

Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas I (P175B118)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Studentas

Lekt. Kęstutis Simonavičius

Dėstytojas

TURINYS

1. Duomenų klasė	3
1.1. Darbo užduotis.....	3
1.2. Programos tekstas	3
1.3. Pradiniai duomenys ir rezultatai	8
1.4. Dėstytojo pastabos.....	10
2. Skaičiavimų klasė	11
2.1. Darbo užduotis.....	11
2.2. Programos tekstas	11
2.3. Pradiniai duomenys ir rezultatai	20
2.4. Dėstytojo pastabos.....	22
3. Konteineris.....	23
3.1. Darbo užduotis.....	23
3.2. Programos tekstas	23
3.3. Pradiniai duomenys ir rezultatai	37
3.4. Dėstytojo pastabos.....	40
4. Teksto analizė ir redagavimas	41
4.1. Darbo užduotis.....	41
4.2. Programos tekstas	41
4.3. Pradiniai duomenys ir rezultatai	46
4.4. Dėstytojo pastabos.....	49
5. Paveldėjimas	50
5.1. Darbo užduotis.....	50
5.2. Programos tekstas	50
5.3. Pradiniai duomenys ir rezultatai	50
5.4. Dėstytojo pastabos.....	50

1. Duomenų klasė

1.1. Darbo užduotis

U1-9. IMDB.

Turite iš IMDB „ištrauktą“ filmų sąrašą. Duomenų faile pateikta informacija apie filmus: filmo pavadinimas, leidimo metai, žanras, kino studija, režisierius, 2 aktoriai, pajamos. • Raskite pelningiausią 2019 m. filmą, ekrane atspausdinkite šio filmo pavadinimą, režisierių, bei kiek filmas uždirbo. Jei yra keli, spausdinkite visus. • Raskite daugiausiai filmų pastačiusį režisierių, ekrane atspausdinkite jo pavardę. Jei yra keli, spausdinkite visus. • Sudarykite filmų, kuriuose vaidino N. Cage, sąrašą, į failą „Cage.csv“ įrašykite filmų pavadinimus, leidimo metus bei kino studijos pavadinimus.

1.2. Programos tekstas

AllMovieInfo.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// Saves Important Information For ALL IMDB Class Objects
    /// </summary>
    static class AllMovieInfo
    {
        private static Dictionary<string, int> DirectorPopularity = new Dictionary<string,
int>();

        /// <summary>
        /// Finds Directors With The Most Movies Directed. Returns List String Object.
        /// </summary>
        /// <returns></returns>
        public static List<string> FindBestDirectors()
        {
            List<string> directors = new List<string>();
            int filmsDirected = 0;

            foreach (string key in DirectorPopularity.Keys)
            {
                if (filmsDirected < DirectorPopularity[key])
                {
                    filmsDirected = DirectorPopularity[key];
                    directors.Clear();
                    directors.Add(key);
                }
                else if (filmsDirected == DirectorPopularity[key])
                {
                    directors.Add(key);
                }
            }
        }
    }
}
```

```

        return directors;
    }

    /// <summary>
    /// Adds a Movie Tally To The Director
    /// </summary>
    /// <param name="director"></param>
    public static void AddDirector(string director)
    {
        /// <summary>
        /// Records how many movies a director has directed.
        /// </summary>

        if (DirectorPopularity.ContainsKey(director) == false)
            DirectorPopularity.Add(director, 0);

        DirectorPopularity[director]++;
    }
}
}

```

IMDB.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Lab01
{
    /// <summary>
    /// IMDB Movie Object
    /// </summary>
    class IMDB
    {
        public string      Name      { get; set; }
        public int         Date      { get; set; }
        public string      Genre     { get; set; }
        public string      Studio    { get; set; }
        public string      Director  { get; set; }
        public List<string> Actors   { get; set; }
        public int         Revenue   { get; set; }

        public IMDB(string name,
                    int date,
                    string genre,
                    string studio,
                    string director,
                    string actor1,
                    string actor2,
                    int revenue)
        {
            Name      = name;
            Date      = date;
            Genre     = genre;
            Director  = director;
            Revenue   = revenue;
            Studio    = studio;

            Actors = new List<string>();
            Actors.Add(actor1);
            Actors.Add(actor2);
        }
    }
}

```

```

        AllMovieInfo.AddDirector(Director);
    }
}

```

InOutHelpers.cs:

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// File Input Output Helper
    /// </summary>
    static class InOutHelpers
    {
        // Text formatting const
        private const int tSize = -20;

        /// <summary>
        /// REwrites Initial Data. Takes List<IMDB>, string outputPath. Returns void
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="outputPath"> output path to where to write the data</param>
        public static void WriteInitialData(List<IMDB> movies, string outputPath)
        {
            using (StreamWriter sw = new StreamWriter(outputPath))
            {
                sw.WriteLine($"{ "Name", tSize}|" +
                    $"{ "Date", tSize}|" +
                    $"{ "Genre", tSize}|" +
                    $"{ "Studio", tSize}|" +
                    $"{ "Director", tSize}|" +
                    $"{ "Actors", (tSize * 2) - 1}|" +
                    $"{ "Revenue", -10}|"");

                foreach (IMDB movie in movies)
                {
                    sw.WriteLine($"{ movie.Name, tSize}|" +
                        $"{ movie.Date, -tSize}|" +
                        $"{ movie.Genre, tSize}|" +
                        $"{ movie.Studio, tSize}|" +
                        $"{ movie.Director, tSize}|" +
                        $"{ movie.Actors[0], tSize}|" +
                        $"{ movie.Actors[1], tSize}|" +
                        $"{ movie.Revenue, 10}|"");
                }
            }

            /// <summary>
            /// Writes Data to Output File
            /// </summary>
            /// <param name="movies">List IMDB Object</param>
            /// <param name="ouputPath">Output File Path</param>
            public static void PrintMoviesToCSV(this List<IMDB> movies, string ouputPath)
            {
                using (StreamWriter sw = new StreamWriter(ouputPath))
                {

```

```

        sw.WriteLine($"{ "Name", tSize }; { "Date", tSize }; { "Studio", tSize }");
        foreach (IMDB movie in movies)
            sw.WriteLine($"{movie.Name}; {movie.Date}; {movie.Studio}");
    }

}

/// <summary>
/// Reads Data, returns List IMDB Object
/// </summary>
/// <param name="filePath">Input File Object</param>
/// <returns></returns>
public static List<IMDB> ReadData(string filePath)
{
    List<IMDB> output = new List<IMDB>();

    using (StreamReader sr = new StreamReader(filePath))
    {
        string line;
        while ((line = sr.ReadLine()) != null)
        {
            string[] data = line.Split(';');
            IMDB imdb = new IMDB(data[0],
                                int.Parse(data[1]),
                                data[2],
                                data[3],
                                data[4],
                                data[5],
                                data[6],
                                int.Parse(data[7]));

            output.Add(imdb);
        }
    }

    return output;
}
}

```

TaskUtils.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// Task Utilities For Console And Extension Methods For Filtering
    /// </summary>
    static class TaskUtils
    {
        /// <summary>
        /// Prints Most Profitable Movies
        /// </summary>
        /// <param name="movies">List IMDB Object</param>
        public static void PrintMostProfitable(this List<IMDB> movies)
        {
            if (movies.Count == 0)
                Console.WriteLine("No Movies Found");
            else

```

```

        {
            Console.WriteLine($"{ "Name", -20} | { "Director", -20} | { "Revenue", -20}");
            foreach (IMDB movie in movies)
            {
                Console.WriteLine($"{movie.Name, -20} | {movie.Director, -
20} | {movie.Revenue, 6}");
            }
        }
    }

    /// <summary>
    /// Prints Directors in "PrintBestDirectors" format
    /// </summary>
    /// <param name="directors">List IMDB Object</param>
    public static void PrintBestDirectors(this List<string> directors)
    {
        Console.WriteLine("Best Directors: ");
        if (directors.Count == 0)
            Console.WriteLine("No Directors Found");
        else
            foreach (string director in directors)
                Console.WriteLine(director);
    }

    /// <summary>
    /// Finds Movies With Given Actor (string)
    /// </summary>
    /// <param name="movies">List IMDB object</param>
    /// <param name="actor">Actor Name to Be Searched</param>
    /// <returns></returns>
    public static List<IMDB> FindMoviesWith(this List<IMDB> movies, string actor)
    {
        List<IMDB> output = new List<IMDB>();

        foreach (IMDB movie in movies)
            if (movie.Actors.Contains(actor))
                output.Add(movie);

        return output;
    }

    /// <summary>
    /// Finds Most Profitable Movies in a given year (int)
    /// </summary>
    /// <param name="movies">List IMDB object</param>
    /// <param name="year">int year when the movie was released</param>
    /// <returns></returns>
    public static List<IMDB> FindMostProfitable(this List<IMDB> movies, int year)
    {
        List<IMDB> output = new List<IMDB>();
        int profitability = 0;

        Console.WriteLine($"Most Profitable Movies in Year: {year}");
        foreach (IMDB movie in movies)
        {
            if (movie.Date == year)
            {
                if (profitability < movie.Revenue)
                {
                    profitability = movie.Revenue;
                    output.Clear();
                    output.Add(movie);
                }
                else if (profitability == movie.Revenue)
                {

```

```

        output.Add(movie);
    }
}
return output;
}
}
}

```

Program.cs:

```

using System;
using System.Collections.Generic;

namespace Lab01
{
    class Program
    {
        const string CDd = @"imdb2.txt";
        const string CDinitial = @"imdbInitial.txt";
        const string CDcsv = @"MoviesWith.csv";
        static void Main(string[] args)
        {
            List<IMDB> imdb = InOutHelpers.ReadData(CDd);
            InOutHelpers.WriteInitialData(imdb, CDinitial);
            imdb.FindMostProfitable(2019).PrintMostProfitable();
            Console.WriteLine(new string('-', 74));
            AllMovieInfo.FindBestDirectors().PrintBestDirectors();
            imdb.FindMoviesWith("N. Cage").PrintMoviesToCSV(CDcsv);
            Console.ReadLine();
        }
    }
}

```

1.3. Pradiniai duomenys ir rezultatai

Pirmas testinis variantas

Pradiniai duomenys:

imdb.txt

```

Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom
Jr;212
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;
118
Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho;
98
Hangover 2;2015;Comedy;Studio A;Director A;J. Beam;L. Nas;318

```

Rezultatai

imdbinitial.txt:

Name	Date	Genre	Studio	Director	Actors	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	318
Hamilton	2020	History	Netflix	Lin-Manuel Miranda	Lin-Manuel Miranda	212
Parasite	2019	Thriller	CJ Entertainment	Bong Joon Ho	Kang-ho Song	212
Ghost Rider	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	118
Snowpiercer	2013	Science Fiction	CJ Entertainment	Bong Joon Ho	C. Evans	98
Hangover 2	2015	Comedy	Studio A	Director A	J. Beam	318

Console:

C:\Users\norsta\Desktop\KTU-OOP-Semester1-main\Lab01\Lab01\bin\Debug\net461\Lab01.exe

Most Profitable Movies in Year: 2019

Name	Director	Revenue
Parasite	Bong Joon Ho	212

Best Directors:

Director A
Bong Joon Ho

MoviesWith.csv:

	A	B	C
1	Name	Date	Studio
2	Hangover	2012	Studio A
3	Ghost Rider	2007	Columbia Pictures
4			

Antras testinis variantas

Pradiniai duomenys:

Imdb2.txt

Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
 Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom Jr;212
 Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
 Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;118
 Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho;98
 Cars 3;2019;Comedy;Studio A;Director A;J. Beam;L. Nas;212

Rezultatai:

ImdbInitial.txt:

Name	Date	Genre	Studio	Director	Actors	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	318
Hamilton	2020	History	Netflix	Lin-Manuel Miranda	Lin-Manuel Miranda	212
Parasite	2019	Thriller	CJ Entertainment	Bong Joon Ho	Kang-ho Song	212
Ghost Rider	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	118
Snowpiercer	2013	Science Fiction	CJ Entertainment	Bong Joon Ho	C. Evans	98
Cars 3	2019	Comedy	Studio A	Director A	J. Beam	212

Console:

```
C:\Users\norsta\Desktop\KTU-OOP-Semester1-main\Lab01\Lab01\bin\Debug\net461\Lab01.exe
Most Profitable Movies in Year: 2019
Name      Director      Revenue
Parasite   Bong Joon Ho   212
Cars 3     Director A     212
-----
Best Directors:
Director A
Bong Joon Ho
```

MoviesWith.csv:

	A	B	C	D	E	F	G
1	Name	Date	Studio				
2	Hangover	2012	Studio A				
3	Ghost Rider	2007	Columbia Pictures				
4							
5							
6							
7							
8							
9							

1.4. Dėstytojo pastabos

2. Skaičiavimų klasė

2.1. Darbo užduotis

U2-9. IMBD. Turite dviejų kinomanų mėgėjų peržiūrėtus filmų sąrašus. Keičiasi duomenų formatas. Pirmoje eilutėje kino mėgėjo vardas pavardė, antroje - gimimo metai, trečioje - miestas. Toliau informacija apie filmus pateikta tokiu pačiu formatu kaip L1 užduotyje.

- Sudarykite filmų, kuriuos peržiūrėjo abu kino mėgėjai, sąrašą. Visus duomenis apie filmus įrašykite į failą „MatėAbu.csv“.
- Raskite pelningiausią filmą. Atspausdinkite ekrane visus jo duomenis. Jei yra keli, spausdinkite visus.
- Sudarykite filmų žanrų sąrašą, įrašykite juos į failą „Žanrai.csv“.

2.2. Programos tekstas

IMDB.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Lab02
{
    /// <summary>
    /// IMDB Movie Object
    /// </summary>
    class IMDB
    {
        public string      Name      { get; set; }
        public int         Date      { get; set; }
        public string      Genre     { get; set; }
        public string      Studio    { get; set; }
        public string      Director  { get; set; }
        public List<string> Actors    { get; set; }
        public int         Revenue   { get; set; }

        public IMDB(string name,
                    int date,
                    string genre,
                    string studio,
                    string director,
                    string actor1,
                    string actor2,
                    int revenue)
        {
            Name      = name;
            Date      = date;
            Genre     = genre;
            Director  = director;
            Revenue   = revenue;
            Studio    = studio;

            Actors = new List<string>();
            Actors.Add(actor1);
            Actors.Add(actor2);
        }

        /// <summary>
        /// Equals() Method Override
    }
```

```

    /// </summary>
    public override bool Equals(object otherObj)
    {
        return this.Name == ((IMDB)otherObj).Name;
    }

    /// <summary>
    /// Returns IMDB.Name's hashCode
    /// </summary>
    public override int GetHashCode()
    {
        return this.Name.GetHashCode();
    }

    /// <summary>
    /// ToString() override
    /// </summary>
    /// <returns></returns>
    public override string ToString() => ToString('|');

    /// <summary>
    /// ToString()
    /// </summary>
    /// <returns></returns>
    public string ToString(char splitter)
    {
        return $"{this.Name,-20}{splitter}" +
            $"{this.Date,20}{splitter}" +
            $"{this.Genre,-20}{splitter}" +
            $"{this.Studio,-20}{splitter}" +
            $"{this.Director,-20}{splitter}" +
            $"{this.Actors[0],-20}{splitter}" +
            $"{this.Actors[1],-20}{splitter}" +
            $"{this.Revenue,10}{splitter}";
    }
}
}

```

User.cs:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace Lab02
{
    /// <summary>
    /// User Class Object.
    /// Saves Name, BirthDate, City, Seen Movies
    /// </summary>
    class User
    {
        public string Name { get; set; }
        public DateTime BirthDate { get; set; }
        public string City { get; set; }
        //public List<IMDB> Movies { get { return movies; } }

        private List<IMDB> movies;

        public User(string name, DateTime birthDate, string city)
        {
            City = city;

```

```

        Name = name;
        movies = new List<IMDB>();
        BirthDate = birthDate;
    }

    public User(string name, DateTime birthDate, string city, List<IMDB> _movies)
    {
        City = city;
        Name = name;
        movies = _movies;
        BirthDate = birthDate;
    }

    /// <summary>
    /// Adds the movie to users catalogue
    /// </summary>
    public void AddMovie(IMDB imdb)
    {
        IMDB temp = AllMovieInfo.GetMovieByTitle(imdb.Name);
        if (temp != null) // If the movie exists, copies the existing movie
            imdb = temp;

        AllMovieInfo.AddMovie(imdb, this); // Adds the movie to all movie catalogue
        movies.Add(imdb); // Adds the movie to this User's catalogue
    }

    public int GetMovieCount()
    {
        return movies.Count;
    }

    public IMDB GetMovieByIndex(int index)
    {
        try
        {
            return movies[index];
        }
        catch (Exception)
        {
            return null;
        }
    }

    /// <summary>
    /// Comparison Methods
    /// </summary>
    public static bool operator < (User user1, User user2)
    {
        return user1.GetMovieCount() < user2.GetMovieCount();
    }
    public static bool operator > (User user1, User user2)
    {
        return user1.GetMovieCount() < user2.GetMovieCount();
    }

    /// <summary>
    /// ToString() override
    /// </summary>
    /// <returns></returns>
    public override string ToString()
    {
        return $"{this.Name} {this.BirthDate} {this.City}";
    }
}
}

```

AllMovieInfo.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab02
{
    /// <summary>
    /// Saves Important Information For ALL IMDB Class Objects
    /// </summary>
    static class AllMovieInfo
    {
        private static List<IMDB> AllMovies { get; set; }
        private static Dictionary<string, int> DirectorPopularity;
        private static Dictionary<string, IMDB> MovieTitleSearch;
        private static Dictionary<string, List<IMDB>> GenreSearch;
        private static Dictionary<IMDB, Dictionary<User, bool>> MovieUsers; // First Key ->
Movie, Second Key - User, Returns if the User Has seen the movie
        static AllMovieInfo()
        {
            AllMovies = new List<IMDB>();
            DirectorPopularity = new Dictionary<string, int>();
            MovieTitleSearch = new Dictionary<string, IMDB>();
            MovieUsers = new Dictionary<IMDB, Dictionary<User, bool>>();
            GenreSearch = new Dictionary<string, List<IMDB>>();
        }

        /// <summary>
        /// Finds Directors With The Most Movies Directed. Returns List String Object.
        /// </summary>
        public static List<string> FindBestDirectors()
        {
            List<string> directors = new List<string>();
            int filmsDirected = 0;

            foreach (string key in DirectorPopularity.Keys)
            {
                if (filmsDirected < DirectorPopularity[key])
                {
                    filmsDirected = DirectorPopularity[key];
                    directors.Clear();
                    directors.Add(key);
                }
                else if (filmsDirected == DirectorPopularity[key])
                {
                    directors.Add(key);
                }
            }

            return directors;
        }

        /// <summary>
        /// Adds the movie to the AllMovieInfo Class. Adds the User who has seen the movie
        /// </summary>
        public static void AddMovie(IMDB imdb, User user)
        {
            if (!MovieTitleSearch.ContainsKey(imdb.Name)) // If Movie Does Not Exist, Add The
movie
                AddMovie(imdb);
        }
    }
}
```

```

        AddUser(imdb, user); // Adds the User to the Movie User List
    }

    /// <summary>
    /// Returns IMDB object by it's title
    /// </summary>
    public static IMDB GetMovieByTitle(string title)
    {
        if (MovieTitleSearch.ContainsKey(title))
            return MovieTitleSearch[title];
        else
            return null;
    }

    /// <summary>
    /// Adds a movie to a genre. If Genre does not exist, creates the genre.
    /// </summary>
    private static void AddToGenre(IMDB imdb)
    {
        if (!GenreSearch.ContainsKey(imdb.Genre)) // Adds the genre if it does not exist
            GenreSearch.Add(imdb.Genre, new List<IMDB>());
        GenreSearch[imdb.Genre].Add(imdb);
    }

    /// <summary>
    /// Adds the movie to AllMovieInfo If it does not exist
    /// </summary>
    private static void AddMovie(IMDB imdb)
    {
        MovieTitleSearch.Add(imdb.Name, imdb);
        AddToGenre(imdb);
        AddDirector(imdb.Director);
        AllMovies.Add(imdb);
    }

    /// <summary>
    /// Adds User as a person who has seen the movie
    /// </summary>
    private static void AddUser(IMDB imdb, User user)
    {
        if (!MovieUsers.ContainsKey(imdb))
            MovieUsers.Add(imdb, new Dictionary<User, bool>());

        MovieUsers[MovieTitleSearch[imdb.Name]].Add(user, true);
    }

    /// <summary>
    /// Adds a Movie Tally To The Director
    /// </summary>
    /// <param name="director"></param>
    private static void AddDirector(string director)
    {
        /// <summary>
        /// Records how many movies a director has directed.
        /// </summary>

        if (DirectorPopularity.ContainsKey(director) == false)
            DirectorPopularity.Add(director, 0);

        DirectorPopularity[director]++;
    }

    /// <summary>

```

```

/// Gets Movies that both users have seen
/// </summary>
/// <param name="user1"></param>
/// <param name="user2"></param>
/// <returns></returns>
public static List<IMDB> GetSeenWith(this User user1, User user2)
{
    List<IMDB> output = new List<IMDB>();

    for(int i = 0; i < user1.GetMovieCount(); i++)
    {
        IMDB imdb = user1.GetMovieByIndex(i);
        if (MovieUsers[imdb].ContainsKey(user2))
            output.Add(imdb);
    }

    return output;
}

/// <summary>
/// Gets the most profitable movies
/// </summary>
public static List<IMDB> GetMostProfitable()
{
    int profit = int.MinValue;
    List<IMDB> output = new List<IMDB>();
    foreach (IMDB imdb in AllMovies)
    {
        if(profit < imdb.Revenue)
        {
            profit = imdb.Revenue;
            output.Clear();
        }

        if (profit == imdb.Revenue)
            output.Add(imdb);
    }

    return output;
}

/// <summary>
/// Returns all the keys of GenreSearch Object
/// </summary>
public static List<string> GetAllGenres()
{
    return new List<string>(GenreSearch.Keys);
}

/// <summary>
/// Return all the movies with specified genre
/// </summary>
public static List<IMDB> GetMoviesWithGenre(string key)
{
    if (GenreSearch.ContainsKey(key))
        return GenreSearch[key];
    else
        return new List<IMDB>();
}
}
}

```


InOutHelpers.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab02
{
    /// <summary>
    /// File Input Output Helper
    /// </summary>
    static class InOutHelpers
    {
        // Text formatting const
        private const int tSize = -20;

        /// <summary>
        /// Creates output file from scratch
        /// </summary>
        /// <param name="outputPath"></param>
        public static void CreateOutputFile(string outputPath)
        {
            if (File.Exists(outputPath))
                File.Delete(outputPath);

            StreamWriter sw = new StreamWriter(outputPath);
            sw.WriteLine("Initial Data:");
            sw.Close();
        }

        /// <summary>
        /// Writes Initial data from List User Object
        /// </summary>
        public static void WriteInitialData(this User user, string outputPath)
        {
            using (StreamWriter sw = new StreamWriter(outputPath, append:true))
            {
                sw.WriteLine();
                sw.WriteLine($"{user.Name,tSize}|{user.BirthDate,tSize}|{user.City,tSize}");
                sw.WriteLine();
                sw.WriteMovieList(user, '|');
            }
        }

        /// <summary>
        /// REwrites Initial Data. Takes List<IMDB>, string outputPath. Returns void
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="outputPath"> output path to where to write the data</param>
        public static void WriteMovieList(this StreamWriter sw, User user, char splitter)
        {
            if (user.GetMovieCount() > 0)
            {
                sw.WriteLine($"{ "Name",tSize}{splitter}" +
                    $"{ "Date",tSize}{splitter}" +
                    $"{ "Genre",tSize}{splitter}" +
                    $"{ "Studio",tSize}{splitter}" +
                    $"{ "Director",tSize}{splitter}" +
                    $"{ "Actor 1",tSize}{splitter}" +
                    $"{ "Actor 2", tSize}{splitter}" +
                    $"{ "Revenue",-10}{splitter}");
            }
        }
    }
}
```

```

        for (int i = 0; i < user.GetMovieCount(); i++)
        {
            IMDB movie = user.GetMovieByIndex(i);
            sw.WriteLine(movie.ToString(splitter));
        }
    }
    else
        sw.WriteLine("No Movies Found");
}

/// <summary>
/// Writes Data to Output File
/// </summary>
/// <param name="movies">List IMDB Object</param>
/// <param name="outputPath">Output File Path</param>
public static void PrintMoviesToCSV(this List<IMDB> movies, string outputPath)
{
    using (StreamWriter sw = new StreamWriter(outputPath))
    {
        WriteMovieList(sw, new User("temp", DateTime.Today, "temp", movies), ';');
    }
}

/// <summary>
/// Reads Data, returns List IMDB Object
/// </summary>
/// <param name="filePath">Input File Object</param>
/// <returns></returns>
public static User Add(this List<User> list, string filePath)
{
    List<User> output = new List<User>();
    using (StreamReader sr = new StreamReader(filePath))
    {
        // Adds New User Data
        string[] data = new string[3];
        data[0] = sr.ReadLine();
        data[1] = sr.ReadLine();
        data[2] = sr.ReadLine().Trim();
        User user = new User(data[0], DateTime.Parse(data[1]), data[2]);
        list.Add(user);

        // Adds User's Movies
        string line;
        while ((line = sr.ReadLine()) != null)
        {
            data = line.Split(';');
            if (data.Length == 8) // Adds a movie for the user
            {
                IMDB imdb = new IMDB(data[0],
                                      int.Parse(data[1]),
                                      data[2],
                                      data[3],
                                      data[4],
                                      data[5],
                                      data[6],
                                      int.Parse(data[7]));
                user.AddMovie(imdb);
            }
        }

        return user;
    }
}

```

```

    }

    /// <summary>
    /// Outputs movie genres to csv file
    /// </summary>
    /// <param name="outputFile"></param>
    public static void OutputGenres(string outputFile)
    {
        List<string> genres = AllMovieInfo.GetAllGenres();
        using (StreamWriter sw = new StreamWriter(outputFile))
        {
            if (genres.Count > 0)
            {
                foreach (var genre in genres)
                {
                    sw.Write(genre);
                    foreach (IMDB imdb in AllMovieInfo.GetMoviesWithGenre(genre))
                    {
                        sw.Write($"{imdb.Name}");
                    }
                    sw.WriteLine();
                }
            }
            else
            {
                sw.WriteLine("No Data Found");
            }
        }
    }

    /// <summary>
    /// Print to screen function
    /// </summary>
    /// <param name="movies"></param>
    public static void PrintToScreen(this List<IMDB> movies)
    {
        char splitter = '|';
        Console.WriteLine("Most Profitable Movies");

        if (movies.Count > 0)
        {
            Console.WriteLine($"{"Name",tSize}{splitter}" +
                               $"{"Date",tSize}{splitter}" +
                               $"{"Genre",tSize}{splitter}" +
                               $"{"Studio",tSize}{splitter}" +
                               $"{"Director",tSize}{splitter}" +
                               $"{"Actor 1",tSize}{splitter}" +
                               $"{"Actor 2",tSize}{splitter}" +
                               $"{"Revenue",-10}{splitter}");

            foreach (IMDB movie in movies)
            {
                Console.WriteLine(movie.ToString(splitter));
            }
        }
        else
        {
            Console.WriteLine("No Movies Found");
        }
    }
}

```

Program.cs:

```

using System;
using System.Collections.Generic;

namespace Lab02
{
    class Program

```

```

{
    const string CDdata1 = @"data1-1.txt";
    const string CDdata2 = @"data1-2.txt";
    const string CDinitial = @"imdbInitial.txt";
    const string CDbothSeen = @"MatėAbu.csv";
    const string CDGenres = @"Žanrai.csv";
    static void Main(string[] args)
    {
        InOutHelpers.CreateOutputFile(CDinitial);

        List<User> users = new List<User>();
        users.Add(CDdata1).WriteInitialData(CDinitial);
        users.Add(CDdata2).WriteInitialData(CDinitial);

        users[0].GetSeenWith(users[1]).PrintMoviesToCSV(CDbothSeen);
        AllMovieInfo.GetMostProfitable().PrintToScreen();
        InOutHelpers.OutputGenres(CDGenres);
        Console.Read();

        int b = 1;
        b.ToString();

    }

    // Add User.AddFile(other files)
    // If new user added, append new data
    // Skaitym
}
}

```

2.3. Pradiniai duomenys ir rezultatai

Pradiniai duomenys:

data1-1.txt:

```

Tomas
2000-04-12
Kaunas
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Titanic;2008;History;Studio B;Director C;N. Cage;J. Sperrow;318
Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom
Jr;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;
118
Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho;
98

```

data1-2.txt:

```

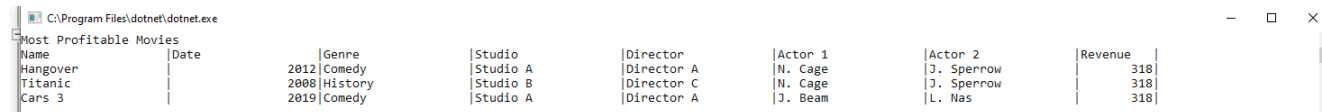
Benas
1988-03-01
Vilnius
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;
118

```

Cars 3;2019;Comedy;Studio A;Director A;J. Beam;L. Nas;318


Rezultatai:

Console:



Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Titanic	2008	History	Studio B	Director C	N. Cage	J. Sperrrow	318
Cars 3	2019	Comedy	Studio A	Director A	J. Beam	L. Nas	318

imdbInitial.txt:



imdbInitial - Notepad

File Edit Format View Help

Initial Data:

Tomas | 2000-04-12 00:00:00 | Kaunas

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Titanic	2008	History	Studio B	Director C	N. Cage	J. Sperrrow	318
Hamilton	2020	History	Netflix	Lin-Manuel Miranda	Lin-Manuel Miranda	Leslie Odom Jr	212
Ghost Rider	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	E. Mendes	118
Snowpiercer	2013	Science Fiction	CJ Entertainment	Bong Joon Ho	C. Evans	S Kang-ho	98

Benas | 1988-03-01 00:00:00 | Vilnius

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Parasite	2019	Thriller	CJ Entertainment	Bong Joon Ho	Kang-ho Song	Sun-kyun Lee	212
Ghost Rider	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	E. Mendes	118
Cars 3	2019	Comedy	Studio A	Director A	J. Beam	L. Nas	318

MatėAbu.csv:

	A	B	C	D	E	F	G	H	I
1	Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue	
2	Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318	
3	Ghost Rider	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	E. Mendes	118	
4									

Žanrai.csv:

	A	B	C
1	Comedy	Hangover	Cars 3
2	History	Titanic	Hamilton
3	Action	Ghost Rider	
4	Science Fiction	Snowpiercer	
5	Thriller	Parasite	

Pradiniai Duomenys:

data2-1.txt:

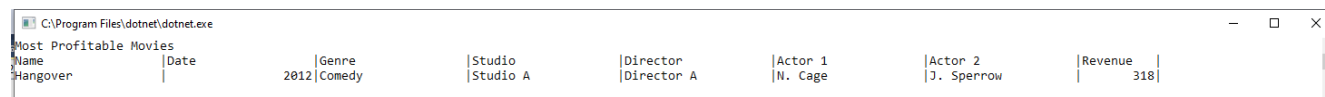
Tomas
2000-04-12
Kaunas
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrrow;318

data2-2.txt:

Benas
1988-03-01
Vilnius

Rezultatai:

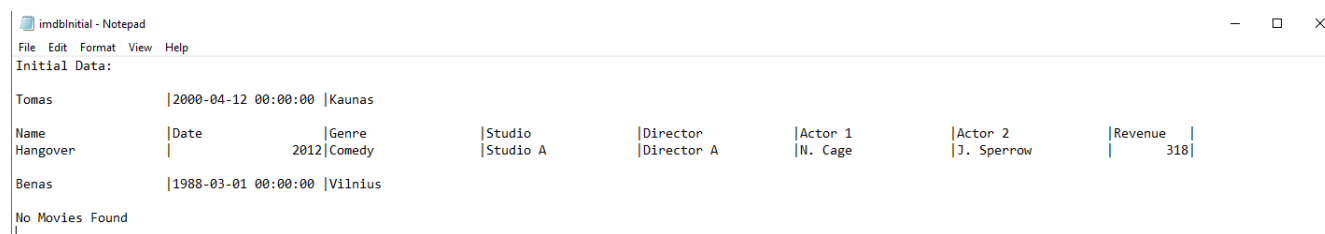
Console:



A screenshot of a console window titled "C:\Program Files\dotnet\dotnet.exe". It displays a table with the following data:

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperron	318

imdbInitial.txt:



A screenshot of a Notepad window titled "imdbInitial - Notepad". It shows "Initial Data:" followed by a table and a message "No Movies Found".

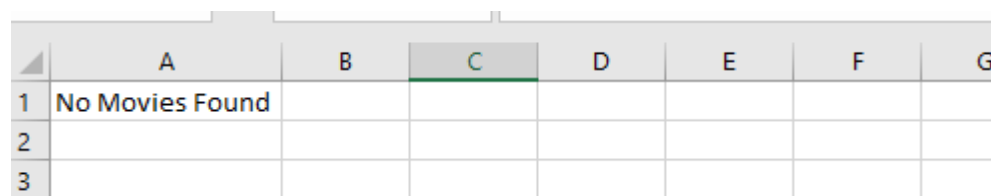
Initial Data:

Tomas | 2000-04-12 00:00:00 | Kaunas

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sperron	318
Benas	1988-03-01 00:00:00	Vilnius					

No Movies Found

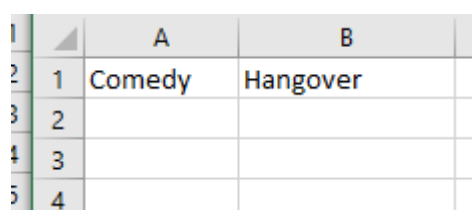
MatėAbu.csv:



A screenshot of a CSV file showing a table with 7 columns labeled A through G. The first row contains the text "No Movies Found" in column A.

	A	B	C	D	E	F	G
1	No Movies Found						
2							
3							

Žanrai.csv:



A screenshot of a CSV file showing a table with 2 columns labeled A and B. The first row contains the text "Comedy" in column A and "Hangover" in column B.

	A	B
1	Comedy	Hangover
2		
3		
4		
5		

2.4. Dėstytojo pastabos

3. Konteineris

3.1. Darbo užduotis

U3_9. **IMBD.** Turite dviejų kinomanų mėgėjų peržiūrėtų filmų sąrašus. Keičiasi duomenų formatas. Pirmoje eilutėje kino mėgėjo vardas pavardė, antroje - gimimo metai, trečioje - miestas. Toliau informacija apie filmus pateikta tokiu pačiu formatu kaip L1 užduotyje.

- Raskite ir atspausdinkite ekrane kiekvieno kino mėgėjo mėgstamiausią režisierių.
- Raskite pelningiausią filmą. Atspausdinkite ekrane visus jo duomenis. Jei yra keli, spausdinkite visus.

- Kiekvienam kino mėgėjui sudarykite rekomenduojamų peržiūrėti filmų sąrašą, į kurį įtraukite filmus, kurių jis nematė, tačiau matė kitas kino mėgėjas. Rekomendacijų sąrašus įrašykite į failus „Rekomendacija_vardas_pavardė.csv“. Surikiuokite filmus pagal leidimo metus ir pavadinimus.
- Sudarykite filmų žanrų sąrašą, įrašykite juos į failą „Žanrai.csv“.

3.2. Programos tekstas

IMDB.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Lab03
{
    /// <summary>
    /// IMDB Movie Object
    /// </summary>
    class IMDB
    {
        public string      Name      { get; set; }
        public int         Date      { get; set; }
        public string      Genre     { get; set; }
        public string      Studio    { get; set; }
        public string      Director  { get; set; }
        public List<string> Actors    { get; set; }
        public int         Revenue   { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        public IMDB(string name,
                    int date,
                    string genre,
                    string studio,
                    string director,
                    string actor1,
                    string actor2,
                    int revenue)
        {
        }
    }
}
```

```

{
    Name      = name;
    Date      = date;
    Genre     = genre;
    Director  = director;
    Revenue   = revenue;
    Studio    = studio;

    Actors = new List<string>();
    Actors.Add(actor1);
    Actors.Add(actor2);
}

/// <summary>
/// Equals() Method Override
/// </summary>
public override bool Equals(object otherObj)
{
    return this.Name == ((IMDB)otherObj).Name;
}

/// <summary>
/// Returns IMDB.Name's hashCode
/// </summary>
public override int GetHashCode()
{
    return this.Name.GetHashCode();
}

/// <summary>
/// ToString() override
/// </summary>
/// <returns></returns>
public override string ToString() => ToString('|');

/// <summary>
/// ToString()
/// </summary>
/// <returns></returns>
public string ToString(char splitter)
{
    return $"{this.Name,-20}{splitter}" +
           $"{this.Date,20}{splitter}" +
           $"{this.Genre,-20}{splitter}" +
           $"{this.Studio,-20}{splitter}" +
           $"{this.Director,-20}{splitter}" +
           $"{this.Actors[0],-20}{splitter}" +
           $"{this.Actors[1],-20}{splitter}" +
           $"{this.Revenue,10}{splitter}";
}

/// <summary>
/// CompareTo Override
/// </summary>
public int CompareTo(IMDB other)
{
    return Revenue.CompareTo(other.Revenue);
}
}
}

```


IMDBContainer.cs:

```
using System;
using System.Text;

namespace Lab03
{
    /// <summary>
    /// IMDB Container Code
    /// </summary>
    class IMDBContainer
    {
        private IMDB[] Movies;
        public int Count { get; private set; }
        public int Capacity { get; set; }
        public IMDBContainer(int capacity = 16)
        {
            this.Movies = new IMDB[capacity]; //default capacity
            Capacity = capacity;
            Count = 0;
        }

        /// <summary>
        /// IMDB Container Code
        /// </summary>
        public IMDBContainer(IMDBContainer container) : this(capacity:
container.Capacity) //calls another constructor
        {
            for (int i = 0; i < container.Count; i++)
            {
                this.Add(container.Get(i));
            }
        }

        /// <summary>
        /// IMDB Container Add
        /// </summary>
        public void Add(IMDB imdb)
        {
            if (this.Count == this.Capacity) //container is full
            {
                EnsureCapacity(this.Capacity * 2);
            }
            this.Movies[this.Count++] = imdb;
        }

        /// <summary>
        /// Get by Index function
        /// </summary>
        public IMDB Get(int index)
        {
            return this.Movies[index];
        }

        /// <summary>
        /// Contains Implementation
        /// </summary>
        public bool Contains(IMDB imdb)
        {
            for (int i = 0; i < this.Count; i++)
                if (this.Movies[i].Equals(imdb))

                return true;
        }
    }
}
```

```

        return false;
    }

    /// <summary>
    /// Ensure Capacity Implementation
    /// </summary>
    private void EnsureCapacity(int minimumCapacity)
    {
        if (minimumCapacity > this.Capacity)
        {
            IMDB[] temp = new IMDB[minimumCapacity];
            for (int i = 0; i < this.Count; i++) // Shallow Copy
            {
                temp[i] = this.Movies[i];
            }
            this.Capacity = minimumCapacity;
            this.Movies = temp;
        }
    }

    /// <summary>
    /// Put Function Container
    /// </summary>
    public IMDB Put(IMDB imdb, int index)
    {
        index = CheckIndex(index);
        if (index == Count)
        {
            if (this.Count == this.Capacity) //container is full
            {
                EnsureCapacity(this.Capacity * 2);
            }
            Count++;
        }

        IMDB otherDog = Movies[index];
        Movies[index] = imdb;

        return otherDog;
    }

    /// <summary>
    /// Insert Implementation
    /// </summary>
    public IMDB Insert(IMDB dog, int index)
    {
        if (this.Count == this.Capacity) //container is full
        {
            EnsureCapacity(this.Capacity * 2);
        }

        index = CheckIndex(index);
        for (int i = Count - 1; i >= index; i--)
        {
            Movies[i + 1] = Movies[i];
        }

        Count++;
        Movies[index] = dog;

        return dog;
    }

    /// <summary>
    /// FindIndex Container Implementation
    /// </summary>

```

```

public int FindIndex(IMDB imdb)
{
    for (int i = 0; i < Count; i++)
    {
        if (Movies[i].Equals(imdb))
            return i;
    }
    return -1;
}

/// <summary>
/// Remove Container Implementation
/// </summary>
public void Remove(IMDB imdb)
{
    int index = FindIndex(imdb);
    if (index != -1)
    {
        RemoveAt(index);
    }
}

/// <summary>
/// RemoveAt implementation
/// </summary>
public void RemoveAt(int index)
{
    if (index < Count)
    {
        // Checks if element exists, if does, removes
        for (int i = index; i < Count; i++)
            Movies[i] = Movies[i + 1];
        Movies[Count] = null;
        Count--;
    }
}

/// <summary>
/// CheckIndex if the index exists implementation
/// </summary>
private int CheckIndex(int index)
{
    if (index >= Count)
        return Count;
    return index;
}

/// <summary>
/// Selection Sort implementation
/// </summary>
public IMDBContainer Sort()
{
    for (int i = 0; i < Count - 1; i++)
    {
        int min_idx = i;
        for (int j = i + 1; j < Count; j++)
            if (Movies[j].CompareTo(Movies[min_idx]) < 0)
                min_idx = j;

        IMDB temp = Movies[min_idx];
        Movies[min_idx] = Movies[i];
        Movies[i] = temp;
    }

    return this;
}

```

```

    /// <summary>
    /// Clears the Container
    /// </summary>
    public void Clear(int capacity = 16)
    {
        this.Movies = new IMDB[capacity]; //default capacity
        Capacity = capacity;
        Count = 0;
    }

    /// <summary>
    /// ToString implementation
    /// </summary>
    public override string ToString()
    {
        return $"Element Count: {Count} Element Capacity: {Capacity}";
    }
}

```

User.cs:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace Lab03
{
    /// <summary>
    /// User Class Object.
    /// Saves Name, BirthDate, City, Seen Movies
    /// </summary>
    class User
    {
        public string Name { get; set; }
        public DateTime BirthDate { get; set; }
        public string City { get; set; }
        //public List<IMDB> Movies { get { return movies; } }

        private IMDBContainer movies;

        public User(string name, DateTime birthDate, string city)
        {
            City = city;
            Name = name;
            movies = new IMDBContainer();
            BirthDate = birthDate;
        }

        /// <summary>
        /// User Constructor with IMDBContainer
        /// </summary>
        public User(string name, DateTime birthDate, string city, IMDBContainer _movies)
        {
            City = city;
            Name = name;
            movies = _movies;
            BirthDate = birthDate;
        }

        /// <summary>
        /// Adds the movie to users catalogue
        /// </summary>
    }
}

```

```

public void AddMovie(IMDB imdb)
{
    IMDB temp = AllMovieInfo.GetMovieByTitle(imdb.Name);
    if (temp != null) // If the movie exists, copies the existing movie
        imdb = temp;

    AllMovieInfo.AddMovie(imdb, this); // Adds the movie to all movie catalogue
    movies.Add(imdb); // Adds the movie to this User's catalogue
}

/// <summary>
/// Returns MovieCount
/// </summary>
public int GetMovieCount()
{
    return movies.Count;
}

public IMDB GetMovieByIndex(int index)
{
    try
    {
        return movies.Get(index);
    }
    catch (Exception)
    {
        return null;
    }
}

/// <summary>
/// Comparison Methods
/// </summary>
public static bool operator < (User user1, User user2)
{
    return user1.GetMovieCount() < user2.GetMovieCount();
}
public static bool operator > (User user1, User user2)
{
    return user1.GetMovieCount() < user2.GetMovieCount();
}

/// <summary>
/// ToString() override
/// </summary>
/// <returns></returns>
public override string ToString()
{
    return $"{this.Name} {this.BirthDate} {this.City}";
}

/// <summary>
/// GetsFavorite Director for provided User
/// </summary>
public string[] GetFavoriteDirector()
{
    string[] names = new string[movies.Count];
    int moviesDirected = 0;
    int n = 0;

    for (int i = 0; i < movies.Count; i++)
    {
        string currName = movies.Get(i).Director;
        int currDirectedCount = 0;
        for (int j = i; j < movies.Count; j++)
            if (movies.Get(j).Director == currName)
                currDirectedCount++;
    }
}

```

```

        // Resets
        if (currDirectedCount > moviesDirected)
        {
            moviesDirected = currDirectedCount;
            names = new string[movies.Count];
            n = 0;
        }

        // Adds users
        if (currDirectedCount == moviesDirected)
        {
            names[n] = currName;
            n++;
        }
    }

    string[] output = new string[n];
    Array.Copy(names, output, n);

    return output;
}
}
}

```

AllMovieInfo.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab03
{
    /// <summary>
    /// Saves Important Information For ALL IMDB Class Objects
    /// </summary>
    static class AllMovieInfo
    {
        private static IMDBContainer AllMovies { get; set; }
        private static Dictionary<string, int> DirectorPopularity;
        private static Dictionary<string, IMDB> MovieTitleSearch;
        private static Dictionary<string, IMDBContainer> GenreSearch;
        private static Dictionary<IMDB, Dictionary<User, bool>> MovieUsers; // First Key
        -> Movie, Second Key - User, Returns if the User Has seen the movie
        static AllMovieInfo()
        {
            AllMovies = new IMDBContainer();
            DirectorPopularity = new Dictionary<string, int>();
            MovieTitleSearch = new Dictionary<string, IMDB>();
            MovieUsers = new Dictionary<IMDB, Dictionary<User, bool>>();
            GenreSearch = new Dictionary<string, IMDBContainer>();
        }

        public static IMDBContainer GetReccomendedMovies(User user)
        {
            IMDBContainer output = new IMDBContainer();

            for (int i = 0; i < AllMovies.Count; i++)
            {
                IMDB imdb = AllMovies.Get(i);
                if (MovieUsers[imdb].ContainsKey(user) == false)
                    output.Add(imdb);
            }
        }
    }
}

```

```

        return output;
    }

    /// <summary>
    /// Adds the movie to the AllMovieInfo Class. Adds the User who has seen the
movie
    /// </summary>
    public static void AddMovie(IMDB imdb, User user)
    {

        if (!MovieTitleSearch.ContainsKey(imdb.Name)) // If Movie Does Not Exist, Add
The movie
            AddMovie(imdb);

        AddUser(imdb, user); // Adds the User to the Movie User Container

    }

    /// <summary>
    /// Returns IMDB object by it's title
    /// </summary>
    public static IMDB GetMovieByTitle(string title)
    {
        if (MovieTitleSearch.ContainsKey(title))
            return MovieTitleSearch[title];
        else
            return null;
    }

    /// <summary>
    /// Adds a movie to a genre. If Genre does not exist, creates the genre.
    /// </summary>
    private static void AddToGenre(IMDB imdb)
    {
        if (!GenreSearch.ContainsKey(imdb.Genre)) // Adds the genre if it does not
exist
            GenreSearch.Add(imdb.Genre, new IMDBContainer());
        GenreSearch[imdb.Genre].Add(imdb);
    }

    /// <summary>
    /// Adds the movie to AllMovieInfo If it does not exist
    /// </summary>
    private static void AddMovie(IMDB imdb)
    {
        MovieTitleSearch.Add(imdb.Name, imdb);
        AddToGenre(imdb);
        AddDirector(imdb.Director);
        AllMovies.Add(imdb);
    }

    /// <summary>
    /// Adds User as a person who has seen the movie
    /// </summary>
    private static void AddUser(IMDB imdb, User user)
    {
        if (!MovieUsers.ContainsKey(imdb))
            MovieUsers.Add(imdb, new Dictionary<User, bool>());

        MovieUsers[MovieTitleSearch[imdb.Name]].Add(user, true);
    }

    /// <summary>
    /// Adds a Movie Tally To The Director
    /// </summary>
    /// <param name="director"></param>

```

```

private static void AddDirector(string director)
{
    /// <summary>
    /// Records how many movies a director has directed.
    /// </summary>

    if (DirectorPopularity.ContainsKey(director) == false)
        DirectorPopularity.Add(director, 0);

    DirectorPopularity[director]++;
}

/// <summary>
/// Gets Movies that both users have seen
/// </summary>
/// <param name="user1"></param>
/// <param name="user2"></param>
/// <returns></returns>
public static IMDBContainer GetSeenWith(this User user1, User user2)
{
    IMDBContainer output = new IMDBContainer();

    for(int i = 0; i < user1.GetMovieCount(); i++)
    {
        IMDB imdb = user1.GetMovieByIndex(i);
        if (MovieUsers[imdb].ContainsKey(user2))
            output.Add(imdb);
    }

    return output;
}

/// <summary>
/// Gets the most profitable movies
/// </summary>
public static IMDBContainer GetMostProfitable()
{
    int profit = int.MinValue;
    IMDBContainer output = new IMDBContainer();
    for (int i = 0; i < AllMovies.Count; i++)
    {
        IMDB imdb = AllMovies.Get(i);
        if(profit < imdb.Revenue)
        {
            profit = imdb.Revenue;
            output.Clear();
        }

        if (profit == imdb.Revenue)
            output.Add(imdb);
    }

    output.Sort();

    return output;
}

/// <summary>
/// Returns all the keys of GenreSearch Object
/// </summary>
public static string[] GetAllGenres()
{
    return GenreSearch.Keys.ToArray();
}

/// <summary>
/// Return all the movies with specified genre

```



```

    /// </summary>
    public static IMDBContainer GetMoviesWithGenre(string key)
    {
        if (GenreSearch.ContainsKey(key))
            return GenreSearch[key];
        else
            return new IMDBContainer();
    }
}

```

InOutHelpers.cs:

```

using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab03
{
    /// <summary>
    /// File Input Output Helper
    /// </summary>
    static class InOutHelpers
    {
        // Text formatting const
        private const int tSize = -20;

        /// <summary>
        /// Creates output file from scratch
        /// </summary>
        /// <param name="outputPath"></param>
        public static void CreateOutputFile(string outputPath)
        {
            if (File.Exists(outputPath))
                File.Delete(outputPath);

            StreamWriter sw = new StreamWriter(outputPath);
            sw.WriteLine("Initial Data:");
            sw.Close();
        }

        /// <summary>
        /// Writes Initial data from List User Object
        /// </summary>
        public static void WriteInitialData(this User user, string outputPath)
        {
            using (StreamWriter sw = new StreamWriter(outputPath, append:true))
            {
                sw.WriteLine();

                sw.WriteLine($"{user.Name,tSize}|{user.BirthDate,tSize}|{user.City,tSize}");
                sw.WriteLine();
                sw.WriteMovieList(user, '|');
            }
        }

        /// <summary>
        /// REwrites Initial Data. Takes List<IMDB>, string outputPath. Returns void
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="outputPath"> output path to where to write the data</param>
        public static void WriteMovieList(this StreamWriter sw, User user, char splitter)
        {

```

```

if (user.GetMovieCount() > 0)
{
    sw.WriteLine($"{ "Name",tSize}{splitter}" +
        $"{ "Date",tSize}{splitter}" +
        $"{ "Genre",tSize}{splitter}" +
        $"{ "Studio",tSize}{splitter}" +
        $"{ "Director",tSize}{splitter}" +
        $"{ "Actor 1",tSize}{splitter}" +
        $"{ "Actor 2", tSize}{splitter}" +
        $"{ "Revenue",-10}{splitter}");

    for (int i = 0; i < user.GetMovieCount(); i++)
    {
        IMDB movie = user.GetMovieByIndex(i);
        sw.WriteLine(movie.ToString(splitter));
    }
}
else
    sw.WriteLine("No Movies Found");
}

/// <summary>
/// Reads Data, returns List IMDB Object
/// </summary>
/// <param name="filePath">Input File Object</param>
/// <returns></returns>
public static User ReadUser(string filePath)
{
    User user;
    using (StreamReader sr = new StreamReader(filePath))
    {
        // Adds New User Data
        string[] data = new string[3];
        data[0] = sr.ReadLine();
        data[1] = sr.ReadLine();
        data[2] = sr.ReadLine().Trim();
        user = new User(data[0], DateTime.Parse(data[1]), data[2]);

        // Adds User's Movies
        string line;
        while ((line = sr.ReadLine()) != null)
        {
            data = line.Split(';');
            if (data.Length == 8) // Adds a movie for the user
            {
                IMDB imdb = new IMDB(data[0],
                                    int.Parse(data[1]),
                                    data[2],
                                    data[3],
                                    data[4],
                                    data[5],
                                    data[6],
                                    int.Parse(data[7]));
                user.AddMovie(imdb);
            }
        }
    }
    return user;
}

/// <summary>
/// Outputs movie genres to csv file
/// </summary>
/// <param name="outputFile"></param>
public static void OutputGenres(string outputFile)
{

```

```

string[] genres = AllMovieInfo.GetAllGenres();
using (StreamWriter sw = new StreamWriter(outputFile))
{
    if (genres.Length > 0)
    {
        foreach (var genre in genres)
        {
            sw.Write(genre);
            IMDBContainer genreCollection =
AllMovieInfo.GetMoviesWithGenre(genre);
            for (int i = 0; i < genreCollection.Count; i++)
            {
                IMDB imdb = genreCollection.Get(i);
                sw.Write($"{imdb.Name}");
            }
            sw.WriteLine();
        }
    }
    else
        sw.WriteLine("No Data Found");
}

/// <summary>
/// Print to screen function
/// </summary>
/// <param name="movies"></param>
public static void PrintToScreen(this IMDBContainer movies, string header)
{
    char splitter = '|';
    Console.WriteLine(header);

    if (movies.Count > 0)
    {
        Console.WriteLine($"{ "Name",tSize}{splitter}" +
                            $"{ "Date",tSize}{splitter}" +
                            $"{ "Genre",tSize}{splitter}" +
                            $"{ "Studio",tSize}{splitter}" +
                            $"{ "Director",tSize}{splitter}" +
                            $"{ "Actor 1",tSize}{splitter}" +
                            $"{ "Actor 2",tSize}{splitter}" +
                            $"{ "Revenue",-10}{splitter}");

        for (int i = 0; i < movies.Count; i++)
            Console.WriteLine(movies.Get(i).ToString(splitter));
    }
    else
        Console.WriteLine("No Movies Found");

    Console.WriteLine();
}

/// <summary>
/// Prints String[] to Console, With provided header at the top
/// </summary>
public static void PrintStrings(string[] strings, string header)
{
    Console.WriteLine(header);
    for (int i = 0; i < strings.Length; i++)
        Console.WriteLine(strings[i]);

    Console.WriteLine();
}

/// <summary>
/// Recommends User movies. Outputs to "[FirstName]_[LastName].csv" file format.

```

```

/// </summary>
public static void ReccomendMovies(User user)
{
    string[] nameElements = user.Name.Split(' ');
    using (StreamWriter sw = new
StreamWriter($"Rekomendacija_{nameElements[0]}_{nameElements[1]}.csv"))
    {
        char splitter = ';';
        sw.WriteLine($"{Name",tSize}{splitter}" +
                        $"{Date",tSize}{splitter}" +
                        $"{Genre",tSize}{splitter}" +
                        $"{Studio",tSize}{splitter}" +
                        $"{Director",tSize}{splitter}" +
                        $"{Actor 1",tSize}{splitter}" +
                        $"{Actor 2",tSize}{splitter}" +
                        $"{Revenue",-10}{splitter}");

        IMDBContainer reccomendedMovies =
AllMovieInfo.GetReccomendedMovies(user).Sort();

        for (int i = 0; i < reccomendedMovies.Count; i++)
            sw.WriteLine(reccomendedMovies.Get(i).ToString(';'));
    }
}
}
}

```

Program.cs:

```

using System;
using System.Collections.Generic;

namespace Lab03
{
    class Program
    {
        // Output/Input location path declarations
        const string CDdata1 = @"data1-1.txt";
        const string CDdata2 = @"data1-2.txt";
        const string COutput = @"imdbInitial.txt";
        const string CDGenres = @"Žanrai.csv";
        static void Main(string[] args)
        {
            InOutHelpers.CreateOutputFile(COutput);

            User user1 = InOutHelpers.ReadUser(CDdata1);
            user1.WriteInitialData(COutput);
            User user2 = InOutHelpers.ReadUser(CDdata2);
            user2.WriteInitialData(COutput);

            // Most profitable
            InOutHelpers.PrintToScreen(AllMovieInfo.GetMostProfitable(), "Most Profitable
Movies:");

            // Movie reccomendations
            InOutHelpers.PrintStrings(user1.GetFavoriteDirector(), $"{user1.Name}
Favorite Director(s):");
            InOutHelpers.PrintStrings(user2.GetFavoriteDirector(), $"{user2.Name}
Favorite Director(s):");

            // Genres
            InOutHelpers.OutputGenres(CDGenres);

            // Movie Reccomendation
            InOutHelpers.ReccomendMovies(user1);

```

```

        InOutHelpers.ReccomendMovies(user2);

        Console.Read();
    }

}

```

3.3. Pradiniai duomenys ir rezultatai

Pradiniai duomenys:

data1-1.txt:

```

Tomas Asas
2000-04-12
Kaunas
Moviee 1;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Titanic;2008;History;Studio B;Director B;N. Cage;J. Sperrow;318
Hangover 2;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Titanic 2;2008;History;Studio B;Director B;N. Cage;J. Sperrow;318
Ghost Rider 5;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;
118
Hangover 3;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Titanic 4;2008;History;Studio B;Director B;N. Cage;J. Sperrow;318

```

data1-2.txt:

```

Benas Fanas
2000-04-12
Kaunas
Moviee 1;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Moviee 2;2012;Comedy;Studio A;Director B;N. Cage;J. Sperrow;318
Moviee 3;2012;Comedy;Studio A;Director C;N. Cage;J. Sperrow;318
Moviee 4;2012;Comedy;Studio A;Director D;N. Cage;J. Sperrow;318
Moviee 5;2012;Comedy;Studio A;Director E;N. Cage;J. Sperrow;318
Moviee 6;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Moviee 7;2012;Comedy;Studio A;Director B;N. Cage;J. Sperrow;318
Moviee 8;2012;Comedy;Studio A;Director C;N. Cage;J. Sperrow;318

```

Rezultatai:

Console:

```

Most Profitable Movies:
Name      Date      Genre      Studio      Director      Actor 1      Actor 2      Revenue
Moviee 1  |         | 2012|Comedy|Studio A|Director A|N. Cage|J. Sperrow|318|
Titanic   |         | 2008|History|Studio B|Director B|N. Cage|J. Sperrow|318|
Hangover 2|         | 2012|Comedy|Studio A|Director A|N. Cage|J. Sperrow|318|
Titanic 2 |         | 2008|History|Studio B|Director B|N. Cage|J. Sperrow|318|
Hangover 3|         | 2012|Comedy|Studio A|Director A|N. Cage|J. Sperrow|318|
Titanic 4 |         | 2008|History|Studio B|Director B|N. Cage|J. Sperrow|318|
Moviee 2  |         | 2012|Comedy|Studio A|Director B|N. Cage|J. Sperrow|318|
Moviee 3  |         | 2012|Comedy|Studio A|Director C|N. Cage|J. Sperrow|318|
Moviee 4  |         | 2012|Comedy|Studio A|Director D|N. Cage|J. Sperrow|318|
Moviee 5  |         | 2012|Comedy|Studio A|Director E|N. Cage|J. Sperrow|318|
Moviee 6  |         | 2012|Comedy|Studio A|Director A|N. Cage|J. Sperrow|318|
Moviee 7  |         | 2012|Comedy|Studio A|Director B|N. Cage|J. Sperrow|318|
Moviee 8  |         | 2012|Comedy|Studio A|Director C|N. Cage|J. Sperrow|318|

Tomas Asas Favorite Director(s):
Director A
Director B

Benas Fanas Favorite Director(s):
Director A
Director B
Director C

```

imdbInitial.txt:

Tomas Asas |4/12/2000 12:00:00 AM|Kaunas

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Movie 1	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Titanic	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318
Hangover 2	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Titanic 2	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318
Ghost Rider 5	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	E. Mendes	118
Hangover 3	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Titanic 4	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318

Benas Fanas |4/12/2000 12:00:00 AM|Kaunas

Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
Movie 1	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Movie 2	2012	Comedy	Studio A	Director B	N. Cage	J. Sperrrow	318
Movie 3	2012	Comedy	Studio A	Director C	N. Cage	J. Sperrrow	318
Movie 4	2012	Comedy	Studio A	Director D	N. Cage	J. Sperrrow	318
Movie 5	2012	Comedy	Studio A	Director E	N. Cage	J. Sperrrow	318
Movie 6	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
Movie 7	2012	Comedy	Studio A	Director B	N. Cage	J. Sperrrow	318
Movie 8	2012	Comedy	Studio A	Director C	N. Cage	J. Sperrrow	318

Rekomendacija_Tomas_Asas.cs:

	A	B	C	D	E	F	G	H
1	Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
2	Movie 2	2012	Comedy	Studio A	Director B	N. Cage	J. Sperrrow	318
3	Movie 3	2012	Comedy	Studio A	Director C	N. Cage	J. Sperrrow	318
4	Movie 4	2012	Comedy	Studio A	Director D	N. Cage	J. Sperrrow	318
5	Movie 5	2012	Comedy	Studio A	Director E	N. Cage	J. Sperrrow	318
6	Movie 6	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
7	Movie 7	2012	Comedy	Studio A	Director B	N. Cage	J. Sperrrow	318
8	Movie 8	2012	Comedy	Studio A	Director C	N. Cage	J. Sperrrow	318
9								

Rekomendacija_Benas_Fanas.cs:

	A	B	C	D	E	F	G	H
1	Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
2	Ghost Rider 5	2007	Action	Columbia Pictures	Mark Steven Johnson	N. Cage	E. Mendes	118
3	Hangover 2	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
4	Titanic 2	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318
5	Titanic	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318
6	Hangover 3	2012	Comedy	Studio A	Director A	N. Cage	J. Sperrrow	318
7	Titanic 4	2008	History	Studio B	Director B	N. Cage	J. Sperrrow	318
8								

Žarai.cs:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Comedy	Movie 1	Hangover	Hangover	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	
2	History	Titanic	Titanic 2	Titanic 4								
3	Action	Ghost Rider 5										
4												

Pradiniai Duomenys:

data2-1.txt:

Tomas Aputis

2000-04-12

Kaunas

Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrrow;318

Hangover 2;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrrow;318

Moviee 1;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;213
 Avengers 1;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;98
 Titanic 2;2008;History;Studio B;Director D;N. Cage;J. Sperrow;318
 Hangover 3;2012;Comedy;Studio A;Director F;N. Cage;J. Sperrow;318
 Hangover 4;2012;Comedy;Studio A;Director F;N. Cage;J. Sperrow;19
 Titanic 4;2008;History;Studio B;Director M;N. Cage;J. Sperrow;318
 Titanic 5;2008;History;Studio B;Director M;N. Cage;J. Sperrow;200
 Titanic 200;2008;History;Studio B;Director K;N. Cage;J. Sperrow;2

data2-2.txt:

Benas Smilkys
 1988-03-01
 Vilnius
 Titanic 2;2008;History;Studio B;Director D;N. Cage;J. Sperrow;318
 Hangover 3;2012;Comedy;Studio A;Director F;N. Cage;J. Sperrow;318
 Hangover 4;2012;Comedy;Studio A;Director F;N. Cage;J. Sperrow;19
 Titanic 4;2008;History;Studio B;Director M;N. Cage;J. Sperrow;318
 Titanic 5;2008;History;Studio B;Director M;N. Cage;J. Sperrow;200
 Titanic 200;2008;History;Studio B;Director K;N. Cage;J. Sperrow;2

Rezultatai:

Console:

```

Most Profitable Movies:
Name      Date      Genre      Studio      Director      Actor 1      Actor 2      Revenue
Hangover   |          | 2012|Comedy    |Studio A      |Director A    |N. Cage      |J. Sperrow   |318
Hangover 2|          | 2012|Comedy    |Studio A      |Director A    |N. Cage      |J. Sperrow   |318
Titanic 2  |          | 2008|History   |Studio B      |Director D    |N. Cage      |J. Sperrow   |318
Hangover 3 |          | 2012|Comedy    |Studio A      |Director F    |N. Cage      |J. Sperrow   |318
Titanic 4  |          | 2008|History   |Studio B      |Director M    |N. Cage      |J. Sperrow   |318

Tomas Aputis Favorite Director(s):
Director A

Benas Smilkys Favorite Director(s):
Director F
Director M
  
```

imdbInitial.txt:

Initial Data:

Tomas Aputis |4/12/2000 12:00:00 AM|Kaunas

Name	Date	Genre	Studio	Director	Actor 1
Hangover		2012 Comedy	Studio A	Director A	N. Cage
Hangover 2		2012 Comedy	Studio A	Director A	N. Cage
Moviee 1		2012 Comedy	Studio A	Director A	N. Cage
Avengers 1		2012 Comedy	Studio A	Director A	N. Cage
Titanic 2		2008 History	Studio B	Director D	N. Cage
Hangover 3		2012 Comedy	Studio A	Director F	N. Cage
Hangover 4		2012 Comedy	Studio A	Director F	N. Cage
Titanic 4		2008 History	Studio B	Director M	N. Cage
Titanic 5		2008 History	Studio B	Director M	N. Cage
Titanic 200		2008 History	Studio B	Director K	N. Cage

Benas Smilkys |3/1/1988 12:00:00 AM|Vilnius

Name	Date	Genre	Studio	Director	Actor 1
Titanic 2		2008 History	Studio B	Director D	N. Cage
Hangover 3		2012 Comedy	Studio A	Director F	N. Cage
Hangover 4		2012 Comedy	Studio A	Director F	N. Cage
Titanic 4		2008 History	Studio B	Director M	N. Cage
Titanic 5		2008 History	Studio B	Director M	N. Cage
Titanic 200		2008 History	Studio B	Director K	N. Cage

Rekomendacija_Benas_Smilkys.cs:

	A	B	C	D	E	F	G	H	I
1	Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue	
2	Avengers 1	2012	Comedy	Studio A	Director A	N. Cage	J. Sparrow	98	
3	Moviee 1	2012	Comedy	Studio A	Director A	N. Cage	J. Sparrow	213	
4	Hangover 2	2012	Comedy	Studio A	Director A	N. Cage	J. Sparrow	318	
5	Hangover	2012	Comedy	Studio A	Director A	N. Cage	J. Sparrow	318	
6									
7									

Rekomendacija_Tomas_Aputis.cs:

	A	B	C	D	E	F	G	H	I
1									
2	1	Name	Date	Genre	Studio	Director	Actor 1	Actor 2	Revenue
3	2								
4	3								
5	4								
6	5								

Žanrai.cs:

	A	B	C	D	E	F	G	H
1	Comedy	Hangover	Hangover 2	Moviee 1	Avengers 1	Hangover 3	Hangover 4	
2	History	Titanic 2	Titanic 4	Titanic 5	Titanic 200			
3								
4								

3.4. Dėstytojo pastabos

4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

U4H-9. Ilgiausias sakiny

22

P175B118 Objektinis programavimas 1

23

Dviejuose tekstiniuose failuose `Knyga1.txt` ir `Knyga2.txt` duotas tekstas sudarytas iš žodžių, atskirtų skyrikliais. Skyriklių aibė žinoma ir abiejuose failuose yra ta pati.

Raskite ir spausdinkite faile `Rodikliai.txt`:

- ilgiausių žodžių, surikiuotų ilgio mažėjimo tvarka, kurie yra abiejuose failuose, sąrašą (ne daugiau nei 10 žodžių) ir jų pasikartojimo skaičių kiekviename iš failų;
- ilgiausią sakinį (didžiausias žodžių kiekis), jo ilgį (simboliais ir žodžiais) ir vietą (sakinio pradžios eilutės numerį) pirmame ir antrame faile.

Spausdinkite faile `ManoKnyga.txt` apjungtą tekstą, sudarytą pagal tokias taisykles:

- kopijuojamas pirmojo failo tekstas tol, kol sutinkamas pirmasis nenukopijuotas antrojo failo žodis arba pasiekama failo pabaiga;
- kopijuojamas antrojo failo tekstas tol, kol sutinkamas pirmasis nenukopijuotas pirmojo failo žodis arba pasiekama failo pabaiga;
- kartojama tol, kol pasiekama abiejų failų pabaiga.

4.2. Programos tekstas

InOut.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab04
{
    static class InOut
    {
        // Punctuation string
        private static char[] punctuation = { ' ', '.', ',', '!', '?', ':', ';', '(',
        ')', '\t', '\n', '\r', '\'', '\"' };

        /// <summary>
        /// Creates file for output
        /// </summary>
        public static void CreateFile(string output)
        {
            new StreamWriter(output).Close();
        }

        /// <summary>
        /// Writes words that appear in both files
        /// </summary>
        public static void WriteWordRepetitions(List<string> commonWords, Dictionary<string,
        int> repetitions, string output, string header)
```

```

    {
        using (StreamWriter sw = new StreamWriter(output, append: true))
        {
            sw.WriteLine(header);
            sw.WriteLine(new string('-', header.Length));
            foreach (string word in commonWords)
                sw.WriteLine($"{word, 25}: {repetitions[word]}");

            sw.WriteLine();
        }
    }

    /// <summary>
    /// Reads text
    /// </summary>
    public static string ReadText(string input)
    {
        return File.ReadAllText(input, Encoding.UTF8);
    }

    /// <summary>
    /// Writes The longest sentence per file
    /// </summary>
    public static void WriteLongestSentence(string output, string header, string[]
sentences, string text)
    {
        using (StreamWriter sw = new StreamWriter(output, append:true))
        {
            sw.WriteLine(header);
            sw.WriteLine(new string('-', header.Length));
            string sentence = TaskUtils.LongestSentence(sentences);
            sw.WriteLine($"Longest Sentence is: {sentence}");
            sw.WriteLine($"Symbol Count: {sentence.Length} Word Count:
{sentence.Split(punctuation, StringSplitOptions.RemoveEmptyEntries).Length}");
            sw.WriteLine($"Line Where the Sentence Starts:
{TaskUtils.GetSentenceStart(text, sentence)}");
            sw.WriteLine();
        }
    }

    /// <summary>
    /// Writes a string to output
    /// </summary>
    public static void WriteString(string output, string text)
    {
        using (StreamWriter sw = new StreamWriter(output))
            sw.WriteLine(text);
    }
}

```

TaskUtils.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

```

namespace Lab04

```

{
    static class TaskUtils
    {
        /// <summary>

```

```

/// Gets words that appear in both files
/// </summary>
private static List<string> GetDuplicates(string[] words1, string[] words2)
{
    List<string> output = new List<string>();
    foreach (string word in words1)
        if (words2.Contains(word) && !output.Contains(word))
            output.Add(word);

    return output;
}

/// <summary>
/// Sorts words by their length
/// </summary>
private static List<string> LengthSort(this List<string> data)
{
    for (int i = 0; i < data.Count - 1; i++)
        for (int j = 0; j < data.Count - 1 - i; j++)
            if (data[j].Length < data[j+1].Length)
            {
                string temp = data[j];
                data[j] = data[j+1];
                data[j+1] = temp;
            }

    return data;
}

// Turns words to lowerCaseStrings
private static void LowerCaseStrings(string[] words)
{
    for (int i = 0; i < words.Length; i++)
        words[i] = words[i].ToLower();
}

/// <summary>
/// Gets common words that appear in both files
/// </summary>
public static List<string> GetCommonWords(string[] words1, string[] words2)
{
    LowerCaseStrings(words1);
    LowerCaseStrings(words2);
    List<string> output = GetDuplicates(words1, words2);
    output.LengthSort();

    // Trims Count
    if (output.Count > 10)
        output.RemoveRange(10, output.Count - 10);
    return output;
}

/// <summary>
/// Gets repetition of words that are common in both files
/// </summary>
public static Dictionary<string, int> GetRepetition(List<string> commonWords, string[]
words)
{
    Dictionary<string,int> repetition = new Dictionary<string,int>();

    foreach (string word in words)
    {
        if (commonWords.Contains(word))
        {
            if (repetition.ContainsKey(word))

```

```

        repetition[word]++;
    else
        repetition.Add(word, 1);
    }
}

return repetition;
}

/// <summary>
/// Gets longest sentence
/// </summary>
public static string LongestSentence(string[] sentences)
{
    string longestSentence = "";
    foreach (string sentence in sentences)
        if (sentence.Length > longestSentence.Length)
            longestSentence = sentence;

    return longestSentence.Trim();
}

/// <summary>
/// Gets where the sentence starts (line)
/// </summary>
public static int GetSentenceStart(string text, string sentence)
{
    text = text.Remove(text.IndexOf(sentence));
    int line = text.Split('\r').Length;
    return line;
}

/// <summary>
/// Writes ManoKnyga.txt file
/// </summary>
public static string WriteBook(string text1, string text2)
{
    string main = text1;
    string other = text2;
    string output = "";
    while(main != "")
    {
        string word = "empty";
        word = Regex.Match(other, @"\w+", RegexOptions.IgnoreCase).Value;

        int index = main.IndexOf(word);
        if (index == -1)
        {
            output += main + " ";
            break;
        }
        else
        {
            // Removes Used up parts
            output += main.Substring(0, index);
            main = main.Remove(0, index + word.Length);
            Match match = Regex.Match(main, @"\w");
            if (match.Success)
                main = main.Remove(0, match.Index);
            else
                main = "";
        }

        // Swaps strings
        string temp = main;

```

```

        main = other;
        other = temp;
    }

    output += other;
    return output;
}
}
}

Program.cs:

using System;
using System.Text.RegularExpressions;
using System.Collections.Generic;

namespace Lab04
{
    internal class Program
    {
        static void Main(string[] args)
        {
            const string input1 = "Knyga1.txt";
            const string input2 = "Knyga2.txt";
            const string output = "Rodikliai.txt";

            char[] sentenceChar = { '!', '?', '.' };
            char[] punctuation = { ' ', '.', ',', '!', '?', ':', ';', '(', ')', '\t', '\r',
'\n', '\\', '"' };

            // Reads Data
            string text1 = InOut.ReadText(input1);
            string text2 = InOut.ReadText(input2);

            // Creates Sentence List
            string[] sentences1 = text1.Split(sentenceChar);
            string[] sentences2 = text2.Split(sentenceChar);

            // Creates word List
            string[] words1 = text1.Split(punctuation, StringSplitOptions.RemoveEmptyEntries);
            string[] words2 = text2.Split(punctuation, StringSplitOptions.RemoveEmptyEntries);

            // Creates Intial File
            InOut.CreateFile(output);

            // Common Word Count
            List<string> CommonWords = TaskUtils.GetCommonWords(words1, words2);
            InOut.WriteWordRepetitions(CommonWords,
                TaskUtils.GetRepetition(CommonWords, words1),
                output, $"{input1} Common Word Count:");

            InOut.WriteWordRepetitions(CommonWords,
                TaskUtils.GetRepetition(CommonWords, words2),
                output, $"{input2} Common Word Count:");

            // Sentences
            InOut.WriteLongestSentence(output,
                $"{input1} Longest Sentence Info:",
                sentences1,
                text1);

            InOut.WriteLongestSentence(output,
                $"{input2} Longest Sentence Info:",
                sentences2,
                text2);
        }
    }
}

```

```
        InOut.WriteString("ManoKnyga.txt", TaskUtils.WriteBook(text1, text2));
    }
}
```

4.3. Pradiniai duomenys ir rezultatai

Pradiniai duomenys 1:

Knyga1.txt:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.

Knyga2.txt:

nostrud Kitoksžodis ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.

Rezultatai:

Rodikliai.txt:

```
File Edit Format View Help
Knyga1.txt Common Word Count:
-----
      reprehenderit: 1
      exercitation: 1
      consequat: 1
      voluptate: 1
      excepteur: 1
      cupidatat: 1
      pariatur: 1
      occaecat: 1
      proident: 1
      deserunt: 1

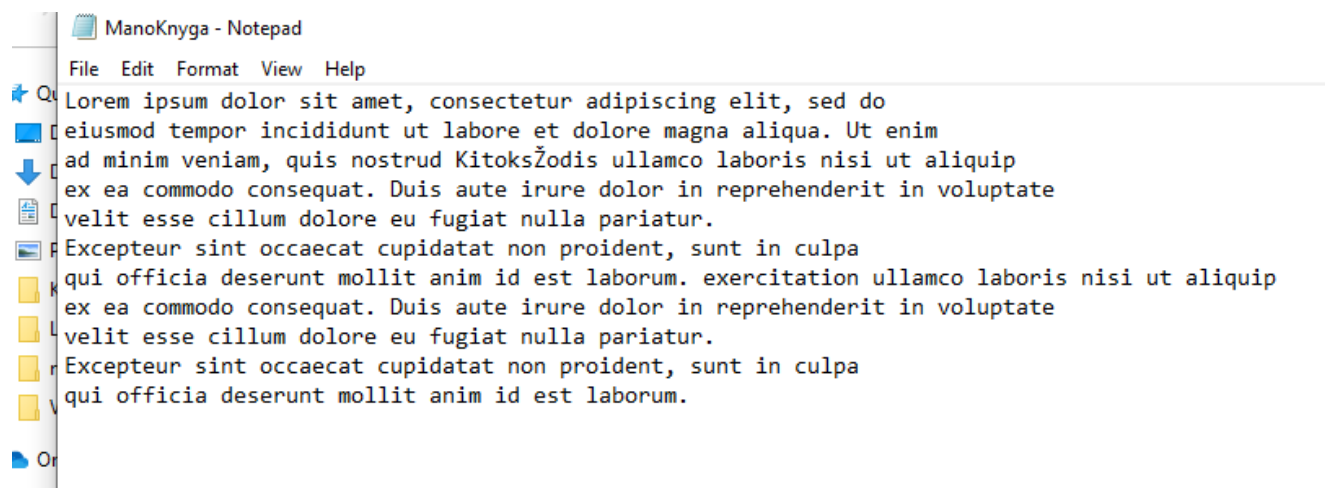
Knyga2.txt Common Word Count:
-----
      reprehenderit: 1
      exercitation: 1
      consequat: 1
      voluptate: 1
      excepteur: 1
      cupidatat: 1
      pariatur: 1
      occaecat: 1
      proident: 1
      deserunt: 1

Knyga1.txt Longest Sentence Info:
-----
Longest Sentence is: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua
Symbol Count: 124 Word Count: 19
Line Where the Sentence Starts: 1

Knyga2.txt Longest Sentence Info:
-----
Longest Sentence is: Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum
Symbol Count: 111 Word Count: 17
Line Where the Sentence Starts: 4

<
```

ManoKnyga.txt:



Pradiniai Duomenys 2:

Knyga1.txt:

1,,2,3?

4,5,,,,6,7

8,9,a.

Knyga2.txt:

3,,,ccee,,midline ddee,,eeee,,

4 ,ff,,gg,,7

Rezultatai:

Rodikliai.txt:

Knyga10.txt Common Word Count:

3: 1

4: 1

7: 1

Knyga20.txt Common Word Count:

3: 1

4: 1

7: 1

Knyga10.txt Longest Sentence Info:

Longest Sentence is: 4,5,,,6,7

8,9,a

Symbol Count: 16 Word Count: 7

Line Where the Sentence Starts: 1

Knyga20.txt Longest Sentence Info:

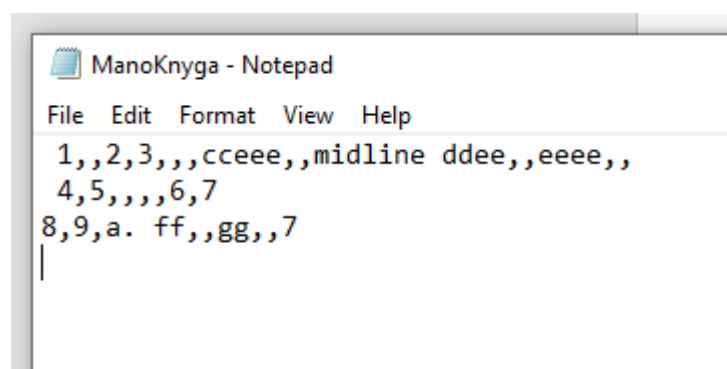
Longest Sentence is: 3,,,ccee,,midline ddee,,eeee,,

4 ,ff,,gg,,7

Symbol Count: 45 Word Count: 9

Line Where the Sentence Starts: 1

ManoKnyga.txt:



4.4. Dėstytojo pastabos

5. Paveldėjimas

5.1. Darbo užduotis

5.2. Programos tekstas

5.3. Pradiniai duomenys ir rezultatai

5.4. Dėstytojo pastabos