KAUNO TECHNOLOGIJOS UNIVERSITETAS
IFORMATIKOS FAKULTETAS

OBJEKTINIO PROGRAMAVIMO PAGRINDAI I (P175B117)

Laboratorinių darbų ataskaita

Atliko:

IFIN-5/3gr. studentas

Benjaminas Čeikauskas

2025 m. gruodžio 16 d.

Priėmė:

dr. Lina Narbutaitė

KAUNAS 2025

TURINYS

1. Pažintis su klase 3

1.1. Darbo užduotis 3

1.2. Programos tekstas 3

1.3. Pradiniai duomenys ir rezultatai 7

1.4. Dėstytojo pastabos 8

2. Objektų rinkinys 9

2.1. Darbo užduotis 9

2.2. Programos tekstas 9

2.3. Pradiniai duomenys ir rezultatai 14

2.4. Dėstytojo pastabos 17

3. Konteinerinė klasė 18

3.1. Darbo užduotis 18

3.2. Programos tekstas 18

3.3. Pradiniai duomenys ir rezultatai 23

3.4. Dėstytojo pastabos 27

4. Teksto analizė ir redagavimas 28

4.1. Darbo užduotis 28

4.2. Programos tekstas 28

4.3. Pradiniai duomenys ir rezultatai 33

4.4. Dėstytojo pastabos 35

5. Susieti rinkiniai 36

5.1. Darbo užduotis 36

5.2. Programos tekstas 36

5.3. Pradiniai duomenys ir rezultatai 44

5.4. Dėstytojo pastabos 49

1. Pažintis su klase

1.1. Darbo užduotis

U1–23. Kepykla. • Sukurkite klasę Duona, kuri turėtų kintamuosius duonos miltų pavadinimui, kepaliuko svoriui ir kainai saugoti. Kepykla kepa duoną iš 3 skirtingų rūšių miltų. Raskite, kuri kepama duona mažiausiai sveria ir kurios duonos kaina didžiausia. • Papildykite klasę Duona kintamuoju, skirtu duonos kepaliuko užimamam ant lentynos plotui saugoti. Sukurkite klasę Kepykla, kuri turėtų kintamąjį kepyklos pavadinimui saugoti ir 3 kintamuosius (kiekvienai duonos miltų rūšiai), skirtus saugoti per pamainą reikalingam iškepti duonos kepaliukų kiekiui. Suskaičiuokite, kiek ploto lentynų kiekvienos rūšies duonai sudėti reikės. • Papildykite klasę Kepykla kintamuoju, kuriame būtų saugoma informacija apie tai, kiek kg duonos kepyklos automobilis gali vežti. Kiek kartų r

1.2. Programos tekstas

```
//IFIN-5/3_Čeikauskas_Benjaminas_U1_23
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
namespace U1_23_Kepykla
{
    /// <summary>
    /// Klasė duonos kepalo duomenims aprašyti
    /// </summary>
    class Bread
    {
        private string fType;       //Miltų tipas
        private double weight,      //Svoris Gramais
                       price,       //Kaina Eurais
                       shelfArea;   //Užimama vieta lentynoje kvadratiniais cm

        public Bread()
        {

        }
        public Bread(string fType, double weight, double price, double shelfArea)
        {
            this.fType = fType;
            this.weight = weight;
            this.price = price;
            this.shelfArea = shelfArea;
        }
        public string GetFlourType() { return fType; }
        public double GetWeight() { return weight; }
        public double GetPrice() { return price; }
        public double GetShelfArea() { return shelfArea; }
    }
    /// <summary>
    /// Klasė kepyklos duomenims aprašyti
    /// </summary>
    class Bakery
    {
        private string name;            //Kepyklos pavadinimas
        private int kCount,             //Kvietinės duonos skaičius
                    aCount,             //Avižinės duonos skaičius
                    gCount;             //Grikių duonos skaičius
        private double carCapacity;   //Automobilio talpa kilogramais
        public Bakery()
        {
```

```
        }
        public Bakery(string name, int kCount, int aCount, int gCount, double truckCapacity)
        {
            this.name = name;
            this.kCount = kCount;
            this.aCount = aCount;
            this.gCount = gCount;
            this.carCapacity = truckCapacity;
        }
        public string GetName() { return name; }
        public int GetKCount() { return kCount; }
        public int GetACount() { return aCount; }
        public int GetGCount() { return gCount; }
        public double GetCarCapacity() { return carCapacity; }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            Bread b1, b2, b3;
            Console.Write("Įveskite duonos miltų tipą:");
            string type = Console.ReadLine();
            Console.Write("Įveskite duonos svorį gramais:");
            double weight = double.Parse(Console.ReadLine());
            Console.Write("Įveskite duonos kainą eurais:");
            double price = double.Parse(Console.ReadLine());
            Console.Write("Įveskite duonos užimamą plotą lentynoje kvadratiniais cm:");
            double shelfArea = double.Parse(Console.ReadLine());

            b1 = new Bread(type, weight, price, shelfArea);
            b2 = new Bread("Avižinių", 930, 2.37, 364);
            b3 = new Bread("Grikių", 500, 3.19, 200);
            PrintBread(b1, b2, b3);
            double minWeight = GetMinWeight(b1, b2, b3);
            PrintLowestWeightBread(b1, b2, b3, minWeight);
            double maxPrice = GetMaxPrice(b1, b2, b3);
            PrintHighestPriceBread(b1, b2, b3, maxPrice);

            Bakery bakery1 = new Bakery("Master Baker", 151, 100, 50, 123);
            PrintBakery(bakery1, b1, b2, b3);

            Console.WriteLine("Visa kvietinių miltų duona užima: {0}cm²\n" +
                              "Visa avižinių miltų duona užima: {1}cm²\n" +
                              "Visa grikių miltų duona užima: {2}cm²",
                              bakery1.GetKCount() * b1.GetShelfArea(),
                              bakery1.GetACount() * b2.GetShelfArea(),
                              bakery1.GetGCount() * b3.GetShelfArea());
            Console.WriteLine();
            double weightTotal = GetTotalBreadWeight(bakery1, b1, b2, b3);
            double routeCount = GetCarRouteCountPerShift(weightTotal, bakery1);
            PrintCarRouteCountPerShift(weightTotal, routeCount);
        }
        /// <summary>
        /// Atspausdina lentelę su duonos tipais, jų svoriu, kaina ir užimamu plotu
        /// </summary>
        /// <param name="b1"></param>
        /// <param name="b2"></param>
        /// <param name="b3"></param>
        static void PrintBread(Bread b1, Bread b2, Bread b3)
        {
            Console.WriteLine("-------------------------------------------------------------" +
                              "----------------------");
            Console.WriteLine("Duonos (miltų tipas,\t Svoris,\t Kaina,\t Užimamas plotas
lentynoje:)");
            Console.WriteLine("-------------------------------------------------------------" +
                              "----------------------");
            Console.WriteLine("{0},\t {1, 14:f2}g, {2, 10:f2}Eur, {3, 10:f2}cm²",
```

```
                                b1.GetFlourType(), b1.GetWeight(), b1.GetPrice(),
b1.GetShelfArea());
            Console.WriteLine("{0},\t {1, 14:f2}g, {2, 10:f2}Eur, {3, 10:f2}cm²",
                                b2.GetFlourType(), b2.GetWeight(), b2.GetPrice(),
b2.GetShelfArea());
            Console.WriteLine("{0},\t {1, 22:f2}g, {2, 10:f2}Eur, {3, 10:f2}cm²",
                                b3.GetFlourType(), b3.GetWeight(), b3.GetPrice(),
b3.GetShelfArea());
            Console.WriteLine("------------------------------------------------------------" +
                                "----------------------");
            Console.WriteLine();
        }
        /// <summary>
        /// Atspausdina Kepyklos per pamainą reikalingos iškepti duonos kiekį ir tipą
        /// </summary>
        /// <param name="bakery"></param>
        /// <param name="bk"></param>
        /// <param name="ba"></param>
        /// <param name="bg"></param>
        static void PrintBakery(Bakery bakery, Bread bk, Bread ba, Bread bg)
        {
            Console.WriteLine("------------------------------------------------------------" +
                                "----------------------");
            Console.WriteLine("Kepykla {0} per pamainą turi iškepti: ", bakery.GetName());
            Console.WriteLine("------------------------------------------------------------" +
                                "----------------------");
            Console.WriteLine("(Duonos miltų tipas,\t kiekis:)");
            Console.WriteLine("{0}\t {1, 14:D}", bk.GetFlourType(), bakery.GetKCount());
            Console.WriteLine("{0}\t {1, 14:D}", ba.GetFlourType(), bakery.GetACount());
            Console.WriteLine("{0}\t {1, 22:D}", bg.GetFlourType(), bakery.GetGCount());
            Console.WriteLine("------------------------------------------------------------" +
                                "----------------------");
            Console.WriteLine();
        }
        /// <summary>
        /// Apskaičiuoja mažiausią duonos svorį
        /// </summary>
        /// <param name="b1"></param>
        /// <param name="b2"></param>
        /// <param name="b3"></param>
        /// <returns></returns>
        static double GetMinWeight(Bread b1, Bread b2, Bread b3)
        {
            double minWeight = b1.GetWeight();
            if (b2.GetWeight() < minWeight)
            {
                minWeight = b2.GetWeight();
            }
            if (b3.GetWeight() < minWeight)
            {
                minWeight = b3.GetWeight();
            }
            return (minWeight);
        }
        /// <summary>
        /// Apskaičiuoja aukščiausią duonos kainą
        /// </summary>
        /// <param name="b1"></param>
        /// <param name="b2"></param>
        /// <param name="b3"></param>
        /// <returns></returns>
        static double GetMaxPrice(Bread b1, Bread b2, Bread b3)
        {
            double maxPrice = b1.GetPrice();
            if (b2.GetPrice() > maxPrice)
            {
                maxPrice = b2.GetPrice();
            }
```

```
    if (b3.GetPrice() > maxPrice)
    {
        maxPrice = b3.GetPrice();
    }
    return (maxPrice);
}
/// <summary>
/// Spausdina duonas kurios sveria mažiausiai
/// </summary>
/// <param name="b1"></param>
/// <param name="b2"></param>
/// <param name="b3"></param>
/// <param name="minWeight"></param>
static void PrintLowestWeightBread(Bread b1, Bread b2, Bread b3, double minWeight)
{
    double lowestWeight = minWeight;
    Console.WriteLine("Mažiausio svorio duona:");
    if (b1.GetWeight() == lowestWeight)
    {
        Console.WriteLine("{0}", b1.GetFlourType());
    }
    if (b2.GetWeight() == lowestWeight)
    {
        Console.WriteLine("{0}", b2.GetFlourType());
    }
    if (b3.GetWeight() == lowestWeight)
    {
        Console.WriteLine("{0}", b3.GetFlourType());
    }
    Console.WriteLine();
}
/// <summary>
/// Spausdina duonas kurios turi aukščiausią kainą
/// </summary>
/// <param name="b1"></param>
/// <param name="b2"></param>
/// <param name="b3"></param>
static void PrintHighestPriceBread(Bread b1, Bread b2, Bread b3, double maxPrice)
{
    double highestPrice = maxPrice;
    Console.WriteLine("Didžiausios kainos duona:");
    if (b1.GetPrice() == highestPrice)
    {
        Console.WriteLine("{0}", b1.GetFlourType());
    }
    if (b2.GetPrice() == highestPrice)
    {
        Console.WriteLine("{0}", b2.GetFlourType());
    }
    if (b3.GetPrice() == highestPrice)
    {
        Console.WriteLine("{0}", b3.GetFlourType());
    }
    Console.WriteLine();
}
/// <summary>
/// Apskaičiuoja bendrą duonos svorį per pamainą kilogramais
/// </summary>
/// <param name="bakery"></param>
/// <param name="bk"></param>
/// <param name="ba"></param>
/// <param name="bg"></param>
/// <returns></returns>
static double GetTotalBreadWeight(Bakery bakery, Bread bk, Bread ba, Bread bg)
{
    return (double)(bakery.GetKCount() * bk.GetWeight() +
                    bakery.GetACount() * ba.GetWeight() +
                    bakery.GetGCount() * bg.GetWeight()) / 1000;
```

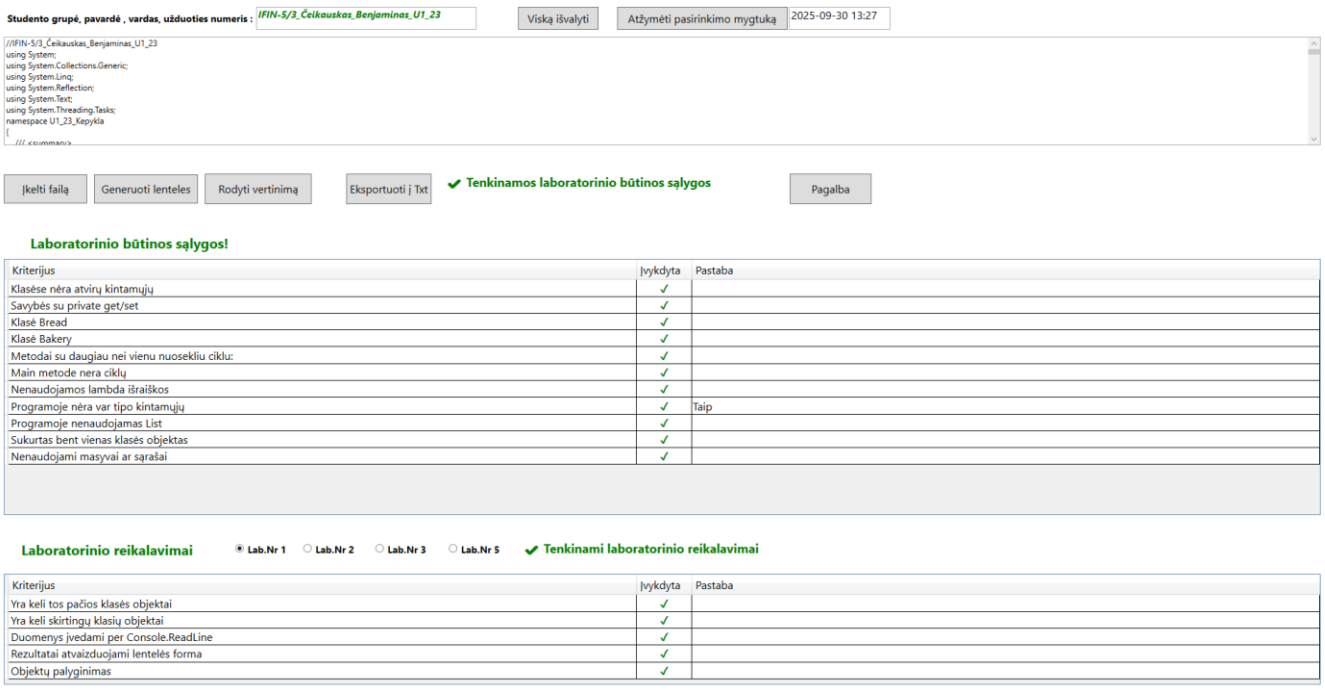
```
        }
        /// <summary>
        /// Apskaičiuoja kepyklos automobilio važiavimų skaičių per pamainą
        /// </summary>
        /// <param name="weightTotal"></param>
        /// <param name="bakery"></param>
        /// <returns></returns>
        static double GetCarRouteCountPerShift(double weightTotal, Bakery bakery)
        {
            double routeCount = weightTotal / bakery.GetCarCapacity();

            if (routeCount == 0)
                return  (routeCount);
            else if (routeCount % 1 == 0)
                return (routeCount);
            else
                return (routeCount - (routeCount % 1) + 1);

        }
        /// <summary>
        /// Atspausdina bendrą duonos svorį ir automobilio važiavimų skaičių per pamainą
        /// </summary>
        /// <param name="weightTotal"></param>
        /// <param name="routeCount"></param>
        static void PrintCarRouteCountPerShift(double weightTotal, double routeCount)
        {
            Console.WriteLine("Bendras duonos svoris {0}kg", weightTotal);
            Console.WriteLine();

            if (routeCount == 0)
                Console.WriteLine("Automobiliui važiuoti nereikės");
            else if (routeCount % 1 == 0)
            {
                Console.WriteLine("Automobiliui važiuoti reiks {0:f0} kart.", routeCount);
                Console.WriteLine();
            }
        }
    }
}
```

1.3. Pradiniai duomenys ir rezultatai

Rezultatai ekrane (konsolėje)

```
Iveskite duonos miltu tipa:Kvietiniai
Iveskite duonos svori gramais:500
Iveskite duonos kaina eurais:2.2
Iveskite duonos uzimama plota lentynoje kvadratiniais cm:400
--------------------------------------------------------------------------------
Duonos (miltu tipas,     Svoris,         Kaina,  Uzimamas plotas lentynoje:)
--------------------------------------------------------------------------------
Kvietiniai,              500.00g,       2.20Eur,     400.00cm²
Aviziniu,                930.00g,       2.37Eur,     364.00cm²
Grikiu,                  500.00g,       3.19Eur,     200.00cm²
--------------------------------------------------------------------------------

Maziausio svorio duona:
Kvietiniai
Grikiu

Didziausios kainos duona:
Grikiu

--------------------------------------------------------------------------------
Kepykla Master Baker per pamaina turi iskepti:
--------------------------------------------------------------------------------
(Duonos miltu tipas,     kiekis:)
Kvietiniai                  151
Aviziniu                    100
Grikiu                       50
--------------------------------------------------------------------------------

Visa kvietiniu miltu duona uzima: 60400cm²
Visa aviziniu miltu duona uzima: 36400cm²
Visa grikiu miltu duona uzima: 10000cm²

Bendras duonos svoris 193.5kg

Automobiliui vaziuoti reiks 2 kart.
```

1.4. Dėstytojo pastabos



2. Objektų rinkinys

2.1. Darbo užduotis

U2–23. Saldainiai • Duomenys apie saldainius yra faile: pavadinimas, tipas ir kilogramo kaina. Pirmoje eilutėje yra savininko pavardė ir vardas. Sukurkite klasę Saldainis vieno saldainio duomenims saugoti. Kiek kainuoja saldainiai, jeigu kiekvieno pavadinimo yra po n kilogramų. Kokie nurodyto tipo saldainiai brangiausi? Jei yra keli, spausdinkite visus. • Papildykite programą veiksmais su dviejų studentų saldainių rinkiniais. Kiekvieno rinkinio duomenys saugomi atskiruose failuose. Kurio studento rinkinys kainuoja brangiau, jeigu pirmasis studentas turi visų saldainių pavadinimų po n1 kilogramų, o antrasis – po n2? Jei abiejų, spausdinkite abu .Surašykite į atskirą rinkinį visus abiejų studentų saldainių, kurių kaina didesne už k pinigų, duomenis

2.2. Programos tekstas

```
//IFIN-5/3_Čeikauskas_Benjaminas_U2_23
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
namespace Lab2
{
    /// <summary>
    /// Klasė saldainio duomenims saugoti
    /// </summary>
    class Candy
    {
        private string candyName; // saldainio pavadinimas
        private string candyType; // saldainio tipas
        private double candyPrice; // kilogramo kaina eurais
        public Candy()
        {

        }
        public Candy(string candyName, string candyType, double candyPrice)
        {
            this.candyName = candyName;
            this.candyType = candyType;
            this.candyPrice = candyPrice;

        }
        public string GetCandyName() {  return candyName; }
        public string GetCandyType() { return candyType; }
        public double GetCandyPrice() { return candyPrice; }
    }
    internal class Program
    {
        const int Cn = 100;
        const string CFcTomas = "TomasCandy.txt";
        const string CFcAusteja = "AustejaCandy.txt";
        const string CFresults = "Results.txt";
        static void Main(string[] args)
        {
            if(File.Exists(CFresults))
                File.Delete(CFresults);

            Candy[] C1 = new Candy[Cn]; //Tomo saldainių rinkinys
            string ownerName1;
            int n1;
            Read(CFcTomas, C1, out ownerName1, out n1);
            Console.WriteLine("-------------------------------------------------");
            Console.WriteLine("---------------SALDAINIŲ DUOMENYS----------------");
```

```
Console.WriteLine("-----------------------------------------------------");
Console.WriteLine("{0} turi {1} skirtingų saldainių", ownerName1, n1);
PrintCandiesData(C1, n1);

Candy[] C2 = new Candy[Cn]; //Austėjos saldainių rinkinys
string ownerName2;
int n2;
Read(CFcAusteja, C2, out ownerName2, out n2);
Console.WriteLine("{0} turi {1} skirtingų saldainių", ownerName2, n2);
PrintCandiesData(C2, n2);

double candyKg1;
Console.WriteLine("Įveskite po kiek kilogramų turi kiekvienas Tomo saldainis");
candyKg1 = double.Parse(Console.ReadLine());

double totalCandyPrice1;
GetTotalCandyPrice(C1, n1, candyKg1, out totalCandyPrice1);
Console.WriteLine("kai kiekvienas pavadinimas yra po {0}kg " +
    "{1} bendra saldainių kaina {2} eur", candyKg1, ownerName1, totalCandyPrice1);
Console.WriteLine();

double candyKg2;
Console.WriteLine("Įveskite po kiek kilogramų turi kiekvienas Austėjos" +
                  " saldainis");
candyKg2 = double.Parse(Console.ReadLine());

double totalCandyPrice2;
GetTotalCandyPrice(C2, n2, candyKg2, out totalCandyPrice2);
Console.WriteLine("kai kiekvienas pavadinimas yra po {0}kg " +
    "{1} bendra saldainių kaina {2} eur", candyKg2, ownerName2, totalCandyPrice2);
Console.WriteLine();

if (totalCandyPrice1 > totalCandyPrice2)
{
    Console.WriteLine("Daugiau kainuoja {0} saldainių rinkinys", ownerName1);
}
else if (totalCandyPrice1 < totalCandyPrice2)
{
    Console.WriteLine("Daugiau kainuoja {0} saldainių rinkinys", ownerName2);
}
else if (totalCandyPrice1 == totalCandyPrice2)
{
    Console.WriteLine("{0} ir {1} saldainių rinkiniai kainuoja po lygiai",
        ownerName1, ownerName2);
}

double highestCandyPrice1;
GetHighestPrice(C1, n1, out highestCandyPrice1);
PrintHighestPriceCandy(C1, n1, ownerName1, highestCandyPrice1);

double highestCandyPrice2;
GetHighestPrice(C2, n2, out highestCandyPrice2);
PrintHighestPriceCandy(C2, n2, ownerName2, highestCandyPrice2);

double filteringPrice; //kaina pagal kuria sukuriamas rinkinys
Console.WriteLine("Įveskite kainą už kurią bus filtruojami didesnės " +
                  "kainos saldainiai");
filteringPrice = double.Parse(Console.ReadLine());

Candy[] Cfiltered1;
int nf1; //naujo masyvo atfiltruotų saldainių skaičius
GetNewCandiesArrayByPrice(C1, n1, filteringPrice, out Cfiltered1, out nf1);

Candy[] Cfiltered2;
int nf2; //naujo masyvo atfiltruotų saldainių skaičius
GetNewCandiesArrayByPrice(C2, n2, filteringPrice, out Cfiltered2, out nf2);

Candy[] CfilteredTotal;
```

```
    int nfTotal;
    MergeCandiesArrays(Cfiltered1, nf1, Cfiltered2, nf2, out CfilteredTotal,
                       out nfTotal);
    PrintCandiesData(CfilteredTotal, nfTotal);

    PrintCandiesDataToResults(CFresults, C1, n1, ownerName1);
    PrintCandiesDataToResults(CFresults, C2, n2, ownerName2);
    string TBfiltered = "Filtruotų";
    PrintCandiesDataToResults(CFresults, CfilteredTotal, nfTotal, TBfiltered);

    Console.WriteLine("Programa baigė darbą");
}
/// <summary>
/// Masyvas nuskaitantis dokumentą
/// </summary>
/// <param name="fv"></param>
/// <param name="candies"></param>
/// <param name="ownerName"></param>
/// <param name="n"></param>
static void Read(string fv, Candy[] candies, out string ownerName, out int n)
{
    string candyName;
    string candyType;
    double candyPrice;

    using (StreamReader reader = new StreamReader(fv))
    {
        string line;
        line = reader.ReadLine();
        string[] parts;
        ownerName = line;
        int i = 0;
        while ((line = reader.ReadLine()) != null && (i < Cn))
        {
            parts = line.Split(';');
            candyName = parts[0];
            candyType = parts[1];
            candyPrice = double.Parse(parts[2]);
            candies[i++] = new Candy(candyName, candyType, candyPrice);
        }
        n = i;
    }
}
/// <summary>
/// Masyvas spausdinantis saldainių masyvo informaciją
/// </summary>
/// <param name="C"></param>
/// <param name="n"></param>
static void PrintCandiesData(Candy[] C, int n)
{
        if (n > 0)
    {
        Console.WriteLine("-------------------------------------------------");
        Console.WriteLine("Pavadinimas     |\tTipas    |\tKaina     |");
        Console.WriteLine("-------------------------------------------------");
        for (int i = 0; i < n; i++)
        {
            Console.WriteLine("{0, -15} | {1, -15}| {2, -15}|",
                C[i].GetCandyName(), C[i].GetCandyType(), C[i].GetCandyPrice());
        }
    }
    else
    {
        Console.WriteLine("-------------------------------------------------");
        Console.WriteLine("               Masyvas tuščias                   ");
        Console.WriteLine("-------------------------------------------------");
    }
        Console.WriteLine("-------------------------------------------------");
```

```
}
/// <summary>
/// Metodas skaičiuojantis bendrą saldainių masyvo kainą
/// </summary>
/// <param name="C"></param>
/// <param name="n"></param>
/// <param name="candyKg"></param>
/// <param name="totalCandyPrice"></param>
static void GetTotalCandyPrice(Candy[] C, int n, double candyKg,
                               out double totalCandyPrice)
{
    totalCandyPrice = 0;
    for (int i = 0;i < n; i++)
    {
        totalCandyPrice += C[i].GetCandyPrice() * candyKg;
    }
}
static void GetHighestPrice(Candy[] C, int n, out double highestPrice)
{
    highestPrice = 0;
    for(int i = 0; i<n; i++)
    {
        if (highestPrice < C[i].GetCandyPrice())
        {
            highestPrice = C[i].GetCandyPrice();
        }
    }
}
/// <summary>
/// Metodas spausdinantis aukščiausios kainos saldainių tipus
/// </summary>
/// <param name="C"></param>
/// <param name="n"></param>
/// <param name="ownerName"></param>
/// <param name="highestPrice"></param>
static void PrintHighestPriceCandy(Candy[] C, int n, string ownerName,
                                   double highestPrice)
{
    Console.WriteLine("-----------------------------------------------------");
    Console.WriteLine("Didžiausios kainos {0, -10} saldainų tipai\t", ownerName);
    Console.WriteLine("-----------------------------------------------------");

    HashSet<string> printedTypes = new HashSet<string>();

    for (int i = 0; i < n; i++)
    {
        if (highestPrice == C[i].GetCandyPrice() &&
            !printedTypes.Contains(C[i].GetCandyType()))
        {
            Console.WriteLine("\t{0,23}", C[i].GetCandyType());
            printedTypes.Add(C[i].GetCandyType());
        }
    }
    Console.WriteLine("-----------------------------------------------------");
}
/// <summary>
/// Metodas išrenkantis saldainius pagal kainą į atskirą masyvą
/// </summary>
/// <param name="C"></param>
/// <param name="n"></param>
/// <param name="filteringPrice"></param>
/// <param name="Cfiltered"></param>
/// <param name="nf"></param>
static void GetNewCandiesArrayByPrice(Candy[] C, int n, double filteringPrice,
                                      out Candy[] Cfiltered, out int nf)
{
    Cfiltered = new Candy[n];
    nf = 0;
```

```
    for (int i = 0; i < n; i++)
    {
        if (C[i].GetCandyPrice() > filteringPrice)
        {
            Cfiltered[nf] = C[i];
            nf++;
        }
    }
}
/// <summary>
/// Metodas sujungiantis masyvus
/// </summary>
/// <param name="C1"></param>
/// <param name="n1"></param>
/// <param name="C2"></param>
/// <param name="n2"></param>
/// <param name="Cmerged"></param>
/// <param name="nm"></param>
static void MergeCandiesArrays(Candy[] C1, int n1, Candy[] C2, int n2,
                               out Candy[] Cmerged, out int nm)
{
    nm = n1 + n2;
    Cmerged = new Candy[nm];

    for (int i=0; i<nm; i++)
    {
        if (i < n1)
        {
            Cmerged[i] = C1[i];
        }
        else
        {
            Cmerged[i] = C2[i-n1];
        }

    }
}
/// <summary>
/// Metodas spausdinantis lentelę į nurodomą failą.
/// </summary>
/// <param name="fv"></param>
/// <param name="C"></param>
/// <param name="n"></param>
/// <param name="tableName"></param>
static void PrintCandiesDataToResults(string fv, Candy[] C, int n, string tableName)
{
    using (var fr = File.AppendText(fv))
    {

        fr.WriteLine("{0} saldainių duomenys", tableName);
        if (n > 0)
        {
            fr.WriteLine("-------------------------------------------------");
            fr.WriteLine("Pavadinimas     |\tTipas    |\tKaina     |");
            fr.WriteLine("-------------------------------------------------");
            for (int i = 0; i < n; i++)
            {
                fr.WriteLine("{0, -15} | {1, -15}| {2, -15}|",
                    C[i].GetCandyName(), C[i].GetCandyType(), C[i].GetCandyPrice());
            }
        }
        else
        {
            fr.WriteLine("-------------------------------------------------");
            fr.WriteLine("               Masyvas tuščias                   ");
            fr.WriteLine("-------------------------------------------------");
        }
```

```
                fr.WriteLine("-------------------------------------------------");
            }
        }
    }
}
```

2.3. Pradiniai duomenys ir rezultatai

Duomenų failas „TomasCandy.txt“

```
Tomas Kazlauskas
MėtaFresh;Kietas; 0.59;
Karvutė;Karamelė; 0.45;
Malonumas;Šokoladas; 1.20;
Citrinukas;Guminukas; 0.35;
Medutis;Karamelė; 0.50;
Riešutukas;Šokoladas; 1.45;
Vaisinukas;Guminukas; 1.45;
Pupelė;Kietas; 0.65;
Braškytė;Guminukas; 1.45;
Perlas;Šokoladas; 1.10;
```

Duomenų failas „AustejaCandy.txt“

```
Austėja Petraitienė
Citrina;Guminukas; 0.72;
Džiaugsmas;Karamelė; 0.55;
Sapnas;Šokoladas; 1.35;
Ledukas;Kietas; 0.48;
Lašas;Karamelė; 0.63;
Debesėlis;Šokoladas; 1.25;
Sprogimas;Guminukas; 0.41;
Kava;Kietas; 0.69;
Obuolys;Guminukas; 0.39;
Skonis;Šokoladas; 1.18;
```

Rezultatai ekrane (konsolėje)

```
--------------------------------------------------
---------------SALDAINIU DUOMENYS-----------------
--------------------------------------------------
Tomas Kazlauskas turi 10 skirtingu saldainiu
--------------------------------------------------
Pavadinimas     |       Tipas    |      Kaina     |
--------------------------------------------------
MetaFresh       | Kietas         | 0.59           |
Karvute         | Karamele       | 0.45           |
Malonumas       | Sokoladas      | 1.2            |
Citrinukas      | Guminukas      | 0.35           |
Medutis         | Karamele       | 0.5            |
Riesutukas      | Sokoladas      | 1.45           |
Vaisinukas      | Guminukas      | 1.45           |
Pupele          | Kietas         | 0.65           |
Braskyte        | Guminukas      | 1.45           |
Perlas          | Sokoladas      | 1.1            |
--------------------------------------------------
Austeja Petraitiene turi 10 skirtingu saldainiu
--------------------------------------------------
Pavadinimas     |       Tipas    |      Kaina     |
--------------------------------------------------
Citrina         | Guminukas      | 0.72           |
Dziaugsmas      | Karamele       | 0.55           |
Sapnas          | Sokoladas      | 1.35           |
Ledukas         | Kietas         | 0.48           |
Lasas           | Karamele       | 0.63           |
Debeselis       | Sokoladas      | 1.25           |
Sprogimas       | Guminukas      | 0.41           |
```

```
Kava            | Kietas         | 0.69          |
Obuolys         | Guminukas      | 0.39          |
Skonis          | Sokoladas      | 1.18          |
-------------------------------------------------
Iveskite po kiek kilogramu turi kiekvienas Tomo saldainis
2
kai kiekvienas pavadinimas yra po 2kg Tomas Kazlauskas bendra saldainiu kaina 18.38 eur

Iveskite po kiek kilogramu turi kiekvienas Austejos saldainis
2
kai kiekvienas pavadinimas yra po 2kg Austeja Petraitiene bendra saldainiu kaina 15.3 eur

Daugiau kainuoja Tomas Kazlauskas saldainiu rinkinys
-------------------------------------------------
Didziausios kainos Tomas Kazlauskas saldainu tipai
-------------------------------------------------
                     Sokoladas
                     Guminukas
-------------------------------------------------
-------------------------------------------------
Didziausios kainos Austeja Petraitiene saldainu tipai
-------------------------------------------------
                     Sokoladas
-------------------------------------------------
Iveskite kaina uz kuria bus filtruojami didesnes kainos saldainiai
0.8
-------------------------------------------------
Pavadinimas     |       Tipas    |      Kaina     |
-------------------------------------------------
Malonumas       | Sokoladas      | 1.2            |
Riesutukas      | Sokoladas      | 1.45           |
Vaisinukas      | Guminukas      | 1.45           |
Braskyte        | Guminukas      | 1.45           |
Perlas          | Sokoladas      | 1.1            |
Sapnas          | Sokoladas      | 1.35           |
Debeselis       | Sokoladas      | 1.25           |
Skonis          | Sokoladas      | 1.18           |
-------------------------------------------------
```

Rezultatai faile „Results.txt“

Tomas Kazlauskas saldainių duomenys

Pavadinimas	Tipas	Kaina
MėtaFresh	Kietas	0.59
Karvutė	Karamelė	0.45
Malonumas	Šokoladas	1.2
Citrinukas	Guminukas	0.35
Medutis	Karamelė	0.5
Riešutukas	Šokoladas	1.45
Vaisinukas	Guminukas	1.45
Pupelė	Kietas	0.65
Braškytė	Guminukas	1.45
Perlas	Šokoladas	1.1

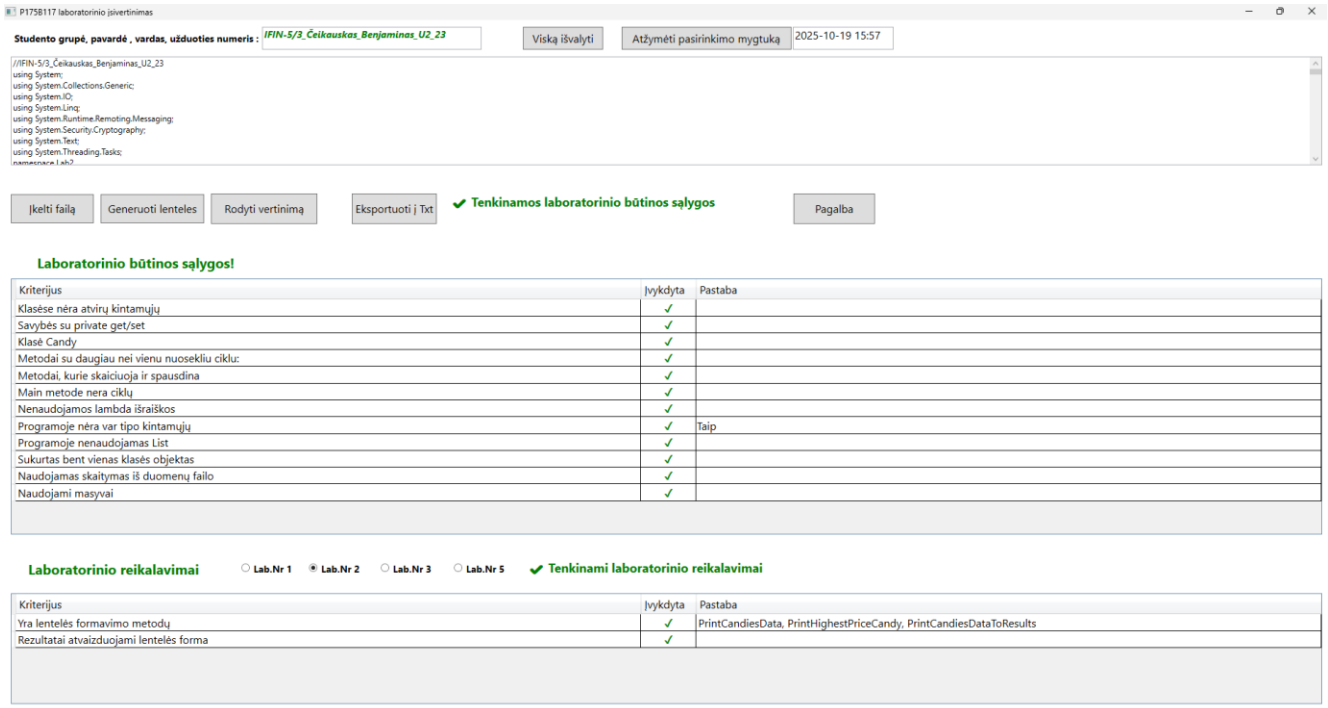
Austėja Petraitienė saldainių duomenys

Pavadinimas	Tipas	Kaina
Citrina	Guminukas	0.72
Džiaugsmas	Karamelė	0.55
Sapnas	Šokoladas	1.35
Ledukas	Kietas	0.48
Lašas	Karamelė	0.63
Debesėlis	Šokoladas	1.25
Sprogimas	Guminukas	0.41
Kava	Kietas	0.69
Obuolys	Guminukas	0.39
Skonis	Šokoladas	1.18

Filtruotų saldainių duomenys

Pavadinimas	Tipas	Kaina
Malonumas	Šokoladas	1.2
Riešutukas	Šokoladas	1.45
Vaisinukas	Guminukas	1.45
Braškytė	Guminukas	1.45
Perlas	Šokoladas	1.1
Sapnas	Šokoladas	1.35
Debesėlis	Šokoladas	1.25
Skonis	Šokoladas	1.18

2.4. Dėstytojo pastabos



3. Konteinerinė klasė

3.1. Darbo užduotis

U3-23. Krepšininkai Tekstiniame faile pateikta informacija apie krepšinio komandos žaidėjus. Eilutėje yra tokia informacija apie krepšininkus: komanda, pavardė, vardas, ūgis, gimimo metai, žaidimo pozicija (puolėjas, gynėjas, centras), žaista rungtynių, įmesta taškų. Sudarykite kiekvienos pozicijos geriausiųjų žaidėjų sąrašus (taškai/rungtynių skaičius). Sąrašus surikiuokite pagal kriterijų (taškai/rungtynių skaičius) ir pavardes alfabetiškai. Iš sąrašų pašalinkite žaidėjus, kurie žaidė ne daugiau kaip vienas rungtynes.

3.2. Programos tekstas

```
//IFIN-5/3_Čeikauskas_Benjaminas_U3_23
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IFIN53_Čeikauskas_Benjaminas_U3_23
{
    /// <summary>
    /// Klasė žaidėjo duomenims saugoti
    /// </summary>
    public class Player
    {
        private string team;    //Komandos pavadinimas
        private string surname; //Pavardė
        private string name;    //Vardas
        private int height;     //Ūgis centimetrais
        private int year;       //Gimimo metai
        private string role;    //Žaidimo pozicija
        private int games;      //Žaista rungtynių
        private int score;      //Įmesta taškų
        private double avarageScore; //Taškų vidurkis
        /// <summary>
        /// Tuščias konstruktorius
        /// </summary>
        public Player()
        {

        }
        /// <summary>
        /// Konstruktorius su parametrais
        /// </summary>
        /// <param name="team"></param>
        /// <param name="surname"></param>
        /// <param name="name"></param>
        /// <param name="height"></param>
        /// <param name="year"></param>
        /// <param name="role"></param>
        /// <param name="games"></param>
        /// <param name="score"></param>
        public Player(string team, string surname, string name, int height, int year,
                      string role, int games, int score)
        {
            this.team = team;
            this.surname = surname;
            this.name = name;
            this.height = height;
            this.year = year;
            this.role = role;
            this.games = games;
```

```
        this.score = score;
        avarageScore = 0;
        if (games > 0) avarageScore = (double)score / games;

    }
    /// <summary>
    /// Sąsajos metodai
    /// </summary>
    public string GetTeam() {return team;}
    public string GetSurname() {return surname;}
    public string GetName() {return name;}
    public int GetHeight() { return  height;}
    public int GetYear() { return year;}
    public string GetRole() {return role;}
    public int GetGames() {return games;}
    public int GetScore() {return score;}
    public double GetAverageScore()
    {
        if (games == 0)
            return 0.0;
        return (double)score / games;
    }
    //Užklotas ToString operatorius
    public override string ToString()
    {
        string line;
        line = string.Format("{0, -18}| {1, -14}| {2, -10}| {3, -5}| {4, -11}| {5, -9}" +
                             "| {6, -9}| {7, -13} | {8, -8:f2} |",
                             team, surname, name, height, year,
                             role, games, score, avarageScore);
        return line;
    }
    //Užklotas Equals operatorius
    public override bool Equals(object obj)
    {
        return obj is Player player &&
               team == player.team &&
               surname == player.surname &&
               name == player.name &&
               height == player.height &&
               year == player.year &&
               role == player.role &&
               games == player.games &&
               score == player.score;
    }
    //Užklotas GetHashCode operatorius
    public override int GetHashCode()
    {
        int hashCode = 133963872;
        hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(team);
        hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(surname);
        hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(name);
        hashCode = hashCode * -1521134295 + height.GetHashCode();
        hashCode = hashCode * -1521134295 + year.GetHashCode();
        hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(role);
        hashCode = hashCode * -1521134295 + games.GetHashCode();
        hashCode = hashCode * -1521134295 + score.GetHashCode();
        return hashCode;
    }
    //Užklotas >= operatorius
    public static bool operator >=(Player player1, Player Player2)
    {
        int score = 0;
        if (player1.avarageScore < Player2.avarageScore)
```

```
            score = -1;
        else if (player1.avarageScore > Player2.avarageScore)
            score = 1;
        else
            score = 0;

        int poz = String.Compare(player1.surname, Player2.surname,
                                 StringComparison.CurrentCulture);
        return (score > 0 || (score == 0 && poz < 0));
    }
    //Užklotas <= operatorius

    public static bool operator <=(Player player1, Player Player2)
    {
        int score = 0;
        if (player1.avarageScore > Player2.avarageScore)
            score = -1;
        else if (player1.avarageScore < Player2.avarageScore)
            score = 1;
        else
            score = 0;

        int poz = String.Compare(player1.surname, Player2.surname,
                                 StringComparison.CurrentCulture);
        return (score < 0 || (score == 0 && poz > 0));
    }
}
/// <summary>
/// Žaidėjų konteinerinė klasė
/// </summary>
public class PlayerArray
{
    const int cMax = 100; //Žaidėjų masyvo dydis
    private Player[] Players;  //Žaidėjų objektų masyvas
    private int count;    //Žaidėjų skaičius
    public PlayerArray()
    {
        count = 0;
        Players = new Player[cMax];
    }
    //Sąsajos metodai
    public Player GetPlayer(int i) { return Players[i]; }
    public int GetCount() { return count; }
    /// <summary>
    /// Žaidėjo pridėjimo metodas
    /// </summary>
    /// <param name="obj"></param>
    public void AddPlayer(Player obj) { Players[count++] = obj; }
    /// <summary>
    /// Rikiavimo metodas
    /// </summary>
    public void Sort()
    {
        int maxInd;
        for (int i = 0; i < count - 1; i++)
        {
            maxInd = i;
            for (int j = i + 1; j < count; j++)
            {
                if (Players[j] >= Players[maxInd])
                    maxInd = j;
            }
            Player temp = Players[i];
            Players[i] = Players[maxInd];
            Players[maxInd] = temp;
        }
    }
    /// <summary>
```

```
        /// Metodas išimantis žaidėjus su ne daugiau nei vienomis rungtynėmis
        /// </summary>
        public void RemovePlayers()
        {
            int m = 0;
            for (int i = 0; i < count; i++)
            {
                if (Players[i].GetGames() > 1)
                    Players[m++] = Players[i];
            }
            count = m;
        }
    }
//-----------------------------------------------------------------------------------------
    internal class Program
    {
        const int Cn = 100;
        const string CFd = "Data1.txt";
        const string CFr = "Results.txt";
        static void Main(string[] args)
        {
            if (File.Exists(CFr))
                File.Delete(CFr);

            using (var fr = File.AppendText(CFr))
            {



                string errorLine =
                    "\t\t\t\t\t" +
                    "*************************************************************\n" +
                    "\t\t\t\t\t" +
                    "*                          TOKIŲ ŽAIDĖJŲ NĖRA                 *\n" +
                    "\t\t\t\t\t" +
                    "*************************************************************\n";

                PlayerArray Players = new PlayerArray();
                string PlayersHeader =
                    "\t\t\t\t\t" +
                    "*************************************************************\n" +
                    "\t\t\t\t\t" +
                    "*                              ŽAIDĖJAI                     *\n" +
                    "\t\t\t\t\t" +
                    "*************************************************************";

                Read(CFd, Players);
                if (Players.GetCount() > 0)
                    Print(fr, Players, PlayersHeader);
                else
                {
                    fr.WriteLine(errorLine);
                }

                PlayerArray PlayersP = new PlayerArray();
                string PlayersPHeader =
                    "\t\t\t\t\t" +
                    "*************************************************************\n" +
                    "\t\t\t\t\t" +
                    "*                          GERIAUSI PUOLĖJAI                *\n" +
                    "\t\t\t\t\t" +
                    "*************************************************************";

                Filter(Players, "Puolėjas", out PlayersP);
                PlayersP.Sort();
                PlayersP.RemovePlayers();
                if (PlayersP.GetCount() > 0)
                    Print(fr, PlayersP, PlayersPHeader);
```

```
                else
                {
                    fr.WriteLine(errorLine);
                }

                PlayerArray PlayersC = new PlayerArray();
                string PlayersCHeader =
                    "\t\t\t\t\t" +
                    "**********************************************************************\n" +
                    "\t\t\t\t\t" +
                    "*                          GERIAUSI CENTRAI                          *\n" +
                    "\t\t\t\t\t" +
                    "**********************************************************************";

                Filter(Players, "Centras", out PlayersC);
                PlayersC.Sort();
                PlayersC.RemovePlayers();
                if (PlayersC.GetCount() > 0)
                    Print(fr, PlayersC, PlayersCHeader);
                else
                {
                    fr.WriteLine(errorLine);
                }

                PlayerArray PlayersG = new PlayerArray();
                string PlayersGHeader =
                    "\t\t\t\t\t" +
                    "**********************************************************************\n" +
                    "\t\t\t\t\t" +
                    "*                          GERIAUSI GYNĖJAI                          *\n" +
                    "\t\t\t\t\t" +
                    "**********************************************************************";

                Filter(Players, "Gynėjas", out PlayersG);
                PlayersG.Sort();
                PlayersG.RemovePlayers();
                if (PlayersG.GetCount() > 0)
                    Print(fr, PlayersG, PlayersGHeader);
                else
                {
                    fr.WriteLine(errorLine);
                }
            }

            Console.WriteLine("PROGRAMA BAIGĖ DARBĄ");
        }
//-----------------------------------------------------------------------------------------
/// <summary>
/// Metodas nuskaitanti duomenis iš duomenų failo ir priskirianti žaidėjų konteineriui
/// </summary>
/// <param name="fn">failas iš kurio skaitomi duomenys</param>
/// <param name="Players">žaidėjų konteineris</param>
        static void Read(string fn, PlayerArray Players)
        {
            using (StreamReader reader = new StreamReader(fn))
            {
                string line;
                while((line = reader.ReadLine()) != null)
                {
                    string[] parts = line.Split(';');
                    string team = parts[0];
                    string surname = parts[1];
                    string name = parts[2];
                    int height = int.Parse(parts[3]);
                    int year = int.Parse((parts[4]));
                    string role = parts[5];
                    int games = int.Parse(parts[6]);
                    int score = int.Parse(parts[7]);
```

```
                    Player player = new Player(team, surname, name, height, year,
                                                role, games, score);
                    Players.AddPlayer(player);
                }
            }
        }
        /// <summary>
        /// Metodas spausdinantis duomenis
        /// </summary>
        /// <param name="fn">Failas į kurį spausdinama</param>
        /// <param name="Players">Žaidėjų konteineris</param>
        /// <param name="header">antraštė</param>
        static void Print(StreamWriter fr, PlayerArray Players, string header)
        {
            const string upperSection =
                "-------------------------------------------------------------------------" +
                "--------------------------------------------------\n" +
                "Nr.\t|\t\tKomanda\t\t|\tPavardė\t\t|\tVardas\t| Ūgis | Gim. Metai | " +
                "Pozicija | Ž. rung. | Pelnyta taškų | Vidurkis |\n" +
                "-------------------------------------------------------------------------" +
                "--------------------------------------------------";

            fr.WriteLine(header);
            fr.WriteLine(upperSection);
            for (int i = 0; i < Players.GetCount(); i++)
            {
                fr.WriteLine("{0,-3}\t| {1, -20}",
                             i + 1, Players.GetPlayer(i).ToString());
            }
            fr.WriteLine(
            "-------------------------------------------------------------------------" +
            "--------------------------------------------------"
                );
            fr.WriteLine();
        }
        /// <summary>
        /// Metodas filtruojantis duomenis
        /// </summary>
        /// <param name="Players">Žaidėjų konteineris iš kurio filtruojame</param>
        /// <param name="position">Pozicija kurią atrenkame</param>
        /// <param name="Filtered">Gražiname filtruotą masyvą su norima pozicija</param>
        static void Filter(PlayerArray Players, string position, out PlayerArray Filtered)
        {
            Filtered = new PlayerArray();

            for (int i = 0; i < Players.GetCount() ; i++)
            {
                Player player = Players.GetPlayer(i);
                if (Players.GetPlayer(i).GetRole().Trim().Equals(position,
                    StringComparison.OrdinalIgnoreCase))
                {
                    Filtered.AddPlayer(player);
                }
            }
        }
    }
}
```

3.3. Pradiniai duomenys ir rezultatai

Testas Nr.1 (Bendrasis atvejis, nėra išimtinių situacijų)

Duomenų failas „Data.txt“

```
Žalgiris; Kazlauskas; Mantas; 198; 1998; Puolėjas; 24; 356
Rytas; Petraitis; Lukas; 205; 2000; Centras; 20; 289
Neptūnas; Jankauskas; Tomas; 192; 1997; Gynėjas; 28; 315
Lietkabelis; Vaitkus; Dovydas; 201; 1999; Puolėjas; 25; 274
Šiauliai; Žemaitis; Karolis; 210; 2001; Centras; 22; 331
Juventus; Paulauskas; Arnas; 190; 1996; Gynėjas; 26; 287
Wolves; Stankevičius; Edvinas; 203; 1998; Puolėjas; 27; 344
CBet; Rimkus; Paulius; 207; 2002; Centras; 18; 222
Nevėžis; Morkūnas; Tadas; 195; 2001; Gynėjas; 23; 266
Pieno_Žvaigždės; Šimkus; Martynas; 200; 1999; Puolėjas; 21; 241
Jonava; Jasiūnas; Rokas; 206; 2003; Centras; 1; 12
Dzūkija; Petrauskas; Laurynas; 202; 2000; Puolėjas; 1; 9
Mažeikiai; Zubrys; Augustas; 188; 1998; Gynėjas; 1; 7
Utena; Giedraitis; Simonas; 209; 1995; Centras; 0; 0
Prienai; Blažys; Arvydas; 196; 1997; Puolėjas; 0; 0
Palanga; Kaminskas; Marius; 191; 2002; Gynėjas; 0; 0
Žalgiris; Sabonis; Jonas; 211; 1999; Centras; 10; 100
Rytas; Kairys; Povilas; 208; 2000; Centras; 10; 100
Neptūnas; Valančius; Arnas; 209; 2001; Centras; 10; 100
Lietkabelis; Balsys; Mantas; 200; 2002; Puolėjas; 15; 150
Šiauliai; Kalvaitis; Rytis; 201; 2002; Puolėjas; 15; 150
Wolves; Barkauskas; Dominykas; 199; 2003; Puolėjas; 15; 150
Juventus; Jonaitis; Deividas; 192; 1999; Gynėjas; 12; 120
Nevėžis; Žvagulis; Marius; 193; 1998; Gynėjas; 12; 120
CBet; Mažeika; Tomas; 194; 2000; Gynėjas; 12; 120
```

Rezultatai faile „Results.txt“

****
* ŽAIDĖJAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Žalgiris	Kazlauskas	Mantas	198	1998	Puolėjas	24	356	14.83
2	Rytas	Petraitis	Lukas	205	2000	Centras	20	289	14.45
3	Neptūnas	Jankauskas	Tomas	192	1997	Gynėjas	28	315	11.25
4	Lietkabelis	Vaitkus	Dovydas	201	1999	Puolėjas	25	274	10.96
5	Šiauliai	Žemaitis	Karolis	210	2001	Centras	22	331	15.05
6	Juventus	Paulauskas	Arnas	190	1996	Gynėjas	26	287	11.04
7	Wolves	Stankevičius	Edvinas	203	1998	Puolėjas	27	344	12.74
8	CBet	Rimkus	Paulius	207	2002	Centras	18	222	12.33
9	Nevėžis	Morkūnas	Tadas	195	2001	Gynėjas	23	266	11.57
10	Pieno_Žvaigždės	Šimkus	Martynas	200	1999	Puolėjas	21	241	11.48
11	Jonava	Jasiūnas	Rokas	206	2003	Centras	1	12	12.00
12	Dzūkija	Petrauskas	Laurynas	202	2000	Puolėjas	1	9	9.00
13	Mažeikiai	Zubrys	Augustas	188	1998	Gynėjas	1	7	7.00
14	Utena	Giedraitis	Simonas	209	1995	Centras	0	0	0.00
15	Prienai	Blažys	Arvydas	196	1997	Puolėjas	0	0	0.00
16	Palanga	Kaminskas	Marius	191	2002	Gynėjas	0	0	0.00
17	Žalgiris	Sabonis	Jonas	211	1999	Centras	10	100	10.00
18	Rytas	Kairys	Povilas	208	2000	Centras	10	100	10.00
19	Neptūnas	Valančius	Arnas	209	2001	Centras	10	100	10.00
20	Lietkabelis	Balsys	Mantas	200	2002	Puolėjas	15	150	10.00
21	Šiauliai	Kalvaitis	Rytis	201	2002	Puolėjas	15	150	10.00
22	Wolves	Barkauskas	Dominykas	199	2003	Puolėjas	15	150	10.00
23	Juventus	Jonaitis	Deividas	192	1999	Gynėjas	12	120	10.00
24	Nevėžis	Žvagulis	Marius	193	1998	Gynėjas	12	120	10.00
25	CBet	Mažeika	Tomas	194	2000	Gynėjas	12	120	10.00

****
* GERIAUSI PUOLĖJAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Žalgiris	Kazlauskas	Mantas	198	1998	Puolėjas	24	356	14.83
2	Wolves	Stankevičius	Edvinas	203	1998	Puolėjas	27	344	12.74
3	Pieno_Žvaigždės	Šimkus	Martynas	200	1999	Puolėjas	21	241	11.48
4	Lietkabelis	Vaitkus	Dovydas	201	1999	Puolėjas	25	274	10.96
5	Lietkabelis	Balsys	Mantas	200	2002	Puolėjas	15	150	10.00
6	Wolves	Barkauskas	Dominykas	199	2003	Puolėjas	15	150	10.00
7	Šiauliai	Kalvaitis	Rytis	201	2002	Puolėjas	15	150	10.00

****
* GERIAUSI CENTRAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Šiauliai	Žemaitis	Karolis	210	2001	Centras	22	331	15.05
2	Rytas	Petraitis	Lukas	205	2000	Centras	20	289	14.45
3	CBet	Rimkus	Paulius	207	2002	Centras	18	222	12.33
4	Rytas	Kairys	Povilas	208	2000	Centras	10	100	10.00
5	Žalgiris	Sabonis	Jonas	211	1999	Centras	10	100	10.00
6	Neptūnas	Valančius	Arnas	209	2001	Centras	10	100	10.00

****
* GERIAUSI GYNĖJAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Nevėžis	Morkūnas	Tadas	195	2001	Gynėjas	23	266	11.57
2	Neptūnas	Jankauskas	Tomas	192	1997	Gynėjas	28	315	11.25
3	Juventus	Paulauskas	Arnas	190	1996	Gynėjas	26	287	11.04
4	Juventus	Jonaitis	Deividas	192	1999	Gynėjas	12	120	10.00
5	CBet	Mažeika	Tomas	194	2000	Gynėjas	12	120	10.00
6	Nevėžis	Žvagulis	Marius	193	1998	Gynėjas	12	120	10.00

Testas Nr.2 (Atvejis, kai nėra centro pozicijos žaidėjų)

Duomenų failas „Data1.txt“

```
Žalgiris; Kazlauskas; Mantas; 198; 1998; Puolėjas; 24; 356
Neptūnas; Jankauskas; Tomas; 192; 1997; Gynėjas; 28; 315
Lietkabelis; Vaitkus; Dovydas; 201; 1999; Puolėjas; 25; 274
Juventus; Paulauskas; Arnas; 190; 1996; Gynėjas; 26; 287
Wolves; Stankevičius; Edvinas; 203; 1998; Puolėjas; 27; 344
Nevėžis; Morkūnas; Tadas; 195; 2001; Gynėjas; 23; 266
Pieno_Žvaigždės; Šimkus; Martynas; 200; 1999; Puolėjas; 21; 241
Dzūkija; Petrauskas; Laurynas; 202; 2000; Puolėjas; 1; 9
Mažeikiai; Zubrys; Augustas; 188; 1998; Gynėjas; 1; 7
Prienai; Blažys; Arvydas; 196; 1997; Puolėjas; 0; 0
Palanga; Kaminskas; Marius; 191; 2002; Gynėjas; 0; 0
Lietkabelis; Balsys; Mantas; 200; 2002; Puolėjas; 15; 150
Šiauliai; Kalvaitis; Rytis; 201; 2002; Puolėjas; 15; 150
Wolves; Barkauskas; Dominykas; 199; 2003; Puolėjas; 15; 150
Juventus; Jonaitis; Deividas; 192; 1999; Gynėjas; 12; 120
Nevėžis; Žvagulis; Marius; 193; 1998; Gynėjas; 12; 120
CBet; Mažeika; Tomas; 194; 2000; Gynėjas; 12; 120
```

Rezultatai faile „Results.txt“

****
* ŽAIDĖJAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Žalgiris	Kazlauskas	Mantas	198	1998	Puolėjas	24	356	14.83
2	Neptūnas	Jankauskas	Tomas	192	1997	Gynėjas	28	315	11.25
3	Lietkabelis	Vaitkus	Dovydas	201	1999	Puolėjas	25	274	10.96
4	Juventus	Paulauskas	Arnas	190	1996	Gynėjas	26	287	11.04
5	Wolves	Stankevičius	Edvinas	203	1998	Puolėjas	27	344	12.74
6	Nevėžis	Morkūnas	Tadas	195	2001	Gynėjas	23	266	11.57
7	Pieno_Žvaigždės	Šimkus	Martynas	200	1999	Puolėjas	21	241	11.48
8	Dzūkija	Petrauskas	Laurynas	202	2000	Puolėjas	1	9	9.00
9	Mažeikiai	Zubrys	Augustas	188	1998	Gynėjas	1	7	7.00
10	Prienai	Blažys	Arvydas	196	1997	Puolėjas	0	0	0.00
11	Palanga	Kaminskas	Marius	191	2002	Gynėjas	0	0	0.00
12	Lietkabelis	Balsys	Mantas	200	2002	Puolėjas	15	150	10.00
13	Šiauliai	Kalvaitis	Rytis	201	2002	Puolėjas	15	150	10.00
14	Wolves	Barkauskas	Dominykas	199	2003	Puolėjas	15	150	10.00
15	Juventus	Jonaitis	Deividas	192	1999	Gynėjas	12	120	10.00
16	Nevėžis	Žvagulis	Marius	193	1998	Gynėjas	12	120	10.00
17	CBet	Mažeika	Tomas	194	2000	Gynėjas	12	120	10.00

****
* GERIAUSI PUOLĖJAI *
****

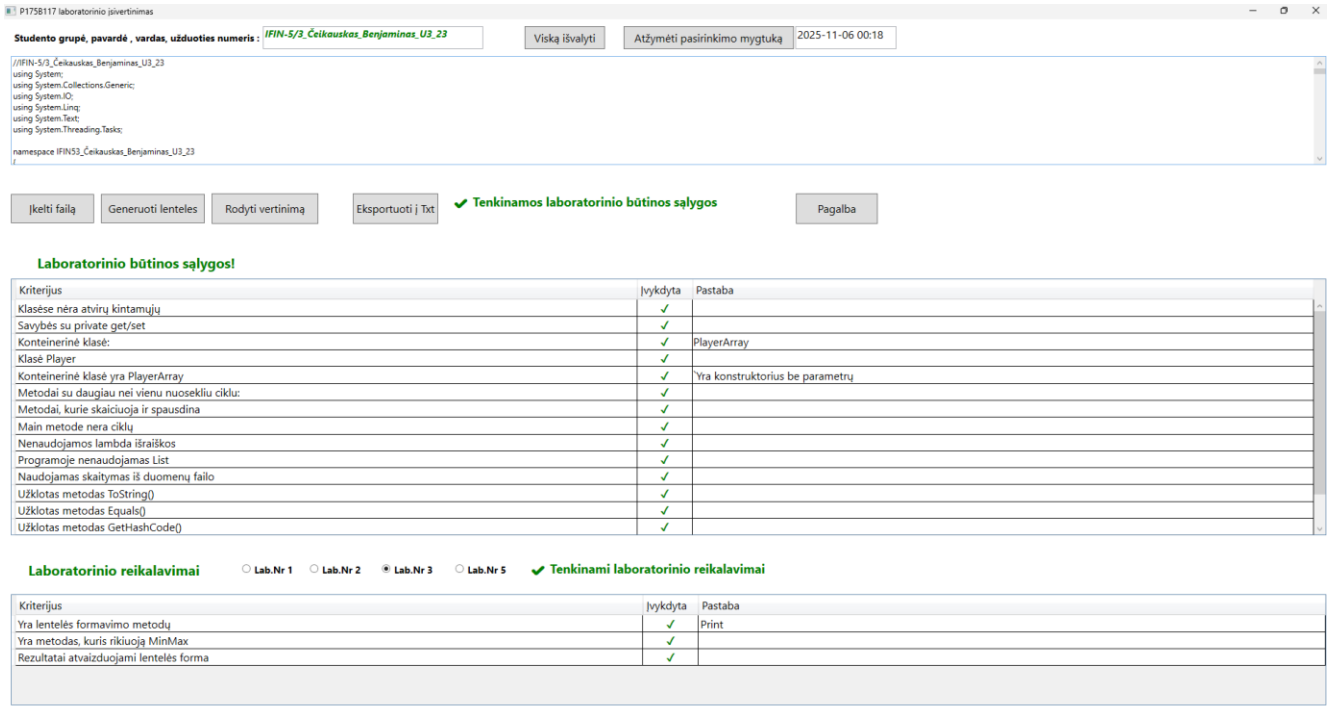
Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Žalgiris	Kazlauskas	Mantas	198	1998	Puolėjas	24	356	14.83
2	Wolves	Stankevičius	Edvinas	203	1998	Puolėjas	27	344	12.74
3	Pieno_Žvaigždės	Šimkus	Martynas	200	1999	Puolėjas	21	241	11.48
4	Lietkabelis	Vaitkus	Dovydas	201	1999	Puolėjas	25	274	10.96
5	Lietkabelis	Balsys	Mantas	200	2002	Puolėjas	15	150	10.00
6	Wolves	Barkauskas	Dominykas	199	2003	Puolėjas	15	150	10.00
7	Šiauliai	Kalvaitis	Rytis	201	2002	Puolėjas	15	150	10.00

****
* TOKIŲ ŽAIDĖJŲ NĖRA *
****

****
* GERIAUSI GYNĖJAI *
****

Nr.	Komanda	Pavardė	Vardas	Ūgis	Gim. Metai	Pozicija	Ž. rung.	Pelnyta taškų	Vidurkis
1	Nevėžis	Morkūnas	Tadas	195	2001	Gynėjas	23	266	11.57
2	Neptūnas	Jankauskas	Tomas	192	1997	Gynėjas	28	315	11.25
3	Juventus	Paulauskas	Arnas	190	1996	Gynėjas	26	287	11.04
4	Juventus	Jonaitis	Deividas	192	1999	Gynėjas	12	120	10.00
5	CBet	Mažeika	Tomas	194	2000	Gynėjas	12	120	10.00
6	Nevėžis	Žvagulis	Marius	193	1998	Gynėjas	12	120	10.00

3.4. Dėstytojo pastabos



4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

U4-23. „Didesnės“ raidės Tekstiniame faile pateikiamas tekstas. Žodžiai iš eilutės į kitą eilutę nekeliami. Žodžiai eilutėse skiriami skyrikliais. Skyrikliai gali būti eilutės pradžioje bei gale, gali būti tuščios eilutės. Skyrikliai žinomi. Kokių žodžių eilutėje daugiau: tų, kurių pirmoji raidė “didesnė” už paskutiniąją ar tų, kurių pirmoji raidė “mažesnė” už paskutiniąją? Kiekvienos eilutės atsakymą įrašykite į naują failą. Duotame faile pašalinkite žodžius, kurių pirmosios dvi raidės sutampa su dviem paskutinėmis raidėmis. Skyrikliai nešalinami.

4.2. Programos tekstas

```
//IFIN-5/3_Čeikauskas_Benjaminas_U4_23
using System;
using System.IO;
using System.Text;

namespace WordComparison
{
    /// <summary>
    /// Class for line info
    /// </summary>
    public class Line
    {
        public string text { get; private set; } // Original line text
        /// <summary>
        /// Empty constructor
        /// </summary>
        public Line()
        {
        }
        /// <summary>
        /// Constructor with parameters
        /// </summary>
        /// <param name="text"></param>
        public Line(string text)
        {
            this.text = text;
        }
        /// <summary>
        /// Counts the number of words where the first character is greater than the last.
        /// </summary>
        /// <param name="separators">An array of characters used as separators </param>
        /// <returns>The count of words where the first character
        /// is greater than the last character.</returns>
        public int GetFirstGreater(char[] separators)
        {
            string[] words = ExtractWords(separators);
            int count = 0;

            for (int i = 0; i < words.Length; i++)
            {
                string w = words[i];
                if (w.Length >= 2 && w[0] > w[w.Length - 1])
                    count++;
            }
            return count;
        }
        /// <summary>
        /// Counts the number of words where the last character is greater than the first.
        /// </summary>
        /// <param name="separators">An array of characters used as separators </param>
        /// <returns>The count of words where the last character
```

```
/// is greater than the first character.</returns>
public int GetLastGreater(char[] separators)
{
    string[] words = ExtractWords(separators);
    int count = 0;

    for (int i = 0; i < words.Length; i++)
    {
        string w = words[i];
        if (w.Length >= 2 && w[w.Length - 1] > w[0])
            count++;
    }
    return count;
}
/// <summary>
/// Retrieves an array of words that have the same first two and last two characters.
/// </summary>
/// <param name="separators">An array of characters used as separators</param>
/// <returns>An array of strings containing words where the first two and last two
/// characters are identical.  The array will be empty if no such words
/// are found.</returns>
public string[] GetSpecialWords(char[] separators)
{
    string[] words = ExtractWords(separators);

    int max = words.Length;
    string[] temp = new string[max];
    int count = 0;

    for (int i = 0; i < words.Length; i++)
    {
        string word = words[i];

        if (word.Length < 4)
            continue;

        string firstTwo = word.Substring(0, 2);
        string lastTwo = word.Substring(word.Length - 2);

        if (firstTwo == lastTwo)
        {
            temp[count++] = word;
        }
    }

    string[] result = new string[count];
    Array.Copy(temp, result, count);
    return result;
}
/// <summary>
/// Removes words from the text that start and end with the same two characters,
/// using specified separators to identify word boundaries.
/// </summary>
/// <param name="separators">An array of characters used as separators</param>
/// <returns>A string with special words removed, preserving the original
/// separators.</returns>
public string RemoveSpecialWords(char[] separators)
{
    string line = text;
    StringBuilder result = new StringBuilder();

    StringBuilder word = new StringBuilder();

    for (int i = 0; i < line.Length; i++)
    {
        char c = line[i];

        bool isSeparator = false;
```

```
            for (int j = 0; j < separators.Length; j++)
            {
                if (c == separators[j])
                {
                    isSeparator = true;
                    break;
                }
            }

            if (isSeparator)
            {
                // if word ended, process it
                if (word.Length > 0)
                {
                    string w = word.ToString();

                    if (w.Length >= 4 &&
                        w.Substring(0, 2) == w.Substring(w.Length - 2))
                    {
                        // SKIP special word – do nothing
                    }
                    else
                    {
                        // not special word – add to result
                        result.Append(w);
                    }

                    word.Clear();
                }

                // add the separator to the result
                result.Append(c);
            }
            else
            {
                word.Append(c);
            }
        }

        // process the last word if any
        if (word.Length > 0)
        {
            string w = word.ToString();

            if (!(w.Length >= 4 &&
                  w.Substring(0, 2) == w.Substring(w.Length - 2)))
            {
                result.Append(w);
            }
        }

        return result.ToString();
    }
    /// <summary>
    /// Extracts words from the current text using the specified separators.
    /// </summary>
    /// <param name="separators">An array of characters used as separators</param>
    /// <returns>An array of strings, each representing a word extracted from the text.
    /// The array will be empty if no words are found.</returns>
    private string[] ExtractWords(char[] separators)
    {
        string line = text;
        int maxWords = line.Length;
        string[] temp = new string[maxWords];
        int count = 0;

        StringBuilder word = new StringBuilder();
```

```
            for (int i = 0; i < line.Length; i++)
            {
                char c = line[i];
                bool isSeparator = false;

                for (int j = 0; j < separators.Length; j++)
                {
                    if (c == separators[j])
                    {
                        isSeparator = true;
                        break;
                    }
                }

                if (isSeparator)
                {
                    if (word.Length > 0)
                    {
                        temp[count++] = word.ToString();
                        word.Clear();
                    }
                }
                else
                {
                    word.Append(c);
                }
            }

            if (word.Length > 0)
                temp[count++] = word.ToString();

            string[] result = new string[count];
            Array.Copy(temp, result, count);

            return result;
        }


}
/// <summary>
/// Container for storing and managing a collection of class Line objects.
/// </summary>
public class LinesContainer
{
    const int Cmax = 100;
    private Line[] Lines;
    private int count;
    /// <summary>
    /// Initializes a new instance of the class.
    /// </summary>
    public LinesContainer()
    {
        count = 0;
        Lines = new Line[Cmax];
    }
    //Methods of the interface
    public Line GetLine(int i) { return Lines[i]; }
    public int GetCount() { return count; }
    public void AddLine(Line obj) { Lines[count++] = obj; }
}
internal class Program
{
    const string CFd = "Data.txt";
    const string CFa = "Analysis.txt";
    const string CFr = "Results.txt";

    static void Main(string[] args)
    {
```

```
    char[] separators =
    {
        ' ', '.', ',', '!', '?', ':', ';', '(', ')', '-', '\t', '\r', '\n'
    };

    LinesContainer container = new LinesContainer();

    ReadLines(container, CFd);

    Print(container, separators, CFa, CFr);

    Console.WriteLine("Finished.");
}
/// <summary>
/// Reads all lines from a specified file and adds them to the provided container.
/// </summary>
/// <param name="container">Container to which the lines will be added.</param>
/// <param name="Fn">The path of the file to read.</param>
static void ReadLines(LinesContainer container, string Fn)
{
    string[] lines = File.ReadAllLines(Fn, Encoding.UTF8);

    for (int i = 0; i < lines.Length; i++)
    {
        Line obj = new Line(lines[i]);
        container.AddLine(obj);
    }
}
/// <summary>
/// Processes each line in the specified container, analyzing and writing results
/// to the given files.
/// </summary>
/// <param name="container">Container containing lines to be processed.</param>
/// <param name="separators">An array of characters used as separators</param>
/// <param name="FnAnalysis">The file path where the analysis results of special
/// words will be written.</param>
/// <param name="FnResults">The file path where the processed line results will
/// be written.</param>
static void Print(LinesContainer container, char[] separators, string FnAnalysis,
                  string FnResults)
{
    using (var analysis = File.CreateText(FnAnalysis))
    {
        using (var result = File.CreateText(FnResults))
        {
            for (int i = 0; i < container.GetCount(); i++)
            {
                Line line = container.GetLine(i);

                int lastGreater = line.GetLastGreater(separators);
                int firstGreater = line.GetFirstGreater(separators);

                string[] specialWords = line.GetSpecialWords(separators);

                string cleanedLine = line.RemoveSpecialWords(separators);
                if (specialWords.Length > 0)
                {
                    analysis.WriteLine(string.Join(" ", specialWords));
                }

                result.WriteLine("----------------------------------------");

                if (cleanedLine.Length == 0)
                {
                result.WriteLine("{0} Line is empty", i + 1);
                    continue;
                }
                result.WriteLine("Line number: " + (i + 1));
```

```
                        result.WriteLine("Cleaned text: " + cleanedLine);

                        if(firstGreater > lastGreater)
                        {
                            result.WriteLine("Text have more words with" +
                                             " higher first letter count: "
                                             + firstGreater);

                        }
                        else if (firstGreater < lastGreater)
                        {
                            result.WriteLine("Text have more words with" +
                                             " higher last letter count: "
                                             + lastGreater);

                        }
                        else if (firstGreater == 0 && lastGreater == 0)
                        {
                            result.WriteLine("There are no words with greater" +
                                             " first or greater last letter");
                        }
                        else
                            result.WriteLine("Words with first higher letter and last" +
                                " higher letter have equal count: " + lastGreater);
                    }
                }
            }
        }
    }
}
```

4.3. Pradiniai duomenys ir rezultatai

Duomenų failas „Data.txt“

```
level test abba code deed more
hello world aaaabaaaa test
abc cba noon xyz

,level... test,,aabbaa?? code!!!  deed;more

11f2211 112221
(noon)  test::word--civic??

level ccabbacc deed rotor kayak stats solos

aa bb ccc deed ggottogg abcba test wowow

test-word--test;racecar...zaz?pop??

word word word word

xxyyxx xyzz ababa ababa

mirror rotor kayak test-word
code;test;data
```

Rezultatų failas „Data.txt“

```
-----------------------------------------
Line number: 1
Cleaned text: level test abba code deed more
Words with first higher letter and last higher letter have equal count: 1
-----------------------------------------
Line number: 2
Cleaned text: hello world  test
```

```
Words with first higher letter and last higher letter have equal count: 1
-------------------------------------------
Line number: 3
Cleaned text: abc cba noon xyz
Text have more words with higher last letter count: 2
-------------------------------------------
4 Line is empty
-------------------------------------------
Line number: 5
Cleaned text: ,level... test,,?? code!!!  deed;more
Words with first higher letter and last higher letter have equal count: 1
-------------------------------------------
6 Line is empty
-------------------------------------------
Line number: 7
Cleaned text:  112221
There are no words with greater first or greater last letter
-------------------------------------------
Line number: 8
Cleaned text: (noon)  test::word--civic??
Text have more words with higher first letter count: 1
-------------------------------------------
9 Line is empty
-------------------------------------------
Line number: 10
Cleaned text: level  deed rotor kayak stats solos
There are no words with greater first or greater last letter
-------------------------------------------
11 Line is empty
-------------------------------------------
Line number: 12
Cleaned text: aa bb ccc deed  abcba test wowow
There are no words with greater first or greater last letter
-------------------------------------------
13 Line is empty
-------------------------------------------
Line number: 14
Cleaned text: test-word--test;racecar...zaz?pop??
Text have more words with higher first letter count: 1
-------------------------------------------
15 Line is empty
-------------------------------------------
Line number: 16
Cleaned text: word word word word
Text have more words with higher first letter count: 4
-------------------------------------------
17 Line is empty
-------------------------------------------
Line number: 18
Cleaned text:  xyzz ababa ababa
Text have more words with higher last letter count: 1
-------------------------------------------
19 Line is empty
-------------------------------------------
Line number: 20
Cleaned text: mirror rotor kayak test-word
Words with first higher letter and last higher letter have equal count: 1
-------------------------------------------
Line number: 21
Cleaned text: code;test;data
Words with first higher letter and last higher letter have equal count: 1
```

Analizės failas „Data.txt“

```
aaaabaaaa
aabbaa
11f2211
ccabbacc
ggottogg
xxyyxx
```

4.4. Dėstytojo pastabos

5. Susieti rinkiniai

5.1. Darbo užduotis

U5–23. Darbo birža Pirmo failo pirmoje failo eilutėje nurodytas miestų skaičius ir mėnesių skaičius. Tolesnėse failo eilutėse nurodyta informacija apie miestus: miesto pavadinimas, gyventojų skaičius, jaunimo nuo 19 iki 25 metų skaičius. Kitame faile pateikta informacija apie jaunimo nuo 19 iki 25 metų nedarbą miestuose: miestai (eilutės), kiek bedarbių registruota kiekvieną mėnesį (stulpeliai). Nustatykite, kurį mėnesį buvo didžiausias nedarbas jaunimo tarpe. Suraskite, kurį mėnesį ir kuriame mieste santykinis nedarbo lygis buvo mažiausias. Surikiuokite miestus pagal jaunimo skaičių ir gyventojų skaičių.

5.2. Programos tekstas

```
//IFIN53_Čeikauskas_Benjaminas_U5_23

//U5-23. Darbo birža
//Pirmo failo pirmoje failo eilutėje nurodytas miestų skaičius ir mėnesių skaičius.
//Tolesnėse failo
//eilutėse nurodyta informacija apie miestus: miesto pavadinimas, gyventojų skaičius,
//jaunimo nuo 19 iki 25 metų skaičius.
//
//Kitame faile pateikta informacija apie jaunimo nuo 19 iki 25 metų nedarbą
//miestuose: miestai(eilutės), kiek bedarbių registruota kiekvieną mėnesį (stulpeliai).
//
//Nustatykite, kurį mėnesį buvo didžiausias nedarbas jaunimo tarpe.
//Suraskite, kurį mėnesį ir kuriame mieste santykinis nedarbo lygis buvo mažiausias.
//Surikiuokite miestus pagal jaunimo skaičių ir gyventojų skaičių.

using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;

namespace IFIN53_Čeikauskas_Benjaminas_U5_23
{
    /// <summary>
    /// Class holding city data
    /// </summary>
    class City
    {
        private string name;
        private int residents;
        private int youth;
        /// <summary>
        /// Empty constructor
        /// </summary>
        public City() { }
        /// <summary>
        /// Constructor with parameters
        /// </summary>
        /// <param name="name"></param>
        /// <param name="residents"></param>
        /// <param name="youth"></param>
        public City(string name, int residents, int youth)
        {
            this.name = name;
            this.residents = residents;
            this.youth = youth;
        }
        /// <summary>
```

```
        /// Adds city data
        /// </summary>
        /// <param name="name"></param>
        /// <param name="residents"></param>
        /// <param name="youth"></param>
        public void Add(string name, int residents, int youth)
        {
            this.name = name;
            this.residents = residents;
            this.youth = youth;
        }
        //interface methods
        public string GetName() { return name; }
        public int GetResidents() { return residents; }
        public int GetYouth() { return youth; }
        public override string ToString()
        {
            string line;
            line = string.Format("{0, -12}|     {1, -10}|    {2,2:d}    |",
            name, residents, youth);

            return line;
        }
        //operator overloads for sorting by youth and residents
        public static bool operator <=(City a, City b)
        {
            return a.youth > b.youth || a.youth == b.youth && a.residents > b.residents;
        }
        public static bool operator >=(City a, City b)
        {
            return a.youth > b.youth || a.youth == b.youth && a.residents > b.residents;
        }
    }
    /// <summary>
    /// Class holding array of cities
    /// </summary>
    class CityArray
    {
        const int CMax = 100;
        public int n { get; set; }
        private City[] Cities;
        public CityArray()
        {
            n = 0;
            Cities = new City[CMax];
        }
        //interface methods
        //gets city from array
        public City GetCity(int i)
        {
            City c1 = new City(Cities[i].GetName(), Cities[i].GetResidents(),
                               Cities[i].GetYouth());
            return c1;
        }//adds city to array
        public void Add(City c)
        {
            City c1 = new City(c.GetName(), c.GetResidents(), c.GetYouth());
            Cities[n++] = c1;
        }
        //changes city in array
        public void Change(int i, City c)
        {
            City c1 = new City(c.GetName(), c.GetResidents(), c.GetYouth());
            Cities[i] = c1;
        }
    }
    /// <summary>
    /// Matrix class holding unemployed youth data
```

```
/// </summary>
class Matrix
{
    const int CMaxRow = 100;
    const int CMaxCol = 12;
    //private City[,] C;
    public int n { get; set; }
    public int m { get; set; }
    private int[,] S; // jaunimo nuo 19 iki 25 metų nedarbas
    public Matrix()
    {
        n = 0;
        m = 0;
        //C = new City[CMaxEil, CMaxSt];
        S = new int[CMaxRow, CMaxCol];
    }
    //interface methods
    //adds unemployed youth data to matrix
    public void AddS(int row, int cols, int num)
    {
        S[row, cols] = num;
    }
    //gets unemployed youth data from matrix
    public int GetS(int row, int cols)
    {
        return S[row, cols];
    }
    //changes rows in matrix
    public void ChangeSRow(int row1, int row2)
    {
        for (int j = 0; j < m; j++)
        {
            int temp = S[row1, j];
            S[row1, j] = S[row2, j];
            S[row2, j] = temp;
        }
    }
}
internal class Program
{
    //TEST1
    //MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT - regular case (1 highest)
    //CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO - regular case (1 lowest)
    const string CFd1 = "Data1_test1.txt";
    const string CFd2 = "Data2_test1.txt";

    //TEST2
    //MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT – 3 highest
    //CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO – 3 lowest
    //const string CFd1 = "Data1_test2.txt";
    //const string CFd2 = "Data2_test2.txt";

    //TEST3
    //MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT - regular case
    //CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO -
    //no unemployed youth and no youth population cases
    //const string CFd1 = "Data1_test3.txt";
    //const string CFd2 = "Data2_test3.txt";

    const string CFr = "Results.txt";
    static void Main(string[] args)
    {
        if (File.Exists(CFr))
        {
            File.Delete(CFr);
        }

        int Snn, Smm; //Matrix row and col count
```

```
        CityArray State = new CityArray();
        ReadCities(CFd1, ref State, out Snn, out Smm);
        Print(CFr, State, "Cities");

        Matrix stateUn = new Matrix();
        ReadMatrix(CFd2, ref stateUn, Snn, Smm);
        PrintMatrix(CFr, stateUn, "Unemployed youth in cities");

        int maxSum = MaxMonthSum(stateUn);

        int[] maxMonths = new int[stateUn.m];
        int count;

        FindMaxMonths(stateUn, maxSum, maxMonths, out count);

        PrintMaxMonths(CFr, maxSum, maxMonths, count);


        double minRatio = MinYouthUnemploymentRatio(stateUn, State);

        int maxCases = stateUn.n * stateUn.m;
        int[] cityIdx = new int[maxCases];
        int[] monthIdx = new int[maxCases];
        int rCount;

        FindMinRatioCases(stateUn, State, minRatio, cityIdx, monthIdx, out rCount);

        PrintMinRatioCases(CFr, State, stateUn, cityIdx, monthIdx, rCount, minRatio);

        SortCities(ref State, ref stateUn);
        Print(CFr, State, "Sorted cities by youth and residents from lower to higher");
        PrintMatrix(CFr, stateUn, "Unemployed youth in cities after sorting");

        Console.WriteLine("Program finished working.");
        Console.WriteLine("All results are written in {0} file.", CFr);
    }
    /// <summary>
    /// Reads city data from a file and fills CityArray with the data.
    /// </summary>
    /// <param name="fd"></param>
    /// <param name="cities"></param>
    /// <param name="nn"></param>
    /// <param name="mm"></param>
    static void ReadCities(string fd, ref CityArray cities, out int nn, out int mm)
    {
        string name;
        int residents, youth;
        string line;
        nn = 0;
        mm = 0;
        using (StreamReader reader = new StreamReader(fd))
        {
            line = reader.ReadLine();
            string[] parts = line.Split(' ');
            nn = int.Parse(parts[0]);
            mm = int.Parse(parts[1]);
            for (int i = 0; i < nn; i++)
            {
                line = reader.ReadLine();
                parts = line.Split(' ');
                name = parts[0];
                residents = int.Parse(parts[1]);
                youth = int.Parse(parts[2]);
                City c;
                c = new City(name, residents, youth);
                cities.Add(c);
            }
```

```
    }
}
/// <summary>
/// Reads unemployed youth data from a file and fills Matrix with the data.
/// </summary>
/// <param name="fd"></param>
/// <param name="matrix"></param>
/// <param name="nn"></param>
/// <param name="mm"></param>
static void ReadMatrix(string fd, ref Matrix matrix, int nn, int mm)
{
    string line;
    matrix.n = nn;
    matrix.m = mm;
    int unYouth;
    using (StreamReader reader = new StreamReader(fd))
    {
        string[] parts;
        for (int i = 0; i < matrix.n; i++)
        {
            line = reader.ReadLine();
            parts = line.Split(' ');
            for (int j = 0; j < matrix.m; j++)
            {
                unYouth = int.Parse(parts[j]);
                matrix.AddS(i, j, unYouth);
            }
        }
    }
}
/// <summary>
/// Prints city data to a file
/// </summary>
/// <param name="fn"></param>
/// <param name="cities"></param>
/// <param name="header"></param>
static void Print(string fn, CityArray cities, string header)
{
    using (var writer = File.AppendText(fn))
    {
        string dash = new string('-', 50);
        writer.WriteLine(header);
        writer.WriteLine(dash);
        writer.WriteLine("| Nr. |    City    |   Residents   |    Youth    |");
        writer.WriteLine(dash);
        for (int i = 0; i < cities.n; i++)
        {
            writer.WriteLine("| {0}.  |{1}", i + 1, cities.GetCity(i).ToString());
        }
        writer.WriteLine(dash);
        writer.WriteLine();
    }
}
/// <summary>
/// Prints unemployed youth matrix to a file
/// </summary>
/// <param name="fn"></param>
/// <param name="unemployed"></param>
/// <param name="comm"></param>
static void PrintMatrix(string fn, Matrix unemployed, string comm)
{
    using (var fr = File.AppendText(fn))
    {
        string dash = new string('-', 50);
        fr.WriteLine(dash);
        fr.WriteLine("{0} per {1} month.", comm, unemployed.m);
        fr.WriteLine(dash);
        for (int i = 0; i < unemployed.n; i++)
```

```
        {
            fr.Write("{0,5:d}. ", i + 1);
            for (int j = 0; j < unemployed.m; j++)
                fr.Write("{0,5:d} ", unemployed.GetS(i, j));
            fr.WriteLine();
        }
        fr.WriteLine(dash);
        fr.WriteLine();

    }
}
/// <summary>
/// Returns the maximum sum of unemployed youth in any month
/// </summary>
/// <param name="matrix"></param>
/// <returns></returns>
static int MaxMonthSum(Matrix matrix)
{
    int max = 0;
    for (int j = 0; j < matrix.m; j++)
    {
        int sum = 0;
        for (int i = 0; i < matrix.n; i++)
            sum += matrix.GetS(i, j);

        if (sum > max)
            max = sum;
    }
    return max;
}
/// <summary>
/// Finds months with the maximum sum of unemployed youth
/// </summary>
/// <param name="matrix"></param>
/// <param name="max"></param>
/// <param name="months"></param>
/// <param name="count"></param>
static void FindMaxMonths(Matrix matrix, int max,
                  int[] months, out int count)
{
    count = 0;

    for (int j = 0; j < matrix.m; j++)
    {
        int sum = 0;
        for (int i = 0; i < matrix.n; i++)
            sum += matrix.GetS(i, j);

        if (sum == max)
            months[count++] = j;
    }
}

/// <summary>
/// Prints months with the maximum sum of unemployed youth to a file
/// </summary>
/// <param name="fn"></param>
/// <param name="max"></param>
/// <param name="months"></param>
/// <param name="count"></param>
static void PrintMaxMonths(string fn, int max,
                           int[] months, int count)
{
    using (var writer = File.AppendText(fn))
    {
        writer.WriteLine("MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT");
        writer.WriteLine("-----------------------------------");
```

```
        for (int i = 0; i < count; i++)
        {
            writer.WriteLine(
                "Month {0}: total unemployed youth = {1}",
                months[i] + 1, max
            );
        }
        writer.WriteLine();
    }
}

/// <summary>
/// Returns the minimum youth unemployment ratio
/// </summary>
/// <param name="matrix"></param>
/// <param name="cities"></param>
/// <returns></returns>
static double MinYouthUnemploymentRatio(Matrix matrix, CityArray cities)
{
    double min = -1;

    for (int i = 0; i < matrix.n; i++)
    {
        int youth = cities.GetCity(i).GetYouth();
        if (youth == 0) continue;

        for (int j = 0; j < matrix.m; j++)
        {
            int unemployed = matrix.GetS(i, j);
            if (unemployed == 0) continue;

            double ratio = (double)unemployed / youth;

            if (min < 0 || ratio < min)
                min = ratio;
        }
    }
    return min;
}
/// <summary>
/// Finds cases with the minimum youth unemployment ratio
/// </summary>
/// <param name="matrix"></param>
/// <param name="cities"></param>
/// <param name="minRatio"></param>
/// <param name="cityIdx"></param>
/// <param name="monthIdx"></param>
/// <param name="count"></param>
static void FindMinRatioCases(Matrix matrix, CityArray cities, double minRatio,
                              int[] cityIdx, int[] monthIdx, out int count)
{
    count = 0;

    for (int i = 0; i < matrix.n; i++)
    {
        int youth = cities.GetCity(i).GetYouth();
        if (youth == 0) continue;

        for (int j = 0; j < matrix.m; j++)
        {
            int unemployed = matrix.GetS(i, j);
            if (unemployed == 0) continue;

            double ratio = (double)unemployed / youth;

            if (ratio == minRatio)
            {
                cityIdx[count] = i;
```

```
                monthIdx[count] = j;
                count++;
            }
        }

    }
}
/// <summary>
/// Prints cases with the minimum youth unemployment ratio to a file
/// </summary>
/// <param name="fn"></param>
/// <param name="cities"></param>
/// <param name="matrix"></param>
/// <param name="cityIdx"></param>
/// <param name="monthIdx"></param>
/// <param name="count"></param>
/// <param name="minRatio"></param>
static void PrintMinRatioCases(string fn, CityArray cities, Matrix matrix,
                           int[] cityIdx, int[] monthIdx, int count, double minRatio)
{
    using (var writer = File.AppendText(fn))
    {
        writer.WriteLine("CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO");
        writer.WriteLine("----------------------------------------");

        bool hasSpecialCases = false;

        // Tikrinam, ar yra unemployed == 0 arba youth == 0
        for (int i = 0; i < matrix.n; i++)
        {
            int youth = cities.GetCity(i).GetYouth();

            for (int j = 0; j < matrix.m; j++)
            {
                int unemployed = matrix.GetS(i, j);

                if (youth == 0 || unemployed == 0)
                {
                    hasSpecialCases = true;
                    break;
                }
            }
            if (hasSpecialCases) break;
        }
        // If special cases exist – print them
        if (hasSpecialCases)
        {
            for (int i = 0; i < matrix.n; i++)
            {
                int youth = cities.GetCity(i).GetYouth();

                for (int j = 0; j < matrix.m; j++)
                {
                    int unemployed = matrix.GetS(i, j);

                    if (youth == 0 && unemployed == 0)
                    {
                        writer.WriteLine(
                            "{0}, month {1}: no youth and no unemployed",
                            cities.GetCity(i).GetName(), j + 1
                            );
                    }
                    else if (youth == 0)
                    {
                        writer.WriteLine(
                            "{0}, month {1}: no youth population",
                            cities.GetCity(i).GetName(),
                            j + 1
```

```
                            );
                        }
                        else if (unemployed == 0)
                        {
                            writer.WriteLine(
                                "{0}, month {1}: no unemployed youth",
                                cities.GetCity(i).GetName(),
                                j + 1
                            );
                        }
                    }
                }
            }
            // If no special cases – print normal cases
            else
            {
                for (int i = 0; i < count; i++)
                {
                    writer.WriteLine(
                        "{0}, month {1}, ratio = {2:F6}",
                        cities.GetCity(cityIdx[i]).GetName(),
                        monthIdx[i] + 1,
                        minRatio
                    );
                }
            }
            writer.WriteLine();
        }
    }


    /// <summary>
    /// Sorts cities by youth and residents in ascending order
    /// </summary>
    /// <param name="cities"></param>
    /// <param name="matrix"></param>
    static void SortCities(ref CityArray cities, ref Matrix matrix)
    {
        for (int i = 0; i < cities.n - 1; i++)
        {
            for (int j = i + 1; j < cities.n; j++)
            {
                if (cities.GetCity(i) <= cities.GetCity(j))
                {
                    City tempCity = cities.GetCity(i);
                    cities.Change(i, cities.GetCity(j));
                    cities.Change(j, tempCity);
                    matrix.ChangeSRow(i, j);
                }
            }
        }
    }
}
```

5.3. Pradiniai duomenys ir rezultatai

Testas Nr.1 (Bendrasis atvejis, nėra išimtinių situacijų)

Duomenų failas „Data1_test1.txt“

```
4 4
Vilnius 600000 40000
Kaunas 300000 30000
Klaipeda 150000 20000
Siauliai 100000 10000
```

Duomenų failas „Data2_test1.txt“

```
200 300 500 100
150 250 400 100
100 200 300 100
50 100 150 100
```

Rezultatai faile „Results.txt“

```
Cities
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |Vilnius     |     600000    |    40000    |
| 2.  |Kaunas      |     300000    |    30000    |
| 3.  |Klaipeda    |     150000    |    20000    |
| 4.  |Siauliai    |     100000    |    10000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities per 4 month.
-------------------------------------------------
    1.   200   300   500   100
    2.   150   250   400   100
    3.   100   200   300   100
    4.    50   100   150   100
-------------------------------------------------

MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT
-----------------------------------
Month 3: total unemployed youth = 1350

CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO
----------------------------------------
Vilnius, month 4, ratio = 0.002500

Sorted cities by youth and residents from lower to higher
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |Siauliai    |     100000    |    10000    |
| 2.  |Klaipeda    |     150000    |    20000    |
| 3.  |Kaunas      |     300000    |    30000    |
| 4.  |Vilnius     |     600000    |    40000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities after sorting per 4 month.
-------------------------------------------------
    1.    50   100   150   100
    2.   100   200   300   100
    3.   150   250   400   100
    4.   200   300   500   100
-------------------------------------------------
```

Testas Nr.2 (Yra 3 vienodi mėnesiai su max nedarbu ir 3 su min nedarbo santykiu)

Duomenų failas „Data1_test2.txt“

```
4 4
CityA 100000 10000
CityB 90000 10000
CityC 80000 10000
CityD 70000 10000
```

Duomenų failas „Data2_test2.txt“

```
10 20 20 20
10 20 20 20
10 20 20 20
20 20 20 20
```

Rezultatai faile „Results.txt“

```
Cities
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |CityA       |     100000    |    10000    |
| 2.  |CityB       |     90000     |    10000    |
| 3.  |CityC       |     80000     |    10000    |
| 4.  |CityD       |     70000     |    10000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities per 4 month.
-------------------------------------------------
    1.    10    20    20    20
    2.    10    20    20    20
    3.    10    20    20    20
    4.    20    20    20    20
-------------------------------------------------

MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT
-----------------------------------
Month 2: total unemployed youth = 80
Month 3: total unemployed youth = 80
Month 4: total unemployed youth = 80

CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO
----------------------------------------
CityA, month 1, ratio = 0.001000
CityB, month 1, ratio = 0.001000
CityC, month 1, ratio = 0.001000

Sorted cities by youth and residents from lower to higher
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |CityD       |     70000     |    10000    |
| 2.  |CityC       |     80000     |    10000    |
| 3.  |CityB       |     90000     |    10000    |
| 4.  |CityA       |     100000    |    10000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities after sorting per 4 month.
-------------------------------------------------
    1.    20    20    20    20
    2.    10    20    20    20
    3.    10    20    20    20
    4.    10    20    20    20
-------------------------------------------------
```

Testas Nr.3 (Yra atvejų kai jaunimo nedarbas arba/ir jaunimo skaičius yra 0)

Duomenų failas „Data1_test3.txt“

```
5 4
CityA 100000 10000
CityB 90000 0
CityC 80000 10000
CityD 70000 10000
CityE 60000 10000
```

Duomenų failas „Data2_test3.txt“

```
5 5 0 0
10 10 0 0
0 0 0 0
1 1 0 0
1 1 0 0
```

Rezultatai faile „Results.txt“

```
Cities
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |CityA       |     100000    |    10000    |
| 2.  |CityB       |     90000     |     0    |
| 3.  |CityC       |     80000     |    10000    |
| 4.  |CityD       |     70000     |    10000    |
| 5.  |CityE       |     60000     |    10000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities per 4 month.
-------------------------------------------------
    1.     5     5     0     0
    2.    10    10     0     0
    3.     0     0     0     0
    4.     1     1     0     0
    5.     1     1     0     0
-------------------------------------------------

MONTHS WITH HIGHEST YOUTH UNEMPLOYMENT
-----------------------------------
Month 1: total unemployed youth = 17
Month 2: total unemployed youth = 17

CITIES WITH LOWEST YOUTH UNEMPLOYMENT RATIO
-----------------------------------------
CityA, month 3: no unemployed youth
CityA, month 4: no unemployed youth
CityB, month 1: no youth population
CityB, month 2: no youth population
CityB, month 3: no youth and no unemployed
CityB, month 4: no youth and no unemployed
CityC, month 1: no unemployed youth
CityC, month 2: no unemployed youth
CityC, month 3: no unemployed youth
CityC, month 4: no unemployed youth
CityD, month 3: no unemployed youth
CityD, month 4: no unemployed youth
CityE, month 3: no unemployed youth
CityE, month 4: no unemployed youth

Sorted cities by youth and residents from lower to higher
-------------------------------------------------
| Nr. |    City    |   Residents   |    Youth    |
-------------------------------------------------
| 1.  |CityB       |     90000     |     0    |
| 2.  |CityE       |     60000     |    10000    |
| 3.  |CityD       |     70000     |    10000    |
| 4.  |CityC       |     80000     |    10000    |
| 5.  |CityA       |     100000    |    10000    |
-------------------------------------------------

-------------------------------------------------
Unemployed youth in cities after sorting per 4 month.
-------------------------------------------------
    1.    10    10     0     0
    2.     1     1     0     0
    3.     1     1     0     0
    4.     0     0     0     0
    5.     5     5     0     0
-------------------------------------------------
```

Tekstas spausdinamas Console“

```
Program finished working.
All results are written in Results.txt file.
```

5.4. Dėstytojo pastabos