

Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Vytenis Kriščiūnas IFF-1/1

Studentas

doc. dr. Svajūnas Sajavičius

Dėstytojas

TURINYS

1. Rekursija (L1)	4
1.1. Darbo užduotis	4
1.2. Grafinės vartotojo sąsajos schema	4
1.3. Sąsajoje panaudotų komponentų keičiamos savybės	4
1.4. Klasių diagrama	5
1.5. Programos vartotojo vadovas	5
1.6. Programos tekstas	5
1.7. Pradiniai duomenys ir rezultatai	15
1.8. Dėstytojo pastabos	18
2. Dinaminis atminties valdymas (L2)	19
2.1. Darbo užduotis	19
2.2. Grafinės vartotojo sąsajos schema	19
2.3. Sąsajoje panaudotų komponentų keičiamos savybės	20
2.4. Klasių diagrama	20
2.5. Programos vartotojo vadovas	21
2.6. Programos tekstas	21
2.7. Pradiniai duomenys ir rezultatai	45
2.8. Dėstytojo pastabos	49
3. Bendrinės klasės ir testavimas (L3)	50
3.1. Darbo užduotis	50
3.2. Grafinės vartotojo sąsajos schema	50
3.3. Sąsajoje panaudotų komponentų keičiamos savybės	50
3.4. Klasių diagrama	51
3.5. Programos vartotojo vadovas	52
3.6. Programos tekstas	52
3.7. Pradiniai duomenys ir rezultatai	77

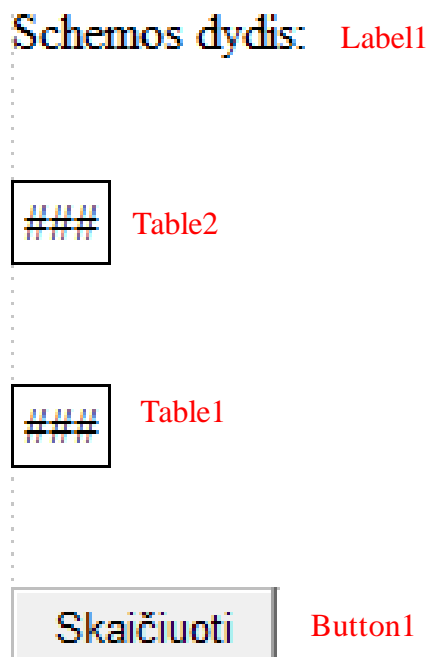
3.8.	Dėstytojo pastabos.....	81
4.	Polimorfizmas ir išimčių valdymas (L4).....	82
4.1.	Darbo užduotis	82
4.2.	Grafinės vartotojo sąsajos schema	82
4.3.	Sąsajoje panaudotų komponentų keičiamos savybės	82
4.4.	Klasių diagrama.....	83
4.5.	Programos vartotojo vadovas.....	83
4.6.	Programos tekstas.....	84
4.7.	Pradiniai duomenys ir rezultatai.....	113
4.8.	Dėstytojo pastabos.....	118

1. Rekursija (L1)

1.1. Darbo užduotis

LD_10. Kurmiai. Pavasarį sode apsigyveno kurmiai. Kiekvienas kormis išsirausė sau atskirą urvą. Suskaičiuokite, kiek kurmių apsigyveno sode ir koks yra kiekvieno kormio išrausto urvo dydis. Duomenys. Faile U3.txt yra pateikta sodo plokštuminė kurmių urvų schema – atvaizduota dvimačiu simbolių masyvu. Pirmoje failo eilutėje yra užrašytas schemos dydis: eilučių skaičius n ($5 \leq n \leq 500$) ir stulpelių skaičius m ($5 \leq m \leq 500$). Tolesnėse n eilučių yra užrašyta po m simbolių: 'z' (žemė) arba 'u' (urvas). Vienas simbolis atitinka 5 cm² plotą. Du urvo simboliai ('u') priklauso tam pačiam urvui, jeigu jie yra greta toje pačioje eilutėje arba greta tame pačiame stulpelyje. Rezultatai. Atskirose eilutėse spausdinkite sode apsigyvenusių kurmių skaičių ir kurmių urvų dydžius (cm²) surikiuotus mažėjimo tvarka.

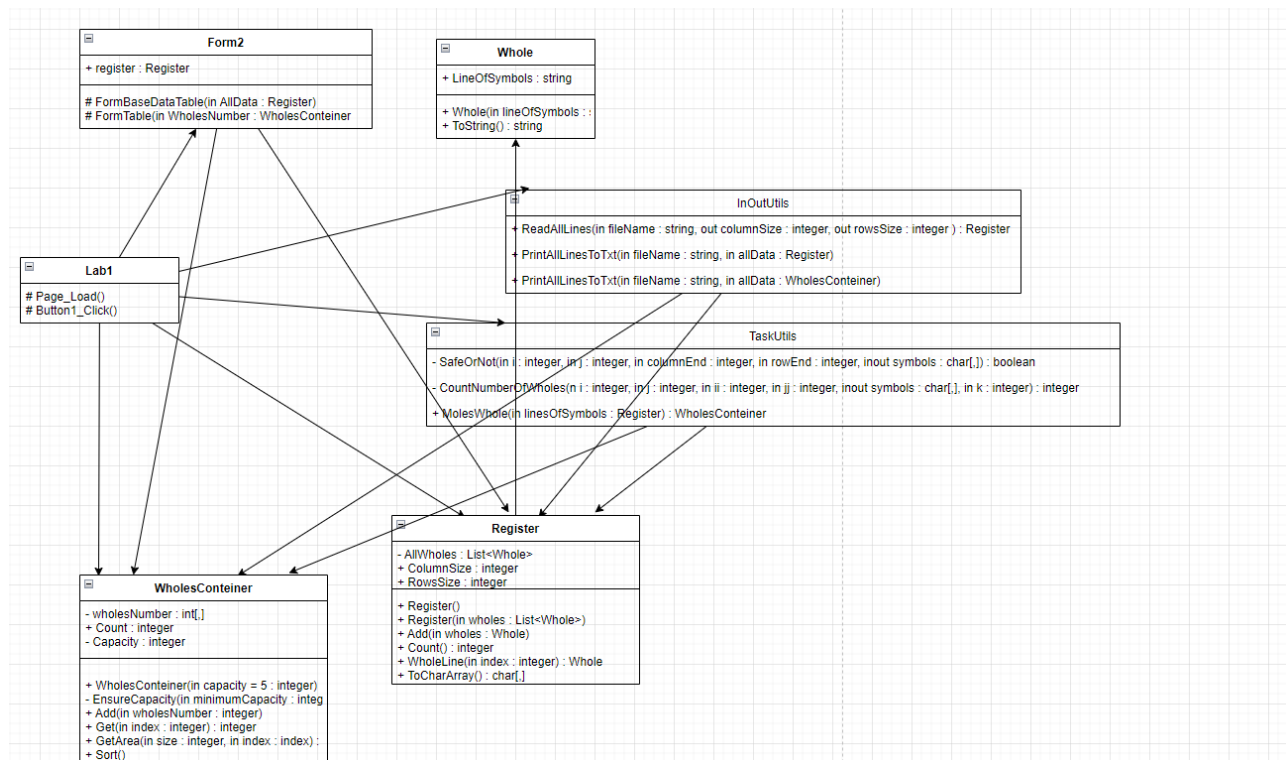
1.2. Grafinės vartotojo sąsajos schema



1.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button1	Text	Skaičiuoti
Label1	Text	Schemos dydis:
Table1	BorderColor	Black
Table1	BorderStyle	Solid
Table1	BorderWidth	1px
Table2	BorderColor	Black
Table2	BorderStyle	Solid
Table2	BorderWidth	1px

1.4. Klasių diagrama



1.5. Programos vartotojo vadovas

Atsidarius puslapiui iškart galima matyti pirmąjį mygtuką ir pradinį duomenis: schemas dydį ir simbolių lentelę. Paspaudus mygtuką, ant kurio parašyta skaičiuoti atsiranda rezultatų lentelė.

1.6. Programos tekstas

InOutUtils.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;

namespace Lab1
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    public static class InOutUtils
    {
        /// <summary>
        /// Creates a register of given information
        /// </summary>
        /// <param name="fileName">Specific file name</param>
        /// <param name="columnSize">Information about the columns size</param>
        /// <param name="rowsSize">Information about the rows size</param>
        /// <returns>Formatted list and rows, columns information </returns>
        public static Register ReadAllLines(string fileName, out int columnSize,
        out int rowsSize)
        {
            Register allWholes = new Register();
            string[] allLines = File.ReadAllLines(fileName);

            string[] Values = allLines[0].Split(' ');
        }
    }
}

```

```

        columnSize = int.Parse(Values[0]);
        rowsSize = int.Parse(Values[1]);

        for (int i = 0; i < allLines.Count(); i++)
        {
            if (i > 0)
            {
                Whole whole = new Whole(allLines[i]);
                allWholes.Add(whole);
            }
        }
        return allWholes;
    }

    /// <summary>
    /// Prints information to .txt file
    /// </summary>
    /// <param name="fileName">Specific file name</param>
    /// <param name="allData">Register information</param>
    public static void PrintAllLinesToTxt(string fileName, Register allData)
    {
        string[] lines = new string[allData.Count() + 6];
        lines[0] = string.Format(new string('-', 47));
        lines[1] = string.Format("| {0,-20} | {1,-20} |", "Eilučių skaičius",
"Stulpelių skaičius");
        lines[2] = string.Format(new string('-', 47));
        lines[3] = string.Format("| {0,20} | {1,20} |", allData.ColumnsSize,
allData.RowsSize);
        lines[4] = string.Format(new string('-', 47));

        for (int i = 0; i < allData.Count(); i++)
        {
            lines[i + 5] = allData.WholeLine(i).ToString();
        }
        lines[allData.Count() + 5] = string.Format(new string('-', 24));

        File.WriteAllLines(fileName, lines);
    }

    /// <summary>
    /// Prints information to .txt file
    /// </summary>
    /// <param name="fileName">Specific file name</param>
    /// <param name="allData">Container informattion</param>
    public static void PrintAllLinesToTxt(string fileName, WholesContainer
allData)
    {
        string[] lines = new string[allData.Count + 8];
        lines[0] = string.Format(new string('-', 24));
        lines[1] = string.Format("| {0,-20} |", "Kurmių skaičius");
        lines[2] = string.Format(new string('-', 24));
        lines[3] = string.Format("| {0,20} |", allData.Count);
        lines[4] = string.Format(new string('-', 24));
        lines[5] = string.Format("| {0,-20} |", "Kurmių urvų dydžiai");
        lines[6] = string.Format(new string('-', 24));

        for (int i = 0; i < allData.Count; i++)
        {
            lines[i + 7] = string.Format("| {0,20} |", allData.GetArea(5, i));
        }
        lines[allData.Count + 7] = string.Format(new string('-', 24));

        File.WriteAllLines(fileName, lines);
    }
}

```

```

    }
}
Whole.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab1
{
    /// <summary>
    /// Class that gives information about a lines of symbols
    /// </summary>
    public class Whole
    {
        /// <summary>
        /// String of symbols
        /// </summary>
        public string LineOfSymbols { get; set; }

        /// <summary>
        /// Line of symbols constructor
        /// </summary>
        /// <param name="lineOfSymbols"></param>
        public Whole(string lineOfSymbols)
        {
            this.LineOfSymbols = lineOfSymbols;
        }

        /// <summary>
        /// Overrriden ToString method
        /// </summary>
        /// <returns>Line of text</returns>
        public override string ToString()
        {
            return string.Format("| {0} |", this.LineOfSymbols);
        }
    }
}

```

Register.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab1
{
    /// <summary>
    /// Class that calculates given information
    /// </summary>
    public class Register
    {
        /// <summary>
        /// List of wholes information
        /// </summary>
        private List<Whole> AllWholes;

        /// <summary>
        /// Size of columns
        /// </summary>
        public int ColumnsSize;
    }
}

```

```

/// <summary>
/// Size of rows
/// </summary>
public int RowsSize;

/// <summary>
/// Formats a list
/// </summary>
public Register()
{
    AllWholes = new List<Whole>();
}

/// <summary>
/// Method that disperses information
/// </summary>
/// <param name="wholes">List of members</param>
public Register(List<Whole> wholes)
{
    AllWholes = new List<Whole>();
    for (int i = 0; i < wholes.Count; i++)
    {
        AllWholes.Add(wholes[i]);
    }
}

/// <summary>
/// Method that adds information to the list
/// </summary>
/// <param name="wholes">Single line of wholes</param>
public void Add(Whole wholes)
{
    AllWholes.Add(wholes);
}

/// <summary>
/// Method that counts how many
/// </summary>
/// <returns>Counted rezult</returns>
public int Count()
{
    return AllWholes.Count;
}

/// <summary>
/// Method that gives line of information from the list
/// </summary>
/// <param name="index"></param>
/// <returns></returns>
public Whole WholeLine(int index)
{
    return AllWholes[index];
}

/// <summary>
/// Method that creates two dimensional char array
/// </summary>
/// <returns>Formatted two dimensional char array</returns>
public char[,] ToCharArray()
{
    char[,] symbols = new char[ColumnsSize, RowsSize];
    for (int i = 0; i < ColumnsSize; i++)
    {
        for (int j = 0; j < RowsSize; j++)
        {

```



```

        symbols[i, j] = AllWholes[i].LineOfSymbols.ToCharArray()[j];
    }
    }
    return symbols;
}
}
}

```

TaskUtils.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab1
{
    /// <summary>
    /// A class that does calculations and other tasks
    /// </summary>
    public static class TaskUtils
    {
        /// <summary>
        /// Method that makes that characters are not out of bound of the array
        and that there is a searched mole's whole
        /// </summary>
        /// <param name="i">Column index</param>
        /// <param name="j">Row index</param>
        /// <param name="columnEnd">Column end index</param>
        /// <param name="rowEnd">Row end index</param>
        /// <param name="symbols">Array of symbols</param>
        /// <returns>True or false</returns>
        private static bool SafeOrNot(int i, int j, int columnEnd, int rowEnd,
char[, ] symbols)
        {
            if (i < columnEnd && i >= 0 && j < rowEnd && j >= 0 && symbols[i, j]
== 'u')
            {
                return true;
            }
            else return false;
        }

        /// <summary>
        /// Method that counts the number of mole's created wholes
        /// </summary>
        /// <param name="i">Column index</param>
        /// <param name="j">Row index</param>
        /// <param name="ii">Column end index</param>
        /// <param name="jj">Row end index</param>
        /// <param name="symbols">Array of symbols</param>
        /// <param name="k">Integer that is equal to one</param>
        /// <returns>Number of wholes</returns>
        private static int CountNumberOfWholes(int i, int j, int ii, int jj,
char[, ] symbols, int k)
        {
            if (SafeOrNot(i, j, ii, jj, symbols) == true)
            {
                symbols[i, j] = 'z';
                return k + CountNumberOfWholes(i, j + 1, ii, jj, symbols, k) +
CountNumberOfWholes(i, j - 1, ii, jj, symbols, k) + CountNumberOfWholes(i + 1, j,
ii, jj, symbols, k) + CountNumberOfWholes(i - 1, j, ii, jj, symbols, k);
            }
            return 0;
        }
    }
}

```

```

    /// <summary>
    /// Method that creates a container of found information about the moles
    /// </summary>
    /// <param name="linesOfSymbols">Register information</param>
    /// <returns>Moles Container</returns>
    public static WholesContainer MolesWholes(Register linesOfSymbols)
    {
        WholesContainer molesWholes = new WholesContainer();
        char[,] symbols = linesOfSymbols.ToCharArray();
        for (int i = 0; i < linesOfSymbols.ColumnsSize; i++)
        {
            for (int j = 0; j < linesOfSymbols.RowsSize; j++)
            {
                int numberOfWholes = CountNumberOfWholes(i, j,
linesOfSymbols.ColumnsSize, linesOfSymbols.RowsSize, symbols, 1);
                if (numberOfWholes != 0)
                {
                    molesWholes.Add(numberOfWholes);
                }
            }
        }
        return molesWholes;
    }
}
}

```

WholesContainer.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab1
{
    /// <summary>
    /// Container of counted wholes
    /// </summary>
    public class WholesContainer
    {
        /// <summary>
        /// Array of wholes
        /// </summary>
        private int[] wholesNumber;

        /// <summary>
        /// Count of moles
        /// </summary>
        public int Count { get; private set; }

        /// <summary>
        /// Total capacity of moles
        /// </summary>
        private int Capacity;

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

```

```

}

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

/// <summary>
/// Method that adds wholes number to the array
/// </summary>
/// <param name="wholesNumber">Wholes number</param>
public void Add(int wholesNumber)
{
    if (Capacity == Count)
    {
        EnsureCapacity(Capacity * 2);
    }
    this.wholesNumber[Count++] = wholesNumber;
}

/// <summary>
/// Method that gives information about mole's number of created wholes
/// </summary>
/// <param name="index">Index of a single mole</param>
/// <returns>Number of wholes</returns>
public int Get(int index)
{
    return wholesNumber[index];
}

/// <summary>
/// Method that calculates and gives information about the size of wholes
/// </summary>
/// <param name="size">Size of one whole</param>
/// <param name="index">Index of a single mole</param>
/// <returns>Claculated area informationn</returns>
public int GetArea(int size, int index)
{
    return wholesNumber[index] * size;
}

/// <summary>
/// Method that sorts information the container
/// </summary>
public void Sort()
{
    for (int i = 0; i < wholesNumber.Count(); i++)
    {
        for (int j = i + 1; j < wholesNumber.Count(); j++)
        {
            if (wholesNumber[i] < wholesNumber[j])
            {

```

Forma2.cs

12

```

        item1.Text = string.Format("Kurmių skaičius: {0}",
WholesNumber.Count);
        row1.Cells.Add(item1);
        Table1.Rows.Add(row1);

        for (int i = 0; i < WholesNumber.Count; i++)
        {
            TableRow row2 = new TableRow();
            TableCell item2 = new TableCell();
            item2.Text = string.Format("{0}", WholesNumber.GetArea(5, i));
            row2.Cells.Add(item2);
            Table1.Rows.Add(row2);
        }
    }
}

```

Lab1.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Lab1.aspx.cs"
Inherits="Lab1.Lab1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Text="Schemas dydis:
"></asp:Label>
            <br />
            <br />
            <asp:Table ID="Table2" runat="server" BorderColor="Black"
BorderStyle="Solid" BorderWidth="1px">
                </asp:Table>
            <br />
            <br />
            <asp:Table ID="Table1" runat="server" BorderColor="Black"
BorderStyle="Solid" BorderWidth="1px">
                </asp:Table>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Skaičiuoti" />
        </div>
    </form>
</body>
</html>

```

Lab1.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

//Main function of this program is to find the number of moles and their created
wholes numbers

//Vytenis Kriščiūnas
namespace Lab1

```

```

{
    /// <summary>
    /// Web form
    /// </summary>
    public partial class Lab1 : System.Web.UI.Page
    {
        /// <summary>
        /// Prints information to the screen when page loads
        /// </summary>
        /// <param name="sender">An object variable</param>
        /// <param name="e">EventArgs variable</param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (register == null)
            {
                register = new Register();
            }

            int columnSize, rowsSize;
            register = InOutUtils.ReadAllLines(Server.MapPath("App_Data/U3.txt"),
out columnSize, out rowsSize);
            register.ColumnsSize = columnSize;
            register.RowsSize = rowsSize;
            InOutUtils.PrintAllLinesToTxt(Server.MapPath("GigvenData.txt"),
register);
            Labell1.Text = "Schemas dydis: " + register.ColumnsSize + " " +
register.RowsSize;
            FormBaseDataTable(register);

        }

        /// <summary>
        /// Prints information to the screen when first button is clicked
        /// </summary>
        /// <param name="sender">An object variable</param>
        /// <param name="e">EventArgs variable</param>
        protected void Button1_Click(object sender, EventArgs e)
        {
            WholesContainer countedWholes = TaskUtils.MolesWholes(register);
            countedWholes.Sort();
            TableRow row = new TableRow();

            TableCell item = new TableCell();
            item.Text = "<b>Rezultatai</b>";
            row.Cells.Add(item);
            Table1.Rows.Add(row);

            FormTable(countedWholes);
            InOutUtils.PrintAllLinesToTxt(Server.MapPath("Results.txt"),
countedWholes);

        }

    }
}

```

Lab1.aspx.designer.cs

```

//-----
// <auto-generated>
// This code was generated by a tool.
//
// Changes to this file may cause incorrect behavior and will be lost if
// the code is regenerated.
// </auto-generated>
//-----

```

```

namespace Lab1 {

    public partial class Lab1 {

        /// <summary>
        /// form1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind
file.
        /// </remarks>
        protected global::System.Web.UI.HtmlControls.HtmlForm form1;

        /// <summary>
        /// Label1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind
file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Label Label1;

        /// <summary>
        /// Table2 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind
file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Table Table2;

        /// <summary>
        /// Table1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind
file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Table Table1;

        /// <summary>
        /// Button1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind
file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Button Button1;
    }
}

```

1.7. Pradiniai duomenys ir rezultatai

Duomenys nr. 1 (Duoti iš sąlygos)

U3.txt

6 15

zzzzzzzzzzuzuzzz

uuzuuzuuzuuzzz

zuzuzzuzuuzuzuz
zuzuzzuzzuzuuuz
zuuuuuuuzuzzzzz
zzzzzuuuzzzzzzz

GivenData.txt

Eilučių skaičius	Stulpelių skaičius

6	15

zzzzzzzzzuzuzzz	
uuzuuzuuuuuzzz	
zuzuzzuzuuzuzuz	
zuzuzzuzzuzuuuz	
zuuuuuuuzuzzzzz	
zzzzzuuuzzzzzzz	

Rezults.txt

Kurmių skaičius	

2	

Kurmių urvų dydžiai	

110	
70	

Vartotojo sąsaja

Schemas dydis: 6 15

Simboliai

zzzzzzzzzuzuzzz
uuzuuzuuuuuzzz
zuzuzzuzuuzuzuz
zuzuzzuzzuzuuuz
zuuuuuuuzuzzzzz
zzzzzuuuzzzzzzz

Rezultatai

Kurmių skaičius: 2
110
70

Skaičiuoti

Duomenys nr. 2

U3.txt

3 8

zzzuzzzu

zzzuzuuu

uuzuzzuu

GivenData.txt

Eilučių skaičius	Stulpelių skaičius

	3
	8

zzzuzzzu	
zzzuzuuu	
uuzuzzuu	

Rezults.txt

Kurmių skaičius	

	5

Kurmių urvų dydžiai	

	35
	10
	5
	5
	5

Vartotojo sąsaja

Schemos dydis: 6 15

Simboliai
zzzzzzzzzzuzuzzz
uuzuuuzuuuuuzzz
zuzuzzzuzuuuzuz
zuzuzzuzzzuzuuuz
zuuuuuuuuzuzzzzz
zzzzzuuuuzzzzzzz

Rezultatai
Kurmių skaičius: 2
110
70

Skačiuoti

1.8. Dėstytojo pastabos

Savarankiško darbo užduotys: 0.75 / nėra validavimo.

Gynimo testas: 0.0.

Programa: 6.

Ataskaita: 1.

Bendras įvertinimas: 7.

Dėstytojo komentarai:

P2 Sugadintas viršelio šablonas (nėra/bloga dėstytojo pavardės, nėra studento pavardės, neteisingi metai) -0,3. Neteisinga dėstytojo pavardė. P5 Vietoje grafinės naudotojo sąsajos schemos yra pateikta suprogramuotos formos ekrano nuotrauka -0,2. Klasių diagramą reikėtų patobulinti. P8 Nepilnas vartotojo vadovas (duomenys, veiksmų seka, rezultatai) -0,2. P13 Nekomentuoti (arba nepilnai pakomentuoti) metodai -0,2.

2. Dinaminis atminties valdymas (L2)

2.1. Darbo užduotis

LD_10. Gamykla. Gamykloje kiekvieną dieną registruojama informacija apie darbininkų pagamintas detales. Darbininkas gali gaminti per dieną skirtingo tipo detales. Suraskite daugiausiai uždirbusio darbininko pavardę, suskaičiuokite, kiek dienų jis dirbo, kiek iš viso detalių pagamino ir už kokią sumą. Sudarykite tik vieno pavadinimo detales gaminusių darbininkų sąrašą, pagamintų detalių skaičių ir sumą. Surikiuokite šį sąrašą pagal pavardes ir vardus. Duomenys:

- Tekstiniame faile U10a.txt surašyta: data (metai, mėnuo, diena), darbininko pavardė ir vardas, detalės kodas, pagamintų vienetų skaičius.

- Tekstiniame faile U10b.txt surašyta: detalės kodas, detalės pavadinimas, įkainis.

Iš duomenų rinkinio faile U10a.txt sudarykite naują duomenų rinkinį pagal nurodytą požymį (pagamintų vienetų skaičius > S, įkainis < K, įvedami klaviatūra). Sąrašas turi būti surikiuotas pagal pavardes ir vardus abėcėlės tvarka.

2.2. Grafinės vartotojo sąsajos schema

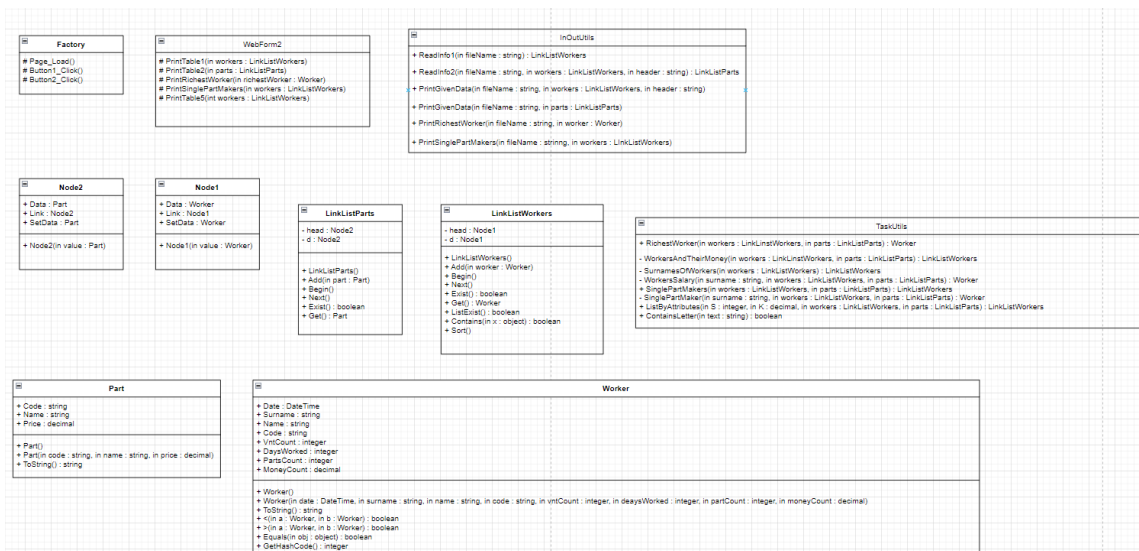
The screenshot shows a web browser window titled "A Web Page" with a URL bar containing "https://". The page content is as follows:

- Pradinių duomenų failas apie darbininkus:** Includes a "Choose File" button, "No file chosen" text, and a "FileUpload1" label.
- Pradinių duomenų failas apie dalis:** Includes a "Choose File" button, "No file chosen" text, and a "FileUpload1" label.
- Užkrauti pradinį duomenį:** A button labeled "Button1".
- Pradiniai darbininkų duomenys:** A table icon followed by the label "Table1".
- Pradiniai dalių duomenys:** A table icon followed by the label "Table2".
- Skaičiuoti:** A button labeled "Button2".
- Daugiausiai uždirbęs darbininkas:** A table icon followed by the label "Table3".
- Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:** A table icon followed by the label "Table4".
- Įveskite pagamintų vienetų skaičių S:** A text input field followed by the label "TextBox1".
- Įveskite įkainio skaičių K:** A text input field followed by the label "TextBox2".
- Sudarytas naujas duomenų rinkinys pagal požymius S ir K:** A table icon followed by the label "Table5".

2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Div	Class	TextSize
Body	Backgroud-color	Cadetblue
FileUpload1	CssClass	BoldText
FileUpload2	CssClass	BoldText
Button1	CssClass	BoldText
Button1	Text	Užkrauti pradinis duomenis
Table1	BorderStyle	Solid
Table1	BorderWidth	1px
Table1	GridLines	Both
Table1	CssClass	Table
Table2	BorderStyle	Solid
Table2	BorderWidth	1px
Table2	GridLines	Both
Table2	CssClass	Table
Table3	BorderStyle	Solid
Table3	BorderWidth	1px
Table3	GridLines	Both
Table3	CssClass	Table
Table4	BorderStyle	Solid
Table4	BorderWidth	1px
Table4	GridLines	Both
Table4	CssClass	Table
Table5	BorderStyle	Solid
Table5	BorderWidth	1px
Table5	GridLines	Both
Table5	CssClass	Table
Button2	CssClass	BoldText
Button2	Text	Skaičiuoti

2.4. Klasių diagrama



2.5. Programos vartotojo vadovas

Kai atidarome programą, matome dvi vietas įkelti failams. Visų pirma reikia įkelti detalių gamintojų duomenis (U10a.txt), tada detalių duomenis (U10b.txt). Teksto rašymo laukeliuose vartotojas gali įrašyti darbininko pagamintų vienos rūšies detalių skaičių ir detalių įkainio vertę. Užkrovus pradinį duomenį reikia spausiti ant mygtuko – užkrauti pradinį duomenį ir tada atsiranda pradinio duomenų lentelė, o į GivenData.txt failą yra išspausdinami pradiniai duomenys. Spustelėjus ant mygtuko – skaičiuoti, atsiranda dar trys lentelės su informacija ir į Rezults.txt failą yra išspausdinami rezultatai. Jeigu pradiniai duomenys nėra užkraunami, tai ir rezultatų failas nėra sukuriamas.

2.6. Programos tekstas

InOutUtils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;
using System.Text;

namespace Lab2
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    public static class InOutUtils
    {
        /// <summary>
        /// Creates a list of workers
        /// </summary>
        /// <param name="fileName">Specific file name of given data</param>
        /// <returns>Formatted list</returns>
        public static LinkedListWorkers ReadInfo1(string fileName)
        {
            LinkedListWorkers list = new LinkedListWorkers();

            string[] lines = File.ReadAllLines(fileName, Encoding.UTF8);

            foreach (string line in lines)
            {
                string[] Values = line.Split(' ');
                DateTime date = DateTime.Parse(Values[0]);
                string surname = Values[1];
                string name = Values[2];
                string code = Values[3];
                int vntCount = int.Parse(Values[4]);

                Worker worker = new Worker(date, surname, name, code, vntCount, 0,
0, 0);

                list.Add(worker);
            }
            return list;
        }

        /// <summary>
        /// Creates a list of different parts
        /// </summary>
        /// <param name="fileName">Specific file name of given data</param>
        /// <returns>Formatted list</returns>
        public static LinkedListParts ReadInfo2(string fileName)
        {

```

```

        LinkListParts list = new LinkListParts();

        string[] lines = File.ReadAllLines(fileName, Encoding.UTF8);

        foreach (string line in lines)
        {
            string[] Values = line.Split(' ');
            string code = Values[0];
            string name = Values[1];
            decimal price = decimal.Parse(Values[2]);

            Part part = new Part(code, name, price);

            list.Add(part);
        }
        return list;
    }

    /// <summary>
    /// Method that print information about workers to .txt file
    /// </summary>
    /// <param name="fileName">Specific file name of the place where
information will be printed</param>
    /// <param name="workers">List of workers</param>
    /// <param name="header">Specific header of the information</param>
    public static void PrintGivenData(string fileName, LinkListWorkers
workers, string header)
    {
        using (var writer = File.AppendText(fileName))
        {
            if (header != "")
            {
                writer.WriteLine(header);
            }
            writer.WriteLine(new string('-', 116));
            for (workers.Begin(); workers.Exist(); workers.Next())
            {
                writer.WriteLine(workers.Get().ToString());
            }
            writer.WriteLine(new string('-', 116));
            writer.WriteLine();
            writer.Close();
        }
    }

    /// <summary>
    /// Method that print information about parts to .txt file
    /// </summary>
    /// <param name="fileName">Specific file name of the place where
information will be printed</param>
    /// <param name="parts">List of parts</param>
    public static void PrintGivenData(string fileName, LinkListParts parts)
    {
        using (var writer = File.AppendText(fileName))
        {

            writer.WriteLine(new string('-', 70));
            for (parts.Begin(); parts.Exist(); parts.Next())
            {
                writer.WriteLine(parts.Get().ToString());
            }
            writer.WriteLine(new string('-', 70));
            writer.WriteLine();
            writer.Close();
        }
    }

```

```

    }

}

/// <summary>
/// Method that prints who is the richest worker
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="worker">Information about one worker</param>
public static void PrintRichestWorker(string fileName, Worker worker)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Daugiausiai uždirbęs darbininkas:");
        writer.WriteLine(new string('-', 93));
        writer.WriteLine(worker.ToString());
        writer.WriteLine(new string('-', 93));
        writer.WriteLine();
        writer.Close();
    }
}

/// <summary>
/// Method that prints information about workers who make only one type of
parts
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="workers">List of workers</param>
public static void PrintSinglePartMakers(string fileName, LinkListWorkers
workers)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Tik vieno pavadinimo detales gaminusių
darbininkų sąrašas:");
        writer.WriteLine(new string('-', 116));
        for (workers.Begin(); workers.Exist(); workers.Next())
        {
            writer.WriteLine(workers.Get().ToString());
        }
        writer.WriteLine(new string('-', 116));
        writer.WriteLine();
        writer.Close();
    }
}

}

}

```

Part.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Parts data class
    /// </summary>

```

```

public class Part
{
    /// <summary>
    /// Part's code
    /// </summary>
    public string Code { get; set; }

    /// <summary>
    /// Name of the part
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Part's value
    /// </summary>
    public decimal Price { get; set; }

    /// <summary>
    /// First constructor
    /// </summary>
    public Part()
    {

    }

    /// <summary>
    /// Second constructor
    /// </summary>
    /// <param name="code">Part's code</param>
    /// <param name="name">Name of the part</param>
    /// <param name="price">Part's value</param>
    public Part(string code, string name, decimal price)
    {
        this.Code = code;
        this.Name = name;
        this.Price = price;
    }

    /// <summary>
    /// Overriden ToString method
    /// </summary>
    /// <returns>A formatted string</returns>
    public override string ToString()
    {
        return string.Format("| {0,-20} | {1,-20} | {2,20} |", Code, Name,
Price);
    }
}

```

Worker.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Workers data class
    /// </summary>
    public class Worker

```



```

{
    /// <summary>
    /// Specific date
    /// </summary>
    public DateTime Date { get; set; }

    /// <summary>
    /// Worker's surname
    /// </summary>
    public string Surname { get; set; }

    /// <summary>
    /// Worker's name
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Part's code
    /// </summary>
    public string Code { get; set; }

    /// <summary>
    /// Number of parts
    /// </summary>
    public int VntCount { get; set; }

    /// <summary>
    /// Total number of days worked
    /// </summary>
    public int DaysWorked { get; set; }

    /// <summary>
    /// Total number of made parts
    /// </summary>
    public int PartsCount { get; set; }

    /// <summary>
    /// Total Salary of a worker
    /// </summary>
    public decimal MoneyCount { get; set; }

    /// <summary>
    /// First constructor
    /// </summary>
    public Worker()
    {
    }

    /// <summary>
    /// Second constructor
    /// </summary>
    /// <param name="date">Specific date </param>
    /// <param name="surname">Worker's surname</param>
    /// <param name="name">Worker's name</param>
    /// <param name="code">Part's code</param>
    /// <param name="vntCount">Number of parts</param>
    /// <param name="daysWorked">Total number of days worked</param>
    /// <param name="partCount">Total number of made parts</param>
    /// <param name="moneyCount">Total Salary of a worker</param>
    public Worker(DateTime date, string surname, string name, string code, int
vntCount, int daysWorked, int partCount, decimal moneyCount)
    {
        this.Date = date;
        this.Surname = surname;

```

```

        this.Name = name;
        this.Code = code;
        this.VntCount = vntCount;
        this.DaysWorked = daysWorked;
        this.PartsCount = partCount;
        this.MoneyCount = moneyCount;
    }

    /// <summary>
    /// Overriden ToString method
    /// </summary>
    /// <returns>A formatted string</returns>
    public override string ToString()
    {
        if (Date != DateTime.MinValue)
        {
            return string.Format("| {0,-20:yyyy-MM-dd} | {1,-20} | {2,-20} | {3,-20} | {4,20} |", Date, Surname, Name, Code, VntCount);
        }
        else if (Name == null)
        {
            return string.Format("| {0,-20} | {1,20} | {2,20} | {3,20} |", Surname, DaysWorked, PartsCount, MoneyCount);
        }
        else
        {
            return string.Format("| {0,-20} | {1,-20} | {2,-20} | {3,20} | {4,20} |", Surname, Name, Code, PartsCount, MoneyCount);
        }
    }

    /// <summary>
    /// Overloading operator <
    /// </summary>
    /// <param name="a">First worker</param>
    /// <param name="b">Second worker</param>
    /// <returns>If it is true or false</returns>
    public static bool operator <(Worker a, Worker b)
    {
        if (a.Surname != b.Surname)
        {
            return a.Surname.CompareTo(b.Surname) < 0;
        }
        else return a.Name.CompareTo(b.Name) < 0;
    }

    /// <summary>
    /// Overloading operator >
    /// </summary>
    /// <param name="a">First worker</param>
    /// <param name="b">Second worker</param>
    /// <returns>If it is true or false</returns>
    public static bool operator >(Worker a, Worker b)
    {
        if (a.Surname != b.Surname)
        {
            return a.Surname.CompareTo(b.Surname) > 0;
        }
        else return a.Name.CompareTo(b.Name) > 0;
    }

    /// <summary>
    /// Overriden Equals method
    /// </summary>
    /// <param name="obj">An object type variable</param>
    /// <returns>If it is true or false</returns>
    public override bool Equals(object obj)

```

```

    {
        if (Surname.Equals(obj) == true)
        {
            return true;
        }
        else if (Code.Equals(obj) == true)
        {
            return true;
        }
        else if (Date.Equals(obj) == true)
        {
            return true;
        }
        return false;
    }

    /// <summary>
    /// Overriden GetHashCode method
    /// </summary>
    /// <returns>Int type value</returns>
    public override int GetHashCode()
    {
        return base.GetHashCode();
    }
}
}

```

Node1.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Workers node class
    /// </summary>
    public sealed class Node1
    {
        /// <summary>
        /// Data about a worker
        /// </summary>
        public Worker Data { get; private set; }

        /// <summary>
        /// A link to further information about workers
        /// </summary>
        public Node1 Link { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="value">One worker's information</param>
        public Node1(Worker value)
        {
            this.Data = value;
            this.Link = null;
        }

        /// <summary>
        /// Allows to define data's information
        /// </summary>
    }
}

```

```

        public Worker SetData
        {
            set { this.Data = value; }
        }
    }
}

```

Node2.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Parts node class
    /// </summary>
    public sealed class Node2
    {
        /// <summary>
        /// Data about a part
        /// </summary>
        public Part Data { get; private set; }

        /// <summary>
        /// A link to further information about parts
        /// </summary>
        public Node2 Link { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="value">One part's information</param>
        public Node2(Part value)
        {
            this.Data = value;
            this.Link = null;
        }

        /// <summary>
        /// Allows to define data's information
        /// </summary>
        public Part SetData
        {
            set { this.Data = value; }
        }
    }
}

```

LinkedListWorkers.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Class that formats a list of workers

```

```

/// </summary>
public sealed class LinkListWorkers
{
    /// <summary>
    /// Head of the information
    /// </summary>
    private Node1 head;

    /// <summary>
    /// Created for interface of list
    /// </summary>
    private Node1 d;

    /// <summary>
    /// Constructor
    /// </summary>
    public LinkListWorkers()
    {
        this.head = new Node1(new Worker());
        this.d = null;
    }

    /// <summary>
    /// Method that adds information to the list
    /// </summary>
    /// <param name="worker">One worker</param>
    public void Add(Worker worker)
    {
        Node1 current = head;
        while (current.Link != null)
        {
            current = current.Link;
        }
        current.Link = new Node1(worker);
    }

    /// <summary>
    /// Address of the head of the list is assigned
    /// </summary>
    public void Begin()
    {
        d = head.Link;
    }

    /// <summary>
    /// Interface variable gets address of the next entry
    /// </summary>
    public void Next()
    {
        d = d.Link;
    }

    /// <summary>
    /// Return true, if list is empty
    /// </summary>
    /// <returns>If it's true or false</returns>
    public bool Exist()
    {
        return d != null;
    }

    /// <summary>
    /// Method that return data according to the interface address
    /// </summary>
    /// <returns>Data of one worker</returns>

```

```

public Worker Get()
{
    return d.Data;
}

/// <summary>
/// Return true if list has data
/// </summary>
/// <returns>If it's true or false</returns>
public bool ListExist()
{
    if (this.head.Link != null)
    {
        return true;
    }
    return false;
}

/// <summary>
/// Method that finds out if list has specific data or not
/// </summary>
/// <param name="x">Specific data</param>
/// <returns>If it is true or false</returns>
public bool Contains(object x)
{
    for (Node1 current = head.Link; current != null; current =
current.Link)
    {
        if (current.Data.Equals(x))
        {
            return true;
        }
    }
    return false;
}

/// <summary>
/// Method that sorts the list
/// </summary>
public void Sort()
{
    for (Node1 d1 = head; d1 != null; d1 = d1.Link)
    {
        Node1 min = d1;
        for (Node1 d2 = d1.Link; d2 != null; d2 = d2.Link)
        {
            if (d2.Data < min.Data)
            {
                min = d2;
            }
        }
        Worker worker = d1.Data;
        d1.SetData = min.Data;
        min.SetData = worker;
    }
}
}
}

```

LinkedListParts.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Class that formats a list of parts
    /// </summary>
    public sealed class LinkListParts
    {
        /// <summary>
        /// Head of the information
        /// </summary>
        private Node2 head;

        /// <summary>
        /// Created for interface of list
        /// </summary>
        private Node2 d;

        /// <summary>
        /// Constructor
        /// </summary>
        public LinkListParts()
        {
            this.head = new Node2(new Part());
            this.d = null;
        }

        /// <summary>
        /// Method that adds information to the list
        /// </summary>
        /// <param name="part">Single part</param>
        public void Add(Part part)
        {
            Node2 current = head;
            while (current.Link != null)
            {
                current = current.Link;
            }
            current.Link = new Node2(part);
        }

        /// <summary>
        /// Address of the head of the list is assigned
        /// </summary>
        public void Begin()
        {
            d = head.Link;
        }

        /// <summary>
        /// Interface variable gets address of the next entry
        /// </summary>
        public void Next()
        {
            d = d.Link;
        }

        /// <summary>
        /// Return true, if list is empty
        /// </summary>
        /// <returns>If it's true or false</returns>

```

```

        public bool Exist()
        {
            return d != null;
        }

        /// <summary>
        /// Method that return data according to the interface address
        /// </summary>
        /// <returns>Data of one part</returns>
        public Part Get()
        {
            return d.Data;
        }
    }
}

```

TaskUtils.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab2
{
    /// <summary>
    /// Class where calculations are being made
    /// </summary>
    public static class TaskUtils
    {
        /// <summary>
        /// Method that finds the richest worker
        /// </summary>
        /// <param name="workers">List of workers</param>
        /// <param name="parts">List of parts</param>
        /// <returns>One worker</returns>
        public static Worker RichestWorker(LinkListWorkers workers, LinkListParts
parts)
        {
            LinkListWorkers workersAndTheirMoney = WorkersAndTheirMoney(workers,
parts);
            Worker worker = new Worker();
            decimal money = 0;

            for (workersAndTheirMoney.Begin(); workersAndTheirMoney.Exist();
workersAndTheirMoney.Next())
            {
                if (workersAndTheirMoney.Get().MoneyCount > money)
                {
                    money = workersAndTheirMoney.Get().MoneyCount;
                    worker = workersAndTheirMoney.Get();
                }
            }
            return worker;
        }

        /// <summary>
        /// Method that formats a list of workers and their salaries
        /// </summary>
        /// <param name="workers">List of workers</param>
        /// <param name="parts">List of parts</param>
        /// <returns>A list</returns>
        private static LinkListWorkers WorkersAndTheirMoney(LinkListWorkers
workers, LinkListParts parts)
        {

```



```

        LinkListWorkers listOfWorkers = new LinkListWorkers();
        LinkListWorkers surnamesOfWorkers = SurnamesOfWorkers(workers);

        for (surnamesOfWorkers.Begin(); surnamesOfWorkers.Exist();
surnamesOfWorkers.Next())
        {
            Worker worker = WorkersSalary(surnamesOfWorkers.Get().Surname,
workers, parts);

            listOfWorkers.Add(worker);

        }
        return listOfWorkers;
    }

    /// <summary>
    /// Method that formats a list of workers who can repeat only once
    /// </summary>
    /// <param name="workers">List of workers</param>
    /// <returns>A list</returns>
    private static LinkListWorkers SurnamesOfWorkers(LinkListWorkers workers)
    {
        LinkListWorkers surnames = new LinkListWorkers();

        for (workers.Begin(); workers.Exist(); workers.Next())
        {
            if (!surnames.Contains(workers.Get().Surname))
            {
                surnames.Add(workers.Get());
            }
        }
        return surnames;
    }

    /// <summary>
    /// Method that count's worker's salary
    /// </summary>
    /// <param name="surname">Worker's surname</param>
    /// <param name="workers">List of workers</param>
    /// <param name="parts">List of parts</param>
    /// <returns>One worker</returns>
    private static Worker WorkersSalary(string surname, LinkListWorkers
workers, LinkListParts parts)
    {
        int DaysWorked = 0;
        int PartsCount = 0;
        decimal MoneyCount = 0;

        LinkListWorkers date = new LinkListWorkers();

        for (workers.Begin(); workers.Exist(); workers.Next())
        {
            if (workers.Get().Surname == surname)
            {
                decimal money = 0;
                PartsCount = PartsCount + workers.Get().VntCount;

                for (parts.Begin(); parts.Exist(); parts.Next())
                {
                    if (workers.Get().Code == parts.Get().Code)
                    {
                        money = workers.Get().VntCount * parts.Get().Price;
                    }
                }
            }
        }
    }

```

```

    }

    MoneyCount = MoneyCount + money;

    if (!date.Contains(workers.Get().Date))
    {
        date.Add(workers.Get());
        DaysWorked++;
    }

    }

    Worker worker = new Worker(DateTime.MinValue, surname, null, null, 0,
DaysWorked, PartsCount, MoneyCount);
    return worker;
}

/// <summary>
/// Method that formats a list of workers who make only one type of parts
/// </summary>
/// <param name="workers">List of workers</param>
/// <param name="parts">List of parts</param>
/// <returns>Formatted list</returns>
public static LinkListWorkers SinglePartMakers(LinkListWorkers workers,
LinkListParts parts)
{
    LinkListWorkers singlePartsWorkers = new LinkListWorkers();
    LinkListWorkers surnamesOfWorkers = SurnamesOfWorkers(workers);

    for (surnamesOfWorkers.Begin(); surnamesOfWorkers.Exist();
surnamesOfWorkers.Next())
    {
        Worker worker = SinglePartMaker(surnamesOfWorkers.Get().Surname,
workers, parts);

        if (worker != null)
        {
            singlePartsWorkers.Add(worker);
        }
    }
    return singlePartsWorkers;
}

/// <summary>
/// Method that finds workers who make only one type of parts
/// </summary>
/// <param name="surname">Worker's surname</param>
/// <param name="workers">List of workers</param>
/// <param name="parts">List of parts</param>
/// <returns>A worker</returns>
private static Worker SinglePartMaker(string surname, LinkListWorkers
workers, LinkListParts parts)
{
    LinkListWorkers onePartMaker = new LinkListWorkers();
    Worker worker = new Worker();
    int totalNumberOfParts = 0;
    decimal moneyCount = 0;
    int howManyParts = 0;

    for (workers.Begin(); workers.Exist(); workers.Next())
    {
        if (workers.Get().Surname == surname &&
!onePartMaker.Contains(workers.Get().Code))
        {

```

```

        onePartMaker.Add(workers.Get());
        howManyParts++;
    }

    if (workers.Get().Surname == surname)
    {
        decimal money = 0;

        totalNumberOfParts = totalNumberOfParts +
workers.Get().VntCount;

        for (parts.Begin(); parts.Exist(); parts.Next())
        {
            if (workers.Get().Code == parts.Get().Code)
            {
                money = workers.Get().VntCount * parts.Get().Price;
            }
        }

        moneyCount = moneyCount + money;

        worker = new Worker(DateTime.MinValue, workers.Get().Surname,
workers.Get().Name, workers.Get().Code, 0, 0, totalNumberOfParts, moneyCount);
    }

    }

    if (howManyParts == 1)
    {
        return worker;
    }

    return null;
}

/// <summary>
/// Method that formats a list of workers defined by attributes
/// </summary>
/// <param name="S">Number of parts</param>
/// <param name="K">Value of price</param>
/// <param name="workers">List of workers</param>
/// <param name="parts">List of parts</param>
/// <returns>A list</returns>
public static LinkedListWorkers ListByAttributes(int S, decimal K,
LinkedListWorkers workers, LinkedListParts parts)
{
    LinkedListWorkers listOfWorkers = new LinkedListWorkers();

    for (workers.Begin(); workers.Exist(); workers.Next())
    {
        if (workers.Get().VntCount > S)
        {
            for (parts.Begin(); parts.Exist(); parts.Next())
            {
                if (workers.Get().Code == parts.Get().Code &&
parts.Get().Price < K)
                {
                    listOfWorkers.Add(workers.Get());
                }
            }
        }
    }

    return listOfWorkers;
}

```

```

    /// <summary>
    /// Method that finds out if a string type variable has a specific symbol
    and returns true or false
    /// </summary>
    /// <param name="text">String type variable</param>
    /// <returns>If it is true or false</returns>
    public static bool ContainsLetter(string text)
    {
        string alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzčėėįšųūžĄČĘĮŠŲŪŽ";
        foreach (char a in alphabet)
        {
            if (text.Contains(a))
            {
                return true;
            }
        }
        return false;
    }
}
}

```

WebForm2.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

namespace Lab2
{
    /// <summary>
    /// Web form extention
    /// </summary>
    public partial class Factory : System.Web.UI.Page
    {
        /// <summary>
        /// Method that forms Table1
        /// </summary>
        /// <param name="workers">List of wrokers</param>
        protected void PrintTable1(LinkListWorkers workers)
        {
            TableRow row = new TableRow();

            TableCell cell1 = new TableCell();
            cell1.Text = "<b>Data</b>";
            row.Cells.Add(cell1);

            TableCell cell2 = new TableCell();
            cell2.Text = "<b>Pavardė</b>";
            row.Cells.Add(cell2);

            TableCell cell3 = new TableCell();
            cell3.Text = "<b>Vardas</b>";
            row.Cells.Add(cell3);

            TableCell cell4 = new TableCell();
            cell4.Text = "<b>Kodas</b>";
            row.Cells.Add(cell4);
        }
    }
}

```

```

        TableCell cell5 = new TableCell();
        cell5.Text = "<b>Detalių skaičius</b>";
        row.Cells.Add(cell5);

        Table1.Rows.Add(row);

        for (workers.Begin(); workers.Exist(); workers.Next())
        {
            TableRow newRow = new TableRow();

            TableCell newCell1 = new TableCell();
            newCell1.Text = string.Format("{0:yyyy-MM-dd}",
workers.Get().Date);
            newRow.Cells.Add(newCell1);

            TableCell newCell2 = new TableCell();
            newCell2.Text = string.Format("{0}", workers.Get().Surname);
            newRow.Cells.Add(newCell2);

            TableCell newCell3 = new TableCell();
            newCell3.Text = string.Format("{0}", workers.Get().Name);
            newRow.Cells.Add(newCell3);

            TableCell newCell4 = new TableCell();
            newCell4.Text = string.Format("{0}", workers.Get().Code);
            newRow.Cells.Add(newCell4);

            TableCell newCell5 = new TableCell();
            newCell5.Text = string.Format("{0}", workers.Get().VntCount);
            newRow.Cells.Add(newCell5);

            Table1.Rows.Add(newRow);
        }
    }

    /// <summary>
    /// Method that forms Table2
    /// </summary>
    /// <param name="parts">List of parts</param>
    protected void PrintTable2(LinkListParts parts)
    {
        TableRow row = new TableRow();

        TableCell cell1 = new TableCell();
        cell1.Text = "<b>Kodas</b>";
        row.Cells.Add(cell1);

        TableCell cell2 = new TableCell();
        cell2.Text = "<b>Pavadinimas</b>";
        row.Cells.Add(cell2);

        TableCell cell3 = new TableCell();
        cell3.Text = "<b>Kaina</b>";
        row.Cells.Add(cell3);

        Table2.Rows.Add(row);

        for (parts.Begin(); parts.Exist(); parts.Next())
        {
            TableRow newRow = new TableRow();

            TableCell newCell1 = new TableCell();
            newCell1.Text = string.Format("{0}", parts.Get().Code);
            newRow.Cells.Add(newCell1);

```

```

        TableCell newCell12 = new TableCell();
        newCell12.Text = string.Format("{0}", parts.Get().Name);
        newRow.Cells.Add(newCell12);

        TableCell newCell13 = new TableCell();
        newCell13.Text = string.Format("{0}", parts.Get().Price);
        newRow.Cells.Add(newCell13);

        Table2.Rows.Add(newRow);
    }
}

/// <summary>
/// Method that forms Table3
/// </summary>
/// <param name="richestWorker">A single worker</param>
protected void PrintRichestWorker(Worker richestWorker)
{
    TableRow row = new TableRow();

    TableCell cell11 = new TableCell();
    cell11.Text = "<b>Pavardė</b>";
    row.Cells.Add(cell11);

    TableCell cell12 = new TableCell();
    cell12.Text = "<b>Dirbtų dienų skaičius</b>";
    row.Cells.Add(cell12);

    TableCell cell13 = new TableCell();
    cell13.Text = "<b>Pagamintų detalių skaičius</b>";
    row.Cells.Add(cell13);

    TableCell cell14 = new TableCell();
    cell14.Text = "<b>Uždirbti pinigai</b>";
    row.Cells.Add(cell14);

    Table3.Rows.Add(row);

    if (richestWorker.Surname != null)
    {
        TableRow newRow = new TableRow();

        TableCell cell15 = new TableCell();
        cell15.Text = string.Format("{0}", richestWorker.Surname);
        newRow.Cells.Add(cell15);

        TableCell cell16 = new TableCell();
        cell16.Text = string.Format("{0}", richestWorker.DaysWorked);
        newRow.Cells.Add(cell16);

        TableCell cell17 = new TableCell();
        cell17.Text = string.Format("{0}", richestWorker.PartsCount);
        newRow.Cells.Add(cell17);

        TableCell cell18 = new TableCell();
        cell18.Text = string.Format("{0}", richestWorker.MoneyCount);
        newRow.Cells.Add(cell18);

        Table3.Rows.Add(newRow);
    }
}

/// <summary>
/// Method that forms Table4

```

```

/// </summary>
/// <param name="workers">List of workers</param>
protected void PrintSinglePartMakers(LinkListWorkers workers)
{
    TableRow row = new TableRow();

    TableCell cell1 = new TableCell();
    cell1.Text = "<b>Pavardė</b>";
    row.Cells.Add(cell1);

    TableCell cell2 = new TableCell();
    cell2.Text = "<b>Vardas</b>";
    row.Cells.Add(cell2);

    TableCell cell3 = new TableCell();
    cell3.Text = "<b>Detalės kodas</b>";
    row.Cells.Add(cell3);

    TableCell cell4 = new TableCell();
    cell4.Text = "<b>Skaičius vienetų detalių</b>";
    row.Cells.Add(cell4);

    TableCell cell5 = new TableCell();
    cell5.Text = "<b>Uždirbtų pinigų suma</b>";
    row.Cells.Add(cell5);

    Table4.Rows.Add(row);

    for (workers.Begin(); workers.Exist(); workers.Next())
    {
        TableRow newRow = new TableRow();

        TableCell newCell1 = new TableCell();
        newCell1.Text = string.Format("{0}", workers.Get().Surname);
        newRow.Cells.Add(newCell1);

        TableCell newCell2 = new TableCell();
        newCell2.Text = string.Format("{0}", workers.Get().Name);
        newRow.Cells.Add(newCell2);

        TableCell newCell3 = new TableCell();
        newCell3.Text = string.Format("{0}", workers.Get().Code);
        newRow.Cells.Add(newCell3);

        TableCell newCell4 = new TableCell();
        newCell4.Text = string.Format("{0}", workers.Get().PartsCount);
        newRow.Cells.Add(newCell4);

        TableCell newCell5 = new TableCell();
        newCell5.Text = string.Format("{0}", workers.Get().MoneyCount);
        newRow.Cells.Add(newCell5);

        Table4.Rows.Add(newRow);
    }
}

/// <summary>
/// Method that forms Table5
/// </summary>
/// <param name="workers">List of workers</param>
protected void PrintTable5(LinkListWorkers workers)
{
    TableRow row = new TableRow();

```

```

        TableCell cell11 = new TableCell();
        cell11.Text = "<b>Data</b>";
        row.Cells.Add(cell11);

        TableCell cell12 = new TableCell();
        cell12.Text = "<b>Pavardė</b>";
        row.Cells.Add(cell12);

        TableCell cell13 = new TableCell();
        cell13.Text = "<b>Vardas</b>";
        row.Cells.Add(cell13);

        TableCell cell14 = new TableCell();
        cell14.Text = "<b>Kodas</b>";
        row.Cells.Add(cell14);

        TableCell cell15 = new TableCell();
        cell15.Text = "<b>Detalių skaičius</b>";
        row.Cells.Add(cell15);

        Table5.Rows.Add(row);

        for (workers.Begin(); workers.Exist(); workers.Next())
        {
            TableRow newRow = new TableRow();

            TableCell newCell11 = new TableCell();
            newCell11.Text = string.Format("{0:yyyy-MM-dd}",
workers.Get().Date);
            newRow.Cells.Add(newCell11);

            TableCell newCell12 = new TableCell();
            newCell12.Text = string.Format("{0}", workers.Get().Surname);
            newRow.Cells.Add(newCell12);

            TableCell newCell13 = new TableCell();
            newCell13.Text = string.Format("{0}", workers.Get().Name);
            newRow.Cells.Add(newCell13);

            TableCell newCell14 = new TableCell();
            newCell14.Text = string.Format("{0}", workers.Get().Code);
            newRow.Cells.Add(newCell14);

            TableCell newCell15 = new TableCell();
            newCell15.Text = string.Format("{0}", workers.Get().VntCount);
            newRow.Cells.Add(newCell15);

            Table5.Rows.Add(newRow);
        }
    }
}

```

StyleSheet1.css

```

div {
    background-color :mistyrose;
}

.Table{
    background-color: yellow;
}

```



```
.BoldText{
    font-weight: bold;
}

body{
    background-color: cadetblue;
}

.TextSize{
    font-size: 18px;
}
```

Factory.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Factory.aspx.cs"
Inherits="Lab2.Factory" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="StyleSheet1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
        <div class="TextSize">
            Pradinių duomenų failas apie darbininkus:<br />
            <asp:FileUpload ID="FileUpload1" runat="server" CssClass="BoldText" />
            <br />
            <br />
            Pradinių duomenų failas apie dalis:<br />
            <asp:FileUpload ID="FileUpload2" runat="server" CssClass="BoldText" />
            <br />
            <br />
            &nbsp;<asp:Button ID="Button1" runat="server" Text="Užkrauti pradinis duomenis"
            OnClick="Button1_Click" CssClass="BoldText" />
            <br />
            <br />
            Pradiniai darbininkų duomenys:<br />
            <asp:Table ID="Table1" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Pradiniai dalių duomenys:<asp:Table ID="Table2" runat="server"
            BorderStyle="Solid" BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            <asp:Button ID="Button2" runat="server" OnClick="Button2_Click"
            Text="Skaičiuoti" CssClass="BoldText" />
            <br />
            <br />
            Daugiausiai uždirbęs darbininkas:<br />
            <asp:Table ID="Table3" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:<br />
            <asp:Table ID="Table4" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Įveskite pagamintų vienetų skaičių S:
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

```

        <br />
        <br />
        Įveskite įkainio skaičių K:
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
        <br />
        <br />
        Sudarytas naujas duomenų rinkinys pagal požymius S ir K:<asp:Table
ID="Table5" runat="server" BorderStyle="Solid" BorderWidth="1px" GridLines="Both"
CssClass="Table">
        </asp:Table>
        <br />
    </div>
</form>
</body>
</html>

```

Factory.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
//Main function of this program is to do all kinds of calculations with given data
of workers and different parts

//Vytenis Kriščiūnas
namespace Lab2
{
    /// <summary>
    /// Web form
    /// </summary>
    public partial class Factory : System.Web.UI.Page
    {
        /// <summary>
        /// List of parts
        /// </summary>
        LinkListParts parts;

        /// <summary>
        /// List of workers
        /// </summary>
        LinkListWorkers workers;

        /// <summary>
        /// The richest worker
        /// </summary>
        Worker richestWorker;

        /// <summary>
        /// List of workers who allways make the same parts
        /// </summary>
        LinkListWorkers singlePartMakers;

        /// <summary>
        /// List formed by given attributes
        /// </summary>
        LinkListWorkers listByAttributes;

        /// <summary>
        /// Prints information to the screen when page loads
    }
}

```

```

/// </summary>
/// <param name="sender">An object variable</param>
/// <param name="e">EventArgs variable</param>
protected void Page_Load(object sender, EventArgs e)
{
    if (parts == null)
    {
        parts = new LinkListParts();
    }
    if (workers == null)
    {
        workers = new LinkListWorkers();
    }
    if (richestWorker == null)
    {
        richestWorker = new Worker();
    }
    if (singlePartMakers == null)
    {
        singlePartMakers = new LinkListWorkers();
    }
    if (listByAttributes == null)
    {
        listByAttributes = new LinkListWorkers();
    }

    parts = Session["parts"] as LinkListParts;
    workers = Session["workers"] as LinkListWorkers;

    if (workers != null && parts != null)
    {
        PrintTable1(workers);
        PrintTable2(parts);
    }
}

/// <summary>
/// Prints and reads given information when the first button is clicked
/// </summary>
/// <param name="sender">An object variable</param>
/// <param name="e">EventArgs variable</param>
protected void Button1_Click(object sender, EventArgs e)
{
    Session.Remove("parts");
    Session.Remove("workers");
    for (int i = Table1.Rows.Count - 1; i >= 0; i--)
    {
        Table1.Rows.RemoveAt(i);
    }
    for (int i = Table2.Rows.Count - 1; i >= 0; i--)
    {
        Table2.Rows.RemoveAt(i);
    }

    string Fr = Server.MapPath("App_Data/GivenData.txt");
    File.Delete(Fr);

    if (FileUpload1.HasFile && FileUpload2.HasFile)
    {
        string Fd1 = Server.MapPath("App_Data/U10a.txt"); // workers
        string Fd2 = Server.MapPath("App_Data/U10b.txt"); // parts
    }
}

```

```

        if (FileUpload1.FileName.EndsWith("a.txt") &&
FileUpload2.FileName.EndsWith("b.txt"))
        {

            parts = InOutUtils.ReadInfo2(Fd2);
            workers = InOutUtils.ReadInfo1(Fd1);

            PrintTable1(workers);
            PrintTable2(parts);

            InOutUtils.PrintGivenData(Fr, workers, "");
            InOutUtils.PrintGivenData(Fr, parts);

            Session["parts"] = parts;
            Session["workers"] = workers;

        }
    }

    /// <summary>
    /// Calculates needed information and prints it when the second button is
clicked
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Button2_Click(object sender, EventArgs e)
    {
        string Fr = Server.MapPath("App_Data/Rezults.txt");
        File.Delete(Fr);
        int S = 0;
        decimal K = 0;

        if (TextBox1.Text != "" && TextBox2.Text != "" &&
TaskUtils.ContainsLetter(TextBox1.Text) == false &&
TaskUtils.ContainsLetter(TextBox2.Text) == false)
        {
            S = int.Parse(TextBox1.Text);
            K = decimal.Parse(TextBox2.Text);
        }

        if (workers != null && parts != null)
        {
            richestWorker = TaskUtils.RichestWorker(workers, parts);

            singlePartMakers = TaskUtils.SinglePartMakers(workers, parts);
            singlePartMakers.Sort();
        }

        PrintRichestWorker(richestWorker);

        PrintSinglePartMakers(singlePartMakers);

        if (workers != null && parts != null && S != 0 && K != 0)
        {
            listByAttributes = TaskUtils.ListByAttributes(S, K, workers,
parts);
            listByAttributes.Sort();
        }

        PrintTable5(listByAttributes);
    }

```

```

        if (richestWorker.Surname != null)
        {
            InOutUtils.PrintRichestWorker(Fr, richestWorker);
        }

        if (singlePartMakers.ListExist() == true)
        {
            InOutUtils.PrintSinglePartMakers(Fr, singlePartMakers);
        }

        if (listByAttributes.ListExist() == true)
        {
            InOutUtils.PrintGivenData(Fr, listByAttributes,
string.Format("Sudarytas naujas duomenų rinkinys pagal požymius S ir K:"));
        }

        if (richestWorker.Surname == null && singlePartMakers.ListExist() ==
false && listByAttributes.ListExist() == false)
        {
            File.Delete(Fr);
        }
    }
}
}

```

2.7. Pradiniai duomenys ir rezultatai

Duomenys nr. 1

U10a.txt

```

2021-07-21 Pavardenis Vardenis IOp5 5
2021-07-21 Pavardenis Vardenis LkD5 4
2021-07-21 Kazkoks Kazkas LkD5 4
2021-06-21 Kazkoks Kazkas LkD5 4
2021-06-21 Kazkoks Kazkas LkD5 7
2022-06-21 Aetraitis Petras Klj 7

```

U10b.txt

```

IOp5 metal 3
LkD5 plast 6.3
Bpm6 kalp 2
Klj milst 9.3

```

GivenData.txt

```

-----
-----
| 2021-07-21          | Pavardenis          | Vardenis            | IOp5
|                    | 5 |                 |                     |
| 2021-07-21          | Pavardenis          | Vardenis            | LkD5
|                    | 4 |                 |                     |
| 2021-07-21          | Kazkoks              | Kazkas               | LkD5
|                    | 4 |                 |                     |
| 2021-06-21          | Kazkoks              | Kazkas               | LkD5
|                    | 4 |                 |                     |
| 2021-06-21          | Kazkoks              | Kazkas               | LkD5
|                    | 7 |                 |                     |
| 2022-06-21          | Aetraitis            | Petras               | Klj
|                    | 7 |                 |                     |
-----
-----

```

IOp5	metal		3	
LkD5	plast		6.3	
Bpm6	kalp		2	
Klj	milst		9.3	

Rezults.txt

Daugiausiai uždirbęs darbininkas:

Kazkoks		2		15	
94.5					

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Aetraitis	Petras	Klj	
7	65.1		
Kazkoks	Kazkas	LkD5	
15	94.5		

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

2022-06-21	Aetraitis	Petras	Klj
	7		
2021-07-21	Kazkoks	Kazkas	LkD5
	4		
2021-06-21	Kazkoks	Kazkas	LkD5
	4		
2021-06-21	Kazkoks	Kazkas	LkD5
	7		
2021-07-21	Pavardenis	Vardenis	LkD5
	4		
2021-07-21	Pavardenis	Vardenis	IOp5
	5		

Vartotojo sąsaja

Pradinių duomenų failas apie darbininkus:
 No file chosen

Pradinių duomenų failas apie dalis:
 No file chosen

Pradiniai darbininkų duomenys:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-07-21	Pavardenis	Vardenis	IOP5	5
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	7
2022-06-21	Aetraitis	Petras	Klj	7

Pradiniai dalių duomenys:

Kodas	Pavadinimas	Kaina
IOP5	metal	3
LkD5	plast	6.3
Bpm6	kalp	2
Klj	milst	9.3

Daugiausiai uždirbęs darbininkas:

Pavardė	Dirbtų dienų skaičius	Pagamintų detalių skaičius	Uždirbti pinigai
Kazkoks	2	15	94.5

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Pavardė	Vardas	Detalės kodas	Skaičius vienodų detalių	Uždirbtų pinigų suma
Aetraitis	Petras	Klj	7	65.1
Kazkoks	Kazkas	LkD5	15	94.5

Įveskite pagamintų vienetų skaičių S:

Įveskite įkainio skaičių K:

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2022-06-21	Aetraitis	Petras	Klj	7
2021-07-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	7
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Pavardenis	Vardenis	IOP5	5

Duomenys nr. 2

U10a.txt

```
2021-07-21 Pavardenis Vardenis IOP5 5
2021-07-21 Pavardenis Vardenis LkD5 4
2021-07-21 Bazkoks Lazkas Bpm6 4
2021-06-21 Bazkoks Lazkas LkD5 4
2021-06-21 Bazkoks Lazkas LkD5 7
2022-06-21 Aetraitis Petras Klj 7
2022-06-28 Valancius Justas IOP5 8
```

U10b.txt

```
IOP5 metal 3
LkD5 plast 2
Bpm6 kalp 2
Klj milst 9.3
```

GivenData.txt

```
-----
| 2021-07-21          | Pavardenis          | Vardenis          | IOP5
|                      | 5 |
| 2021-07-21          | Pavardenis          | Vardenis          | LkD5
|                      | 4 |
```

2021-07-21		Bazkoks		Lazkas		Bpm6
	4					
2021-06-21		Bazkoks		Lazkas		LkD5
	4					
2021-06-21		Bazkoks		Lazkas		LkD5
	7					
2022-06-21		Aetraitis		Petras		Klj
	7					
2022-06-28		Valancius		Justas		IOp5
	8					

IOp5	metal		3	
LkD5	plast		2	
Bpm6	kalp		2	
Klj	milst		9.3	

Rezults.txt

Daugiausiai uždirbęs darbininkas:

Aetraitis		1		7
65.1				

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Aetraitis		Petras		Klj	
7		65.1			
Valancius		Justas		IOp5	
8		24			

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

2021-06-21		Bazkoks		Lazkas		LkD5
	7					
2021-07-21		Pavardenis		Vardenis		IOp5
	5					
2022-06-28		Valancius		Justas		IOp5
	8					

Vartotojo sąsaja

Pradinių duomenų failas apie darbininkus:

No file chosen

Pradinių duomenų failas apie dalis:

No file chosen

Pradiniai darbininkų duomenys:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-07-21	Pavardenis	Vardenis	IOp5	5
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Bazkoks	Lazkas	Bpm6	4
2021-06-21	Bazkoks	Lazkas	LkD5	4
2021-06-21	Bazkoks	Lazkas	LkD5	7
2022-06-21	Aetraitis	Petras	Klj	7
2022-06-28	Valancius	Justas	IOp5	8

Pradiniai dalių duomenys:

Kodas	Pavadinimas	Kaina
IOp5	metal	3
LkD5	plast	2
Bpm6	kalp	2
Klj	mišt	9.3

Daugiausiai uždirbęs darbininkas:

Pavardė	Dirbtų dienų skaičius	Pagamintų detalių skaičius	Uždirbti pinigai
Aetraitis	1	7	65.1

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Pavardė	Vardas	Detalės kodas	Skaičius vienodų detalių	Uždirbtų pinigų suma
Aetraitis	Petras	Klj	7	65.1
Valancius	Justas	IOp5	8	24

Įveskite pagamintų vienetų skaičių S:

Įveskite įkainio skaičių K:

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-06-21	Bazkoks	Lazkas	LkD5	7
2021-07-21	Pavardenis	Vardenis	IOp5	5
2022-06-28	Valancius	Justas	IOp5	8

2.8. Dėstytojo pastabos

Gynimo testas: 0.0.

Programa: 6.

Ataskaita: 1.

Bendras įvertinimas: 7.

3. Bendrinės klasės ir testavimas (L3)

3.1. Darbo užduotis

LD_10. Gamykla. Gamykloje kiekvieną dieną registruojama informacija apie darbininkų pagamintas detales. Darbininkas gali gaminti per dieną skirtingo tipo detales. Suraskite daugiausiai uždirbusio darbininko pavardę, suskaičiuokite, kiek dienų jis dirbo, kiek iš viso detalių pagamino ir už kokią sumą. Sudarykite tik vieno pavadinimo detales gaminusių darbininkų sąrašą, pagamintų detalių skaičių ir sumą. Surikiuokite šį sąrašą pagal pavardes ir vardus. Duomenys:

- Tekstiniame faile U10a.txt surašyta: data (metai, mėnuo, diena), darbininko pavardė ir vardas, detalės kodas, pagamintų vienetų skaičius.

- Tekstiniame faile U10b.txt surašyta: detalės kodas, detalės pavadinimas, įkainis.

Iš duomenų rinkinio faile U10a.txt sudarykite naują duomenų rinkinį pagal nurodytą požymį (pagamintų vienetų skaičius > S, įkainis < K, įvedami klaviatūra). Sąrašas turi būti surikiuotas pagal pavardes ir vardus abėcėlės tvarka.

3.2. Grafinės vartotojo sąsajos schema

A Web Page

https://

Pradiniai duomenų failai apie darbininkus:
Choose File No file chosen FileUpload1

Pradiniai duomenų failai apie dalis:
Choose File No file chosen FileUpload1

Užkrauti pradinis duomenis Button1

Pradiniai darbininkų duomenys:
Table1

Pradiniai dalių duomenys:
Table2

Skaičiuoti Button2

Daugiausiai uždirbęs darbininkas:
Table3

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:
Table4

Įveskite pagamintų vienetų skaičių S: TextBox1

Įveskite įkainio skaičių K: TextBox2

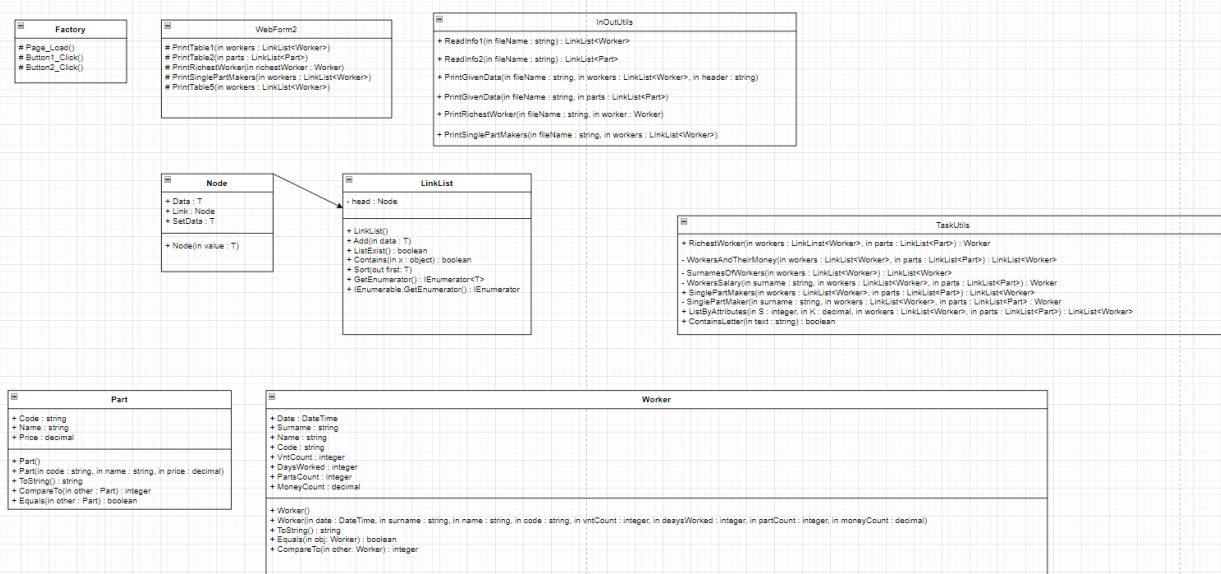
Sudarytas naujas duomenų rinkinys pagal požymius S ir K:
Table5

3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Div	Class	FontSize

Body	Backgroud-color	Cadetblue
FileUpload1	CssClass	BoldText
FileUpload2	CssClass	BoldText
Button1	CssClass	BoldText
Button1	Text	Užkrauti pradinis duomenis
Table1	BorderStyle	Solid
Table1	BorderWidth	1px
Table1	GridLines	Both
Table1	CssClass	Table
Table2	BorderStyle	Solid
Table2	BorderWidth	1px
Table2	GridLines	Both
Table2	CssClass	Table
Table3	BorderStyle	Solid
Table3	BorderWidth	1px
Table3	GridLines	Both
Table3	CssClass	Table
Table4	BorderStyle	Solid
Table4	BorderWidth	1px
Table4	GridLines	Both
Table4	CssClass	Table
Table5	BorderStyle	Solid
Table5	BorderWidth	1px
Table5	GridLines	Both
Table5	CssClass	Table
Button2	CssClass	BoldText
Button2	Text	Skaičiuoti

3.4. Klasių diagrama



3.5. Programos vartotojo vadovas

Kai atidarome programą, matome dvi vietas įkelti failams. Visų pirma reikia įkelti detalių gamintojų duomenis (U10a.txt), tada detalių duomenis (U10b.txt). Teksto rašymo laukeliuose vartotojas gali įrašyti darbininko pagamintų vienos rūšies detalių skaičių ir detalių įkainio vertę. Užkrovus pradinį duomenį reikia spausiti ant mygtuko – užkrauti pradinį duomenį ir tada atsiranda pradinio duomenų lentelė, o į GivenData.txt failą yra išspausdinami pradiniai duomenys. Spustelėjus ant mygtuko – skaičiuoti, atsiranda dar trys lentelės su informacija ir į Rezults.txt failą yra išspausdinami rezultatai. Jeigu pradiniai duomenys nėra užkraunami, tai ir rezultatų failas nėra sukuriamas.

3.6. Programos tekstas

InOutUtils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;
using System.Text;

namespace Lab3
{
    /// <summary>
    /// Class that prints or reads information
    /// </summary>
    public static class InOutUtils
    {
        /// <summary>
        /// Creates a list of workers
        /// </summary>
        /// <param name="fileName">Specific file name of given data</param>
        /// <returns>Formatted list</returns>
        public static LinkedList<Worker> ReadInfo1(string fileName)
        {
            LinkedList<Worker> list = new LinkedList<Worker>();

            string[] lines = File.ReadAllLines(fileName, Encoding.UTF8);

            foreach (string line in lines)
            {
                string[] Values = line.Split(' ');
                DateTime date = DateTime.Parse(Values[0]);
                string surname = Values[1];
                string name = Values[2];
                string code = Values[3];
                int vntCount = int.Parse(Values[4]);

                Worker worker = new Worker(date, surname, name, code, vntCount, 0,
0, 0);

                list.Add(worker);
            }
            return list;
        }

        /// <summary>
        /// Creates a list of different parts
        /// </summary>
        /// <param name="fileName">Specific file name of given data</param>
        /// <returns>Formatted list</returns>
        public static LinkedList<Part> ReadInfo2(string fileName)
        {

```

```

LinkedList<Part> list = new LinkedList<Part>();

string[] lines = File.ReadAllLines(fileName, Encoding.UTF8);

foreach (string line in lines)
{
    string[] Values = line.Split(' ');
    string code = Values[0];
    string name = Values[1];
    decimal price = decimal.Parse(Values[2]);

    Part part = new Part(code, name, price);

    list.Add(part);
}
return list;
}

/// <summary>
/// Method that print information about workers to .txt file
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="workers">List of workers</param>
/// <param name="header">Specific header of the information</param>
public static void PrintGivenData(string fileName, LinkedList<Worker>
workers, string header)
{
    using (var writer = File.AppendText(fileName))
    {
        if (header != "")
        {
            writer.WriteLine(header);
        }
        writer.WriteLine(new string('-', 116));

        foreach (Worker worker in workers)
        {
            writer.WriteLine(worker.ToString());
        }
        writer.WriteLine(new string('-', 116));
        writer.WriteLine();
        writer.Close();
    }
}

/// <summary>
/// Method that print information about parts to .txt file
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="parts">List of parts</param>
public static void PrintGivenData(string fileName, LinkedList<Part> parts)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine(new string('-', 70));
        foreach (Part part in parts)
        {
            writer.WriteLine(part.ToString());
        }
        writer.WriteLine(new string('-', 70));
        writer.WriteLine();
    }
}

```

```

        writer.Close();
    }

}

/// <summary>
/// Method that prints who is the richest worker
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="worker">Information about one worker</param>
public static void PrintRichestWorker(string fileName, Worker worker)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Daugiausiai uždirbęs darbininkas:");
        writer.WriteLine(new string('-', 93));
        writer.WriteLine(worker.ToString());
        writer.WriteLine(new string('-', 93));
        writer.WriteLine();
        writer.Close();
    }
}

/// <summary>
/// Method that prints information about workers who make only one type of
parts
/// </summary>
/// <param name="fileName">Specific file name of the place where
information will be printed</param>
/// <param name="workers">List of workers</param>
public static void PrintSinglePartMakers(string fileName, LinkList<Worker>
workers)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Tik vieno pavadinimo detales gaminusių
darbininkų sąrašas:");
        writer.WriteLine(new string('-', 116));
        foreach (Worker worker in workers)
        {
            writer.WriteLine(worker.ToString());
        }
        writer.WriteLine(new string('-', 116));
        writer.WriteLine();
        writer.Close();
    }
}

}

}

```

Part.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab3
{
    /// <summary>
    /// Parts data class

```

```

/// </summary>
public class Part : IComparable<Part>, IEquatable<Part>
{
    /// <summary>
    /// Part's code
    /// </summary>
    public string Code { get; set; }

    /// <summary>
    /// Name of the part
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Part's value
    /// </summary>
    public decimal Price { get; set; }

    /// <summary>
    /// First constructor
    /// </summary>
    public Part()
    {

    }

    /// <summary>
    /// Second constructor
    /// </summary>
    /// <param name="code">Part's code</param>
    /// <param name="name">Name of the part</param>
    /// <param name="price">Part's value</param>
    public Part(string code, string name, decimal price)
    {
        this.Code = code;
        this.Name = name;
        this.Price = price;
    }

    /// <summary>
    /// Overriden ToString method
    /// </summary>
    /// <returns>A formatted string</returns>
    public override string ToString()
    {
        return string.Format("| {0,-20} | {1,-20} | {2,20} |", Code, Name,
Price);
    }

    /// <summary>
    /// Method that compares parts
    /// </summary>
    /// <param name="other">Second part</param>
    /// <returns>NotImplementedException</returns>
    public int CompareTo(Part other)
    {
        throw new NotImplementedException();
    }

    /// <summary>
    /// Method that finds out if parts are equal or not
    /// </summary>
    /// <param name="other">Second part</param>
    /// <returns>NotImplementedException</returns>
    public bool Equals(Part other)

```

```

        {
            throw new NotImplementedException();
        }
    }
}

```

Worker.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab3
{
    /// <summary>
    /// Workers data class
    /// </summary>
    public class Worker: IComparable<Worker>, IEquatable<Worker>
    {
        /// <summary>
        /// Specific date
        /// </summary>
        public DateTime Date { get; set; }

        /// <summary>
        /// Worker's surname
        /// </summary>
        public string Surname { get; set; }

        /// <summary>
        /// Worker's name
        /// </summary>
        public string Name { get; set; }

        /// <summary>
        /// Part's code
        /// </summary>
        public string Code { get; set; }

        /// <summary>
        /// Number of parts
        /// </summary>
        public int VntCount { get; set; }

        /// <summary>
        /// Total number of days worked
        /// </summary>
        public int DaysWorked { get; set; }

        /// <summary>
        /// Total number of made parts
        /// </summary>
        public int PartsCount { get; set; }

        /// <summary>
        /// Total Salary of a worker
        /// </summary>
        public decimal MoneyCount { get; set; }

        /// <summary>
        /// First constructor
        /// </summary>

```



```

public Worker()
{

}

/// <summary>
/// Second constructor
/// </summary>
/// <param name="date">Specific date </param>
/// <param name="surname">Worker's surname</param>
/// <param name="name">Worker's name</param>
/// <param name="code">Part's code</param>
/// <param name="vntCount">Number of parts</param>
/// <param name="daysWorked">Total number of days worked</param>
/// <param name="partCount">Total number of made parts</param>
/// <param name="moneyCount">Total Salary of a worker</param>
public Worker(DateTime date, string surname, string name, string code, int
vntCount, int daysWorked, int partCount, decimal moneyCount)
{
    this.Date = date;
    this.Surname = surname;
    this.Name = name;
    this.Code = code;
    this.VntCount = vntCount;
    this.DaysWorked = daysWorked;
    this.PartsCount = partCount;
    this.MoneyCount = moneyCount;
}

/// <summary>
/// Overriden ToString method
/// </summary>
/// <returns>A formated string</returns>
public override string ToString()
{
    if (Date != DateTime.MinValue)
    {
        return string.Format("| {0,-20:yyyy-MM-dd} | {1,-20} | {2,-20} |
{3,-20} | {4,20} |", Date, Surname, Name, Code, VntCount);
    }
    else if (Name == null)
    {
        return string.Format("| {0,-20} | {1,20} | {2,20} | {3,20} |",
Surname, DaysWorked, PartsCount, MoneyCount);
    }
    else
        return string.Format("| {0,-20} | {1,-20} | {2,-20} | {3,20} |
{4,20} |", Surname, Name, Code, PartsCount, MoneyCount);
}

/// <summary>
/// Equals method
/// </summary>
/// <param name="obj">Other worker</param>
/// <returns>If it is true or false</returns>
public bool Equals(Worker obj)
{
    if (Surname.Equals(obj) == true)
    {
        return true;
    }
    else if (Code.Equals(obj) == true)
    {
        return true;
    }
}

```

```

    }
    else if (Date.Equals(obj) == true)
    {
        return true;
    }
    return false;
}

/// <summary>
/// Method that compares two workers
/// </summary>
/// <param name="other">Other worker</param>
/// <returns>An integer</returns>
public int CompareTo(Worker other)
{
    if ((object)other == null)
    {
        return 1;
    }
    if (Surname.CompareTo(other.Surname) != 0)
    {
        return Surname.CompareTo(other.Surname);
    }
    else
        return Name.CompareTo(other.Name);
}

}
}

```

LinkedList.cs

```

using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Web;

namespace Lab3
{
    /// <summary>
    /// Class that formats a list of given data
    /// </summary>
    public sealed class LinkedList<T> : IEnumerable<T> where T : IComparable<T>,
new()
    {
        private sealed class Node
        {
            /// <summary>
            /// Information about given data
            /// </summary>
            public T Data { get; private set; }

            /// <summary>
            /// A link to further information about given data
            /// </summary>
            public Node Link { get; set; }

            /// <summary>
            /// Constructor
            /// </summary>
            /// <param name="value">Data's information</param>
            public Node(T value)

```

```

    {
        this.Data = value;
        this.Link = null;
    }

    /// <summary>
    /// Allows to define data's information
    /// </summary>
    public T SetData
    {
        set { this.Data = value; }
    }

}

/// <summary>
/// Head of the information
/// </summary>
private Node head;

/// <summary>
/// Constructor
/// </summary>
public LinkList()
{
    this.head = new Node(new T());
}

/// <summary>
/// Method that adds information to the list
/// </summary>
/// <param name="worker">Given data</param>
public void Add(T data)
{
    Node current = head;
    while (current.Link != null)
    {
        current = current.Link;
    }
    current.Link = new Node(data);
}

/// <summary>
/// Return true if list has data
/// </summary>
/// <returns>If it's true or false</returns>
public bool ListExist()
{
    if (this.head.Link != null)
    {
        return true;
    }
    return false;
}

/// <summary>
/// Method that finds out if list has specific data or not
/// </summary>
/// <param name="x">Specific data</param>
/// <returns>If it is true or false</returns>
public bool Contains(object x)
{

```

```

        for (Node current = head.Link; current != null; current =
current.Link)
        {
            if (current.Data.Equals(x))
            {
                return true;
            }
        }
        return false;
    }

    /// <summary>
    /// Method that sorts the list
    /// </summary>
    /// <param name="first">Returns first member of the list</param>
    public void Sort(out T first)
    {
        int x = 0;
        first = default(T);

        for (Node d1 = head.Link; d1 != null; d1 = d1.Link)
        {
            Node min = d1;
            for (Node d2 = d1.Link; d2 != null; d2 = d2.Link)
            {
                if (d2.Data.CompareTo(min.Data) < 0)
                {
                    min = d2;
                }
            }
            T worker = d1.Data;
            d1.SetData = min.Data;
            min.SetData = worker;

            if (x == 0)
            {
                first = d1.Data;
            }
            x++;
        }
    }

    public IEnumerator<T> GetEnumerator()
    {
        for (Node dd = head.Link; dd != null; dd = dd.Link)
        {
            yield return dd.Data;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
}

```

TaskUtils.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

```

```

namespace Lab3
{
    /// <summary>
    /// Class where calculations are being made
    /// </summary>
    public static class TaskUtils
    {
        /// <summary>
        /// Method that finds the richest worker
        /// </summary>
        /// <param name="workers">List of workers</param>
        /// <param name="parts">List of parts</param>
        /// <returns>One worker</returns>
        public static Worker RichestWorker(LinkedList<Worker> workers,
LinkedList<Part> parts)
        {
            LinkedList<Worker> workersAndTheirMoney = WorkersAndTheirMoney(workers,
parts);
            Worker worker = new Worker();
            decimal money = 0;

            foreach (Worker w in workersAndTheirMoney)
            {
                if (w.MoneyCount > money)
                {
                    money = w.MoneyCount;
                    worker = w;
                }
            }

            return worker;
        }

        /// <summary>
        /// Method that formats a list of workers and their salaries
        /// </summary>
        /// <param name="workers">List of workers</param>
        /// <param name="parts">List of parts</param>
        /// <returns>A list</returns>
        private static LinkedList<Worker> WorkersAndTheirMoney(LinkedList<Worker>
workers, LinkedList<Part> parts)
        {
            LinkedList<Worker> listOfWorkers = new LinkedList<Worker>();
            LinkedList<Worker> surnamesOfWorkers = SurnamesOfWorkers(workers);

            foreach (Worker w in surnamesOfWorkers)
            {
                Worker worker = WorkersSalary(w.Surname, workers, parts);

                listOfWorkers.Add(worker);
            }

            return listOfWorkers;
        }

        /// <summary>
        /// Method that formats a list of workers who can repeat only once
        /// </summary>
        /// <param name="workers">List of workers</param>
        /// <returns>A list</returns>
        private static LinkedList<Worker> SurnamesOfWorkers(LinkedList<Worker>
workers)
        {
            LinkedList<Worker> surnames = new LinkedList<Worker>();

```

```

        foreach (Worker w in workers)
        {
            if (!surnames.Contains(w.Surname))
            {
                surnames.Add(w);
            }
        }

        return surnames;
    }

    /// <summary>
    /// Method that count's worker's salary
    /// </summary>
    /// <param name="surname">Worker's surname</param>
    /// <param name="workers">List of workers</param>
    /// <param name="parts">List of parts</param>
    /// <returns>One worker</returns>
    private static Worker WorkersSalary(string surname, LinkList<Worker>
workers, LinkList<Part> parts)
    {
        int DaysWorked = 0;
        int PartsCount = 0;
        decimal MoneyCount = 0;

        LinkList<Worker> date = new LinkList<Worker>();

        foreach (Worker w in workers)
        {
            if (w.Surname == surname)
            {
                decimal money = 0;
                PartsCount = PartsCount + w.VntCount;

                foreach (Part p in parts)
                {
                    if (w.Code == p.Code)
                    {
                        money = w.VntCount * p.Price;
                    }
                }

                MoneyCount = MoneyCount + money;

                if (!date.Contains(w.Date))
                {
                    date.Add(w);
                    DaysWorked++;
                }
            }
        }

        Worker worker = new Worker(DateTime.MinValue, surname, null, null, 0,
DaysWorked, PartsCount, MoneyCount);
        return worker;
    }

    /// <summary>
    /// Method that formats a list of workers who make only one type of parts
    /// </summary>
    /// <param name="workers">List of workers</param>
    /// <param name="parts">List of parts</param>
    /// <returns>Formatted list</returns>

```

```

        public static LinkList<Worker> SinglePartMakers(LinkList<Worker> workers,
LinkList<Part> parts)
        {
            LinkList<Worker> singlePartsWorkers = new LinkList<Worker>();
            LinkList<Worker> surnamesOfWorkers = SurnamesOfWorkers(workers);

            foreach (Worker w in surnamesOfWorkers)
            {
                Worker worker = SinglePartMaker(w.Surname, workers, parts);

                if (worker != null)
                {
                    singlePartsWorkers.Add(worker);
                }
            }

            return singlePartsWorkers;
        }

        /// <summary>
        /// Method that finds workers who make only one type of parts
        /// </summary>
        /// <param name="surname">Worker's surname</param>
        /// <param name="workers">List of workers</param>
        /// <param name="parts">List of parts</param>
        /// <returns>A worker</returns>
        private static Worker SinglePartMaker(string surname, LinkList<Worker>
workers, LinkList<Part> parts)
        {
            LinkList<Worker> onePartMaker = new LinkList<Worker>();
            Worker worker = new Worker();
            int totalNumberOfParts = 0;
            decimal moneyCount = 0;
            int howManyParts = 0;

            foreach (Worker w in workers)
            {
                if (w.Surname == surname && !onePartMaker.Contains(w.Code))
                {
                    onePartMaker.Add(w);
                    howManyParts++;
                }

                if (w.Surname == surname)
                {
                    decimal money = 0;

                    totalNumberOfParts = totalNumberOfParts + w.VntCount;

                    foreach (Part p in parts)
                    {
                        if (w.Code == p.Code)
                        {
                            money = w.VntCount * p.Price;
                        }
                    }

                    moneyCount = moneyCount + money;

                    worker = new Worker(DateTime.MinValue, w.Surname, w.Name,
w.Code, 0, 0, totalNumberOfParts, moneyCount);
                }
            }
        }

```

```

        if (howManyParts == 1)
        {
            return worker;
        }

        return null;
    }

    /// <summary>
    /// Method that formats a list of workers defined by attributes
    /// </summary>
    /// <param name="S">Number of parts</param>
    /// <param name="K">Value of price</param>
    /// <param name="workers">List of workers</param>
    /// <param name="parts">List of parts</param>
    /// <returns>A list</returns>
    public static LinkList<Worker> ListByAttributes(int S, decimal K,
LinkList<Worker> workers, LinkList<Part> parts)
    {
        LinkList<Worker> listOfWorkers = new LinkList<Worker>();

        foreach (Worker w in workers)
        {
            if (w.VntCount > S)
            {
                foreach (Part p in parts)
                {
                    if (w.Code == p.Code && p.Price < K)
                    {
                        listOfWorkers.Add(w);
                    }
                }
            }
        }

        return listOfWorkers;
    }

    /// <summary>
    /// Method that finds out if a string type variable has a specific symbol
and returns true or false
    /// </summary>
    /// <param name="text">String type variable</param>
    /// <returns>If it is true or false</returns>
    public static bool ContainsLetter(string text)
    {
        string alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzáčěĭšůžȺĈĖĬŠŮŽ";
        foreach (char a in alphabet)
        {
            if (text.Contains(a))
            {
                return true;
            }
        }
        return false;
    }
}

```



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

namespace Lab3
{
    /// <summary>
    /// Web form extention
    /// </summary>
    public partial class Factory : System.Web.UI.Page
    {
        /// <summary>
        /// Method that forms Table1
        /// </summary>
        /// <param name="workers">List of wrokers</param>
        protected void PrintTable1(LinkList<Worker> workers)
        {
            TableRow row = new TableRow();

            TableCell cell1 = new TableCell();
            cell1.Text = "<b>Data</b>";
            row.Cells.Add(cell1);

            TableCell cell2 = new TableCell();
            cell2.Text = "<b>Pavardė</b>";
            row.Cells.Add(cell2);

            TableCell cell3 = new TableCell();
            cell3.Text = "<b>Vardas</b>";
            row.Cells.Add(cell3);

            TableCell cell4 = new TableCell();
            cell4.Text = "<b>Kodas</b>";
            row.Cells.Add(cell4);

            TableCell cell5 = new TableCell();
            cell5.Text = "<b>Detalių skaičius</b>";
            row.Cells.Add(cell5);

            Table1.Rows.Add(row);

            foreach (Worker w in workers)
            {
                TableRow newRow = new TableRow();

                TableCell newCell1 = new TableCell();
                newCell1.Text = string.Format("{0:yyyy-MM-dd}", w.Date);
                newRow.Cells.Add(newCell1);

                TableCell newCell2 = new TableCell();
                newCell2.Text = string.Format("{0}", w.Surname);
                newRow.Cells.Add(newCell2);

                TableCell newCell3 = new TableCell();
                newCell3.Text = string.Format("{0}", w.Name);
                newRow.Cells.Add(newCell3);

                TableCell newCell4 = new TableCell();
                newCell4.Text = string.Format("{0}", w.Code);
                newRow.Cells.Add(newCell4);
            }
        }
    }
}

```

```

        TableCell newCell15 = new TableCell();
        newCell15.Text = string.Format("{0}", w.VntCount);
        newRow.Cells.Add(newCell15);

        Table1.Rows.Add(newRow);
    }

}

/// <summary>
/// Method that forms Table2
/// </summary>
/// <param name="parts">List of parts</param>
protected void PrintTable2(LinkList<Part> parts)
{
    TableRow row = new TableRow();

    TableCell cell1 = new TableCell();
    cell1.Text = "<b>Kodas</b>";
    row.Cells.Add(cell1);

    TableCell cell2 = new TableCell();
    cell2.Text = "<b>Pavadinimas</b>";
    row.Cells.Add(cell2);

    TableCell cell3 = new TableCell();
    cell3.Text = "<b>Kaina</b>";
    row.Cells.Add(cell3);

    Table2.Rows.Add(row);

    foreach (Part p in parts)
    {
        TableRow newRow = new TableRow();

        TableCell newCell1 = new TableCell();
        newCell1.Text = string.Format("{0}", p.Code);
        newRow.Cells.Add(newCell1);

        TableCell newCell2 = new TableCell();
        newCell2.Text = string.Format("{0}", p.Name);
        newRow.Cells.Add(newCell2);

        TableCell newCell3 = new TableCell();
        newCell3.Text = string.Format("{0}", p.Price);
        newRow.Cells.Add(newCell3);

        Table2.Rows.Add(newRow);
    }
}

/// <summary>
/// Method that forms Table3
/// </summary>
/// <param name="richestWorker">A single worker</param>
protected void PrintRichestWorker(Worker richestWorker)
{
    TableRow row = new TableRow();

    TableCell cell1 = new TableCell();
    cell1.Text = "<b>Pavardė</b>";
    row.Cells.Add(cell1);

    TableCell cell2 = new TableCell();

```

```

cell12.Text = "<b>Dirbtų dienų skaičius</b>";
row.Cells.Add(cell12);

TableCell cell13 = new TableCell();
cell13.Text = "<b>Pagamintų detalių skaičius</b>";
row.Cells.Add(cell13);

TableCell cell14 = new TableCell();
cell14.Text = "<b>Uždirbti pinigai</b>";
row.Cells.Add(cell14);

Table3.Rows.Add(row);

if (richestWorker.Surname != null)
{
    TableRow newRow = new TableRow();

    TableCell cell15 = new TableCell();
    cell15.Text = string.Format("{0}", richestWorker.Surname);
    newRow.Cells.Add(cell15);

    TableCell cell16 = new TableCell();
    cell16.Text = string.Format("{0}", richestWorker.DaysWorked);
    newRow.Cells.Add(cell16);

    TableCell cell17 = new TableCell();
    cell17.Text = string.Format("{0}", richestWorker.PartsCount);
    newRow.Cells.Add(cell17);

    TableCell cell18 = new TableCell();
    cell18.Text = string.Format("{0}", richestWorker.MoneyCount);
    newRow.Cells.Add(cell18);

    Table3.Rows.Add(newRow);
}

}

/// <summary>
/// Method that forms Table4
/// </summary>
/// <param name="workers">List of workers</param>
protected void PrintSinglePartMakers(LinkList<Worker> workers)
{
    TableRow row = new TableRow();

    TableCell cell11 = new TableCell();
    cell11.Text = "<b>Pavardė</b>";
    row.Cells.Add(cell11);

    TableCell cell12 = new TableCell();
    cell12.Text = "<b>Vardas</b>";
    row.Cells.Add(cell12);

    TableCell cell13 = new TableCell();
    cell13.Text = "<b>Detalės kodas</b>";
    row.Cells.Add(cell13);

    TableCell cell14 = new TableCell();
    cell14.Text = "<b>Skaičius vienodų detalių</b>";
    row.Cells.Add(cell14);

    TableCell cell15 = new TableCell();
    cell15.Text = "<b>Uždirbtų pinigų suma</b>";

```

```

        row.Cells.Add(cell5);

        Table4.Rows.Add(row);

        foreach (Worker w in workers)
        {
            TableRow newRow = new TableRow();

            TableCell newCell1 = new TableCell();
            newCell1.Text = string.Format("{0}", w.Surname);
            newRow.Cells.Add(newCell1);

            TableCell newCell2 = new TableCell();
            newCell2.Text = string.Format("{0}", w.Name);
            newRow.Cells.Add(newCell2);

            TableCell newCell3 = new TableCell();
            newCell3.Text = string.Format("{0}", w.Code);
            newRow.Cells.Add(newCell3);

            TableCell newCell4 = new TableCell();
            newCell4.Text = string.Format("{0}", w.PartsCount);
            newRow.Cells.Add(newCell4);

            TableCell newCell5 = new TableCell();
            newCell5.Text = string.Format("{0}", w.MoneyCount);
            newRow.Cells.Add(newCell5);

            Table4.Rows.Add(newRow);
        }
    }

    /// <summary>
    /// Method that forms Table5
    /// </summary>
    /// <param name="workers">List of workers</param>
    protected void PrintTable5(LinkList<Worker> workers)
    {
        TableRow row = new TableRow();

        TableCell cell1 = new TableCell();
        cell1.Text = "<b>Data</b>";
        row.Cells.Add(cell1);

        TableCell cell2 = new TableCell();
        cell2.Text = "<b>Pavardė</b>";
        row.Cells.Add(cell2);

        TableCell cell3 = new TableCell();
        cell3.Text = "<b>Vardas</b>";
        row.Cells.Add(cell3);

        TableCell cell4 = new TableCell();
        cell4.Text = "<b>Kodas</b>";
        row.Cells.Add(cell4);

        TableCell cell5 = new TableCell();
        cell5.Text = "<b>Detalių skaičius</b>";
        row.Cells.Add(cell5);

        Table5.Rows.Add(row);

        foreach (Worker w in workers)
        {

```

```

        TableRow newRow = new TableRow();

        TableCell newCell1 = new TableCell();
        newCell1.Text = string.Format("{0:yyyy-MM-dd}", w.Date);
        newRow.Cells.Add(newCell1);

        TableCell newCell2 = new TableCell();
        newCell2.Text = string.Format("{0}", w.Surname);
        newRow.Cells.Add(newCell2);

        TableCell newCell3 = new TableCell();
        newCell3.Text = string.Format("{0}", w.Name);
        newRow.Cells.Add(newCell3);

        TableCell newCell4 = new TableCell();
        newCell4.Text = string.Format("{0}", w.Code);
        newRow.Cells.Add(newCell4);

        TableCell newCell5 = new TableCell();
        newCell5.Text = string.Format("{0}", w.VntCount);
        newRow.Cells.Add(newCell5);

        Table5.Rows.Add(newRow);
    }
}
}
}

```

StyleSheet1.css

```

div {
    background-color :mistyrose;
}

.Table{
    background-color: yellow;
}

.BoldText{
    font-weight: bold;
}

body{
    background-color: cadetblue;
}

.TextSize{
    font-size: 18px;
}

```

LinkListTests.cs

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Lab3;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab3.Tests
{
    [TestClass()]
    public class LinkListTests

```

```

{
    [TestMethod()]
    public void LinkList_EmptyContainerFormsAList_ReturnFalse()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Assert.IsFalse(list.ListExist());
    }

    [TestMethod()]
    public void LinkList_EmptyContainerContainsData_ReturnFalse()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Worker worker = new Worker();
        Assert.IsFalse(list.Contains(worker));
    }

    [TestMethod()]
    public void Add_PutsOneWorkerInTheList_ReturnTrue()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Worker worker = new Worker(DateTime.MinValue, null, "Vytautas", null,
0, 0, 0, 0);
        list.Add(worker);

        Assert.IsTrue(list.Contains(worker));
    }

    [TestMethod()]
    public void Add_FormsAList_ReturnTrue()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Worker worker = new Worker(DateTime.MinValue, null, "Vytautas", null,
0, 0, 0, 0);
        list.Add(worker);

        Assert.IsTrue(list.ListExist());
    }

    [TestMethod()]
    public void Add_PutsTwoWorkersInTheList_ReturnTrue()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Worker worker1 = new Worker(DateTime.MinValue, null, "Vytautas", null,
0, 0, 0, 0);
        Worker worker2 = new Worker(DateTime.MinValue, null, "Nojus", null, 0,
0, 0, 0);

        list.Add(worker1);
        list.Add(worker2);

        Assert.IsTrue(list.Contains(worker1));
        Assert.IsTrue(list.Contains(worker2));
    }

    [TestMethod()]
    public void ListExist_EmptyListHasData_ReturnFalse()
    {
        LinkList<Worker> list = new LinkList<Worker>();

        Assert.IsFalse(list.ListExist());
    }
}

```

```

    }

    [TestMethod()]
    public void ListExist_WhenListHasData_ReturnTrue()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker = new Worker();
        list.Add(worker);

        Assert.IsTrue(list.ListExist());
    }

    [TestMethod()]
    public void ListExist_TwoListsHaveData_ReturnTrue()
    {
        LinkedList<Worker> list1 = new LinkedList<Worker>();
        LinkedList<Worker> list2 = new LinkedList<Worker>();

        Worker worker = new Worker();

        list1.Add(worker);
        list2.Add(worker);

        Assert.IsTrue(list1.ListExist());
        Assert.IsTrue(list2.ListExist());
    }

    [TestMethod()]
    public void Contains_NoInformationAboutGivenData_ReturnTrue()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = new Worker(DateTime.Parse("2001-07-01"), "Jonaitis",
"Rokas", "ifs-0", 5, 1, 5, 6);

        list.Add(worker2);

        Assert.IsTrue(!list.Contains(worker1));
    }

    [TestMethod()]
    public void Contains_GivenDataAboutOneWorker_ReturnTrue()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = new Worker(DateTime.Parse("2001-07-01"), "Jonaitis",
"Rokas", "ifs-0", 5, 1, 5, 6);

        list.Add(worker2);

        Assert.IsTrue(list.Contains(worker2));
    }

    [TestMethod()]
    public void Contains_AllTheGivenData_ReturnTrue()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = new Worker(DateTime.Parse("2001-07-01"), "Jonaitis",
"Rokas", "ifs-0", 5, 1, 5, 6);

        list.Add(worker1);
    }

```

```

        list.Add(worker2);

        Assert.IsTrue(list.Contains(worker1));
        Assert.IsTrue(list.Contains(worker2));
    }

    [TestMethod()]
    public void Sort_TheSameInformation_ReturnFirstAddedWorkerIsFirst()
    {
        var list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = worker1;

        list.Add(worker1);
        list.Add(worker2);

        Worker w = new Worker();
        list.Sort(out w);

        Assert.AreEqual(worker1.Surname, w.Surname);
    }

    [TestMethod()]
    public void Sort_DifferentInformation_ReturnSecondAddedWorkerIsFirst()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = new Worker(DateTime.Parse("2000-07-01"),
"Aaulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);

        list.Add(worker1);
        list.Add(worker2);

        Worker w = new Worker();
        list.Sort(out w);
        Assert.AreEqual(worker2.Surname, w.Surname);
    }

    [TestMethod()]
    public void Sort_DifferentInformation_ReturnFirstAddedWorkerIsNotFirst()
    {
        LinkedList<Worker> list = new LinkedList<Worker>();
        Worker worker1 = new Worker(DateTime.Parse("2000-07-01"),
"Paulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);
        Worker worker2 = new Worker(DateTime.Parse("2000-07-01"),
"Aaulavicius", "Rokas", "ifs-0", 5, 1, 5, 6);

        list.Add(worker1);
        list.Add(worker2);

        Worker w = new Worker();
        list.Sort(out w);
        Assert.AreNotEqual(worker1.Surname, w.Surname);
    }
}
}

```

Factory.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Factory.aspx.cs"
Inherits="Lab3.Factory" %>

<!DOCTYPE html>

```



```

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="StyleSheet1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
        <div class="TextSize">
            Pradinių duomenų failas apie darbininkus:<br />
            <asp:FileUpload ID="FileUpload1" runat="server" CssClass="BoldText" />
            <br />
            <br />
            Pradinių duomenų failas apie dalis:<br />
            <asp:FileUpload ID="FileUpload2" runat="server" CssClass="BoldText" />
            <br />
            <br />
            &nbsp;<asp:Button ID="Button1" runat="server" Text="Užkrauti pradinis duomenis"
            OnClick="Button1_Click" CssClass="BoldText" />
            <br />
            <br />
            Pradiniai darbininkų duomenys:<br />
            <asp:Table ID="Table1" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Pradiniai dalių duomenys:<asp:Table ID="Table2" runat="server"
            BorderStyle="Solid" BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            <asp:Button ID="Button2" runat="server" OnClick="Button2_Click"
            Text="Skaičiuoti" CssClass="BoldText" />
            <br />
            <br />
            Daugiausiai uždirbęs darbininkas:<br />
            <asp:Table ID="Table3" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:<br />
            <asp:Table ID="Table4" runat="server" BorderStyle="Solid"
            BorderWidth="1px" GridLines="Both" CssClass="Table">
            </asp:Table>
            <br />
            Įveskite pagamintų vienetų skaičių S:
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <br />
            <br />
            Įveskite įkainio skaičių K:
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <br />
            <br />
            Sudarytas naujas duomenų rinkinys pagal požymius S ir K:<asp:Table
            ID="Table5" runat="server" BorderStyle="Solid" BorderWidth="1px" GridLines="Both"
            CssClass="Table">
            </asp:Table>
            <br />
        </div>
    </form>
</body>
</html>

```

Factory.aspx.cs

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
//Main function of this program is to do all kinds of calculations with given data
of workers and different parts

//Vytenis Kriščiūnas
namespace Lab3
{
    /// <summary>
    /// Web form
    /// </summary>
    public partial class Factory : System.Web.UI.Page
    {
        /// <summary>
        /// List of parts
        /// </summary>
        LinkedList<Part> parts;

        /// <summary>
        /// List of workers
        /// </summary>
        LinkedList<Worker> workers;

        /// <summary>
        /// The richest worker
        /// </summary>
        Worker richestWorker;

        /// <summary>
        /// List of workers who allways make the same parts
        /// </summary>
        LinkedList<Worker> singlePartMakers;

        /// <summary>
        /// List formed by given attributes
        /// </summary>
        LinkedList<Worker> listByAttributes;

        /// <summary>
        /// Prints information to the screen when page loads
        /// </summary>
        /// <param name="sender">An object variable</param>
        /// <param name="e">EventArgs variable</param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (parts == null)
            {
                parts = new LinkedList<Part>();
            }
            if (workers == null)
            {
                workers = new LinkedList<Worker>();
            }
            if (richestWorker == null)
            {
                richestWorker = new Worker();
            }
            if (singlePartMakers == null)
            {
                singlePartMakers = new LinkedList<Worker>();
            }
        }
    }
}

```

```

        if (listByAttributes == null)
        {
            listByAttributes = new LinkedList<Worker>();
        }

        parts = Session["parts"] as LinkedList<Part>;
        workers = Session["workers"] as LinkedList<Worker>;

        if (workers != null && parts != null)
        {
            PrintTable1(workers);
            PrintTable2(parts);
        }
    }

    /// <summary>
    /// Prints and reads given information when the first button is clicked
    /// </summary>
    /// <param name="sender">An object variable</param>
    /// <param name="e">EventArgs variable</param>
    protected void Button1_Click(object sender, EventArgs e)
    {
        Session.Remove("parts");
        Session.Remove("workers");
        for (int i = Table1.Rows.Count - 1; i >= 0; i--)
        {
            Table1.Rows.RemoveAt(i);
        }
        for (int i = Table2.Rows.Count - 1; i >= 0; i--)
        {
            Table2.Rows.RemoveAt(i);
        }

        string Fr = Server.MapPath("App_Data/GivenData.txt");
        File.Delete(Fr);

        if (FileUpload1.HasFile && FileUpload2.HasFile)
        {
            string Fd1 = Server.MapPath("App_Data/U10a.txt"); // workers
            string Fd2 = Server.MapPath("App_Data/U10b.txt"); // parts

            if (FileUpload1.FileName.EndsWith("a.txt") &&
                FileUpload2.FileName.EndsWith("b.txt"))
            {
                parts = InOutUtils.ReadInfo2(Fd2);
                workers = InOutUtils.ReadInfo1(Fd1);

                PrintTable1(workers);
                PrintTable2(parts);

                InOutUtils.PrintGivenData(Fr, workers, "");
                InOutUtils.PrintGivenData(Fr, parts);

                Session["parts"] = parts;
                Session["workers"] = workers;
            }
        }
    }

    /// <summary>

```

```

    /// Calculates needed information and prints it when the second button is
    clicked
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Button2_Click(object sender, EventArgs e)
    {
        string Fr = Server.MapPath("App_Data/Rezults.txt");
        File.Delete(Fr);
        int S = 0;
        decimal K = 0;

        if (TextBox1.Text != "" && TextBox2.Text != "" &&
TaskUtils.ContainsLetter(TextBox1.Text) == false &&
TaskUtils.ContainsLetter(TextBox2.Text) == false)
        {
            S = int.Parse(TextBox1.Text);
            K = decimal.Parse(TextBox2.Text);
        }

        if (workers != null && parts != null)
        {
            richestWorker = TaskUtils.RichestWorker(workers, parts);

            singlePartMakers = TaskUtils.SinglePartMakers(workers, parts);
            Worker w = new Worker();
            singlePartMakers.Sort(out w);
        }

        PrintRichestWorker(richestWorker);

        PrintSinglePartMakers(singlePartMakers);

        if (workers != null && parts != null && S != 0 && K != 0)
        {
            listByAttributes = TaskUtils.ListByAttributes(S, K, workers,
parts);
            Worker w = new Worker();
            listByAttributes.Sort(out w);
        }

        PrintTable5(listByAttributes);

        if (richestWorker.Surname != null)
        {
            InOutUtils.PrintRichestWorker(Fr, richestWorker);
        }

        if (singlePartMakers.ListExist() == true)
        {
            InOutUtils.PrintSinglePartMakers(Fr, singlePartMakers);
        }

        if (listByAttributes.ListExist() == true)
        {
            InOutUtils.PrintGivenData(Fr, listByAttributes,
string.Format("Sudarytas naujas duomenų rinkinys pagal požymius S ir K:"));
        }

        if (richestWorker.Surname == null && singlePartMakers.ListExist() ==
false && listByAttributes.ListExist() == false)
        {
            File.Delete(Fr);
        }
    }
}

```

```

    }
  }
}
}

```

3.7. Pradiniai duomenys ir rezultatai

Duomenys nr. 1

U10a.txt

```

2021-07-21 Pavardenis Vardenis IOp5 5
2021-07-21 Pavardenis Vardenis LkD5 4
2021-07-21 Kazkoks Kazkas LkD5 4
2021-06-21 Kazkoks Kazkas LkD5 4
2021-06-21 Kazkoks Kazkas LkD5 7
2022-06-21 Aetraitis Petras Klj 7

```

U10b.txt

```

IOp5 metal 3
LkD5 plast 6.3
Bpm6 kalp 2
Klj milst 9.3

```

GivenData.txt

```

-----
-----
| 2021-07-21          | Pavardenis          | Vardenis          | IOp5
|                    | 5 |
| 2021-07-21          | Pavardenis          | Vardenis          | LkD5
|                    | 4 |
| 2021-07-21          | Kazkoks             | Kazkas            | LkD5
|                    | 4 |
| 2021-06-21          | Kazkoks             | Kazkas            | LkD5
|                    | 4 |
| 2021-06-21          | Kazkoks             | Kazkas            | LkD5
|                    | 7 |
| 2022-06-21          | Aetraitis           | Petras            | Klj
|                    | 7 |
-----
-----

```

```

-----
-----
| IOp5                | metal              |                    | 3 |
| LkD5                | plast              |                    | 6.3 |
| Bpm6                | kalp               |                    | 2 |
| Klj                 | milst              |                    | 9.3 |
-----
-----

```

Rezults.txt

Daugiausiai uždirbęs darbininkas:

```

-----
-----
| Kazkoks              |                    | 2 |                    | 15 |
94.5 |
-----
-----

```

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

```
-----  
| Aetraitis          | Petras          | Klj          |  
7 |                  65.1 |  
| Kazkoks            | Kazkas          | Lkd5         |  
15 |                  94.5 |  
-----  
-----
```

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

```
-----  
| 2022-06-21         | Aetraitis       | Petras       | Klj  
|                   7 |  
| 2021-07-21         | Kazkoks         | Kazkas       | Lkd5  
|                   4 |  
| 2021-06-21         | Kazkoks         | Kazkas       | Lkd5  
|                   4 |  
| 2021-06-21         | Kazkoks         | Kazkas       | Lkd5  
|                   7 |  
| 2021-07-21         | Pavardenis      | Vardenis     | Lkd5  
|                   4 |  
| 2021-07-21         | Pavardenis      | Vardenis     | IOp5  
|                   5 |  
-----  
-----
```

Vartotojo sąsaja

Pradinių duomenų failas apie darbininkus:
 No file chosen

Pradinių duomenų failas apie dalis:
 No file chosen

Pradiniai darbininkų duomenys:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-07-21	Pavardenis	Vardenis	IOP5	5
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	7
2022-06-21	Aetraitis	Petras	Klj	7

Pradiniai dalių duomenys:

Kodas	Pavadinimas	Kaina
IOP5	metal	3
LkD5	plast	6.3
Bpm6	kalp	2
Klj	milst	9.3

Daugiausiai uždirbęs darbininkas:

Pavardė	Dirbtų dienų skaičius	Pagamintų detalių skaičius	Uždirbti pinigai
Kazkoks	2	15	94.5

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Pavardė	Vardas	Detalės kodas	Skaičius vienodų detalių	Uždirbtų pinigų suma
Aetraitis	Petras	Klj	7	65.1
Kazkoks	Kazkas	LkD5	15	94.5

Įveskite pagamintų vienetų skaičių S:

Įveskite įkainio skaičių K:

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2022-06-21	Aetraitis	Petras	Klj	7
2021-07-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	4
2021-06-21	Kazkoks	Kazkas	LkD5	7
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Pavardenis	Vardenis	IOP5	5

Duomenys nr. 2

U10a.txt

```
2021-07-21 Pavardenis Vardenis IOP5 5
2021-07-21 Pavardenis Vardenis LkD5 4
2021-07-21 Bazkoks Lazkas Bpm6 4
2021-06-21 Bazkoks Lazkas LkD5 4
2021-06-21 Bazkoks Lazkas LkD5 7
2022-06-21 Aetraitis Petras Klj 7
2022-06-28 Valancius Justas IOP5 8
```

U10b.txt

```
IOP5 metal 3
LkD5 plast 2
Bpm6 kalp 2
Klj milst 9.3
```

GivenData.txt

```
-----
| 2021-07-21          | Pavardenis          | Vardenis          | IOP5
|                      | 5 |
| 2021-07-21          | Pavardenis          | Vardenis          | LkD5
|                      | 4 |
```

2021-07-21		Bazkoks		Lazkas		Bpm6
	4					
2021-06-21		Bazkoks		Lazkas		LkD5
	4					
2021-06-21		Bazkoks		Lazkas		LkD5
	7					
2022-06-21		Aetraitis		Petras		Klj
	7					
2022-06-28		Valancius		Justas		IOp5
	8					

IOp5	metal		3	
LkD5	plast		2	
Bpm6	kalp		2	
Klj	milst		9.3	

Rezults.txt

Daugiausiai uždirbęs darbininkas:

Aetraitis		1		7
65.1				

Tik vieno pavadinimo detales gaminusių darbininkų sąrašas:

Aetraitis		Petras		Klj	
7		65.1			
Valancius		Justas		IOp5	
8		24			

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

2021-06-21		Bazkoks		Lazkas		LkD5
	7					
2021-07-21		Pavardenis		Vardenis		IOp5
	5					
2022-06-28		Valancius		Justas		IOp5
	8					

Vartotojo sąsaja

Pradinių duomenų failas apie darbininkus:
 No file chosen

Pradinių duomenų failas apie dalis:
 No file chosen

Pradiniai darbininkų duomenys:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-07-21	Pavardenis	Vardenis	IOP5	5
2021-07-21	Pavardenis	Vardenis	LkD5	4
2021-07-21	Bazkoks	Lazkas	Bpm6	4
2021-06-21	Bazkoks	Lazkas	LkD5	4
2021-06-21	Bazkoks	Lazkas	LkD5	7
2022-06-21	Aetraitis	Petras	Klj	7
2022-06-28	Valancius	Justas	IOP5	8

Pradiniai dalių duomenys:

Kodas	Pavadinimas	Kaina
IOP5	metal	3
LkD5	plast	2
Bpm6	kalp	2
Klj	milst	9.3

Daugiausiai uždirbęs darbininkas:

Pavardė	Dirbtų dienų skaičius	Pagamintų detalių skaičius	Uždirbti pinigai
Aetraitis	1	7	65.1

Tik vieno pavadinimo detales gaminsius darbininkų sąrašas:

Pavardė	Vardas	Detalės kodas	Skaičius vienodų detalių	Uždirbtų pinigų suma
Aetraitis	Petras	Klj	7	65.1
Valancius	Justas	IOP5	8	24

Įveskite pagamintų vienetų skaičių S:

Įveskite įkainio skaičių K:

Sudarytas naujas duomenų rinkinys pagal požymius S ir K:

Data	Pavardė	Vardas	Kodas	Detalių skaičius
2021-06-21	Bazkoks	Lazkas	LkD5	7
2021-07-21	Pavardenis	Vardenis	IOP5	5
2022-06-28	Valancius	Justas	IOP5	8

3.8. Dėstytojo pastabos

Ginimo testas: 1.0

Programa: 5.5

Ataskaita: 1.0

Bendras įvertinimas: 7

Dėstytojo komentarai:

Testavimas yra, bet jo nepavyko pademonstruoti;

Ataskaitoje nėra testavimo protokolo.

4. Polimorfizmas ir išimčių valdymas (L4)

4.1. Darbo užduotis

U4_10. Buitinės technikos parduotuvė. Turite informaciją apie skirtingose buitinės technikos parduotuvėse esančius šaldytuvus, mikrobangų krosneles ir elektrinius virdulius. Pirmoje eilutėje yra pavadinimas, antroje – adresas, trečioje – telefonas. Sukurkite abstrakčią klasę „Device“ (savybės - gamintojas, modelis, energijos klasė, spalva, kaina), kurią paveldės „Fridge“ (savybės - talpa, montavimo tipas, požymis „turi šaldiklį“, aukštis, plotis, gylis), „Oven“ (savybės – galingumas, programų skaičius) ir „Kettle“ (savybės – galia, tūris).

- Suskaičiuokite, kiek skirtingų „Siemens“ šaldytuvų, mikrobangų krosnelių ir virdulių modelių siūlo kiekviena parduotuvė, rezultatą atspausdinkite ekrane.

- Sudarykite dešimties pigiausių pastatomų šaldytuvų, kurių talpa 80 litrų ar didesnė, sąrašą. Ekrane atspausdinkite šaldytuvo gamintoją, modelį, talpą ir kainą. P175B123 Objektinis programavimas 2 4

- Ar yra tokių buitinių prietaisų, kuriuos galima įsigyti tik vienoje parduotuvėje? Atspausdinkite tokių prietaisų sąrašą faile „TikTen.csv“.

- Sudarykite ir surikiuokite brangių buitinių prietaisų sąrašą, pateikdami pilną informaciją apie juos. Šaldytuvas yra brangus, jei jo kaina viršija 1000€. Mikrobangų krosnelė yra brangi, jei jos kaina viršija 500€. Virdulys yra brangus, jei jo kaina viršija 50€. Šaldytuvus rikiuokite pagal aukštį, mikrobangų krosneles – pagal galingumą, o virdulius – pagal galią. Rezultatus įrašykite į failą „Brangus.csv“.

4.2. Grafinės vartotojo sąsajos schema

Užkraukite bent vieną pradinį duomenų failą iš aplanko, kad užkrautumėte likusius:

FileUpload1

[Label1] Label1

Button1

Table1

Skaičius „Siemens“ šaldytuvų, mikrobangų krosnelių ir virdulių modelių, kurį siūlo kiekviena parduotuvė:

Table2

Dešimt pigiausių pastatomų šaldytuvų, kurių talpa yra 80 litrų ar didesnė:

Table3

Išspausdinama, ar yra tokių buitinių prietaisų, kuriuos galima įsigyti tik vienoje parduotuvėje:

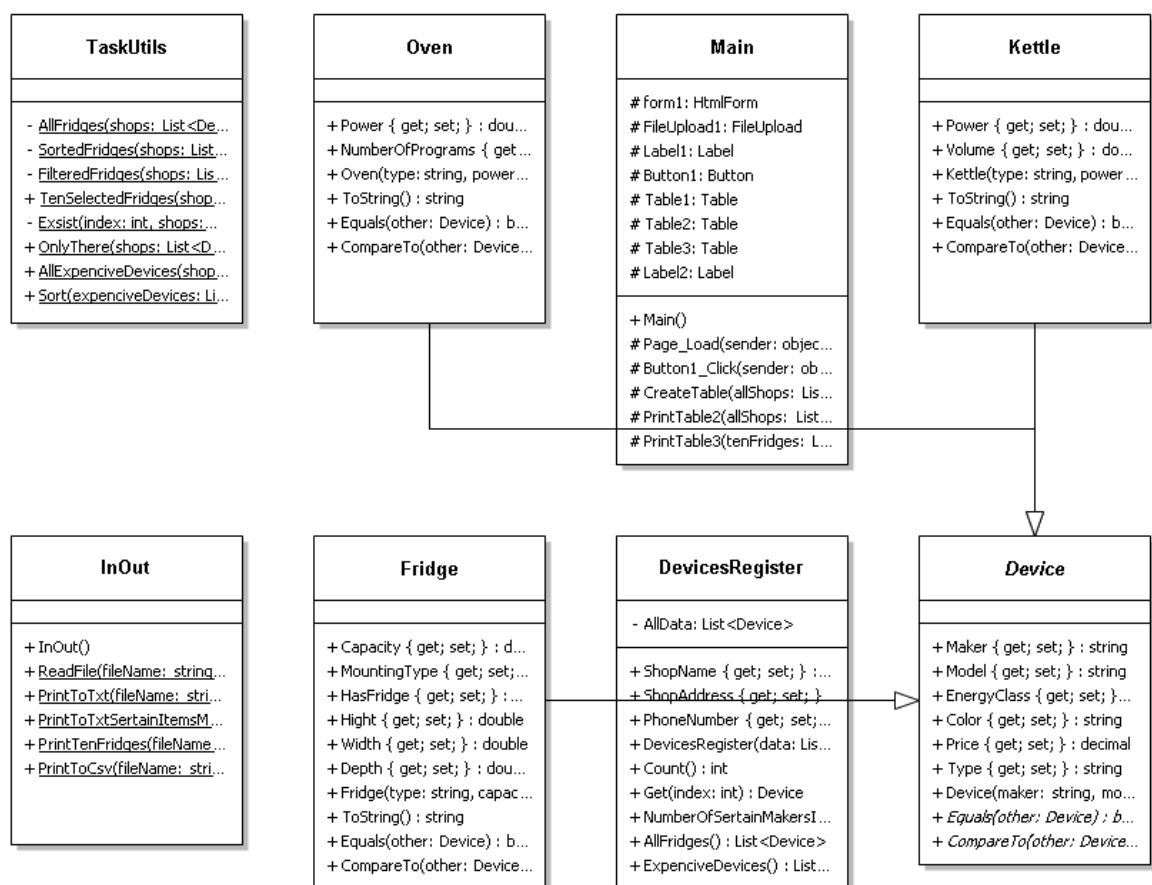
[Label2] Label2

4.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Div	Class	TextSize
Body	Backgroud-color	Cadetblue
FileUpload1	CssClass	BoldText

Label1	CssClass	TextSize
Button1	Text	Užkrauti duomenis
Button1	CssClass	BoldText
Table1	BorderWidth	1px
Table1	GridLines	Both
Table1	CssClass	Table
Table1	ForeColor	Black
Table2	BorderWidth	1px
Table2	GridLines	Both
Table2	CssClass	Table
Table2	ForeColor	Black
Table3	BorderWidth	1px
Table3	GridLines	Both
Table3	CssClass	Table
Table3	ForeColor	Black
Label2	CssClass	TextSize

4.4. Klasių diagrama



4.5. Programos vartotojo vadovas

Kai atidarome programą, matome vieną vietą skirtą įkeltis nors vienam duomenų failui iš App_Data/Data/ aplanko. Pasirinkus vieną duomenų failą, reikia spausti mygtuką: „Užkrauti duomenis“. Mygtuką paspaudus yra sudaromos trys lentelės: pradinį duomenų ir dvi rezultatų lentelės. Taip pat pradiniai duomenys ir rezultatai yra

išspausdinami į Rezults.txt failą ir likę rezultatai į TikTen.csv, Brangus.csv failus. Jeigu pradiniai duomenys nėra užkraunami, tai ir rezultatai nėra spausdinami į ekraną ir rezultatų failus.

4.6. Programos tekstas

InOut.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;

namespace Lab_4
{
    /// <summary>
    /// Class that reads and prints information
    /// </summary>
    public class InOut
    {
        /// <summary>
        /// Method that read information and forms a list
        /// </summary>
        /// <param name="fileName">Name of a data file</param>
        /// <returns>Formatted list</returns>
        public static List<DevicesRegister> ReadFile(string fileName)
        {
            List<DevicesRegister> LibraryOfShops = new List<DevicesRegister>();

            foreach (string txtName in Directory.GetFiles(fileName, "*.txt"))
            {
                List<Device> Shop = new List<Device>();
                string[] lines = File.ReadAllLines(txtName);
                string shopName = lines[0];
                string shopAddress = lines[1];
                string phoneNumber = lines[2];
                for (int i = 3; i < lines.Length; i++)
                {
                    string[] Values = lines[i].Split(';');
                    string type = Values[0];
                    string maker = Values[1];
                    string model = Values[2];
                    string energyClass = Values[3];
                    string color = Values[4];
                    decimal price = decimal.Parse(Values[5]);

                    switch (type)
                    {
                        case "Fridge":
                            double capacity = double.Parse(Values[6]);
                            string mountingType = Values[7];
                            string has = Values[8];
                            bool hasFridge = false;
                            if (has == "turi šaldiklį")
                            {
                                hasFridge = true;
                            }
                            double hight = double.Parse(Values[9]);
                            double width = double.Parse(Values[10]);
                            double depth = double.Parse(Values[11]);
                            Fridge fridge = new Fridge(type, capacity,
                                mountingType, hasFridge, hight, width, depth, maker, model, energyClass, color,
                                price);

                                Shop.Add(fridge);
                        }
                    }
                }
            }
        }
    }
}
```

```

        break;

        case "Kettle":
            double power = double.Parse(Values[6]);
            double volume = double.Parse(Values[7]);
            Kettle kettle = new Kettle(type, power, volume, maker,
model, energyClass, color, price);
            Shop.Add(kettle);
            break;

        case "Oven":
            power = double.Parse(Values[6]);
            int numberOfPrograms = int.Parse(Values[7]);
            Oven oven = new Oven(type, power, numberOfPrograms,
maker, model, energyClass, color, price);
            Shop.Add(oven);
            break;

        default:
            break;
    }

    DevicesRegister devices = new DevicesRegister(Shop, shopName,
shopAddress, phoneNumber);
    LibraryOfShops.Add(devices);
}
return LibraryOfShops;
}

/// <summary>
/// Method that prints initial data information to .txt file
/// </summary>
/// <param name="fileName">Name of a rezult file</param>
/// <param name="allShops">List of information</param>
public static void PrintToTxt(string fileName, List<DevicesRegister>
allShops)
{
    try
    {
        using (var writer = File.CreateText(fileName))
        {
            foreach (DevicesRegister shop in allShops)
            {
                writer.WriteLine(shop.ShopName);
                writer.WriteLine(shop.ShopAddress);
                writer.WriteLine(shop.PhoneNumber);

                writer.WriteLine(new string('-', 303));
                writer.WriteLine(string.Format("| {0, 15} | {1, 15} | {2,
15} | {3, 15} | {4, 15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} |
{10, -15} | {11, -15} | {12, -15} | {13, -15} | {14, -15} | {15, -15} |", "Tipas",
"Gamintojas", "Modelis", "Energijos klasė", "Spalva", "Kaina", "Talpa", "Montavimo
tipas", "Požymis: turi šaldiklį ar ne", "Aukštis", "Plotis", "Gylis", "Galia",
"Tūris", "Galingumas", "Programų skaičius"));
                writer.WriteLine(new string('-', 303));
                for (int i = 0; i < shop.Count(); i++)
                {
                    writer.WriteLine(shop.Get(i).ToString());
                }
                writer.WriteLine(new string('-', 303));
                writer.WriteLine();
            }
        }
        writer.Close();
    }
}

```

```

    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

/// <summary>
/// Method that prints certain maker's devices to .txt file
/// </summary>
/// <param name="fileName">Name of a result file</param>
/// <param name="allShops">List of information</param>
public static void PrintToTxtCertainItemsMakers(string fileName,
List<DevicesRegister> allShops)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Skaičius „Siemens“ šaldytuvų, mikrobangu
krosnelių ir virdulių modelių, kurių siūlo kiekviena parduotuvė:");
        foreach (DevicesRegister shop in allShops)
        {
            writer.WriteLine("{0}", shop.NumberOfCertainMakersItems());
            writer.WriteLine("Parduotuvės kontaktinė informacija:");
            writer.WriteLine(shop.ShopName);
            writer.WriteLine(shop.ShopAddress);
            writer.WriteLine(shop.PhoneNumber);

            writer.WriteLine();
        }
        writer.Close();
    }
}

/// <summary>
/// Method that print ten selected fridges to .csv file
/// </summary>
/// <param name="fileName">Name of a result file</param>
/// <param name="tenFridges">List of fridges</param>
public static void PrintTenFridges(string fileName, List<Device>
tenFridges)
{
    using (var writer = File.AppendText(fileName))
    {
        writer.WriteLine("Dešimt pigiausių pastatomų šaldytuvų, kurių
talpa yra 80 litrų ar didesnė:");
        writer.WriteLine(new string('-', 74));
        writer.WriteLine("| {0, -15} | {1, -15} | {2, 15} | {3, 15} |",
"Gamintojas", "Modelis", "Talpa", "Kaina");
        writer.WriteLine(new string('-', 74));

        for (int i = 0; i < tenFridges.Count(); i++)
        {
            Device device = tenFridges[i];
            if (device is Fridge)
            {
                writer.WriteLine("| {0, -15} | {1, -15} | {2, 15} | {3,
15} |", device.Maker, (device as Fridge).MountingType, (device as
Fridge).Capacity, device.Price);
            }

        }
        writer.WriteLine(new string('-', 74));

        writer.WriteLine();
    }
}

```

```

        writer.Close();
    }

}

/// <summary>
/// Method that prints certain information by a given type to .csv file
/// </summary>
/// <param name="fileName">Name of a result file</param>
/// <param name="devices">List of devices</param>
/// <param name="type">Certain type of devices</param>
public static void PrintToCsv(string fileName, List<Device> devices,
string type)
{
    using (var writer = File.AppendText(fileName))
    {
        for (int i = 0; i < devices.Count(); i++)
        {
            if (devices[i].Type == type)
            {
                writer.WriteLine(devices[i].ToString());
            }
        }

        writer.WriteLine();
        writer.Close();
    }
}
}
}

```

Device.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{
    /// <summary>
    /// Class of device data
    /// </summary>
    public abstract class Device : IEquatable<Device>, IComparable<Device>
    {
        /// <summary>
        /// Device maker
        /// </summary>
        public string Maker { get; set; }

        /// <summary>
        /// Device model
        /// </summary>
        public string Model { get; set; }

        /// <summary>
        /// Device energy class
        /// </summary>
        public string EnergyClass { get; set; }

        /// <summary>

```

```

    /// Device color
    /// </summary>
    public string Color { get; set; }

    /// <summary>
    /// Device price
    /// </summary>
    public decimal Price { get; set; }

    /// <summary>
    /// Device type
    /// </summary>
    public string Type { get; set; }

    /// <summary>
    /// Device class constructor
    /// </summary>
    /// <param name="maker">Device maker</param>
    /// <param name="model">Device model</param>
    /// <param name="energyClass">Device energy class</param>
    /// <param name="color">Device color</param>
    /// <param name="price">Device price</param>
    /// <param name="type">Device type</param>
    public Device(string maker, string model, string energyClass, string
color, decimal price, string type)
    {
        this.Maker = maker;
        this.Model = model;
        this.EnergyClass = energyClass;
        this.Color = color;
        this.Price = price;
        this.Type = type;
    }

    /// <summary>
    /// An abstract Equals method
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>True of false value</returns>
    public abstract bool Equals(Device other);

    /// <summary>
    /// An abstract CompareTo method
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>True of false value</returns>
    public abstract int CompareTo(Device other);
}
}

```

Fridge.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{
    /// <summary>
    /// Class of fridge data
    /// </summary>
    public class Fridge : Device
    {

```



```

    /// <summary>
    /// Fridge capacity
    /// </summary>
    public double Capacity { get; set; }

    /// <summary>
    /// Fridge mounting type
    /// </summary>
    public string MountingType { get; set; }

    /// <summary>
    /// Fridge has a cooler or not
    /// </summary>
    public bool HasFridge { get; set; }

    /// <summary>
    /// Fridge hight
    /// </summary>
    public double Hight { get; set; }

    /// <summary>
    /// Fridge width
    /// </summary>
    public double Width { get; set; }

    /// <summary>
    /// Fridge depth
    /// </summary>
    public double Depth { get; set; }

    /// <summary>
    /// Fridge constructor
    /// </summary>
    /// <param name="type">Device type</param>
    /// <param name="capacity">Fridge capacity</param>
    /// <param name="mountingType">Fridge mounting type</param>
    /// <param name="hasFridge">Fridge has a cooler or not</param>
    /// <param name="hight">Fridge hight</param>
    /// <param name="width">Fridge width</param>
    /// <param name="depth">Fridge depth</param>
    /// <param name="maker">Fridge maker</param>
    /// <param name="model">Fridge model</param>
    /// <param name="energyClass">Frdige energy class</param>
    /// <param name="color">Fridge color</param>
    /// <param name="price">Fridge price</param>
    public Fridge(string type, double capacity, string mountingType, bool
hasFridge, double hight, double width, double depth, string maker, string model,
string energyClass, string color, decimal price) : base(maker, model, energyClass,
color, price, type)
    {
        this.Capacity = capacity;
        this.MountingType = mountingType;
        this.HasFridge = hasFridge;
        this.Hight = hight;
        this.Width = width;
        this.Depth = depth;
    }

    /// <summary>
    /// Method that forms a line of data in text
    /// </summary>
    /// <returns>Formatted string</returns>
    public override string ToString()
    {

```

```

        return string.Format("| {0, 15} | {1, 15} | {2, 15} | {3, 15} | {4, 15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} | {10, -15} | {11, -15} | {12, -15} | {13, -15} | {14, -15} | {15, -15} |", Type, Maker, Model, EnergyClass, Color, Price, Capacity, MountingType, HasFridge, Hight, Width, Depth, "-", "-", "-", "-");
    }

    /// <summary>
    /// Method finds out if information is equal or not
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>Ture or false value</returns>
    public override bool Equals(Device other)
    {
        return Type.CompareTo(other.Type) == 0 &&
        Maker.CompareTo(other.Maker) == 0 && Model.CompareTo(other.Model) == 0 &&
        EnergyClass.CompareTo(other.EnergyClass) == 0 && Color.CompareTo(other.Color) == 0
        && Price.CompareTo(other.Price) == 0;
    }

    /// <summary>
    /// Method that compares information by hight
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>Int type value</returns>
    public override int CompareTo(Device other)
    {
        if (other is Fridge)
        {
            return this.Hight.CompareTo((other as Fridge).Hight);
        }
        return 0;
    }
}

```

Kettle.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{
    /// <summary>
    /// Class of kettle data
    /// </summary>
    public class Kettle : Device
    {
        /// <summary>
        /// Kettle's power
        /// </summary>
        public double Power { get; set; }

        /// <summary>
        /// Kettle's volume
        /// </summary>
        public double Volume { get; set; }

        /// <summary>
        /// Kettle's class constructor
        /// </summary>
        /// <param name="type">Device type</param>
        /// <param name="power">Kettle power</param>
    }
}

```

```

    /// <param name="volume">Kettle volume</param>
    /// <param name="maker">Kettle maker</param>
    /// <param name="model">Kettle model</param>
    /// <param name="energyClass">Kettle energy class</param>
    /// <param name="color">Kettle color</param>
    /// <param name="price">Kettle price</param>
    public Kettle(string type, double power, double volume, string maker,
string model, string energyClass, string color, decimal price) : base(maker,
model, energyClass, color, price, type)
    {
        this.Power = power;
        this.Volume = volume;
    }

    /// <summary>
    /// Method that forms a line of data in text
    /// </summary>
    /// <returns>Formatted string</returns>
    public override string ToString()
    {
        return string.Format("| {0, 15} | {1, 15} | {2, 15} | {3, 15} | {4,
15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} | {10, -15} | {11, -15}
| {12, -15} | {13, -15} | {14, -15} | {15, -15} |", Type, Maker, Model,
EnergyClass, Color, Price, "-", "-", "-", "-", "-", "-", "-", Power, Volume, "-", "-
");
    }

    /// <summary>
    /// Method finds out if information is equal or not
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>True or false value</returns>
    public override bool Equals(Device other)
    {
        return Type.CompareTo(other.Type) == 0 && Maker.CompareTo(other.Maker)
== 0 && Model.CompareTo(other.Model) == 0 &&
EnergyClass.CompareTo(other.EnergyClass) == 0 && Color.CompareTo(other.Color) == 0
&& Price.CompareTo(other.Price) == 0;
    }

    /// <summary>
    /// Method that compares information by hight
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>Int type value</returns>
    public override int CompareTo(Device other)
    {
        if (other is Kettle)
        {
            return this.Power.CompareTo((other as Kettle).Power);
        }
        return 0;
    }
}
}

```

Oven.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{

```

```

/// <summary>
/// Class of oven data
/// </summary>
public class Oven : Device
{
    /// <summary>
    /// Oven power
    /// </summary>
    public double Power { get; set; }

    /// <summary>
    /// Oven number of programs
    /// </summary>
    public int NumberOfPrograms { get; set; }

    /// <summary>
    /// Oven class constructor
    /// </summary>
    /// <param name="type">Device type</param>
    /// <param name="power">Oven power</param>
    /// <param name="numberOfPrograms">Oven number of programs</param>
    /// <param name="maker">Oven maker</param>
    /// <param name="model">Oven model</param>
    /// <param name="energyClass">Oven energy class</param>
    /// <param name="color">Oven color</param>
    /// <param name="price">Oven price</param>
    public Oven(string type, double power, int numberOfPrograms, string maker,
string model, string energyClass, string color, decimal price) : base(maker,
model, energyClass, color, price, type)
    {
        this.Power = power;
        this.NumberOfPrograms = numberOfPrograms;
    }

    /// <summary>
    /// Method that forms a line of data in text
    /// </summary>
    /// <returns>Formatted string</returns>
    public override string ToString()
    {
        return string.Format("| {0, 15} | {1, 15} | {2, 15} | {3, 15} | {4,
15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} | {10, -15} | {11, -15}
| {12, -15} | {13, -15} | {14, -15} | {15, -15} |", Type, Maker, Model,
EnergyClass, Color, Price, "-", "-", "-", "-", "-", "-", "-", "-", "-", Power,
NumberOfPrograms);
    }

    /// <summary>
    /// Method finds out if information is equal or not
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>True of false value</returns>
    public override bool Equals(Device other)
    {
        return Type.CompareTo(other.Type) == 0 && Maker.CompareTo(other.Maker)
== 0 && Model.CompareTo(other.Model) == 0 &&
EnergyClass.CompareTo(other.EnergyClass) == 0 && Color.CompareTo(other.Color) == 0
&& Price.CompareTo(other.Price) == 0;
    }

    /// <summary>
    /// Method that compares information by hight
    /// </summary>
    /// <param name="other">Device information</param>
    /// <returns>Int type value</returns>

```

```

        public override int CompareTo(Device other)
        {
            if (other is Oven)
            {
                return this.Power.CompareTo((other as Oven).Power);
            }
            return 0;
        }
    }
}

```

DeviceRegister.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{
    /// <summary>
    /// Register of devices list
    /// </summary>
    public class DevicesRegister
    {
        /// <summary>
        /// List of all data
        /// </summary>
        private List<Device> AllData;

        /// <summary>
        /// Name of the shop
        /// </summary>
        public string ShopName { get; set; }

        /// <summary>
        /// Address of the shop
        /// </summary>
        public string ShopAddress { get; set; }

        /// <summary>
        /// Shop's phone number
        /// </summary>
        public string PhoneNumber { get; set; }

        /// <summary>
        /// DeviceRegister class constructor
        /// </summary>
        /// <param name="data">List of all data</param>
        /// <param name="shopName">Name of the shop</param>
        /// <param name="shopAddress">Address of the shop</param>
        /// <param name="phoneNumber">Shop's phone number</param>
        public DevicesRegister(List<Device> data, string shopName, string
shopAddress, string phoneNumber)
        {
            this.AllData = data;
            this.ShopName = shopName;
            this.ShopAddress = shopAddress;
            this.PhoneNumber = phoneNumber;
        }

        /// <summary>
        /// Method that count the number of total data
        /// </summary>
        /// <returns>Int type value</returns>

```

```

public int Count()
{
    return AllData.Count;
}

/// <summary>
/// Method that gives information about one device
/// </summary>
/// <param name="index">Int type variable</param>
/// <returns>Single device</returns>
public Device Get(int index)
{
    return AllData[index];
}

/// <summary>
/// Method that finds a number of certain maker's devices
/// </summary>
/// <returns>Int type variable</returns>
public int NumberOfSertainMakersItems()
{
    int sum = 0;

    for (int i = 0; i < AllData.Count(); i++)
    {
        if (AllData[i].Maker == "Siemens")
        {
            sum++;
        }
    }
    return sum;
}

/// <summary>
/// Method that forms a list of fridges
/// </summary>
/// <returns>List of fridges</returns>
public List<Device> AllFridges()
{
    List<Device> fridges = new List<Device>();

    for (int i = 0; i < AllData.Count(); i++)
    {
        if (AllData[i] is Fridge)
        {
            fridges.Add(AllData[i]);
        }
    }
    return fridges;
}

/// <summary>
/// Method that forms a list of expensive devices
/// </summary>
/// <returns>List of expensive devices</returns>
public List<Device> ExpensiveDevices()
{
    List<Device> expensive = new List<Device>();

    for (int i = 0; i < AllData.Count(); i++)
    {
        if (AllData[i] is Fridge && (AllData[i] as Fridge).Price > 1000)
        {
            expensive.Add(AllData[i]);
        }
    }
}

```

```

        else if (AllData[i] is Oven && (AllData[i] as Oven).Price > 500)
        {
            expencive.Add(AllData[i]);
        }
        else if (AllData[i] is Kettle && (AllData[i] as Kettle).Price >
50)
        {
            expencive.Add(AllData[i]);
        }
    }
    return expencive;
}
}
}

```

TaskUtils.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Lab_4
{
    /// <summary>
    /// Class for certain calculatians
    /// </summary>
    public static class TaskUtils
    {
        /// <summary>
        /// Method that forms a list of all fridges
        /// </summary>
        /// <param name="shops">List of all devices</param>
        /// <returns>Formated list</returns>
        private static List<Device> AllFridges(List<DevicesRegister> shops)
        {
            List<Device> allFridges = new List<Device>();

            foreach (DevicesRegister shop in shops)
            {
                List<Device> fridges = shop.AllFridges();

                for (int i = 0; i < fridges.Count(); i++)
                {
                    if (!allFridges.Exists(x => x.Equals(fridges[i])))
                    {
                        allFridges.Add(fridges[i]);
                    }
                }
            }
            return allFridges;
        }

        /// <summary>
        /// Method that forms a list of sorted fridges by price
        /// </summary>
        /// <param name="shops">List of all devices</param>
        /// <returns>formated list</returns>
        private static List<Device> SortedFridges(List<DevicesRegister> shops)
        {
            List<Device> allFridges = AllFridges(shops);

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

```

```

        for (int j = i + 1; j < allFridges.Count(); j++)
        {
            if (allFridges[j].Price < allFridges[i].Price)
            {
                Device device = allFridges[i];
                allFridges[i] = allFridges[j];
                allFridges[j] = device;
            }
        }
    }
    return allFridges;
}

/// <summary>
/// Method that filters list of fridges by mounting type and capacity
/// </summary>
/// <param name="shops">List of all devices</param>
/// <returns>Formatted list</returns>
private static List<Device> FilteredFridges(List<DevicesRegister> shops)
{
    List<Device> sortedFridges = SortedFridges(shops);
    List<Device> filteredFridges = new List<Device>();

    for (int i = 0; i < sortedFridges.Count(); i++)
    {
        if ((sortedFridges[i] as Fridge).MountingType == "Pastatomas" &&
(sortedFridges[i] as Fridge).Capacity >= 80)
        {
            filteredFridges.Add(sortedFridges[i]);
        }
    }
    return filteredFridges;
}

/// <summary>
/// Method that forms a list of first ten fridges
/// </summary>
/// <param name="shops">List of all devices</param>
/// <returns>Formatted list</returns>
public static List<Device> TenSelectedFridges(List<DevicesRegister> shops)
{
    List<Device> allFridges = FilteredFridges(shops);

    List<Device> onlyTenFridges = new List<Device>();

    for (int i = 0; i < allFridges.Count(); i++)
    {
        if (i == 10)
        {
            break;
        }
        onlyTenFridges.Add(allFridges[i]);
    }
    return onlyTenFridges;
}

/// <summary>
/// Method that finds out if a device exist only in one shop or not
/// </summary>
/// <param name="index">An index of certain place in the list</param>
/// <param name="shops">List of all devices</param>
/// <param name="device">Certain device</param>
/// <returns>True or false value</returns>
private static bool Exsist(int index, List<DevicesRegister> shops, Device
device)

```



```

{
    int x = 0;
    foreach (DevicesRegister shop in shops)
    {
        if (x != index)
        {
            for (int i = 0; i < shop.Count(); i++)
            {
                if (shop.Get(i).Equals(device))
                {
                    return true;
                }
            }
        }
        x++;
    }
    return false;
}

/// <summary>
/// Method that forms a list of devices that belong only in one shop
/// </summary>
/// <param name="shops">List of all devices</param>
/// <returns>Formatted list</returns>
public static List<Device> OnlyThere(List<DevicesRegister> shops)
{
    List<Device> onlyThere = new List<Device>();

    int index = 0;
    foreach (DevicesRegister shop in shops)
    {
        for (int i = 0; i < shop.Count(); i++)
        {
            if (Exsist(index, shops, shop.Get(i)) == false)
            {
                onlyThere.Add(shop.Get(i));
            }
        }
        index++;
    }
    return onlyThere;
}

/// <summary>
/// Method that forms a list of all expensive devices
/// </summary>
/// <param name="shops">List of all devices</param>
/// <returns>Formatted list</returns>
public static List<Device> AllExpensiveDevices(List<DevicesRegister>
shops)
{
    List<Device> allExpensiveDevices = new List<Device>();

    foreach (DevicesRegister shop in shops)
    {
        List<Device> expensiveDevices = shop.ExpensiveDevices();

        for (int i = 0; i < expensiveDevices.Count(); i++)
        {
            if (!allExpensiveDevices.Exists(x =>
x.Equals(expensiveDevices[i])))
            {
                allExpensiveDevices.Add(expensiveDevices[i]);
            }
        }
    }
}

```

```

    }

    return allExpensiveDevices;
}

/// <summary>
/// Method that sorts a list of expensive devices by certain attributes
/// </summary>
/// <param name="expensiveDevices">List of expenncive devices</param>
public static void Sort(List<Device> expensiveDevices)
{
    for (int i = 0; i < expensiveDevices.Count(); i++)
    {
        for (int j = i + 1; j < expensiveDevices.Count(); j++)
        {
            if (expensiveDevices[j].CompareTo(expensiveDevices[i]) < 0)
            {
                Device device = expensiveDevices[i];
                expensiveDevices[i] = expensiveDevices[j];
                expensiveDevices[j] = device;
            }
        }
    }
}
}
}

```

Main2.cs

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.IO;

namespace Lab_4
{
    /// <summary>
    /// Class that forms tables
    /// </summary>

    public partial class Main : System.Web.UI.Page
    {
        /// <summary>
        /// Method that creates first table of given data
        /// </summary>

```

```

/// <param name="allShops">List of all devices</param>
protected void CreateTable(List<DevicesRegister> allShops)
{
    foreach (DevicesRegister shop in allShops)
    {
        TableRow row1 = new TableRow();

        TableCell Cell1 = new TableCell();
        Cell1.Text = string.Format("{0}", shop.ShopName);
        row1.Cells.Add(Cell1);
        row1.CssClass = "ShopInformation";
        Table1.Rows.Add(row1);

        TableRow row2 = new TableRow();

        TableCell Cell2 = new TableCell();
        Cell2.Text = string.Format("{0}", shop.ShopAddress);
        row2.Cells.Add(Cell2);

        TableCell Cell20 = new TableCell();
        Cell20.Text = "Parduotuvės informacija";
        row2.Cells.Add(Cell20);
        row2.CssClass = "ShopInformation";
        Table1.Rows.Add(row2);

        TableRow row3 = new TableRow();

        TableCell Cell3 = new TableCell();
        Cell3.Text = string.Format("{0}", shop.PhoneNumber);
        row3.Cells.Add(Cell3);
        row3.CssClass = "ShopInformation";
        Table1.Rows.Add(row3);
    }
}

```

```

TableRow row = new TableRow();

TableCell cell0 = new TableCell();
cell0.Text = "Tipas";
row.Cells.Add(cell0);

TableCell cell1 = new TableCell();
cell1.Text = "Gamintojas";
row.Cells.Add(cell1);

TableCell cell2 = new TableCell();
cell2.Text = "Modelis";
row.Cells.Add(cell2);

TableCell cell3 = new TableCell();
cell3.Text = "Energijos klasė";
row.Cells.Add(cell3);

TableCell cell4 = new TableCell();
cell4.Text = "Spalva";
row.Cells.Add(cell4);

TableCell cell5 = new TableCell();
cell5.Text = "Kaina";
row.Cells.Add(cell5);

TableCell cell6 = new TableCell();
cell6.Text = "Talpa";
row.Cells.Add(cell6);

TableCell cell7 = new TableCell();

```

```
cell7.Text = "Montavimo tipas";
row.Cells.Add(cell7);

TableCell cell8 = new TableCell();
cell8.Text = "Požymis: turi šaldiklį ar ne";
row.Cells.Add(cell8);

TableCell cell9 = new TableCell();
cell9.Text = "Aukštis";
row.Cells.Add(cell9);

TableCell cell10 = new TableCell();
cell10.Text = "Plotis";
row.Cells.Add(cell10);

TableCell cell11 = new TableCell();
cell11.Text = "Gylis";
row.Cells.Add(cell11);

TableCell cell12 = new TableCell();
cell12.Text = "Galia";
row.Cells.Add(cell12);

TableCell cell13 = new TableCell();
cell13.Text = "Tūris";
row.Cells.Add(cell13);

TableCell cell14 = new TableCell();
cell14.Text = "Galingumas";
row.Cells.Add(cell14);

TableCell cell15 = new TableCell();
```

```

cell15.Text = "Programų skaičius";

row.Cells.Add(cell15);

row.CssClass = "Header";

Table1.Rows.Add(row);


for (int i = 0; i < shop.Count(); i++)
{
    Device device = shop.Get(i);

    TableRow newRow = new TableRow();

    TableCell newCell0 = new TableCell();
    newCell0.Text = string.Format("{0}", device.Type);
    newRow.Cells.Add(newCell0);

    TableCell newCell1 = new TableCell();
    newCell1.Text = string.Format("{0}", device.Maker);
    newRow.Cells.Add(newCell1);

    TableCell newCell2 = new TableCell();
    newCell2.Text = string.Format("{0}", device.Model);
    newRow.Cells.Add(newCell2);

    TableCell newCell3 = new TableCell();
    newCell3.Text = string.Format("{0}", device.EnergyClass);
    newRow.Cells.Add(newCell3);

    TableCell newCell4 = new TableCell();
    newCell4.Text = string.Format("{0}", device.Color);
    newRow.Cells.Add(newCell4);

    TableCell newCell5 = new TableCell();

```

```

newCell15.Text = string.Format("{0}", device.Price);
newRow.Cells.Add(newCell15);

if (device is Fridge)
{
    TableCell newCell16 = new TableCell();
    newCell16.Text = string.Format("{0}", (device as
Fridge).Capacity);
    newRow.Cells.Add(newCell16);

    TableCell newCell17 = new TableCell();
    newCell17.Text = string.Format("{0}", (device as
Fridge).MountingType);
    newRow.Cells.Add(newCell17);

    TableCell newCell18 = new TableCell();
    newCell18.Text = string.Format("{0}", (device as
Fridge).HasFridge);
    newRow.Cells.Add(newCell18);

    TableCell newCell19 = new TableCell();
    newCell19.Text = string.Format("{0}", (device as
Fridge).Hight);
    newRow.Cells.Add(newCell19);

    TableCell newCell110 = new TableCell();
    newCell110.Text = string.Format("{0}", (device as
Fridge).Width);
    newRow.Cells.Add(newCell110);

    TableCell newCell111 = new TableCell();
    newCell111.Text = string.Format("{0}", (device as
Fridge).Depth);
    newRow.Cells.Add(newCell111);
}

```

```

if (device is Kettle)
{
    TableCell newCell6 = new TableCell();
    newCell6.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell6);

    TableCell newCell7 = new TableCell();
    newCell7.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell7);

    TableCell newCell8 = new TableCell();
    newCell8.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell8);

    TableCell newCell9 = new TableCell();
    newCell9.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell9);

    TableCell newCell10 = new TableCell();
    newCell10.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell10);

    TableCell newCell11 = new TableCell();
    newCell11.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell11);

    TableCell newCell12 = new TableCell();
    newCell12.Text = string.Format("{0}", (device as
Kettle).Power);

    newRow.Cells.Add(newCell12);

    TableCell newCell13 = new TableCell();

```



```

Kettle).Volume);

newCell13.Text = string.Format("{0}", (device as

newRow.Cells.Add(newCell13);

}

if (device is Oven)
{
    TableCell newCell6 = new TableCell();
    newCell6.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell6);

    TableCell newCell7 = new TableCell();
    newCell7.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell7);

    TableCell newCell8 = new TableCell();
    newCell8.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell8);

    TableCell newCell9 = new TableCell();
    newCell9.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell9);

    TableCell newCell10 = new TableCell();
    newCell10.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell10);

    TableCell newCell11 = new TableCell();
    newCell11.Text = string.Format("{0}", " ");
    newRow.Cells.Add(newCell11);

    TableCell newCell12 = new TableCell();
    newCell12.Text = string.Format("{0}", " ");

```

```

newRow.Cells.Add(newCell12);

TableCell newCell13 = new TableCell();
newCell13.Text = string.Format("{0}", " ");
newRow.Cells.Add(newCell13);

TableCell newCell14 = new TableCell();
newCell14.Text = string.Format("{0}", (device as
Oven).Power);

newRow.Cells.Add(newCell14);

TableCell newCell15 = new TableCell();
newCell15.Text = string.Format("{0}", (device as
Oven).NumberOfPrograms);

newRow.Cells.Add(newCell15);
}

Table1.Rows.Add(newRow);
}

}

}

/// <summary>
/// Method that print information to second table
/// </summary>
/// <param name="allShops">List of all data</param>
protected void PrintTable2(List<DevicesRegister> allShops)
{
    foreach (DevicesRegister shop in allShops)
    {
        TableRow row1 = new TableRow();

```

```

        TableCell Cell11 = new TableCell();

        Cell11.Text = string.Format("{0}", shop.ShopName);

        row1.Cells.Add(Cell11);

        row1.CssClass = "ShopInformation";

        Table2.Rows.Add(row1);


        TableRow row2 = new TableRow();


        TableCell Cell12 = new TableCell();

        Cell12.Text = string.Format("{0}", shop.ShopAddress);

        row2.Cells.Add(Cell12);


        TableCell Cell20 = new TableCell();

        Cell20.Text = "Parduotuvės informacija";

        row2.Cells.Add(Cell20);

        row2.CssClass = "ShopInformation";

        Table2.Rows.Add(row2);


        TableRow row3 = new TableRow();


        TableCell Cell13 = new TableCell();

        Cell13.Text = string.Format("{0}", shop.PhoneNumber);

        row3.Cells.Add(Cell13);

        row3.CssClass = "ShopInformation";

        Table2.Rows.Add(row3);


        TableRow row4 = new TableRow();


        TableCell Cell14 = new TableCell();

        Cell14.Text = string.Format("{0}",
shop.NumberOfSertainMakersItems());

        row4.Cells.Add(Cell14);

        Table2.Rows.Add(row4);

```

```

    }
}

/// <summary>
/// Method that forms a third table
/// </summary>
/// <param name="tenFridges">List of ten fridges</param>
protected void PrintTable3(List<Device> tenFridges)
{
    TableRow row = new TableRow();

    TableCell cell1 = new TableCell();
    cell1.Text = "Gamintojas";
    row.Cells.Add(cell1);

    TableCell cell2 = new TableCell();
    cell2.Text = "Modelis";
    row.Cells.Add(cell2);

    TableCell cell3 = new TableCell();
    cell3.Text = "Talpa";
    row.Cells.Add(cell3);

    TableCell cell4 = new TableCell();
    cell4.Text = "Kaina";
    row.Cells.Add(cell4);

    row.CssClass = "Header";
    Table3.Rows.Add(row);

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

```

```

        TableRow newRow = new TableRow();

        TableCell Cell1 = new TableCell();

        Cell1.Text = string.Format("{0}", tenFridges[i].Maker);

        newRow.Cells.Add(Cell1);

        TableCell Cell2 = new TableCell();

        Cell2.Text = string.Format("{0}", (tenFridges[i] as
Fridge).MountingType);

        newRow.Cells.Add(Cell2);

        TableCell Cell3 = new TableCell();

        Cell3.Text = string.Format("{0}", (tenFridges[i] as
Fridge).Capacity);

        newRow.Cells.Add(Cell3);

        TableCell Cell4 = new TableCell();

        Cell4.Text = string.Format("{0}", (tenFridges[i] as
Fridge).Price);

        newRow.Cells.Add(Cell4);

        Table3.Rows.Add(newRow);
    }
}

```

StyleSheet1.cs

```

div {
    background-color: mistyrose;
}

.Table {
    background-color: yellow;
}

```

```

body {
    background-color: cadetblue;
}

.BoldText {
    font-weight: bold;
}

.TextSize {
    font-size: 18px;
}

.ShopInformation {
    background-color: lightblue;
}

.Header {
    background-color: white;
}

.LineColor {
    background-color: black;
}

```

Main.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Main.aspx.cs"
Inherits="Lab_4.Main" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="StyleSheet1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
        <div class="TextSize">
            Užkraukite bent vieną pradinį duomenų failą iš aplanko, kad
            užkrautumėte likusius:<br />
            <asp:FileUpload ID="FileUpload1" runat="server" CssClass="BoldText"/>
            <br />
            <asp:Label ID="Label1" runat="server" CssClass="TextSize"></asp:Label>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
            Text="Užkrauti duomenis" CssClass="BoldText" />
            &nbsp;<br />
            <asp:Table ID="Table1" runat="server" BorderWidth="1px"
            ForeColor="Black" GridLines="Both" CssClass="Table">
                </asp:Table>
                <br />
                Skaičius „Siemens“ šaldytuvų, mikrobangų krosnelių ir virdulių
                modelių, kuri siūlo kiekviena parduotuvė:<br />
                <asp:Table ID="Table2" runat="server" BorderWidth="1px"
                ForeColor="Black" GridLines="Both" CssClass="Table">
                    </asp:Table>
                    <br />
                    Dešimt pigiausių pastatomų šaldytuvų, kurių talpa yra 80 litrų ar
                    didesnė:<br />
                    <asp:Table ID="Table3" runat="server" BorderWidth="1px"
                    ForeColor="Black" GridLines="Both" CssClass="Table">
                        </asp:Table>
                        <br />

```

```

        Išspausdinama, ar yra tokių buitinių prietaisų, kuriuos galima įsigyti
        tik vienoje parduotuvėje:<br />
        <br />
        <asp:Label ID="Label2" runat="server" CssClass="TextSize"></asp:Label>
        <br />
    </div>
</form>
</body>
</html>

```

Main.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
//Main function of this program is to do all kinds of calculations with given data
of devices in shops

//Vytenis Kriščiūnas
namespace Lab_4
{
    /// <summary>
    /// Web Form
    /// </summary>
    public partial class Main : System.Web.UI.Page
    {
        /// <summary>
        /// Prints information to the screen when page loads
        /// </summary>
        /// <param name="sender">An object variable</param>
        /// <param name="e">EventArgs variable</param>
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        /// <summary>
        /// Prints and reads given information when the first button is clicked
        /// </summary>
        /// <param name="sender">An object variable</param>
        /// <param name="e">EventArgs variable</param>
        protected void Button1_Click(object sender, EventArgs e)
        {

            List<DevicesRegister> allShops = new List<DevicesRegister>();
            List<Device> TenFridges = new List<Device>();
            List<Device> OnlyThere = new List<Device>();
            List<Device> allExpensiveDevices = new List<Device>();

            Label2.Text = "";
            string pathToFile1 = Server.MapPath("App_Data/Data");
            string pathToFile2 = Server.MapPath("App_Data/Results/Results.txt");
            string pathToFile3 = Server.MapPath("App_Data/Results/TikTen.csv");
            string pathToFile4 = Server.MapPath("App_Data/Results/Brangus.csv");

            File.Delete(pathToFile2);
            File.Delete(pathToFile3);
            File.Delete(pathToFile4);

```

```

if (FileUpload1.HasFile)
{
    Label1.Text = "Pradiniai duomenys pasirinkti.";
    allShops = InOut.ReadFile(pathToFile1);
    InOut.PrintToTxt(pathToFile2, allShops);
    CreateTable(allShops);
    Session["allShops"] = allShops;

    PrintTable2(allShops);
    InOut.PrintToTxtSertainItemsMakers(pathToFile2, allShops);

    TenFridges = TaskUtils.TenSelectedFridges(allShops);
    PrintTable3(TenFridges);
    InOut.PrintTenFridges(pathToFile2, TenFridges);
    Session["TenFridges"] = TenFridges;

    OnlyThere = TaskUtils.OnlyThere(allShops);
    try
    {
        string type = OnlyThere[0].Type;
        Label2.Text = "Tokių buitinių prietaisų yra.";

        File.AppendAllText(pathToFile3, string.Format("Prietaisai, kurių galima įsigyti tik vienoje parduotuvėje: \n"));
        File.AppendAllText(pathToFile3, string.Format("| {0, 15} | {1, 15} | {2, 15} | {3, 15} | {4, 15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} | {10, -15} | {11, -15} | {12, -15} | {13, -15} | {14, -15} |\n", "Tipas", "Gamintojas", "Modelis", "Energijos klasė", "Spalva", "Kaina", "Talpa", "Montavimo tipas", "Požymis: turi šaldiklį ar ne", "Aukštis", "Plotis", "Gylis", "Galia", "Tūris", "Galingumas", "Programų skaičius"));
        InOut.PrintToCsv(pathToFile3, OnlyThere, "Fridge");
        InOut.PrintToCsv(pathToFile3, OnlyThere, "Kettle");
        InOut.PrintToCsv(pathToFile3, OnlyThere, "Oven");
    }
    catch (Exception)
    {
        Label2.Text = "Tokių buitinių prietaisų nėra.";
    }

    allExpensiveDevices = TaskUtils.AllExpensiveDevices(allShops);
    try
    {
        string type = allExpensiveDevices[0].Type;
        TaskUtils.Sort(allExpensiveDevices);

        File.AppendAllText(pathToFile4, string.Format("Brangiausių buitinių prekių sąrašas: \n"));
        File.AppendAllText(pathToFile4, string.Format("| {0, 15} | {1, 15} | {2, 15} | {3, 15} | {4, 15} | {5, -15} | {6, -15} | {7, 15} | {8, -28} | {9, -15} | {10, -15} | {11, -15} | {12, -15} | {13, -15} | {14, -15} |\n", "Tipas", "Gamintojas", "Modelis", "Energijos klasė", "Spalva", "Kaina", "Talpa", "Montavimo tipas", "Požymis: turi šaldiklį ar ne", "Aukštis", "Plotis", "Gylis", "Galia", "Tūris", "Galingumas", "Programų skaičius"));
        InOut.PrintToCsv(pathToFile4, allExpensiveDevices, "Fridge");
        InOut.PrintToCsv(pathToFile4, allExpensiveDevices, "Kettle");
        InOut.PrintToCsv(pathToFile4, allExpensiveDevices, "Oven");
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

```



```

    }
    else
    {
        Label1.Text = "Pradiniai duomenys nepasirinkti.";
    }
}
}
}

```

4.7. Pradiniai duomenys ir rezultatai

Shop1.txt

Senukai

Pakalnio 51 g.

8643 5034 68

```

Fridge;Siemens;fd6;df-5;raudona;1000;80;ypatingasis;turi šaldiklį;5;2;2
Fridge;Tiesto;35fd9;df-5;Juodas;200;90;Pastatomas;turi šaldiklį;7;3;3
Fridge;Siemens;fd9;df-5;Rudas;1500;50;Pastatomas;neturi šaldiklį;1;1.2;1.1
Kettle;Siemens;35fd4;df-5;raudona;1000;50;35
Kettle;Kold;35fd9;df-5;Juodas;200;50;35
Kettle;Siemens;35gd4;df-5;Rudas;1500;10;35
Oven;Old;35fd4;df-5;raudona;1000;50;10
Oven;Tiesto;35fd9;df-5;Juodas;200;50;10
Oven;Siemens;35gd4;df-5;Rudas;1500;50;10

```

Shop2.txt

Maksima

Pakalnio 53 g.

8643 5034 68

```

Fridge;Siemens;fd6;df-5;raudona;1000;80;ypatingasis;turi šaldiklį;5;2;2
Fridge;Tiesto;35fd9;df-5;Juodas;200;90;Pastatomas;turi šaldiklį;7;3;3
Fridge;Siemens;fd9;df-5;Rudas;1500;50;Pastatomas;neturi šaldiklį;1;1.2;1.1
Kettle;Siemens;35fd4;df-5;raudona;1000;50;35
Kettle;Kold;35fd9;df-5;Juodas;200;50;35
Kettle;Siemens;35gd4;df-5;Rudas;1500;10;35
Oven;Old;35fd4;df-5;raudona;1000;50;10
Oven;Tiesto;35fd9;df-5;Juodas;200;50;10
Oven;Siemens;35gd4;df-5;Rudas;1500;50;10

```

Shop3.txt

Lidlas

Piliakaln 53 g.

8643 5034 68

```

Fridge;Siemens;fd6;df-5;raudona;1500;80;Pastatomas;turi šaldiklį;5;2;2
Fridge;Jonas;35fd9;df-5;Juodas;200;90;Pastatomas;turi šaldiklį;7;3;3
Fridge;Pol;fd9;df-5;Rudas;1500;50;Pastatomas;neturi šaldiklį;1;1.2;1.1
Kettle;Pols;35fd4;df-5;raudona;1000;50;35
Kettle;Kold;35fd9;df-5;Juodas;200;50;35
Kettle;Nindas;35gd4;df-5;Rudas;1500;10;35
Oven;Old;35fd4;df-5;raudona;1000;50;10
Oven;Tiesto;35fd9;df-5;Juodas;200;50;10
Oven;Sold;35gd4;df-5;Rudas;1500;50;10

```

Rezultats.txt

Senukai

Pakalnio 51 g.

8643 5034 68

```

-----
|          Tipas |          Gamintojas |          Modelis | Energijos klasė |
Spalva | Kaina          | Talpa            | Montavimo tipas | Požymis: turi
šaldiklį ar ne | Aukštis          | Plotis           | Gylis            | Galia
| Tūris          | Galingumas       | Programų skaičius |
-----

```

```

-----
|          Fridge |          Siemens |          fd6 |          df-5 |
raudona | 1000            | 80             | ypatingasis | True
| 5              | 2              | 2            | -            | -
| -              | -              |              |              |
|          Fridge |          Tiesto  |          35fd9 |          df-5 |
Juodas  | 200             | 90             | Pastatomas  | True
| 7              | 3              | 3            | -            | -
| -              | -              |              |              |
|          Fridge |          Siemens |          fd9 |          df-5 |
Rudas   | 1500            | 50             | Pastatomas  | False
| 1              | 1.2            | 1.1          | -            | -
| -              | -              |              |              |
|          Kettle  |          Siemens |          35fd4 |          df-5 |
raudona | 1000            | -              |              | - | -
| -              | -              | -            | 50           | 35
| -              | -              |              |              |
|          Kettle  |          Kold    |          35fd9 |          df-5 |
Juodas  | 200             | -              |              | - | -
| -              | -              | -            | 50           | 35
| -              | -              |              |              |
|          Kettle  |          Siemens |          35gd4 |          df-5 |
Rudas   | 1500            | -              |              | - | -
| -              | -              | -            | 10           | 35
| -              | -              |              |              |
|          Oven    |          Old     |          35fd4 |          df-5 |
raudona | 1000            | -              |              | - | -
| -              | -              | -            | -            | -
| 50             | 10             |              |              |
|          Oven    |          Tiesto  |          35fd9 |          df-5 |
Juodas  | 200             | -              |              | - | -
| -              | -              | -            | -            | -
| 50             | 10             |              |              |
|          Oven    |          Siemens |          35gd4 |          df-5 |
Rudas   | 1500            | -              |              | - | -
| -              | -              | -            | -            | -
| 50             | 10             |              |              |
-----

```

Maksima
Pakalnio 53 g.
8643 5034 68

```

-----
-----
-----
-----
|          Tipas |          Gamintojas |          Modelis | Energijos klasė |
Spalva | Kaina          | Talpa          | Montavimo tipas | Požymis: turi
šaldiklį ar ne | Aukštis        | Plotis        | Gylis          | Galia
| Tūris          | Galingumas     | Programų skaičius |
-----
-----
-----

```

```

-----
-----
|          Fridge |          Siemens |          fd6 |          df-5 |
raudona | 1000          | 80          | ypatingasis | True
| 5          | 2          | 2          | -          | -
| -          | -          |          |          |
|          Fridge |          Tiesto |          35fd9 |          df-5 |
Juodas | 200          | 90          | Pastatomas | True
| 7          | 3          | 3          | -          | -
| -          | -          |          |          |
|          Fridge |          Siemens |          fd9 |          df-5 |
Rudas | 1500          | 50          | Pastatomas | False
| 1          | 1.2        | 1.1        | -          | -
| -          | -          |          |          |
|          Kettle |          Siemens |          35fd4 |          df-5 |
raudona | 1000          | -          | -          | -
| -          | -          | -          | 50         | 35
| -          | -          |          |          |
|          Kettle |          Kold |          35fd9 |          df-5 |
Juodas | 200          | -          | -          | -
| -          | -          | -          | 50         | 35
| -          | -          |          |          |
|          Kettle |          Siemens |          35gd4 |          df-5 |
Rudas | 1500          | -          | -          | -
| -          | -          | -          | 10         | 35
| -          | -          |          |          |
|          Oven |          Old |          35fd4 |          df-5 |
raudona | 1000          | -          | -          | -
| -          | -          | -          | -          | -
| 50         | 10         |          |          |
|          Oven |          Tiesto |          35fd9 |          df-5 |
Juodas | 200          | -          | -          | -
| -          | -          | -          | -          | -
| 50         | 10         |          |          |
|          Oven |          Siemens |          35gd4 |          df-5 |
Rudas | 1500          | -          | -          | -
| -          | -          | -          | -          | -
| 50         | 10         |          |          |
-----
-----
-----
-----

```

Lidl
Piliakaln 53 g.
8643 5034 68

Tipas	Gamintojas	Modelis	Energijos klasė
Spalva Kaina	Talpa	Montavimo tipas	Požymis: turi šaldiklį ar ne
Aukštis	Plotis	Gylis	Galia
Tūris	Galingumas	Programų skaičius	

Fridge	Siemens	fd6	df-5
raudona 1500	80	Pastatomas	True
5	2	-	-
-	-		
Fridge	Ionas	35fd9	df-5
Juodas 200	90	Pastatomas	True
7	3	-	-
-	-		
Fridge	Pol	fd9	df-5
Rudas 1500	50	Pastatomas	False
1	1.2	-	-
-	-		
Kettle	Pols	35fd4	df-5
raudona 1000	-	-	-
-	-	50	35
-	-		
Kettle	Kold	35fd9	df-5
Juodas 200	-	-	-
-	-	50	35
-	-		
Kettle	Nindas	35gd4	df-5
Rudas 1500	-	-	-
-	-	10	35
-	-		
Oven	Old	35fd4	df-5
raudona 1000	-	-	-
-	-	-	-
50	10		
Oven	Tiesto	35fd9	df-5
Juodas 200	-	-	-
-	-	-	-
50	10		
Oven	Sold	35gd4	df-5
Rudas 1500	-	-	-
-	-	-	-
50	10		

Skaičius „Siemens“ šaldytuvų, mikrobangų krosnelių ir virdulių modelių, kurių siūlo kiekviena parduotuvė:

5

Parduotuvės kontaktinė informacija:

Senukai

Pakalnio 51 g.

8643 5034 68

5

Parduotuvės kontaktinė informacija:

Maksima

Pakalnio 53 g.

8643 5034 68

1

Parduotuvės kontaktinė informacija:

Lidlas

Piliakaln 53 g.

8643 5034 68

Dešimt pigiausių pastatomų šaldytuvų, kurių talpa yra 80 litrų ar didesnė:

Gamintojas	Modelis	Talpa	Kaina

Tiesto	Pastatomas	90	200
Ionas	Pastatomas	90	200
Siemens	Pastatomas	80	1500

Vartotojo sąsąja

Užkraukite bent vieną pradinį duomenų failą iš aplanko, kad užkrautumėte likusius:

[Choose File](#) No file chosen

Pradiniai duomenys pasirinkti.

Užkrauti duomenis

Serukai

Pakalnio 51 g Parduotuvės informacija

8643 5034 68

Tipas	Gamintojas	Modelis	Energijos klase	Spalva	Kaina	Talpa	Montavimo tipas	Pažymis: turi šaldiklį ar ne	Aukštis	Plotis	Gylis	Galio	Tūris	Galingumas	Programų skaičius
Fridge	Siemens	f66	dif-5	raudona	1000	80	ypatingasis	True	5	2	2				
Fridge	Tiesto	35f69	dif-5	Juodas	200	90	Pastatomas	True	7	3	3				
Fridge	Siemens	f69	dif-5	Rudas	1500	50	Pastatomas	False	1	1.2	1.1				
Kettle	Siemens	35f64	dif-5	raudona	1000							50	35		
Kettle	Kold	35f69	dif-5	Juodas	200							50	35		
Kettle	Siemens	35g64	dif-5	Rudas	1500							10	35		
Oven	Old	35f64	dif-5	raudona	1000									50	10
Oven	Tiesto	35f69	dif-5	Juodas	200									50	10
Oven	Siemens	35g64	dif-5	Rudas	1500									50	10

Maksima

Pakalnio 53 g Parduotuvės informacija

8643 5034 68

Tipas	Gamintojas	Modelis	Energijos klase	Spalva	Kaina	Talpa	Montavimo tipas	Pažymis: turi šaldiklį ar ne	Aukštis	Plotis	Gylis	Galio	Tūris	Galingumas	Programų skaičius
Fridge	Siemens	f66	dif-5	raudona	1000	80	ypatingasis	True	5	2	2				
Fridge	Tiesto	35f69	dif-5	Juodas	200	90	Pastatomas	True	7	3	3				
Fridge	Siemens	f69	dif-5	Rudas	1500	50	Pastatomas	False	1	1.2	1.1				
Kettle	Siemens	35f64	dif-5	raudona	1000							50	35		
Kettle	Kold	35f69	dif-5	Juodas	200							50	35		
Kettle	Siemens	35g64	dif-5	Rudas	1500							10	35		
Oven	Old	35f64	dif-5	raudona	1000									50	10
Oven	Tiesto	35f69	dif-5	Juodas	200									50	10
Oven	Siemens	35g64	dif-5	Rudas	1500									50	10

Lidias

Piliakaln 53 g Parduotuvės informacija

8643 5034 68

Tipas	Gamintojas	Modelis	Energijos klase	Spalva	Kaina	Talpa	Montavimo tipas	Pažymis: turi šaldiklį ar ne	Aukštis	Plotis	Gylis	Galio	Tūris	Galingumas	Programų skaičius
Fridge	Siemens	f66	dif-5	raudona	1500	80	Pastatomas	True	5	2	2				
Fridge	Jonas	35f69	dif-5	Juodas	200	90	Pastatomas	True	7	3	3				
Fridge	Pol	f69	dif-5	Rudas	1500	50	Pastatomas	False	1	1.2	1.1				
Kettle	Polis	35f64	dif-5	raudona	1000							50	35		
Kettle	Kold	35f69	dif-5	Juodas	200							50	35		
Kettle	Nindas	35g64	dif-5	Rudas	1500							10	35		
Oven	Old	35f64	dif-5	raudona	1000									50	10
Oven	Tiesto	35f69	dif-5	Juodas	200									50	10
Oven	Sold	35g64	dif-5	Rudas	1500									50	10

Skaičius „Siemens“ šaldytuvų, mikrobangų krosnelių ir virdulių modelių, kurį siūlo kiekviena parduotuvė:

Serukai

Pakalnio 51 g Parduotuvės informacija

8643 5034 68

5

Maksima

Pakalnio 53 g Parduotuvės informacija

8643 5034 68

5

Lidias

Piliakaln 53 g Parduotuvės informacija

8643 5034 68

1

Dešimt pigiausių pastatomų šaldytuvų, kurių talpa yra 80 litrų ar didesnė:

Gamintojas	Modelis	Talpa	Kaina
Tiesto	Pastatomas	90	200
Jonas	Pastatomas	90	200
Siemens	Pastatomas	80	1500

Išspausdinama, ar yra tokių buitinių prietaisų, kuriuos galima įsigyti tik vienoje parduotuvėje:

Tokių buitinių prietaisų yra.

4.8. Dėstytojo pastabos