

Kauno technologijos universitetas

Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Aistis Jakutonis IFF3/1

Studentas

Prof. Vacius Jusas

Kaunas 2024

TURINYS

1. Rekursija (L1) 4

1.1. Darbo užduotis 4

1.2. Grafinės vartotojo sąsajos schema 4

1.3. Sąsajoje panaudotų komponentų keičiamos savybės 5

1.4. Klasių diagrama 5

1.5. Programos vartotojo vadovas 6

1.6. Programos tekstas 6

1.7. Pradiniai duomenys ir rezultatai 16

1.7.1 Pradiniai duomenys ir rezultatai 1 16

1.7.2 Pradiniai duomenys ir rezultatai 2 18

1.8. Dėstytojo pastabos 20

2. Dinaminis atminties valdymas (L2) 21

2.1. Darbo užduotis 21

2.2. Grafinės vartotojo sąsajos schema 21

2.3. Sąsajoje panaudotų komponentų keičiamos savybės 22

2.4. Klasių diagrama 23

2.5. Programos vartotojo vadovas 23

2.6. Programos tekstas 24

2.7. Pradiniai duomenys ir rezultatai 44

2.7.1 Duomenys ir rezultatai 1 44

2.7.2 Duomenys ir rezultatai 2 47

2.8. Dėstytojo pastabos 49

3. Bendrinės klasės ir testavimas (L3) 50

3.1. Darbo užduotis 50

3.2. Grafinės vartotojo sąsajos schema 51

3.3. Sąsajoje panaudotų komponentų keičiamos savybės 51

3.4. Klasių diagrama 53

3.5. Programos vartotojo vadovas 53

3.6. Programos tekstas 53

3.7. Pradiniai duomenys ir rezultatai 80

3.7.1 Duomenys ir rezultatai 1 81

3.7.2 Duomenys ir rezultatai 2 83

3.8. Dėstytojo pastabos 85

4. Polimorfizmas ir išimčių valdymas (L4) 86

4.1. Darbo užduotis 86

4.2. Grafinės vartotojo sąsajos schema 87

4.3. Sąsajoje panaudotų komponentų keičiamos savybės 87

4.4. Klasių diagrama 89

4.5. Programos vartotojo vadovas 89

4.6. Programos tekstas 89

4.7. Pradiniai duomenys ir rezultatai 114

4.7.1 Duomenys ir rezultatai 1 114

4.7.2 Duomenys ir rezultatai 2 117

4.8. Dėstytojo pastabos 120

5. Deklaratyvusis programavimas (L5) 121

5.1. Darbo užduotis 121

5.2. Grafinės vartotojo sąsajos schema 121

5.3. Sąsajoje panaudotų komponentų keičiamos savybės 121

5.4. Klasių diagrama 121

5.5. Programos vartotojo vadovas 121

5.6. Programos tekstas 121

5.7. Pradiniai duomenys ir rezultatai 121

5.8. Dėstytojo pastabos 122

# Rekursija (L1)

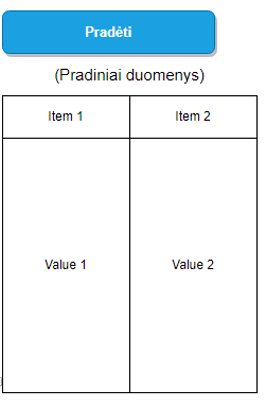
## Darbo užduotis

**LD\_22.** **Kelias tarp vietovių.**

Gūdučių universiteto informatikos fakulteto I kurso studentai nutarė dalyvauti orientavimosi dviračiais varžybose. Jie sudarė komandą ir atvyko į vietovę Preivai, kur bus duotas startas. Buvo pranešta, kad finišas Balkuose. Kaip ir kitų komandų atstovai, jie gavo vietovės žemėlapį, kuriame pažymėti visi keliai ir surašyti kelių ilgiai. Padėkite studentams surasti trumpiausią kelią tarp nurodytų vietovių, jei žinoma, kad kelias tarp starto ir finišo vietovių gali būti tiesioginis (be tarpinių vietovių) arba tarpe jų gali būti ne daugiau kaip 5 tarpinės vietovės.

**Duomenys**. Tekstinio failo ‘U3.txt’ pirmoje eilutėje nurodytas vietovių skaičius N (2≤N≤10) ir visų kelių kiekis M (1≤M≤50). Tolimesnėse N eilutėse surašytos visos galimos vietovės po vieną eilutėje. Po to eilutėje surašytos starto ir finišo vietovių pavadinimai. Šiai eilutei iš viršaus ir apačios palikta po vieną tuščią eilutę. Po antros tuščios eilutės M eilutėse surašyti visi keliai po vieną eilutėje. Tokios eilutės struktūra: pradinė vietovė, galinė vietovė, atstumas tarp jų. Vietovės pavadinimas – iki 10 simbolių.

## Grafinės vartotojo sąsajos schema

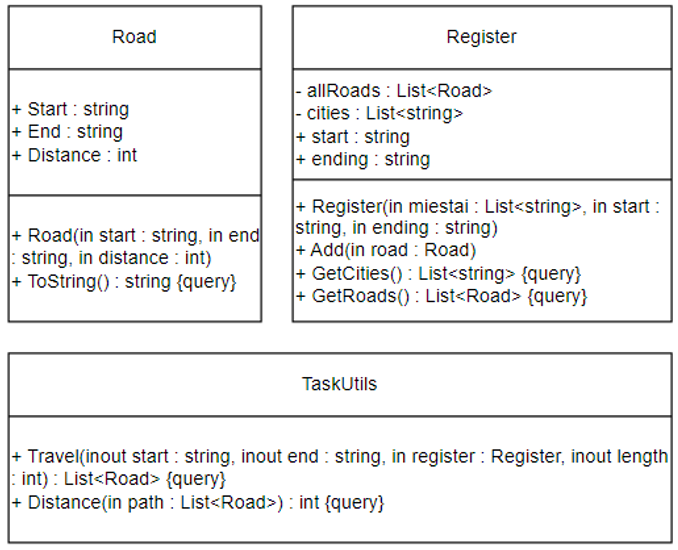


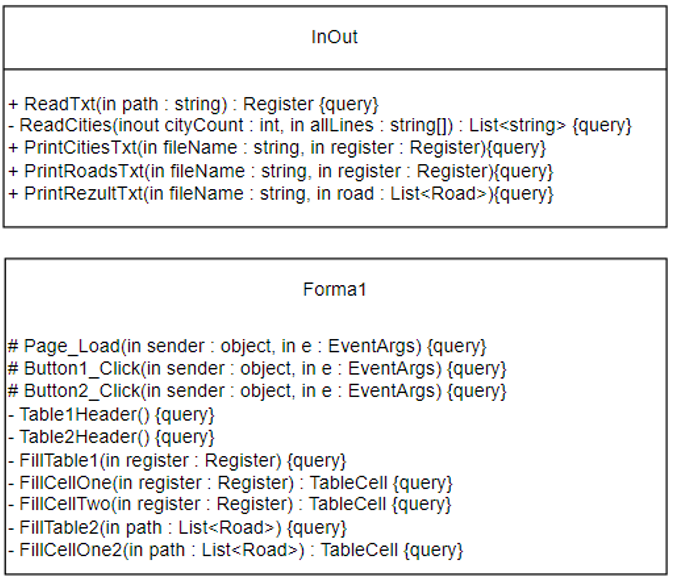


## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| Button | ID | Button1 |
| Button | OnClick | Button1\_Click |
| Button | Text | Pradėti |
| Button | Width | 103px |
| Button | ID | Button2 |
| Button | OnClick | Button2\_Click |
| Button | Text | Apskaičiuoti |
| Button | Width | 107px |
| Table | ID | Table1 |
| Table | BorderColor | Black |
| Table | BorderStyle | Solid |
| Table | BorderWidth | 1px |
| Table | Width | 390px |
| Table | ID | Table2 |
| Table | BorderColor | Black |
| Table | BorderStyle | Solid |
| Table | BorderWidth | 1px |
| Table | Width | 390px |

## Klasių diagrama





## Programos vartotojo vadovas

Programos darbiniame aplanke atidarome App\_Data aplanką, jame sukuriame failą U3.txt, kuriame pateikiame duomenis apie orientavimosi varžybas: Vietovių skaičius bei galimų maršrutų kiekis; išvardijame vietoves bei maršrutus bei jų ilgius; įrašome pradinę ir galutinę vietovę.

Įjungę programą, pirmiausia užkrauname duomenų failus. Tai padarome paspausdami mygtuką „Pradėti“. Paspaudę mygtuką patikriname ar duomenys buvo įvesti teisingai. Jei lentelėje duomenys teisingi, spaudžiame mygtuką „Apskaičiuoti“. Paspaudus mygtuką programa apdoros duomenis ir į ekraną išves lentelę su rezultatais.

## Programos tekstas

using System;

namespace LD22\_kelias\_tarp\_vietoviu

{

/// <summary>

/// Constructor class

/// </summary>

public class Road

{

public string Start { get; }

public string End { get; }

public int Distance { get; }

public Road(string start,

string end, int distance)

{

this.Start = start;

this.End = end;

this.Distance = distance;

}

public override string ToString()

{

string line;

line = String.Format($"| {this.Start,-11} " +

$"| {this.End,-11} | {this.Distance,8} km |");

return line;

}

}

}

using System.Collections.Generic;

namespace LD22\_kelias\_tarp\_vietoviu

{

/// <summary>

/// Register class in which the main

/// information is stored

/// </summary>

public class Register

{

private List<Road> allRoads = new List<Road>();

private List<string> cities { get; }

public string start { get; }

public string ending { get; }

/// <summary>

/// Gets cities, start and ending

/// </summary>

/// <param name="miestai"></param>

/// <param name="start"></param>

/// <param name="ending"></param>

public Register(List<string> miestai,

string start, string ending)

{

cities = new List<string>();

foreach (string city in miestai)

{

cities.Add(city);

}

this.start = start;

this.ending = ending;

}

/// <summary>

/// Method adds road to the allRoads list

/// </summary>

/// <param name="road"></param>

public void Add(Road road)

{

allRoads.Add(road);

}

/// <summary>

/// Method returns the cities list

/// </summary>

/// <returns></returns>

public List<string> GetCities()

{

return cities;

}

/// <summary>

/// Method returns the roads list

/// </summary>

/// <returns></returns>

public List<Road> GetRoads()

{

return allRoads;

}

}

}

using System.Collections.Generic;

namespace LD22\_kelias\_tarp\_vietoviu

{

/// <summary>

/// Class contains calculations and recursion

/// </summary>

public class TaskUtils

{

/// <summary>

/// Method with recursion to solve the task

/// </summary>

/// <param name="start"></param>

/// <param name="end"></param>

/// <param name="register"></param>

/// <param name="length"></param>

/// <returns></returns>

public static List<Road> Travel(string start,

string end, Register register, int length)

{

if (length > 5)

{

return null;

}

List<Road> path = null;

int distance = -1;

foreach (Road kelias in register.GetRoads())

{

List<Road> subpath;

if (kelias.Start == start

&& kelias.End == end)

{

subpath = new List<Road>();

subpath.Add(kelias);

}

else if (kelias.Start == start)

{

subpath = Travel(kelias.End, end,

register, length + 1);

if (subpath == null)

{

continue;

}

subpath.Insert(0, kelias);

}

else

{

continue;

}

int subdistance = Distance(subpath);

if (distance < 0 || distance

> subdistance)

{

distance = subdistance;

path = subpath;

}

}

return path;

}

/// <summary>

/// Method calculates the distance

/// between all given roads

/// </summary>

/// <param name="path"></param>

/// <returns></returns>

public static int Distance(List<Road> path)

{

int distance = 0;

foreach (Road road in path)

{

distance += road.Distance;

}

return distance;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

using System.Text.RegularExpressions;

namespace LD22\_kelias\_tarp\_vietoviu

{

/// <summary>

/// Reading and printing class

/// </summary>

public static class InOut

{

/// <summary>

/// Reads the data from the given file

/// </summary>

/// <param name="path"></param>

/// <returns></returns>

public static Register ReadTxt(string path)

{

string[] allLines = File.ReadAllLines(path);

string pattern = "\\s+";

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

int cityCount = int.Parse(parts[0]);

int roadCount = int.Parse(parts[1]);

List<string> cities = ReadCities(cityCount,

allLines);

string[] matches =

Regex.Split(allLines[cityCount + 2],

pattern);

string begining = matches[0];

string ending = matches[1];

Register register = new Register(cities,

begining, ending);

for (int i = cityCount + 4;

i < roadCount + cityCount + 4; i++)

{

string[] line = Regex.Split(allLines[i],

pattern);

Road road = new Road(line[0], line[1],

int.Parse(line[2]));

register.Add(road);

}

return register;

}

/// <summary>

/// Separately reads cities and returns the list

/// </summary>

/// <param name="cityCount"></param>

/// <param name="allLines"></param>

/// <returns></returns>

private static List<string> ReadCities(int cityCount,

string[] allLines)

{

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

for (int i = 1; i < cityCount + 1; i++)

{

cities.Add(allLines[i]);

}

return cities;

}

/// <summary>

/// Prints a table to txt file with all the cities

/// </summary>

/// <param name="fileName"></param>

/// <param name="register"></param>

public static void PrintCitiesTxt(string fileName,

Register register)

{

File.AppendAllText(fileName,

"Pradiniai duomenys:\r\n", Encoding.UTF8);

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

lines.Add(new string('-', 21));

lines.Add(String.Format($"| " +

$"{"Galimos vietovės",-17} |"));

lines.Add(new string('-', 21));

foreach (string city in register.GetCities())

{

lines.Add(String.Format($"| {city,-17} |"));

lines.Add(new string('-', 21));

}

File.AppendAllLines(fileName, lines,

Encoding.UTF8);

}

/// <summary>

/// Prints the table to txt file of all possible roads

/// </summary>

/// <param name="fileName"></param>

/// <param name="register"></param>

public static void PrintRoadsTxt(string fileName,

Register register)

{

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

lines.Add("");

lines.Add(new string('-', 43));

lines.Add(String.Format($"| {"Pradžia",-11} " +

$"| {"Pabaiga",-11} | {"Atstumas",-8} km |"));

lines.Add(new string('-', 43));

foreach (Road road in register.GetRoads())

{

lines.Add(road.ToString());

lines.Add(new string('-', 43));

}

lines.Add("");

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

}

/// <summary>

/// Prints the rezults to txt file in a table

/// </summary>

/// <param name="fileName"></param>

/// <param name="road"></param>

public static void PrintRezultTxt(string fileName,

List<Road> road)

{

File.AppendAllText(fileName, "Rezultatai:\r\n",

Encoding.UTF8);

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

lines.Add(new string('-', 43));

lines.Add(String.Format("| {0,-39} |", "Minimalus atstumas tarp vietovių"));

lines.Add(new string('-', 43));

lines.Add(String.Format($"| {"Pradžia",-11} " +

$"| {"Pabaiga",-11} | {"Atstumas",-8} km |"));

lines.Add(new string('-', 43));

lines.Add(String.Format($"| {road[0].Start,-11} " +

$"| {road[road.Count - 1].End,-11} | " +

$"{TaskUtils.Distance(road),8} km |"));

lines.Add(new string('-', 43));

lines.Add(String.Format("| {0,-39} |", "Trasa eina per vietoves"));

lines.Add(new string('-', 43));

lines.Add(String.Format($"| {road[0].Start,-39} |"));

lines.Add(new string('-', 43));

foreach (Road r in road)

{

lines.Add(String.Format($"| {r.End,-39} |"));

lines.Add(new string('-', 43));

}

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

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Web.UI.WebControls;

namespace LD22\_kelias\_tarp\_vietoviu

{

public partial class Forma1 : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

Table1.Visible = false;

Table2.Visible = false;

Button2.Visible = false;

}

protected void Button1\_Click(object sender, EventArgs e)

{

Register register =

InOut.ReadTxt(Server.MapPath("App\_Data/U3.txt"));

Table1Header();

FillTable1(register);

Button1.Visible = false;

Table1.Visible = true;

Button2.Visible = true;

List<Road> road = TaskUtils.Travel(register.start,

register.ending, register, 0);

File.Delete(Server.MapPath("Rezultatai.txt"));

InOut.PrintCitiesTxt(Server.MapPath("Rezultatai.txt"), register);

InOut.PrintRoadsTxt(Server.MapPath("Rezultatai.txt"), register);

InOut.PrintRezultTxt(Server.MapPath("Rezultatai.txt"), road);

Session["keliai"] = road;

}

protected void Button2\_Click(object sender, EventArgs e)

{

Button1.Visible = false;

List<Road> path = (List<Road>)Session["keliai"];

if (path.Count == 0)

{

TableCell cell = new TableCell();

cell.Text = "Nėra trumpiausio kelio";

TableRow row = new TableRow();

row.Cells.Add(cell);

Table2.Rows.Add(row);

}

else

{

Table2Header();

FillTable2(path);

Table2.Visible = true;

}

}

/// <summary>

/// Makes a table1 header

/// </summary>

private void Table1Header()

{

TableCell cell = new TableCell();

cell.Text = "Duomenys";

TableCell cellOne = new TableCell();

cellOne.Text = "Miestai";

TableCell cellTwo = new TableCell();

cellTwo.Text = "Keliai";

TableRow rowZero = new TableRow();

rowZero.Cells.Add(cell);

TableRow row = new TableRow();

row.Cells.Add(cellOne);

row.Cells.Add(cellTwo);

Table1.Rows.Add(rowZero);

Table1.Rows.Add(row);

}

/// <summary>

/// Makes a table2 header

/// </summary>

private void Table2Header()

{

TableCell cellOne = new TableCell();

cellOne.Text = "Rezultatai";

TableRow row = new TableRow();

row.Cells.Add(cellOne);

Table2.Rows.Add(row);

}

/// <summary>

/// Fills table1 with given parameters

/// </summary>

/// <param name="register"></param>

private void FillTable1(Register register)

{

TableCell cellOne = new TableCell();

cellOne = FillCellOne(register);

TableCell cellTwo = new TableCell();

cellTwo = FillCellTwo(register);

TableRow row = new TableRow();

row.Cells.Add(cellOne);

row.Cells.Add(cellTwo);

Table1.Rows.Add(row);

}

/// <summary>

/// Table1 cell is filled with cities

/// </summary>

/// <param name="register"></param>

/// <returns></returns>

private TableCell FillCellOne(Register register)

{

TableCell cellOne = new TableCell();

foreach (string city in register.GetCities())

{

cellOne.Text += city + "<br />";

}

return cellOne;

}

/// <summary>

/// Table1 second cell is filled with posible roads

/// </summary>

/// <param name="register"></param>

/// <returns></returns>

private TableCell FillCellTwo(Register register)

{

TableCell cellTwo = new TableCell();

foreach (Road road in register.GetRoads())

{

cellTwo.Text += road.Start + " -> "

+ road.End + " " + road.Distance + " km" + "<br />";

}

return cellTwo;

}

/// <summary>

/// Fills table2 with calculated results

/// </summary>

/// <param name="path"></param>

private void FillTable2(List<Road> path)

{

TableCell cellOne = new TableCell();

cellOne = FillCellOne2(path);

TableRow row = new TableRow();

row.Cells.Add(cellOne);

Table2.Rows.Add(row);

}

/// <summary>

/// Table2 cell is filled with results

/// </summary>

/// <param name="path"></param>

/// <returns></returns>

private TableCell FillCellOne2(List<Road> path)

{

TableCell cellOne = new TableCell();

cellOne.Text = "Minimalus atstumas tarp vietovių" + "<br />";

cellOne.Text += path[0].Start + " ir "

+ path[path.Count - 1].End + " "

+ TaskUtils.Distance(path) + " km" + "<br />";

cellOne.Text += "Trasa eina per vietoves:" + "<br />";

foreach(Road kelias in path)

{

cellOne.Text += kelias.Start + "<br />";

}

cellOne.Text += path[path.Count - 1].End;

return cellOne;

}

}

}

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Forma1.aspx.cs" Inherits="LD22\_kelias\_tarp\_vietoviu.Forma1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="Pradėti" Width="103px" />

<br />

<br />

<asp:Table ID="Table1" runat="server" BorderColor="Black" BorderStyle="Solid" BorderWidth="1px" Width="390px">

</asp:Table>

<br />

<asp:Button ID="Button2" runat="server" Text="Apskaičiuoti" OnClick="Button2\_Click" Width="107px" />

<br />

<br />

<asp:Table ID="Table2" runat="server" BorderColor="Black" BorderStyle="Solid" BorderWidth="1px" Width="390px">

</asp:Table>

</div>

</form>

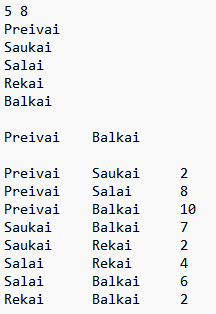
</body>

</html>

## Pradiniai duomenys ir rezultatai

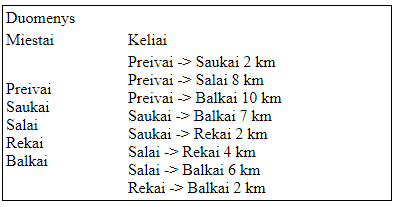
### Pradiniai duomenys ir rezultatai 1

Pradiniai duomenys:

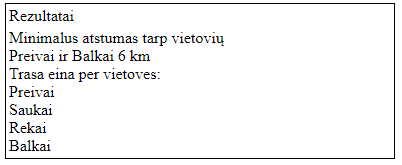


Šiais duomenimis tikrinama ar veikia programa ar teisingai yra apdorojami duomenys pritaikant rekursiją.

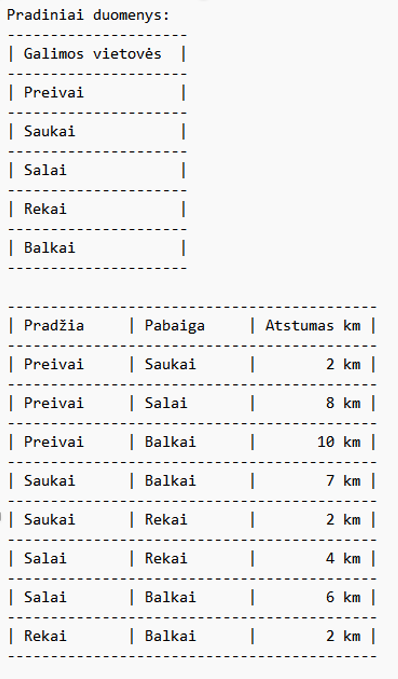
Pradiniai duomenys web:

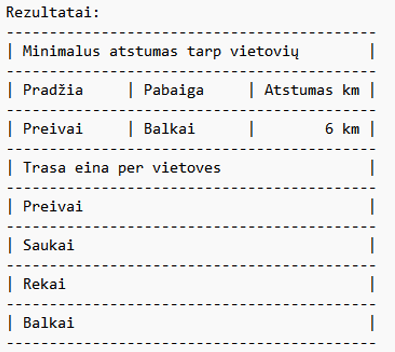


Rezultatai web:



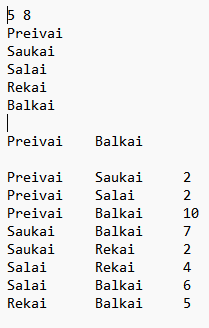
Duomenys ir rezultatai txt faile:





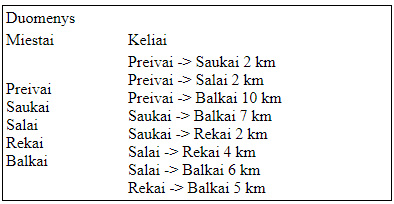
### Pradiniai duomenys ir rezultatai 2

Pradiniai duomenys:

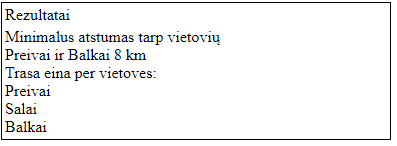


Šiais duomenimis tikrinama ar rekursija pereina per kitus variantus, o ne tik per pirmąjį.

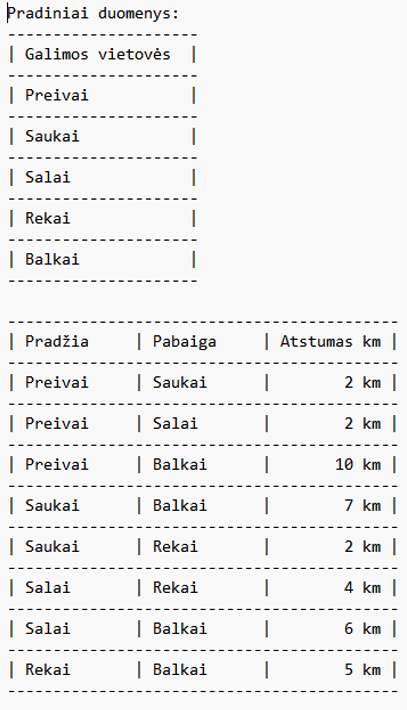
Pradiniai duomenys web:

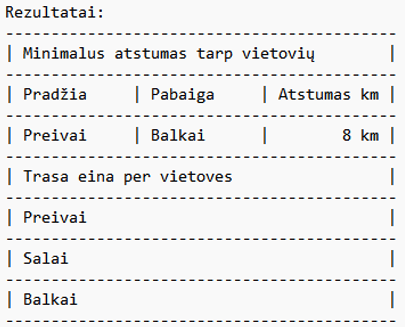


Rezultatai web:



Duomenys ir rezultatai txt faile:





## Dėstytojo pastabos

1. Ataskaitos titulinis puslapis
2. Klasių diagramoje trūksta ryšių tarp klasių
3. Du list Register klasėje

Testo rezultatas – 0

Papildomi balai – 1

Gautas įvertinimas – 7

# Dinaminis atminties valdymas (L2)

## Darbo užduotis

**LD\_22. Darbai.**

Studentai renkasi projektinių darbų temas. Už projektinių darbų temas yra atsakingi

dėstytojai. Dėstytojas gali būti atsakingas už keletą projektinių darbų temų. Sudarykite dėstytojų sąrašą

(dėstytojo pavardė ir vardas). Sąrašas turi būti surikiuotas pagal dėstytojų pavardes ir vardus abėcėlės

tvarka. Pašalinkite iš sąrašo dėstytojus, kurių siūlomų temų studentai nepasirinko. Suraskite, kuris

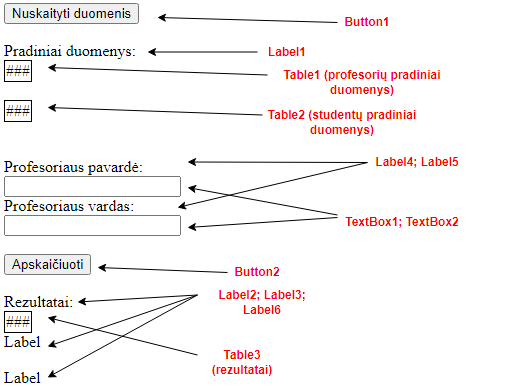
dėstytojas turi daugiausiai projektinių darbų.

Duomenys:

* tekstiniame faile U22a.txt yra informacija apie studentų pasirenkamus projektinius darbus: projektinio darbo pavadinimas, studento pavardė, vardas, grupė;
* tekstiniame faile U22b.txt yra informacija apie projektinius darbus: projektinio darbo pavadinimas, atsakingo dėstytojo pavardė ir vardas, projektiniam darbui skirtų valandų skaičius.

Sudarykite nurodyto dėstytojo (įvedama klaviatūra) projektinių darbų sąrašą.

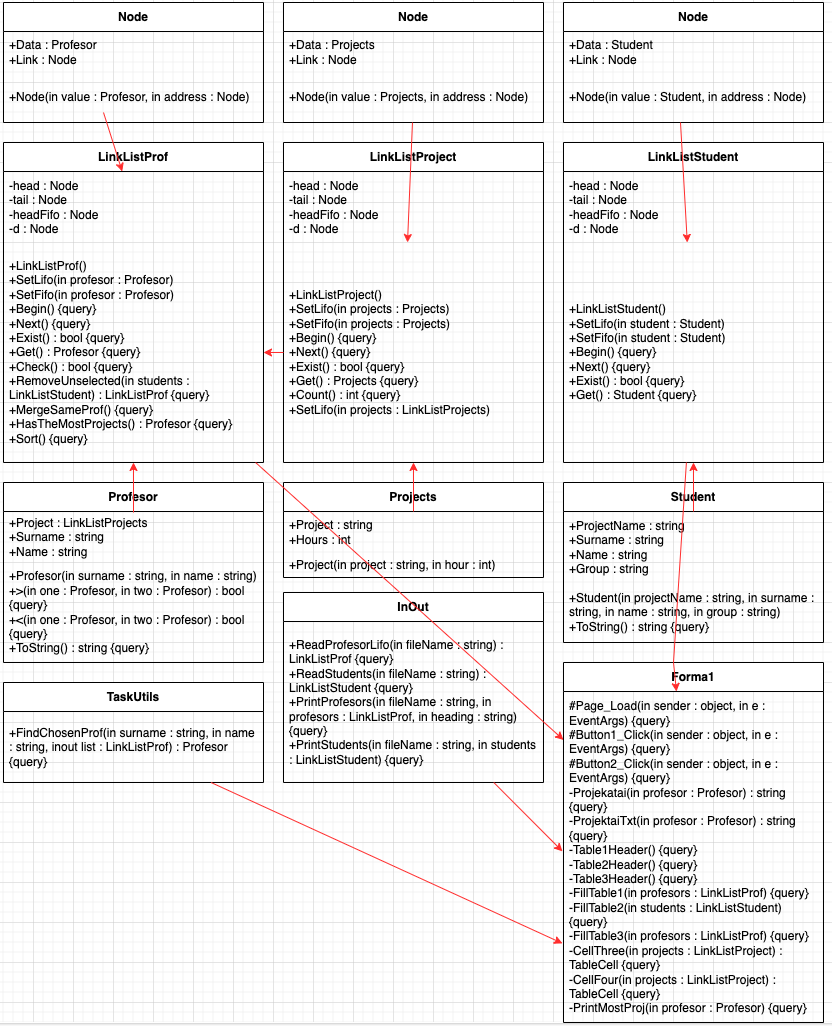
## Grafinės vartotojo sąsajos schema



## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| Button | ID | Button1 |
| Button | Text | Nuskaityti duomenis |
| Button | OnClick | Button1\_Click |
| Label | ID | Label1 |
| Label | Text | Pradiniai duomenys: |
| Table | ID | Table1 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | GridLines | Horizontal |
| Table | BorderStyle | Solid |
| Table | ID | Table2 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | GridLines | Horizontal |
| Label | ID | Label4 |
| Label | Text | Profesoriaus pavardė: |
| TextBox | ID | TextBox1 |
| TextBox | ID | TextBox2 |
| Label | ID | Label5 |
| Label | Text | Profesoriaus vardas |
| Button | ID | Button2 |
| Button | Text | Apskaičiuoti |
| Button | OnClick | Button2\_Click |
| Label | ID | Label2 |
| Label | Text | Rezultatai: |
| Table | ID | Table3 |
| Table | BorderWidth | 1px |
| Table | BorderColor | Black |
| Table | GridLines | Horizontal |
| Label | ID | Label3 |
| Label | Text | Label |
| Label | ID | Label6 |
| Label | Text | Label |

## Klasių diagrama



## Programos vartotojo vadovas

Programos darbiniame aplanke atidarome App\_Data aplanką, jame sukuriame failus U22a.txt ir U22b.txt, kuriame pateikiame duomenis apie profesorius ir studentus bei jiems priklausančius projektus.

Įjungę programą, pirmiausia užkrauname duomenų failus. Tai padarome paspausdami mygtuką „Nuskaityti duomenis“. Paspaudę mygtuką patikriname ar duomenys buvo įvesti teisingai. Jei lentelėje duomenys teisingi, spaudžiame mygtuką „Apskaičiuoti“. Paspaudus mygtuką programa apdoros duomenis ir į ekraną išves lentelę su rezultatais.

## Programos tekstas

**using** **System**;

**namespace** **\_2Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Student**

{

**public** **string** ProjectName { **get**; }

**public** **string** Surname { **get**; }

**public** **string** Name { **get**; }

**public** **string** Group { **get**; }

**public** **Student**(**string** projectName,

**string** surname, **string** name, **string** **group**)

{

**this**.ProjectName = projectName;

**this**.Surname = surname;

**this**.Name = name;

**this**.Group = **group**;

}

/// <summary>

/// ToString override for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**string** line;

line = String.Format($"| {this.Surname,-9} | " +

$"{this.Name,-9} | {this.ProjectName,-22} " +

$"| {this.Group,-11} |");

**return** line;

}

}

}

**namespace** **\_2Laboras**

{

/// <summary>

/// Linked List class

/// </summary>

**public** **sealed** **class** **LinkListStudent**

{

/// <summary>

/// Class for a Linked List Node

/// </summary>

**private** **sealed** **class** **Node**

{

**public** Student Data { **get**; **set**; }

**public** Node Link { **get**; **set**; }

**public** **Node**(Student **value**, Node address)

{

**this**.Data = **value**;

**this**.Link = address;

}

}

**private** Node head;

**private** Node tail;

**private** Node headFifo;

**private** Node d;

**public** **LinkListStudent**()

{

**this**.tail = **new** Node(**null**, **null**);

**this**.head = **new** Node(**null**, tail);

headFifo = head;

**this**.d = **null**;

}

/// <summary>

/// Adding elements to the linked list

/// (stacking, Last in first out))

/// </summary>

/// <param name="student"></param>

**public** **void** **SetLifo**(Student student)

{

head.Link = **new** Node(student, head.Link);

**if** (head == headFifo)

{

headFifo = head.Link;

}

}

/// <summary>

/// Adding elements to the linked list

/// (First in first out)

/// </summary>

/// <param name="student"></param>

**public** **void** **SetFifo**(Student student)

{

headFifo.Link = **new** Node(student, tail);

headFifo = headFifo.Link;

}

/// <summary>

/// Begining of the linked list

/// </summary>

**public** **void** **Begin**()

{

d = head.Link;

}

/// <summary>

/// For looping the next element in the

/// linked list

/// </summary>

**public** **void** **Next**()

{

d = d.Link;

}

/// <summary>

/// Checks if there is an actual element

/// </summary>

/// <returns></returns>

**public** **bool** **Exist**()

{

**return** d != **null** && d.Data != **null**;

}

/// <summary>

/// Gets the profesor

/// </summary>

/// <returns></returns>

**public** Student **Get**()

{

**return** d.Data;

}

}

}

**namespace** **\_2Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Projects**

{

**public** **string** Project { **get**; }

**public** **int** Hours { **get**; }

**public** **Projects**(**string** project, **int** hour)

{

**this**.Project = project;

**this**.Hours = hour;

}

}

}

**namespace** **\_2Laboras**

{

/// <summary>

/// Linked List class

/// </summary>

**public** **sealed** **class** **LinkListProject**

{

/// <summary>

/// Class for a Linked List Node

/// </summary>

**private** **sealed** **class** **Node**

{

**public** Projects Data { **get**; **set**; }

**public** Node Link { **get**; **set**; }

**public** **Node**(Projects **value**, Node address)

{

**this**.Data = **value**;

**this**.Link = address;

}

}

**private** Node head;

**private** Node tail;

**private** Node headFifo;

**private** Node d;

**public** **LinkListProject**()

{

**this**.tail = **new** Node(**null**, **null**);

**this**.head = **new** Node(**null**, tail);

headFifo = head;

**this**.d = **null**;

}

/// <summary>

/// Adding elements to the linked list

/// (stacking, Last in first out))

/// </summary>

/// <param name="projects"></param>

**public** **void** **SetLifo**(Projects projects)

{

head.Link = **new** Node(projects, head.Link);

**if** (head == headFifo)

{

headFifo = head.Link;

}

}

/// <summary>

/// Adding elements to the linked list

/// (First in first out)

/// </summary>

/// <param name="projects"></param>

**public** **void** **SetFifo**(Projects projects)

{

headFifo.Link = **new** Node(projects, tail);

headFifo = headFifo.Link;

}

/// <summary>

/// Begining of the linked list

/// </summary>

**public** **void** **Begin**()

{

d = head.Link;

}

/// <summary>

/// For looping the next element in the

/// linked list

/// </summary>

**public** **void** **Next**()

{

d = d.Link;

}

/// <summary>

/// Checks if there is an actual element

/// </summary>

/// <returns></returns>

**public** **bool** **Exist**()

{

**return** d != **null** && d.Data != **null**;

}

/// <summary>

/// Gets the profesor

/// </summary>

/// <returns></returns>

**public** Projects **Get**()

{

**return** d.Data;

}

/// <summary>

/// Returns the count of the linked list elements

/// </summary>

/// <returns></returns>

**public** **int** **Count**()

{

Node node = head.Link;

**int** count = **0**;

**while** (node != **null** && node.Data != **null**)

{

count++;

node = node.Link;

}

**return** count;

}

/// <summary>

/// Adds projects to the main project linked list

/// </summary>

/// <param name="projects"></param>

**public** **void** **SetLifo**(LinkListProject projects)

{

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

SetLifo(projects.Get());

}

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **\_2Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Profesor**

{

**public** LinkListProject Project { **get**; }

**public** **string** Surname { **get**; }

**public** **string** Name { **get**; }

**public** **Profesor**(**string** surname, **string** name)

{

**this**.Surname = surname;

**this**.Name = name;

**this**.Project = **new** LinkListProject();

}

/// <summary>

/// Operator overload for sorting

/// </summary>

/// <param name="one"></param>

/// <param name="two"></param>

/// <returns></returns>

**static** **public** **bool** **operator** >(Profesor one,

Profesor two)

{

**int** temp = one.Surname.CompareTo(two.Surname);

**if** (temp == **0**)

{

**if** (one.Name.CompareTo(two.Name) > **0**)

{

**return** **true**;

}

**return** **false**;

}

**else** **if** (temp > **0**)

{

**return** **true**;

}

**return** **false**;

}

/// <summary>

/// Operator overload for sorting

/// </summary>

/// <param name="one"></param>

/// <param name="two"></param>

/// <returns></returns>

**static** **public** **bool** **operator** <(Profesor one,

Profesor two)

{

**if** (one == **null**)

{

**return** **true**;

}

**int** temp = one.Surname.CompareTo(two.Surname);

**if** (temp == **0**)

{

**return** (one.Name.CompareTo(two.Name) < **0**);

}

**else** **if** (temp < **0**)

{

**return** **true**;

}

**return** **false**;

}

/// <summary>

/// ToString override for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**string** line = "";

line = String.Format($"| {this.Surname,-9} " +

$"| {this.Name,-9} |");

**return** line;

}

}

}

**namespace** **\_2Laboras**

{

/// <summary>

/// Linked List class

/// </summary>

**public** **sealed** **class** **LinkListProf**

{

/// <summary>

/// Class for a Linked List Node

/// </summary>

**private** **sealed** **class** **Node**

{

**public** Profesor Data { **get**; **set**; }

**public** Node Link { **get**; **set**; }

**public** **Node**(Profesor **value**, Node address)

{

**this**.Data = **value**;

**this**.Link = address;

}

}

**private** Node head;

**private** Node tail;

**private** Node headFifo;

**private** Node d;

**public** **LinkListProf**()

{

**this**.tail = **new** Node(**null**, **null**);

**this**.head = **new** Node(**null**, tail);

headFifo = head;

**this**.d = **null**;

}

/// <summary>

/// Adding elements to the linked list

/// (stacking, Last in first out))

/// </summary>

**public** **void** **SetLifo**(Profesor profesor)

{

head.Link = **new** Node(profesor, head.Link);

**if** (head == headFifo)

{

headFifo = head.Link;

}

}

/// <summary>

/// Adding elements to the linked list

/// (First in first out)

/// </summary>

**public** **void** **SetFifo**(Profesor profesor)

{

headFifo.Link = **new** Node(profesor, tail);

headFifo = headFifo.Link;

}

/// <summary>

/// Begining of the linked list

/// </summary>

**public** **void** **Begin**()

{

d = head.Link;

}

/// <summary>

/// For looping the next element in the

/// linked list

/// </summary>

**public** **void** **Next**()

{

d = d.Link;

}

/// <summary>

/// Checks if there is an actual element

/// </summary>

/// <returns></returns>

**public** **bool** **Exist**()

{

**return** d != **null** && d.Data != **null**;

}

/// <summary>

/// Gets the profesor

/// </summary>

/// <returns></returns>

**public** Profesor **Get**()

{

**return** d.Data;

}

/// <summary>

/// Checks if linked list is not empty

/// </summary>

/// <returns></returns>

**public** **bool** **Check**()

{

**return** headFifo.Data != **null**;

}

/// <summary>

/// Makes a new linked list only with

/// chosen projects

/// </summary>

/// <param name="students"></param>

/// <returns></returns>

**public** LinkListProf **RemoveUnselected**

(LinkListStudent students)

{

LinkListProf selectedProfesors

= **new** LinkListProf();

**for** (Node d = head.Link; d != **null**; d = d.Link)

{

**for** (students.Begin();

students.Exist(); students.Next())

{

**if** (d.Data != **null**

&& d.Data.Project.Get().Project

== students.Get().ProjectName)

{

selectedProfesors.SetLifo(d.Data);

**break**;

}

}

}

**return** selectedProfesors;

}

/// <summary>

/// Merges projects if the same professor is

/// responsible for them

/// </summary>

**public** **void** **MergeSameProf**()

{

**for** (Node d = head.Link; d != **null**; d = d.Link)

{

Node pj = d;

**for** (Node j = d.Link; j != **null**; j = j.Link)

{

**if** (d.Data != **null** && j.Data != **null**

&& d.Data.Name == j.Data.Name

&& d.Data.Surname == j.Data.Surname)

{

d.Data.Project.SetLifo(j.Data.Project);

pj.Link = j.Link;

}

**else**

{

pj = j;

}

}

}

}

/// <summary>

/// Returns the professor which has the most projects

/// </summary>

/// <returns></returns>

**public** Profesor **HasTheMostProjects**()

{

**int** count = **0**;

Profesor profesor = **null**;

**for** (Node d = head; d != **null**; d = d.Link)

{

**if** (d.Data != **null** &&

d.Data.Project.Count() > count)

{

count = d.Data.Project.Count();

profesor = d.Data;

}

}

**return** profesor;

}

/// <summary>

/// Sorts professors by surname and name

/// </summary>

**public** **void** **Sort**()

{

**for** (Node d1 = head.Link; d1 != **null**; d1 = d1.Link)

{

**if** (d1.Data == **null**)

{

**continue**;

}

Node minv = d1;

**for** (Node d2 = d1.Link; d2 != **null**; d2 = d2.Link)

{

**if** (d2.Data != **null** && d2.Data < minv.Data)

{

minv = d2;

}

}

Profesor profesor = d1.Data;

d1.Data = minv.Data;

minv.Data = profesor;

}

}

}

}

**namespace** **\_2Laboras**

{

/// <summary>

/// Class for calculations

/// </summary>

**public** **static** **class** **TaskUtils**

{

/// <summary>

/// Finds the requested professor

/// </summary>

/// <param name="surname"></param>

/// <param name="name"></param>

/// <param name="list"></param>

/// <returns></returns>

**public** **static** Profesor **FindChosenProf**(**string** surname,

**string** name, LinkListProf list)

{

Profesor profesor = **null**;

**if** (list != **null**)

{

**for** (list.Begin(); list.Exist(); list.Next())

{

**if** (list.Get() != **null** && surname

== list.Get().Surname && name

== list.Get().Name)

{

profesor = list.Get();

**break**;

}

}

}

**return** profesor;

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.IO**;

**namespace** **\_2Laboras**

{

/// <summary>

/// Reading and printing class

/// </summary>

**public** **class** **InOut**

{

/// <summary>

/// Reads professors and their info from the file

/// </summary>

/// <param name="fileName"></param>

/// <returns></returns>

**public** **static** LinkListProf **ReadProfesorLifo**(**string** fileName)

{

**string** projectName;

**string** surname;

**string** name;

**int** hours;

**string** line;

LinkListProf list = **new** LinkListProf();

**using** (**var** file =

**new** System.IO.StreamReader(fileName,

Encoding.UTF8))

{

**while** ((line = file.ReadLine()) != **null**)

{

**string**[] values = line.Split(**new** **char**[] { ';' },

StringSplitOptions.RemoveEmptyEntries);

projectName = values[**0**];

**string**[] strings = values[**1**].Split(**new** **char**[] { ' ' },

StringSplitOptions.RemoveEmptyEntries);

surname = strings[**0**];

name = strings[**1**];

hours = **int**.Parse(strings[**2**]);

Profesor profesor = **new** Profesor(surname, name);

Projects projects = **new** Projects(projectName, hours);

profesor.Project.SetLifo(projects);

list.SetLifo(profesor);

}

}

**return** list;

}

**public** **static** LinkListStudent **ReadStudentLifo**(**string** fileName)

{

**string** projectName;

**string** surname;

**string** name;

**string** **group**;

**string** line;

LinkListStudent list = **new** LinkListStudent();

**using** (**var** file =

**new** System.IO.StreamReader(fileName, Encoding.UTF8))

{

**while** ((line = file.ReadLine()) != **null**)

{

**string**[] values = line.Split(**new** **char**[] { ';' },

StringSplitOptions.RemoveEmptyEntries);

projectName = values[**0**];

**string**[] strings = values[**1**].Split(**new** **char**[] { ' ' },

StringSplitOptions.RemoveEmptyEntries);

surname = strings[**0**];

name = strings[**1**];

**group** = strings[**2**];

Student profesor = **new** Student(projectName,

surname, name, **group**);

list.SetLifo(profesor);

}

}

**return** list;

}

/// <summary>

/// Prints professors with their info to the file in a table

/// </summary>

/// <param name="fileName"></param>

/// <param name="profesors"></param>

/// <param name="heading"></param>

**public** **static** **void** **PrintProfesors**(**string** fileName,

LinkListProf profesors, **string** heading)

{

File.AppendAllText(fileName, $"{heading}\r\n", Encoding.UTF8);

List<**string**> lines = **new** List<**string**>();

lines.Add(**new** **string**('-', **64**));

lines.Add(String.Format($"| {"Pavardė",-9} | {"Vardas",-9} " +

$"| {"Projekto pavadinimas",-22} | {"Valandų sk.",-11} |"));

lines.Add(**new** **string**('-', **64**));

**for** (profesors.Begin(); profesors.Exist(); profesors.Next())

{

profesors.Get().Project.Begin();

lines.Add(profesors.Get().ToString() + " " +

$"{profesors.Get().Project.Get().Project,-22} " +

$"| {profesors.Get().Project.Get().Hours,11} |");

**for** (profesors.Get().Project.Next();

profesors.Get().Project.Exist();

profesors.Get().Project.Next())

{

lines.Add(String.Format($"| {"",-9} " +

$"| {"",-9} | " +

$"{profesors.Get().Project.Get().Project,-22} | " +

$"{profesors.Get().Project.Get().Hours,11} |"));

}

lines.Add(**new** **string**('-', **64**));

}

lines.Add("");

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

}

/// <summary>

/// Print students with their info to the file in a table

/// </summary>

/// <param name="fileName"></param>

/// <param name="students"></param>

**public** **static** **void** **PrintStudents**(**string** fileName,

LinkListStudent students)

{

List<**string**> lines = **new** List<**string**>();

lines.Add(**new** **string**('-', **64**));

lines.Add(String.Format($"| {"Pavardė",-9} | {"Vardas",-9} " +

$"| {"Projekto pavadinimas",-22} | {"Grupė",-11} |"));

lines.Add(**new** **string**('-', **64**));

**for** (students.Begin(); students.Exist(); students.Next())

{

**if** (students.Get() != **null**)

{

lines.Add(students.Get().ToString());

lines.Add(**new** **string**('-', **64**));

}

}

lines.Add("");

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

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.IO**;

**using** **System.Web.UI**;

**using** **System.Web.UI.WebControls**;

**namespace** **\_2Laboras**

{

**public** **partial** **class** **Forma1** : System.Web.UI.Page

{

**protected** **void** **Page\_Load**(**object** sender, EventArgs e)

{

**if** (Page.IsPostBack)

{

Profesor profesor1 = **null**;

LinkListProf list = (LinkListProf)Session["profesoriai"];

profesor1 = TaskUtils.FindChosenProf(TextBox1.Text,

TextBox2.Text, list);

Session["pasirinktas"] = profesor1;

}

Label1.Visible = **false**;

Label2.Visible = **false**;

Label3.Visible = **false**;

Table1.Visible = **false**;

Table2.Visible = **false**;

Table3.Visible = **false**;

Button2.Visible = **false**;

Label4.Visible = **false**;

Label5.Visible = **false**;

Label6.Visible = **false**;

TextBox1.Visible = **false**;

TextBox2.Visible = **false**;

}

**protected** **void** **Button1\_Click**(**object** sender,

EventArgs e)

{

LinkListProf profesors =

InOut.ReadProfesorLifo(Server.MapPath("App\_Data/U22b.txt"));

LinkListStudent students =

InOut.ReadStudentLifo(Server.MapPath("App\_Data/U22a.txt"));

File.Delete(Server.MapPath("Rezultatai.txt"));

InOut.PrintProfesors(Server.MapPath("Rezultatai.txt"),

profesors, "Pradiniai duomenys:");

InOut.PrintStudents(Server.MapPath("Rezultatai.txt"),

students);

Table1Header();

Table2Header();

FillTable1(profesors);

FillTable2(students);

Button1.Visible = **false**;

Label1.Visible = **true**;

Label4.Visible = **true**;

Label5.Visible = **true**;

Table1.Visible = **true**;

Table2.Visible = **true**;

Button2.Visible = **true**;

TextBox1.Visible = **true**;

TextBox2.Visible = **true**;

**if** (profesors.Check())

{

profesors = profesors.RemoveUnselected(students);

}

profesors.MergeSameProf();

profesors.Sort();

Profesor profesor = profesors.HasTheMostProjects();

InOut.PrintProfesors(Server.MapPath("Rezultatai.txt"),

profesors, "Rezultatai:");

**if** (profesor != **null**)

{

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"Daugiausiai projektų turintis dėstytojas: " +

$"{profesor.Surname} {profesor.Name} " +

$"({profesor.Project.Count()})\r\n");

}

**else**

{

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"Nėra profesoriaus su daugiausiai projektų\r\n");

}

Session["profesoriai"] = profesors;

Session["daugiausiai"] = profesor;

}

**protected** **void** **Button2\_Click**(**object** sender, EventArgs e)

{

Profesor profesor = (Profesor)Session["daugiausiai"];

LinkListProf profesors = (LinkListProf)Session["profesoriai"];

Profesor profesor1 = (Profesor)Session["pasirinktas"];

Label2.Visible = **true**;

Label3.Visible = **true**;

Table3.Visible = **true**;

Label6.Visible = **true**;

Table3Header();

FillTable3(profesors);

PrintMostProj(profesor);

**if** (profesor1 == **null**)

{

Label6.Text = "Pasirinktas profesorius neegzistuoja";

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"\r\nPasirinktas profesorius neegzistuoja");

}

**else**

{

Label6.Text = $"Pasirinktas profesorius: " +

$"{profesor1.Surname} {profesor1.Name} " +

$"<br />Projektai:<br />{Projektai(profesor1)}";

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"\r\nPasirinktas profesorius: {profesor1.Surname} " +

$"{profesor1.Name} \r\nProjektai:\r\n{ProjektaiTxt(profesor1)}");

}

}

/// <summary>

/// For project printing

/// </summary>

/// <param name="profesor"></param>

/// <returns></returns>

**private** **string** **Projektai**(Profesor profesor)

{

**string** pro = "";

LinkListProject projects = profesor.Project;

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

pro += projects.Get().Project + "<br />";

}

}

**return** pro;

}

/// <summary>

/// For project printing to the file

/// </summary>

/// <param name="profesor"></param>

/// <returns></returns>

**private** **string** **ProjektaiTxt**(Profesor profesor)

{

**string** pro = "";

LinkListProject projects = profesor.Project;

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

pro += projects.Get().Project + "\r\n";

}

}

**return** pro;

}

/// <summary>

/// Makes table1 header

/// </summary>

**private** **void** **Table1Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Profesoriai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Projektas ";

TableCell four = **new** TableCell();

four.Text = "Valandų sk.";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table1.Rows.Add(row);

Table1.Rows.Add(row2);

}

/// <summary>

/// Makes table2 header

/// </summary>

**private** **void** **Table2Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Studentai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Grupė ";

TableCell four = **new** TableCell();

four.Text = "Projektas";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table2.Rows.Add(row);

Table2.Rows.Add(row2);

}

/// <summary>

/// Makes table3 header

/// </summary>

**private** **void** **Table3Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Profesoriai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Projektas ";

TableCell four = **new** TableCell();

four.Text = "Valandų sk.";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table3.Rows.Add(row);

Table3.Rows.Add(row2);

}

/// <summary>

/// Fills table1

/// </summary>

/// <param name="profesors"></param>

**private** **void** **FillTable1**(LinkListProf profesors)

{

**for** (profesors.Begin(); profesors.Exist();

profesors.Next())

{

profesors.Get().Project.Begin();

TableCell one = **new** TableCell();

one.Text = profesors.Get().Surname;

TableCell two = **new** TableCell();

two.Text = profesors.Get().Name;

TableCell three = **new** TableCell();

three.Text = profesors.Get().Project.Get().Project;

TableCell four = **new** TableCell();

four.Text = profesors.Get().Project.Get().Hours.ToString();

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table1.Rows.Add(row);

}

}

/// <summary>

/// Fills table2

/// </summary>

/// <param name="students"></param>

**private** **void** **FillTable2**(LinkListStudent students)

{

**for** (students.Begin(); students.Exist(); students.Next())

{

**if** (students.Get() != **null**)

{

TableCell one = **new** TableCell();

one.Text = students.Get().Surname;

TableCell two = **new** TableCell();

two.Text = students.Get().Name;

TableCell three = **new** TableCell();

three.Text = students.Get().Group;

TableCell four = **new** TableCell();

four.Text = students.Get().ProjectName;

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table2.Rows.Add(row);

}

}

}

/// <summary>

/// Fills table3

/// </summary>

/// <param name="profesors"></param>

**private** **void** **FillTable3**(LinkListProf profesors)

{

**for** (profesors.Begin(); profesors.Exist();

profesors.Next())

{

**if** (profesors.Get() != **null**)

{

TableCell one = **new** TableCell();

one.Text = profesors.Get().Surname;

TableCell two = **new** TableCell();

two.Text = profesors.Get().Name;

TableCell three = **new** TableCell();

three = CellThree(profesors.Get().Project);

TableCell four = **new** TableCell();

four = CellFour(profesors.Get().Project);

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table3.Rows.Add(row);

}

}

}

/// <summary>

/// Fills table cell with projects

/// </summary>

/// <param name="projects"></param>

/// <returns></returns>

**private** TableCell **CellThree**(LinkListProject projects)

{

TableCell three = **new** TableCell();

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

three.Text += projects.Get().Project + "<br />";

}

}

**return** three;

}

/// <summary>

/// Fills table cell with hours

/// </summary>

/// <param name="projects"></param>

/// <returns></returns>

**private** TableCell **CellFour**(LinkListProject projects)

{

TableCell four = **new** TableCell();

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

four.Text += projects.Get().Hours + "<br />";

}

}

**return** four;

}

/// <summary>

/// Prints the professor which has most projects

/// </summary>

/// <param name="profesor"></param>

**private** **void** **PrintMostProj**(Profesor profesor)

{

**if** (profesor != **null**)

{

Label3.Text = $"Daugiausiai projektų " +

$"turintis profesorius: {profesor.Surname} " +

$"{profesor.Name} ({profesor.Project.Count()})";

}

**else**

{

Label3.Text = "Nėra profesoriaus su daugiausiai " +

"projektų";

}

}

}

}

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Forma1.aspx.cs" Inherits="\_2Laboras.Forma1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Button ID="Button1" runat="server" Text="Nuskaityti duomenis" OnClick="Button1\_Click" />

<br />

<br />

<asp:Label ID="Label1" runat="server" Text="Pradiniai duomenys:"></asp:Label>

<asp:Table ID="Table1" runat="server" BorderColor="Black" BorderWidth="1px" GridLines="Horizontal" BorderStyle="Solid"></asp:Table>

<br />

<asp:Table ID="Table2" runat="server" BorderColor="Black" BorderWidth="1px" GridLines="Horizontal"></asp:Table>

<br />

<br />

<asp:Label ID="Label4" runat="server" Text="Profesoriaus pavardė:"></asp:Label>

<br />

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<br />

<asp:Label ID="Label5" runat="server" Text="Profesoriaus vardas:"></asp:Label>

<br />

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<br />

<br />

<asp:Button ID="Button2" runat="server" Text="Apskaičiuoti" OnClick="Button2\_Click" />

<br />

<br />

<asp:Label ID="Label2" runat="server" Text="Rezultatai:"></asp:Label>

<asp:Table ID="Table3" runat="server" BorderWidth="1px" BorderColor="Black" GridLines="Horizontal"></asp:Table>

<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>

<br />

<br />

<asp:Label ID="Label6" runat="server" Text="Label"></asp:Label>

</div>

</form>

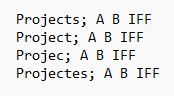
</body>

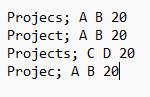
</html>

## Pradiniai duomenys ir rezultatai

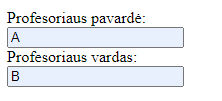
### Duomenys ir rezultatai 1

Pradiniai duomenys:



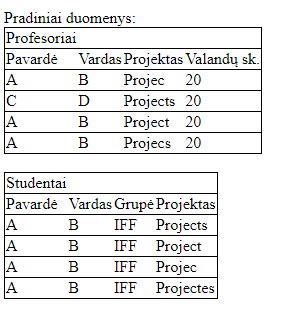


Įvedami duomenys:

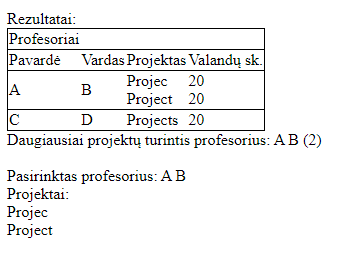


Šiais duomenimis tikrinama ar veikia profesorių, kurių projektai buvo nepasirinkti pašalinimas. Taip pat tikrinama ar veikia profesorių, kurie turi daugiau projektų suliejimas. Tikrinama ar randamas teisingas daugiausiai projektų turintis dėstytojas. Tikrinama ar atspausdinamas pasirinktas profesorius.

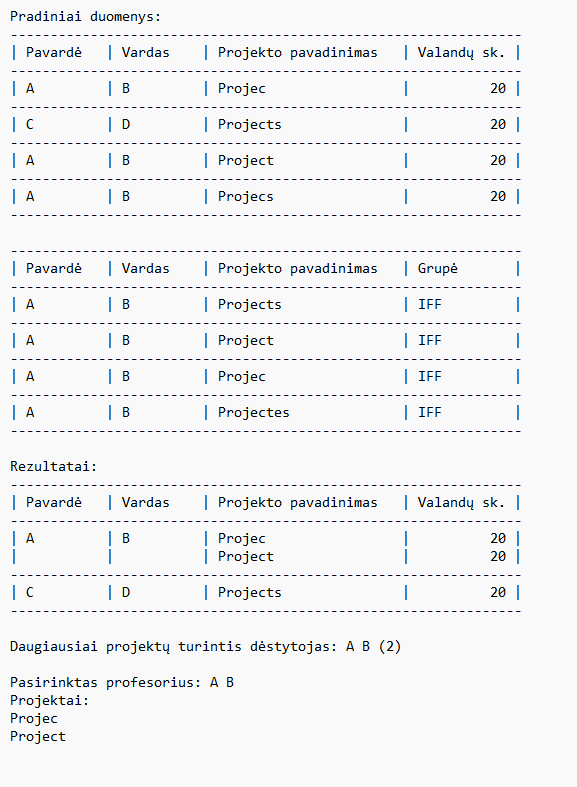
Pradiniai duomenys web:



Rezultatai web:

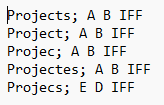


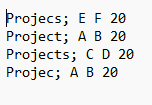
Rezultatų tekstinis failas:



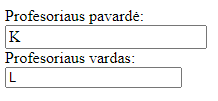
### Duomenys ir rezultatai 2

Pradiniai duomenys:



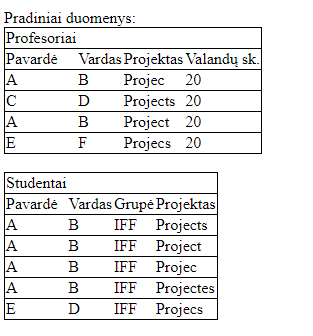


Įvedami duomenys:

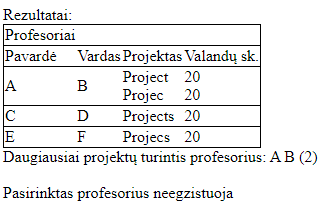


Šiais duomenimis tikrinama, kas yra atspausdinama, jei pasirenkamas neegzistuojantis dėstytojas. Tikrinama ar tikrai veikia rūšiavimas.

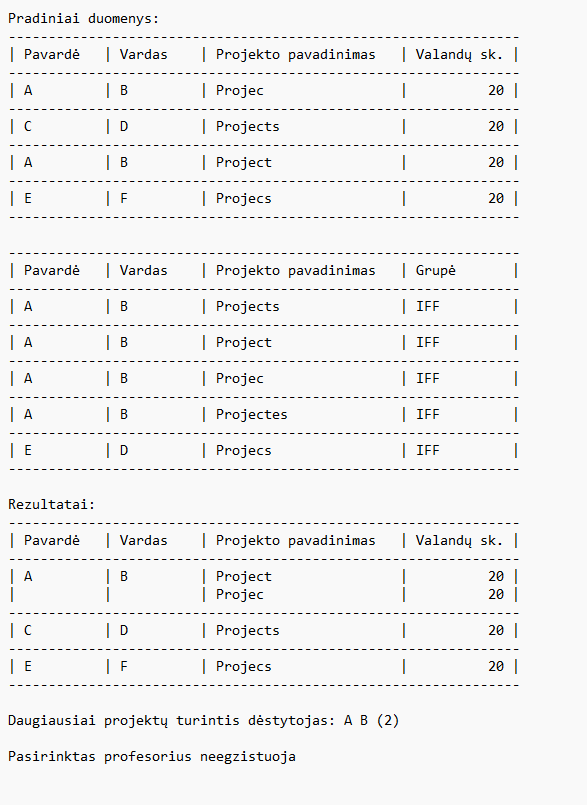
Pradiniai duomenys web:



Rezultatai web:



Rezultatų tekstinis failas:



## Dėstytojo pastabos

1. Tituliniame puslapyje skliausteliai – pataisyta
2. LD1 pastabos neįdėtos – pataisyta
3. Per mažai klasių – pataisyta
4. Neturi būti List – pataisyta
5. LinkList list TaskUtils klasėje – pataisyta
6. Projektinių darbų sąrašas irgi turi būti susietas sąrašas – pataisyta
7. Ryšiai tarp klasių neteisingi
8. Raudona spalva naudojama klaidoms žymėti
9. Negali būti d1.Data == null

Testo balai – 2

Gautas įvertinimas – 8

# Bendrinės klasės ir testavimas (L3)

## Darbo užduotis

**LD\_22. Darbai.**

Studentai renkasi projektinių darbų temas. Už projektinių darbų temas yra atsakingi

dėstytojai. Dėstytojas gali būti atsakingas už keletą projektinių darbų temų. Sudarykite dėstytojų sąrašą

(dėstytojo pavardė ir vardas). Sąrašas turi būti surikiuotas pagal dėstytojų pavardes ir vardus abėcėlės

tvarka. Pašalinkite iš sąrašo dėstytojus, kurių siūlomų temų studentai nepasirinko. Suraskite, kuris

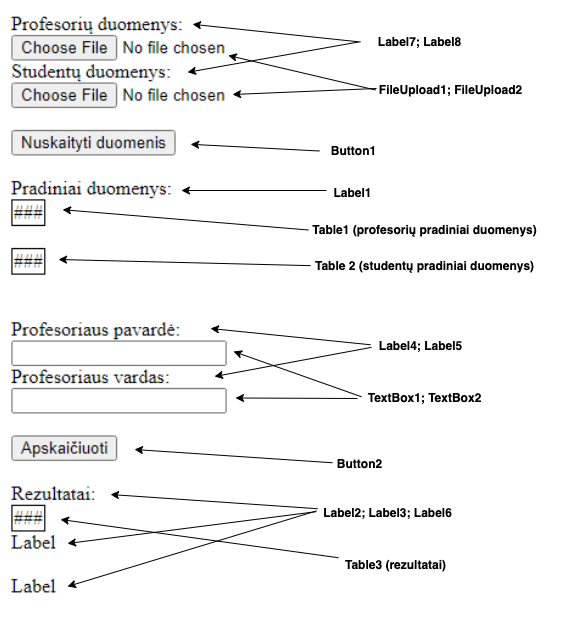
dėstytojas turi daugiausiai projektinių darbų.

Duomenys:

* tekstiniame faile U22a.txt yra informacija apie studentų pasirenkamus projektinius darbus: projektinio darbo pavadinimas, studento pavardė, vardas, grupė;
* tekstiniame faile U22b.txt yra informacija apie projektinius darbus: projektinio darbo pavadinimas, atsakingo dėstytojo pavardė ir vardas, projektiniam darbui skirtų valandų skaičius.

Sudarykite nurodyto dėstytojo (įvedama klaviatūra) projektinių darbų sąrašą.

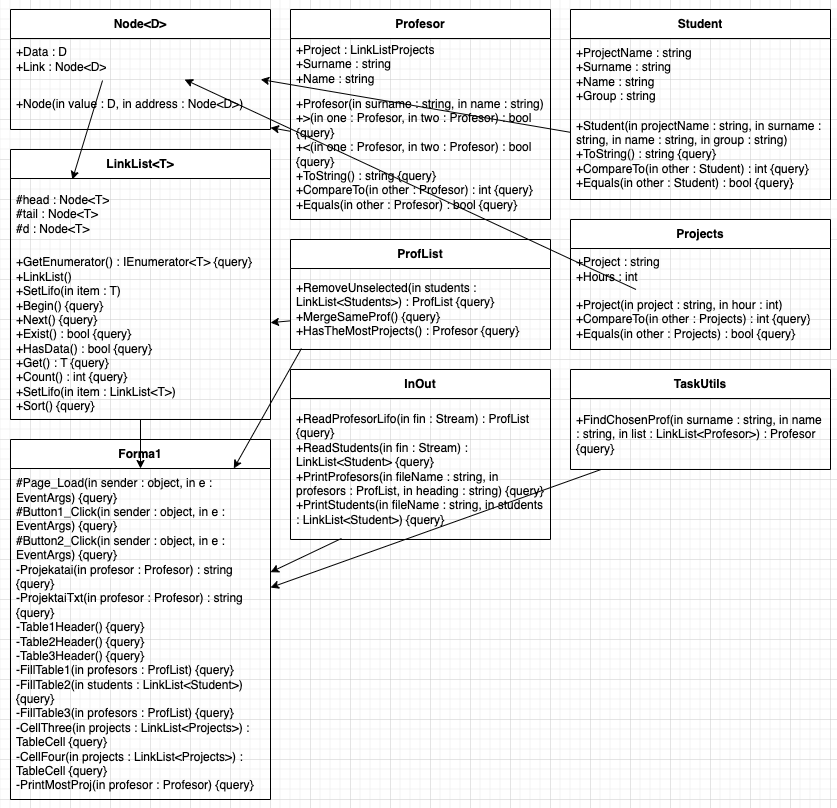
## Grafinės vartotojo sąsajos schema



## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| Button | ID | Button1 |
| Button | Text | Nuskaityti duomenis |
| Button | OnClick | Button1\_Click |
| Label | ID | Label1 |
| Label | Text | Pradiniai duomenys: |
| Table | ID | Table1 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | GridLines | Horizontal |
| Table | BorderStyle | Solid |
| Table | ID | Table2 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | GridLines | Horizontal |
| Label | ID | Label4 |
| Label | Text | Profesoriaus pavardė: |
| TextBox | ID | TextBox1 |
| TextBox | ID | TextBox2 |
| Label | ID | Label5 |
| Label | Text | Profesoriaus vardas |
| Button | ID | Button2 |
| Button | Text | Apskaičiuoti |
| Button | OnClick | Button2\_Click |
| Label | ID | Label2 |
| Label | Text | Rezultatai: |
| Table | ID | Table3 |
| Table | BorderWidth | 1px |
| Table | BorderColor | Black |
| Table | GridLines | Horizontal |
| Label | ID | Label3 |
| Label | Text | Label |
| Label | ID | Label6 |
| Label | Text | Label |
| Label | ID | Label7 |
| Label | Text | Profesorių duomenys |
| Label | ID | Label8 |
| Label | Text | Studentų duomenys |
| FileUpload | ID | FileUpload1 |
| FileUpload | ID | FileUpload2 |

## Klasių diagrama



## Programos vartotojo vadovas

Susikuriame du duomenų failus, kuriuose pateikiame duomenis apie profesorius ir studentus bei jiems priklausančius projektus.

Įjungę programą, pirmiausia užkrauname duomenų failus paspaude „FileUpload“ mygtukus. Tuomet nuskaitome visus duomenis - tai padarome paspausdami mygtuką „Nuskaityti duomenis“. Paspaudę mygtuką patikriname ar duomenys buvo nuskaityti teisingai. Jei lentelėse duomenys teisingi, spaudžiame mygtuką „Apskaičiuoti“. Paspaudus mygtuką programa apdoros duomenis ir į ekraną išves lentelę su rezultatais.

## Programos tekstas

**using** **System**;

**namespace** **\_3Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Student** :

IComparable<Student>, IEquatable<Student>

{

**public** **string** ProjectName { **get**; }

**public** **string** Surname { **get**; }

**public** **string** Name { **get**; }

**public** **string** Group { **get**; }

**public** **Student**(**string** projectName,

**string** surname, **string** name, **string** **group**)

{

**this**.ProjectName = projectName;

**this**.Surname = surname;

**this**.Name = name;

**this**.Group = **group**;

}

/// <summary>

/// ToString override for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**string** line;

line = String.Format($"| {this.Surname,-9} | " +

$"{this.Name,-9} | {this.ProjectName,-22} " +

$"| {this.Group,-11} |");

**return** line;

}

/// <summary>

/// Compares students by their surnames and names

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

**public** **int** **CompareTo**(Student other)

{

**if** (**this**.Surname.CompareTo(other.Surname) != **0**)

{

**return** **this**.Surname.CompareTo(other.Surname);

}

**return** **this**.Name.CompareTo(other.Name);

}

/// <summary>

/// Checks if students names and surnames are equal

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

**public** **bool** **Equals**(Student other)

{

**return** **this**.Name.Equals(other.Name)

&& **this**.Surname.Equals(other.Surname);

}

}

}

**using** **System**;

**namespace** **\_3Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Projects** :

IComparable<Projects>, IEquatable<Projects>

{

**public** **string** Project { **get**; }

**public** **int** Hours { **get**; }

**public** **Projects**(**string** project, **int** hour)

{

**this**.Project = project;

**this**.Hours = hour;

}

/// <summary>

/// Compares two project names

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

**public** **int** **CompareTo**(Projects other)

{

**return** **this**.Project.CompareTo(other.Project);

}

/// <summary>

/// Checks if one project name is equal to the other

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

**public** **bool** **Equals**(Projects other)

{

**return** **this**.Project.Equals(other.Project);

}

}

}

**using** **System**;

**namespace** **\_3Laboras**

{

/// <summary>

/// Constructor class

/// </summary>

**public** **class** **Profesor** :

IComparable<Profesor>, IEquatable<Profesor>

{

**public** LinkList<Projects> Project { **get**; }

**public** **string** Surname { **get**; }

**public** **string** Name { **get**; }

**public** **Profesor**(**string** surname, **string** name)

{

**this**.Surname = surname;

**this**.Name = name;

**this**.Project = **new** LinkList<Projects>();

}

/// <summary>

/// Operator overloading

/// </summary>

/// <param name="one"></param>

/// <param name="two"></param>

/// <returns></returns>

**static** **public** **bool** **operator** >(

Profesor one,

Profesor two)

{

**return** one.CompareTo(two) == **1**;

}

/// <summary>

/// Operator overloading

/// </summary>

/// <param name="one"></param>

/// <param name="two"></param>

/// <returns></returns>

**static** **public** **bool** **operator** <(

Profesor one,

Profesor two)

{

**return** one.CompareTo(two) == -**1**;

}

/// <summary>

/// Overriding ToString method for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**string** line;

line = String.Format($"| {this.Surname,-9} " +

$"| {this.Name,-9} |");

**return** line;

}

/// <summary>

/// Compares two profesors by surname and name

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **int** **CompareTo**(Profesor other)

{

**if** ((**object**)other == **null**)

{

**return** **1**;

}

**if** (Surname.CompareTo(other.Surname) != **0**)

{

**return** Surname.CompareTo(other.Surname);

}

**else**

{

**return** Name.CompareTo(other.Name);

}

}

/// <summary>

/// Checks if profesors names

/// and surnames are equal

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **bool** **Equals**(Profesor other)

{

**return** **this**.Name.Equals(other.Name)

&& **this**.Surname.Equals(other.Surname);

}

}

}

**namespace** **\_3Laboras**

{

/// <summary>

/// Profesors list class

/// </summary>

**public** **class** **ProfList** : LinkList<Profesor>

{

/// <summary>

/// Makes a new list only with chosen projects

/// </summary>

/// <param name="students"></param>

/// <returns></returns>

**public** ProfList **RemoveUnselected**

(LinkList<Student> students)

{

ProfList selectedProfesors = **new** ProfList();

**for** (Node<Profesor> d = head; d != **null**; d = d.Link)

{

**for** (students.Begin();

students.Exist(); students.Next())

{

**if** (d.Data.Project.Get().Project

== students.Get().ProjectName)

{

selectedProfesors.SetLifo(d.Data);

**break**;

}

}

}

**return** selectedProfesors;

}

/// <summary>

/// Merges projects if the same profesor

/// is responsible for them

/// </summary>

**public** **void** **MergeSameProf**()

{

**for** (Node<Profesor> d = head; d != **null**; d = d.Link)

{

Node<Profesor> pj = d;

**for** (Node<Profesor> j = d.Link;

j != **null**; j = j.Link)

{

**if** (d.Data.Equals(j.Data))

{

d.Data.Project.SetLifo(j.Data.Project);

pj.Link = j.Link;

}

**else**

{

pj = j;

}

}

}

}

/// <summary>

/// Returns the profesor which has the most projects

/// </summary>

/// <returns></returns>

**public** Profesor **HasTheMostProjects**()

{

**int** count = **0**;

Profesor profesor = **null**;

**for** (Node<Profesor> d = head; d != **null**; d = d.Link)

{

**if** (d.Data.Project.Count() > count)

{

count = d.Data.Project.Count();

profesor = d.Data;

}

}

**return** profesor;

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Collections**;

**namespace** **\_3Laboras**

{

**public** **class** **LinkList**<T> : IEnumerable<T>

**where** T : IComparable<T>, IEquatable<T>

{

/// <summary>

/// Gets enumerator

/// </summary>

/// <returns></returns>

**public** IEnumerator<T> GetEnumerator()

{

**for** (Node<T> dd = head; dd != **null**; dd = dd.Link)

{

**yield** **return** dd.Data;

}

}

/// <summary>

/// Obligatory, since IEnumerable<T>

/// inherits IEnumerable

/// </summary>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

IEnumerator IEnumerable.GetEnumerator()

{

**throw** **new** **NotImplementedException**();

}

/// <summary>

/// Class for the LinkList node

/// </summary>

/// <typeparam name="D"></typeparam>

**protected** **sealed** **class** **Node**<D>

{

**public** D Data { **get**; **set**; }

**public** Node<D> Link { **get**; **set**; }

**public** **Node**(D **value**, Node<D> address)

{

**this**.Data = **value**;

**this**.Link = address;

}

}

**protected** Node<T> head;

**protected** Node<T> tail;

**protected** Node<T> d;

**public** **LinkList**()

{

**this**.tail = **null**;

**this**.head = **null**;

**this**.d = **null**;

}

/// <summary>

/// Adding elements to the LinkList

/// (LIFO - last in first out)

/// </summary>

/// <param name="item"></param>

**public** **void** **SetLifo**(T item)

{

**if** (item == **null**)

{

**throw** **new** **ArgumentNullException**();

}

head = **new** Node<T>(item, head);

}

/// <summary>

/// Begining of the LinkList

/// </summary>

**public** **void** **Begin**()

{

**if** (head == **null**)

{

**throw** **new** **InvalidOperationException**("Add data");

}

d = head;

}

/// <summary>

/// Gets the next element in the LinkList

/// </summary>

**public** **void** **Next**()

{

**if** (d == **null**)

{

**throw** **new** **InvalidOperationException**("Call Begin()");

}

d = d.Link;

}

/// <summary>

/// Checks if LinkList exists

/// </summary>

/// <returns></returns>

**public** **bool** **Exist**()

{

**return** d != **null**;

}

/// <summary>

/// Checks if LinkList has any items

/// </summary>

/// <returns></returns>

**public** **bool** **HasData**()

{

**return** head != **null**;

}

/// <summary>

/// Gets the item form the LinkList

/// </summary>

/// <returns></returns>

**public** T **Get**()

{

**if** (d == **null**)

{

**throw** **new** **InvalidOperationException**("No data found");

}

**return** d.Data;

}

/// <summary>

/// Returns the count of the LinkList elements

/// </summary>

/// <returns></returns>

**public** **int** **Count**()

{

Node<T> node = head;

**int** count = **0**;

**while** (node != **null** && node.Data != **null**)

{

count++;

node = node.Link;

}

**return** count;

}

/// <summary>

/// Adds an object to the main objects LinkList

/// </summary>

/// <param name="item"></param>

**public** **void** **SetLifo**(LinkList<T> item)

{

**if** (item == **null**)

{

**throw** **new** **ArgumentNullException**("Cannot " +

"add empty list");

}

**for** (item.Begin(); item.Exist(); item.Next())

{

SetLifo(item.Get());

}

}

/// <summary>

/// Sorts the LinkList

/// </summary>

**public** **void** **Sort**()

{

**if** (head == **null**)

{

**throw** **new** **InvalidOperationException**("Cannot " +

"sort empty list");

}

**for** (Node<T> d1 = head; d1 != **null**; d1 = d1.Link)

{

Node<T> minv = d1;

**for** (Node<T> d2 = d1; d2 != **null**; d2 = d2.Link)

{

**if** (d2.Data.CompareTo(minv.Data) < **0**)

{

minv = d2;

}

}

T item = d1.Data;

d1.Data = minv.Data;

minv.Data = item;

}

}

}

}

**namespace** **\_3Laboras**

{

/// <summary>

/// Class for calculations

/// </summary>

**public** **static** **class** **TaskUtils**

{

/// <summary>

/// Finds the requested professor

/// </summary>

/// <param name="surname"></param>

/// <param name="name"></param>

/// <param name="list"></param>

/// <returns></returns>

**public** **static** Profesor **FindChosenProf**(**string** surname,

**string** name, LinkList<Profesor> list)

{

Profesor profesor = **null**;

**if** (list != **null**)

{

**for** (list.Begin(); list.Exist(); list.Next())

{

**if** (list.Get() != **null** && surname

== list.Get().Surname && name

== list.Get().Name)

{

profesor = list.Get();

**break**;

}

}

}

**return** profesor;

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Text**;

**using** **System.IO**;

**namespace** **\_3Laboras**

{

/// <summary>

/// Reading and printing class

/// </summary>

**public** **class** **InOut**

{

/// <summary>

/// Reads professors and their info from the file

/// </summary>

/// <param name="fileName"></param>

/// <returns></returns>

**public** **static** ProfList **ReadProfesorLifo**(Stream fin)

{

**string** projectName;

**string** surname;

**string** name;

**int** hours;

**string** line = **null**;

ProfList list = **new** ProfList();

**using** (StreamReader file =

**new** **StreamReader**(fin, Encoding.UTF8))

{

**while** ((line = file.ReadLine()) != **null**)

{

**string**[] values =

line.Split(**new** **char**[] { ';' },

StringSplitOptions.RemoveEmptyEntries);

projectName = values[**0**];

**string**[] strings =

values[**1**].Split(**new** **char**[] { ' ' },

StringSplitOptions.RemoveEmptyEntries);

surname = strings[**0**];

name = strings[**1**];

hours = **int**.Parse(strings[**2**]);

Profesor profesor =

**new** **Profesor**(surname, name);

Projects projects =

**new** **Projects**(projectName, hours);

profesor.Project.SetLifo(projects);

list.SetLifo(profesor);

}

}

**return** list;

}

**public** **static** LinkList<Student> ReadStudentLifo(Stream fin)

{

**string** projectName;

**string** surname;

**string** name;

**string** **group**;

**string** line = **null**;

LinkList<Student> list = **new** LinkList<Student>();

**using** (StreamReader file =

**new** **StreamReader**(fin, Encoding.UTF8))

{

**while** ((line = file.ReadLine()) != **null**)

{

**string**[] values = line.Split(**new** **char**[] { ';' },

StringSplitOptions.RemoveEmptyEntries);

projectName = values[**0**];

**string**[] strings =

values[**1**].Split(**new** **char**[] { ' ' },

StringSplitOptions.RemoveEmptyEntries);

surname = strings[**0**];

name = strings[**1**];

**group** = strings[**2**];

Student profesor = **new** Student(projectName,

surname, name, **group**);

list.SetLifo(profesor);

}

}

**return** list;

}

/// <summary>

/// Prints professors with their info to the file in a table

/// </summary>

/// <param name="fileName"></param>

/// <param name="profesors"></param>

/// <param name="heading"></param>

**public** **static** **void** **PrintProfesors**(**string** fileName,

ProfList profesors, **string** heading)

{

File.AppendAllText(fileName, $"{heading}\r\n", Encoding.UTF8);

List<**string**> lines = **new** List<**string**>();

lines.Add(**new** **string**('-', **64**));

lines.Add(String.Format($"| {"Pavardė",-9} " +

$"| {"Vardas",-9} " +

$"| {"Projekto pavadinimas",-22} " +

$"| {"Valandų sk.",-11} |"));

lines.Add(**new** **string**('-', **64**));

**foreach** (Profesor one **in** profesors)

{

one.Project.Begin();

lines.Add(one.ToString() + " " +

$"{one.Project.Get().Project,-22} " +

$"| {one.Project.Get().Hours,11} |");

**for** (one.Project.Next();

one.Project.Exist();

one.Project.Next())

{

lines.Add(String.Format($"| {"",-9} " +

$"| {"",-9} | " +

$"{one.Project.Get().Project,-22} | " +

$"{one.Project.Get().Hours,11} |"));

}

lines.Add(**new** **string**('-', **64**));

}

lines.Add("");

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

}

/// <summary>

/// Print students with their info to the file in a table

/// </summary>

/// <param name="fileName"></param>

/// <param name="students"></param>

**public** **static** **void** **PrintStudents**(**string** fileName,

LinkList<Student> students)

{

List<**string**> lines = **new** List<**string**>();

lines.Add(**new** **string**('-', **64**));

lines.Add(String.Format($"| {"Pavardė",-9} " +

$"| {"Vardas",-9} " +

$"| {"Projekto pavadinimas",-22} " +

$"| {"Grupė",-11} |"));

lines.Add(**new** **string**('-', **64**));

**foreach**(Student one **in** students)

{

**if** (one != **null**)

{

lines.Add(one.ToString());

lines.Add(**new** **string**('-', **64**));

}

}

lines.Add("");

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

}

}

}

**using** **System**;

**using** **System.IO**;

**using** **System.Web.UI**;

**using** **System.Web.UI.WebControls**;

**namespace** **\_3Laboras**

{

**public** **partial** **class** **Forma1** : System.Web.UI.Page

{

**protected** **void** **Page\_Load**(**object** sender, EventArgs e)

{

**if** (Page.IsPostBack)

{

Profesor profesor1 = **null**;

ProfList list = (ProfList)Session["profesoriai"];

profesor1 = TaskUtils.FindChosenProf(TextBox1.Text,

TextBox2.Text, list);

Session["pasirinktas"] = profesor1;

}

Label1.Visible = **false**;

Label2.Visible = **false**;

Label3.Visible = **false**;

Table1.Visible = **false**;

Table2.Visible = **false**;

Table3.Visible = **false**;

Button2.Visible = **false**;

Label4.Visible = **false**;

Label5.Visible = **false**;

Label6.Visible = **false**;

TextBox1.Visible = **false**;

TextBox2.Visible = **false**;

}

**protected** **void** **Button1\_Click**(**object** sender,

EventArgs e)

{

Button1.Visible = **false**;

FileUpload1.Visible = **false**;

FileUpload2.Visible = **false**;

Label7.Visible = **false**;

Label8.Visible = **false**;

Label1.Visible = **true**;

**if** (!FileUpload1.HasFile || !FileUpload2.HasFile

|| !FileUpload1.FileName.EndsWith("b.txt")

|| !FileUpload2.FileName.EndsWith("a.txt"))

{

Label1.Text = "Pateikti blogi duomenys";

**return**;

}

ProfList profesors =

InOut.ReadProfesorLifo(FileUpload1.FileContent);

LinkList<Student> students =

InOut.ReadStudentLifo(FileUpload2.FileContent);

File.Delete(Server.MapPath("Rezultatai.txt"));

InOut.PrintProfesors(Server.MapPath("Rezultatai.txt"),

profesors, "Pradiniai duomenys:");

InOut.PrintStudents(Server.MapPath("Rezultatai.txt"),

students);

Table1Header();

Table2Header();

FillTable1(profesors);

FillTable2(students);

Label4.Visible = **true**;

Label5.Visible = **true**;

Table1.Visible = **true**;

Table2.Visible = **true**;

Button2.Visible = **true**;

TextBox1.Visible = **true**;

TextBox2.Visible = **true**;

**if** (profesors.HasData())

{

profesors = profesors.RemoveUnselected(students);

}

profesors.MergeSameProf();

profesors.Sort();

Profesor profesor = profesors.HasTheMostProjects();

InOut.PrintProfesors(Server.MapPath("Rezultatai.txt"),

profesors, "Rezultatai:");

**if** (profesor != **null**)

{

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"Daugiausiai projektų turintis dėstytojas: " +

$"{profesor.Surname} {profesor.Name} " +

$"({profesor.Project.Count()})\r\n");

}

**else**

{

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"Nėra profesoriaus su daugiausiai projektų\r\n");

}

Session["profesoriai"] = profesors;

Session["daugiausiai"] = profesor;

}

**protected** **void** **Button2\_Click**(**object** sender, EventArgs e)

{

Profesor profesor = (Profesor)Session["daugiausiai"];

ProfList profesors = (ProfList)Session["profesoriai"];

Profesor profesor1 = (Profesor)Session["pasirinktas"];

Label2.Visible = **true**;

Label3.Visible = **true**;

Table3.Visible = **true**;

Label6.Visible = **true**;

Table3Header();

FillTable3(profesors);

PrintMostProj(profesor);

**if** (profesor1 == **null**)

{

Label6.Text = "Pasirinktas profesorius neegzistuoja";

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"\r\nPasirinktas profesorius neegzistuoja");

}

**else**

{

Label6.Text = $"Pasirinktas profesorius: " +

$"{profesor1.Surname} {profesor1.Name} " +

$"<br />Projektai:<br />{Projektai(profesor1)}";

File.AppendAllText(Server.MapPath("Rezultatai.txt"),

$"\r\nPasirinktas profesorius: {profesor1.Surname} " +

$"{profesor1.Name} " +

$"\r\nProjektai:\r\n{ProjektaiTxt(profesor1)}");

}

}

/// <summary>

/// For project printing

/// </summary>

/// <param name="profesor"></param>

/// <returns></returns>

**private** **string** **Projektai**(Profesor profesor)

{

**string** pro = "";

LinkList<Projects> projects = profesor.Project;

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

pro += projects.Get().Project + "<br />";

}

}

**return** pro;

}

/// <summary>

/// For project printing to the file

/// </summary>

/// <param name="profesor"></param>

/// <returns></returns>

**private** **string** **ProjektaiTxt**(Profesor profesor)

{

**string** pro = "";

LinkList<Projects> projects = profesor.Project;

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

pro += projects.Get().Project + "\r\n";

}

}

**return** pro;

}

/// <summary>

/// Makes table1 header

/// </summary>

**private** **void** **Table1Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Profesoriai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Projektas ";

TableCell four = **new** TableCell();

four.Text = "Valandų sk.";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table1.Rows.Add(row);

Table1.Rows.Add(row2);

}

/// <summary>

/// Makes table2 header

/// </summary>

**private** **void** **Table2Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Studentai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Grupė ";

TableCell four = **new** TableCell();

four.Text = "Projektas";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table2.Rows.Add(row);

Table2.Rows.Add(row2);

}

/// <summary>

/// Makes table3 header

/// </summary>

**private** **void** **Table3Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Profesoriai";

TableCell one = **new** TableCell();

one.Text = "Pavardė ";

TableCell two = **new** TableCell();

two.Text = "Vardas ";

TableCell three = **new** TableCell();

three.Text = "Projektas ";

TableCell four = **new** TableCell();

four.Text = "Valandų sk.";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

TableRow row2 = **new** TableRow();

row2.Cells.Add(one);

row2.Cells.Add(two);

row2.Cells.Add(three);

row2.Cells.Add(four);

Table3.Rows.Add(row);

Table3.Rows.Add(row2);

}

/// <summary>

/// Fills table1

/// </summary>

/// <param name="profesors"></param>

**private** **void** **FillTable1**(ProfList profesors)

{

**for** (profesors.Begin(); profesors.Exist();

profesors.Next())

{

profesors.Get().Project.Begin();

TableCell one = **new** TableCell();

one.Text = profesors.Get().Surname;

TableCell two = **new** TableCell();

two.Text = profesors.Get().Name;

TableCell three = **new** TableCell();

three.Text = profesors.Get().Project.Get().Project;

TableCell four = **new** TableCell();

four.Text =

profesors.Get().Project.Get().Hours.ToString();

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table1.Rows.Add(row);

}

}

/// <summary>

/// Fills table2

/// </summary>

/// <param name="students"></param>

**private** **void** **FillTable2**(LinkList<Student> students)

{

**for** (students.Begin(); students.Exist(); students.Next())

{

**if** (students.Get() != **null**)

{

TableCell one = **new** TableCell();

one.Text = students.Get().Surname;

TableCell two = **new** TableCell();

two.Text = students.Get().Name;

TableCell three = **new** TableCell();

three.Text = students.Get().Group;

TableCell four = **new** TableCell();

four.Text = students.Get().ProjectName;

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table2.Rows.Add(row);

}

}

}

/// <summary>

/// Fills table3

/// </summary>

/// <param name="profesors"></param>

**private** **void** **FillTable3**(ProfList profesors)

{

**for** (profesors.Begin(); profesors.Exist();

profesors.Next())

{

**if** (profesors.Get() != **null**)

{

TableCell one = **new** TableCell();

one.Text = profesors.Get().Surname;

TableCell two = **new** TableCell();

two.Text = profesors.Get().Name;

TableCell three = **new** TableCell();

three = CellThree(profesors.Get().Project);

TableCell four = **new** TableCell();

four = CellFour(profesors.Get().Project);

TableRow row = **new** TableRow();

row.Cells.Add(one);

row.Cells.Add(two);

row.Cells.Add(three);

row.Cells.Add(four);

Table3.Rows.Add(row);

}

}

}

/// <summary>

/// Fills table cell with projects

/// </summary>

/// <param name="projects"></param>

/// <returns></returns>

**private** TableCell **CellThree**(LinkList<Projects> projects)

{

TableCell three = **new** TableCell();

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

three.Text += projects.Get().Project + "<br />";

}

}

**return** three;

}

/// <summary>

/// Fills table cell with hours

/// </summary>

/// <param name="projects"></param>

/// <returns></returns>

**private** TableCell **CellFour**(LinkList<Projects> projects)

{

TableCell four = **new** TableCell();

**for** (projects.Begin(); projects.Exist(); projects.Next())

{

**if** (projects.Get() != **null**)

{

four.Text += projects.Get().Hours + "<br />";

}

}

**return** four;

}

/// <summary>

/// Prints the professor which has most projects

/// </summary>

/// <param name="profesor"></param>

**private** **void** **PrintMostProj**(Profesor profesor)

{

**if** (profesor != **null**)

{

Label3.Text = $"Daugiausiai projektų " +

$"turintis profesorius: {profesor.Surname} " +

$"{profesor.Name} ({profesor.Project.Count()})";

}

**else**

{

Label3.Text = "Nėra profesoriaus su daugiausiai " +

"projektų";

}

}

}

}

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Forma1.aspx.cs" Inherits="\_3Laboras.Forma1" %>

<!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="Label7" runat="server" Text="Profesorių duomenys:"></asp:Label>

<br />

<asp:FileUpload ID="FileUpload1" runat="server" />

<br />

<asp:Label ID="Label8" runat="server" Text="Studentų duomenys:"></asp:Label>

<br />

<asp:FileUpload ID="FileUpload2" runat="server" />

<br />

<br />

<asp:Button ID="Button1" runat="server" Text="Nuskaityti duomenis" OnClick="Button1\_Click" />

<br />

<br />

<asp:Label ID="Label1" runat="server" Text="Pradiniai duomenys:"></asp:Label>

<asp:Table ID="Table1" runat="server" BorderColor="Black" BorderWidth="1px" GridLines="Horizontal" BorderStyle="Solid"></asp:Table>

<br />

<asp:Table ID="Table2" runat="server" BorderColor="Black" BorderWidth="1px" GridLines="Horizontal"></asp:Table>

<br />

<br />

<asp:Label ID="Label4" runat="server" Text="Profesoriaus pavardė:"></asp:Label>

<br />

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<br />

<asp:Label ID="Label5" runat="server" Text="Profesoriaus vardas:"></asp:Label>

<br />

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<br />

<br />

<asp:Button ID="Button2" runat="server" Text="Apskaičiuoti" OnClick="Button2\_Click" />

<br />

<br />

<asp:Label ID="Label2" runat="server" Text="Rezultatai:"></asp:Label>

<asp:Table ID="Table3" runat="server" BorderWidth="1px" BorderColor="Black" GridLines="Horizontal"></asp:Table>

<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>

<br />

<br />

<asp:Label ID="Label6" runat="server" Text="Label"></asp:Label>

</div>

</form>

</body>

</html>

**using** **Microsoft.VisualStudio.TestTools.UnitTesting**;

**using** **FluentAssertions**;

**using** **System**;

**namespace** **\_3Laboras.Tests**

{

/// <summary>

/// Class for tests

/// </summary>

[TestClass()]

**public** **class** **LinkListTests**

{

/// <summary>

/// Tests if SetLifo method adds one item

/// </summary>

[TestMethod()]

**public** **void** **SetLifoOneItemTest**()

{

LinkList<Student> students =

**new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Begin();

students.Get().Should().Be(student);

}

/// <summary>

/// Tests if SetLifo methor throws

/// "ArgumentNullException" when trying to add null

/// </summary>

[TestMethod()]

**public** **void** **SetLifoNullItemTest**()

{

LinkList<Student> students =

**new** LinkList<Student>();

Action act = () => students.SetLifo((Student) **null**);

act.Should().Throw<ArgumentNullException>();

}

/// <summary>

/// Tests if SetLifo method adds two items

/// </summary>

[TestMethod()]

**public** **void** **SetLifoTwoItemsTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student1 = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student2 = **new** Student("Proj2",

"Sur2", "Name2", "Grp2");

students.SetLifo(student1);

students.SetLifo(student2);

students.Begin();

students.Get().Should().Be(student2);

students.Next();

students.Get().Should().Be(student1);

}

/// <summary>

/// Tests if Begin method would throw

/// "InvalidOperationException" when the

/// trying to choose the empty item

/// </summary>

[TestMethod()]

**public** **void** **BeginEmptyTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Action act = () => students.Begin();

act.Should().Throw<InvalidOperationException>();

}

/// <summary>

/// Tests if Begin method works at all

/// </summary>

[TestMethod()]

**public** **void** **BeginTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Begin();

students.Get().Should().Be(student);

}

/// <summary>

/// Tests if Next method goes to the next item

/// </summary>

[TestMethod()]

**public** **void** **NextTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student1 = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student2 = **new** Student("Proj2",

"Sur2", "Name2", "Grp2");

students.SetLifo(student1);

students.SetLifo(student2);

students.Begin();

students.Get().Should().Be(student2);

students.Next();

students.Get().Should().Be(student1);

}

/// <summary>

/// Tests if Next method throws exception

/// when it is called without begining

/// </summary>

[TestMethod()]

**public** **void** **NextNoBeginTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

Action act = () => students.Next();

act.Should().Throw<InvalidOperationException>();

}

/// <summary>

/// Tests if Next method throws exception

/// when there is no next item

/// </summary>

[TestMethod()]

**public** **void** **NextAfterTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Begin();

students.Next();

Action act = () => students.Next();

act.Should().Throw<InvalidOperationException>();

}

/// <summary>

/// Tests if Exist method is false then

/// the LinkedList is null

/// </summary>

[TestMethod()]

**public** **void** **ExistBeforeBeginTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

students.Exist().Should().Be(**false**);

}

/// <summary>

/// Tests if Exist method returns true

/// while going through full LinkedList

/// </summary>

[TestMethod()]

**public** **void** **ExistTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student2 = **new** Student("Proj2",

"Sur2", "Name2", "Grp2");

students.SetLifo(student);

students.SetLifo(student2);

students.Begin();

students.Exist().Should().Be(**true**);

}

/// <summary>

/// Tests if Exist method returns false

/// when the chosen item is null

/// </summary>

[TestMethod()]

**public** **void** **ExistAfterTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Begin();

students.Next();

students.Exist().Should().Be(**false**);

}

/// <summary>

/// Tests if HasData method returns

/// false before adding data

/// </summary>

[TestMethod()]

**public** **void** **HasDataBeforeSetTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

students.HasData().Should().Be(**false**);

}

/// <summary>

/// Tests if HasData method returns

/// true when there is some data

/// </summary>

[TestMethod()]

**public** **void** **HasDataTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.HasData().Should().Be(**true**);

}

/// <summary>

/// Tests if HasData method returns true

/// when there is some data

/// </summary>

[TestMethod()]

**public** **void** **HasDataAfterExistTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Begin();

students.Next();

students.HasData().Should().Be(**true**);

}

/// <summary>

/// Tests if Count method returns zero when

/// there are no elements in the list

/// </summary>

[TestMethod()]

**public** **void** **CountZeroItemTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

students.Count().Should().Be(**0**);

}

/// <summary>

/// Tests if Count method returns 2 when

/// there are two items in the list

/// </summary>

[TestMethod()]

**public** **void** **CountWhenHasTwoItemsTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student1 = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student2 = **new** Student("Proj2",

"Sur2", "Name2", "Grp2");

students.SetLifo(student1);

students.SetLifo(student2);

students.Count().Should().Be(**2**);

}

/// <summary>

/// Tests if Count method returns 1 when

/// there is one item in the list

/// </summary>

[TestMethod()]

**public** **void** **CountWhenHasOneItemTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Count().Should().Be(**1**);

}

/// <summary>

/// Tests if SetLifo method throws exception

/// when trying to add empty list

/// </summary>

[TestMethod()]

**public** **void** **SetLifoNullListItemsTest**()

{

LinkList<Student> test = **new** LinkList<Student>();

Action act = () => test.SetLifo((LinkList<Student>) **null**);

act.Should().Throw<ArgumentNullException>();

}

/// <summary>

/// Tests if SetLifo method adds given

/// one item list to the other list

/// </summary>

[TestMethod()]

**public** **void** **SetLifoOneListItemTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

LinkList<Student> test = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student1 = **new** Student("Proj1",

"Sur1", "Name1", "Grp1");

test.SetLifo(student);

students.SetLifo(student1);

test.SetLifo(students);

test.Count().Should().Be(**2**);

}

/// <summary>

/// Tests if SetLifo method adds given

/// two item list to the other list

/// </summary>

[TestMethod()]

**public** **void** **SetLifoTwoListItemsTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

LinkList<Student> test = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

Student student1 = **new** Student("Proj1",

"Sur1", "Name1", "Grp1");

Student student2 = **new** Student("Proj2",

"Sur2", "Name2", "Grp2");

test.SetLifo(student);

students.SetLifo(student1);

students.SetLifo(student2);

test.SetLifo(students);

test.Count().Should().Be(**3**);

}

/// <summary>

/// Tests if Sort method throws exception

/// when trying to sort empty list

/// </summary>

[TestMethod()]

**public** **void** **SortEmptyListTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Action act = () => students.Sort();

act.Should().Throw<InvalidOperationException>();

}

/// <summary>

/// Tests if Sort method doesnt

/// fail while sorting one item list

/// </summary>

[TestMethod()]

**public** **void** **SortOneItemListTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"Sur", "Name", "Grp");

students.SetLifo(student);

students.Sort();

students.Begin();

students.Get().Should().Be(student);

}

/// <summary>

/// Tests if the Sort method works

/// with three item list

/// </summary>

[TestMethod()]

**public** **void** **SortThreeItemListTest**()

{

LinkList<Student> students = **new** LinkList<Student>();

Student student = **new** Student("Proj",

"ABC", "BCD", "Grp");

Student student1 = **new** Student("Proj1",

"ABC", "ABC", "Grp1");

Student student2 = **new** Student("Proj2",

"AAB", "BCD", "Grp2");

students.SetLifo(student2);

students.SetLifo(student1);

students.SetLifo(student);

students.Sort();

students.Begin();

students.Get().Should().Be(student2);

students.Next();

students.Get().Should().Be(student1);

students.Next();

students.Get().Should().Be(student);

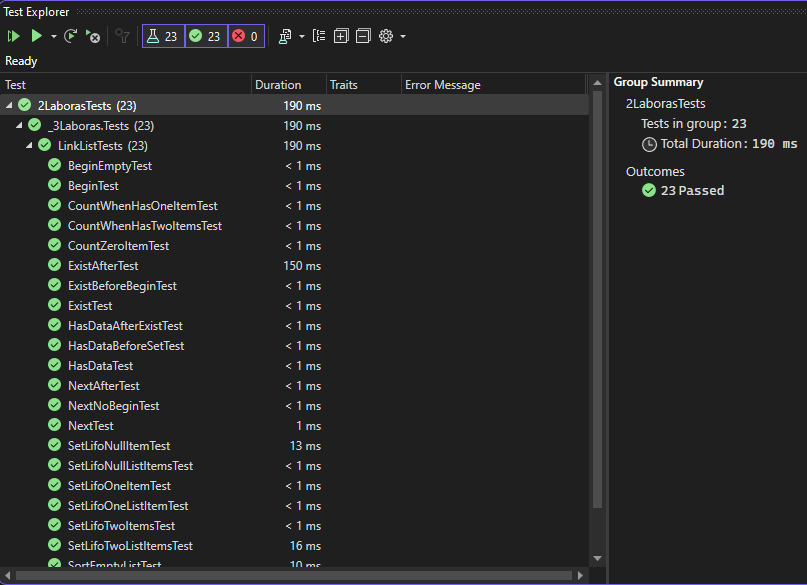
}

}

}

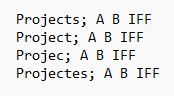
## Pradiniai duomenys ir rezultatai

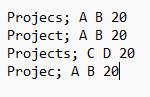
Testų rezultatai:



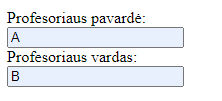
### Duomenys ir rezultatai 1

Pradiniai duomenys:



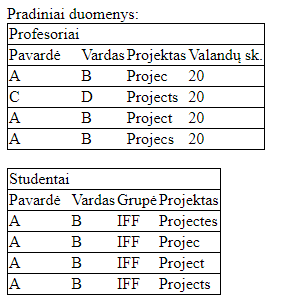


Įvedami duomenys:

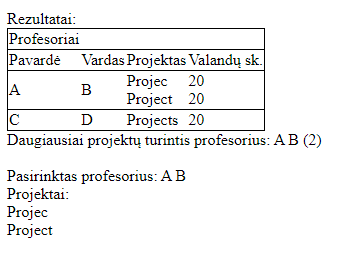


Šiais duomenimis tikrinama ar veikia profesorių, kurių projektai buvo nepasirinkti pašalinimas. Taip pat tikrinama ar veikia profesorių, kurie turi daugiau projektų suliejimas. Tikrinama ar randamas teisingas daugiausiai projektų turintis dėstytojas. Tikrinama ar atspausdinamas pasirinktas profesorius.

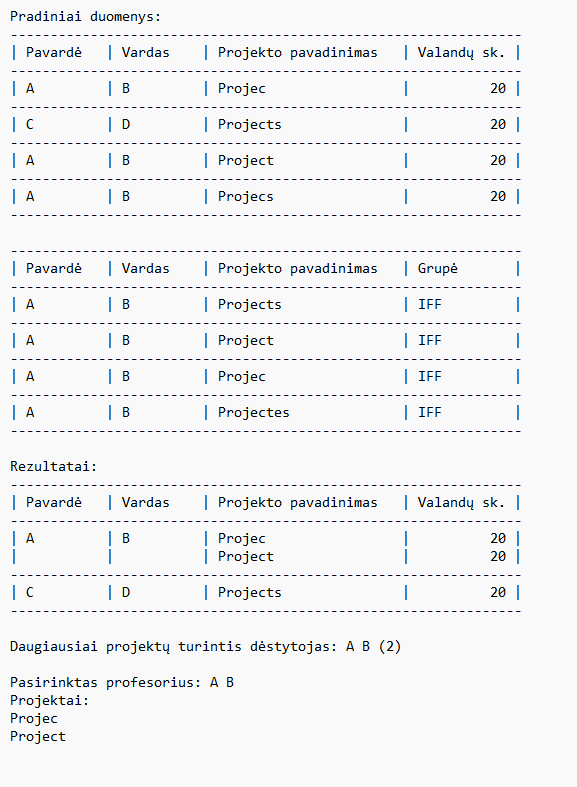
Pradiniai duomenys web:



Rezultatai web:

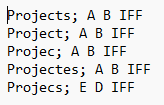


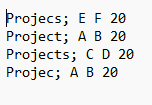
Rezultatų tekstinis failas:



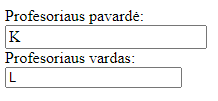
### Duomenys ir rezultatai 2

Pradiniai duomenys:



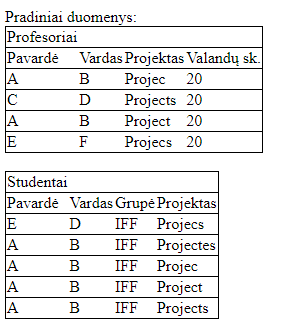


Įvedami duomenys:

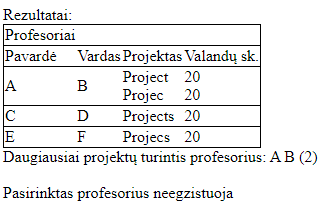


Šiais duomenimis tikrinama, kas yra atspausdinama, jei pasirenkamas neegzistuojantis dėstytojas. Tikrinama ar tikrai veikia rūšiavimas.

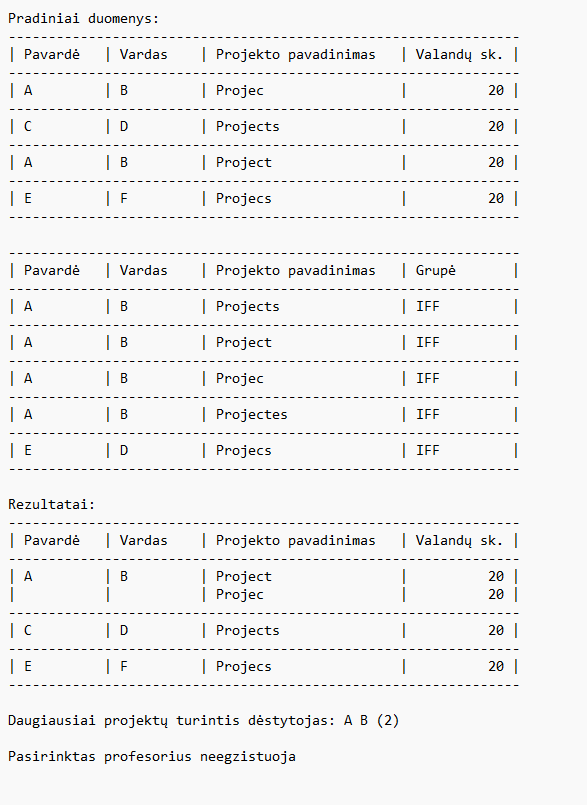
Pradiniai duomenys web:



Rezultatai web:



Rezultatų tekstinis failas:



## Dėstytojo pastabos

1. Dvi klasės ProfList ir LinkList.
2. Protected Node head.
3. Kūrybiškesni duomenys – pataisyta

Testas balai – 2

Gautas įvertinimas – 9

# Polimorfizmas ir išimčių valdymas (L4)

## Darbo užduotis

**U4\_22. Buitinės technikos parduotuvė.**

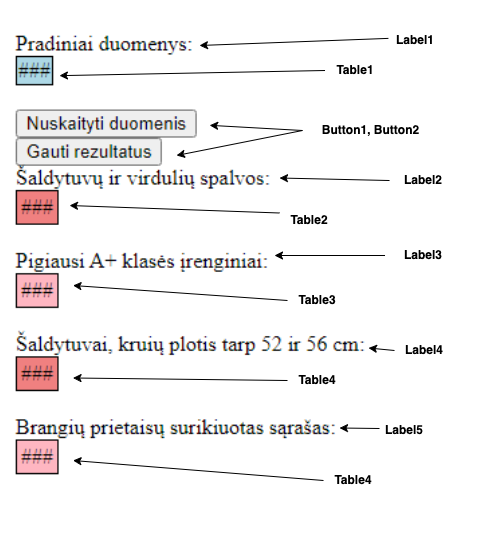
Turite informaciją apie skirtingose buitinės technikos parduotuvėse (>=3) esančius šaldytuvus, mikrobangų krosneles ir elektrinius virdulius. Pirmoje eilutėje – pavadinimas, antroje – adresas, trečioje – telefonas. Sukurkite abstrakčią klasę „Device“ (savybės - gamintojas, modelis, energijos klasė, spalva, kaina), kurią paveldės “Fridge” (savybės – talpa, montavimo tipas, požymis „turi šaldiklį“, aukštis, plotis, gylis), “Oven” (savybės – galingumas, programų skaičius) ir “Kettle” (savybės – galia, tūris).

• Raskite, kokių skirtingų spalvų šaldytuvus galima įsigyti, spalvų sąrašą atspausdinkite ekrane. Raskite, kokių skirtingų spalvų elektrinius virdulius galima įsigyti, spalvų sąrašą atspausdinkite ekrane.

• Raskite pigiausią A+ klasės šaldytuvą, mikrobangų krosnelę ir virdulį, kiekvieno jų visą informaciją atspausdinkite ekrane. • Sudarykite šaldytuvų, kurių plotis nuo 52 iki 56 cm, sąrašą ir įrašykite visus duomenis apie šiuos šaldytuvus į failą „Tilps.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“.

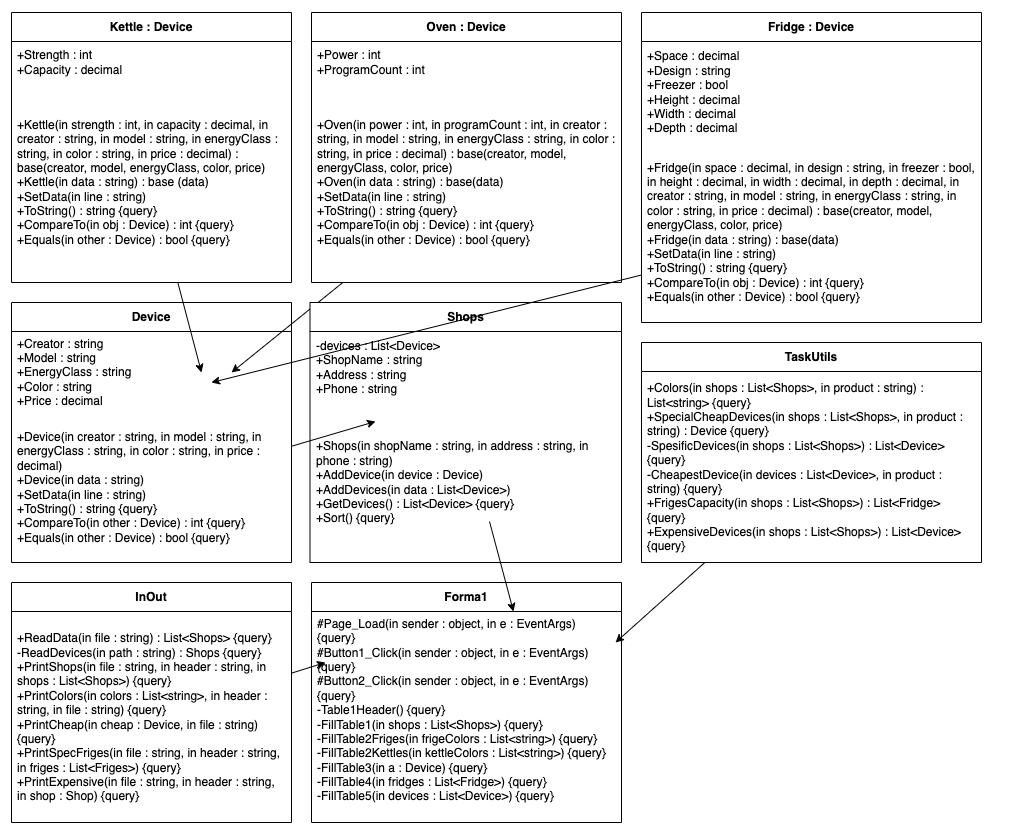
## Grafinės vartotojo sąsajos schema



## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| Button | ID | Button1 |
| Button | Text | Nuskaityti duomenis |
| Button | OnClick | Button1\_Click |
| Label | ID | Label1 |
| Label | Text | Pradiniai duomenys: |
| Table | ID | Table1 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | GridLines | Horizontal |
| Table | BorderStyle | Solid |
| Table | BackColor | LightBlue |
| Table | ID | Table2 |
| Table | BorderColor | Black |
| Table | BorderWidth | 1px |
| Table | BorderStyle | Solid |
| Table | BackColor | LightCoral |
| Label | ID | Label2 |
| Label | Text | Šaldytuvų ir virdulių spalvos: |
| Label | ID | Label3 |
| Label | Text | Pigiausi A+ klasės įrenginiai: |
| Button | ID | Button2 |
| Button | Text | Gauti rezultatus |
| Button | OnClick | Button2\_Click |
| Label | ID | Label4 |
| Label | Text | Šaldytuvai, kurių plotis tarp 52 ir 56 cm: |
| Table | ID | Table3 |
| Table | BorderWidth | 1px |
| Table | BorderColor | Black |
| Table | BorderStyle | Solid |
| Table | BackColor | LightPink |
| Label | ID | Label5 |
| Label | Text | Brangių prietaisų surikiuotas sąrašas: |
| Table | ID | Table4 |
| Table | BorderWidth | 1px |
| Table | BorderColor | Black |
| Table | BorderStyle | Solid |
| Table | BackColor | LightCoral |
| Table | ID | Table5 |
| Table | BorderWidth | 1px |
| Table | BorderColor | Black |
| Table | BorderStyle | Solid |
| Table | BackColor | LightPink |
| Table | GridLines | Horizontal |

## Klasių diagrama



## Programos vartotojo vadovas

Programos darbiniame aplanke atidarome App\_Data aplanką, jame sukuriame failus su atskiromis parduotuvėmis ir parduotuvių prekėmis.

Įjungę programą, pirmiausia užkrauname duomenų failus. Tai padarome nuspausdami mygtuką „Nuskaityti duomenis“. Paspaudus mygtuką patikriname ar duomenys buvo įvesti teisingai. Jei lentelėje duomenys teisingi, spaudžiame mygtuką „Gauti rezultatus“. Paspaudus mygtuką programa apdoros duomenis ir į ekraną bei rezultatų failus išves rezultatus.

## Programos tekstas

**using** **System**;

**namespace** **\_4Laboras**

{

**public** **class** **Kettle** : Device

{

**public** **int** Strength { **get**; **set**; }

**public** **decimal** Capacity { **get**; **set**; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="strength"></param>

/// <param name="capacity"></param>

/// <param name="creator"></param>

/// <param name="model"></param>

/// <param name="energyClass"></param>

/// <param name="color"></param>

/// <param name="price"></param>

**public** **Kettle**(**int** strength, **decimal** capacity, **string** creator,

**string** model, **string** energyClass, **string** color, **decimal** price)

: **base**(creator, model, energyClass, color, price)

{

Strength = strength;

Capacity = capacity;

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="data"></param>

**public** **Kettle**(**string** data) : **base**(data)

{

SetData(data);

}

/// <summary>

/// Sets the data from the given line

/// Redirects it first to the base class

/// </summary>

/// <param name="line"></param>

**public** **override** **void** **SetData**(**string** line)

{

**base**.SetData(line);

**string**[] values = line.Split(',');

Strength = **int**.Parse(values[**6**]);

Capacity = **decimal**.Parse(values[**7**].Replace('.', ','));

}

/// <summary>

/// Overriden for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**return** **base**.ToString() + String.Format($",{Strength}," +

$"{Capacity}");

}

/// <summary>

/// Compares strength if device ir the same from the same class

/// For sorting

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

**public** **override** **int** **CompareTo**(Device obj)

{

**int** baseResult = **base**.CompareTo(obj);

**if** (baseResult == **0**)

{

**return** **this**.Strength.CompareTo((obj **as** Kettle).Strength);

}

**else**

{

**return** baseResult;

}

}

/// <summary>

/// Checks if device is Kettle

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **override** **bool** **Equals**(Device other)

{

**return** other **is** Kettle;

}

}

}

**using** **System**;

**namespace** **\_4Laboras**

{

**public** **class** **Oven** : Device

{

**public** **int** Power { **get**; **set**; }

**public** **int** ProgramCount { **get**; **set**; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="power"></param>

/// <param name="programCount"></param>

/// <param name="creator"></param>

/// <param name="model"></param>

/// <param name="energyClass"></param>

/// <param name="color"></param>

/// <param name="price"></param>

**public** **Oven**(**int** power, **int** programCount, **string** creator, **string** model,

**string** energyClass, **string** color, **decimal** price)

: **base**(creator, model, energyClass, color, price)

{

Power = power;

ProgramCount = programCount;

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="data"></param>

**public** **Oven**(**string** data) : **base**(data)

{

SetData(data);

}

/// <summary>

/// Sets the data from the given line

/// Redirects it first to the base class

/// </summary>

/// <param name="line"></param>

**public** **override** **void** **SetData**(**string** line)

{

**base**.SetData(line);

**string**[] values = line.Split(',');

Power = **int**.Parse(values[**6**]);

ProgramCount = **int**.Parse(values[**7**]);

}

/// <summary>

/// Overriden ToString for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**return** **base**.ToString() + String.Format($",{Power}," +

$"{ProgramCount}");

}

/// <summary>

/// Compares powers if device is from the same class

/// For sorting

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

**public** **override** **int** **CompareTo**(Device obj)

{

**int** baseResult = **base**.CompareTo(obj);

**if** (baseResult == **0**)

{

**return** **this**.Power.CompareTo((obj **as** Oven).Power);

}

**else**

{

**return** baseResult;

}

}

/// <summary>

/// Checks if device is Oven

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **override** **bool** **Equals**(Device other)

{

**return** other **is** Oven;

}

}

}

**using** **System**;

**namespace** **\_4Laboras**

{

**public** **class** **Fridge** : Device

{

**public** **decimal** Space { **get**; **set**; }

**public** **string** Design { **get**; **set**; }

**public** **bool** Freezer { **get**; **set**; }

**public** **decimal** Height { **get**; **set**; }

**public** **decimal** Width { **get**; **set**; }

**public** **decimal** Depth { **get**; **set**; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="space"></param>

/// <param name="design"></param>

/// <param name="freezer"></param>

/// <param name="height"></param>

/// <param name="width"></param>

/// <param name="depth"></param>

/// <param name="creator"></param>

/// <param name="model"></param>

/// <param name="energyClass"></param>

/// <param name="color"></param>

/// <param name="price"></param>

**public** **Fridge**(**decimal** space, **string** design, **bool** freezer, **decimal** height,

**decimal** width, **decimal** depth, **string** creator, **string** model,

**string** energyClass, **string** color, **decimal** price)

: **base**(creator, model, energyClass, color, price)

{

Space = space;

Design = design;

Freezer = freezer;

Height = height;

Width = width;

Depth = depth;

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="data"></param>

**public** **Fridge**(**string** data) : **base**(data)

{

SetData(data);

}

/// <summary>

/// Sets data from the line

/// Redirects it first to the base class

/// </summary>

/// <param name="line"></param>

**public** **override** **void** **SetData**(**string** line)

{

**base**.SetData(line);

**string**[] values = line.Split(',');

Space = **decimal**.Parse(values[**6**].Replace('.', ','));

Design = values[**7**];

Freezer = **bool**.Parse(values[**8**]);

Height = **decimal**.Parse(values[**9**].Replace('.', ','));

Width = **decimal**.Parse(values[**10**].Replace('.', ','));

Depth = **decimal**.Parse(values[**11**].Replace('.', ','));

}

/// <summary>

/// Overriden ToString for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**return** **base**.ToString() + String.Format($",{Space},{Design},{Freezer}," +

$"{Height},{Width},{Depth}");

}

/// <summary>

/// Compares heights if types are the same

/// For sorting

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

**public** **override** **int** **CompareTo**(Device obj)

{

**int** baseResult = **base**.CompareTo(obj);

**if** (baseResult == **0**)

{

**return** **this**.Height.CompareTo((obj **as** Fridge).Height);

}

**else**

{

**return** baseResult;

}

}

/// <summary>

/// Checks if device is Fridge

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **override** **bool** **Equals**(Device other)

{

**return** other **is** Fridge;

}

}

}

**using** **System**;

**namespace** **\_4Laboras**

{

**public** **abstract** **class** **Device** : IComparable<Device>, IEquatable<Device>

{

**public** **string** Creator { **get**; **set**; }

**public** **string** Model { **get**; **set**; }

**public** **string** EnergyClass { **get**; **set**; }

**public** **string** Color { **get**; **set**; }

**public** **decimal** Price { **get**; **set**; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="creator"></param>

/// <param name="model"></param>

/// <param name="energyClass"></param>

/// <param name="color"></param>

/// <param name="price"></param>

**public** **Device**(**string** creator, **string** model,

**string** energyClass, **string** color, **decimal** price)

{

Creator = creator;

Model = model;

EnergyClass = energyClass;

Color = color;

Price = price;

}

/// <summary>

/// Constructor

/// </summary>

/// <param name="data"></param>

**public** **Device**(**string** data)

{

SetData(data);

}

/// <summary>

/// Sets data from the line

/// </summary>

/// <param name="line"></param>

**public** **virtual** **void** **SetData**(**string** line)

{

**string**[] values = line.Split(',');

Creator = values[**1**];

Model = values[**2**];

EnergyClass = values[**3**];

Color = values[**4**];

Price = **decimal**.Parse(values[**5**].Replace('.', ','));

}

/// <summary>

/// ToString overriden for printing

/// </summary>

/// <returns></returns>

**public** **override** **string** **ToString**()

{

**return** String.Format($"{Creator},{Model},{EnergyClass}," +

$"{Color},{Price}");

}

/// <summary>

/// Compares the types

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **virtual** **int** **CompareTo**(Device other)

{

**return** **this**.GetType().Name.CompareTo(other.GetType().Name);

}

/// <summary>

/// Checks if the device model is equal to other device model

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

**public** **virtual** **bool** **Equals**(Device other)

{

**return** **this**.Model.Equals(other.Model);

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **\_4Laboras**

{

/// <summary>

/// Class for storing data

/// </summary>

**public** **class** **Shops**

{

**private** List<Device> devices;

**public** **string** ShopName { **get**; **private** **set**; }

**public** **string** Address { **get**; **set**; }

**public** **string** Phone { **get**; **set**; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="shopName"></param>

/// <param name="address"></param>

/// <param name="phone"></param>

/// <exception cref="ArgumentNullException"></exception>

**public** **Shops**(**string** shopName, **string** address, **string** phone)

{

**if** (shopName == **null** && address == **null** && phone == **null**)

{

**throw** **new** **ArgumentNullException**("Parduotuvė turi " +

"turėti bent vieną iš trijų savybių");

}

**else**

{

ShopName = shopName;

Address = address;

Phone = phone;

devices = **new** List<Device>();

}

}

/// <summary>

/// Adds device to the list

/// </summary>

/// <param name="device"></param>

/// <exception cref="ArgumentNullException"></exception>

**public** **void** **AddDevice**(Device device)

{

**if** (device == **null**)

{

**throw** **new** **ArgumentNullException**("Pridedamas įrenginys " +

"negali būti tuščias");

}

devices.Add(device);

}

/// <summary>

/// Adds a device list to the list

/// </summary>

/// <param name="data"></param>

**public** **void** **AddDevices**(List<Device> data)

{

**foreach** (Device device **in** data)

{

devices.Add(device);

}

}

/// <summary>

/// Returns the devices list

/// </summary>

/// <returns></returns>

**public** List<Device> GetDevices()

{

**return** devices;

}

/// <summary>

/// Sorts devices list

/// </summary>

/// <exception cref="ArgumentNullException"></exception>

**public** **void** **Sort**()

{

**if** (devices.Count == **0**)

{

**throw** **new** **ArgumentNullException**("Rušiuojamas " +

"sąrašas tuščias");

}

**bool** flag = **true**;

**while** (flag)

{

flag = **false**;

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

{

Device one = devices[i];

Device two = devices[i + **1**];

**if** (one.CompareTo(two) > **0**)

{

devices[i] = two;

devices[i + **1**] = one;

flag = **true**;

**break**;

}

}

}

}

}

}

**using** **System.Collections.Generic**;

**namespace** **\_4Laboras**

{

/// <summary>

/// Class for calculating

/// </summary>

**public** **static** **class** **TaskUtils**

{

/// <summary>

/// Returns the list of different colors

/// of chosen product

/// </summary>

/// <param name="shops"></param>

/// <param name="product"></param>

/// <returns></returns>

**public** **static** List<**string**> Colors(

List<Shops> shops,

**string** product)

{

List<**string**> colors = **new** List<**string**>();

**foreach** (Shops shop **in** shops)

{

**foreach** (Device device **in** shop.GetDevices())

{

**if** (device **is** Fridge && product == "fridge"

&& !colors.Contains(device.Color))

{

colors.Add(device.Color);

}

**else** **if** (device **is** Kettle && product == "kettle"

&& !colors.Contains(device.Color))

{

colors.Add(device.Color);

}

}

}

**return** colors;

}

/// <summary>

/// Returns the device which is the cheapest and its

/// energy class is A+

/// </summary>

/// <param name="shops"></param>

/// <param name="product"></param>

/// <returns></returns>

**public** **static** Device **SpecialCheapDevices**(

List<Shops> shops,

**string** product)

{

List<Device> temp = SpesificDevices(shops);

**return** **CheapestDevice**(temp, product);

}

/// <summary>

/// Returns list of devices which all have A+ energy class

/// </summary>

/// <param name="shops"></param>

/// <returns></returns>

**private** **static** List<Device> SpesificDevices(List<Shops> shops)

{

List<Device> result = **new** List<Device>();

**foreach** (Shops shop **in** shops)

{

**foreach** (Device device **in** shop.GetDevices())

{

**if** (device.EnergyClass.Equals("A+"))

{

result.Add(device);

}

}

}

**return** result;

}

/// <summary>

/// Returns the device which is the cheapest from the list

/// </summary>

/// <param name="devices"></param>

/// <param name="product"></param>

/// <returns></returns>

**private** **static** Device **CheapestDevice**(

List<Device> devices,

**string** product)

{

**decimal** price = -**1**;

Device res = **null**;

**foreach** (Device device **in** devices)

{

**if** (device.Price < price || price == -**1**)

{

**if** (device **is** Fridge && product == "fridge")

{

price = device.Price;

res = device;

}

**else** **if** (device **is** Oven && product == "oven")

{

price = device.Price;

res = device;

}

**else** **if** (device **is** Kettle && product == "kettle")

{

price = device.Price;

res = device;

}

}

}

**return** res;

}

/// <summary>

/// Returns friges list which contains all friges

/// with the width between 52 and 56 cm

/// </summary>

/// <param name="shops"></param>

/// <returns></returns>

**public** **static** List<Fridge> FrigesCapacity(List<Shops> shops)

{

List<Fridge> fridges = **new** List<Fridge>();

**foreach** (Shops shop **in** shops)

{

**foreach** (Device device **in** shop.GetDevices())

{

**if** (device **is** Fridge fridge)

{

**if** (fridge.Width >= **52** && fridge.Width <= **56**)

{

fridges.Add(fridge);

}

}

}

}

**return** fridges;

}

/// <summary>

/// Returns list of devices which are considered to be expensive

/// </summary>

/// <param name="shops"></param>

/// <returns></returns>

**public** **static** List<Device> ExpensiveDevices(List<Shops> shops)

{

List<Device> devices = **new** List<Device>();

**foreach** (Shops shop **in** shops)

{

**foreach** (Device device **in** shop.GetDevices())

{

**if** (device **is** Fridge && device.Price > **1000**)

{

devices.Add(device);

}

**else** **if** (device **is** Oven && device.Price > **500**)

{

devices.Add(device);

}

**else** **if** (device **is** Kettle && device.Price > **50**)

{

devices.Add(device);

}

}

}

**return** devices;

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.IO**;

**using** **System.Text**;

**namespace** **\_4Laboras**

{

/// <summary>

/// Class for reading and printing

/// </summary>

**public** **static** **class** **InOut**

{

/// <summary>

/// Reads the data from all the given files

/// and returns list of shops

/// </summary>

/// <param name="file"></param>

/// <returns></returns>

**public** **static** List<Shops> ReadData(**string** file)

{

List<Shops> shops = **new** List<Shops>();

**string**[] filePaths =

Directory.GetFiles(file, "\*.csv");

**foreach** (**string** path **in** filePaths)

{

shops.Add(ReadDevices(path));

}

**return** shops;

}

/// <summary>

/// Reads the data from the file and returns shop

/// </summary>

/// <param name="path"></param>

/// <returns></returns>

**private** **static** Shops **ReadDevices**(**string** path)

{

**using** (StreamReader reader = **new** StreamReader(

@path,

Encoding.UTF8))

{

**string** line;

**string** shopName = reader.ReadLine();

**string** address = reader.ReadLine();

**string** phone = reader.ReadLine();

Shops shop = **new** Shops(shopName, address, phone);

**while** ((line = reader.ReadLine()) != **null**)

{

**switch** (line[**0**])

{

**case** 'F':

shop.AddDevice(**new** Fridge(line));

**break**;

**case** 'O':

shop.AddDevice(**new** Oven(line));

**break**;

**case** 'K':

shop.AddDevice(**new** Kettle(line));

**break**;

}

}

**return** shop;

}

}

/// <summary>

/// Prints the data from the shops list

/// </summary>

/// <param name="file"></param>

/// <param name="header"></param>

/// <param name="shops"></param>

**public** **static** **void** **PrintShops**(**string** file,

**string** header,

List<Shops> shops)

{

**using** (StreamWriter writer = **new** StreamWriter(file, **true**))

{

writer.WriteLine(header);

**foreach** (Shops shop **in** shops)

{

writer.WriteLine($"{shop.ShopName}," +

$"{shop.Address},{shop.Phone}");

**foreach** (Device device **in** shop.GetDevices())

{

writer.WriteLine(device.ToString());

}

writer.WriteLine();

}

}

}

/// <summary>

/// Prints all the colors from the list

/// </summary>

/// <param name="colors"></param>

/// <param name="header"></param>

/// <param name="file"></param>

**public** **static** **void** **PrintColors**(List<**string**> colors,

**string** header,

**string** file)

{

**using** (StreamWriter writer = **new** StreamWriter(file, **true**))

{

writer.WriteLine(header);

**foreach** (**string** color **in** colors)

{

writer.WriteLine(color);

}

writer.WriteLine();

}

}

/// <summary>

/// Prints cheapest fridge, oven and kettle

/// </summary>

/// <param name="cheap"></param>

/// <param name="file"></param>

**public** **static** **void** **PrintCheap**(Device cheap, **string** file)

{

**using** (StreamWriter writer = **new** StreamWriter(file, **true**))

{

**if** (cheap != **null**)

{

writer.WriteLine(cheap.ToString());

}

writer.WriteLine();

}

}

/// <summary>

/// Prints all friges with specific width

/// </summary>

/// <param name="file"></param>

/// <param name="header"></param>

/// <param name="fridges"></param>

**public** **static** **void** **PrintSpecFriges**(**string** file,

**string** header,

List<Fridge> fridges)

{

**using** (StreamWriter writer = **new** StreamWriter(file, **true**))

{

writer.WriteLine(header);

**foreach** (Fridge fridge **in** fridges)

{

writer.WriteLine(fridge.ToString());

}

writer.WriteLine();

}

}

/// <summary>

/// Prints sorted expensive devices

/// </summary>

/// <param name="file"></param>

/// <param name="header"></param>

/// <param name="shop"></param>

**public** **static** **void** **PrintExpensive**(**string** file,

**string** header,

Shops shop)

{

**using** (StreamWriter writer = **new** StreamWriter(file, **true**))

{

writer.WriteLine(header);

**foreach** (Device device **in** shop.GetDevices())

{

writer.WriteLine(device.ToString());

}

writer.WriteLine();

}

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.IO**;

**using** **System.Web.UI.WebControls**;

**namespace** **\_4Laboras**

{

**public** **partial** **class** **WebForm1** : System.Web.UI.Page

{

**protected** **void** **Page\_Load**(**object** sender, EventArgs e)

{

Label1.Visible = **false**;

Label2.Visible = **false**;

Label3.Visible = **false**;

Label4.Visible = **false**;

Label5.Visible = **false**;

Button2.Visible = **false**;

Table1.Visible = **false**;

Table2.Visible = **false**;

Table3.Visible = **false**;

Table4.Visible = **false**;

Table5.Visible = **false**;

}

**protected** **void** **Button1\_Click**(**object** sender, EventArgs e)

{

Button1.Visible = **false**;

Button2.Visible = **true**;

Label1.Visible = **true**;

Table1.Visible = **true**;

File.Delete(Server.MapPath("Rezultatai.csv"));

File.Delete(Server.MapPath("Tilps.csv"));

File.Delete(Server.MapPath("Brangus.csv"));

List<Shops> shops =

InOut.ReadData(Server.MapPath("App\_Data"));

Session["pradiniai"] = shops;

Table1Header();

FillTable1(shops);

}

**protected** **void** **Button2\_Click**(**object** sender, EventArgs e)

{

List<Shops> shops = (List<Shops>)Session["pradiniai"];

InOut.PrintShops(Server.MapPath("Rezultatai.csv"),

"Pradiniai duomenys:",

shops);

Table1Header();

FillTable1(shops);

List<**string**> fridgeColors =

TaskUtils.Colors(shops, "fridge");

List<**string**> kettleColors =

TaskUtils.Colors(shops, "kettle");

InOut.PrintColors(fridgeColors,

"Šaldytuvų spalvos:",

Server.MapPath("Rezultatai.csv"));

InOut.PrintColors(kettleColors,

"Virdulių spalvos:",

Server.MapPath("Rezultatai.csv"));

FillTable2Friges(fridgeColors);

FillTable2Kettles(kettleColors);

Device cheapFridge =

TaskUtils.SpecialCheapDevices(shops, "fridge");

Device cheapOven =

TaskUtils.SpecialCheapDevices(shops, "oven");

Device cheapKettle =

TaskUtils.SpecialCheapDevices(shops, "kettle");

InOut.PrintCheap(cheapFridge, cheapOven,

cheapKettle, Server.MapPath("Rezultatai.csv"),

"Pigiausi įrenginiai:");

FillTable3(cheapFridge, cheapOven, cheapKettle);

List<Fridge> specFriges = TaskUtils.FrigesCapacity(shops);

InOut.PrintSpecFriges(Server.MapPath("Tilps.csv"),

"Šaldytuvai su nustatytu pločiu:",

specFriges);

FillTable4(specFriges);

List<Device> expensiveDevices = TaskUtils.ExpensiveDevices(shops);

Shops shop = **new** Shops("", "", "");

shop.AddDevices(expensiveDevices);

shop.Sort();

InOut.PrintExpensive(Server.MapPath("Brangus.csv"),

"Brangūs prietaisai:",

shop);

FillTable5(shop.GetDevices());

Button2.Visible = **false**;

Button1.Visible = **false**;

Label1.Visible = **true**;

Table1.Visible = **true**;

Label2.Visible = **true**;

Table2.Visible = **true**;

Label3.Visible = **true**;

Table3.Visible = **true**;

Label4.Visible = **true**;

Table4.Visible = **true**;

Label5.Visible = **true**;

Table5.Visible = **true**;

}

/// <summary>

/// Forms table1 header

/// </summary>

**private** **void** **Table1Header**()

{

TableCell cell = **new** TableCell();

cell.Text = "Parduotuvės duomenys:";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

Table1.Rows.Add(row);

}

/// <summary>

/// Fills table1 with data

/// </summary>

/// <param name="shops"></param>

**private** **void** **FillTable1**(List<Shops> shops)

{

**int** count = shops.Count;

**foreach** (Shops shop **in** shops)

{

TableRow one = **new** TableRow();

TableCell cellone = **new** TableCell();

cellone.Text = shop.ShopName;

one.Cells.Add(cellone);

TableRow two = **new** TableRow();

TableCell celltwo = **new** TableCell();

celltwo.Text = shop.Address;

two.Cells.Add(celltwo);

TableRow three = **new** TableRow();

TableCell cellthree = **new** TableCell();

cellthree.Text = shop.Phone;

three.Cells.Add(cellthree);

TableCell title = **new** TableCell();

title.Text = "Įrenginiai:";

TableRow extra = **new** TableRow();

extra.Cells.Add(title);

Table1.Rows.Add(one);

Table1.Rows.Add(two);

Table1.Rows.Add(three);

Table1.Rows.Add(extra);

**foreach** (Device device **in** shop.GetDevices())

{

TableCell a = **new** TableCell();

TableRow row = **new** TableRow();

a.Text = device.ToString();

row.Cells.Add(a);

Table1.Rows.Add(row);

}

**if** (count > **1**)

{

Table1Header();

count--;

}

}

}

/// <summary>

/// Fills table2 with fridge colors

/// </summary>

/// <param name="frigeColors"></param>

**private** **void** **FillTable2Friges**(List<**string**> frigeColors)

{

TableCell cell = **new** TableCell();

cell.Text = "Šaldytuvų spalvos:";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

Table2.Rows.Add(row);

**foreach** (**string** color **in** frigeColors)

{

TableCell celltwo = **new** TableCell();

celltwo.Text = color;

TableRow one = **new** TableRow();

one.Cells.Add(celltwo);

Table2.Rows.Add(one);

}

}

/// <summary>

/// Fills table2 with kettle colors

/// </summary>

/// <param name="kettleColors"></param>

**private** **void** **FillTable2Kettles**(List<**string**> kettleColors)

{

TableCell cell = **new** TableCell();

cell.Text = "Virdulių spalvos:";

TableRow row = **new** TableRow();

row.Cells.Add(cell);

Table2.Rows.Add(row);

**foreach** (**string** color **in** kettleColors)

{

TableCell celltwo = **new** TableCell();

celltwo.Text = color;

TableRow one = **new** TableRow();

one.Cells.Add(celltwo);

Table2.Rows.Add(one);

}

}

/// <summary>

/// Fills table3 with data

/// </summary>

/// <param name="a"></param>

**private** **void** **FillTable3**(Device a)

{

**if** (a != **null**)

{

TableCell one = **new** TableCell();

one.Text = a.ToString();

TableRow rowone = **new** TableRow();

rowone.Cells.Add(one);

Table3.Rows.Add(rowone);

}

}

/// <summary>

/// Fills table4 with data

/// </summary>

/// <param name="fridges"></param>

**private** **void** **FillTable4**(List<Fridge> fridges)

{

**foreach** (Fridge fridge **in** fridges)

{

TableCell one = **new** TableCell();

one.Text = fridge.ToString();

TableRow row = **new** TableRow();

row.Cells.Add(one);

Table4.Rows.Add(row);

}

}

/// <summary>

/// Fills table5 with data

/// </summary>

/// <param name="devices"></param>

**private** **void** **FillTable5**(List<Device> devices)

{

**foreach** (Device device **in** devices)

{

TableCell one = **new** TableCell();

one.Text = device.ToString();

TableRow row = **new** TableRow();

row.Cells.Add(one);

Table5.Rows.Add(row);

}

}

}

}

<%@ Page Language="C#" AutoEventWireup="true"

CodeBehind="WebForm1.aspx.cs" Inherits="\_4Laboras.WebForm1" %>

<!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="Pradiniai duomenys:"></asp:Label>

<br />

<asp:Table ID="Table1" runat="server" BorderWidth="1px" BorderStyle="Solid"

BorderColor="Black" BackColor="LightBlue" GridLines="Horizontal"></asp:Table>

<br />

<asp:Button ID="Button1" runat="server" Text="Nuskaityti duomenis"

OnClick="Button1\_Click" />

<br />

<asp:Button ID="Button2" runat="server" Text="Gauti rezultatus"

OnClick="Button2\_Click" />

<br />

<asp:Label ID="Label2" runat="server" Text="Šaldytuvų

ir virdulių spalvos:"></asp:Label>

<asp:Table ID="Table2" runat="server" BorderWidth="1px"

BorderColor="Black" BorderStyle="Solid" BackColor="LightCoral"></asp:Table>

<br />

<asp:Label ID="Label3" runat="server" Text="Pigiausi A+

klasės įrenginiai:"></asp:Label>

<asp:Table ID="Table3" runat="server" BorderWidth="1px"

BorderColor="Black" BorderStyle="Solid" BackColor="LightPink"></asp:Table>

<br />

<asp:Label ID="Label4" runat="server" Text="Šaldytuvai,

kruių plotis tarp 52 ir 56 cm:"></asp:Label>

<asp:Table ID="Table4" runat="server" BorderWidth="1px"

BorderColor="Black" BorderStyle="Solid" BackColor="LightCoral"></asp:Table>

<br />

<asp:Label ID="Label5" runat="server" Text="Brangių

prietaisų surikiuotas sąrašas:"></asp:Label>

<asp:Table ID="Table5" runat="server" BorderWidth="1px"

BorderColor="Black" BorderStyle="Solid" BackColor="LightPink"

GridLines="Horizontal"></asp:Table>

</div>

</form>

</body>

</html>

**using** **Microsoft.VisualStudio.TestTools.UnitTesting**;

**using** **FluentAssertions**;

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **\_4Laboras.Tests**

{

/// <summary>

/// Class for tests

/// </summary>

[TestClass()]

**public** **class** **ShopsTests**

{

/// <summary>

/// Checks if Shops class constructor works

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Constructor\_SetsPropertiesCorrectly**()

{

**string** shopName = "Test Shop";

**string** address = "1 Main St";

**string** phone = "555-555-5555";

Shops shop = **new** Shops(shopName, address, phone);

shop.ShopName.Should().Be(shopName);

shop.Address.Should().Be(address);

shop.Phone.Should().Be(phone);

}

/// <summary>

/// Checks if no arguments given will constructor

/// throw exception

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Constructor\_Empty**()

{

Action act = () => **new** Shops(**null**, **null**, **null**);

act.Should().Throw<ArgumentNullException>();

}

/// <summary>

/// Checks if constructor will work if only one parameter

/// is not givens

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Constructor\_NotAll\_Arguments**()

{

**string** address = "1 Main St";

**string** phone = "555-555-5555";

Shops shop = **new** Shops(**null**, address, phone);

shop.ShopName.Should().Be(**null**);

shop.Address.Should().Be(address);

shop.Phone.Should().Be(phone);

}

/// <summary>

/// Checks if AddDevice method works

/// </summary>

[TestMethod()]

**public** **void** **Shops\_AddDevice\_AddsDeviceToList**()

{

Shops shop = **new** Shops("Test Shop", "", "");

Device device = **new** TestDevice("", "", "", "", **0**);

shop.AddDevice(device);

shop.GetDevices().Count.Should().Be(**1**);

}

/// <summary>

/// Checks if AddDevice method works

/// </summary>

[TestMethod()]

**public** **void** **Shops\_AddDevice\_AddsDevicesToList**()

{

Shops shop = **new** Shops("Test Shop", "", "");

Device device1 = **new** TestDevice("", "", "", "", **0**);

Device device2 = **new** TestDevice("", "", "", "", **0**);

shop.AddDevice(device1);

shop.AddDevice(device2);

shop.GetDevices().Count.Should().Be(**2**);

}

/// <summary>

/// Checks if exception is thrown when trying to add

/// empty device

/// </summary>

[TestMethod()]

**public** **void** **Shops\_AddDevice\_AddsEmptyDevice**()

{

Shops shop = **new** Shops("Test Shop", "", "");

Action act = () => shop.AddDevice(**null**);

act.Should().Throw<ArgumentNullException>();

}

/// <summary>

/// Class for imitating the Device class, because Device

/// class cannot be reached as it is abstract

/// </summary>

**public** **class** **TestDevice** : Device

{

**public** **TestDevice**(**string** creator, **string** model, **string**

energyClass, **string** color, **decimal** price)

: **base**(creator, model, energyClass, color, price)

{

}

**public** **override** **int** **CompareTo**(Device other)

{

// Implement comparison logic for TestDevice (not

// required for this test)

**return** **base**.CompareTo(other);

}

**public** **override** **bool** **Equals**(Device other)

{

// Implement equality logic for TestDevice (not

// required for this test)

**return** **base**.Equals(other);

}

}

/// <summary>

/// Checks if adding device list to the list works

/// </summary>

[TestMethod()]

**public** **void** **Shops\_AddDevice\_AddsDeviceListToList**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

Device device1 = **new** TestDevice("", "", "", "", **0**);

Device device2 = **new** TestDevice("", "", "", "", **0**);

devices.Add(device1);

devices.Add(device2);

shop.AddDevices(devices);

shop.GetDevices().Count.Should().Be(**2**);

shop.GetDevices()[**0**].Should().Be(device1);

}

/// <summary>

/// Checks if program wont crash while trying to add empty

/// list

/// </summary>

[TestMethod()]

**public** **void** **Shops\_AddDevice\_AddsEmptyDeviceList**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

shop.AddDevices(devices);

shop.GetDevices().Count.Should().Be(**0**);

}

/// <summary>

/// Checks if GetDevices returns diveces list

/// </summary>

[TestMethod()]

**public** **void** **Shops\_GetDevices**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

Device device1 = **new** TestDevice("", "", "", "", **0**);

Device device2 = **new** TestDevice("", "", "", "", **0**);

devices.Add(device1);

devices.Add(device2);

shop.AddDevices(devices);

shop.GetDevices()[**0**].Should().Be(device1);

shop.GetDevices()[**1**].Should().Be(device2);

}

/// <summary>

/// Checks if sort and CompareTo methods are working for a

/// Fridge class

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Sort\_SortsFriges**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

devices.Add(**new** Fridge("F,Samsung,RS27T5200SR,A+,Green," +

"750.1,27.4,Freestanding,TRUE,55.5,56,36.1"));

devices.Add(**new** Fridge("F,Apple,RS27T52555SR,A+,Purpule," +

"1501,27.4,Imortal,FALSE,70.5,35.8,36.1"));

devices.Add(**new** Fridge("F,Sonny,RS27T5201SR,A,Silver,801," +

"27.4,Standing,TRUE,68.5,55,36.1"));

shop.AddDevices(devices);

shop.Sort();

shop.GetDevices()[**0**].Creator.Should().Be("Samsung");

shop.GetDevices()[**1**].Creator.Should().Be("Sonny");

shop.GetDevices()[**2**].Creator.Should().Be("Apple");

}

/// <summary>

/// Checks if sort and CompareTo methods are working for

/// an Oven class

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Sort\_SortsOvens**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

devices.Add(**new** Oven("O,Panasonic,AISNNSD967S,A+,Silver," +

"150.99,120,10"));

devices.Add(**new** Oven("O,Sonic,NNSKILD967S,A,Red,230.99," +

"1500,10"));

devices.Add(**new** Oven("O,Simple,NNSssD967S,A,Brown,339.99," +

"1201,10"));

shop.AddDevices(devices);

shop.Sort();

shop.GetDevices()[**0**].Creator.Should().Be("Panasonic");

shop.GetDevices()[**1**].Creator.Should().Be("Simple");

shop.GetDevices()[**2**].Creator.Should().Be("Sonic");

}

/// <summary>

/// Checks if sort and CompareTo methods are working for a

/// Kettle class

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Sort\_SortsKettles**()

{

Shops shop = **new** Shops("Test Shop", "", "");

List<Device> devices = **new** List<Device>();

devices.Add(**new** Kettle("K,Bruxel,KEKS123,A,Gold,19.99,1801,1.7"));

devices.Add(**new** Kettle("K,Breville,BKE820SIL,A+,Stainless " +

"Steel,159.99,1891,1.7"));

devices.Add(**new** Kettle("K,Maxim,LOLMASK,A,Grey,109.99,1701,1.7"));

shop.AddDevices(devices);

shop.Sort();

shop.GetDevices()[**0**].Creator.Should().Be("Maxim");

shop.GetDevices()[**1**].Creator.Should().Be("Bruxel");

shop.GetDevices()[**2**].Creator.Should().Be("Breville");

}

/// <summary>

/// Checks if sort method throws the exception when trying to sort

/// empty list

/// </summary>

[TestMethod()]

**public** **void** **Shops\_Sort\_EmptyList**()

{

Shops shop = **new** Shops("Test Shop", "", "");

Action act = () => shop.Sort();

act.Should().Throw<ArgumentNullException>();

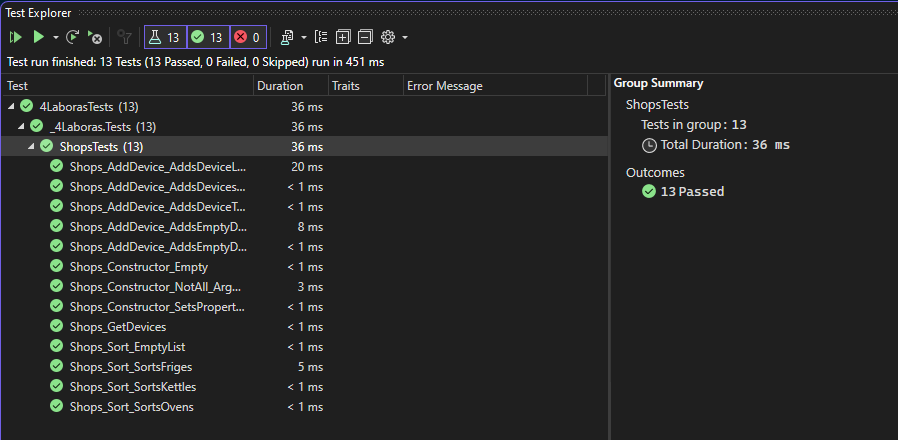
}

}

}

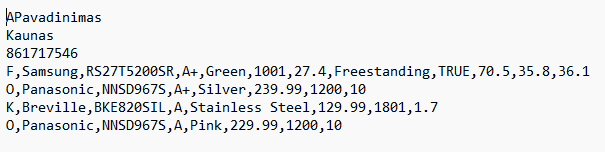
## Pradiniai duomenys ir rezultatai

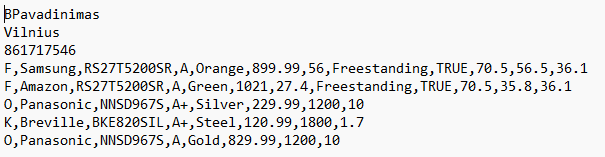
Testų rezultatai:

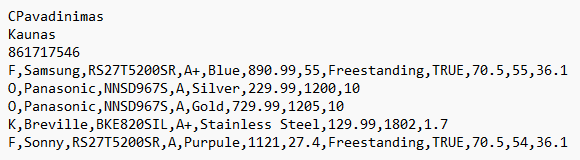


### Duomenys ir rezultatai 1

Pradiniai duomenys:

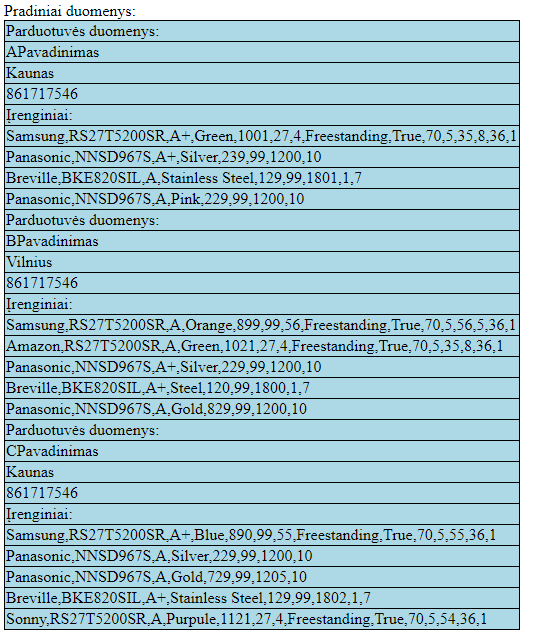




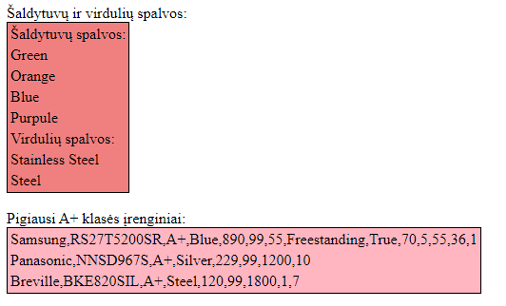


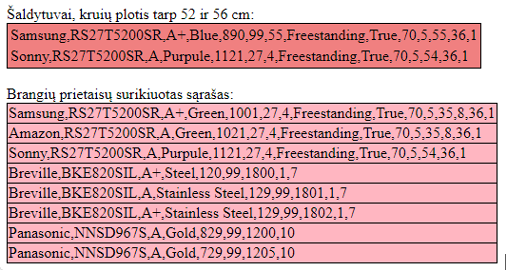
Šiais duomenimis tikrinama ar teisingai yra nuskaitomi ir spausdinami duomenys. Tikrinama ar teisingai yra atliekamas rūšiavimas bei visų rūšių duomenų atrinkimas.

Pradiniai duomenys web:

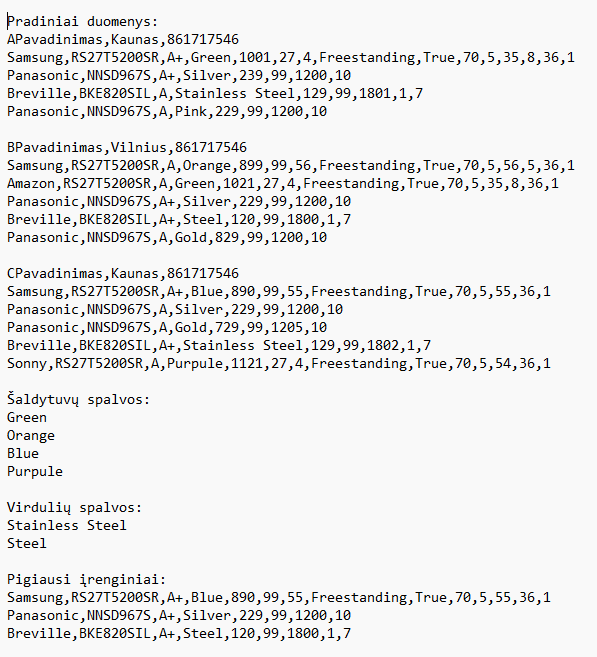


Rezultatai web:

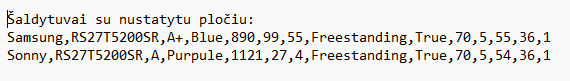




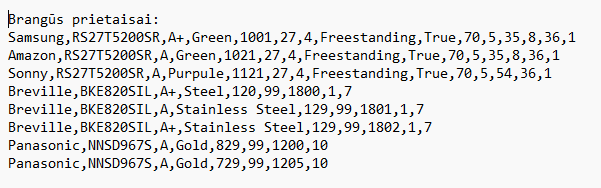
Rezultatų tekstinis failas („Rezultatai.csv“):



Rezultatų tekstinis failas („Tilps.csv“):

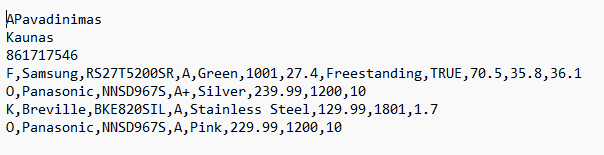


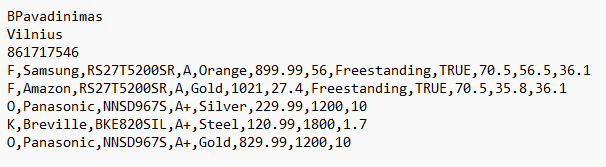
Rezultatų tekstinis failas („Brangus.csv“):

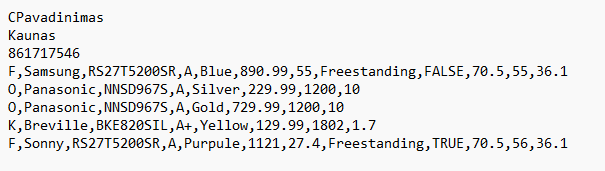


### Duomenys ir rezultatai 2

Pradiniai duomenys:

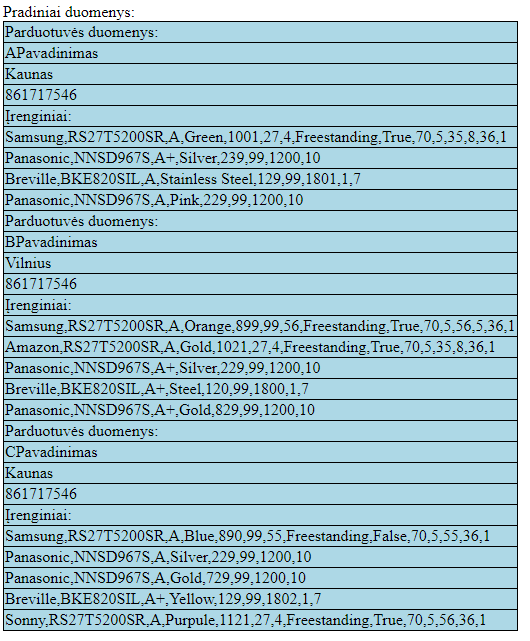




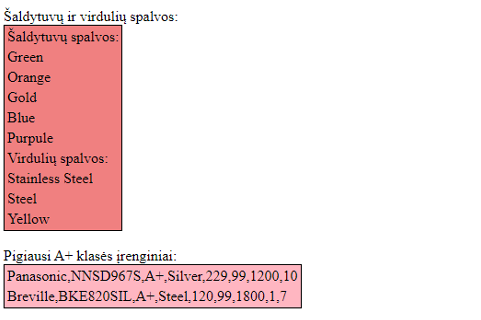


Šiais duomenimis tikrinama ar programa veiks teisingai pakeitus duomenis.

Pradiniai duomenys web:

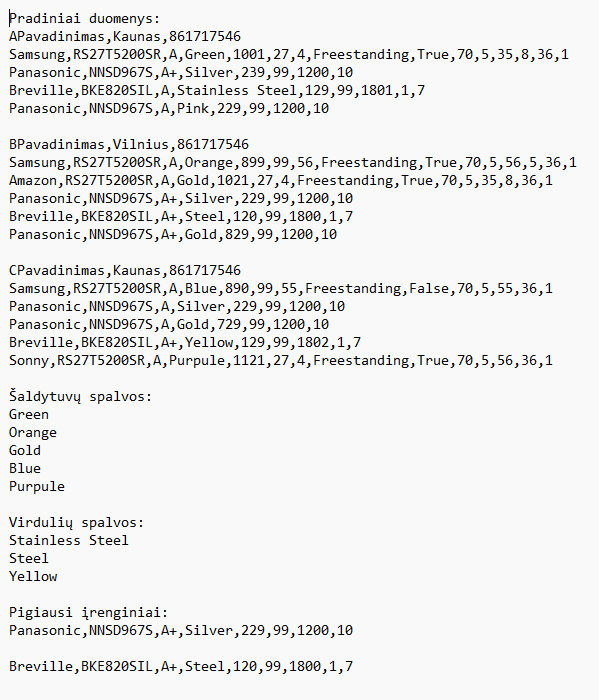


Rezultatai web:

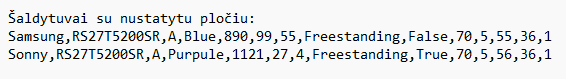




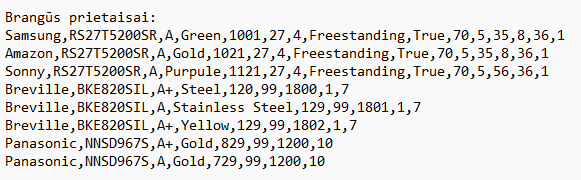
Rezultatų tekstinis failas („Rezultatai.csv“):



Rezultatų tekstinis failas („Tilps.csv“):



Rezultatų tekstinis failas („Brangus.csv“):



## Dėstytojo pastabos

# Deklaratyvusis programavimas (L5)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos