TransportNode current = transports.Head;

            while (current != null)

            {

                if (current.Data.Length >= minLength)

                {

                    Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", current.Data.Type,

                        current.Data.RouteNumber, current.Data.Length, current.Data.Time);

                }

                current = current.Next;

            }

        }

        else if (filterChoice == 2)

        {

            Console.Write("Введите минимальное время маршрута (в минутах): ");

            int minTime = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("\nРезультаты поиска по времени маршрута:");

            Console.WriteLine("------------------------------------------------------------" +

                "------------------------");

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта", "Маршрут",

                "Дистанция (км)", "Длительность (мин)");

            Console.WriteLine("--------------------------------------------------------------" +

                "----------------------");

            TransportNode current = transports.Head;

            while (current != null)

            {

                if (current.Data.Time >= minTime)

                {

                    Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", current.Data.Type,

                        current.Data.RouteNumber, current.Data.Length, current.Data.Time);

                }

                current = current.Next;

            }

        }

        else

        {

            Console.WriteLine("Неверный выбор фильтра.");

        }

        Console.WriteLine("---------------------------------------------------------------" +

            "---------------------");

    }

    // Метод для отображения лога

    static void ViewLog(LogEntry[] logs, int count, TimeSpan maxIdleTime)

    {

        Console.WriteLine("\nЛог действий:");

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

        {

            Console.WriteLine($"{logs[i].Timestamp:HH:mm:ss} – {logs[i].Action}: {logs[i].Details}");

        }

        Console.WriteLine($"\n{maxIdleTime:hh\\:mm\\:ss} – Самый долгий период бездействия.");

    }

    // Метод для добавления записи в лог

    static void AddLog(ref LogEntry[] logs, ref int count, string action, string details)

    {

        if (count >= logs.Length)

        {

            for (int i = 1; i < logs.Length; i++)

            {

                logs[i - 1] = logs[i];

            }

            count--;

        }

        logs[count++] = new LogEntry

        {

            Action = action,

            Details = details,

            Timestamp = DateTime.Now

        };

    }

}

Тестирование отображать в отчёте не имеет ценности, так как функционал программы не изменился, всё работает как прежде.

**С использованием .NET List**

Использование .NET List значительно упрощает работу с данными, освобождая от необходимости вручную управлять размером или ссылками, как в массиве или двусвязных списках, при этом сохраняя эффективность.

Код:

using System;

using System.Collections.Generic;

using System.IO;

class Program

{

    // Структура для хранения информации о транспорте

    public struct Transport

    {

        public string Type;

        public string RouteNumber;

        public double Length;

        public int Time;

    }

    // Структура для записи логов действий пользователя

    struct LogEntry

    {

        public string Action;

        public string Details;

        public DateTime Timestamp;

    }

    // Класс для хранения списка информации о транспорте

    public class TransportList

    {

        public List<Transport> Transports; // Список для хранения данных о транспорте

        // Конструктор

        public TransportList()

        {

            Transports = new List<Transport>();

        }

        // Метод для добавления записи в список

        public void Add(Transport transport)

        {

            Transports.Add(transport);

        }

        // Метод для удаления записи по индексу

        public void Delete(int index)

        {

            if (index >= 0 && index < Transports.Count)

            {

                Transports.RemoveAt(index);

            }

        }

        // Метод для сортировки списка по выбранному критерию

        public void Sort(int sortChoice)

        {

            if (sortChoice == 1)

            {

                Transports.Sort((x, y) => x.Length.CompareTo(y.Length));

            }

            else if (sortChoice == 2)

            {

                Transports.Sort((x, y) => x.Time.CompareTo(y.Time));

            }

        }

        // Метод для получения записи по индексу

        public Transport Get(int index)

        {

            if (index >= 0 && index < Transports.Count)

            {

                return Transports[index];

            }

            return default(Transport);

        }

        // Метод для обновления записи по индексу

        public void Update(int index, Transport updatedTransport)

        {

            if (index >= 0 && index < Transports.Count)

            {

                Transports[index] = updatedTransport;

            }

        }

    }

    static void Main()

    {

        TransportList transports = new TransportList();  // Список транспорта

        LogEntry[] logs = new LogEntry[50];

        int logCount = 0;

        DateTime lastActionTime = DateTime.Now;

        TimeSpan maxIdleTime = TimeSpan.Zero;

        // Чтение данных из файла при запуске программы

        LoadData(ref transports, ref logs, ref logCount);

        while (true)

        {

            Console.WriteLine("\nДоступные действия:");

            Console.WriteLine("1 – Показать таблицу");

            Console.WriteLine("2 – Отсортировать таблицу");

            Console.WriteLine("3 – Добавить новую запись");

            Console.WriteLine("4 – Удалить запись");

            Console.WriteLine("5 – Обновить запись");

            Console.WriteLine("6 – Найти записи");

            Console.WriteLine("7 – Показать лог действий");

            Console.WriteLine("8 – Завершить работу");

            Console.Write("Введите номер действия: ");

            int choice = Convert.ToInt32(Console.ReadLine());

            // Время простоя

            TimeSpan idleTime = DateTime.Now - lastActionTime;

            if (idleTime > maxIdleTime)

                maxIdleTime = idleTime;

            // Время последнего действия

            lastActionTime = DateTime.Now;

            switch (choice)

            {

                case 1:

                    ViewTable(transports);

                    break;

                case 2:

                    Console.WriteLine("\nВыберите столбец для сортировки:");

                    Console.WriteLine("1 – По длине маршрута");

                    Console.WriteLine("2 – По времени маршрута");

                    int sortChoice = Convert.ToInt32(Console.ReadLine());

                    transports.Sort(sortChoice);

                    break;

                case 3:

                    AddRecord(ref transports, ref logs, ref logCount);

                    break;

                case 4:

                    DeleteRecord(ref transports, ref logs, ref logCount);

                    break;

                case 5:

                    UpdateRecord(ref transports, ref logs, ref logCount);

                    break;

                case 6:

                    SearchRecords(transports);

                    break;

                case 7:

                    ViewLog(logs, logCount, maxIdleTime);

                    break;

                case 8:

                    Console.WriteLine("Завершение работы программы.");

                    // Сохранение данных в файл

                    SaveData(transports, logs, logCount);

                    return;

                default:

                    Console.WriteLine("Неверный ввод. Пожалуйста, выберите пункт из списка.");

                    break;

            }

        }

    }

    // Метод для записи данных в файл

    static void SaveData(TransportList transports, LogEntry[] logs, int logCount)

    {

        using (BinaryWriter writer = new BinaryWriter(File.Open("lab.dat", FileMode.Create)))

        {

            // Записываем количество элементов в списке и в логе

            writer.Write(transports.Transports.Count);

            writer.Write(logCount);

            // Записываем данные о транспорте

            foreach (var transport in transports.Transports)

            {

                writer.Write(transport.Type);

                writer.Write(transport.RouteNumber);

                writer.Write(transport.Length);

                writer.Write(transport.Time);

            }

            // Записываем данные о логах

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

            {

                writer.Write(logs[i].Action);

                writer.Write(logs[i].Details);

                writer.Write(logs[i].Timestamp.ToString());

            }

        }

    }

    // Метод для чтения данных из файла

    static void LoadData(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        if (File.Exists("lab.dat"))

        {

            using (BinaryReader reader = new BinaryReader(File.Open("lab.dat", FileMode.Open)))

            {

                int transportCount = reader.ReadInt32();

                logCount = reader.ReadInt32();

                // Загружаем данные о транспорте

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

                {

                    Transport newTransport = new Transport

                    {

                        Type = reader.ReadString(),

                        RouteNumber = reader.ReadString(),

                        Length = reader.ReadDouble(),

                        Time = reader.ReadInt32()

                    };

                    transports.Add(newTransport);

                }

                // Загружаем данные о логах

                logs = new LogEntry[logCount];

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

                {

                    logs[i].Action = reader.ReadString();

                    logs[i].Details = reader.ReadString();

                    logs[i].Timestamp = DateTime.Parse(reader.ReadString());

                }

            }

        }

    }

    // Метод для отображения таблицы

    static void ViewTable(TransportList transports)

    {

        Console.WriteLine("\nСписок транспорта");

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

        Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта",

            "Маршрут", "Дистанция (км)", "Длительность (мин)");

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

        foreach (var transport in transports.Transports)

        {

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", transport.Type,

                transport.RouteNumber, transport.Length, transport.Time);

        }

        Console.WriteLine("--------------------------------------------------------------" +

            "----------------------");

    }

    // Метод для добавления записи

    static void AddRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Transport newTransport = new Transport();

        string[] validTypes = { "Тр", "А", "М" };

        bool validType = false;

        while (!validType)

        {

            Console.Write("Введите тип транспорта (Тр - трамвай, А - автобус, М - метро): ");

            newTransport.Type = Console.ReadLine().Trim();

            if (Array.Exists(validTypes, type => type == newTransport.Type))

            {

                validType = true;

            }

            else

            {

                Console.WriteLine("Неверный тип транспорта. Пожалуйста, выберите " +

                    "один из предложенных.");

            }

        }

        Console.Write("Введите номер маршрута: ");

        newTransport.RouteNumber = Console.ReadLine();

        Console.Write("Введите длину маршрута (в км): ");

        newTransport.Length = Convert.ToDouble(Console.ReadLine());

        Console.Write("Введите длительность маршрута (в минутах): ");

        newTransport.Time = Convert.ToInt32(Console.ReadLine());

        transports.Add(newTransport);

        AddLog(ref logs, ref logCount, "Добавление", $"Добавлен маршрут {newTransport.RouteNumber}");

    }

    // Метод для удаления записи

    static void DeleteRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Console.Write("Введите номер записи для удаления: ");

        int index = Convert.ToInt32(Console.ReadLine()) - 1;

        if (index < 0 || index >= transports.Transports.Count)

        {

            Console.WriteLine("Некорректный номер записи.");

            return;

        }

        Transport deletedTransport = transports.Get(index);

        transports.Delete(index);

        AddLog(ref logs, ref logCount, "Удаление", $"Удален маршрут {deletedTransport.RouteNumber}");

    }

    // Метод для обновления записи

    static void UpdateRecord(ref TransportList transports, ref LogEntry[] logs, ref int logCount)

    {

        Console.Write("Введите номер записи для обновления: ");

        int index = Convert.ToInt32(Console.ReadLine()) - 1;

        if (index < 0 || index >= transports.Transports.Count)

        {

            Console.WriteLine("Ошибка: некорректный номер записи.");

            return;

        }

        Transport updatedTransport;

        Console.Write("Введите новый тип транспорта: ");

        updatedTransport.Type = Console.ReadLine();

        Console.Write("Введите новый номер маршрута: ");

        updatedTransport.RouteNumber = Console.ReadLine();

        Console.Write("Введите новую длину маршрута (в км): ");

        updatedTransport.Length = Convert.ToDouble(Console.ReadLine());

        Console.Write("Введите новую длительность маршрута (в минутах): ");

        updatedTransport.Time = Convert.ToInt32(Console.ReadLine());

        transports.Update(index, updatedTransport);

        AddLog(ref logs, ref logCount, "Обновление", $"Обновлен маршрут {updatedTransport.RouteNumber}");

    }

    // Метод для поиска записей

    static void SearchRecords(TransportList transports)

    {

        Console.WriteLine("Выберите фильтр поиска:");

        Console.WriteLine("1 – Поиск по длине маршрута");

        Console.WriteLine("2 – Поиск по времени маршрута");

        Console.Write("Введите номер фильтра: ");

        int filterChoice = Convert.ToInt32(Console.ReadLine());

        if (filterChoice == 1)

        {

            Console.Write("Введите минимальную длину маршрута (в км): ");

            double minLength = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("\nРезультаты поиска по длине маршрута:");

            Console.WriteLine("-------------------------------------------------------------" +

                "-----------------------");

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта", "Маршрут",

                "Дистанция (км)", "Длительность (мин)");

            Console.WriteLine("--------------------------------------------------------------" +

                "----------------------");

            foreach (var transport in transports.Transports)

            {

                if (transport.Length >= minLength)

                {

                    Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", transport.Type,

                        transport.RouteNumber, transport.Length, transport.Time);

                }

            }

        }

        else if (filterChoice == 2)

        {

            Console.Write("Введите минимальное время маршрута (в минутах): ");

            int minTime = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("\nРезультаты поиска по времени маршрута:");

            Console.WriteLine("------------------------------------------------------------" +

                "------------------------");

            Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", "Тип транспорта", "Маршрут",

                "Дистанция (км)", "Длительность (мин)");

            Console.WriteLine("--------------------------------------------------------------" +

                "----------------------");

            foreach (var transport in transports.Transports)

            {

                if (transport.Time >= minTime)

                {

                    Console.WriteLine("{0,-15} {1,-15} {2,-30} {3,-20}", transport.Type,

                        transport.RouteNumber, transport.Length, transport.Time);

                }

            }

        }

        else

        {

            Console.WriteLine("Неверный выбор фильтра.");

        }

        Console.WriteLine("---------------------------------------------------------------" +

            "---------------------");

    }

    // Метод для отображения лога

    static void ViewLog(LogEntry[] logs, int count, TimeSpan maxIdleTime)

    {

        Console.WriteLine("\nЛог действий:");

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

        {

            Console.WriteLine($"{logs[i].Timestamp:HH:mm:ss} – {logs[i].Action}: {logs[i].Details}");

        }

        Console.WriteLine($"\n{maxIdleTime:hh\\:mm\\:ss} – Самый долгий период бездействия.");

    }

    // Метод для добавления записи в лог

    static void AddLog(ref LogEntry[] logs, ref int count, string action, string details)

    {

        if (count >= logs.Length)

        {

            for (int i = 1; i < logs.Length; i++)

            {

                logs[i - 1] = logs[i];

            }

            count--;

        }

        logs[count++] = new LogEntry

        {

            Action = action,

            Details = details,

            Timestamp = DateTime.Now

        };

    }

}

В этом случае отображение тестирования в отчёте также излишне, так как программа имеет тот же функционал.

2. *С помощью стека* проверить, что математическое выражение в скобках записано корректно. Примеры корректных выражений:

(2+3)(1+6)(((2-3)(5+1)))

2(3)(1+6(7+2))((2-3)(5+1))

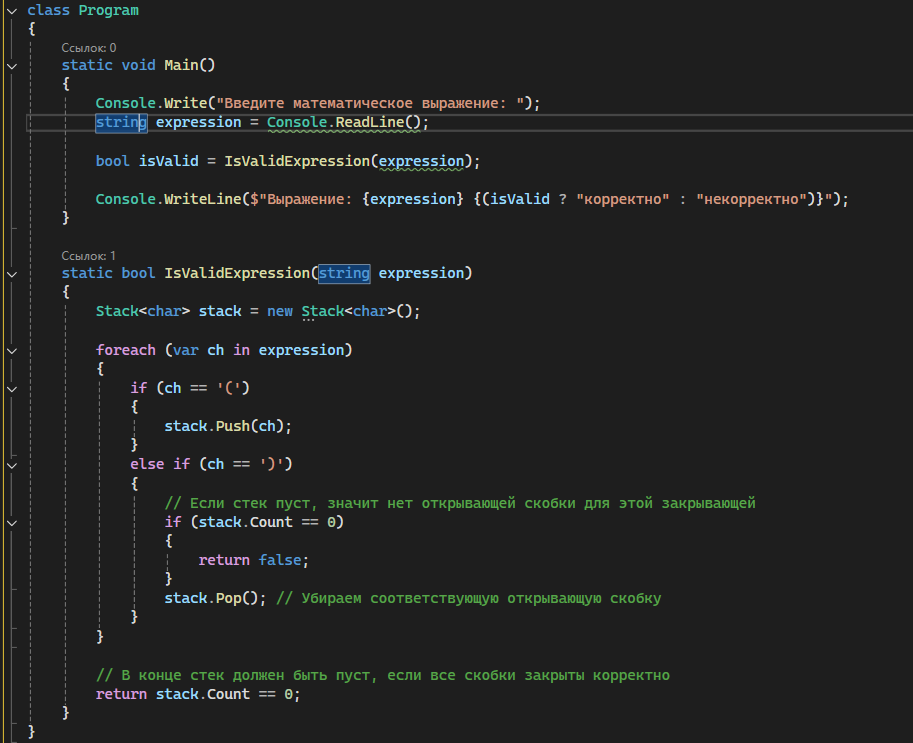
2(3+5(((6))))

Примеры некорректных выражений:

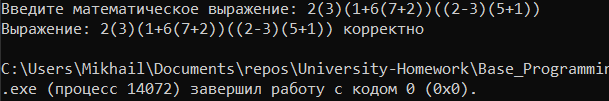
((2+3)(4-1)))

2(3+5(((6))

*Примечание.* Основной признак корректности выражения – наличие для каждой открывающей скобки соответствующей закрывающей.



Тестирование:









3. Написать игру «Считалка», в коде которой будет эмулироваться сам процесс игры с помощью циклического связного списка. В игре участвуют от 5 до 10 человек (имена участников ввести из текстового файла либо «прошить» в коде). Пользователь вводит строку считалки и указывает человека, с которого необходимо начать. Программа должна вывести человека, на которого придется последнее слово строки. Как можно данную задачу решить без дополнительных структур данных?

Код:

using System;

using System.Collections.Generic;

class Program

{

    // Класс-узел в циклическом связном списке

    class Node

    {

        public string Name;

        public Node Next;    // Ссылка на следующий элемент

        public Node(string name)

        {

            Name = name;

            Next = null;

        }

    }

    // Класс-циклический связный список

    class CircularLinkedList

    {

        public Node Head;

        // Метод для добавления элемента в список

        public void Add(string name)

        {

            Node newNode = new Node(name);

            if (Head == null)

            {

                Head = newNode;

                Head.Next = Head;  // Ссылка на себя для замыкания цикла

            }

            else

            {

                Node current = Head;

                while (current.Next != Head)

                {

                    current = current.Next;

                }

                current.Next = newNode;

                newNode.Next = Head;

            }

        }

        // Метод для получения участника по индексу

        public Node GetNodeAt(int index)

        {

            Node current = Head;

            int count = 0;

            while (count < index)

            {

                current = current.Next;

                count++;

            }

            return current;

        }

    }

    static void Main(string[] args)

    {

        // Загрузка имен участников

        List<string> participants = new List<string>

        {

            "Иван", "Мария", "Петр", "Анна", "Елена", "Дмитрий", "Оля"

        };

        // Создаем связный список

        CircularLinkedList circle = new CircularLinkedList();

        foreach (var name in participants)

        {

            circle.Add(name);

        }

        // Строки считалки

        string[] rhymeLines = new string[]

        {

            "Посчитаем дыры в сыре.",

            "Если в сыре много дыр,",

            "Значит, вкусным будет сыр.",

            "Если в нём одна дыра, ",

            "Значит, вкусным был вчера!"

        };

        //Console.WriteLine("Строки считалки:");

        //for (int i = 0; i < rhymeLines.Length; i++)

        //{

        //    Console.WriteLine($"{i + 1}. {rhymeLines[i]}");

        //}

        Console.WriteLine("Выберите строку считалки (введите номер от 1 до 5):");

        int rhymeChoice = int.Parse(Console.ReadLine()) - 1;

        if (rhymeChoice < 0 || rhymeChoice >= rhymeLines.Length)

        {

            Console.WriteLine("Неверный выбор строки считалки!");

            return;

        }

        string rhyme = rhymeLines[rhymeChoice];

        string[] words = rhyme.Split(new[] { ' ', '.', ',', '!', '?' }, StringSplitOptions.RemoveEmptyEntries);

        int numWords = words.Length;

        Console.WriteLine("\nСписок участников:");

        for (int i = 0; i < participants.Count; i++)

        {

            Console.WriteLine($"{i}. {participants[i]}");

        }

        Console.WriteLine("\nС какого участника начинаем? (Введите индекс от 0 до {0}):", participants.Count - 1);

        int startIndex = int.Parse(Console.ReadLine());

        if (startIndex < 0 || startIndex >= participants.Count)

        {

            Console.WriteLine("Неверный индекс участника!");

            return;

        }

        // Получаем начальную точку

        Node startNode = circle.GetNodeAt(startIndex);

        // Моделируем игру

        Node current = startNode;

        for (int i = 0; i < numWords - 1; i++)

        {

            current = current.Next;

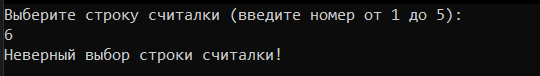
        }

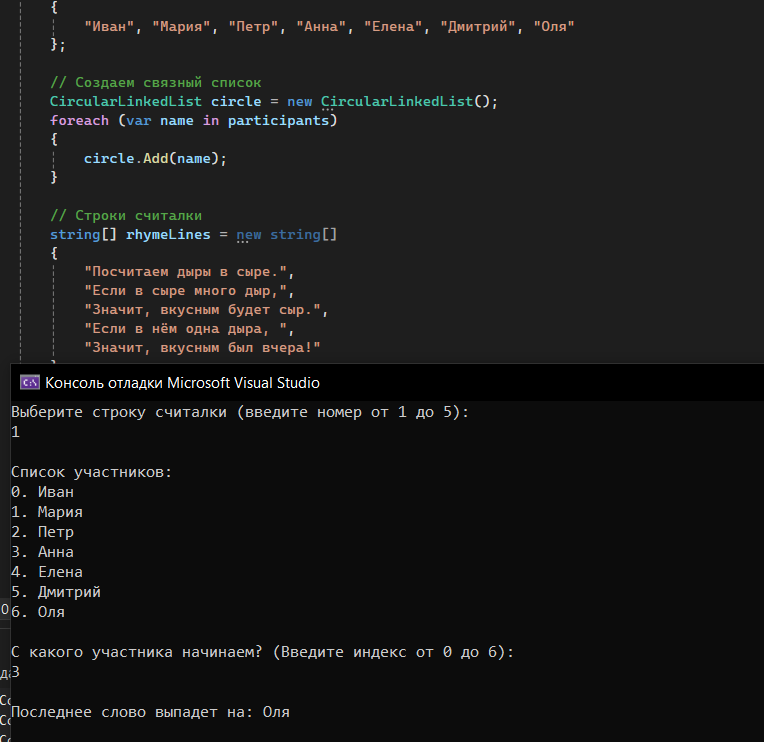
        Console.WriteLine("\nПоследнее слово выпадет на: " + current.Name);

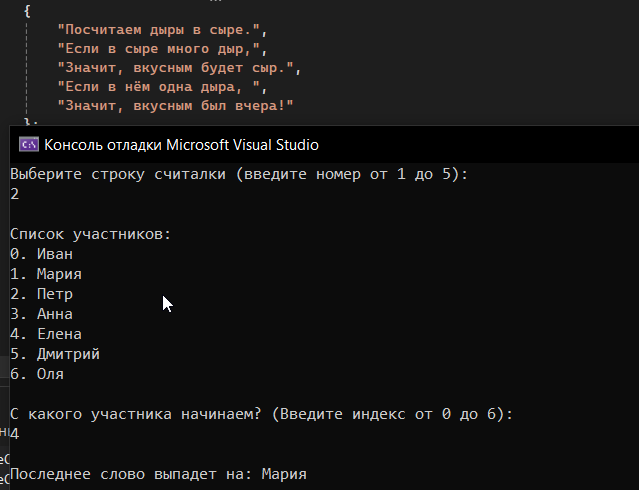
    }

}

Тестирование:







4. Модифицировать код задания 5 из лабораторной работы №2: перебрать все числа N от 1 до 50000 и вывести только те числа, для которых существует более двух (три и более) комбинаций суммы кубов. Использовать ассоциативный массив .NET (Dictionary) для решения задачи.

Код:

using System;

using System.Collections.Generic;

class Program

{

    static void Main()

    {

        int N = 50000;

        // Словарь для хранения суммы кубов и их количества

        Dictionary<int, List<string>> sumOfCubes = new Dictionary<int, List<string>>();

        // Перебираем все возможные кубы для x, y и z

        for (int x = 1; x \* x \* x <= N; x++)

        {

            for (int y = x; y \* y \* y <= N; y++) // y >= x для избегания повторений

            {

                for (int z = y; z \* z \* z <= N; z++) // z >= y для избегания повторений

                {

                    int sum = x \* x \* x + y \* y \* y + z \* z \* z;

                    if (sum <= N)

                    {

                        string combination = $"x = {x}, y = {y}, z = {z}";

                        if (!sumOfCubes.ContainsKey(sum))

                        {

                            sumOfCubes[sum] = new List<string>();

                        }

                        sumOfCubes[sum].Add(combination);

                    }

                }

            }

        }

        bool found = false;

        foreach (var entry in sumOfCubes)

        {

            if (entry.Value.Count >= 3)

            {

                found = true;

                Console.WriteLine($"Число {entry.Key} может быть представлено следующими комбинациями:");

                foreach (var combo in entry.Value)

                {

                    Console.WriteLine(combo);

                }

            }

        }

        if (!found)

        {

            Console.WriteLine("Не было найдено подходящих чисел.");

        }

    }

}

Тестирование:

Число 17604 может быть представлено следующими комбинациями:

x = 1, y = 3, z = 26

x = 7, y = 20, z = 21

x = 17, y = 18, z = 19

Число 46684 может быть представлено следующими комбинациями:

x = 1, y = 3, z = 36

x = 1, y = 27, z = 30

x = 7, y = 28, z = 29

Число 12384 может быть представлено следующими комбинациями:

x = 1, y = 6, z = 23

x = 2, y = 12, z = 22

x = 15, y = 16, z = 17

Число 5104 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 15

x = 2, y = 10, z = 16

x = 9, y = 10, z = 15

Число 9729 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 20

x = 5, y = 7, z = 21

x = 9, y = 10, z = 20

Число 13896 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 23

x = 2, y = 4, z = 24

x = 4, y = 18, z = 20

x = 9, y = 10, z = 23

Число 21412 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 27

x = 9, y = 10, z = 27

x = 9, y = 19, z = 24

Число 34497 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 32

x = 9, y = 10, z = 32

x = 11, y = 15, z = 31

Число 41033 может быть представлено следующими комбинациями:

x = 1, y = 12, z = 34

x = 9, y = 10, z = 34

x = 10, y = 16, z = 33

Число 14175 может быть представлено следующими комбинациями:

x = 1, y = 17, z = 21

x = 2, y = 7, z = 24

x = 7, y = 18, z = 20

Число 40851 может быть представлено следующими комбинациями:

x = 1, y = 17, z = 33

x = 3, y = 24, z = 30

x = 6, y = 11, z = 34

Число 46593 может быть представлено следующими комбинациями:

x = 1, y = 24, z = 32

x = 2, y = 22, z = 33

x = 7, y = 15, z = 35

Число 41966 может быть представлено следующими комбинациями:

x = 1, y = 26, z = 29

x = 2, y = 23, z = 31

x = 11, y = 11, z = 34

Число 39339 может быть представлено следующими комбинациями:

x = 2, y = 3, z = 34

x = 3, y = 15, z = 33

x = 18, y = 24, z = 27

Число 39376 может быть представлено следующими комбинациями:

x = 2, y = 4, z = 34

x = 4, y = 15, z = 33

x = 12, y = 22, z = 30

Число 36288 может быть представлено следующими комбинациями:

x = 2, y = 7, z = 33

x = 3, y = 21, z = 30

x = 8, y = 24, z = 28

Число 30528 может быть представлено следующими комбинациями:

x = 2, y = 9, z = 31

x = 11, y = 13, z = 30

x = 14, y = 18, z = 28

Число 40041 может быть представлено следующими комбинациями:

x = 2, y = 9, z = 34

x = 2, y = 16, z = 33

x = 3, y = 25, z = 29

x = 9, y = 15, z = 33

Число 20691 может быть представлено следующими комбинациями:

x = 2, y = 10, z = 27

x = 2, y = 19, z = 24

x = 18, y = 19, z = 20

Число 41040 может быть представлено следующими комбинациями:

x = 2, y = 12, z = 34

x = 6, y = 24, z = 30

x = 12, y = 15, z = 33

Число 21888 может быть представлено следующими комбинациями:

x = 2, y = 13, z = 27

x = 4, y = 20, z = 24

x = 6, y = 16, z = 26

Число 42056 может быть представлено следующими комбинациями:

x = 2, y = 14, z = 34

x = 3, y = 21, z = 32

x = 14, y = 15, z = 33

Число 42687 может быть представлено следующими комбинациями:

x = 2, y = 15, z = 34

x = 9, y = 23, z = 31

x = 15, y = 15, z = 33

Число 12104 может быть представлено следующими комбинациями:

x = 2, y = 16, z = 20

x = 5, y = 11, z = 22

x = 9, y = 15, z = 20

Число 17928 может быть представлено следующими комбинациями:

x = 2, y = 16, z = 24

x = 9, y = 15, z = 24

x = 16, y = 18, z = 20

Число 43408 может быть представлено следующими комбинациями:

x = 2, y = 16, z = 34

x = 9, y = 15, z = 34

x = 15, y = 16, z = 33

Число 45144 может быть представлено следующими комбинациями:

x = 2, y = 18, z = 34

x = 12, y = 22, z = 32

x = 15, y = 18, z = 33

Число 19034 может быть представлено следующими комбинациями:

x = 2, y = 19, z = 23

x = 8, y = 21, z = 21

x = 9, y = 9, z = 26

Число 31221 может быть представлено следующими комбинациями:

x = 2, y = 21, z = 28

x = 5, y = 16, z = 30

x = 10, y = 18, z = 29

Число 24480 может быть представлено следующими комбинациями:

x = 2, y = 22, z = 24

x = 3, y = 4, z = 29

x = 18, y = 20, z = 22

Число 29457 может быть представлено следующими комбинациями:

x = 2, y = 24, z = 25

x = 9, y = 12, z = 30

x = 18, y = 20, z = 25

Число 40832 может быть представлено следующими комбинациями:

x = 2, y = 24, z = 30

x = 4, y = 20, z = 32

x = 18, y = 20, z = 30

Число 12221 может быть представлено следующими комбинациями:

x = 3, y = 3, z = 23

x = 5, y = 16, z = 20

x = 6, y = 14, z = 21

Число 46710 может быть представлено следующими комбинациями:

x = 3, y = 3, z = 36

x = 3, y = 27, z = 30

x = 5, y = 22, z = 33

Число 46747 может быть представлено следующими комбинациями:

x = 3, y = 4, z = 36

x = 4, y = 27, z = 30

x = 11, y = 25, z = 31

Число 46808 может быть представлено следующими комбинациями:

x = 3, y = 5, z = 36

x = 5, y = 27, z = 30

x = 6, y = 24, z = 32

Число 40060 может быть представлено следующими комбинациями:

x = 3, y = 9, z = 34

x = 3, y = 16, z = 33

x = 19, y = 25, z = 26

Число 47683 может быть представлено следующими комбинациями:

x = 3, y = 10, z = 36

x = 10, y = 27, z = 30

x = 19, y = 24, z = 30

Число 48014 может быть представлено следующими комбинациями:

x = 3, y = 11, z = 36

x = 11, y = 27, z = 30

x = 20, y = 25, z = 29

Число 40097 может быть представлено следующими комбинациями:

x = 4, y = 9, z = 34

x = 4, y = 16, z = 33

x = 22, y = 24, z = 25

Число 28792 может быть представлено следующими комбинациями:

x = 4, y = 12, z = 30

x = 10, y = 23, z = 25

x = 14, y = 16, z = 28

Число 48168 может быть представлено следующими комбинациями:

x = 4, y = 23, z = 33

x = 8, y = 10, z = 36

x = 16, y = 27, z = 29

Число 35216 может быть представлено следующими комбинациями:

x = 4, y = 26, z = 26

x = 6, y = 20, z = 30

x = 8, y = 17, z = 31

Число 38259 может быть представлено следующими комбинациями:

x = 5, y = 13, z = 33

x = 10, y = 26, z = 27

x = 19, y = 24, z = 26

Число 44946 может быть представлено следующими комбинациями:

x = 7, y = 12, z = 35

x = 9, y = 17, z = 34

x = 11, y = 24, z = 31

x = 16, y = 17, z = 33

Число 41364 может быть представлено следующими комбинациями:

x = 9, y = 11, z = 34

x = 11, y = 16, z = 33

x = 13, y = 23, z = 30

Число 32850 может быть представлено следующими комбинациями:

x = 10, y = 23, z = 27

x = 11, y = 12, z = 31

x = 19, y = 23, z = 24

Число 48313 может быть представлено следующими комбинациями:

x = 12, y = 22, z = 33

x = 16, y = 17, z = 34

x = 21, y = 21, z = 31

5. Дан текстовый файл со словами, разделенными пробелами. Написать программу, которая выводит 10 наиболее встречаемых слов в файле, отсортированных в порядке убывания частоты встречаемости. При равном числе вхождений слова упорядочивать по алфавиту.

Код:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

class Program

{

    static void Main()

    {

        Console.WriteLine("Введите полный путь к файлу:");

        string filePath = Console.ReadLine();

        if (File.Exists(filePath))

        {

            string fileContent = File.ReadAllText(filePath);

            string[] words = fileContent.Split(new[] { ' ', '\n', '\r', '\t' }, StringSplitOptions.RemoveEmptyEntries);

            // Словарь для подсчета частоты слов

            Dictionary<string, int> wordCounts = new Dictionary<string, int>(StringComparer.OrdinalIgnoreCase);

            foreach (var word in words)

            {

                string cleanedWord = word.Trim().ToLower();

                if (wordCounts.ContainsKey(cleanedWord))

                {

                    wordCounts[cleanedWord]++;

                }

                else

                {

                    wordCounts[cleanedWord] = 1;

                }

            }

            // Сортировка по частоте и по алфавиту в случае равенства

            var sortedWords = wordCounts.OrderByDescending(kvp => kvp.Value)

                                        .ThenBy(kvp => kvp.Key)

                                        .Take(10);

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

            foreach (var kvp in sortedWords)

            {

                Console.WriteLine($"{kvp.Key}: {kvp.Value}");

            }

        }

        else

        {

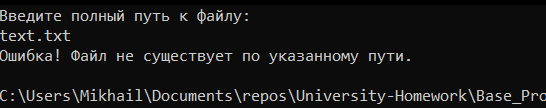
            Console.WriteLine("Ошибка! Файл не существует по указанному пути.");

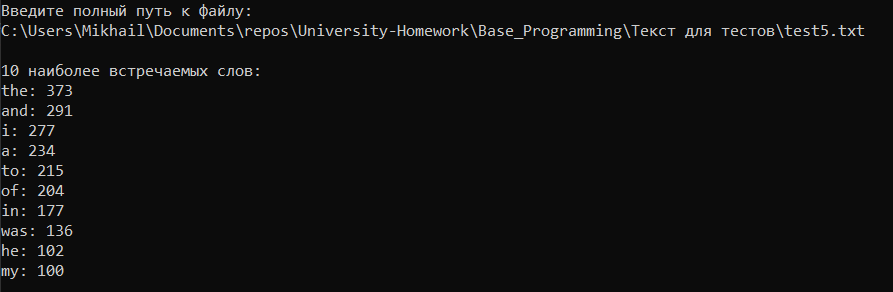
        }

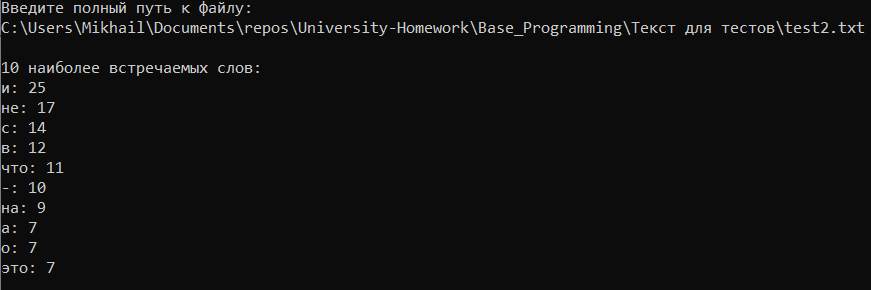
    }

}

Тестирование:







6. Представить турнирную сетку раунда плей-офф чемпионата мира в виде дерева. Названия команд считать из файла либо «прошить» в коде. Сначала корень дерева и все узлы, кроме листьев, заполнить пустыми строками и неизвестными результатами. Листья соответствуют матчам этапа 1/16 финала. Результаты матчей генерировать произвольным образом, продвигая команд-победителей от листьев к корню. Вывести результаты розыгрыша в таком виде:

BRA - ARG : 0 - 2

BRA - FRA : 2 - 1

BRA - COL : 4 - 3

BRA - CHI : 2 - 1

COL - URU : 3 - 2

FRA - GER : 1 - 0

FRA - NIG : 1 - 0

GER - ALG : 3 - 1

CRC - ARG : 0 - 2

MEX - CRC : 0 - 1

NED - MEX : 1 - 2

CRC - GRE : 2 - 1

ARG - BEL : 1 - 0

ARG - SWI : 3 - 2

BEL - USA : 3 – 2

Код:

using System;

using System.Collections.Generic;

public class TournamentSimulator

{

    static Random rng = new Random();

    class Game

    {

        public string Player1;

        public string Player2;

        public int Score1;

        public int Score2;

        public Game LeftMatch;

        public Game RightMatch;

        public string Winner => Score1 > Score2 ? Player1 : Player2;

    }

    static Game SimulateGame(string player1, string player2)

    {

        int score1 = rng.Next(0, 6);

        int score2 = rng.Next(0, 6);

        while (score1 == score2)

        {

            score1 = rng.Next(0, 6);

            score2 = rng.Next(0, 6);

        }

        return new Game { Player1 = player1, Player2 = player2, Score1 = score1, Score2 = score2 };

    }

    static Game SimulateRound(List<Game> previousGames)

    {

        var upcomingMatches = new List<Game>();

        for (int i = 0; i < previousGames.Count; i += 2)

        {

            var left = previousGames[i];

            var right = previousGames[i + 1];

            var game = SimulateGame(left.Winner, right.Winner);

            game.LeftMatch = left;

            game.RightMatch = right;

            upcomingMatches.Add(game);

        }

        return upcomingMatches.Count == 1 ? upcomingMatches[0] : SimulateRound(upcomingMatches);

    }

    static void DisplayTournamentTree(Game game, int level = 0)

    {

        if (game == null) return;

        string indent = new string(' ', level \* 4);

        Console.WriteLine($"{indent}{game.Player1} vs {game.Player2} : {game.Score1} - {game.Score2}");

        DisplayTournamentTree(game.LeftMatch, level + 1);

        DisplayTournamentTree(game.RightMatch, level + 1);

    }

    public static void Main()

    {

        var competitors = new List<string>

        {

            "ААА", "БББ", "ВВВ", "ГГГ",

            "ДДД", "ЕЕЕ", "ЁЁЁ", "ЖЖЖ",

            "ИИИ", "ЙЙЙ", "ККК", "ЛЛЛ",

            "МММ", "ННН", "ООО", "ППП"

        };

        var firstStageMatches = new List<Game>();

        for (int i = 0; i < competitors.Count; i += 2)

        {

            firstStageMatches.Add(SimulateGame(competitors[i], competitors[i + 1]));

        }

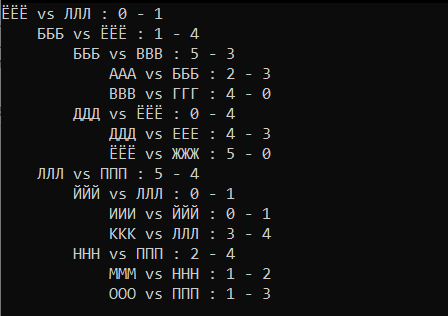
        var finalMatch = SimulateRound(firstStageMatches);

        DisplayTournamentTree(finalMatch);

    }

}

Тестирование:



7. Написать программу, в соответствии с вариантом. Решить задачу как на основе самостоятельно разработанного списка, так и на основе коллекции .NET.

**Вариант 8**. Написать программу, которая оставляет в первом списке только те элементы, которые содержатся и в первом, и во втором списках

**Самостоятельно разработанный список**

Код:

using System;

class Program

{

    static void Main()

    {

        Random random = new Random();

        MyLinkedList list1 = new MyLinkedList();

        MyLinkedList list2 = new MyLinkedList();

        // список 1

        for (int i = 0; i < 10; i++)

        {

            list1.Add(random.Next(1, 21));  // Числа от 1 до 20

        }

        // список 2

        for (int i = 0; i < 8; i++)

        {

            list2.Add(random.Next(1, 21));

        }

        // Выводим исходные списки

        Console.WriteLine("Первый список:");

        list1.Print();

        Console.WriteLine("Второй список:");

        list2.Print();

        // Оставляем в list1 только те элементы, которые есть и во втором

        list1.FilterCommonElements(list2);

        Console.WriteLine("Общие элементы:");

        list1.Print();

    }

}

public class MyLinkedList

{

    private Node head;

    public void Add(int value)

    {

        Node newNode = new Node(value);

        if (head == null)

        {

            head = newNode;

        }

        else

        {

            Node current = head;

            while (current.Next != null)

            {

                current = current.Next;

            }

            current.Next = newNode;

        }

    }

    public void FilterCommonElements(MyLinkedList otherList)

    {

        Node current = head;

        while (current != null)

        {

            if (!otherList.Contains(current.Value))

            {

                Remove(current.Value);

            }

            current = current.Next;

        }

    }

    private void Remove(int value)

    {

        if (head == null) return;

        if (head.Value == value)

        {

            head = head.Next;

            return;

        }

        Node current = head;

        while (current.Next != null && current.Next.Value != value)

        {

            current = current.Next;

        }

        if (current.Next != null)

        {

            current.Next = current.Next.Next;

        }

    }

    // Проверка, содержится ли элемент в списке

    private bool Contains(int value)

    {

        Node current = head;

        while (current != null)

        {

            if (current.Value == value)

            {

                return true;

            }

            current = current.Next;

        }

        return false;

    }

    public void Print()

    {

        Node current = head;

        while (current != null)

        {

            Console.Write(current.Value + " ");

            current = current.Next;

        }

        Console.WriteLine();

    }

    // Класс узла связного списка

    private class Node

    {

        public int Value;

        public Node Next;

        public Node(int value)

        {

            Value = value;

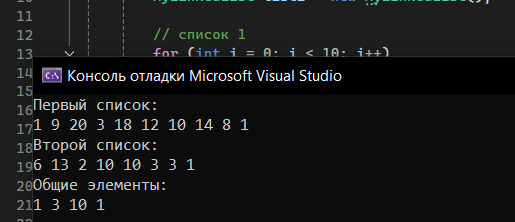
            Next = null;

        }

    }

}

Тестирование:

****

**На основе коллекции .NET**

using System;

using System.Collections.Generic;

class Program

{

    static void Main()

    {

        Random random = new Random();

        HashSet<int> set1 = new HashSet<int>();

        HashSet<int> set2 = new HashSet<int>();

        for (int i = 0; i < 16; i++)

        {

            set1.Add(random.Next(1, 21));

        }

        for (int i = 0; i < 9; i++)

        {

            set2.Add(random.Next(1, 21));

        }

        Console.WriteLine("Первый список:");

        PrintHashSet(set1);

        Console.WriteLine("Второй список:");

        PrintHashSet(set2);

        set1.IntersectWith(set2);

        Console.WriteLine("Общие элементы:");

        PrintHashSet(set1);

    }

    static void PrintHashSet(HashSet<int> set)

    {

        foreach (var item in set)

        {

            Console.Write(item + " ");

        }

        Console.WriteLine();

    }

}

Тестирование:

