

Kauno technologijos universitetas

Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Studentas

Prof. Vacius Jusas

Kaunas 2022

TURINYS

1. Rekursija (L1) 4

1.1. Darbo užduotis 4

1.2. Grafinės vartotojo sąsajos schema 5

1.3. Sąsajoje panaudotų komponentų keičiamos savybės 5

1.4. Klasių diagrama 6

1.5. Programos vartotojo vadovas 6

1.6. Programos tekstas 6

1.7. Pradiniai duomenys ir rezultatai 13

1.8. Dėstytojo pastabos 17

2. Dinaminis atminties valdymas (L2) 18

2.1. Darbo užduotis 18

2.2. Grafinės vartotojo sąsajos schema 18

2.3. Sąsajoje panaudotų komponentų keičiamos savybės 18

2.4. Klasių diagrama 18

2.5. Programos vartotojo vadovas 18

2.6. Programos tekstas 18

2.7. Pradiniai duomenys ir rezultatai 18

2.8. Dėstytojo pastabos 19

3. Bendrinės klasės ir testavimas (L3) 20

3.1. Darbo užduotis 20

3.2. Grafinės vartotojo sąsajos schema 20

3.3. Sąsajoje panaudotų komponentų keičiamos savybės 20

3.4. Klasių diagrama 20

3.5. Programos vartotojo vadovas 20

3.6. Programos tekstas 20

3.7. Pradiniai duomenys ir rezultatai 20

3.8. Dėstytojo pastabos 21

4. Polimorfizmas ir išimčių valdymas (L4) 22

4.1. Darbo užduotis 22

4.2. Grafinės vartotojo sąsajos schema 22

4.3. Sąsajoje panaudotų komponentų keičiamos savybės 22

4.4. Klasių diagrama 22

4.5. Programos vartotojo vadovas 22

4.6. Programos tekstas 22

4.7. Pradiniai duomenys ir rezultatai 22

4.8. Dėstytojo pastabos 23

5. Deklaratyvusis programavimas (L5) 24

5.1. Darbo užduotis 24

5.2. Grafinės vartotojo sąsajos schema 24

5.3. Sąsajoje panaudotų komponentų keičiamos savybės 24

5.4. Klasių diagrama 24

5.5. Programos vartotojo vadovas 24

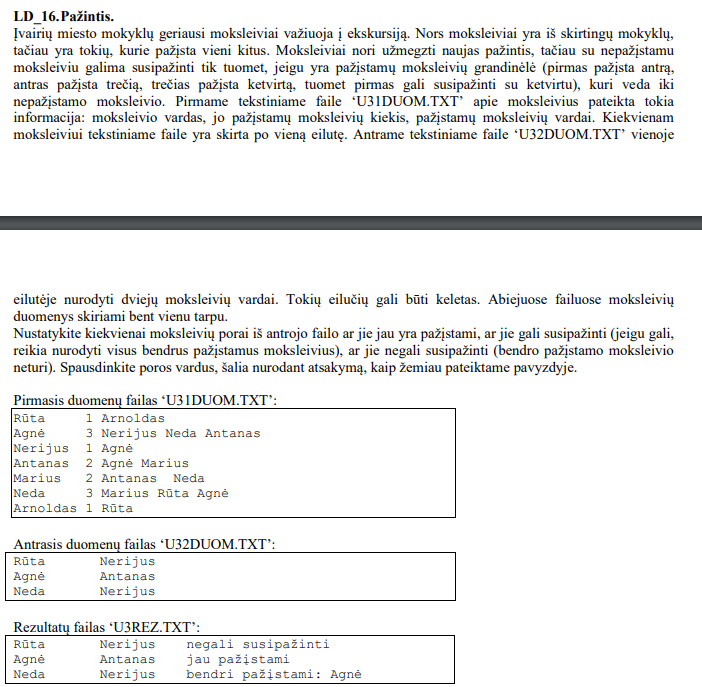
5.6. Programos tekstas 24

5.7. Pradiniai duomenys ir rezultatai 24

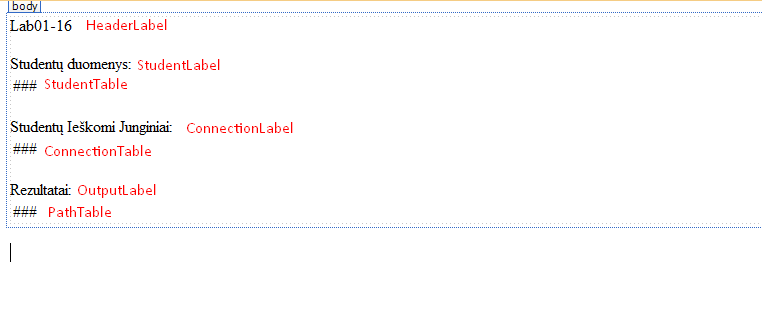
5.8. Dėstytojo pastabos 25

# Rekursija (L1)

## Darbo užduotis



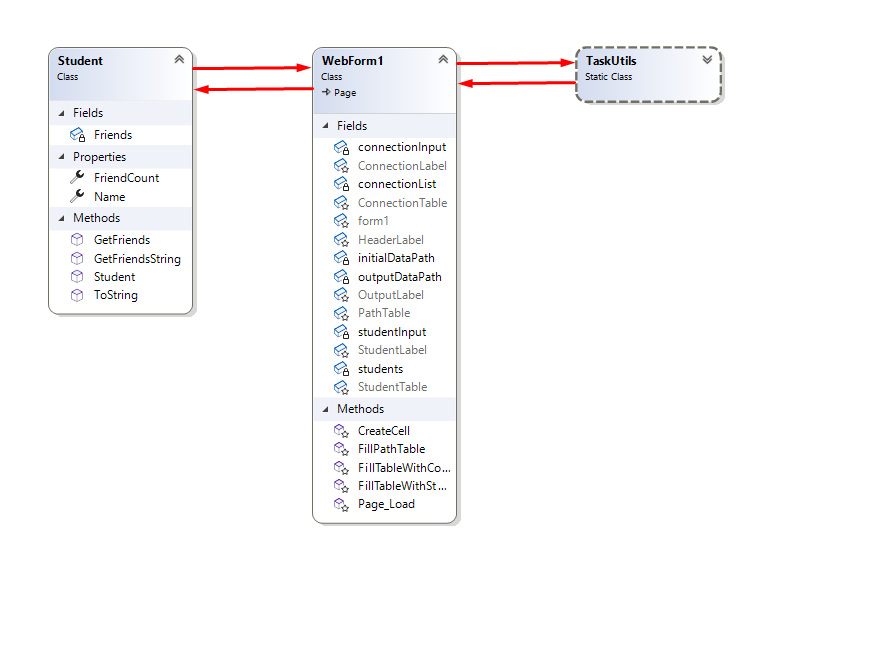
## Grafinės vartotojo sąsajos schema



## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| HeaderLabel | Text | "Lab01-16" |
| StudentLabel | Text | "Studentų duomenys:" |
| ConnectionLabel | Text | "Studentų Ieškomi Junginiai:" |
| OutputLabel | Text | "Rezultatai:" |

## Klasių diagrama



## Programos vartotojo vadovas

Atsidarius programą, programa nuskaito App\_Data/students.txt ir App\_Data/connections.txt. Naudojant tą informaciją, parašo visą informaciją į StudentTable, ConnectionTable, PathTable su duota ir apskaičiuota informacija.

## Programos tekstas

InOutUtils.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// InOutUtils class for reading and writing data from/to a file

/// </summary>

public static class InOutUtils

{

/// <summary>

/// Creates a new empty file, ready for appending data

/// </summary>

/// <param name="path">path to the file</param>

public static void CreateFile(string path)

{

using (FileStream fs = new FileStream(path, FileMode.Create))

new StreamWriter(fs, encoding: System.Text.Encoding.UTF8).Close();

}

/// <summary>

/// appends initial student data to TXT file

/// </summary>

/// <param name="students">List of all students (Student object)</param>

/// <param name="path">path to the file where information will be appended</param>

public static void AppendInitStudents(List<Student> students, string path)

{

using (StreamWriter sr = new StreamWriter(path, append: true))

{

sr.WriteLine("Studentai ir jų draugai");

sr.WriteLine($"{"Studentas",-20}|{"Draugų kiekis",-20}|{"Draugai:"}");

foreach (Student student in students)

sr.WriteLine(student);

sr.WriteLine();

}

}

/// <summary>

/// Appends initial connection data to output file

/// </summary>

/// <param name="connections">List of Tuples(string, string) that work as nodes from student a to student b while using DFS</param>

/// <param name="path">path to the file where to append initial data</param>

public static void AppendInitData(List<Tuple<string, string>> connections, string path)

{

using (StreamWriter sr = new StreamWriter(path, append: true))

{

sr.WriteLine("Studentai ir jų ieškomi draugai:");

sr.WriteLine($"{"Studentas",-20} {"Ieškomas draugas",-20}");

foreach (Tuple<string, string> connection in connections)

sr.WriteLine($"{connection.Item1,-20} {connection.Item2,-20}");

sr.WriteLine();

}

}

/// <summary>

/// Appends output connection data to output file

/// </summary>

/// <param name="students">Dictionary, key -> string, name of the student, value -> Student class object of the student</param>

/// <param name="connections">List of tuples(string, string) that is compromised of student names that work as nodes that are used for DFS</param>

/// <param name="outputPath">output path to the txt file where data will be APPENDED</param>

public static void AppendConnectionResults(Dictionary<string, Student> students, List<Tuple<string, string>> connections, string outputPath)

{

using (StreamWriter sr = new StreamWriter(outputPath))

{

sr.WriteLine("Draugai ir jų junginiai, bei keliai:");

sr.WriteLine($"{"Draugas",-20}|{"Ieškomas draugas:",-20}|{"Kelias:"}");

foreach (Tuple<string, string> connection in connections)

{

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

studentPath.Add(connection.Item1);

studentPath = TaskUtils.FindConnection(connection.Item1, connection.Item2, studentPath, students);

string pathText = TaskUtils.CreatePathText(studentPath);

sr.WriteLine($"{connection.Item1,-20}|{connection.Item2,-20}|{pathText}");

}

}

}

/// <summary>

/// Creates a name to Student class object relation dictionary

/// </summary>

/// <param name="path">Path to the the text file containing the data</param>

/// <returns>Dictionary(key -> string, value -> Student class object) </returns>

public static Dictionary<string, Student> ReadStudents(string path)

{

Dictionary<string, Student> students = new Dictionary<string, Student>();

using (StreamReader sr = new StreamReader(path))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] elements = line.Split(' ');

string name = elements[0];

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

for (int i = 2; i < elements.Length; i++)

friends.Add(elements[i]);

students.Add(name, new Student(name, friends));

}

}

return students;

}

/// <summary>

/// Gets the connections of students

/// </summary>

/// <param name="path">.txt file to the input</param>

/// <returns>List of Tupples(string, string)</returns>

public static List<Tuple<string, string>> ReadConnections(string path)

{

List<Tuple<string, string>> conncetions = new List<Tuple<string, string>>();

using (StreamReader sr = new StreamReader(path))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] elements = line.Split(' ');

conncetions.Add(new Tuple<string, string>(elements[0], elements[1]));

}

}

return conncetions;

}

}

}

TaskUtils.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// TaskUtils class for extra (backend) computation functions

/// </summary>

public static class TaskUtils

{

/// <summary>

/// Recursive implementation of DFS

/// </summary>

/// <param name="start">Start of the person</param>

/// <param name="end">End of the person</param>

/// <param name="path">path to current position from initial start</param>

/// <param name="students">Dictionary, key: string (name of the student), value Student class object</param>

/// <returns>List of strings, that create a path from student a to b</returns>

public static List<string> FindConnection(string start, string end, List<string> path, Dictionary<string, Student> students)

{

Student curr = students[start];

List<string> outputPath = null;

foreach(string next in curr.GetFriends())

{

if (next == end)

return path;

else if (path.Contains(next)) // Checks if the current node has been visited, so it does not loop

continue;

Student nextStudent = students[next];

List<string> pathCopy = path.CopyPath();

pathCopy.Add(next);

List<String> pathToEnd = FindConnection(next, end, pathCopy, students); // Recursion Call

if(outputPath == null || (pathToEnd != null && pathToEnd.Count < outputPath.Count))

outputPath = pathToEnd;

}

return outputPath; // Did not found the path

}

/// <summary>

/// Deep copies a string list

/// </summary>

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

/// <returns>string list</returns>

private static List<string> CopyPath(this List<string> path)

{

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

foreach (string s in path)

copy.Add(s);

return copy;

}

/// <summary>

/// Creates connection depending on the path

/// </summary>

/// <param name="path"> List of strings that the path is compromised of </param>

/// <returns>a string form of the path from student a to student b</returns>

public static string CreatePathText(List<string> path)

{

if (path == null)

return "negali susipažinti";

else if (path.Count == 1)

return "jau pažįstami";

else

{

path.RemoveAt(0);

return $"bendri pažįstami: {String.Join(" ", path)}";

}

}

}

}

Student.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// Student Class Data Object that stores the name and connection

/// </summary>

public class Student

{

public string Name { get; set; }

private List<string> Friends;

public int FriendCount { get { return Friends.Count; } }

/// <summary>

/// Constructor

/// </summary>

public Student(string name, List<string> friends)

{

Name = name;

Friends = new List<string>();

foreach (string friend in friends)

Friends.Add(friend);

}

/// <summary>

/// Copies friends

/// </summary>

/// <returns>Deep copy of Friends List</returns>

public List<string> GetFriends()

{

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

foreach (string friend in Friends)

friendList.Add(friend);

return friendList;

}

/// <summary>

/// Transforms Friends list into a string seperated by spaces

/// </summary>

/// <returns>string of all friends</returns>

public string GetFriendsString() => String.Join(" ", Friends);

/// <summary>

/// ToString Override

/// </summary>

/// <returns>string version of the object: Name, Friend Count, Friends</returns>

public override string ToString()

{

return $"{Name,-20}|{Friends.Count,20}|{GetFriendsString()}";

}

}

}

WebForm1.aspx:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Lab01.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="HeaderLabel" runat="server" Text="Lab01-16"></asp:Label>

<br />

<br />

<asp:Label ID="StudentLabel" runat="server" Text="Studentų duomenys:"></asp:Label>

<br />

<asp:Table ID="StudentTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="ConnectionLabel" runat="server" Text="Studentų Ieškomi Junginiai:"></asp:Label>

<br />

<asp:Table ID="ConnectionTable" runat="server">

</asp:Table>

<br />

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

<br />

<asp:Table ID="PathTable" runat="server">

</asp:Table>

</div>

</form>

</body>

</html>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Lab01

{

public partial class WebForm1 : System.Web.UI.Page

{

private string studentInput = @"App\_Data/students.txt";

private string connectionInput = @"App\_Data/connections.txt";

private string initialDataPath = @"App\_Data/initial\_data.txt";

private string outputDataPath = @"App\_Data/result.txt";

private Dictionary<string, Student> students;

private List<Tuple<string, string>> connectionList;

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

{

// Initial Data

InOutUtils.CreateFile(Server.MapPath(initialDataPath));

students = InOutUtils.ReadStudents(Server.MapPath(studentInput));

FillTableWithStudents(new List<Student>(students.Values),

StudentTable);

InOutUtils.AppendInitStudents(new List<Student>(students.Values),

Server.MapPath(initialDataPath));

connectionList = InOutUtils.ReadConnections(Server.MapPath(connectionInput));

FillTableWithConnections(connectionList,

ConnectionTable);

InOutUtils.AppendInitData(connectionList,

Server.MapPath(initialDataPath));

FillPathTable(students, connectionList, PathTable);

InOutUtils.CreateFile(Server.MapPath(outputDataPath));

InOutUtils.AppendConnectionResults(students,

connectionList,

Server.MapPath(outputDataPath));

}

/// <summary>

/// Used to show initial Student Data

/// </summary>

/// <param name="students">List Student data type</param>

/// <param name="table">Table Object data type</param>

protected void FillTableWithStudents(List<Student> students, Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Studentas"));

row.Cells.Add(CreateCell("Draugų Kiekis"));

row.Cells.Add(CreateCell("Studentų Draugai:"));

table.Rows.Add(row);

foreach (Student student in students)

{

row = new TableRow();

row.Cells.Add(CreateCell(student.Name));

row.Cells.Add(CreateCell(student.FriendCount.ToString()));

row.Cells.Add(CreateCell(student.GetFriendsString()));

table.Rows.Add(row);

}

}

/// <summary>

/// Used to show initial connection data

/// </summary>

/// <param name="connections">List of Tuples compromised of string, string containing the initial node and end node to use for DFS</param>

/// <param name="table">Table object data type</param>

protected void FillTableWithConnections(List<Tuple<string,

string>> connections,

Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Draugas"));

row.Cells.Add(CreateCell("Ieškomas Draugas"));

table.Rows.Add(row);

foreach (Tuple<string, string> connection in connections)

{

row = new TableRow();

row.Cells.Add(CreateCell(connection.Item1));

row.Cells.Add(CreateCell(connection.Item2));

table.Rows.Add(row);

}

}

/// <summary>

/// Creates A cell with provided Text

/// </summary>

/// <param name="text">text to be added to the Cell.text param</param>

/// <returns>TableCell object</returns>

protected TableCell CreateCell(string text)

{

TableCell cell = new TableCell();

cell.Style.Add("padding", "5px");

cell.Text = text;

return cell;

}

/// <summary>

/// Fills the table with paths from student a to b

/// </summary>

/// <param name="students"> Dictionary, key -> string of the student, value -> student object</param>

/// <param name="connections">List of Tuples compromised of string, string containing the initial node and end node to use for DFS</param>

/// <param name="table">Table object where the data will be added</param>

protected void FillPathTable(Dictionary<string, Student> students,

List<Tuple<string, string>> connections,

Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Draugas"));

row.Cells.Add(CreateCell("Ieškomas Draugas"));

row.Cells.Add(CreateCell("Kelias: "));

table.Rows.Add(row);

foreach (Tuple<string, string> connection in connections)

{

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

path.Add(connection.Item1);

path = TaskUtils.FindConnection(connection.Item1,

connection.Item2,

path, students);

string pathText = TaskUtils.CreatePathText(path);

row = new TableRow();

row.Cells.Add(CreateCell(connection.Item1));

row.Cells.Add(CreateCell(connection.Item2));

row.Cells.Add(CreateCell(pathText));

table.Rows.Add(row);

}

}

}

}

## Pradiniai duomenys ir rezultatai

Pradiniai Duomenys 1:

students.txt:

Rūta 1 Arnoldas

Agnė 3 Nerijus Neda Antanas

Nerijus 1 Agnė

Antanas 2 Agnė Marius

Marius 2 Antanas Neda

Neda 3 Marius Rūta Agnė

Arnoldas 1 Rūta

connections.txt:

Rūta Nerijus

Agnė Antanas

Neda Nerijus

Rezultatai 1:

Vartotojo sąsaja:



initial\_data.txt:

Studentai ir jų draugai

Studentas |Draugų kiekis |Draugai:

Rūta | 1|Arnoldas

Agnė | 3|Nerijus Neda Antanas

Nerijus | 1|Agnė

Antanas | 2|Agnė Marius

Marius | 2|Antanas Neda

Neda | 3|Marius Rūta Agnė

Arnoldas | 1|Rūta

Studentai ir jų ieškomi draugai:

Studentas Ieškomas draugas

Rūta Nerijus

Agnė Antanas

Neda Nerijus

Result.txt:

Draugai ir jų junginiai, bei keliai:

Draugas |Ieškomas draugas: |Kelias:

Rūta |Nerijus |negali susipažinti

Agnė |Antanas |jau pažįstami

Neda |Nerijus |bendri pažįstami: Agnė

Pradiniai Duomenys 2:

students.txt:

a 2 g b

b 2 a c

c 2 b f

d 1 e

e 1 d

f 2 h c

g 2 a h

h 2 g f

connections.txt:

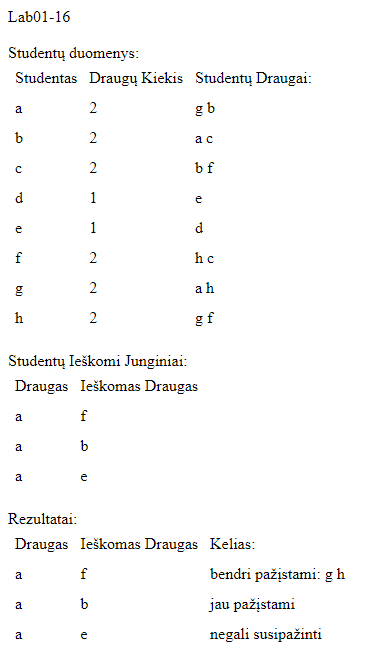
a f

a b

a e

Rezultatai 2:

Vartotojo Sąsaja:



Initial\_data.txt:

Studentai ir jų draugai

Studentas |Draugų kiekis |Draugai:

a | 2|g b

b | 2|a c

c | 2|b f

d | 1|e

e | 1|d

f | 2|h c

g | 2|a h

h | 2|g f

Studentai ir jų ieškomi draugai:

Studentas Ieškomas draugas

a f

a b

a e

result.txt:

Draugai ir jų junginiai, bei keliai:

Draugas |Ieškomas draugas: |Kelias:

a |f |bendri pažįstami: g h

a |b |jau pažįstami

a |e |negali susipažinti

## Dėstytojo pastabos

# Dinaminis atminties valdymas (L2)

## 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

# Bendrinės klasės ir testavimas (L3)

## 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

# Polimorfizmas ir išimčių valdymas (L4)

## 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

# 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