Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas I (P175B118)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Studentas

Lekt. Kęstutis Simonavičius

Dėstytojas

Kaunas 2021

TURINYS

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

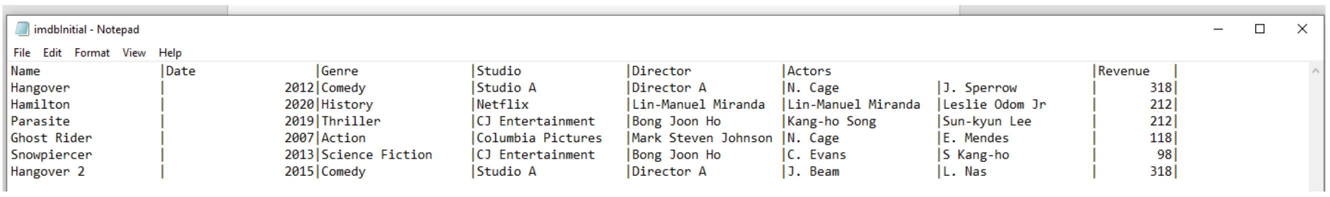
Pradiniai duomenys:

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
```

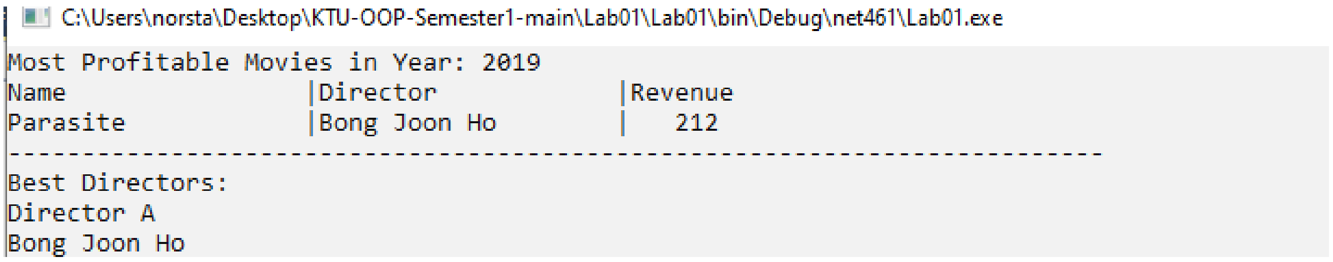
Rezultatai

imdbinitial.txt:



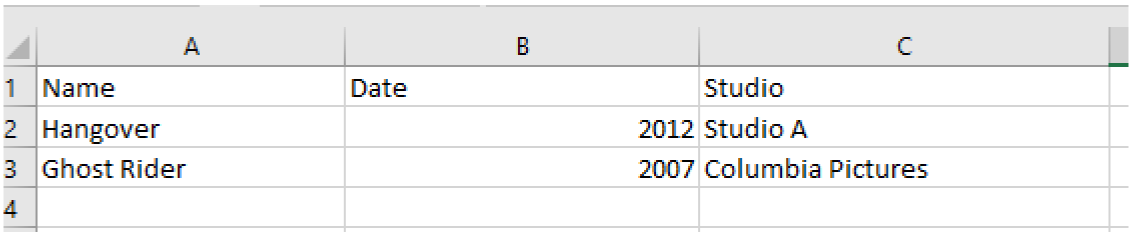
```
Name                |Date               |Genre              |Studio             |Director           |Actors                                   |Revenue   |
Hangover            |               2012|Comedy             |Studio A           |Director A         |N. Cage            |J. Sperrow           |       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|
```

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

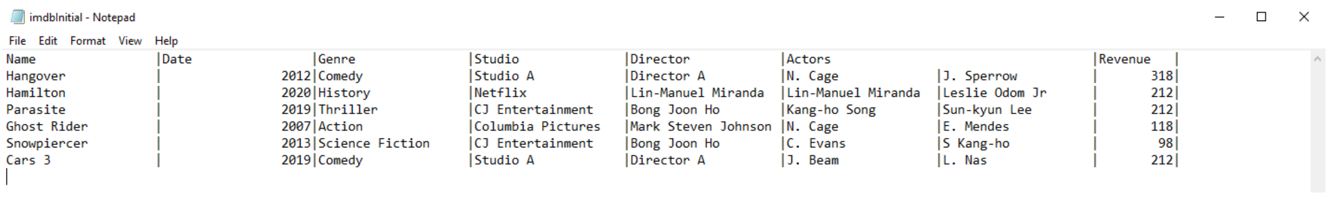
Pradiniai duomenys:

Imdb2.txt

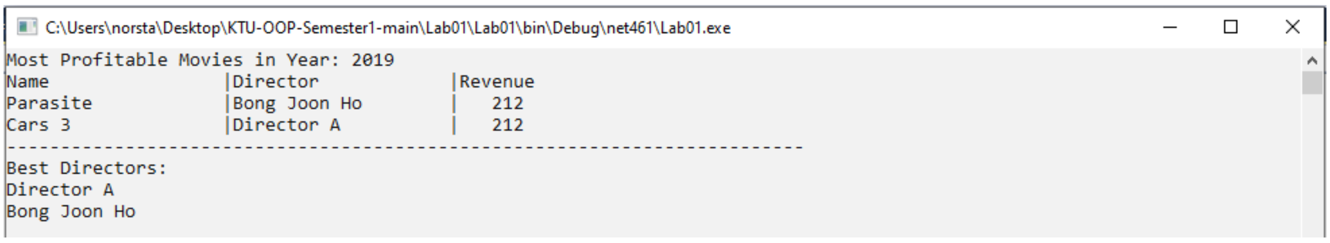
```
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom
Jr;212
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. 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:



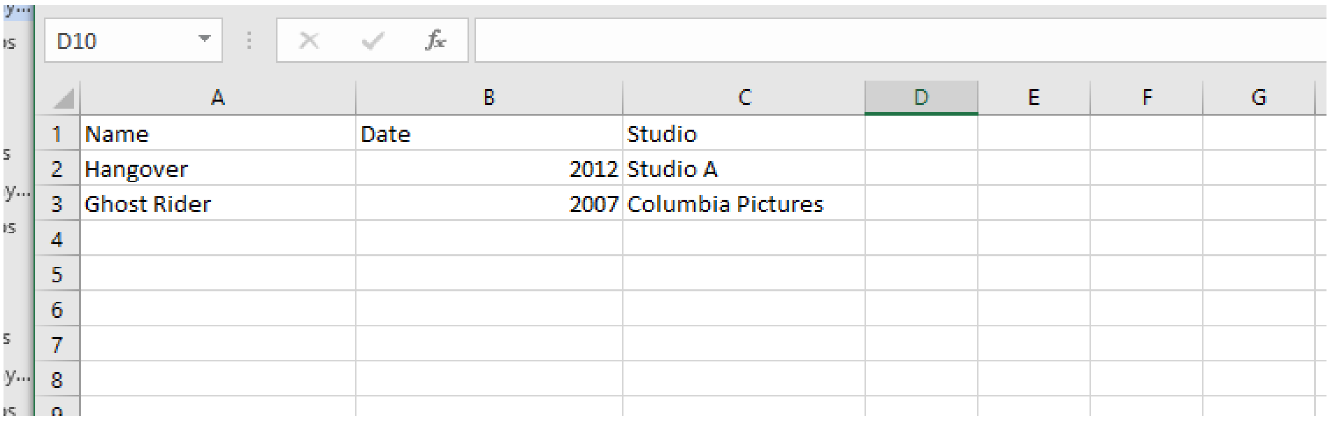
```
Name                |Date               |Genre              |Studio             |Director           |Actors                                   |Revenue   |
Hangover            |               2012|Comedy             |Studio A           |Director A         |N. Cage            |J. Sperrow           |       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|
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:
```



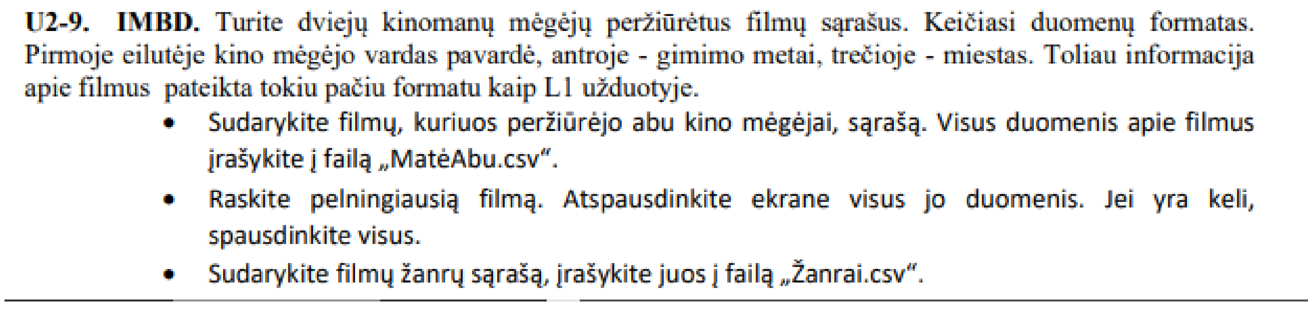
D10

	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							

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 catologue
        /// </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 catologue
        }

        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

    }
}
```

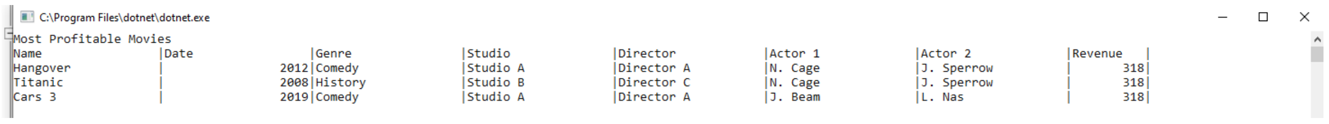
2.3. Pradiniai duomenys ir rezultatai

```
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

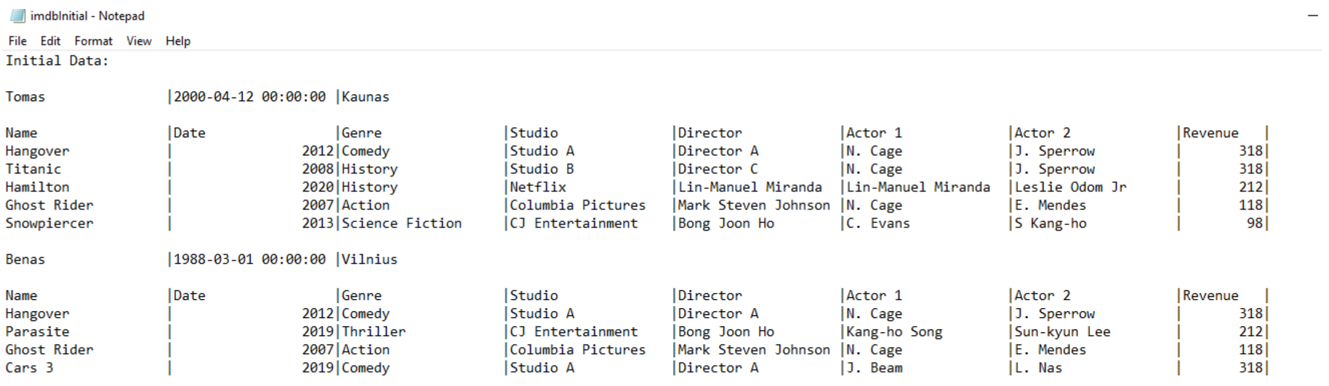
Rezultatai:

Console:



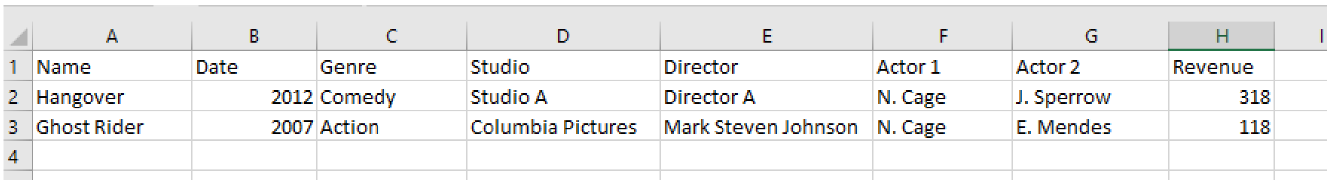
```
C:\Program Files\dotnet\dotnet.exe
Most Profitable Movies
Name                |Date                |Genre               |Studio              |Director            |Actor 1             |Actor 2             |Revenue   |
Hangover            |                2012|Comedy              |Studio A            |Director A          |N. Cage             |J. Sperrow          |       318|
Titanic             |                2008|History             |Studio B            |Director C          |N. Cage             |J. Sperrow          |       318|
Cars 3              |                2019|Comedy              |Studio A            |Director A          |J. Beam             |L. Nas              |       318|
```

imdbInitial.txt:



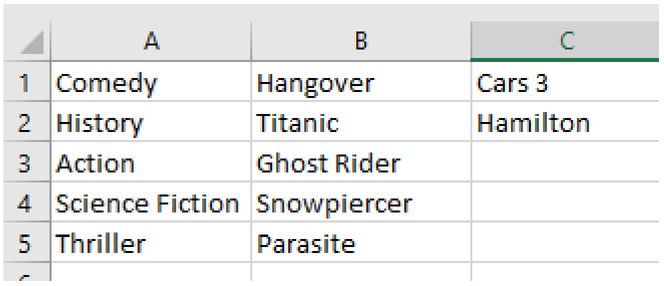
```
imdbInitial - Notepad
File  Edit  Format  View  Help
Initial Data:

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. 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|

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. 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|
```

MatėAbu.csv:



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

Žanrai.csv:



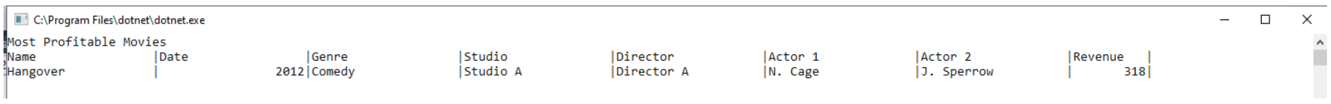
	A	B	C
1	Comedy	Hangover	Cars 3
2	History	Titanic	Hamilton
3	Action	Ghost Rider	
4	Science Fiction	Snowpiercer	
5	Thriller	Parasite	

Pradiniai Duomenys:

data2-1.txt:

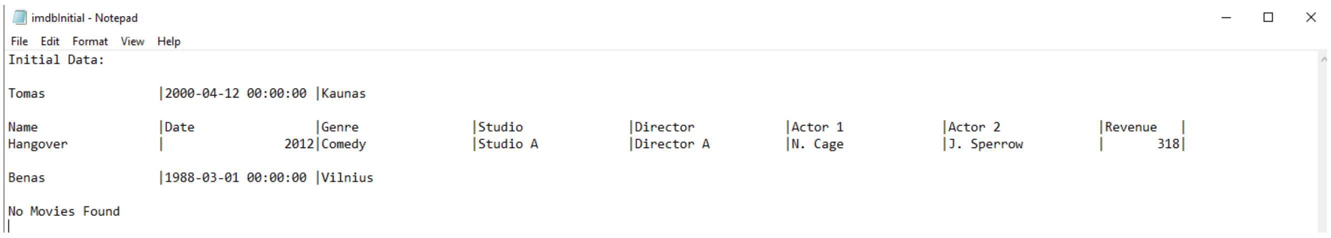
Tomas
2000-04-12
Kaunas
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318

```
data2-2.txt:

Benas
1988-03-01
Vilnius

Rezultatai:

Console:
```



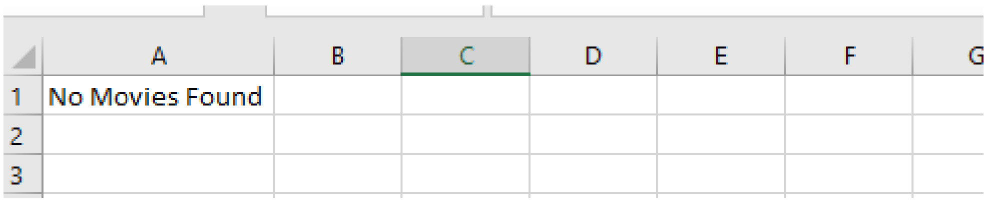
```
C:\Program Files\dotnet\dotnet.exe
Most Profitable Movies
Name                |Date                |Genre               |Studio              |Director            |Actor 1             |Actor 2             |Revenue   |
Hangover            |                2012|Comedy              |Studio A            |Director A          |N. Cage             |J. Sperrow          |       318|
```

```
imdbInitial.txt:
```



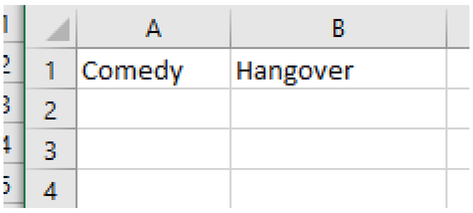
```
imdbInitial - Notepad
File  Edit  Format  View  Help
Initial Data:

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. Sperrow          |       318|

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

No Movies Found
```

```
MatėAbu.csv:
```



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

```
Žanrai.csv:
```



	A	B
1	Comedy	Hangover
2		
3		
4		

2.4. Dėstytojo pastabos

3. Konteineris

3.1. Darbo užduotis

3.2. Programos tekstas

3.3. Pradiniai duomenys ir rezultatai

3.4. Dėstytojo pastabos

4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

4.2. Programos tekstas

4.3. Pradiniai duomenys ir rezultatai

4.4. Dėstytojo pastabos

5. Paveldėjimas

5.1. Darbo užduotis

5.2. Programos tekstas

5.3. Pradiniai duomenys ir rezultatai

5.4. Dėstytojo pastabos