



Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas I (P175B118)

Laboratorinių darbų ataskaita

Vytenis Kriščiūnas IFF-1/1

Studentas

Docentas Giedrius Ziberkas

Dėstytojas

TURINYS

1. Duomenų klasė.....	4
1.1. Darbo užduotis	4
1.2. Programos tekstas.....	4
1.3. Pradiniai duomenys ir rezultatai.....	9
1.4. Dėstytojo pastabos.....	11
2. Skaičiavimų klasė	12
2.1. Darbo užduotis	12
2.2. Programos tekstas.....	12
2.3. Pradiniai duomenys ir rezultatai.....	22
2.4. Dėstytojo pastabos.....	26
3. Konteineris.....	27
3.1. Darbo užduotis	27
3.2. Programos tekstas.....	27
3.3. Pradiniai duomenys ir rezultatai.....	43
3.4. Dėstytojo pastabos.....	46
4. Teksto analizė ir redagavimas	47
4.1. Darbo užduotis	47
4.2. Programos tekstas.....	47
4.3. Pradiniai duomenys ir rezultatai.....	52
4.4. Dėstytojo pastabos.....	54

Paveikslų sąrašas

1 pav. Atspausdinti rezultatai ekrane	10
2 pav. Atspausdinti rezultatai ekrane	11
3 pav. Atspausdinti rezultatai ekrane	24
4 pav. Atspausdinti rezultatai ekrane	26
5 pav. Atspausdinti rezultatai ekrane	44
6 pav. Atspausdinti rezultatai ekrane	46

1. Duomenų klasė

1.1. Darbo užduotis

U1-13. Krepšinio rinktinė. Artėja Pasaulio vyrų krepšinio čempionatas. Turime į rinktinės stovyklą pakviestų kandidatų sąrašą. Duomenų faile pateikiama informacija apie pakviestus krepšininkus: vardas, pavardė, gimimo data, ūgis, pozicija, klubas, požymis „pakviestas“, požymis „kapitonas“ (true, false).

- Raskite jauniausią į rinktinę pakviestą krepšininką, ekrane atspausdinkite jo vardą, pavardę, amžių ir poziciją. Jei yra keli, spausdinkite visus.

- Raskite krepšininkus, žaidusius Kauno „Žalgiryje“, ekrane atspausdinkite jų vardus, pavardes bei pozicijas.

- Krepšininkai mėgsta švęsti gimtadienius. Sudarykite sąrašą krepšininkų, kurie švęs gimtadienius pasirengimo krepšinio čempionatui metu (liepos 20d. – rugsėjo 3d.), į failą „Gimtadieniai.csv“ įrašykite krepšininkų vardus, pavardes bei gimimo mėnesį ir dieną.

1.2. Programos tekstas

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

namespace _13_uzduotis
{
    //Class that saves information about one basketball player
    class Player
    {
        public string Name { get; set; }
        public string Surname { get; set; }
        public DateTime BirthDate { get; set; }
        public int Hight { get; set; }
        public int Number { get; set; }
        public string Club { get; set; }
        public bool Invited { get; set; }
        public bool CaptainOrNot { get; set; }

        /// <summary>
        /// Creates public method with the same name as class name
        /// </summary>
        /// <param name="name">Name of player</param>
        /// <param name="surname">Surname of player</param>
        /// <param name="birthDate">Birth date of player</param>
        /// <param name="hight">Hight of player</param>
        /// <param name="number">Number of player</param>
        /// <param name="club">Club of player</param>
        /// <param name="invited">Invited or not invited player</param>
        /// <param name="captainOrNot">Player who is captain or not</param>
        public Player(string name, string surname, DateTime birthDate, int
hight, int number, string club, bool invited, bool captainOrNot)
        {
            this.Name = name;
            this.Surname = surname;
            this.BirthDate = birthDate;
            this.Hight = hight;
            this.Number = number;
            this.Club = club;
        }
    }
}
```

```

        this.Invited = invited;
        this.CaptainOrNot = captainOrNot;
    }

    /// <summary>
    /// Creates int method
    /// </summary>
    /// <returns>Formatted int value</returns>
    public int CalculateAge()
    {
        DateTime today = DateTime.Today;
        int age = today.Year - this.BirthDate.Year;
        if (this.BirthDate.Date > today.AddYears(-age))
        {
            age--;
        }
        return age;
    }
}

```

```

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

namespace _13_uzduotis
{
    //Class that prints or reads information
    class InOutUtils
    {
        /// <summary>
        /// Creates a list to disperse the information
        /// </summary>
        /// <param name="fileName">Specific file name</param>
        /// <returns>Formatted list</returns>
        public static List<Player> ReadFile(string fileName)
        {
            List<Player> Players = new List<Player>();
            string[] Lines = File.ReadAllLines(fileName, Encoding.UTF8);
            foreach (string line in Lines)
            {
                string[] Values = line.Split(';');
                string name = Values[0];
                string surname = Values[1];
                DateTime birthDate = DateTime.Parse(Values[2]);
                int hight = int.Parse(Values[3]);
                int number = int.Parse(Values[4]);
                string club = Values[5];
                //Finding out if player is invited or not
                bool Invited = false;
                if (Values[6] == "pakviestas")
                {
                    Invited = true;
                }
                //Finding out if player is captain or not
                bool captainOrNot = false;
                if (Values[7] == "kapitonas")
                {
                    captainOrNot = true;
                }
            }
        }
    }
}

```

```

        Player Player = new Player(name, surname, birthDate, height,
number, club, Invited, captainOrNot);
        Players.Add(Player);
    }
    return Players;
}

/// <summary>
/// Creates a void function where information is printed
/// </summary>
/// <param name="players">Array of players</param>
/// <param name="fileName">Specific file name</param>
public static void PrintToTxt(List<Player> players, string fileName)
{
    string[] lines = new string[players.Count + 4];
    lines[0] = String.Format(new string('-', 121));
    lines[1] = String.Format("| {0, -8} | {1, -12} | {2, -6} | {3, 8}
| {4, 8} | {5, -8} | {6, -8} | {7, -8} |", "Vardas", "Pavardė", "Gimimo data",
"Žaidėjo ūgis", "Numeris", "Klubas", "Ar pakviestas", "Komandos kapitonas ar ne");
    lines[2] = String.Format(new string('-', 121));
    for (int i = 0; i < players.Count; i++)
    {
        lines[i + 3] = String.Format("| {0, -8} | {1, -12} | {2, -
11:yyyy-MM-dd} | {3, 12} | {4, 8} | {5, -8} | {6, -13} | {7, -24} |", players[i].Name,
players[i].Surname, players[i].BirthDate, players[i].Height, players[i].Number,
players[i].Club, players[i].Invited, players[i].CaptainOrNot);
    }
    lines[players.Count + 3] = String.Format(new string('-', 121));
    File.WriteAllLines(fileName, lines, Encoding.UTF8);
}

/// <summary>
/// Creates a void function where information is printed
/// </summary>
/// <param name="players">Array of players</param>
public static void PrintYoungestPlayers(List<Player> players)
{
    foreach (Player player in players)
    {
        Console.WriteLine("{0};{1};{2};{3}", player.Name,
player.Surname, player.CalculateAge(), player.Number);
    }
}

/// <summary>
/// Creates a void function where information is printed
/// </summary>
/// <param name="players">Array of players</param>
public static void PrintClubPlayers(List<Player> players)
{
    foreach (Player player in players)
    {
        Console.WriteLine("{0};{1};{2}", player.Name, player.Surname,
player.Number);
    }
}

/// <summary>
/// Creates a void function where information is printed
/// </summary>
/// <param name="fileName">Specific file name</param>
/// <param name="players">Array of players</param>
public static void PrintToCsv(string fileName, List<Player> players)

```

```

        {
            string[] lines = new string[players.Count];
            for (int i = 0; i < players.Count; i++)
            {
                lines[i] = string.Format("{0};{1};{2:MM-dd}",
players[i].Name, players[i].Surname, players[i].BirthDate);
            }
            File.WriteAllLines(fileName, lines, Encoding.UTF8);
        }
    }

}

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

namespace _13_uzduotis
{
    //Class that calculates given information and forms lists
    class TaskUtils
    {
        /// <summary>
        /// Creates a list to disperse the information
        /// </summary>
        /// <param name="players">Array of players</param>
        /// <returns>Formatted list</returns>
        public static List<Player> Youngest(List<Player> players)
        {
            List<Player> youngest = new List<Player>();
            Player age = players[0];
            for (int i = 1; i < players.Count; i++)
            {
                if (DateTime.Compare(players[i].BirthDate, age.BirthDate) >
0) //Searching for earliest DateTime information
                {
                    age = players[i];
                }
            }
            for (int i = 0; i < players.Count; i++)
            {
                if (age.BirthDate == players[i].BirthDate) //Comparing
ealiest DateTime information to player birth date
                {
                    youngest.Add(players[i]);
                }
            }
            return youngest;
        }

        /// <summary>
        /// Creates a list to disperse the information
        /// </summary>
        /// <param name="players">Array of players</param>
        /// <param name="team">string representing a team</param>
        /// <returns>Formatted list</returns>
        public static List<Player> Zalgiris(List<Player> players, string team)
        {
            List<Player> InTheTeam = new List<Player>();
            foreach (Player player in players)
            {
                if (player.Club.Equals(team))

```

```

        {
            InTheTeam.Add(player);
        }
    }
    return InTheTeam;
}

/// <summary>
/// Creates a list to disperse the information
/// </summary>
/// <param name="players">Array of players</param>
/// <returns>Formatted list</returns>
public static List<Player> CelebratesBirthDays(List<Player> players)
{
    List<Player> Celebrates = new List<Player>();
    DateTime DateBeginning = new DateTime(DateTime.Now.Year, 7, 20);
//Intodusing new DateTime variable
    DateTime DateEnding = new DateTime(DateTime.Now.Year, 9, 3);
//Intodusing new DateTime variable
    foreach (Player player in players)
    {
        if (player.BirthDate.DayOfYear >= DateBeginning.DayOfYear &&
player.BirthDate.DayOfYear <= DateEnding.DayOfYear) //Converting DateTime
information to values and then comparing them
        {
            Celebrates.Add(player);
        }
    }
    return Celebrates;
}

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading.Tasks;
//Main function of this program is to do all kinds of calculations with
different basketball players information

//Vytenis Kriščiūnas

namespace _13_uzduotis
{
    //Main class
    class Program
    {
        const string CFd = @"Players.txt"; //Represents a .txt file from which
data will be read
        const string CFr1 = "Results.txt"; //Represents a .txt file where data
will be put
        const string CFr2 = "Gimtadieniai.csv"; //Represents a .csv file where
data will be put

        static void Main(string[] args)
        {
            List<Player> allPlayers = InOutUtils.ReadFile(CFd);
            InOutUtils.PrintToTxt(allPlayers, CFr1);

            //Finding and printing yougest players
            List<Player> youngest = TaskUtils.Youngest(allPlayers);
            Console.WriteLine("Jauniausi krepšininkai:");
        }
    }
}

```



```

        InOutUtils.PrintYoungestPlayers(youngest);
        Console.WriteLine();

        //Finding and printing players who play in Žalgiris
        List<Player>    inTheTeam    =    TaskUtils.Zalgiris(allPlayers,
"Žalgiris");

        Console.WriteLine("Krepšininkai žaidę Žalgiryje:");
        InOutUtils.PrintClubPlayers(inTheTeam);
        Console.WriteLine();

        //Printing players who celebrates their birthdays of a given time
frame
        List<Player>                                whoCelebrates    =
TaskUtils.CelebratesBirthDays(allPlayers);
        InOutUtils.PrintToCsv(CFr2, whoCelebrates);

    }
}

```

1.3. Pradiniai duomenys ir rezultatai

Pirmas pavyzdys:

```

Jonas;Valančiūnas;2002-07-28;180;14;Žalgiris;pakviestas;kapitonas
Marius;Grigonis;2002-08-28;179;24;Rytas;pakviestas;žaidėjas

```


Vardas	Pavardė	Gimimo data	Žaidėjo ūgis	Numeris	Klubas	Ar

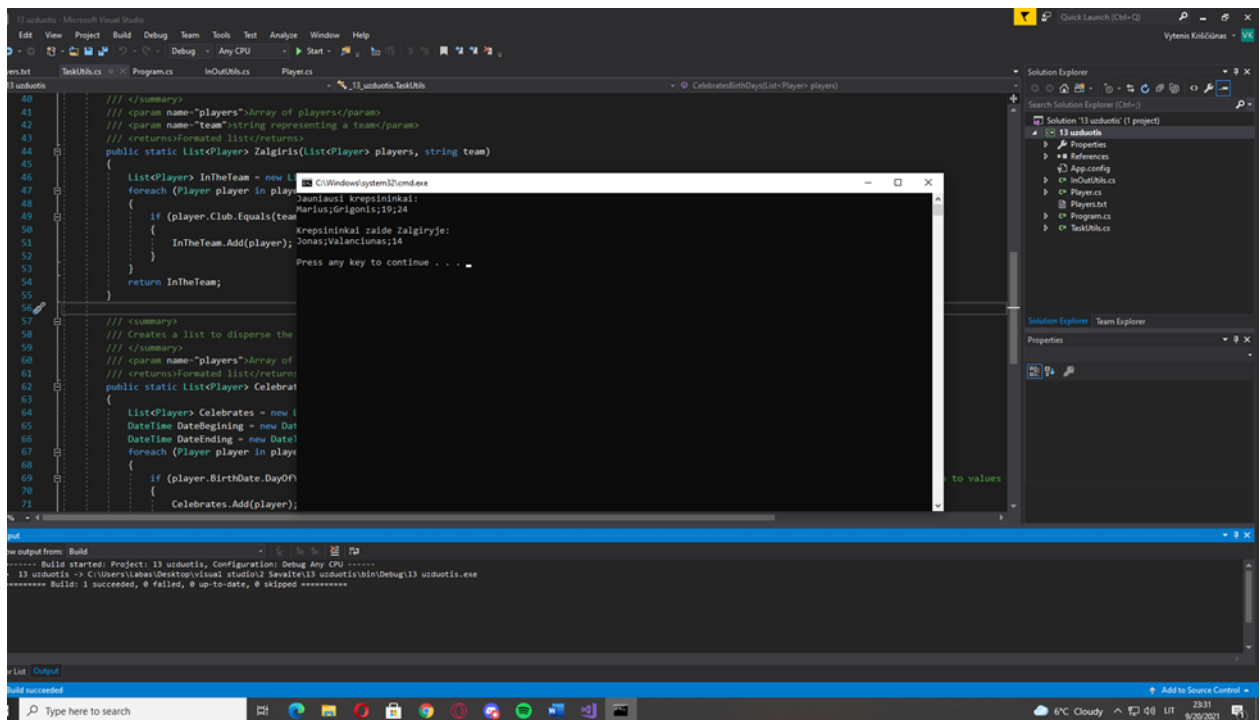
Jonas	Valančiūnas	2002-07-28	180	14	Žalgiris	
True	True					
Marius	Grigonis	2002-08-28	179	24	Rytas	
True	False					

Jauniausi krepšininkai:

```
Marius;Grigonis;19;24
```

Krepšininkai žaidę Žalgiryje:

```
Jonas;Valanciunas;14
```



1 pav. Atspausdinti rezultatai ekrane

Antras pavyzdys:

Jonas;Valančiūnas;2002-07-28;180;14;Žalgiris;pakviestas;žaidėjas
 Mantas;Kalnietis;2000-06-28;179;24;Rytas;pakviestas;kapitonas

```

-----
| Vardas      | Pavardė      | Gimimo data | Žaidėjo ūgis | Numeris | Klubas | Ar
pakviestas | Komandos kapitonas ar ne |
-----

```

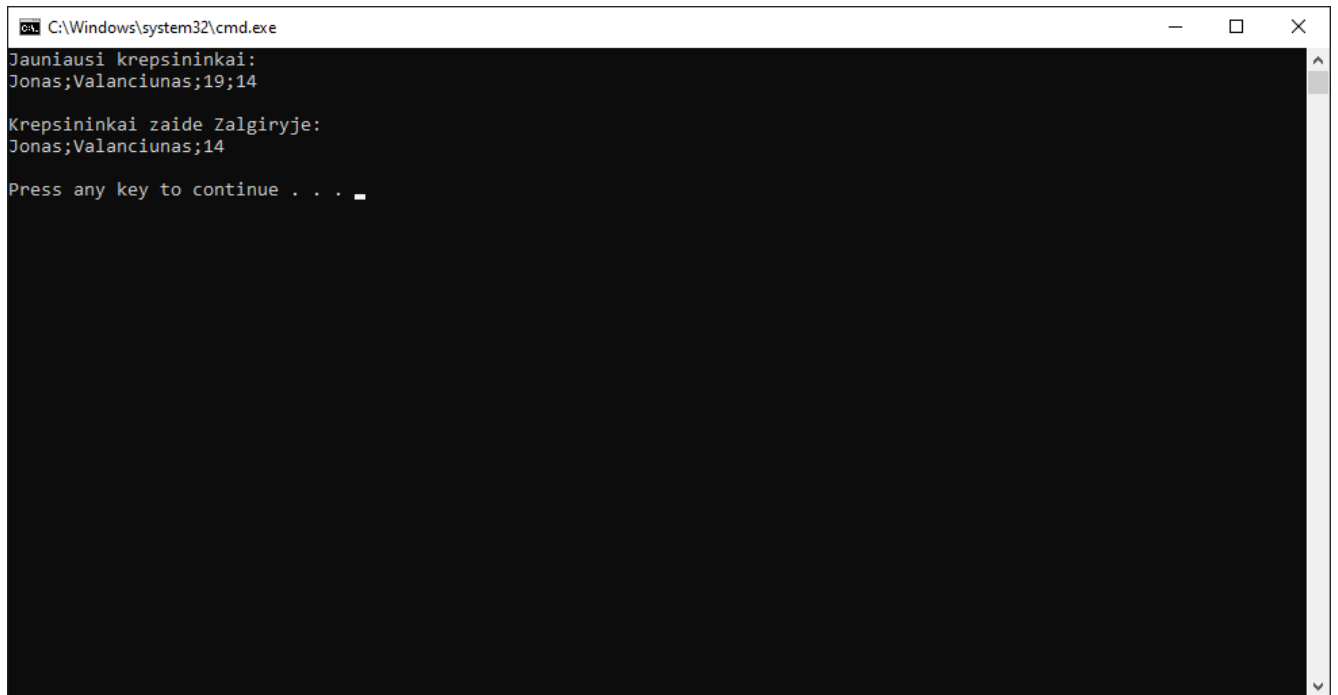
```

-----
| Jonas      | Valančiūnas | 2002-07-28 | 180 | 14 | Žalgiris |
True        | True        |
| Marius     | Grigonis   | 2002-08-28 | 179 | 24 | Rytas    |
True        | False       |
-----

```

Jauniausi krepšininkai:
 Jonas;Valanciunas;19;14

Krepšininkai zaide Zalgiryje:
 Jonas;Valanciunas;14

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text: 'Jauniausi krepšininkai:', 'Jonas;Valanciunas;19;14', 'Krepšininkai žaidė Zalgirijoje:', 'Jonas;Valanciunas;14', and 'Press any key to continue . . .'.

```
C:\Windows\system32\cmd.exe
Jauniausi krepšininkai:
Jonas;Valanciunas;19;14

Krepšininkai žaidė Zalgirijoje:
Jonas;Valanciunas;14

Press any key to continue . . .
```

2 pav. Atspausdinti rezultatai ekrane

1.4. Dėstytojo pastabos

L1 ataskaitos pastabos. Ataskaitos puslapis yra Letter, o ne A4 dydžio. Viršelis neatitinka metodiniuose nurodymuose pateikiamo viršelio šablono. 1.1 skyrelyje nenurodytas užduoties numeris. Kai programą sudaro keletas klasių .cs failų, programos teksto skyrelyje (1.2) klasės išdėstomos nuo žemiausiojo lygmens, o lygmens viduje - abėcėlės tvarka, išskyrus failą, kuriame yra Main(). Šis failas turi būti paskutinis. Jūsų atveju - Player, InOutUtils, TaskUtils, Program. Programos testavimas 1.3 skyrelyje tik su vienu duomenų rinkiniu (už šį trūkumą vertinimas gali būti mažinamas iki pusės taško!). L1 ataskaita vertinama 0,5 tšk.

Programos trūkumai:

1. Programoje trūksta komentarų;
2. Programa spausdina ne visus rezultatus, kurių reikia jos aiškumui;
3. Du for ciklai metodo viduje.

Pratybų vertinimas : 0,8.

Testo balas: 1,75.

Galutinis įvertinimas: 7.

2. Skaičiavimų klasė

2.1. Darbo užduotis

U2-13. Krepšinio rinktinė. Turite ne tik šių, bet ir vienų ankstesniųjų metų į stovyklas pakviestų krepšininkų sąrašus. Keičiasi duomenų failų formatas. Pirmoje eilutėje metai, antroje – stovyklos pradžios data, trečioje – stovyklos pabaigos data. Toliau informacija apie krepšininkus pateikta tokiu pačiu formatu kaip L1 užduotyje.

- Raskite krepšininkus, žaidusius Kauno „Žalgiryje“, ekrane atspausdinkite jų vardus, pavardes bei pozicijas.
- Raskite aukščiausią krepšininką, ir ekrane atspausdinkite jo vardą, pavardę bei amžių. Jei yra keli, spausdinkite visus.
- Sudarykite sąrašą krepšininkų, kurių ūgis – 2 metrai ir daugiau, į failą „Aukštaūgiai.csv“ įrašykite krepšininkų vardus, pavardes ir ūgį. Jei krepšininkas buvo pakviestas į rinktinę du metus, įtraukite jį į sąrašą tik vieną kartą.

2.2. Programos tekstas

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

namespace _13_uzduotis
{
    /// <summary>
    /// Class that saves information about one basketball player
    /// </summary>
    class Player
    {
        /// <summary>
        /// Represents year
        /// </summary>
        public int Year { get; set; }
        /// <summary>
        /// Start of year
        /// </summary>
        public DateTime YearStart { get; set; }
        /// <summary>
        /// End of year
        /// </summary>
        public DateTime YearEnd { get; set; }
        /// <summary>
        /// Name of the player
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Surname of the player
        /// </summary>
        public string Surname { get; set; }
        /// <summary>
        /// Birth date of player
        /// </summary>
        public DateTime BirthDate { get; set; }
    }
}
```

```

    /// <summary>
    /// Hight of player
    /// </summary>
    public int Hight { get; set; }
    /// <summary>
    /// Number of player
    /// </summary>
    public int Number { get; set; }
    /// <summary>
    /// Club of player
    /// </summary>
    public string Club { get; set; }
    /// <summary>
    /// Invited or not invited player
    /// </summary>
    public bool Invited { get; set; }
    /// <summary>
    /// Player who is captain or not
    /// </summary>
    public bool CaptainOrNot { get; set; }
    /// <summary>
    /// Represents given value
    /// </summary>
    public int Value { get; set; }

    /// <summary>
    /// Creates Player method
    /// </summary>
    /// <param name="year">Represents year</param>
    /// <param name="yearStart">Start of year</param>
    /// <param name="yearEnd">End of year</param>
    /// <param name="name">Name of player</param>
    /// <param name="surname">Surname of player</param>
    /// <param name="birthDate">Birth date of player</param>
    /// <param name="hight">Hight of player</param>
    /// <param name="number">Number of player</param>
    /// <param name="club">Club of player</param>
    /// <param name="invited">Invited or not invited player</param>
    /// <param name="captainOrNot">Player who is captain or not</param>
    public Player(int year, DateTime yearStart, DateTime yearEnd, string name,
string surname, DateTime birthDate, int hight, int number, string club, bool
invited, bool captainOrNot)
    {
        this.Year = year;
        this.YearStart = yearStart;
        this.YearEnd = yearEnd;
        this.Name = name;
        this.Surname = surname;
        this.BirthDate = birthDate;
        this.Hight = hight;
        this.Number = number;
        this.Club = club;
        this.Invited = invited;
        this.CaptainOrNot = captainOrNot;
    }

    /// <summary>
    /// Creates int method
    /// </summary>
    /// <returns>Formatted int value</returns>
    public int CalculateAge()
    {
        DateTime today = DateTime.Today;
        int age = today.Year - this.BirthDate.Year;

```

```

        if (this.BirthDate.Date > today.AddYears(-age))
        {
            age--;
        }
        return age;
    }

    /// <summary>
    /// Creates overridden string method
    /// </summary>
    /// <returns>String line of text</returns>
    public override string ToString()
    {
        string line;
        line = String.Format("| {0, 8} | {1, -23:yyyy-MM-dd} | {2, -23:yyyy-MM-dd} | {3, -8} | {4, -15} | {5, -12:yyyy-MM-dd} | {6, 8} | {7, 8} | {8, -12} | {9, -20} | {10, -17} |", Year, YearStart, YearEnd, Name, Surname, BirthDate, Hight, Number, Club, Invited, CaptainOrNot);

        return line;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player1">Information about the player</param>
    /// <param name="player2">Information about the player</param>
    /// <returns>True or false</returns>
    public static bool operator ==(Player player1, Player player2)
    {
        if (player1.Name.Equals(player2.Name) && player1.Surname.Equals(player2.Surname))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player1">Information about the player</param>
    /// <param name="player2">Information about the player</param>
    /// <returns>True or false</returns>
    public static bool operator !=(Player player1, Player player2)
    {
        if (!player1.Name.Equals(player2.Name) && !player1.Surname.Equals(player2.Surname))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="club">String variable</param>
    /// <returns>True or false</returns>
    public static bool operator ==(Player player, string club)
    {
        if (player.Club.Equals(club))

```

```

        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="club">String variable</param>
    /// <returns>True or false</returns>
    public static bool operator !=(Player player, string club)
    {
        if (!player.Club.Equals(club))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">Int type variable</param>
    /// <returns>Biggest value</returns>
    public static bool operator >(Player player, int x)
    {
        return player.Hight.CompareTo(x) > 0;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">Int type variable</param>
    /// <returns>Smallest value</returns>
    public static bool operator <(Player player, int x)
    {
        return player.Hight.CompareTo(x) < 0;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="Max">Int type variable</param>
    /// <returns>True or false</returns>
    public static bool operator ==(Player player, int Max)
    {
        if (player.Hight.Equals(Max))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>

```

```

    /// <param name="Max">Int type variable</param>
    /// <returns>True or false</returns>
    public static bool operator !=(Player player, int Max)
    {
        if (!player.Hight.Equals(Max))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">Int type variable</param>
    /// <returns>Greater or equal value</returns>
    public static bool operator >=(Player player, int x)
    {
        return player.Hight.CompareTo(x) >= 0;
    }

    /// <summary>
    /// Creates overridden bool operator
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">Int type variable</param>
    /// <returns>Lesser or equal value</returns>
    public static bool operator <=(Player player, int x)
    {
        return player.Hight.CompareTo(x) <= 0;
    }

    /// <summary>
    /// Creates overridden equals method
    /// </summary>
    /// <param name="other">Given object</param>
    /// <returns>Values are equal</returns>
    public override bool Equals(object other)
    {
        return this.Value == (int)other;
    }

    /// <summary>
    /// Creates overridden GetHashCode method
    /// </summary>
    /// <returns>Value</returns>
    public override int GetHashCode()
    {
        return this.Value.GetHashCode();
    }
}

}

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

namespace _13_uzduotis
{
    /// <summary>

```



```

/// Class that calculates given information
/// </summary>
class PlayersRegister
{
    /// <summary>
    /// Array of players
    /// </summary>
    private List<Player> AllPlayers;

    /// <summary>
    /// Creates method to format list;
    /// </summary>
    public PlayersRegister()
    {
        AllPlayers = new List<Player>();
    }

    /// <summary>
    /// Creates method to disperse given information
    /// </summary>
    /// <param name="players">Array of players</param>
    public PlayersRegister(List<Player> players)
    {
        AllPlayers = new List<Player>();
        foreach (Player player in players)
        {
            this.AllPlayers.Add(player);
        }
    }

    /// <summary>
    /// Creates void method which adds specific information to the list
    /// </summary>
    /// <param name="player">Single player information</param>
    public void Add(Player player)
    {
        AllPlayers.Add(player);
    }

    /// <summary>
    /// Creates int method which counts how many
    /// </summary>
    /// <returns>Specific number</returns>
    public int Count()
    {
        return this.AllPlayers.Count;
    }

    /// <summary>
    /// Creates method of Specific information by index
    /// </summary>
    /// <param name="index">An index</param>
    /// <returns>Player information</returns>
    public Player OnePlayer(int index)
    {
        return AllPlayers[index];
    }

    /// <summary>
    /// Creates void method
    /// </summary>
    /// <param name="players">PlayersRegister variable</param>
    /// <returns>Certain Value</returns>
    public void Number(PlayersRegister players)
    {

```

```

        for (int i = 0; i < players.Count(); i++)
        {
            for (int j = 0; j < AllPlayers.Count(); j++)
            {
                if (players.OnePlayer(i) == AllPlayers[j])
                {
                    players.OnePlayer(i).Value++;
                }
            }
        }
    }

    /// <summary>
    /// Creates a method to disperse the information
    /// </summary>
    /// <param name="club">Club that player represents</param>
    /// <returns>Formatted lis</returns>
    public PlayersRegister Zalgiris(string club)
    {
        PlayersRegister played = new PlayersRegister();
        for (int i = 0; i < AllPlayers.Count(); i++)
        {
            if (AllPlayers[i] == club && AllPlayers[i].Equals(0))
            {
                played.Add(AllPlayers[i]);
            }
        }
        return played;
    }

    /// <summary>
    /// Creates int method
    /// </summary>
    /// <returns>Certain number</returns>
    public int Highest()
    {
        int x = 0;
        for (int i = 0; i < AllPlayers.Count(); i++)
        {
            if (AllPlayers[i] > x)
            {
                x = AllPlayers[i].Hight;
            }
        }
        return x;
    }

    /// <summary>
    /// Creates int method
    /// </summary>
    /// <param name="players">PlayersRegister variable</param>
    /// <returns>Biggest number</returns>
    public int MaxHight(PlayersRegister players)
    {
        return Math.Max(Highest(), players.Highest());
    }

    /// <summary>
    /// Creates a method to disperse the information
    /// </summary>
    /// <param name="players">PlayersRegister variable</param>
    /// <returns>Formatted list</returns>
    public PlayersRegister FilteredByHighest(PlayersRegister players)
    {
        PlayersRegister HighestPlayers = new PlayersRegister();
    }

```

```

        for (int i = 0; i < AllPlayers.Count(); i++)
        {
            if (AllPlayers[i] == MaxHight(players) && AllPlayers[i].Equals(0))
            {
                HighestPlayers.Add(AllPlayers[i]);
            }
        }
        return HighestPlayers;
    }

    /// <summary>
    /// Creates a method to disperse the information
    /// </summary>
    /// <returns>Formatted list</returns>
    public PlayersRegister TwoMetersOrHigher()
    {
        int x = 200;
        PlayersRegister moreThanTwoMeters = new PlayersRegister();
        for (int i = 0; i < AllPlayers.Count(); i++)
        {
            if (AllPlayers[i] >= x && AllPlayers[i].Equals(0))
            {
                moreThanTwoMeters.Add(AllPlayers[i]);
            }
        }
        return moreThanTwoMeters;
    }
}

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

namespace _13_uzduotis
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    class InOutUtils
    {
        /// <summary>
        /// Creates a list to disperse the information
        /// </summary>
        /// <param name="fileName">Specific file name</param>
        /// <returns>Formatted list</returns>
        public static PlayersRegister ReadFile(string fileName)
        {
            PlayersRegister Players = new PlayersRegister();
            string[] Lines = File.ReadAllLines(fileName, Encoding.UTF8);
            foreach (string line in Lines)
            {
                string[] Values = line.Split(';');
                int year = int.Parse(Values[0]);
                DateTime startYear = DateTime.Parse(Values[1]);
                DateTime endYear = DateTime.Parse(Values[2]);
                string name = Values[3];
                string surname = Values[4];
                DateTime birthDate = DateTime.Parse(Values[5]);
                int hight = int.Parse(Values[6]);
                int number = int.Parse(Values[7]);
            }
        }
    }
}

```

```

        string club = Values[8];
        //Finding out if player is invited or not
        bool Invited = false;
        if (Values[9] == "pakviestas")
        {
            Invited = true;
        }
        //Finding out if player is captain or not
        bool captainOrNot = false;
        if (Values[10] == "kapitonas")
        {
            captainOrNot = true;
        }
        Player Player = new Player(year, startYear, endYear, name,
surname, birthDate, hight, number, club, Invited, captainOrNot);
        Players.Add(Player);
    }
    return Players;
}

/// <summary>
/// Creates a void function where information is printed to screen
/// </summary>
/// <param name="players">PlayersRegister variable</param>
/// <param name="fileName">Specific file name</param>
public static void PrintToTxt(PlayersRegister players, string fileName)
{
    string[] lines = new string[players.Count() + 4];
    lines[0] = String.Format(new string('-', 188));
    lines[1] = String.Format("| {0, -8} | {1, -8} | {2, -8} | {3, -8} |
{4, -15} | {5, -8} | {6, 8} | {7, 8} | {8, -12} | {9, -8} | {10, -8} |", "Metai",
"Stovyklos pradžios data", "Stovyklos pabaigos data", "Vardas", "Pavardė", "Gimimo
metai", "Ūgis", "Numeris", "Klubas", "Pakviestas į komandą", "Kapitonas arba ne");
    lines[2] = String.Format(new string('-', 188));

    for (int i = 0; i < players.Count(); i++)
    {
        lines[i + 3] = players.OnePlayer(i).ToString();
    }

    lines[players.Count() + 3] = String.Format(new string('-', 188));

    File.AppendAllLines(fileName, lines, Encoding.UTF8);
}

/// <summary>
/// Creates void method where information is printed to screen
/// </summary>
/// <param name="players">PlayersRegister variable</param>
public static void PrintPlayers(PlayersRegister players)
{
    for (int i = 0; i < players.Count(); i++)
    {
        Console.WriteLine("{0};{1};{2}", players.OnePlayer(i).Name,
players.OnePlayer(i).Surname, players.OnePlayer(i).Invited);
    }
}

/// <summary>
/// Creates void method where information is printed to screen
/// </summary>
/// <param name="players">PlayersRegister variable</param>
public static void PrintHigestPlayers(PlayersRegister players)
{

```

```

        for (int i = 0; i < players.Count(); i++)
        {
            Console.WriteLine("{0};{1};{2}", players.OnePlayer(i).Name,
players.OnePlayer(i).Surname, players.OnePlayer(i).CalculateAge());
        }
    }

    /// <summary>
    /// Creates a void function where information is printed to .csv file
    /// </summary>
    /// <param name="players">PlayersRegister variable</param>
    /// <param name="fileName">Specific file name</param>
    public static void ToCsv(PlayersRegister players, string fileName)
    {
        string[] lines = new string[players.Count()];
        for (int i = 0; i < players.Count(); i++)
        {
            lines[i] = String.Format("{0};{1};{2}", players.OnePlayer(i).Name,
players.OnePlayer(i).Surname, players.OnePlayer(i).Height);
        }

        File.AppendAllLines(fileName, lines, Encoding.UTF8);
    }

}

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading.Tasks;
//Main function of this program is to do all kinds of calculations with different
basketball players information

//Vytenis Kriščiūnas

namespace _13_uzduotis
{
    /// <summary>
    /// Main class
    /// </summary>
    class Program
    {
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd1 = @"FirstYearPlayers.txt";
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd2 = @"SecondYearPlayers.txt";
        /// <summary>
        /// Represents a .txt file where data will be put
        /// </summary>
        const string CFr1 = "Rezults.txt";
        /// <summary>
        /// Represents a .csv file where data will be put
        /// </summary>
        const string CFr2 = "Aukštaūgiai.csv";

        static void Main(string[] args)

```

```

{
    File.Delete(CFr2);
    File.Delete(CFr1);
    PlayersRegister register1 = InOutUtils.ReadFile(CFd1);
    PlayersRegister register2 = InOutUtils.ReadFile(CFd2);
    InOutUtils.PrintToTxt(register1, CFr1);
    InOutUtils.PrintToTxt(register2, CFr1);

    //Players who played in Žalgiris
    Console.WriteLine("Krepšininkai žaidę Žalgiryje:");
    register1.Number(register2);
    InOutUtils.PrintPlayers(register1.Zalgiris("Žalgiris"));
    InOutUtils.PrintPlayers(register2.Zalgiris("Žalgiris"));
    if (register1.Zalgiris("Žalgiris").Count() == 0 &&
register2.Zalgiris("Žalgiris").Count() == 0)
    {
        Console.WriteLine("Tokių žaidėjų nėra.");
    }
    Console.WriteLine();

    //Highest players
    Console.WriteLine("Aukščiausi krepšininkai:");
    register1.Highest();
    register2.Highest();
    register1.MaxHight(register2);
    InOutUtils.PrintHigestPlayers(register1.FilteredByHighest(register2));
    InOutUtils.PrintHigestPlayers(register2.FilteredByHighest(register2));
    if (register1.FilteredByHighest(register2).Count() == 0 &&
register2.FilteredByHighest(register2).Count() == 0)
    {
        Console.WriteLine("Tokių žaidėjų nėra.");
    }
    Console.WriteLine();

    //Players who are two meters or higher
    InOutUtils.ToCsv(register1.TwoMetersOrHigher(), CFr2);
    InOutUtils.ToCsv(register2.TwoMetersOrHigher(), CFr2);

    if (register1.TwoMetersOrHigher().Count() == 0 &&
register2.TwoMetersOrHigher().Count() == 0)
    {
        File.Delete(CFr2);
    }

}
}
}

```

2.3. Pradiniai duomenys ir rezultatai

Pirmas pavyzdys tikrina žaidėjus, kurie visi žaidėjai yra žaidę Žalgiryje ir yra aukštesni nei du metrai, saraše yra tik vienas aukščiausias žaidėjas.

Pirmojo pavyzdžio duomenys iš .txt failo.

```

2020;2020-07-28;2020-11-12;Jonas;Valanciunas;2002-07-
28;210;14;Žalgiris;pakviestas;kapitonas
2020;2020-07-28;2020-11-12;Marius;Grigonis;2002-08-
28;210;24;Žalgiris;pakviestas;žaidėjas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-
28;220;30;Žalgiris;pakviestas;žaidėjas

```

```
2021;2021-01-17;2021-10-12;Jonas;Valanciunas;2002-07-
28;210;14;Žalgiris;pakviestas;kapitonas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-
28;220;30;Žalgiris;pakviestas;žaidėjas
2021;2021-06-28;2021-09-12;Domantas;Sabonis;2001-08-
28;230;13;Žalgiris;pakviestas;žaidėjas
```

Atspausdinta pradinių duomenų lentelė į .txt failą.

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

```
-----
-----
|      2020 | 2020-07-28      | 2020-11-12      | Jonas      |
Valanciunas | 2002-07-28      | 210 | 14 | Žalgiris      | True
| True      |
|      2020 | 2020-07-28      | 2020-11-12      | Marius      |
Grigonis    | 2002-08-28      | 210 | 24 | Žalgiris      | True
| False     |
|      2021 | 2021-05-28      | 2021-11-12      | Mantas      |
Kalnietis    | 2000-08-28      | 220 | 30 | Žalgiris      | True
| False     |
```

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

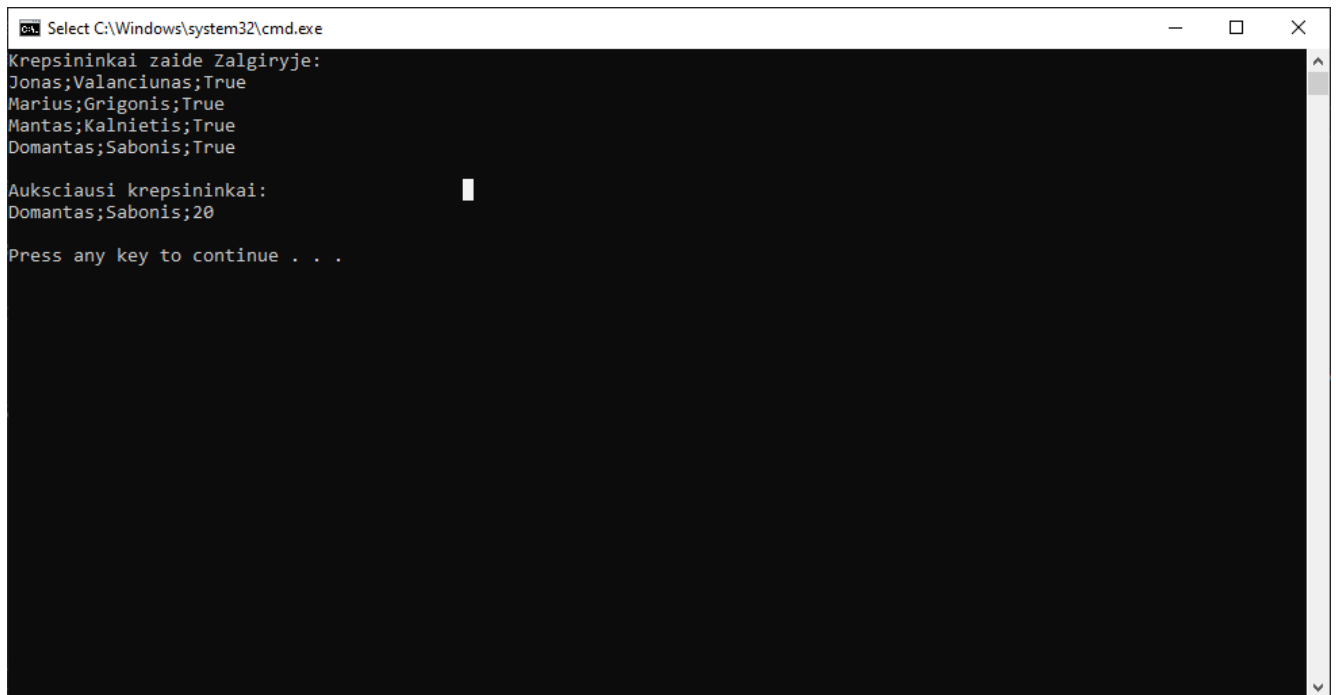
```
-----
-----
|      2021 | 2021-01-17      | 2021-10-12      | Jonas      |
Valanciunas | 2002-07-28      | 210 | 14 | Žalgiris      | True
| True      |
|      2021 | 2021-05-28      | 2021-11-12      | Mantas      |
Kalnietis    | 2000-08-28      | 220 | 30 | Žalgiris      | True
| False     |
|      2021 | 2021-06-28      | 2021-09-12      | Domantas      |
Sabonis      | 2001-08-28      | 230 | 13 | Žalgiris      | True
| False     |
```

Rezultatai atspausdinti į ekraną.

Krepsininkai zaide Zalgiryje:

```
Jonas;Valanciunas;True
Marius;Grigonis;True
Mantas;Kalnietis;True
Domantas;Sabonis;True
```

Aukšciausi krepsininkai:
Domantas;Sabonis;20



```
Select C:\Windows\system32\cmd.exe
Krepsininkai zaide Zalgiryje:
Jonas;Valanciunas;True
Marius;Grigonis;True
Mantas;Kalnietis;True
Domantas;Sabonis;True

Aukšciausi krepsininkai:
Domantas;Sabonis;20

Press any key to continue . . .
```

3 pav. Atspausdinti rezultatai ekrane

Antras pavyzdys tikrina žaidėjus, kurie ne visi yra žaidę Žalgiryje, yra du aukščiausi žaidėjai ir nėra nė vieno žaidėjo, kuris būtų aukštesnis arba lygus dviem metrams:

Antrojo pavyzdžio duomenys iš .txt failo.

```
2020;2020-07-28;2020-11-12;Jonas;Valanciunas;2002-07-28;175;14;Žalgiris;pakviestas;kapitonas
2020;2020-07-28;2020-11-12;Marius;Grigonis;2002-08-28;162;24;Lietkabelis;pakviestas;žaidėjas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-28;175;30;Rytas;pakviestas;žaidėjas
```

```
2021;2021-01-17;2021-10-12;Jonas;Valanciunas;2002-07-28;175;14;Žalgiris;pakviestas;kapitonas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-28;175;30;Rytas;pakviestas;žaidėjas
2021;2021-06-28;2021-09-12;Domantas;Sabonis;2001-08-28;160;13;Žalgiris;pakviestas;žaidėjas
```

Atspausdinta pradinių duomenų lentelė į .txt failą.

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
-----
-----
-----
```



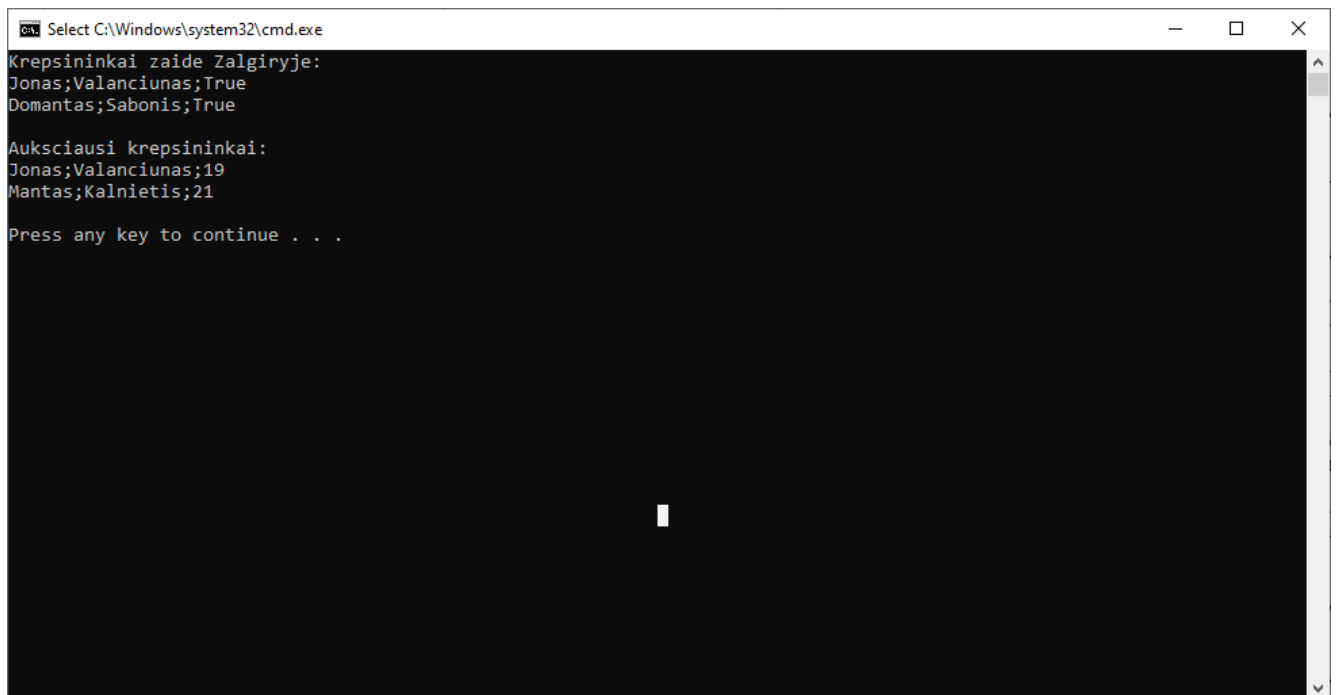
```

|      2020 | 2020-07-28          | 2020-11-12          | Jonas      |
Valanciunas | 2002-07-28 | 175 | 14 | Žalgiris | True
| True      |
|      2020 | 2020-07-28          | 2020-11-12          | Marius     |
Grigonis    | 2002-08-28 | 162 | 24 | Lietkabelis | True
| False     |
|      2021 | 2021-05-28          | 2021-11-12          | Mantas     |
Kalnietis   | 2000-08-28 | 175 | 30 | Rytas      | True
| False     |
-----
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
-----
-----
-----
|      2021 | 2021-01-17          | 2021-10-12          | Jonas      |
Valanciunas | 2002-07-28 | 175 | 14 | Žalgiris | True
| True      |
|      2021 | 2021-05-28          | 2021-11-12          | Mantas     |
Kalnietis   | 2000-08-28 | 175 | 30 | Rytas      | True
| False     |
|      2021 | 2021-06-28          | 2021-09-12          | Domantas   |
Sabonis     | 2001-08-28 | 160 | 13 | Žalgiris | True
| False     |
-----
-----
-----

Rezultatai atspausdinti į ekraną.
Krepšininkai zaide Zalgiryje:
Jonas;Valanciunas;True
Domantas;Sabonis;True

Auksciausi krepšininkai:
Jonas;Valanciunas;19
Mantas;Kalnietis;21

```

A screenshot of a Windows command prompt window. The title bar reads "Select C:\Windows\system32\cmd.exe". The window contains the following text:

```
Krepsininkai zaide Zalgirye:  
Jonas;Valanciunas;True  
Domantas;Sabonis;True  
  
Auksciausi krepsininkai:  
Jonas;Valanciunas;19  
Mantas;Kalnietis;21  
  
Press any key to continue . . .
```

A cursor is visible on the line "Press any key to continue . . .".

4 pav. Atspausdinti rezultatai ekrane

2.4. Dėstytojo pastabos

L2 ataskaitos pastabos. 2.3 skyrelis netvarkingas: netaisyklingi sakiniai; tekste nėra nuorodų į paveikslėlius; nepaaiškinta, kokius programos kelius tikrina testavimo atvejai; nepaaiškinama, kokia informacija yra pateikiama. L2 ataskaita vertinama 0,8 tšk.

Programos trūkumai:

1. Netaisyklingi komentarai;
2. Netaisyklingi metodų pavadinimai;
3. Reikia hardcodint kintamuosius.

Pratybų vertinimas: 1.

Testo balas: 1.

Galutinis įvertinimas: 7.

3. Konteineris

3.1. Darbo užduotis

U3_13. Krepšinio rinktinė. Turite ne tik šių, bet ir vienų ankstesniųjų metų į stovyklas pakviestų krepšininkų sąrašus. Keičiasi duomenų failų formatas. Pirmoje eilutėje metai, antroje – stovyklos pradžios data, trečioje – stovyklos pabaigos data. Toliau informacija apie krepšininkus pateikta tokiu pačiu formatu kaip L1 užduotyje.

- Raskite jauniausią į rinktinę pakviestą krepšininką, ekrane atspausdinkite jo vardą, pavardę, amžių ir poziciją. Jei yra keli, spausdinkite visus.
- Raskite krepšininkus, žaidusius Kauno „Žalgiryje“, ekrane atspausdinkite jų vardus, pavardes bei pozicijas.
- Sudarykite į rinktinę pakviestų krepšininkų, kurie dalyvavo dvejose stovyklose, sąrašą. Jų duomenis įrašykite į failą „Senbuviai.csv“. Surikiuokite krepšininkus pagal ūgį, pavardes ir vardus.
- Sudarykite sąrašą krepšininkų, kurių ūgis – 2 metrai ir daugiau, į failą „Aukštaūgiai.csv“ įrašykite krepšininkų vardus, pavardes ir ūgį. Jei krepšininkas buvo pakviestas į rinktinę du metus, įtraukite jį į sąrašą tik vieną kartą.

3.2. Programos tekstas

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

namespace _13_uzduotis
{
    /// <summary>
    /// Class that saves information about one basketball player
    /// </summary>
    class Player
    {
        /// <summary>
        /// Represents year
        /// </summary>
        public int Year { get; set; }
        /// <summary>
        /// Start of year
        /// </summary>
        public DateTime YearStart { get; set; }
        /// <summary>
        /// End of year
        /// </summary>
        public DateTime YearEnd { get; set; }
        /// <summary>
        /// Name of the player
        /// </summary>
        public string Name { get; set; }
    }
}
```

```

    /// <summary>
    /// Surname of the player
    /// </summary>
    public string Surname { get; set; }
    /// <summary>
    /// Birth date of player
    /// </summary>
    public DateTime BirthDate { get; set; }
    /// <summary>
    /// Hight of player
    /// </summary>
    public int Hight { get; set; }
    /// <summary>
    /// Number of player
    /// </summary>
    public int Number { get; set; }
    /// <summary>
    /// Club of player
    /// </summary>
    public string Club { get; set; }
    /// <summary>
    /// Invited or not invited player
    /// </summary>
    public bool Invited { get; set; }
    /// <summary>
    /// Player who is captain or not
    /// </summary>
    public bool CaptainOrNot { get; set; }
    /// <summary>
    /// Represents given value
    /// </summary>
    public int Value { get; set; }

    /// <summary>
    /// Creates Player constructor
    /// </summary>
    /// <param name="year">Represents year</param>
    /// <param name="yearStart">Start of year</param>
    /// <param name="yearEnd">End of year</param>
    /// <param name="name">Name of player</param>
    /// <param name="surname">Surname of player</param>
    /// <param name="birthDate">Birth date of player</param>
    /// <param name="hight">Hight of player</param>
    /// <param name="number">Number of player</param>
    /// <param name="club">Club of player</param>
    /// <param name="invited">Invited or not invited player</param>
    /// <param name="captainOrNot">Player who is captain or not</param>
    public Player(int year, DateTime yearStart, DateTime yearEnd, string name,
string surname, DateTime birthDate, int hight, int number, string club, bool
invited, bool captainOrNot)
    {
        this.Year = year;
        this.YearStart = yearStart;
        this.YearEnd = yearEnd;
        this.Name = name;
        this.Surname = surname;
        this.BirthDate = birthDate;

```

```

        this.Hight = hight;
        this.Number = number;
        this.Club = club;
        this.Invited = invited;
        this.CaptainOrNot = captainOrNot;
    }

    /// <summary>
    /// Calculates player's age
    /// </summary>
    /// <returns>Age value</returns>
    public int CalculateAge()
    {
        DateTime today = DateTime.Today;
        int age = today.Year - this.BirthDate.Year;
        if (this.BirthDate.Date > today.AddYears(-age))
        {
            age--;
        }
        return age;
    }

    /// <summary>
    /// Overriden to string method
    /// </summary>
    /// <returns>String line of text</returns>
    public override string ToString()
    {
        string line;
        line = String.Format("| {0, 8} | {1, -23:yyyy-MM-dd} | {2, -23:yyyy-MM-dd} | {3, -8} | {4, -15} | {5, -12:yyyy-MM-dd} | {6, 8} | {7, 8} | {8, -12} | {9, -20} | {10, -17} |", Year, YearStart, YearEnd, Name, Surname, BirthDate, Hight, Number, Club, Invited, CaptainOrNot);

        return line;
    }

    /// <summary>
    /// Operator that compares players from different lists
    /// </summary>
    /// <param name="player1">Information about the player</param>
    /// <param name="player2">Information about the player</param>
    /// <returns>True or false</returns>
    public static bool operator ==(Player player1, Player player2)
    {
        if (player1.Name.Equals(player2.Name) && player1.Surname.Equals(player2.Surname))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Operator that compares players from different lists

```

```

    /// </summary>
    /// <param name="player1">Information about the player</param>
    /// <param name="player2">Information about the player</param>
    /// <returns>True or false</returns>
    public static bool operator !=(Player player1, Player player2)
    {
        if (!player1.Name.Equals(player2.Name) &&
!player1.Surname.Equals(player2.Surname))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Operator that compares player class club to club
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="club">String variable</param>
    /// <returns>True or false</returns>
    public static bool operator ==(Player player, string club)
    {
        if (player.Club.Equals(club))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Operator that compares player class club to club
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="club">String variable</param>
    /// <returns>True or false</returns>
    public static bool operator !=(Player player, string club)
    {
        if (!player.Club.Equals(club))
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Operator that compares player's birthdate to DateTime variable
    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">DateTime variable</param>
    /// <returns>Largest value</returns>
    public static bool operator >(Player player, DateTime x)
    {
        return player.BirthDate.CompareTo(x) > 0;
    }

```

```

/// <summary>
/// Operator that compares player's birthdate to DateTime variable
/// </summary>
/// <param name="player">Information about the player</param>
/// <param name="x">DateTime variable</param>
/// <returns>Smallest value</returns>
public static bool operator <(Player player, DateTime x)
{
    return player.BirthDate.CompareTo(x) < 0;
}

/// <summary>
/// Operator that compares player's age to DateTime variable
/// </summary>
/// <param name="player">Information about the player</param>
/// <param name="Min">DateTime type variable</param>
/// <returns>True or false</returns>
public static bool operator ==(Player player, DateTime Min)
{
    if (player.BirthDate.Equals(Min))
    {
        return true;
    }
    return false;
}

/// <summary>
/// Operator that compares player's age to DateTime variable
/// </summary>
/// <param name="player">Information about the player</param>
/// <param name="Min">DateTime type variable</param>
/// <returns>True or false</returns>
public static bool operator !=(Player player, DateTime Min)
{
    if (!player.BirthDate.Equals(Min))
    {
        return true;
    }
    return false;
}

/// <summary>
/// Operator that finds out if player's hight is bigger than given integer
/// </summary>
/// <param name="player">Information about the player</param>
/// <param name="x">Int type variable</param>
/// <returns>Greater or equal value</returns>
public static bool operator >=(Player player, int x)
{
    return player.Hight.CompareTo(x) >= 0;
}

/// <summary>
/// Operator that finds out if player's hight is lesser than given integer

```

```

    /// </summary>
    /// <param name="player">Information about the player</param>
    /// <param name="x">Int type variable</param>
    /// <returns>Lesser or equal value</returns>
    public static bool operator <=(Player player, int x)
    {
        return player.Hight.CompareTo(x) <= 0;
    }

    /// <summary>
    /// Overriden Equals method that finds out if player's value is equal to
given object
    /// </summary>
    /// <param name="other">Given object</param>
    /// <returns>Values are equal</returns>
    public override bool Equals(object other)
    {
        return this.Value == (int)other;
    }

    /// <summary>
    /// Overriden GetHashCode of value method
    /// </summary>
    /// <returns>Value</returns>
    public override int GetHashCode()
    {
        return this.Value.GetHashCode();
    }

    /// <summary>
    /// Overriden CompareTo method that sorts given information by hight,
surname and name
    /// </summary>
    /// <param name="other">Information about the player</param>
    /// <returns>Sorted information</returns>
    public int CompareTo(Player other)
    {
        if (this.Hight != other.Hight)
        {
            return this.Hight.CompareTo(other.Hight);
        }
        else if (this.Surname != other.Surname)
        {
            return this.Surname.CompareTo(other.Surname);
        }
        else
        {
            return this.Name.CompareTo(other.Name);
        }
    }
}

```

```

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

```



```

namespace _13_uzduotis
{
    /// <summary>
    /// Container of basketball players
    /// </summary>
    class PlayersContainer
    {
        /// <summary>
        /// Array of players
        /// </summary>
        private Player[] players;
        /// <summary>
        /// Total count of players
        /// </summary>
        public int Count { get; private set; }
        /// <summary>
        /// Total capacity of players
        /// </summary>
        private int Capacity;

        /// <summary>
        /// PlayersContainer constructor
        /// </summary>
        /// <param name="capacity">Capacity of players</param>
        public PlayersContainer(int capacity = 5)
        {
            this.Capacity = capacity;
            this.players = new Player[capacity];
        }

        /// <summary>
        /// Method that makes sure of the capacity in the list of players
        /// </summary>
        /// <param name="minimumCapacity">Minimum capacity of players</param>
        public void EnsureCapacity(int minimumCapacity)
        {
            if (minimumCapacity > this.Capacity)
            {
                Player[] temp = new Player[minimumCapacity];
                for (int i = 0; i < this.Count; i++)
                {
                    temp[i] = this.players[i];
                }
                this.Capacity = minimumCapacity;
                this.players = temp;
            }
        }

        /// <summary>
        /// Method that adds information to the list
        /// </summary>
        /// <param name="player">Player variable</param>
        public void Add(Player player)
        {
            if (Count == Capacity)
            {
                EnsureCapacity(Capacity * 2);
            }
            this.players[Count++] = player;
        }

        /// <summary>
        /// Method that gives one players information
        /// </summary>
        /// <param name="index">An index</param>

```

```

/// <returns>Players information</returns>
public Player Get(int index)
{
    return this.players[index];
}

/// <summary>
/// Method that finds out if a list contains something
/// </summary>
/// <param name="player">Player variable</param>
/// <returns></returns>
public bool Contains(Player player)
{
    for (int i = 0; i < this.Count; i++)
    {
        if (this.players[i].Equals(player))
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Method that puts information in an indicated place
/// </summary>
/// <param name="player">Player variable</param>
/// <param name="index">An index</param>
public void Put(Player player, int index)
{
    players[index] = player;
}

/// <summary>
/// Method that inserts information in an indicated place
/// </summary>
/// <param name="player">Player variable</param>
/// <param name="index">An index</param>
public void Insert(Player player, int index)
{
    if (this.Count == this.Capacity)
    {
        EnsureCapacity(this.Capacity * 2);
    }

    if (index == this.Count)
    {
        this.Count++;
    }

    int j = index;
    for (int i = index + 1; i < this.Count; i++)
    {
        players[i] = players[j];
        j++;
    }
    players[index] = player;
}

/// <summary>
/// Method that removes information from the list
/// </summary>
/// <param name="player">Player variable</param>
public void Remove(Player player)
{
    if (this.Count == this.Capacity)

```

```

    {
        EnsureCapacity(this.Capacity * 2);
    }

    for (int i = 0; i < this.Count; i++)
    {
        if(players[i] == player)
        {
            for (int j = i; j < this.Count; j++)
            {
                players[j] = players[j + 1];
            }
            Count--;
            i--;
        }
    }
}

/// <summary>
/// Method that removes information from an indicated place in the list
/// </summary>
/// <param name="index">An index</param>
public void RemoveAt(int index)
{
    if (this.Count == this.Capacity)
    {
        EnsureCapacity(this.Capacity * 2);
    }

    for (int i = 0; i < this.Count; i++)
    {
        if (i == index)
        {
            for (int j = i; j < this.Count; j++)
            {
                players[j] = players[j + 1];
            }
            Count--;
        }
    }
}

/// <summary>
/// Method that sorts information in the list of players
/// </summary>
public void Sort()
{
    bool flag = true;

    while (flag)
    {
        flag = false;
        for (int i = 0; i < this.Count - 1; i++)
        {
            Player a = this.players[i];
            Player b = this.players[i + 1];
            if (a.CompareTo(b) > 0)
            {
                this.players[i] = b;
                this.players[i + 1] = a;
                flag = true;
            }
        }
    }
}
}

```

```

    }
}

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

namespace _13_uzduotis
{
    /// <summary>
    /// Class that calculates given information
    /// </summary>
    class PlayersRegister
    {
        /// <summary>
        /// Array of players
        /// </summary>
        private PlayersContainer AllPlayers;

        /// <summary>
        /// Formats a list;
        /// </summary>
        public PlayersRegister()
        {
            AllPlayers = new PlayersContainer();
        }

        /// <summary>
        /// Method that disperses information
        /// </summary>
        /// <param name="players">Array of players</param>
        public PlayersRegister(PlayersContainer players)
        {
            AllPlayers = new PlayersContainer();
            for (int i = 0; i < players.Count; i++)
            {
                AllPlayers.Add(players.Get(i));
            }
        }

        /// <summary>
        /// Void method that adds specific information to the list
        /// </summary>
        /// <param name="player">Single player information</param>
        public void Add(Player player)
        {
            AllPlayers.Add(player);
        }

        /// <summary>
        /// Method which counts how many
        /// </summary>
        /// <returns>Specific number</returns>
        public int Count()
        {
            return this.AllPlayers.Count;
        }

        /// <summary>
        /// Method that gives one players information
        /// </summary>
        /// <param name="index">An index</param>
        /// <returns>Player information</returns>
    }
}

```

```

public Player OnePlayer(int index)
{
    return AllPlayers.Get(index);
}

/// <summary>
/// Void method that calls a second method
/// </summary>
/// <param name="players">PlayersRegister variable</param>
public void Number(PlayersRegister players)
{
    for (int i = 0; i < players.Count(); i++)
    {
        Value(players.OnePlayer(i));
    }
}

/// <summary>
/// Method that finds out if a player from another list has been in this list
/// </summary>
/// <param name="players">Player variable</param>
public void Value(Player players)
{
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (players == AllPlayers.Get(i))
        {
            players.Value++;
        }
    }
}

/// <summary>
/// Finds youngest player
/// </summary>
/// <returns>Age of youngest player represented in date format</returns>
public DateTime Youngest()
{
    DateTime date = new DateTime(1000, 01, 01);
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (AllPlayers.Get(i) > date)
        {
            date = AllPlayers.Get(i).BirthDate;
        }
    }
    return date;
}

/// <summary>
/// Finds youngest player from two lists of players
/// </summary>
/// <param name="players">PlayersRegister variable</param>
/// <returns>Longest date</returns>
public DateTime MinAge(PlayersRegister players)
{
    if (Youngest() > players.Youngest())
    {
        return Youngest();
    }
    else
        return players.Youngest();
}

/// <summary>

```

```

/// Method that formats a list of youngest players
/// </summary>
/// <param name="players">PlayersRegister variable</param>
/// <returns>Formatted list</returns>
public PlayersRegister FilteredByYoungest(PlayersRegister players)
{
    PlayersRegister YoungestPlayers = new PlayersRegister();
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (AllPlayers.Get(i) == MinAge(players) && AllPlayers.Get(i).Equals(0))
        {
            YoungestPlayers.Add(AllPlayers.Get(i));
        }
    }
    return YoungestPlayers;
}

/// <summary>
/// Method that finds out if players played in certain club
/// </summary>
/// <param name="club">Club that player represents</param>
/// <returns>Formatted list</returns>
public PlayersRegister FilteredByClub(string club)
{
    PlayersRegister played = new PlayersRegister();
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (AllPlayers.Get(i) == club && AllPlayers.Get(i).Equals(0))
        {
            played.Add(AllPlayers.Get(i));
        }
    }
    return played;
}

/// <summary>
/// Method that finds players who participated in both camps
/// </summary>
/// <returns>Formatted list</returns>
public void BeenToBothCamps(PlayersContainer players)
{
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (AllPlayers.Get(i).Value > 0)
        {
            players.Add(AllPlayers.Get(i));
        }
    }
}

/// <summary>
/// Method that finds players who are higher or equal to two meters
/// </summary>
/// <param name="x">Certain integer</param>
/// <returns>Formatted list</returns>
public PlayersRegister TwoMetersOrHigher(int x)
{
    PlayersRegister moreThanTwoMeters = new PlayersRegister();
    for (int i = 0; i < AllPlayers.Count; i++)
    {
        if (AllPlayers.Get(i) >= x && AllPlayers.Get(i).Equals(0))
        {
            moreThanTwoMeters.Add(AllPlayers.Get(i));
        }
    }
}

```

```

        return moreThanTwoMeters;
    }
}

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

namespace _13_uzduotis
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    class InOutUtils
    {
        /// <summary>
        /// Creates a list to disperse the information
        /// </summary>
        /// <param name="fileName">Specific file name</param>
        /// <returns>Formatted list</returns>
        public static PlayersContainer ReadFile(string fileName)
        {
            PlayersContainer Players = new PlayersContainer();
            string[] Lines = File.ReadAllLines(fileName, Encoding.UTF8);
            foreach (string line in Lines)
            {
                string[] Values = line.Split(';');
                int year = int.Parse(Values[0]);
                DateTime startYear = DateTime.Parse(Values[1]);
                DateTime endYear = DateTime.Parse(Values[2]);
                string name = Values[3];
                string surname = Values[4];
                DateTime birthDate = DateTime.Parse(Values[5]);
                int hight = int.Parse(Values[6]);
                int number = int.Parse(Values[7]);
                string club = Values[8];
                //Finding out if player is invited or not
                bool Invited = false;
                if (Values[9] == "pakviestas")
                {
                    Invited = true;
                }
                //Finding out if player is captain or not
                bool captainOrNot = false;
                if (Values[10] == "kapitonas")
                {
                    captainOrNot = true;
                }
                Player Player = new Player(year, startYear, endYear, name, surname, birthDate,
hight, number, club, Invited, captainOrNot);
                Players.Add(Player);
            }
            return Players;
        }

        /// <summary>
        /// A method that prints information to the screen
        /// </summary>
        /// <param name="players">PlayersRegister variable</param>
        /// <param name="fileName">Specific file name</param>

```

```

public static void PrintToTxt(PlayersContainer players, string fileName)
{
    string[] lines = new string[players.Count + 4];
    lines[0] = String.Format(new string('-', 188));
    lines[1] = String.Format("| {0, -8} | {1, -8} | {2, -8} | {3, -8} | {4, -15} | {5, -8} | {6, 8} | {7, 8} | {8, -12} | {9, -8} | {10, -8} |", "Metai", "Stovyklos pradžios data", "Stovyklos pabaigos data", "Vardas", "Pavardė", "Gimimo metai", "Ūgis", "Numeris", "Klubas", "Pakviestas į komandą", "Kapitonas arba ne");
    lines[2] = String.Format(new string('-', 188));

    for (int i = 0; i < players.Count; i++)
    {
        lines[i + 3] = players.Get(i).ToString();
    }

    lines[players.Count + 3] = String.Format(new string('-', 188));

    File.AppendAllLines(fileName, lines, Encoding.UTF8);
}

/// <summary>
/// A method that prints information about players who played in certain club to the
screen
/// </summary>
/// <param name="players">PlayersRegister variable</param>
public static void PrintPlayers(PlayersRegister players)
{
    for (int i = 0; i < players.Count(); i++)
    {
        Console.WriteLine("{0};{1};{2}", players.OnePlayer(i).Name,
players.OnePlayer(i).Surname, players.OnePlayer(i).Invited);
    }
}

/// <summary>
/// A method that prints information about youngest players to the screen
/// </summary>
/// <param name="players">PlayersRegister variable</param>
public static void PrintYoungestPlayers(PlayersRegister players)
{
    for (int i = 0; i < players.Count(); i++)
    {
        Console.WriteLine("{0};{1};{2};{3}", players.OnePlayer(i).Name,
players.OnePlayer(i).Surname, players.OnePlayer(i).CalculateAge(),
players.OnePlayer(i).Invited);
    }
}

/// <summary>
/// Method that prints information about players who participated in both camps to
.csv file
/// </summary>
/// <param name="players">PlayerRegister variable</param>
/// <param name="fileName">Specific file name</param>
public static void ToSenbuviaiCSV(PlayersContainer players, string fileName)
{
    string[] lines = new string[players.Count];
    for (int i = 0; i < players.Count; i++)
    {
        lines[i] = players.Get(i).ToString();
    }

    File.WriteAllLines(fileName, lines, Encoding.UTF8);
}

```



```

file
    /// <summary>
    /// Method that prints information about players who are two meters or higher to .csv
    /// </summary>
    /// <param name="players">PlayersRegister variable</param>
    /// <param name="fileName">Specific file name</param>
    public static void ToCsv(PlayersRegister players, string fileName)
    {
        string[] lines = new string[players.Count()];
        for (int i = 0; i < players.Count(); i++)
        {
            lines[i] = String.Format("{0};{1};{2}", players.OnePlayer(i).Name,
            players.OnePlayer(i).Surname, players.OnePlayer(i).Hight);
        }

        File.AppendAllLines(fileName, lines, Encoding.UTF8);
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading.Tasks;
//Main function of this program is to do all kinds of calculations with different basketball
players information

```

//Vytenis Kriščiūnas

```

namespace _13_uzduotis
{
    /// <summary>
    /// Main class
    /// </summary>
    class Program
    {
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd1 = @"FirstYearPlayers.txt";
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd2 = @"SecondYearPlayers.txt";
        /// <summary>
        /// Represents a .txt file where data will be put
        /// </summary>
        const string CFr1 = "Rezults.txt";
        /// <summary>
        /// Represents a .csv file where data will be put
        /// </summary>
        const string CFr2 = "Senbuviai.csv";
        /// <summary>
        /// Represents a .csv file where data will be put
        /// </summary>
        const string CFr3 = "Aukštaūgiai.csv";

        static void Main(string[] args)
        {
            File.Delete(CFr1);
            File.Delete(CFr3);
        }
    }
}

```

```

PlayersContainer FirstList = InOutUtils.ReadFile(CFd1);
PlayersContainer SecondList = InOutUtils.ReadFile(CFd2);
InOutUtils.PrintToTxt(FirstList, CFr1);
InOutUtils.PrintToTxt(SecondList, CFr1);

PlayersRegister register1 = new PlayersRegister(FirstList);
PlayersRegister register2 = new PlayersRegister(SecondList);

register1.Number(register2);

//Youngest players
Console.WriteLine("Jauniausi krepšininkai:");
InOutUtils.PrintYoungestPlayers(register1.FilteredByYoungest(register2));
InOutUtils.PrintYoungestPlayers(register2.FilteredByYoungest(register1));

if (register1.FilteredByYoungest(register2).Count() == 0 &&
register2.FilteredByYoungest(register2).Count() == 0)
{
    Console.WriteLine("Tokių žaidėjų nėra.");
}
Console.WriteLine();

//Players who played in Žalgiris
Console.WriteLine("Krepšininkai žaidę Žalgiryje:");
InOutUtils.PrintPlayers(register1.FilteredByClub("Žalgiris"));
InOutUtils.PrintPlayers(register2.FilteredByClub("Žalgiris"));

if (register1.FilteredByClub("Žalgiris").Count() == 0 &&
register2.FilteredByClub("Žalgiris").Count() == 0)
{
    Console.WriteLine("Tokių žaidėjų nėra.");
}
Console.WriteLine();

//Players who participated in both camps
PlayersContainer ListOfFiltered = new PlayersContainer();
register2.BeenToBothCamps(ListOfFiltered);
ListOfFiltered.Sort();
InOutUtils.ToSenbuviaiCSV(ListOfFiltered, CFr2);

if (ListOfFiltered.Count == 0)
{
    File.Delete(CFr2);
}
Console.WriteLine();

//Players who are two meters or higher
InOutUtils.ToCsv(register1.TwoMetersOrHigher(200), CFr3);
InOutUtils.ToCsv(register2.TwoMetersOrHigher(200), CFr3);

if (register1.TwoMetersOrHigher(200).Count() == 0 &&
register2.TwoMetersOrHigher(200).Count() == 0)
{
    File.Delete(CFr3);
}

    }
}
}

```

3.3. Pradiniai duomenys ir rezultatai

Pirmas pavyzdys tikrina krepšininkus, kai visi krepšininkai žaidė Žalgiryje, randa vieną jauniausią krepšininką pagal gimimo datą, randa du krepšininkus, kurie dalyvavo abejose stovyklose ir keturis krepšininkus, kurių ūgi 2 m. ir daugiau.

Pirmojo pavyzdžio duomenys iš .txt failo.

```
2020;2020-07-28;2020-11-12;Jonas;Valanciunas;2002-07-28;210;14;Žalgiris;pakviestas;kapitonas
2020;2020-07-28;2020-11-12;Marius;Grigonis;2002-08-28;210;24;Žalgiris;pakviestas;žaidėjas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-28;220;30;Žalgiris;pakviestas;žaidėjas
```

```
2021;2021-01-17;2021-10-12;Jonas;Valanciunas;2002-07-28;210;14;Žalgiris;pakviestas;kapitonas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-28;220;30;Žalgiris;pakviestas;žaidėjas
2021;2021-06-28;2021-09-12;Domantas;Sabonis;2001-08-28;230;13;Žalgiris;pakviestas;žaidėjas
```

Atspausdinta pradinių duomenų lentelė į .txt failą.

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

```
-----
-----
|      2020 | 2020-07-28      | 2020-11-12      | Jonas      |
Valanciunas | 2002-07-28      | 210 | 14 | Žalgiris      | True
| True      |
|      2020 | 2020-07-28      | 2020-11-12      | Marius      |
Grigonis    | 2002-08-28      | 210 | 24 | Žalgiris      | True
| False     |
|      2021 | 2021-05-28      | 2021-11-12      | Mantas      |
Kalnietis    | 2000-08-28      | 220 | 30 | Žalgiris      | True
| False     |
```

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas      |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

```
-----
-----
|      2021 | 2021-01-17      | 2021-10-12      | Jonas      |
Valanciunas | 2002-07-28      | 210 | 14 | Žalgiris      | True
| True      |
```

	2021		2021-05-28			2021-11-12			Mantas	
Kalnietis			2000-08-28		220		30		Žalgiris	
	False								True	
	2021		2021-06-28			2021-09-12			Domantas	
Sabonis			2001-08-28		230		13		Žalgiris	
	False								True	

Rezultatai atspausdinti į ekraną.

Jauniausi krepšininkai:

Marius;Grigonis;19;True

Krepšininkai žaidę Žalgiryje:

Jonas;Valanciunas;True

Marius;Grigonis;True

Mantas;Kalnietis;True

Domantas;Sabonis;True

```

Select C:\windows\system32\cmd.exe
Jauniausi krepšininkai:
Marius;Grigonis;19;True

Krepšininkai žaidę Žalgiryje:
Jonas;Valanciunas;True
Marius;Grigonis;True
Mantas;Kalnietis;True
Domantas;Sabonis;True

Press any key to continue . . .

```

5 pav. Atspausdinti rezultatai ekrane

Antras pavyzdys tikrina krepšininkus, kai nėra nei vieno krepšininko žaidusio Žalgiryje, du žaidėjai yra jauniausi, du žaidėjai yra dalyvavę abejose stovyklose ir nėra žaidėjų, kurių ūgis būtų 2 metrai ir daugiau.

Antrojo pavyzdžio duomenys iš .txt failo.

```

2020;2020-07-28;2020-11-12;Jonas;Valanciunas;2002-09-
28;175;14;Rytas;pakviestas;kapitonas
2020;2020-07-28;2020-11-12;Marius;Grigonis;2002-08-
28;162;24;Lietkabelis;pakviestas;žaidėjas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-
28;175;30;Rytas;pakviestas;žaidėjas

```

```
2021;2021-01-17;2021-10-12;Jonas;Valanciunas;2002-09-
28;175;14;Rytas;pakviestas;kapitonas
2021;2021-05-28;2021-11-12;Mantas;Kalnietis;2000-08-
28;175;30;Rytas;pakviestas;žaidėjas
2021;2021-06-28;2021-09-12;Domantas;Sabonis;2002-09-
28;160;13;Rytas;pakviestas;žaidėjas
```

Atspausdinta pradinių duomenų lentelė į .txt failą.

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas  |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

```
-----
-----
|      2020 | 2020-07-28      | 2020-11-12      | Jonas    |
Valanciunas | 2002-09-28      | 175 | 14 | Rytas      | True
| True      |
|      2020 | 2020-07-28      | 2020-11-12      | Marius   |
Grigonis   | 2002-08-28      | 162 | 24 | Lietkabelis | True
| False     |
|      2021 | 2021-05-28      | 2021-11-12      | Mantas   |
Kalnietis   | 2000-08-28      | 175 | 30 | Rytas      | True
| False     |
```

```
-----
-----
-----
| Metai      | Stovyklos pradžios data | Stovyklos pabaigos data | Vardas  |
Pavardė      | Gimimo metai | Ūgis | Numeris | Klubas      | Pakviestas į
komandą | Kapitonas arba ne |
```

```
-----
-----
|      2021 | 2021-01-17      | 2021-10-12      | Jonas    |
Valanciunas | 2002-09-28      | 175 | 14 | Rytas      | True
| True      |
|      2021 | 2021-05-28      | 2021-11-12      | Mantas   |
Kalnietis   | 2000-08-28      | 175 | 30 | Rytas      | True
| False     |
|      2021 | 2021-06-28      | 2021-09-12      | Domantas |
Sabonis     | 2002-09-28      | 160 | 13 | Rytas      | True
| False     |
```

Rezultatai atspausdinti į ekraną.

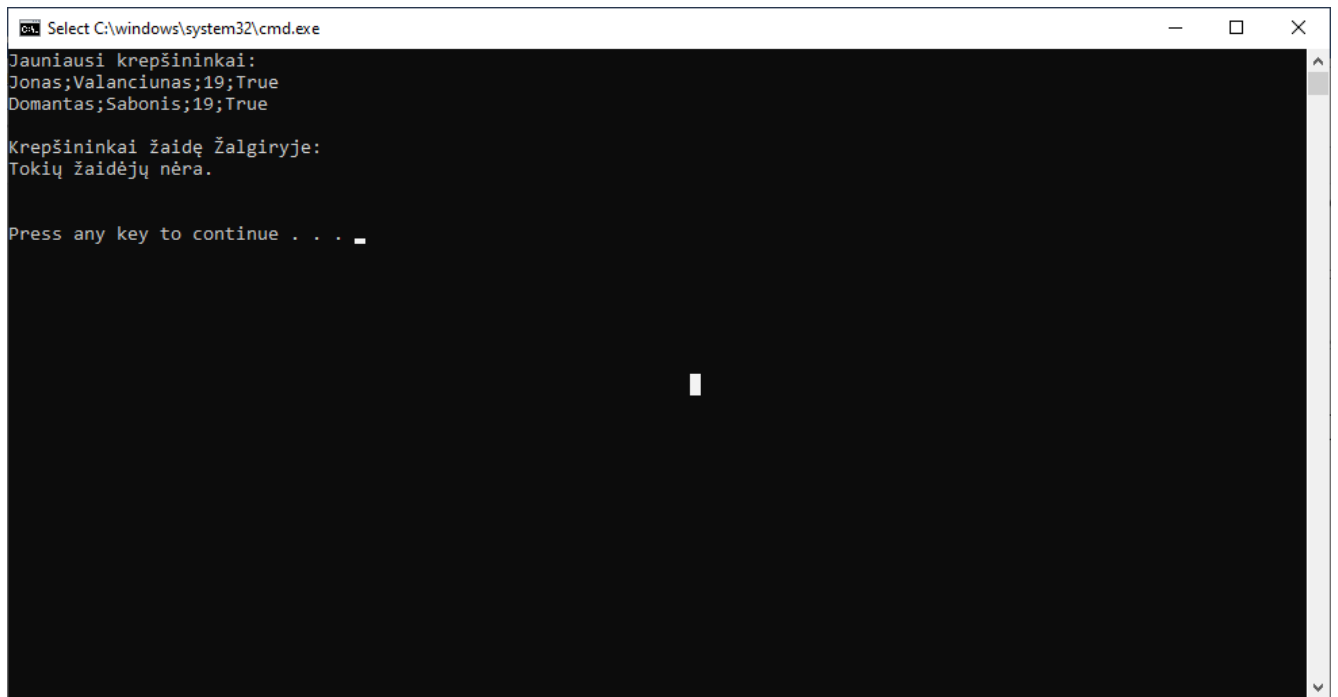
Jauniausi krepšininkai:

Jonas;Valanciunas;19;True

Domantas;Sabonis;19;True

Krepšininkai žaidę Žalgiryje:

Tokių žaidėjų nėra.

A screenshot of a Windows command prompt window. The title bar reads "Select C:\windows\system32\cmd.exe". The window contains the following text:

```
Jauniausi krepšininkai:  
Jonas;Valanciunas;19;True  
Domantas;Sabonis;19;True  
  
Krepšininkai žaidę Žalgiryje:  
Tokių žaidėjų nėra.  
  
Press any key to continue . . .
```

A cursor is visible on the line "Press any key to continue . . .".

6 pav. Atspausdinti rezultatai ekrane

3.4. Dėstytojo pastabos

L3 ataskaitos pastabos. 3.2 skyrelyje kodą reikėtų pateikti arba visą su nuspelvinta sintakse arba visą su nenuspalvinta sintakse ir būtinai vienodu šriftu. 3.3 skyrelyje nėra nuorodų į pateiktus paveikslėlius. Testavimo pavyzdžiai neatskirti vienas nuo kito. 3.3 skyrelyje aiškinamieji sakiniai turi būti Times New Roman šriftu, o failų turiniai - Courier New šriftu. Aiškinamieji sakiniai netikslūs. L3 ataskaita vertinama 0,8 tšk. Programos trūkumai:

1. Neteisingai užklotas Equals metodas;
2. Neuniversalaus sprendimo metodai;

Pratybų vertinimas: 0.5.

Testo balas: 2.

Galutinis įvertinimas: 8.

4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

U4L-13. Nesikartojantys žodžiai Dviejuose tekstiniuose failuose Knyga1.txt ir Knyga2.txt duotas tekstas sudarytas iš žodžių, atskirtų skyrikliais. Skyriklių aibė žinoma ir abejuose failuose yra ta pati. Raskite ir spausdinkite faile Rodikliai.txt:

- žodžių, kurie faile Knyga1.txt kartojasi tik po vieną kartą ir jų nėra faile Knyga2.txt, skaičių ir tokių žodžių sąrašą (ne daugiau nei 10 žodžių);
- žodžių, kurie yra abejuose failuose, skaičių ir tokių žodžių sąrašą (ne daugiau nei 10 žodžių), surikiuotą pagal pasikartojimo skaičių mažėjimo tvarka, o kai pasikartojimų skaičius sutampa – pagal abėcėlę;

4.2. Programos tekstas

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

namespace _13_Uzd_Nesikartojantys_žodžiai
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    class InOut
    {
        /// <summary>
        /// Prints information to .txt file
        /// </summary>
        /// <param name="words">array of strings</param>
        /// <param name="fileName">Specific file name</param>
        public static void SpecificWords(string[] words, string fileName)
        {
            int count = 0;
            using (var writer = File.AppendText(fileName))
            {
                foreach (string word in words)
                {
                    if (word != null)
                    {
                        writer.WriteLine(word);
                        count++; //Count of words
                    }
                }

                if (count == 0)
                {
                    writer.WriteLine("Tokių žodžių nėra.");
                    writer.WriteLine();
                }
                else
                {
                    writer.WriteLine("Žodžių skaičius:");
                    writer.WriteLine(count);
                    writer.WriteLine();
                }
            }
        }
    }
}
```

```

    }
}

}

}

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

namespace _13_Uzd_Nesikartojantys_zodziai
{
    /// <summary>
    /// Class that do all kinds of actions with strings and characters
    /// </summary>
    class TaskUtils
    {
        /// <summary>
        /// Formats an array of strings that repeat one time in first file and don not exist
        in a second file
        /// </summary>
        /// <param name="readFile1">Specific file name</param>
        /// <param name="readFile2">Specific file name</param>
        /// <param name="punctuation">String of punctuations</param>
        /// <returns>Formatted array of strings</returns>
        public static string[] WordsRepeatOneTime(string readFile1, string readFile2, string
punctuation)
        {
            string[] lines = File.ReadAllLines(readFile1, Encoding.UTF8);
            string[] words = new string[10];
            int x = 0;
            foreach (string line in lines)
            {
                if (line.Length > 0)
                {
                    string[] parts = Regex.Split(line, punctuation + "+");
                    foreach (string word in parts)
                    {
                        if (word.Length > 0)
                        {
                            int count = Count(word, lines, punctuation);
                            bool trueFalse = SecondFileMatch(word, readFile2, punctuation);

                            if (Contains(words, word) == false && count == 1 && trueFalse ==
false && x < 10)
                            {
                                words[x] = word;
                                x++;
                            }
                        }
                    }
                }
            }
            return words;
        }

        /// <summary>
        /// Finds out if a word contains in a array of words

```



```

/// </summary>
/// <param name="words">Array of words</param>
/// <param name="word">One word</param>
/// <returns>True of false</returns>
private static bool Contains(string[] words, string word)
{
    foreach (string Word in words)
    {
        if (Word == word)
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Counts how many times a word was repeated in the first file
/// </summary>
/// <param name="word">One word</param>
/// <param name="lines">Single line in a text</param>
/// <param name="punctuation">String of punctuations</param>
/// <returns>Counted integer</returns>
private static int Count(string word, string[] lines, string punctuation)
{
    int count = -1;
    string newWord = string.Format("(?<={0}){1}{2}+", punctuation, word, punctuation);
    Regex expression = new Regex(newWord);
    foreach (string line in lines)
    {
        if (line.Length > 0)
        {
            foreach (var MyMatch in expression.Matches(punctuation[1] + line +
punctuation[1]))
            {
                count++;
            }
        }
    }
    return count;
}

/// <summary>
/// Finds out if word was used in a second file
/// </summary>
/// <param name="word">One word</param>
/// <param name="readFile2">Specific file</param>
/// <param name="punctuation">String of punctuations</param>
/// <returns>True or false</returns>
private static bool SecondFileMatch(string word, string readFile2, string punctuation)
{
    string[] lines = File.ReadAllLines(readFile2, Encoding.UTF8);
    int count = 0;
    string newWord = string.Format("(?<={0}){1}{2}+", punctuation, word, punctuation);
    Regex expression = new Regex(newWord);
    foreach (string line in lines)
    {
        if (line.Length > 0)
        {
            foreach (var MyMatch in expression.Matches(punctuation[1] + line +
punctuation[1]))
            {
                count++;
            }
        }
    }
}

```

```

    }
}

if (count > 0)
{
    return true;
}
else
    return false;
}

/// <summary>
/// Formats an array of strings that repeat in the first and the second files, also
this method sorts the array
/// </summary>
/// <param name="readFile1">Specific file</param>
/// <param name="readFile2">Specific file</param>
/// <param name="punctuation">String of punctuations</param>
/// <returns>Formatted array of strings</returns>
public static string[] WordsThatRepeat(string readFile1, string readFile2, string
punctuation)
{
    string[] lines = File.ReadAllLines(readFile1, Encoding.UTF8);
    string[] words = new string[10];
    int[] count = new int[10];
    int x = 0;

    foreach (string line in lines)
    {
        if (line.Length > 0)
        {
            string[] parts = Regex.Split(line, punctuation + "+");

            foreach (string word in parts)
            {
                if (word.Length > 0)
                {
                    int howMany = TimesOfRepeat(word, readFile2, punctuation);

                    if (Contains(words, word) == false && howMany > 0 && x < 10)
                    {
                        words[x] = word;
                        count[x] = howMany;
                        x++;
                    }
                }
            }
        }
    }

    //Sort by count or alphabet
    for (int i = 0; i < x; i++)
    {
        int n = count[i];
        string m = words[i];
        for (int j = i + 1; j < x; j++)
        {
            if (count[i] < count[j] || count[i] == count[j] &&
words[i].CompareTo(words[j]) > 0)
            {
                count[i] = count[j];
                count[j] = n;
                n = count[i];

                words[i] = words[j];
                words[j] = m;
            }
        }
    }
}

```

```

        m = words[i];
    }
}

return words;
}

/// <summary>
/// Counts how many times a word was repeated in the second file
/// </summary>
/// <param name="word">One word</param>
/// <param name="fileName">Specific file</param>
/// <param name="punctuation">String of punctuations</param>
/// <returns>Counted integer</returns>
private static int TimesOfRepeat(string word, string fileName, string punctuation)
{
    string[] lines = File.ReadAllLines(fileName, Encoding.UTF8);
    string newWord = string.Format("(?<={0}){1}{2}+", punctuation, word, punctuation);
    Regex expression = new Regex(newWord);
    int count = 0;

    foreach (string line in lines)
    {
        if (line.Length > 0)
        {
            foreach (var MyMatch in expression.Matches(punctuation[1] + line +
punctuation[1]))
            {
                count++;
            }
        }
    }
    return count;
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using System.Text;
using System.Threading.Tasks;
using System.IO;
//Main function of this program is to format all kinds of text with simbols and characters
//Vytenis Kriščiūnas

namespace _13_Uzd_Nesikartojantys_žodžiai
{
    /// <summary>
    /// Main class
    /// </summary>
    class Program
    {
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd1 = @"Knyga1.txt";
        /// <summary>
        /// Represents a .txt file from which data will be read
        /// </summary>
        const string CFd2 = @"Knyga2.txt";
    }
}

```

```

/// <summary>
/// Represents a .txt file where data will be put
/// </summary>
const string CFr = "Rodikliai.txt";

static void Main(string[] args)
{
    File.Delete(CFr);
    string punctuation = "[ ,\\-!.;1]";
    string[] singleWords = TaskUtils.WordsRepeatOneTime(CFd1, CFd2, punctuation);
    string[] lines = new string[2];

    lines[0] = string.Format("Žodžiai, kurie kartojasi tik po vieną kartą ir jų nėra  
antrame faile .txt:\n");
    File.AppendAllText(CFr, lines[0], Encoding.UTF8);
    InOut.SpecificWords(singleWords, CFr);

    lines[1] = string.Format("Žodžiai, kurie yra abejuose failuose surikiuoti  
pasikartojimo skaičių mažėjimo tvarka, o kai pasikartojimų skaičius sutampa – pagal  
abėcėlę:\n");
    File.AppendAllText(CFr, lines[1], Encoding.UTF8);
    string[] multiWords = TaskUtils.WordsThatRepeat(CFd1, CFd2, punctuation);
    InOut.SpecificWords(multiWords, CFr);

    }
}

```

4.3. Pradiniai duomenys ir rezultatai

1. Pirmas pavyzdys.

Abeji tekstiniai failai identiški, todėl randami tik žodžiai, kurie kartojasi ir pirmame ir antrame tekstiniuose failuose po vieną kartą. Randami dešimt žodžių, kurie tenkina antrąjį uždavinio sąlygą.

Pirmas tekstinis failas:

Anksčiau neteistas, apie 25 metus taksi vairuotoju dirbęs buvęs Vilniaus kriminalinės policijos pareigūnas I. Piaseckis tikina, kad ne jis turi sėdėti kaltinamųjų suole esą dvi gyvybes nusinešusią avariją sukėlė kito automobilio vairuotoja. Bet tokiai jo pozicijai nepritaria ne tik valstybinį kaltinimą palaikantys prokurorai, į baisią avariją patekę kitų transporto priemonių vairuotojai, bet ir savo artimųjų netekę nukentėjusieji. Maža to, baudžiamojoje byloje yra net vaizdo įrašas, kuriame užfiksuota, kaip I. Piaseckio vairuojamas automobilis sukelia avariją, kuri baigėsi net penkių automobilių kolonos susidūrimu.

Skaitykite daugiau:

Antras tekstinis failas:

Anksčiau neteistas, apie 25 metus taksi vairuotoju dirbęs buvęs Vilniaus kriminalinės policijos pareigūnas I. Piaseckis tikina, kad ne jis turi sėdėti kaltinamųjų suole esą dvi gyvybes nusinešusią avariją sukėlė kito automobilio vairuotoja. Bet tokiai jo pozicijai nepritaria ne tik valstybinį

kaltinimą palaikantys prokurorai, į baisią avariją patekę kitų transporto priemonių vairuotojai, bet ir savo artimųjų netekę nukentėjusieji. Maža to, baudžiamojoje byloje yra net vaizdo įrašas, kuriame užfiksuota, kaip I. Piaseckio vairuojamas automobilis sukelia avariją, kuri baigėsi net penkių automobilių kolonos susidūrimu.

Skaitykite daugiau:

Rezultatai iš Rodikliai.txt:

Žodžiai, kurie kartojasi tik po vieną kartą ir jų nėra antrame faile .txt:
Tokių žodžių nėra.

Žodžiai, kurie yra abėjuose failuose surikiuoti pasikartojimo skaičių mažėjimo tvarka, o kai pasikartojimų skaičius sutampa – pagal abėcėlę:

25

Anksčiau

apie

buvęs

dirbęs

metus

neteistas

taksi

vairuotoju

Vi

Žodžių skaičius:

10

2. Antras pavyzdys.

Abeji tekstiniai failai identiški išskyrus žodį: tikrina, kuris pirmame tekstiniame faile kartojasi vieną kartą ir šio žodžio nėra antrame tekstiniame faile. Randamas vienas žodis, kuris tenkina pirmąją uždavinio sąlygą ir dešimt žodžių, kurie tenkina antrąją uždavinio sąlygą

Pirmas tekstinis failas:

Anksčiau neteistas, apie 25 metus taksi vairuotoju dirbęs buvęs Vilniaus kriminalinės policijos pareigūnas I. Piaseckis tikina; tikina, kad ne jis turi sėdėti kaltinamųjų suole esą dvi gyvybes nusinešusią avariją sukėlė kito automobilio vairuotoja. Bet tokiai jo pozicijai nepitaria ne tik valstybinį kaltinimą palaikantys prokurorai, į baisią avariją patekę kitų transporto priemonių vairuotojai, bet ir savo artimųjų netekę nukentėjusieji. Maža to, baudžiamojoje byloje yra net vaizdo įrašas, kuriame užfiksuota, kaip I. Piaseckio vairuojamas automobilis sukelia avariją, kuri baigėsi net penkių automobilių kolonos susidūrimu.

Skaitykite daugiau:

Antras tekstinis failas:

Anksčiau neteistas, apie 25 metus taksi vairuotoju dirbęs buvęs Vilniaus kriminalinės policijos pareigūnas I. Piaseckis, kad ne jis turi sėdėti kaltinamųjų suole esą

dvi gyvybes nusinešusią avariją sukėlė kito automobilio vairuotoja. Bet tokiai jo pozicijai nepritaria ne tik valstybinį kaltinimą palaikantys prokurorai, į baisią avariją patekę kitų transporto priemonių vairuotojai, bet ir savo artimųjų netekę nukentėjusieji. Maža to, baudžiamojoje byloje yra net vaizdo įrašas, kuriame užfiksuota, kaip I. Piaseckio vairuojamas automobilis sukelia avariją, kuri baigėsi net penkių automobilių kolonos susidūrimu.

Skaitykite daugiau:

Rezultatai iš Rodikliai.txt:

Žodžiai, kurie kartojasi tik po vieną kartą ir jų nėra antrame faile .txt:

tikina

Žodžių skaičius:

1

Žodžiai, kurie yra abėjuose failuose surikiuoti pasikartojimo skaičių mažėjimo tvarka, o kai pasikartojimų skaičius sutampa – pagal abėcėlę:

25

Anksčiau

apie

buvęs

dirbęs

metus

neteistas

taksi

vairuotoju

Vi

Žodžių skaičius:

10

4.4. Dėstytojo pastabos