

Kauno technologijos universitetas

Informatikos fakultetas

Objektinis programavimas I (P175B118)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Lekt. Kęstutis Simonavičius

Kaunas 2021

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 67

5.4. Dėstytojo pastabos 72

# Duomenų klasė

## 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.

## 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();

}

}

}

## 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

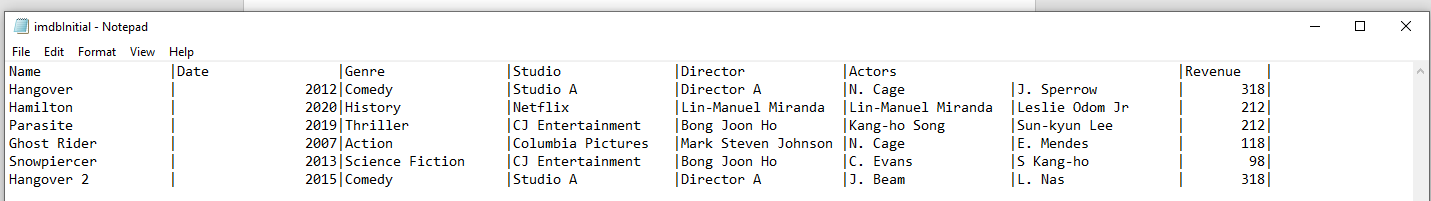
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes; 118

Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho; 98

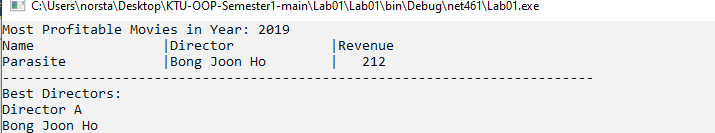
Hangover 2;2015;Comedy;Studio A;Director A;J. Beam;L. Nas;318

Rezultatai

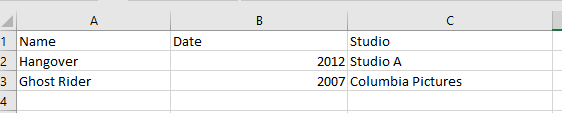
imdbinitial.txt:



Console:



MoviesWith.csv:



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

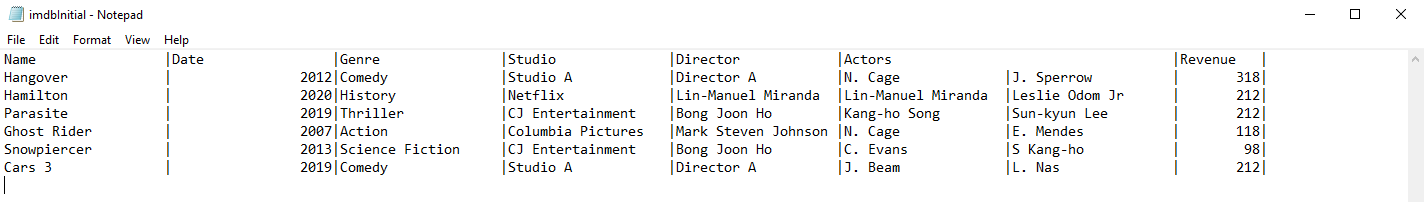
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes; 118

Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho; 98

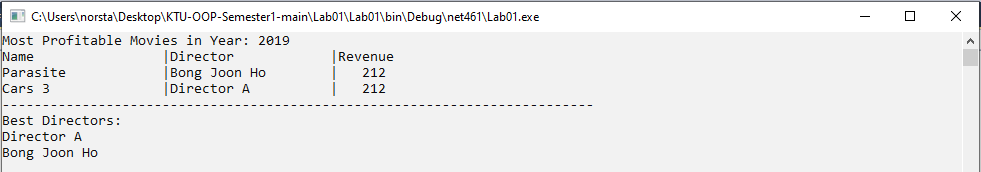
Cars 3;2019;Comedy;Studio A;Director A;J. Beam;L. Nas;212

Rezultatai:

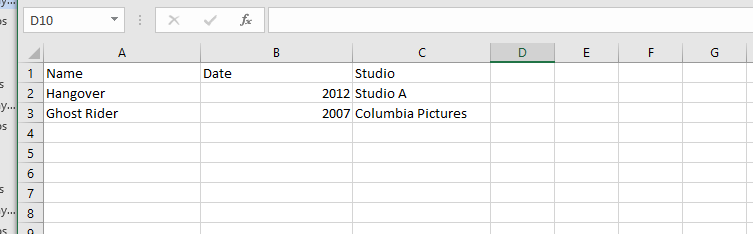
ImdbInitial.txt:



Console:



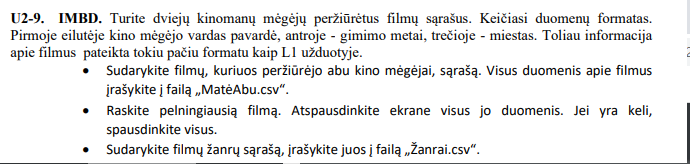
MoviesWith.csv:



## Dėstytojo pastabos

# Skaičiavimų klasė

## Darbo užduotis



## 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

}

}

## 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

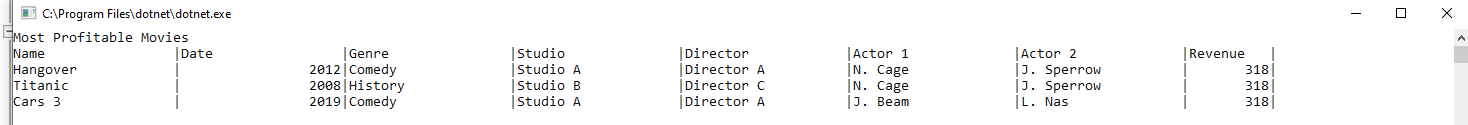
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212

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:

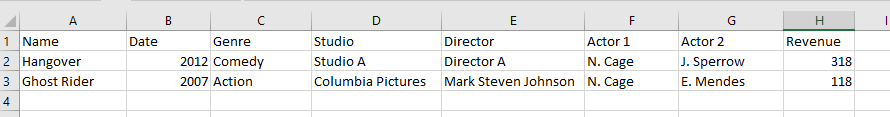
Console:



imdbInitial.txt:



MatėAbu.csv:



Žanrai.csv:



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:

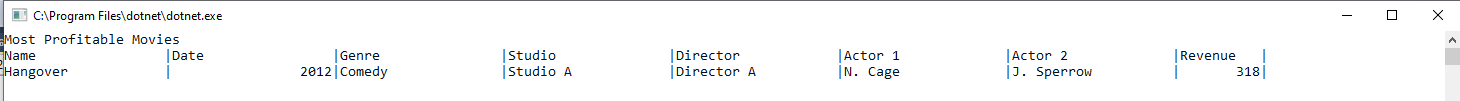
Benas

1988-03-01

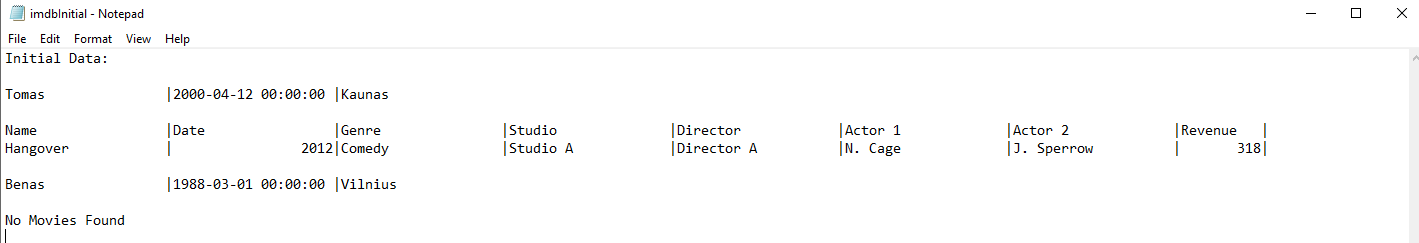
Vilnius

Rezultatai:

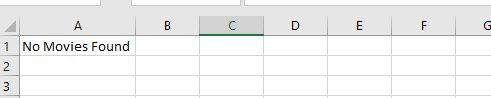
Console:



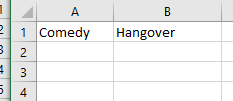
imdbInitial.txt:



MatėAbu.csv:



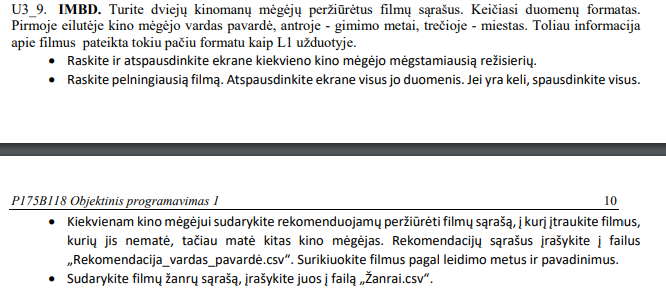
Žanrai.csv:



## Dėstytojo pastabos

# Konteineris

## Darbo užduotis



## 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 idmb)

{

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

if (this.Movies[i].Equals(idmb))

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 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

}

/// <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>

/// Reccomends 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 CDoutput = @"imdbInitial.txt";

const string CDGenres = @"Žanrai.csv";

static void Main(string[] args)

{

InOutHelpers.CreateOutputFile(CDoutput);

User user1 = InOutHelpers.ReadUser(CDdata1);

user1.WriteInitialData(CDoutput);

User user2 = InOutHelpers.ReadUser(CDdata2);

user2.WriteInitialData(CDoutput);

// 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();

}

}

}

## 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

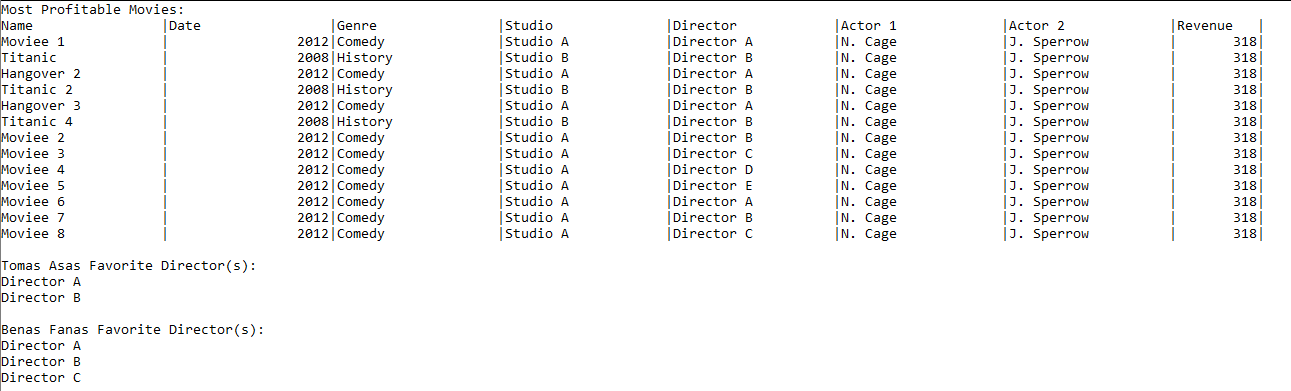
Moviee 6;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318

Moviee 7;2012;Comedy;Studio A;Director B;N. Cage;J. Sperrow;318

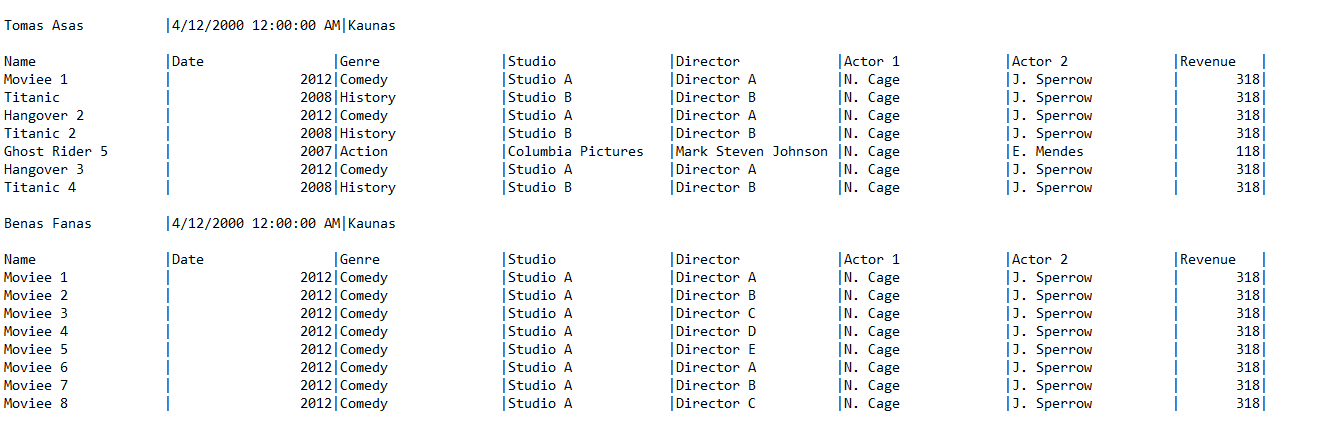
Moviee 8;2012;Comedy;Studio A;Director C;N. Cage;J. Sperrow;318

Rezultatai:

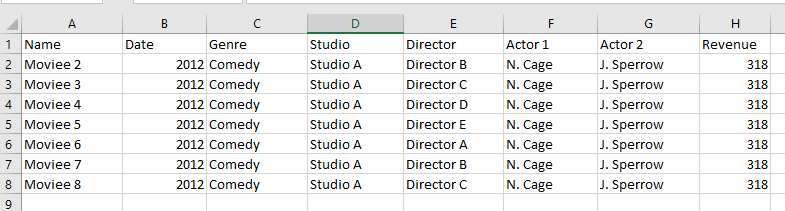
Console:



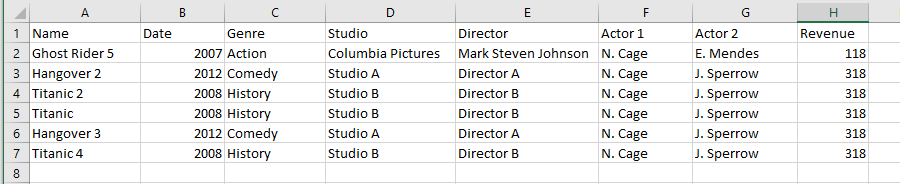
imdbInitial.txt:



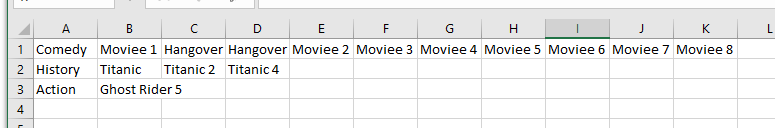
Rekomendacija\_Tomas\_Asas.cs:



Rekomendacija\_Benas\_Fanas.cs:



Žarai.cs:



Pradiniai Duomenys:

data2-1.txt:

Tomas Aputis

2000-04-12

Kaunas

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

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

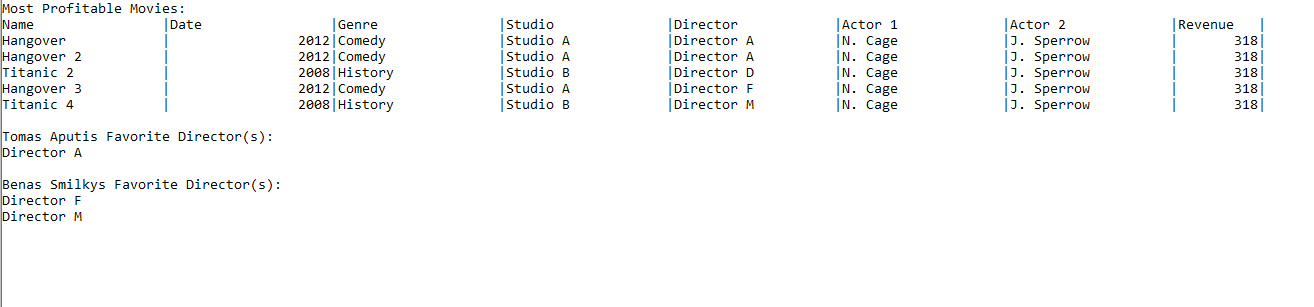
Titanic 4;2008;History;Studio B;Director M;N. Cage;J. Sperrow;318

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:

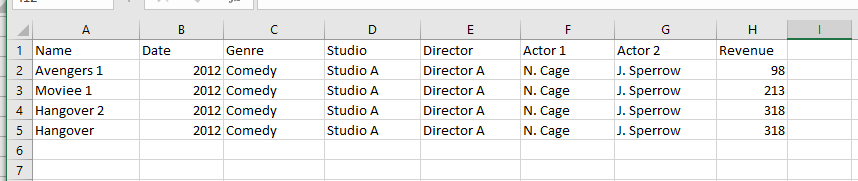
Console:



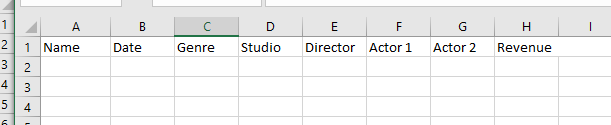
imdbInitial.txt:



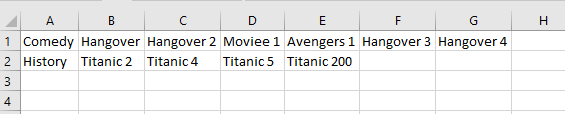
Rekomendacija\_Benas\_Smilkys.cs:



Rekomendacija\_Tomas\_Aputis.cs:



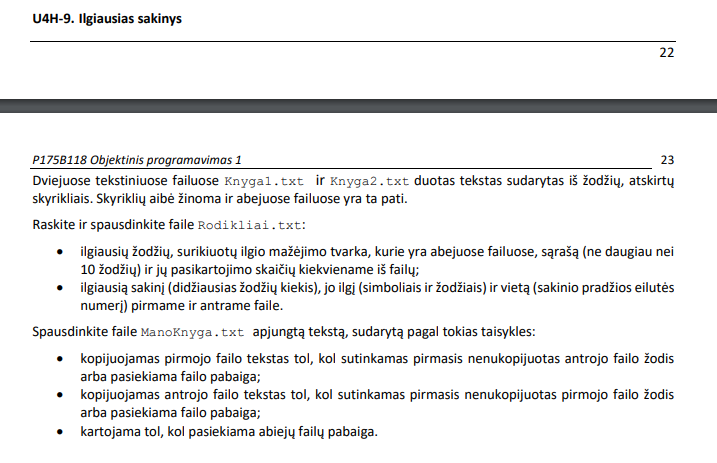
Žanrai.cs:



## Dėstytojo pastabos

# Teksto analizė ir redagavimas

## Darbo užduotis



## 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', '\r', '\n', '\'', '"' };

/// <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));

}

}

}

## 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

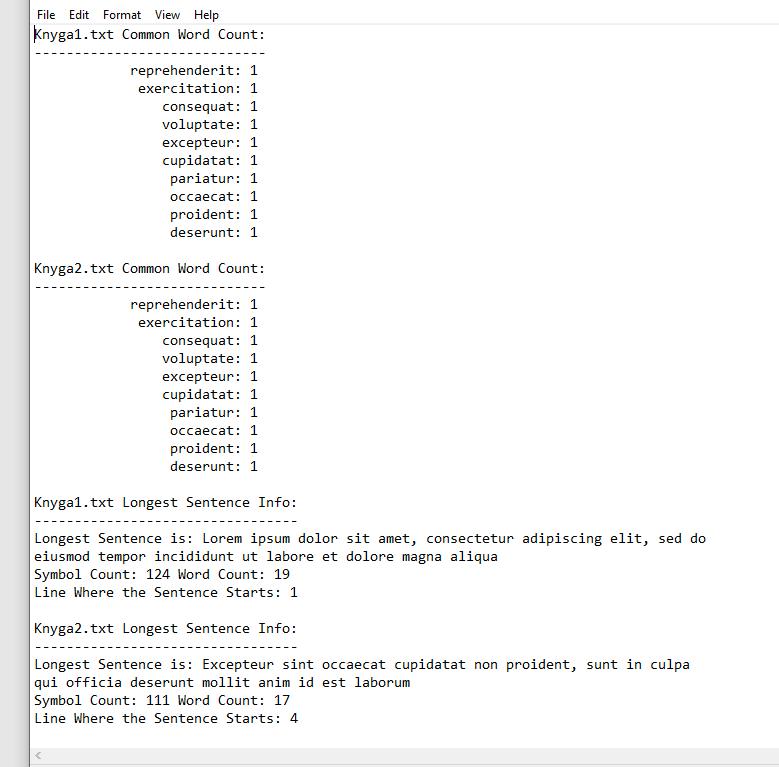
velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa

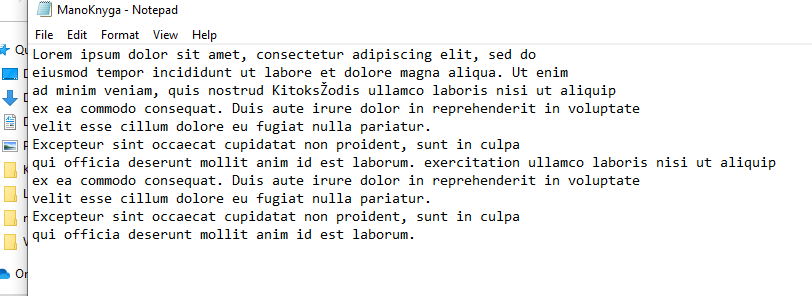
qui officia deserunt mollit anim id est laborum.

Rezultatai:

Rodikliai.txt:



ManoKnyga.txt:



Pradiniai Duomenys 2:

Knyga1.txt:

1,,2,3?

4,5,,,,6,7

8,9,a.

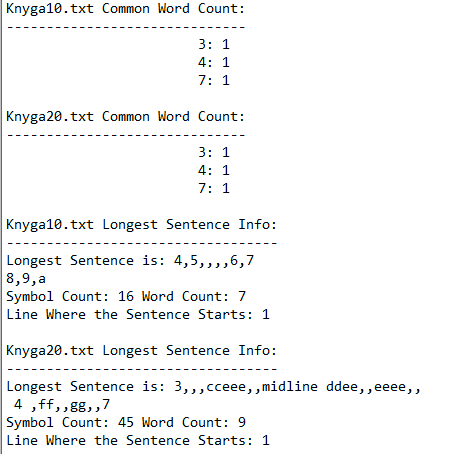
Knyga2.txt:

3,,,cceee,,midline ddee,,eeee,,

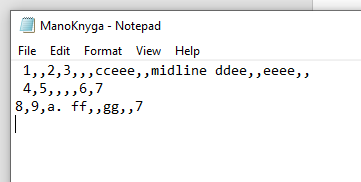
4 ,ff,,gg,,7

Rezultatai:

Rodikliai.txt:



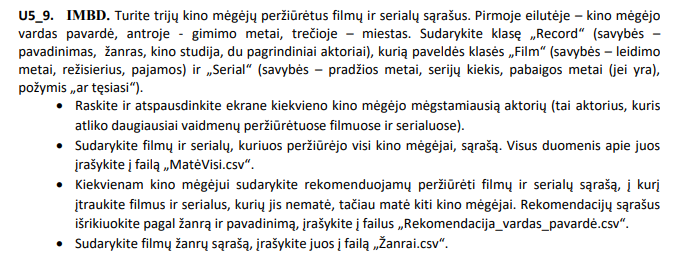
ManoKnyga.txt:



## Dėstytojo pastabos

# Paveldėjimas

## Darbo užduotis



## Programos tekstas

Record.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace Lab05

{

/// <summary>

/// IMDB Movie Object

/// </summary>

public abstract class Record

{

public string Name { get; set; }

public string Genre { get; set; }

public string Studio { get; set; }

public List<string> Actors { get; set; }

/// <summary>

/// Constructor

/// </summary>

public Record(string name,

string genre,

string studio,

string actor1,

string actor2)

{

Name = name;

Genre = genre;

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 == ((Record)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 virtual string ToString(char splitter)

{

return $"{this.Name,-20}{splitter}" +

$"{this.Genre,-20}{splitter}" +

$"{this.Studio,-20}{splitter}" +

$"{this.Actors[0],-20}{splitter}" +

$"{this.Actors[1],-20}{splitter}";

}

/// <summary>

/// Compare To Method implementation

/// </summary>

internal int CompareTo(Record record)

{

int comparison = Genre.CompareTo(record.Genre);

if (comparison == 0)

comparison = Name.CompareTo(record.Name);

return comparison;

}

}

}

Film.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab05

{

/// <summary>

/// Film Class, Inherited from Record

/// </summary>

class Film : Record

{

public int RealeaseDate { get; set; }

public string Director { get; set; }

public int Revenue { get; set; }

/// <summary>

/// Constructor

/// </summary>

public Film(string name, string genre, string studio, string actor1, string actor2, int date, string director, int revenue) : base(name, genre, studio, actor1, actor2)

{

RealeaseDate = date;

Director = director;

Revenue = revenue;

}

/// <summary>

/// ToString() Override

/// </summary>

public override string ToString()

{

return ToString('|');

}

/// <summary>

/// ToString(char splitter) Override

/// </summary>

public override string ToString(char splitter)

{

string output = $"{"Film",-20}{splitter}" +

base.ToString(splitter) +

$"{RealeaseDate,-20}{splitter}" +

$"{Director, -20}{splitter}" +

$"{Revenue, 20}{splitter}";

return output;

}

}

}

Serial.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab05

{

/// <summary>

/// Serial Class inherited from record claass.

/// Meant to be used for Tv Shows (Serialized) media

/// </summary>

class Serial : Record

{

public bool StillGoing { get; set; }

public int StartYear { get; set; }

public int EndYear { get; set; }

public int EpisodeCount { get; set; }

/// <summary>

/// Constructor for a serial that ended

/// </summary>

public Serial(string name, string genre, string studio, string actor1, string actor2, int episodeCount, int startYear, int endYear, bool status) : base(name, genre, studio, actor1, actor2)

{

EpisodeCount = episodeCount;

StartYear = startYear;

EndYear = endYear;

StillGoing = status;

}

/// <summary>

/// Constructor for a serial that is still airing

/// </summary>

public Serial(string name, string genre, string studio, string actor1, string actor2, int episodeCount, int startYear, bool status) : base(name, genre, studio, actor1, actor2)

{

EpisodeCount = episodeCount;

StartYear = startYear;

StillGoing = status;

}

/// <summary>

/// ToString() override

/// </summary>

public override string ToString()

{

return ToString('|');

}

/// <summary>

/// ToString(char splitter) from class Record override

/// </summary>

public override string ToString(char splitter)

{

string output = $"{"Serial",-20}{splitter}" +

base.ToString(splitter) +

$"{EpisodeCount,20}{splitter}" +

$"{StartYear,20}{splitter}" +

$"{StillGoing,-20}{splitter}";

if (StillGoing == false)

output += $"{EndYear,-20}{splitter}";

return output;

}

}

}

IMDBContainer.cs:

using System;

using System.Text;

namespace Lab05

{

/// <summary>

/// IMDB Container Code

/// </summary>

class IMDBContainer

{

private Record[] Movies;

public int Count { get; private set; }

public int Capacity { get; set; }

public IMDBContainer(int capacity = 16)

{

this.Movies = new Record[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(Record 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 Record Get(int index)

{

return this.Movies[index];

}

/// <summary>

/// Contains Implementation

/// </summary>

public bool Contains(Record idmb)

{

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

if (this.Movies[i].Equals(idmb))

return true;

return false;

}

/// <summary>

/// Ensure Capacity Implementation

/// </summary>

private void EnsureCapacity(int minimumCapacity)

{

if (minimumCapacity > this.Capacity)

{

Record[] temp = new Record[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 Record Put(Record imdb, int index)

{

index = CheckIndex(index);

if (index == Count)

{

if (this.Count == this.Capacity) //container is full

{

EnsureCapacity(this.Capacity \* 2);

}

Count++;

}

Record otherDog = Movies[index];

Movies[index] = imdb;

return otherDog;

}

/// <summary>

/// Insert Implementation

/// </summary>

public Record Insert(Record 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(Record 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(Record 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>

/// Clears the Container

/// </summary>

public void Clear(int capacity = 16)

{

this.Movies = new Record[capacity];//default capacity

Capacity = capacity;

Count = 0;

}

/// <summary>

/// ToString implementation

/// </summary>

public override string ToString()

{

return $"Element Count: {Count} Element Capacity: {Capacity}";

}

/// <summary>

/// Bubble Sort

/// </summary>

/// <returns></returns>

public IMDBContainer Sort()

{

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

{

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

if(Movies[j].CompareTo(Movies[j+1]) > 0)

{

// SWAP

Record temp = Movies[j];

Movies[j] = Movies[j + 1];

Movies[j+1] = temp;

}

}

return this;

}

}

}

User.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab05

{

/// <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; }

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 catologue

/// </summary>

public void AddMovie(Record imdb)

{

Record 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

}

/// <summary>

/// Returns MovieCount

/// </summary>

public int GetMovieCount()

{

return movies.Count;

}

public Record 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>

/// Returns All Actors in a List

/// </summary>

private List<string> GetAllActors()

{

List<string> actors = new List<string>();

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

{

Record record = movies.Get(i);

actors.Add(record.Actors[0]);

if(record.Actors[1] != record.Actors[0]) // Checks for duplicates

actors.Add(record.Actors[1]);

}

return actors;

}

/// <summary>

/// GetsFavorite Director for provided User

/// </summary>

public List<string> GetFavoriteActors()

{

List<string> favoriteActors = new List<string>();

int moviesInMax = 0;

List<string> actors = GetAllActors();

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

{

string currActor = actors[i];

int moviesIn = 0;

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

if (actors[j] == currActor)

moviesIn++;

// Resets

if (moviesIn > moviesInMax)

{

moviesInMax = moviesIn;

favoriteActors.Clear();

}

// Adds users

if (moviesInMax == moviesIn)

{

favoriteActors.Add(currActor);

}

}

return favoriteActors;

}

}

}

AllMovieInfo.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab05

{

/// <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, Record> MovieTitleSearch;

private static Dictionary<string, IMDBContainer> GenreSearch;

private static Dictionary<Record, Dictionary<User, bool>> MovieUsers; // First Key -> Movie, Second Key - User, Returns if the User Has seen the movie

/// <summary>

/// AllMovieInfo Static Constructor

/// </summary>

static AllMovieInfo()

{

AllMovies = new IMDBContainer();

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

MovieTitleSearch = new Dictionary<string, Record>();

MovieUsers = new Dictionary<Record, Dictionary<User, bool>>();

GenreSearch = new Dictionary<string, IMDBContainer>();

}

/// <summary>

/// Returns reccomended Movies

/// </summary>

public static IMDBContainer GetReccomendedMovies(User user)

{

IMDBContainer output = new IMDBContainer();

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

{

Record imdb = AllMovies.Get(i);

if (MovieUsers[imdb].ContainsKey(user) == false)

output.Add(imdb);

}

return output.Sort();

}

/// <summary>

/// Gets Movies that all users have seen

/// </summary>

internal static List<Record> GetAllSeen(List<User> users)

{

List<Record> allSeen = new List<Record>();

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

{

Record record = AllMovies.Get(i);

if(HasSeen(users, record))

allSeen.Add(record);

}

return allSeen;

}

/// <summary>

/// Checks if the selected users have seen the provided record

/// </summary>

private static bool HasSeen(List<User> users, Record record)

{

foreach (User user in users)

if (MovieUsers[record].ContainsKey(user) == false)

return false;

return true;

}

/// <summary>

/// Adds the movie to the AllMovieInfo Class. Adds the User who has seen the movie

/// </summary>

public static void AddMovie(Record 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 Record 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(Record 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(Record imdb)

{

MovieTitleSearch.Add(imdb.Name, imdb);

AddToGenre(imdb);

AllMovies.Add(imdb);

}

/// <summary>

/// Adds User as a person who has seen the movie

/// </summary>

private static void AddUser(Record 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++)

{

Record imdb = user1.GetMovieByIndex(i);

if (MovieUsers[imdb].ContainsKey(user2))

output.Add(imdb);

}

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.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab05

{

/// <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>

/// InfoString for Console or .txt files

/// </summary>

private static string InfoStringText(char splitter)

{

return $"{"Type",tSize}{splitter}" +

$"{"Name",tSize}{splitter}" +

$"{"Genre",tSize}{splitter}" +

$"{"Studio",tSize}{splitter}" +

$"{"Actor 1",tSize}{splitter}" +

$"{"Actor 2",tSize}{splitter}" +

$"{"Episodes/Date",tSize}{splitter}" +

$"{"Start Year/Director",tSize}{splitter}" +

$"{"Status/Revenue",tSize}{splitter}" +

$"{"End Year (Serial)",tSize}{splitter}";

}

/// <summary>

/// InfoString used for CSV files for easy access

/// </summary>

private static string InfoStringCSV(char splitter)

{

return $"{"Type"}{splitter}" +

$"{"Name"}{splitter}" +

$"{"Genre"}{splitter}" +

$"{"Studio"}{splitter}" +

$"{"Actor 1"}{splitter}" +

$"{"Actor 2"}{splitter}" +

$"{"Episodes/Date"}{splitter}" +

$"{"Start Year/Director"}{splitter}" +

$"{"Status/Revenue"}{splitter}" +

$"{"End Year (Serial)"}{splitter}";

}

/// <summary>

/// Writes movies that all users have seen

/// </summary>

internal static void AllSeen(List<User> users, string fileOutput)

{

List<Record> allSeen = AllMovieInfo.GetAllSeen(users);

using (StreamWriter sw = new StreamWriter(fileOutput))

{

sw.WriteLine(InfoStringCSV(';'));

if (allSeen.Count > 0)

foreach (Record record in allSeen)

sw.WriteLine(record.ToString(';'));

else

sw.WriteLine("No Data Found");

}

}

/// <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.ToShortDateString(),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(InfoStringText(splitter));

for (int i = 0; i < user.GetMovieCount(); i++)

{

Record record = user.GetMovieByIndex(i);

sw.WriteLine(record);

}

}

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(";".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

string type = data[0].ToLower();

string name = data[1];

string genre = data[2];

string studio = data[3];

string actor1 = data[4];

string actor2 = data[5];

switch (type)

{

case "f": // FILM

int date = int.Parse(data[6]);

string director = data[7];

int revenue = int.Parse(data[8]);

user.AddMovie(new Film(name, genre, studio, actor1, actor2, date, director, revenue));

break;

case "s": // SERIAL

int episodeCount = int.Parse(data[6]);

int startYear = int.Parse(data[7]);

bool status = bool.Parse(data[8]);

if (status == false)

{

int endYear = int.Parse(data[7]);

user.AddMovie(new Serial(name, genre, studio, actor1, actor2, episodeCount, startYear, endYear, status));

}

else

user.AddMovie(new Serial(name, genre, studio, actor1, actor2, episodeCount, startYear, status));

break;

}

}

}

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++)

{

Record imdb = genreCollection.Get(i);

sw.Write($";{imdb.Name}");

}

sw.WriteLine();

}

}

else

sw.WriteLine("No Data Found");

}

}

/// <summary>

/// Reccomends 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(InfoStringCSV(splitter));

IMDBContainer reccomendedMovies = AllMovieInfo.GetReccomendedMovies(user);

if (reccomendedMovies.Count > 0)

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

sw.WriteLine(reccomendedMovies.Get(i).ToString(';'));

else

sw.WriteLine("No Data Found");

}

}

/// <summary>

/// Prints Favorite Actors

/// </summary>

public static void PrintFavoriteActors(this User user)

{

Console.WriteLine($"{user.Name} Favorite actors are: ");

List<string> actors = user.GetFavoriteActors();

if (actors.Count > 0)

foreach (string actor in actors)

Console.WriteLine(actor);

else

Console.WriteLine("No Actor Found");

Console.WriteLine();

}

}

}

Program.cs:

using System;

using System.Collections.Generic;

namespace Lab05

{

internal class Program

{

static void Main(string[] args)

{

// Paths:

const string FD1 = "data1-1.txt";

const string FD2 = "data1-2.txt";

const string FD3 = "data1-3.txt";

const string FOgenres = "Žanrai.csv";

const string FOseenAll = "MatėVisi.csv";

const string FOmain = "output.txt";

// Reads Initial Data and Rewrites initial Data

InOutHelpers.CreateOutputFile(FOmain);

List<User> users = new List<User>();

users.Add(InOutHelpers.ReadUser(FD1));

InOutHelpers.WriteInitialData(users[0], FOmain);

users.Add(InOutHelpers.ReadUser(FD2));

InOutHelpers.WriteInitialData(users[1], FOmain);

users.Add(InOutHelpers.ReadUser(FD3));

InOutHelpers.WriteInitialData(users[2], FOmain);

// T1 Prints User's favorite actor/actors

users[0].PrintFavoriteActors();

users[1].PrintFavoriteActors();

users[2].PrintFavoriteActors();

//users[3].PrintFavoriteActors();

// T2 Seen Both

InOutHelpers.AllSeen(users, FOseenAll);

// T3 Reccomendation csv Files

InOutHelpers.ReccomendMovies(users[0]);

InOutHelpers.ReccomendMovies(users[1]);

InOutHelpers.ReccomendMovies(users[2]);

//InOutHelpers.ReccomendMovies(users[3]);

// T4 Outputs Genres

InOutHelpers.OutputGenres(FOgenres);

}

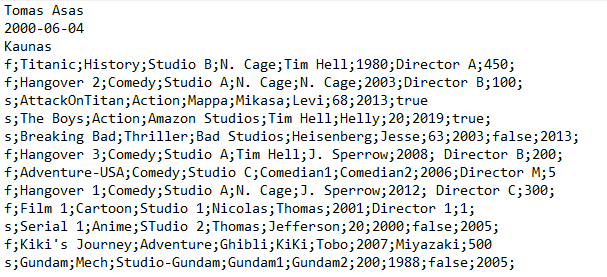
}

}

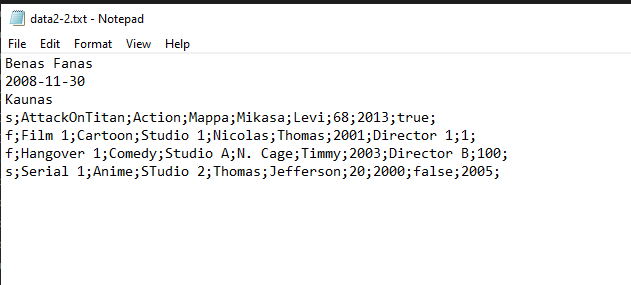
## Pradiniai duomenys ir rezultatai

Pradiniai duomenys:

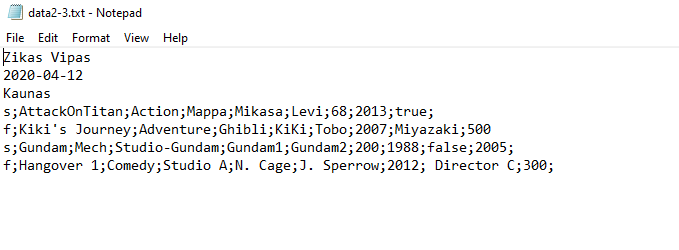
Data2-1.txt:



Data2-2.txt:

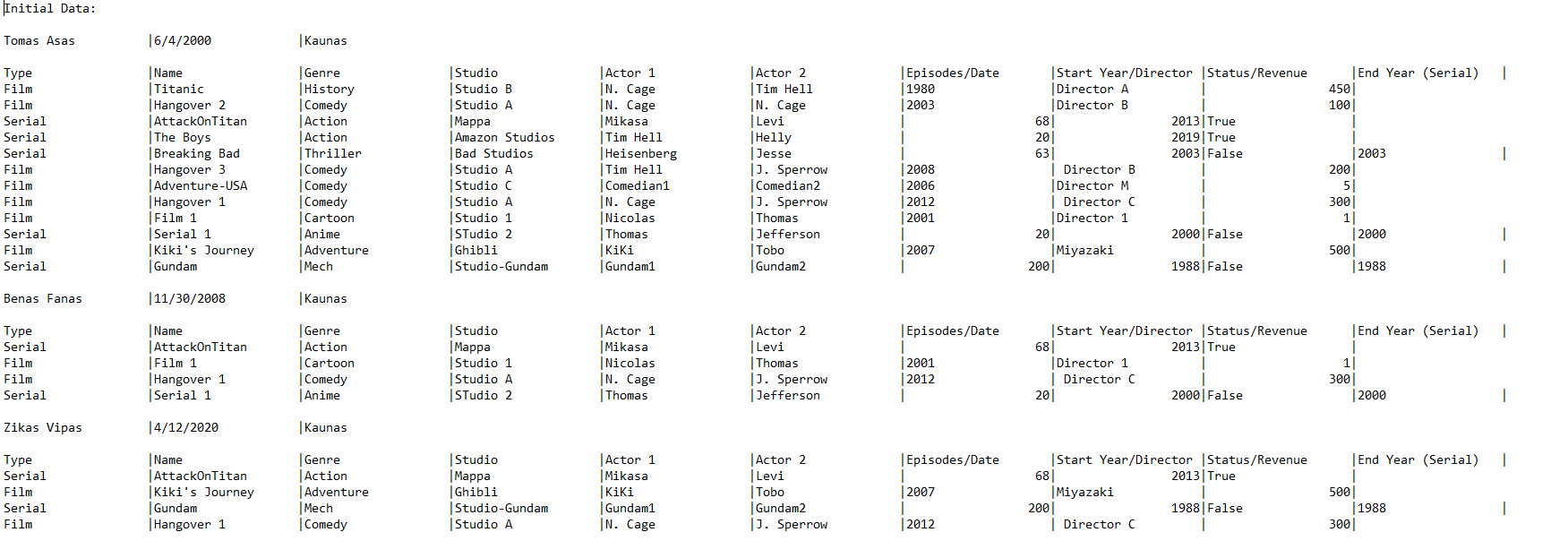


Data2-3.txt:

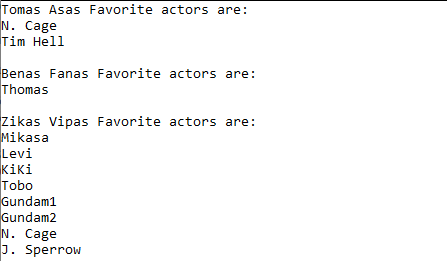


Rezultatai:

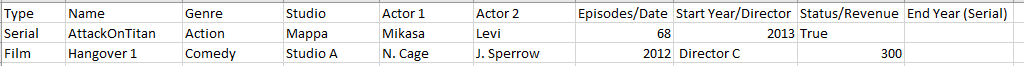
Output.txt:



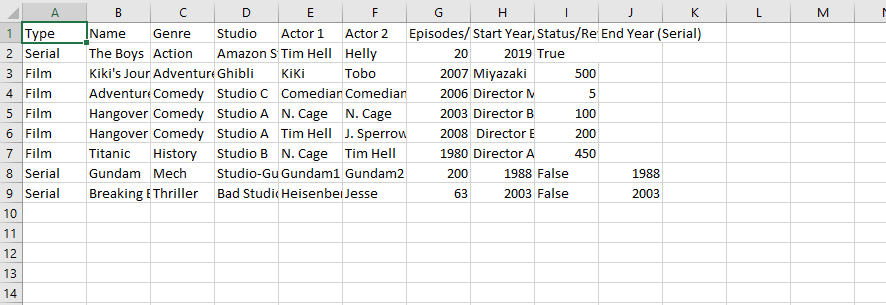
Console:



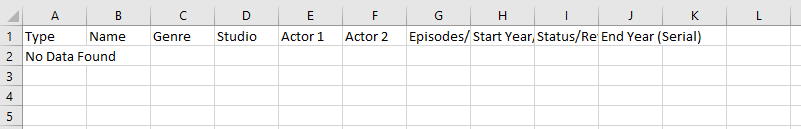
MatėVisi.csv:



Rekomendacija\_Benas\_Fanas.csv:



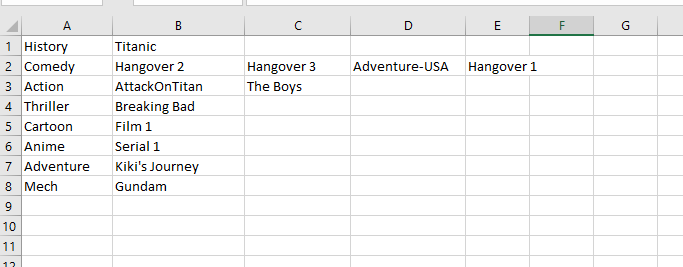
Rekomendacija\_Tomas\_Asas.csv:



Rekomendacija\_Zikas\_Vipas.csv:



Žanrai.csv:



Pradiniai Duomenys 2:

Data2-1.txt (3 Kartus tas pats):

Tomas Asas

2000-06-04

Kaunas

f;Titanic;History;Studio B;N. Cage;Tim Hell;1980;Director A;450;

f;Hangover 2;Comedy;Studio A;N. Cage;N. Cage;2003;Director B;100;

s;AttackOnTitan;Action;Mappa;Mikasa;Levi;68;2013;true

s;The Boys;Action;Amazon Studios;Tim Hell;Helly;20;2019;true;

s;Breaking Bad;Thriller;Bad Studios;Heisenberg;Jesse;63;2003;false;2013;

f;Hangover 3;Comedy;Studio A;Tim Hell;J. Sperrow;2008; Director B;200;

f;Adventure-USA;Comedy;Studio C;Comedian1;Comedian2;2006;Director M;5

f;Hangover 1;Comedy;Studio A;N. Cage;J. Sperrow;2012; Director C;300;

f;Film 1;Cartoon;Studio 1;Nicolas;Thomas;2001;Director 1;1;

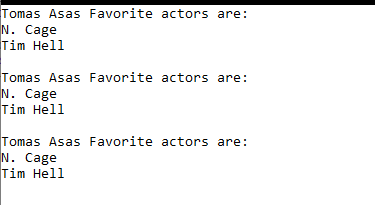
s;Serial 1;Anime;STudio 2;Thomas;Jefferson;20;2000;false;2005;

f;Kiki's Journey;Adventure;Ghibli;KiKi;Tobo;2007;Miyazaki;500

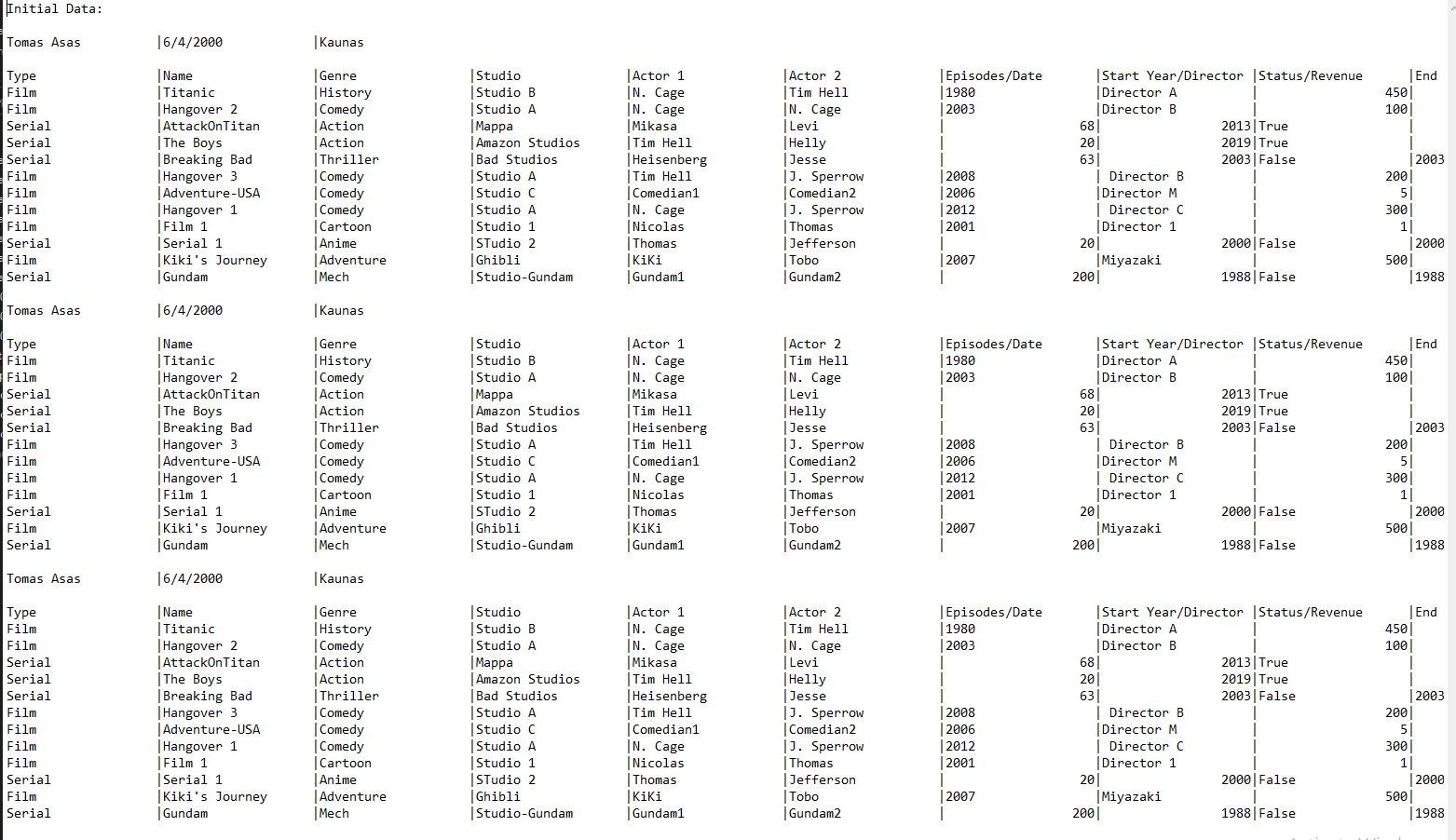
s;Gundam;Mech;Studio-Gundam;Gundam1;Gundam2;200;1988;false;2005;

Rezultatai:

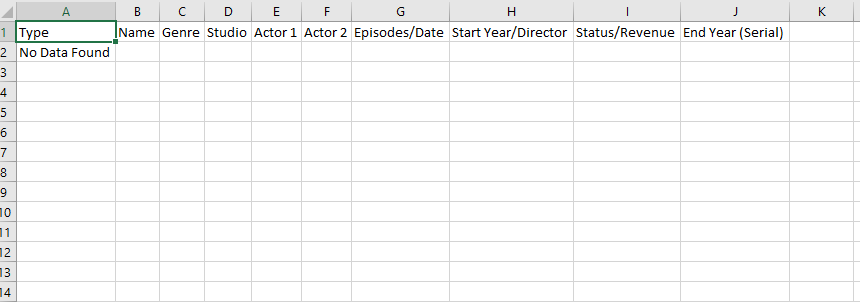
Console:



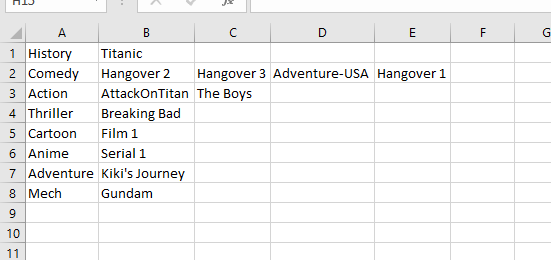
Output.txt:



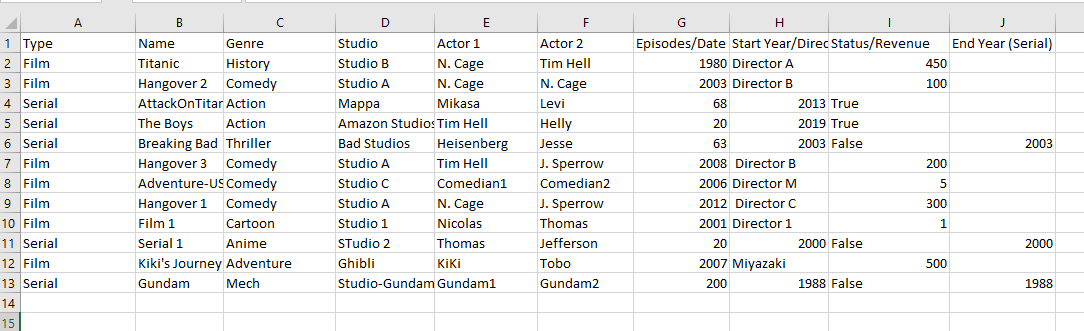
Rekomendacija\_Tomas\_Asas.csv:



Žanrai.csv:



MatėVisi.csv:



## Dėstytojo pastabos