

Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Отчет по лабораторной работе №3
по дисциплине «Объектно-Оrientированное Программирование»
на тему «Файлы данных»

Выполнил:
студент группы БИВТ-24-5

Черных Богдан

Проверил:
Стучилин В. В.

Москва, 2025

1 ЦЕЛЬ РАБОТЫ:

.....

.....

Цель данной лабораторной работы заключается в модификации исходного кода программы для выполнения различных задач (соревнования по прыжкам в длину, кросс, шахматы, хоккей, фигурное катание, лыжные гонки, футбол и т.д.) с использованием ввода исходных данных из файлов и формирования файлов с результатами. Это позволяет автоматизировать обработку данных, усовершенствовать навыки работы с файловой системой, коллекциями и методами LINQ в C#, а также продемонстрировать применение принципов модульного программирования в реальных задачах.

2. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ.

-
-
12. Считая, что в памяти компьютера хранится таблица кодов часто встречающихся слов, ввести текст в массив, заменяя слова кодами после ввода. Распечатать текст в исходном виде, т.е. заменяя коды словами.
13. Определить долю в процентах слов, начинающихся на различные буквы. Выписать эти буквы и доли начинающихся на них слов.
14. Текст содержит слова и целые числа от 1 до 10. Найти сумму включенных в текст чисел.

3. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

.....

.....

Теоретическое введение. *Файл данных* — это совокупность (последовательность) компонент, имеющая имя, расположенная на внешнем носителе. Файлы могут быть объединены в каталоги (директории, папки), также имеющие имя. Использование файлов данных позволяет хранить данные на внешнем носителе, обрабатывая при необходимости порциями (например, при больших объемах данных), позволяет многократно использовать один и тот же набор данных (например, при отладке), позволяет использовать результаты выполнения одной программы (формируя из них файл) как входные данные при выполнении другой программы и т. п.

Файлы и потоки

Для существующего файла данных, который хранится на внешнем носителе и имеет имя, при обращении к файлу создается поток с целью сохранения данных в резервном

хранилище. Резервное хранилище — это устройство хранения информации, например, диск.

Поток — это абстракция последовательности байтов, например файл или другое устройство, предоставляющее данные. Класс `Stream` (поток) и его производные классы предоставляют универсальное представление различных типов ввода и вывода, избавляя программиста от необходимости знания отдельных сведений операционной системы и базовых устройств.

Потоки включают три основные операции:

1. Чтение из потока — это перенос информации из потока в структуру данных, такую как массив байтов.
2. Запись в поток — это передача данных из структуры данных в поток.
3. Потоки также могут поддерживать поиск.

Программы, составленные на языке C#, работают с каталогами, файлами и потоками при помощи специально предназначенных для этого классов, входящих в состав библиотеки классов Microsoft .NET Framework и содержащихся в пространстве имен `System.IO`, которое необходимо подключить, чтобы обеспечить доступ к классам, определенным для потоков ввода-вывода (см. пример 9.1).

Для работы с папками и файлами предназначены следующие основные классы:

[Directory](#) предоставляет статические методы операций создания, перемещения и перечисления в директориях и поддиректориях. Класс `DirectoryInfo` предоставляет методы экземпляра.

[File](#) предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов, а также помогает при создании объектов [FileStream](#). Класс `FileInfo` предоставляет методы экземпляра.

Для работы с потоками предназначены следующие основные классы:

[FileStream](#) предоставляет [поток](#) в файле, поддерживая операции чтения и записи.

Класс [StreamReader](#) считывает символы из потоков с учетом кодировки, класс [StreamWriter](#) записывает символы в потоки, используя кодировку для преобразования символов в байты (см. пример 9.1).

Стандартные потоки связаны, как правило, с консолью и клавиатурой. Для вывода данных в стандартный поток вывода и для ввода из стандартного потока ввода используются методы класса `Console`: `Console.ReadLine`, `Console.Write` и `Console.WriteLine`. Эти методы и использовались до сих пор во всех примерах программ.

Количество классов, предназначенных для работы с файлами, достаточно велико. Здесь будут рассмотрены только классы, предназначенные для чтения из текстового файла или записи в текстовый файл: `StreamReader` и `StreamWriter`.

Для ввода из файла (созданного заранее в текстовом редакторе) необходимо вначале открыть поток класса `StreamReader`, связав его с файлом. В приведенном ниже примере файл, из которого предполагается считывать данные, расположен по адресу `C:\st\Koord.txt` (это полный путь к файлу). Открытие потока и его привязка к файлу осуществляются с помощью конструктора (возможны и другие способы, которые здесь не рассматриваются).

```
StreamReader sr = new StreamReader(path);
```

Здесь `sr` – экземпляр класса `StreamReader`, а аргумент `path` передает конструктору строку, содержащую полный путь к файлу (в качестве аргумента можно использовать и константу, содержащую полный адрес файла). Далее строки из файла (в программе это поток `sr`) по очереди считываются в переменную `line`, из которой далее как обычно извлекаются отдельные значения.

После окончания работы с объявленным потоком, его следует закрыть методом `Close`:

```
sr.Close();
```

Пример 9.1. Координаты произвольного количества точек на плоскости размещены в файле `Koord.txt` на диске `C` в папке (директории) `st` по два числа (значения `x` и `y`) в строке. В первой строке файла размещено одно число — радиус окружности `r`. Требуется определить, сколько точек попадет в круг радиуса `r`.

```
using System;

using System.IO;

class Program
{
    static void Main()
    {
        string path = "C:\\st\\Koord.txt";

        StreamReader sr = new StreamReader(path);

        int n = 0;

        string line;
```

```

        line = sr.ReadLine();

        int r = int.Parse(line);

        Console.WriteLine("Радиус {0}", r);

        while ((line = sr.ReadLine()) != null)
        {

            string[] koord = line.Split(' ');

            int x = int.Parse(koord[0]);

            int y = int.Parse(koord[1]);

            Console.WriteLine("Координаты точек x = {0} y = {1}", x,
y);

            if (x * x + y * y < r * r) n = n + 1;

        }

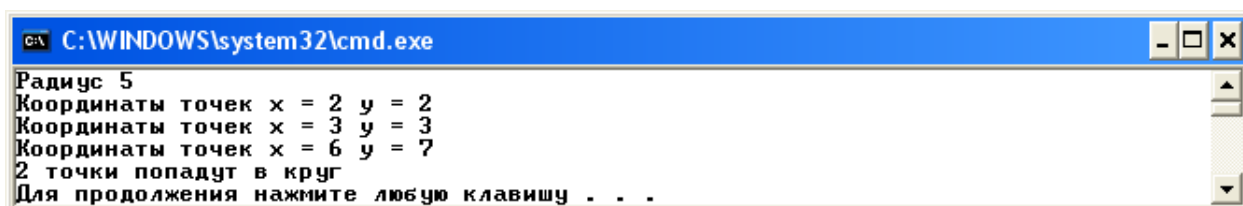
        sr.Close();

        Console.WriteLine("{0} точки попадут в круг ", n);

    }

}

```



```

C:\WINDOWS\system32\cmd.exe
Радиус 5
Координаты точек x = 2 y = 2
Координаты точек x = 3 y = 3
Координаты точек x = 6 y = 7
2 точки попадут в круг
Для продолжения нажмите любую клавишу . . .

```

З а м е ч а н и е . В программе в адресе файла вместо одной наклонной черты используется две. Одна наклонная черта в строке могла бы восприниматься как первый символ управляющей последовательности. Использование двух наклонных позволяет избежать этой двусмысленности. Теперь наклонная черта будет восприниматься как символ строки, а не как управляющая последовательность (см. Часть 1). В С# предусмотрен также способ объявления строки, в которой все символы между кавычками трактуются как часть строки. Это специальное объявление – буквальная строка – задается указанием символа @ перед всей строкой и обычно используется для задания пути к файлу. С использованием этого объявления задание строки `path` может выглядеть так

```
string path = @"C:\st\Koord.txt";
```

При выводе в файл необходимо выполнить аналогичные действия: открыть поток класса `StreamWriter`, задав имя потока и связав его с файлом, предназначенным для размещения выводимых результатов, вывести в этот поток (т.е. в указанный файл) необходимые результаты и закрыть поток оператором `Close()`.

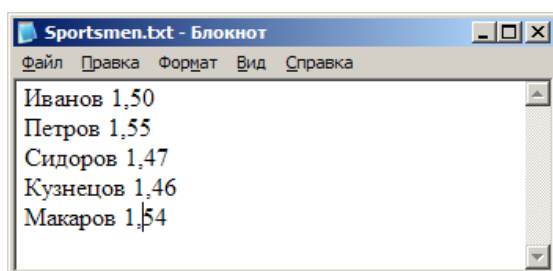
Если в примере 9.1 предполагается также вывод результата в файл, то необходимо добавить следующие инструкции

```
string path1 = "c:\\st\\Koord1.txt";  
  
StreamWriter sw = new StreamWriter(path1);  
  
sw.WriteLine(n);  
  
sw.Close();
```

Далее рассмотрим пример, в котором осуществляется ввод из файла исходных данных и вывод результатов в файл с учетом регионального стандарта (см. Приложение 2).

Пример. 9.2. Протокол соревнований по прыжкам в высоту содержит список фамилий и результатов (одна попытка) в порядке стартовых номеров. Получить итоговую таблицу, содержащую фамилии и результаты в порядке занятых мест. Количество спортсменов не более 30. Для размещения исходных данных используется массив структур. Структура содержит информацию – фамилия и результат спортсмена. Ввод данных осуществлять из заранее подготовленного файла, вывод итоговой таблицы осуществлять в файл. (В примере 6.1 эта же задача решена без использования файлов данных.)

Исходный файл



```
using System;  
  
using System.Text;  
  
using System.IO;  
  
namespace ConsoleApplication1
```

```

{

    struct Sportsmen

    {

        public string famile;

        public double rez;

    }

    class Program

    {

        static void Main(string[] args)

        {

            Sportsmen[] sp = new Sportsmen[5];

            string line;

            string path = "c:\\st\\Sportsmen.txt";

            //кодovая страница операционной системы Windows для

            //кириллицы имеет идентификатор 1251

            StreamReader sr = new

StreamReader(path, Encoding.GetEncoding(1251));

            int i = 0;

            while ((line = sr.ReadLine()) != null)

            {

                string[] sports = line.Split(' ');

                sp[i].famile = sports[0];

                sp[i].rez = double.Parse(sports[1]);

                Console.WriteLine("Фамилия {0}\\t Результат {1:f2}",

sp[i].famile, sp[i].rez);

                i++;

            }

            sr.Close();

```

```

for (i = 0; i < sp.Length - 1; i++)
{
    double amax = sp[i].rez;
    int imax = i;
    for (int j = i + 1; j < sp.Length; j++)
    {
        if (sp[j].rez > amax)
        {
            amax = sp[j].rez;
            imax = j;
        }
    }
    Sportsmen temp;
    temp = sp[imax];
    sp[imax] = sp[i];
    sp[i] = temp;
}

Console.WriteLine();

for (i = 0; i < sp.Length; i++)
{
    Console.WriteLine("Фамилия      {0}\t  Результат
{1:f2}", sp[i].famile, sp[i].rez);
}

string path1 = "c:\\st\\Sportsmen1.txt";
StreamWriter sw = new StreamWriter(path1);
for (i = 0; i < sp.Length; i++)
{
    sw.WriteLine("{0} {1:f2}", sp[i].famile, sp[i].rez);
}

```



```

    }

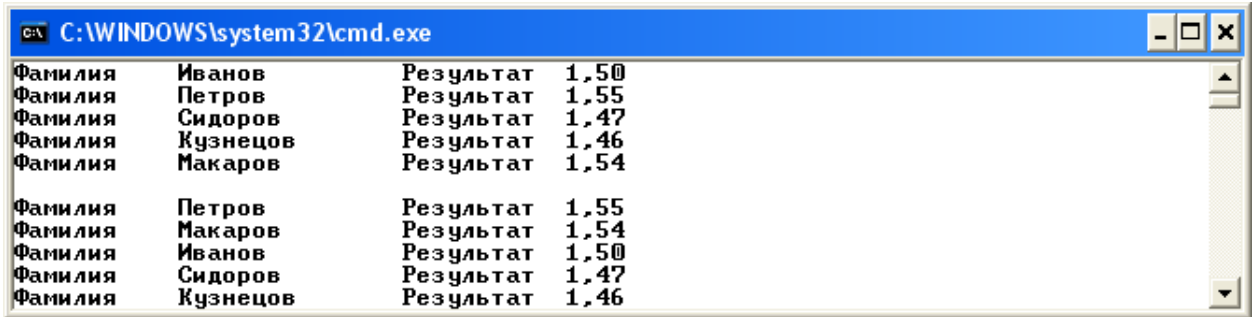
    sw.Close();

}

}

}

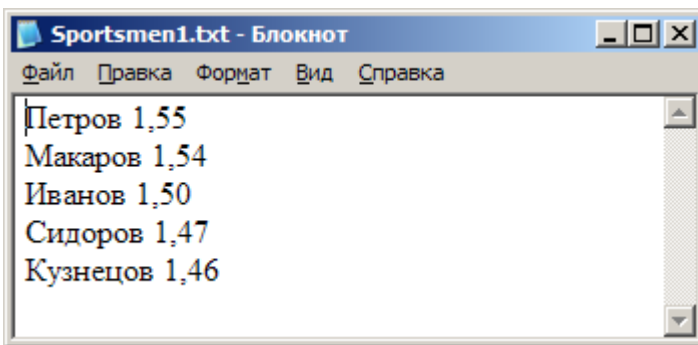
```



The screenshot shows a Windows command prompt window with the title "C:\WINDOWS\system32\cmd.exe". The window contains a table with three columns: "Фамилия" (Surname), "Имя" (Name), and "Результат" (Result). The data is as follows:

Фамилия	Имя	Результат
Иванов	Петров	1,50
Петров	Сидоров	1,55
Сидоров	Кузнецов	1,47
Кузнецов	Макаров	1,46
Макаров	Петров	1,54
Петров	Макаров	1,55
Макаров	Иванов	1,54
Иванов	Сидоров	1,50
Сидоров	Кузнецов	1,47
Кузнецов	Макаров	1,46

Файл с результатами



The screenshot shows a Notepad window titled "Sportsmen1.txt - Блокнот". The window contains the following text:

```

Петров 1,55
Макаров 1,54
Иванов 1,50
Сидоров 1,47
Кузнецов 1,46

```

5. ЛИСТИНГ ПРОГРАММЫ

```

.....

//svg does precious
using System; // Аналог <iostream> для работы с консолью основными функциями
using System.Collections.Generic; // Аналог <vector>, <list>, <map>, <set>,<
unordered_map >, < unordered_set >, < stack >, < queue >
using System.Text; // Аналог <string>, <cstring> (для работы со строками и
StringBuilder)
using System.Linq; // Аналог <algorithm> (для работы с LINQ, сортировок, поиска и
т.д.)
using System.IO; // Аналог <cstdio>, <fstream> (работа с файлами)
using System.Globalization; // Аналог <iomanip> (для форматирования)
using System.Collections; // Работа с различными коллекциями (например, ArrayList)
using System.Threading; // Поток и многопоточность
using System.Runtime.Serialization; // Аналог <std::exception> (работа с исключениями)
using System.Reflection; // Аналог <typeinfo> (информация о типах, рефлексия)
using System.Diagnostics; // Аналог <utility>, <std::pair> (вспомогательные
функции и классы)
using System.ComponentModel; // Дополнительные утилиты и атрибуты

```

```

using System.Numerics; // Работа с большими числами и математическими операциями
using System.Globalization;
using System.Diagnostics;
using System.Net;
using System.Numerics;
// Для работы с потоками данных:
using System.Threading.Tasks; // Асинхронные задачи
// Для работы с датами и временем:
using System.Timers; // Для работы с таймерами и временем
using System.Collections.Generic;
using System.Text;
using System.Linq;
using System.IO; //important
using System.Globalization;
using System.Collections;
using System.Threading;
using System.Runtime.Serialization;
using System.Reflection;
using System.Diagnostics;
using System.ComponentModel;
using System.Numerics;
using System.Globalization;
using System.Diagnostics;
using System.Net;
using System.Numerics;
using System.Threading.Tasks;
using System.Security.Cryptography;
using System.Data;
using System.Data.SqlClient;
using System.Xml;
using System.Xml.Linq;
using System.Runtime.InteropServices;
using System.Security;
using System.Web;
using System.Media;
using System.Drawing;
using System.Configuration;
using System.Timers;
using System.Runtime.Remoting;
using System.Runtime.CompilerServices;
using System.Runtime.Versioning;
using System.CodeDom;
using System.CodeDom.Compiler;
using System.Collections.Concurrent;
using System.Runtime;
using System.Windows;
using System.Windows.Input;
using System.Security.Principal;
using System.Security.Permissions;
using System.Resources;
using C = System.Console; //console
using dl = System.Decimal; //decimal
using str = System.String; //string
using l = System.Int64; //long
using u = System.UInt64; //Ulong
using db = System.Double; //Double
/*
Izzspot - 19 years
Boogie B - 20 years
SJ - 21 years
Bandokay - life sentence
Youngest in the charge
OFB, we don't window shop
Bro caught him an opp and tried turn him off (Bow, bow)
In this X3, man's swervin' off (Skrr, skrr)
Free Boogie Bando, he got birded off (Free my bro, free my bro)
Whenever we get a burner loss

```

```

We just cop a next one and go burst it off (Ay)
Lil bro's tellin' me he got his earnings wrong
We just took him OT, now his trapline's gone (Ring, ring)
Hashtag
Bro backed this ting and just started squeezin' (Clarted)
When it broad day, it was freezin'
Hashtag fuckery, hashtag screamin'
One on the hand ting woi, left man leanin', leanin' (Fucker)
Show us cause it's good to feel it
Shortie's cooze and she must be dreamin'
Vogue
Ra-ra-racks came in, hello (Hello?)
Fu-fucked this girl on Vogue
Big Range, ain't no Evoque
Big chain, this ain't a choker
Sh-sh-she like when her neck get choked
I know that her boyfriend knows
You tell white lies like cocaine
I know that her boyfriend knows
You tell white- (Cocaine)
She hate when her man get home
*/
/* ;Adelante Barcelona, adelante Cataluña! Visca el Barça! Visca Catalunya!
;Al diablo con todos los demás, porque lo más importante en la vida es el
fútbol!
*/
//-----
-----
//-----
-----
//-----
-----

```

```

// Лабораторная Работа №9
// Модифицировано для считывания данных из файлов и записи результатов в файлы

```

```

// 1 Уровень
// Задание 1: Результаты соревнований по прыжкам в длину
class LongJumpParticipantCustom
{
    public string LastName { get; set; }
    public string Club { get; set; }
    public double JumpAttempt1 { get; set; }
    public double JumpAttempt2 { get; set; }
    public double TotalJumpScore => JumpAttempt1 + JumpAttempt2;

    public LongJumpParticipantCustom(string lastName, string club, double
attempt1, double attempt2)
    {
        LastName = lastName;
        Club = club;
        JumpAttempt1 = attempt1;
        JumpAttempt2 = attempt2;
    }
}

class LongJumpCompetitionCustom
{
    public List<LongJumpParticipantCustom> Participants { get; set; } = new();

    public void AddParticipant(LongJumpParticipantCustom participant)
    {
        Participants.Add(participant);
    }

    public void DisplayResults()

```

```

    {
        var sortedParticipants = Participants.OrderByDescending(p =>
p.TotalJumpScore).ToList();
        C.WriteLine("Результаты соревнований по прыжкам в длину:");
        C.WriteLine("Место\tФамилия\t\tОбщество\tПопытка 1\tПопытка 2\tСумма");
        for (int i = 0; i < sortedParticipants.Count; i++)
        {
            var p = sortedParticipants[i];
            C.WriteLine($"{i +
1}\t{p.LastName}\t\t{p.Club}\t\t{p.JumpAttempt1:F2}\t\t{p.JumpAttempt2:F2}\t\t{p.T
otalJumpScore:F2}");
        }
    }
}

// Задание 2: Результаты кросса на 500 м для женщин
class CrossCountryParticipantCustom
{
    public string LastName { get; set; }
    public string Group { get; set; }
    public string Coach { get; set; }
    public double Result { get; set; }
    public bool NormativeMet { get; set; }

    public CrossCountryParticipantCustom(string lastName, string group, string
coach, double result, double normative)
    {
        LastName = lastName;
        Group = group;
        Coach = coach;
        Result = result;
        NormativeMet = result <= normative;
    }
}

class CrossCountryCompetitionCustom
{
    public List<CrossCountryParticipantCustom> Participants { get; set; } = new();

    public void AddParticipant(CrossCountryParticipantCustom participant)
    {
        Participants.Add(participant);
    }

    public void DisplayResults(double normative)
    {
        var sortedParticipants = Participants.OrderBy(p => p.Result).ToList();
        int normativeCount = sortedParticipants.Count(p => p.NormativeMet);

        C.WriteLine("Результаты кросса на 500 м для женщин:");
        C.WriteLine("Фамилия\t\tГруппа\t\tПреподаватель\tРезультат\tНорматив");
        foreach (var p in sortedParticipants)
        {
            C.WriteLine($"{p.LastName}\t\t{p.Group}\t{p.Coach}\t\t{p.Result:F1}\t\t{ (p.Normati
veMet ? "Выполнен" : "Не выполнен")}");
        }
        C.WriteLine($"Всего участниц, выполнивших норматив: {normativeCount}");
    }
}

// Задание 3: Опрос радиокompании «Человек года»
class VoteCustom
{
    public string Candidate { get; set; }
    public int VoteCount { get; set; }
}

```

```

        public VoteCustom(string candidate, int count)
        {
            Candidate = candidate;
            VoteCount = count;
        }
    }

class SurveyCustom
{
    public List<string> Votes { get; set; } = new();

    public void AddVote(string candidate)
    {
        Votes.Add(candidate);
    }

    public void DisplayTopCandidates(int topN)
    {
        var voteCounts = Votes.GroupBy(v => v)
                                .Select(g => new VoteCustom(g.Key, g.Count()))
                                .OrderByDescending(v => v.VoteCount)
                                .Take(topN)
                                .ToList();

        int totalVotes = Votes.Count;
        C.WriteLine("Результаты опроса 'Человек года':");
        C.WriteLine("Кандидат\tГолосов\tДоля (%)");
        foreach (var v in voteCounts)
        {
            double percentage = (double)v.VoteCount / totalVotes * 100;
            C.WriteLine($"{v.Candidate}\t\t{v.VoteCount}\t\t{percentage:F2}");
        }
    }
}

// 2 Уровень

// Задание 7: Турнирная таблица по шахматам
class ChessPlayerCustom
{
    public string LastName { get; set; }
    public double Points { get; set; }

    public ChessPlayerCustom(string lastName)
    {
        LastName = lastName;
        Points = 0.0;
    }

    public void AddResult(double result)
    {
        Points += result;
    }
}

class ChessTournamentCustom
{
    public List<ChessPlayerCustom> Players { get; set; } = new();

    public void AddPlayer(ChessPlayerCustom player)
    {
        Players.Add(player);
    }

    public void AddGameResult(string player1LastName, string player2LastName,
double player1Result, double player2Result)
    {

```

```

        var player1 = Players.FirstOrDefault(p => p.LastName == player1LastName);
        var player2 = Players.FirstOrDefault(p => p.LastName == player2LastName);

        if (player1 != null && player2 != null)
        {
            player1.AddResult(player1Result);
            player2.AddResult(player2Result);
        }
    }

    public void DisplayResults()
    {
        var sortedPlayers = Players.OrderByDescending(p => p.Points).ToList();
        C.WriteLine("Турнирная таблица по шахматам:");
        C.WriteLine("Место\tФамилия\t\tОчки");
        for (int i = 0; i < sortedPlayers.Count; i++)
        {
            var player = sortedPlayers[i];
            C.WriteLine($"{i + 1}\t{player.LastName}\t\t{player.Points:F2}");
        }
    }
}

// Задание 8: Сборная по хоккею
class HockeyPlayerCustom
{
    public string LastName { get; set; }
    public List<int> PenaltyTimes { get; set; } = new();

    public HockeyPlayerCustom(string lastName)
    {
        LastName = lastName;
    }

    public void AddPenaltyTime(int penaltyTime)
    {
        PenaltyTimes.Add(penaltyTime);
    }

    public int TotalPenaltyTime => PenaltyTimes.Sum();
}

class HockeyTeamSelectionCustom
{
    public List<HockeyPlayerCustom> Players { get; set; } = new();

    public void AddPlayer(HockeyPlayerCustom player)
    {
        Players.Add(player);
    }

    public void DisplaySelectedPlayers()
    {
        var eligiblePlayers = Players.Where(p => p.TotalPenaltyTime <
10).OrderBy(p => p.TotalPenaltyTime).ToList();
        C.WriteLine("Список кандидатов в сборную по хоккею:");
        C.WriteLine("Фамилия\t\tШтрафное время");
        foreach (var player in eligiblePlayers)
        {
            C.WriteLine($"{player.LastName}\t\t{player.TotalPenaltyTime} мин.");
        }
    }
}

// Задание 9: Результаты соревнований фигуристов
class FigureSkaterCustom
{

```

```

        public string LastName { get; set; }
        public List<double> JudgeRanks { get; set; } = new();

        public FigureSkaterCustom(string lastName)
        {
            LastName = lastName;
        }

        public void AddJudgeRank(double rank)
        {
            JudgeRanks.Add(rank);
        }

        public double TotalPlace => JudgeRanks.Sum();
    }

    class FigureSkatingCompetitionCustom
    {
        public List<FigureSkaterCustom> Skaters { get; set; } = new();

        public void AddSkater(FigureSkaterCustom skater)
        {
            Skaters.Add(skater);
        }

        public void DisplayResults()
        {
            var sortedSkaters = Skaters.OrderBy(s => s.TotalPlace).ToList();
            C.WriteLine("Результаты соревнований фигуристов:");
            C.WriteLine("Место\tФамилия\t\tСумма мест");
            for (int i = 0; i < sortedSkaters.Count; i++)
            {
                var skater = sortedSkaters[i];
                C.WriteLine($"{i + 1}\t{skater.LastName}\t\t{skater.TotalPlace:F2}");
            }
        }
    }

    // 3 Уровень

    // Задание 3: Определение победителя среди команд
    class TeamMemberCustom
    {
        public string Name { get; set; }
        public int Position { get; set; }

        public TeamMemberCustom(string name, int position)
        {
            Name = name;
            Position = position;
        }

        public int GetPoints()
        {
            switch (Position)
            {
                case 1: return 5;
                case 2: return 4;
                case 3: return 3;
                case 4: return 2;
                case 5: return 1;
                default: return 0;
            }
        }
    }

    class TeamCustom

```

```

{
    public string TeamName { get; set; }
    public List<TeamMemberCustom> Members { get; set; } = new();

    public TeamCustom(string teamName)
    {
        TeamName = teamName;
    }

    public void AddMember(TeamMemberCustom member)
    {
        Members.Add(member);
    }

    public int TotalPoints()
    {
        return Members.Sum(member => member.GetPoints());
    }

    public TeamMemberCustom GetFirstPlaceMember()
    {
        return Members.OrderBy(m => m.Position).First();
    }
}

class TeamCompetitionCustom
{
    public List<TeamCustom> Teams { get; set; } = new();

    public void AddTeam(TeamCustom team)
    {
        Teams.Add(team);
    }

    public void DisplayWinner()
    {
        var sortedTeams = Teams.OrderByDescending(t => t.TotalPoints()).ToList();
        var winningTeam = sortedTeams.First();
        var firstPlaceMember = winningTeam.GetFirstPlaceMember();
        C.WriteLine($"Победитель: Команда {winningTeam.TeamName}");
        C.WriteLine($"Первое место: {firstPlaceMember.Name}, очки: {firstPlaceMember.GetPoints()}");
    }
}

// Задание 4: Лыжные гонки
class SkiRaceParticipantCustom
{
    public string Name { get; set; }
    public int Position { get; set; }

    public SkiRaceParticipantCustom(string name, int position)
    {
        Name = name;
        Position = position;
    }
}

class SkiRaceGroupCustom
{
    public string GroupName { get; set; }
    public List<SkiRaceParticipantCustom> Participants { get; set; } = new();

    public SkiRaceGroupCustom(string groupName)
    {
        GroupName = groupName;
    }
}

```



```

public void AddParticipant(SkiRaceParticipantCustom participant)
{
    Participants.Add(participant);
}

public void DisplayResults()
{
    var sortedParticipants = Participants.OrderBy(p => p.Position).ToList();
    C.WriteLine($"Результаты {GroupName}:");
    C.WriteLine("Место\tФамилия");
    foreach (var participant in sortedParticipants)
    {
        C.WriteLine($"{participant.Position}\t{participant.Name}");
    }
}

}

class CombinedSkiRaceResultsCustom
{
    public List<SkiRaceParticipantCustom> AllParticipants { get; set; } = new();

    public void AddParticipant(SkiRaceParticipantCustom participant)
    {
        AllParticipants.Add(participant);
    }

    public void DisplayCombinedResults()
    {
        var sortedParticipants = AllParticipants.OrderBy(p =>
p.Position).ToList();
        C.WriteLine("Общие результаты лыжных гонок:");
        C.WriteLine("Место\tФамилия");
        foreach (var participant in sortedParticipants)
        {
            C.WriteLine($"{participant.Position}\t{participant.Name}");
        }
    }
}

// Задание 5: Результаты первенства по футболу
class FootballTeamCustom
{
    public string TeamName { get; set; }
    public int Points { get; set; }
    public int GoalsScored { get; set; }
    public int GoalsConceded { get; set; }

    public FootballTeamCustom(string teamName)
    {
        TeamName = teamName;
        Points = 0;
        GoalsScored = 0;
        GoalsConceded = 0;
    }

    public void AddGameResult(int goalsScored, int goalsConceded)
    {
        GoalsScored += goalsScored;
        GoalsConceded += goalsConceded;
        if (goalsScored > goalsConceded)
        {
            Points += 3; // win
        }
        else if (goalsScored == goalsConceded)
        {
            Points += 1; // draw
        }
    }
}

```

```

        }
        // No points for a loss
    }

    public int GoalDifference => GoalsScored - GoalsConceded;
}

class FootballLeagueCustom
{
    public List<FootballTeamCustom> Teams { get; set; } = new();

    public void AddTeam(FootballTeamCustom team)
    {
        Teams.Add(team);
    }

    public void DisplayLeagueResults()
    {
        var sortedTeams = Teams.OrderByDescending(t => t.Points)
            .ThenByDescending(t => t.GoalDifference)
            .ToList();

        C.WriteLine("Таблица футбольного первенства:");
        C.WriteLine("Место\tКоманда\tОчки\tРазница мячей");
        for (int i = 0; i < sortedTeams.Count; i++)
        {
            var team = sortedTeams[i];
            C.WriteLine($"{i +
1}\t{team.TeamName}\t{team.Points}\t{team.GoalDifference}");
        }
    }
}

// Методы для загрузки данных из файлов
class DataLoader
{
    public static LongJumpCompetitionCustom LoadLongJumpCompetition(string
filePath)
    {
        var competition = new LongJumpCompetitionCustom();
        if (File.Exists(filePath))
        {
            var lines = File.ReadAllLines(filePath);
            foreach (var line in lines)
            {
                if (string.IsNullOrEmpty(line)) continue;
                var parts = line.Split(';');
                if (parts.Length == 4)
                {
                    var lastName = parts[0].Trim();
                    var club = parts[1].Trim();
                    if (double.TryParse(parts[2].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double attempt1) &&
                        double.TryParse(parts[3].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double attempt2))
                    {
                        competition.AddParticipant(new
LongJumpParticipantCustom(lastName, club, attempt1, attempt2));
                    }
                }
            }
        }
        return competition;
    }

    public static CrossCountryCompetitionCustom LoadCrossCountryCompetition(string
filePath, out double normative)
    {

```

```

var competition = new CrossCountryCompetitionCustom();
normative = 90.0; // default
if (File.Exists(filePath))
{
    var lines = File.ReadAllLines(filePath);
    if (lines.Length > 0)
    {
        if (double.TryParse(lines[0].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double norm))
        {
            normative = norm;
        }
    }
    for (int i = 1; i < lines.Length; i++)
    {
        var line = lines[i];
        if (string.IsNullOrEmpty(line)) continue;
        var parts = line.Split(';');
        if (parts.Length == 4)
        {
            var lastName = parts[0].Trim();
            var group = parts[1].Trim();
            var coach = parts[2].Trim();
            if (double.TryParse(parts[3].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double result))
            {
                competition.AddParticipant(new
CrossCountryParticipantCustom(lastName, group, coach, result, normative));
            }
        }
    }
    return competition;
}

public static SurveyCustom LoadSurvey(string filePath)
{
    var survey = new SurveyCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            survey.AddVote(line.Trim());
        }
    }
    return survey;
}

public static ChessTournamentCustom LoadChessTournament(string filePath)
{
    var tournament = new ChessTournamentCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(';');
            if (parts.Length == 4)
            {
                var player1 = parts[0].Trim();
                var player2 = parts[1].Trim();
                if (double.TryParse(parts[2].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double res1) &&

```

```

        double.TryParse(parts[3].Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double res2))
    {
        if (tournament.Players.All(p => p.LastName != player1))
            tournament.AddPlayer(new ChessPlayerCustom(player1));
        if (tournament.Players.All(p => p.LastName != player2))
            tournament.AddPlayer(new ChessPlayerCustom(player2));
        tournament.AddGameResult(player1, player2, res1, res2);
    }
}

return tournament;
}

public static HockeyTeamSelectionCustom LoadHockeyTeam(string filePath)
{
    var team = new HockeyTeamSelectionCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(';');
            if (parts.Length == 2)
            {
                var lastName = parts[0].Trim();
                if (int.TryParse(parts[1].Trim(), out int penalty))
                {
                    var player = team.Players.FirstOrDefault(p => p.LastName
== lastName);

                    if (player == null)
                    {
                        player = new HockeyPlayerCustom(lastName);
                        team.AddPlayer(player);
                    }
                    player.AddPenaltyTime(penalty);
                }
            }
        }
    }
    return team;
}

public static FigureSkatingCompetitionCustom LoadFigureSkating(string
filePath)
{
    var competition = new FigureSkatingCompetitionCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(';');
            if (parts.Length == 2)
            {
                var lastName = parts[0].Trim();
                var ranks = parts[1].Split(',');
                var skater = new FigureSkaterCustom(lastName);
                foreach (var r in ranks)
                {
                    if (double.TryParse(r.Trim(), NumberStyles.Any,
CultureInfo.InvariantCulture, out double rank))
                    {
                        skater.AddJudgeRank(rank);
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    competition.AddSkater(skater);
}
}
return competition;
}

public static TeamCompetitionCustom LoadTeamCompetition(string filePath)
{
    var teamComp = new TeamCompetitionCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(':');
            if (parts.Length == 2)
            {
                var teamName = parts[0].Trim();
                var team = new TeamCustom(teamName);
                var membersData = parts[1].Split(',');
                foreach (var memberStr in membersData)
                {
                    var subparts = memberStr.Split('-');
                    if (subparts.Length == 2)
                    {
                        var name = subparts[0].Trim();
                        if (int.TryParse(subparts[1].Trim(), out int pos))
                        {
                            team.AddMember(new TeamMemberCustom(name, pos));
                        }
                    }
                }
                teamComp.AddTeam(team);
            }
        }
    }
    return teamComp;
}

public static SkiRaceGroupCustom LoadSkiRaceGroup(string filePath, string
groupName)
{
    var group = new SkiRaceGroupCustom(groupName);
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(';');
            if (parts.Length == 2)
            {
                var lastName = parts[0].Trim();
                if (int.TryParse(parts[1].Trim(), out int pos))
                {
                    group.AddParticipant(new
SkiRaceParticipantCustom(lastName, pos));
                }
            }
        }
    }
    return group;
}

```

```

public static FootballLeagueCustom LoadFootballLeague(string filePath)
{
    var league = new FootballLeagueCustom();
    if (File.Exists(filePath))
    {
        var lines = File.ReadAllLines(filePath);
        foreach (var line in lines)
        {
            if (string.IsNullOrEmpty(line)) continue;
            var parts = line.Split(';');
            if (parts.Length == 5)
            {
                var teamName = parts[0].Trim();
                var team = new FootballTeamCustom(teamName);
                if (int.TryParse(parts[1].Trim(), out int g1) &&
                    int.TryParse(parts[2].Trim(), out int gc1))
                {
                    team.AddGameResult(g1, gc1);
                }
                if (int.TryParse(parts[3].Trim(), out int g2) &&
                    int.TryParse(parts[4].Trim(), out int gc2))
                {
                    team.AddGameResult(g2, gc2);
                }
                league.AddTeam(team);
            }
        }
    }
    return league;
}

// Помощник для записи результатов в файлы
class FileOutputHelper
{
    public static void WriteCompetitionResultToFile(string fileName, Action
displayAction)
    {
        var originalOut = C.Out;
        using (StreamWriter sw = new StreamWriter(fileName))
        {
            C.SetOut(sw);
            displayAction();
        }
        C.SetOut(originalOut);
    }
}

// Главный класс программы
class Program
{
    static void Main(string[] args)
    {
        // Для корректной работы с CultureInfo при чтении чисел
        CultureInfo.DefaultThreadCurrentCulture = CultureInfo.InvariantCulture;

        // Задача 1: Соревнования по прыжкам в длину
        var longJumpCompetition = DataLoader.LoadLongJumpCompetition("D://Shit
delete/longjump_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("D://Shit
delete/longjump_results.txt", () =>
        {
            C.WriteLine("Задача 1: Соревнования по прыжкам в длину");
            longJumpCompetition.DisplayResults();
        });
    }
}

```

```

        // Задача 2: Кросс на 500 м для женщин
        double normative;
        var crossCountryCompetition =
DataLoader.LoadCrossCountryCompetition("D://Shit delete/crosscountry_input.txt",
out normative);
        FileOutputHelper.WriteCompetitionResultToFile("D://Shit
delete/crosscountry_results.txt", () =>
        {
            C.WriteLine("Задача 2: Кросс на 500 м для женщин");
            crossCountryCompetition.DisplayResults(normative);
        });

        // Задача 3: Опрос радиокompании «Человек года»
        var survey = DataLoader.LoadSurvey("D://Shit delete/survey_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("D://Shit
delete/survey_results.txt", () =>
        {
            C.WriteLine("Задача 3: Опрос радиокompании 'Человек года'");
            survey.DisplayTopCandidates(5);
        });

        // Задание 7: Турнирная таблица по шахматам
        var chessTournament = DataLoader.LoadChessTournament("chess_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("chess_results.txt", () =>
        {
            C.WriteLine("Задача 7: Турнирная таблица по шахматам");
            chessTournament.DisplayResults();
        });

        // Задание 8: Сборная по хоккею
        var hockeyTeam = DataLoader.LoadHockeyTeam("hockey_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("hockey_results.txt", () =>
        {
            C.WriteLine("Задача 8: Сборная по хоккею");
            hockeyTeam.DisplaySelectedPlayers();
        });

        // Задание 9: Результаты соревнований фигуристов
        var figureSkatingCompetition =
DataLoader.LoadFigureSkating("figureskating_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("figureskating_results.txt",
        () =>
        {
            C.WriteLine("Задача 9: Результаты соревнований фигуристов");
            figureSkatingCompetition.DisplayResults();
        });

        // Задание 3: Определение победителя среди команд
        var teamCompetition = DataLoader.LoadTeamCompetition("team_input.txt");
        FileOutputHelper.WriteCompetitionResultToFile("team_results.txt", () =>
        {
            C.WriteLine("Задача 3: Определение победителя среди команд");
            teamCompetition.DisplayWinner();
        });

        // Задание 4: Лыжные гонки
        var skiGroupA = DataLoader.LoadSkiRaceGroup("skirace_groupA.txt", "Группа
A");
        var skiGroupB = DataLoader.LoadSkiRaceGroup("skirace_groupB.txt", "Группа
B");
        var combinedResults = new CombinedSkiRaceResultsCustom();
        foreach (var participant in skiGroupA.Participants)
        {
            combinedResults.AddParticipant(participant);
        }
        foreach (var participant in skiGroupB.Participants)
        {

```

```

        combinedResults.AddParticipant(participant);
    }
    FileOutputHelper.WriteCompetitionResultToFile("skirace_results.txt", () =>
    {
        C.WriteLine("Задача 4: Лыжные гонки");
        skiGroupA.DisplayResults();
        skiGroupB.DisplayResults();
        combinedResults.DisplayCombinedResults();
    });

    // Задание 5: Результаты первенства по футболу
    var footballLeague = DataLoader.LoadFootballLeague("football_input.txt");
    FileOutputHelper.WriteCompetitionResultToFile("football_results.txt", ()
=>
    {
        C.WriteLine("Задача 5: Результаты первенства по футболу");
        footballLeague.DisplayLeagueResults();
    });

    // Информировать пользователя о завершении работы
    C.WriteLine("Обработка завершена. Результаты записаны в файлы.");
}
}

```

6. ПРИМЕР ВЫПОЛНЕНИЯ (СКРИНШОТЫ)

.....

.....

```

Lewandowski;Spartak;6.5;7.1
Yamal;Dynamo;7.2;6.9
Raphinha;Labor;6.8;7.3
|

```

Задача 1: Соревнования по прыжкам в длину
Результаты соревнований по прыжкам в длину:

Место	Фамилия	Общество	Попытка 1	Попытка 2	Сумма
1	Yamal	Dynamo	7.20	6.90	14.10
2	Raphinha	Labor	6.80	7.30	14.10
3	Lewandowski	Spartak	6.50	7.10	13.60

Файл Изменить Просмотр

90.0

Lewandowski;Group 1;Smirnov;85.4

Yamal;Group 2;Petrov;92.1

Raphinha;Group 1;Smirnov;88.7

Файл Изменить Просмотр

Задача 2: Кросс на 500 м для женщин

Результаты кросса на 500 м для женщин:

Фамилия	Группа	Преподаватель	Результат	Норматив
<u>Lewandowski</u>	Group 1	<u>Smirnov</u>	85.4	Выполнен
<u>Raphinha</u>	Group 1	<u>Smirnov</u>	88.7	Выполнен
<u>Yamal</u>	Group 2	<u>Petrov</u>	92.1	Не выполнен

Всего участниц, выполнивших норматив: 2

Lewandowski

Yamal

Raphinha

Lewandowski

Raphinha

Lewandowski

Yamal

Yamal

Raphinha

Raphinha

Файл Изменить Просмотр		
Задача 3: Опрос радиокompании 'Человек года'		
Результаты опроса 'Человек года':		
Кандидат	Голосов	Доля (%)
<u>Raphinha</u>	4	40.00
<u>Lewandowski</u>	3	30.00
<u>Yamal</u>	3	30.00

7. ВЫВОД

.....

.....

В результате выполнения лабораторной работы была разработана комплексная программа, способная считывать данные из файлов, корректно их обрабатывать и записывать результаты выполнения задач в отдельные файлы. Работа показала, что применение файлового ввода-вывода в сочетании с мощными инструментами языка C# (такими как коллекции, LINQ и методы парсинга) существенно повышает гибкость и автоматизацию программных решений, что подтверждает эффективность выбранного подхода и успешное освоение поставленных задач.