|  |
| --- |
| **KAUNO TECHNOLOGIJOS UNIVERSITETAS <FAKULTETAS>**                        **OBJEKTINIS PROGRAMAVIMAS I (P175B118)**  ***Laboratorinio darbo ataskaita***              Atliko:  IFF-6/8 gr. studentas  Tadas Laurinaitis  2016 m. lapkričio 7 d.  Priėmė:  Dėstytojo Pareigos Vardas Ir Pavardė  Lektorius Mindaugas Vasiljevas |

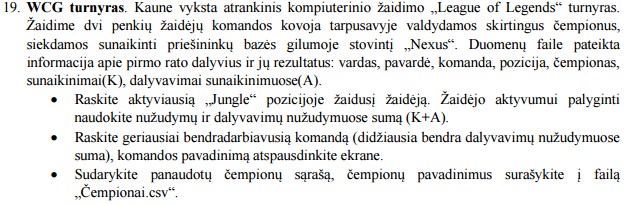
|  |  |
| --- | --- |
| **KAUNAS 2016** | 2 |

# TURINYS

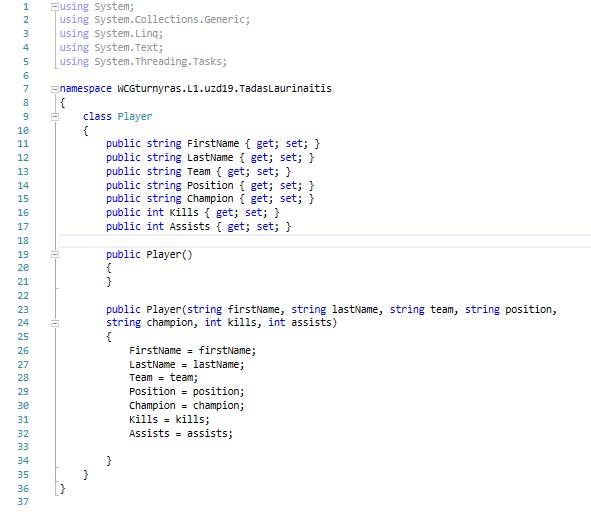
1. **Objektų rinkinys ................................................................................................. 4**
   1. Darbo užduotis .................................................................................................................. 4
   2. Programos tekstas .............................................................................................................. 4 1.3. Pradiniai duomenys ir rezultatai ........................................................................................ 8 **2.****Konteineris ........................................................................................................... 9**
   3. Darbo užduotis .................................................................................................................. 9
   4. Programos tekstas .............................................................................................................. 9 2.3. Pradiniai duomenys ir rezultatai ........................................................................................ 9 **3.****Paveldėjimas ...................................................................................................... 10**
   5. Darbo užduotis ................................................................................................................ 10
   6. Programos tekstas ............................................................................................................ 10 3.3. Pradiniai duomenys ir rezultatai ...................................................................................... 15 **4.** **Teksto analizė ir redagavimas ......................................................................... 17**
   7. Darbo užduotis ................................................................................................................ 17
   8. Programos tekstas ............................................................................................................ 17 4.3. Pradiniai duomenys ir rezultatai ...................................................................................... 17 **5.** **Sudėtingesnis konteineris ................................................................................. 18**
   9. Darbo užduotis ................................................................................................................ 18
   10. Programos tekstas ............................................................................................................ 18
   11. Pradiniai duomenys ir rezultatai ...................................................................................... 18

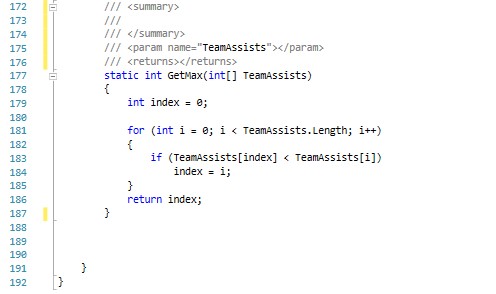
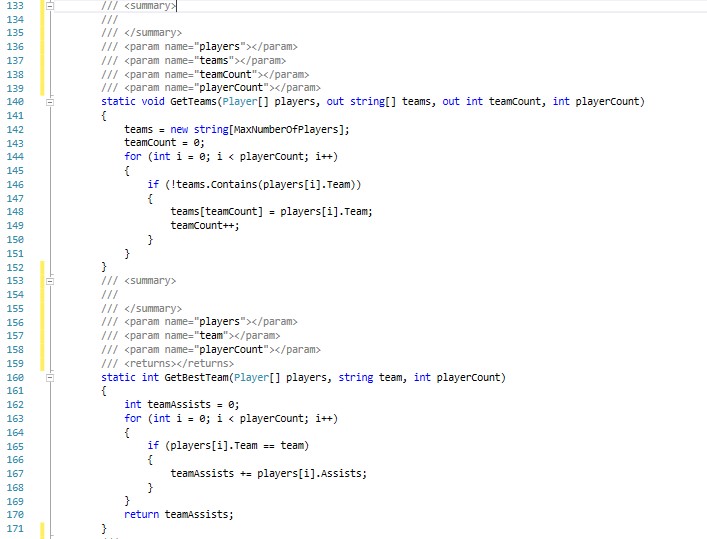
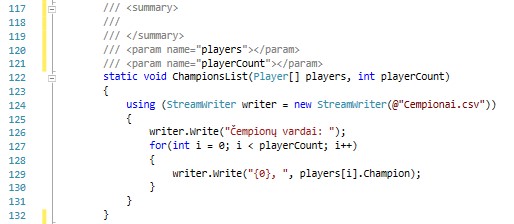
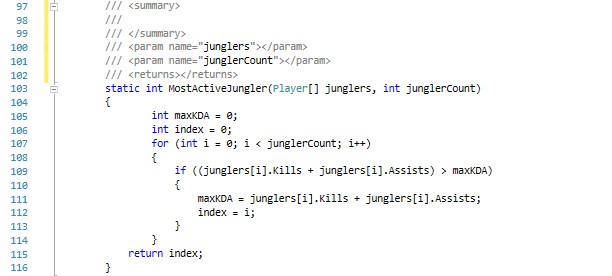
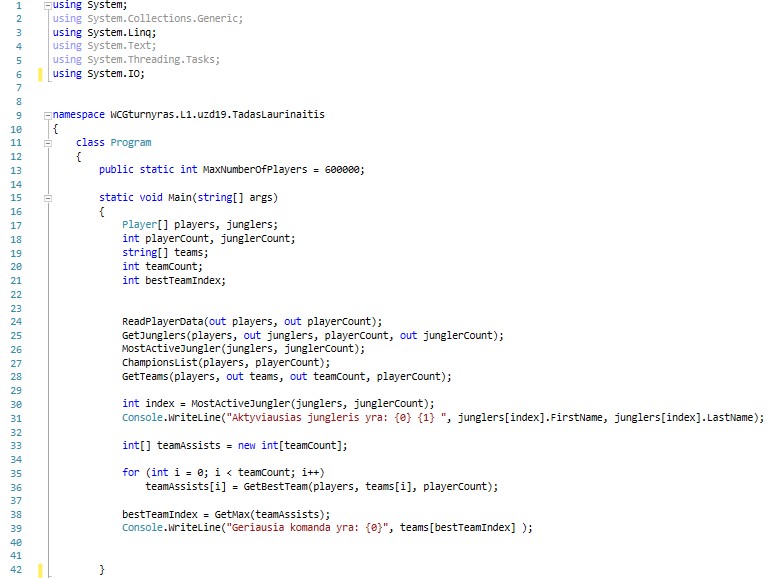
# 1. Objektų rinkinys

## 1.1. Darbo užduotis



## 1.2. Programos tekstas





## 1.3. Pradiniai duomenys ir rezultatai

Duomenys faile „Players.csv“:

Kevin;Yarnell;TeamSoloMid;Top;Gangplank;13;18;

Dennis;Johnsen;TeamSoloMid;Jungler;Kha'zix;7;24;

Soren;Bjerg;TeamSoloMid;Mid;Ahri;21;8;

Yiliang;Peng;TeamSoloMid;ADC;Vayne;19;9;

Vincent;Wang;TeamSoloMid;Support;Thresh;2;29;

Mateusz;Szkudlarek;Fnatic;Top;Gnar;5;16;

Lee;Da-yoon;Fnatic;Jungler;Sejuani;4;22;

Fabian;Diepstraten;Fnatic;Mid;Lux;15;17;

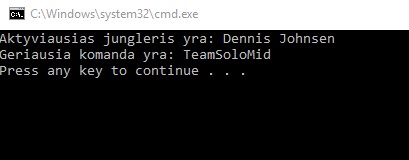
Martin;Larsson;Fnatic;ADC;Jhin;22;7;

Bora;Kim;Fnatic;Support;Lulu;7;19;

Duomenys rezultatų faile “Cempionai.csv”:

Čempionų vardai: Gangplank, Kha'zix, Ahri, Vayne, Thresh, Gnar, Sejuani, Lux, Jhin, Lulu,

Rezultatai išvedami konsolėje:



# 2. Konteineris

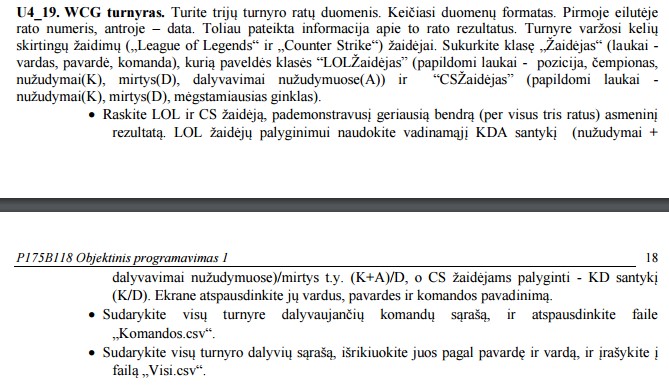
***2.1. Darbo užduotis***

***2.2. Programos tekstas***

***2.3. Pradiniai duomenys ir rezultatai***

# 3. Paveldėjimas

## 3.1. Darbo užduotis



## 3.2. Programos tekstas

using System;

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

namespace U4WCG19

{

class Program

{

const int maxNumberOfPlayers = 90; const int numberOfBranches = 3;

static void Main(string[] args)

{

Branch[] branches = new Branch[numberOfBranches]; branches[0] = new Branch(1); branches[1] = new Branch(2); branches[2] = new Branch(3); int index1; int index2; int index3; int index4;

string[] teams = new string[maxNumberOfPlayers];

PlayersContainer Lolplayers = new PlayersContainer(maxNumberOfPlayers);

string[] filePath = Directory.GetFiles(Directory.GetCurrentDirectory(), "\*.csv"); foreach (string file in filePath)

{

ReadPlayerData(file, branches);

}

int teamCount = GetTeams(branches, teams);

PrintTeams(teams, teamCount, @"..\..\Komandos.csv");

BestLOLPlayer(branches, out index1, out index2);

LOLPlayers player = branches[index1].LOLplayers.GetPlayer(index2) as LOLPlayers; Console.WriteLine("Geriausias Lol zaidejas yra: {0} {1} is {2} komandos", player.name, player.surname, player.team);

BestCSPlayer(branches, out index3, out index4);

CSGOPlayers players = branches[index3].CSGOplayers.GetPlayer(index4) as CSGOPlayers;

Console.WriteLine("Geriausias Lol zaidejas yra: {0} {1} is {2} komandos", players.name, players.surname, players.team);

PlayersContainer participants = AllParticipants(branches);

PrintSortedParticipants(participants, @"..\..\Visi.csv");

AllParticipants(branches);

participants = SortParticipants(participants.player, participants.Count);

}

/// <summary>

/// gauna branches masyvo atitikmeni kuris atitinka tour skaiciu

/// </summary>

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

/// <param name="tour"></param> /// <returns>branches</returns>

private static Branch GetBranchByTour(Branch[] branches, int tour)

{

for (int i = 0; i < numberOfBranches; i++)

{

if (branches[i].tour == tour)

{

return branches[i];

} }

return null;

}

/// <summary>

/// Nuskaito kiekvieno zaidejo duomenis

/// </summary>

/// <param name="file"></param> /// <param name="branches"></param>

private static void ReadPlayerData(string file, Branch[] branches)

{

int tour = 0;

using (StreamReader reader = new StreamReader(@file))

{

string line = null; line = reader.ReadLine(); if (line != null)

{

tour = int.Parse(line);

}

Branch branch = GetBranchByTour(branches, tour); line = reader.ReadLine(); string date = line;

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

{

string[] values = line.Split(';'); string game = values[0]; string name = values[1]; string surname = values[2]; string team = values[3]; if (game == "LOL")

{

string position = values[4]; string champion = values[5]; int kills = int.Parse(values[6]); int deaths = int.Parse(values[7]); int assists = int.Parse(values[8]); LOLPlayers LOLPlayer = new LOLPlayers(game, name, surname, team, position, champion, kills, deaths, assists);

branch.LOLplayers.AddPlayer(LOLPlayer);

}

else if (game == "CS")

{

int kills = int.Parse(values[4]); int deaths = int.Parse(values[5]); string favouriteWeapon = values[6];

CSGOPlayers CSGOPlayer = new CSGOPlayers(game, name, surname, team, kills, deaths, favouriteWeapon);

branch.CSGOplayers.AddPlayer(CSGOPlayer);

}

}

}

}

/// <summary>

/// Suranda visas komandas

/// </summary>

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

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

/// <returns>teamCount teams masyve</returns> static int GetTeams(Branch[] branches, string[] teams)

{

int teamCount = 1; bool same;

teams[0] = branches[0].LOLplayers.GetPlayer(0).team; for (int i = 0; i < branches[0].LOLplayers.Count; i++)

{

same = false;

for (int j = 0; j < teamCount; j++)

{

if (teams[j] == branches[0].LOLplayers.GetPlayer(i).team) same = true;

}

if (same == false)

{

teams[teamCount++] = branches[0].LOLplayers.GetPlayer(i).team;

} }

for (int i = 0; i < branches[1].CSGOplayers.Count; i++)

{

same = false;

for (int j = 0; j < teamCount; j++)

{

if (teams[j] == branches[1].CSGOplayers.GetPlayer(i).team) same = true;

}

if (same == false)

{

teams[teamCount++] = branches[1].CSGOplayers.GetPlayer(i).team;

} }

for (int i = 0; i < branches[2].LOLplayers.Count; i++)

{

same = false;

for (int j = 0; j < teamCount; j++)

{

if (teams[j] == branches[2].LOLplayers.GetPlayer(i).team) same = true;

}

if (same == false)

{

teams[teamCount++] = branches[2].LOLplayers.GetPlayer(i).team;

} }

return teamCount;

}

/// <summary>

/// israso visas komandas i textini faila

/// </summary>

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

/// <param name="teamCount"></param> /// <param name="file"></param>

static void PrintTeams(string[] teams, int teamCount, string file)

{

using (StreamWriter writer = new StreamWriter(file))

{

for (int i = 0; i < teamCount; i++)

{

writer.WriteLine("{0}", teams[i]);

}

}

}

/// <summary>

/// Suranda geriausio LOL playerio indexa brances masyve ir indexa tarp LOL players klases objektu /// </summary>

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

/// <param name="index1"></param> /// <param name="index2"></param>

static void BestLOLPlayer(Branch[] branches, out int index1, out int index2)

{

int maxKDA = 0; index1 = 0; index2 = 0;

for(int i = 0; i < numberOfBranches; i++)

{

for(int j = 0; j < branches[i].LOLplayers.Count; j++)

{

LOLPlayers players = branches[i].LOLplayers.GetPlayer(j) as LOLPlayers; if(maxKDA < players.kills + players.assists)

{

maxKDA = players.kills + players.assists; index1 = i; index2 = j;

}

}

}

}

/// <summary>

/// Suranda geriausio LOL playerio indexa brances masyve ir indexa tarp LOL players klases objektu /// </summary>

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

/// <param name="index3"></param> /// <param name="index4"></param>

static void BestCSPlayer(Branch[] branches, out int index3, out int index4)

{

int maxKD = 0; index3 = 0; index4 = 0;

for (int i = 0; i < numberOfBranches; i++)

{

for (int j = 0; j < branches[i].CSGOplayers.Count; j++) {

CSGOPlayers players = branches[i].CSGOplayers.GetPlayer(j) as CSGOPlayers; if (maxKD < players.kills/players.deaths)

{

maxKD = players.kills / players.deaths; index3 = i; index4 = j;

}

}

}

}

/// <summary>

/// suranda visus sio turnyro dalyvius is visu zaidimu

/// </summary>

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

/// <returns></returns>

static PlayersContainer AllParticipants(Branch[] branches)

{

PlayersContainer participants = new PlayersContainer(maxNumberOfPlayers); for(int i = 0; i < numberOfBranches; i++)

{

for(int j = 0; j < branches[i].LOLplayers.Count; j++ )

{

participants.AddPlayer(branches[i].LOLplayers.GetPlayer(j));

} }

for (int i = 0; i < numberOfBranches; i++)

{

for (int j = 0; j < branches[i].CSGOplayers.Count; j++)

{

participants.AddPlayer(branches[i].CSGOplayers.GetPlayer(j));

} }

return participants;

}

/// <summary>

/// surikiuoja dalyvius is ALLParticipants funkcijos pagal pavarde ir varda

/// </summary>

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

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

/// <returns></returns>

static Players[] SortParticipants(Players[] participants, int Count)

{

Players players = new Players(); for (int i = 0; i < Count; i++)

{

for (int j = 1; j < Count; j++)

{

if (participants[i].surname.CompareTo(participants[j].surname)>0)

{

players = participants[i]; participants[i] = participants[j]; participants[j] = players;

}

}

}

return participants;

}

/// <summary>

/// israso surikiuotus dalyvius i textini faila

/// </summary>

/// <param name="participants"></param> /// <param name="file"></param>

static void PrintSortedParticipants(PlayersContainer participants, string file)

{

using (StreamWriter writer = new StreamWriter(@file))

{

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

{

writer.WriteLine("Dalyvis Nr. {0} yra {1} {2}", (i + 1), participants.GetPlayer(i).surname, participants.GetPlayer(i).name); }

}

}

}

}

***3.3. Pradiniai duomenys ir rezultatai***

Pradiniai duomenys:

1. 1

Data

LOL;Kevin;Yarnell;TeamSoloMid;Top;Gangplank;13;18;8;

LOL;Dennis;Johnsen;TeamSoloMid;Jungle;Kha'zix;7;24;9;

LOL;Soren;Bjerg;TeamSoloMid;Mid;Ahri;21;8;7;

LOL;Yiliang;Peng;TeamSoloMid;ADC;Vayne;19;9;6;

LOL;Vincent;Wang;TeamSoloMid;Support;Thresh;2;29;16;

LOL;Mateusz;Szkudlarek;Fnatic;Top;Gnar;5;16;4;

LOL;Lee;Da-yoon;Fnatic;Jungle;Sejuani;4;22;18;

LOL;Fabian;Diepstraten;Fnatic;Mid;Lux;15;17;25;

LOL;Martin;Larsson;Fnatic;ADC;Jhin;22;7;13;

LOL;Bora;Kim;Fnatic;Support;Lulu;7;19;14;

1. 2 DATA

CS;Gabriel;Toledo;SK;11;11;AK;

CS;Fernando;Alvarenga;SK;14;1;M4;

CS;Marcelo;David;SK;18;3;AWP;

CS;Lincoln;Lau;SK;2;10;AK;

CS;Epitacio;Pessoa;SK;1;6;AWP;

CS;Nicholas;Cannella;Liquid;14;14;AK;

CS;Jonathan;Jablonowski;Liquid;20;4;AK;

CS;Spencer;Martin;Liquid;17;2;AWP;

CS;Joshua;Marzano;Liquid;5;5;M4;

CS;Jacob;Winneche;Liquid;12;3;AWP;

1. 3 Data

LOL;Lee;Ho-Seong;SKT1;Top;Kled;14;14;17;

LOL;Kang;Sun-gu;SKT1;Jungle;Shaco;24;16;10;

LOL;Lee;Sang-hyeok;SKT1;Mid;Fizz;15;12;2;

LOL;Bae;Jun-sik;SKT1;ADC;Vayne;14;14;3;

LOL;Lee;Jae-wan;SKT1;Support;Braum;1;33;4;

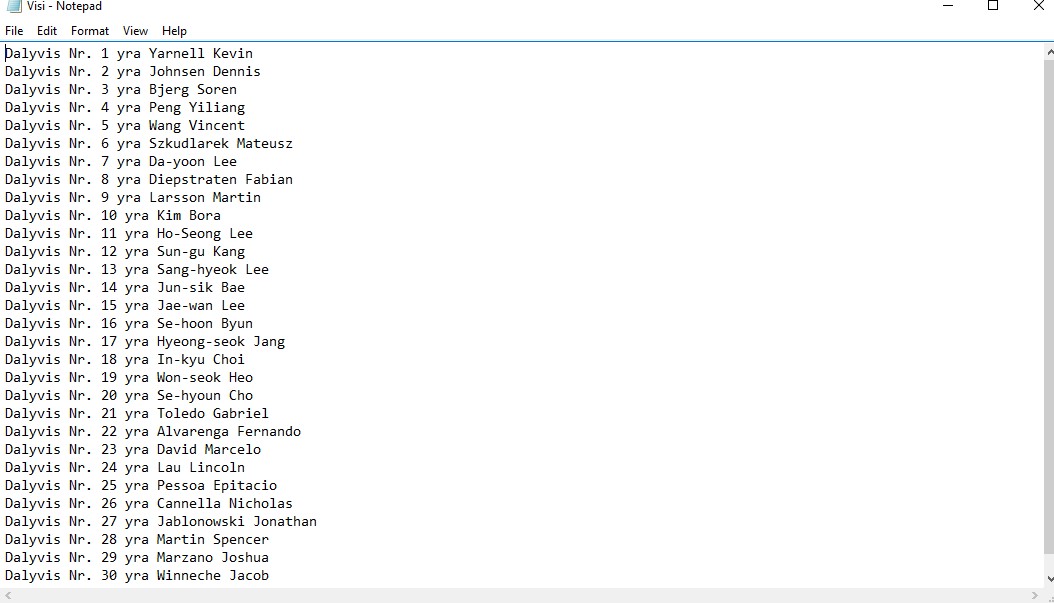
LOL;Byun;Se-hoon;SSW;ADC;Jinx;22;22;5;

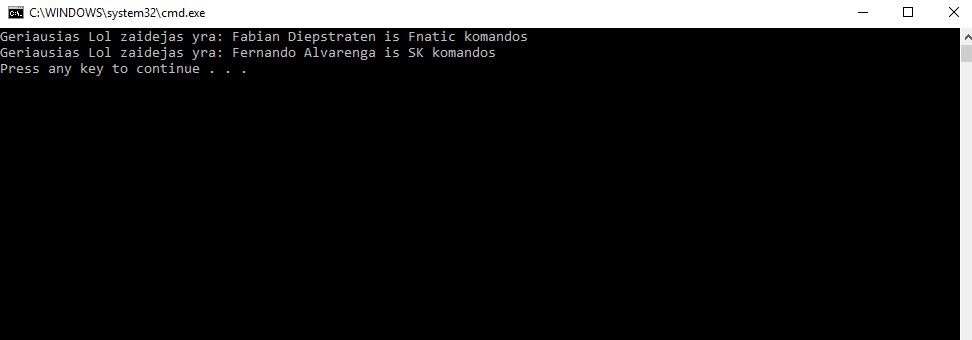
LOL;Jang;Hyeong-seok;SSW;Top;Orianna;10;13;7;

LOL;Choi;In-kyu;SSW;Jungle;Elise;11;11;9;

LOL;Heo;Won-seok;SSW;Mid;Wukong;17;17;11;

LOL;Cho;Se-hyoun;SSW;Support;Blitzcrank;14;14;13;





# 4. Teksto analizė ir redagavimas

***4.1. Darbo užduotis***

U5-19. Pasikartojimai Dviejuose tekstiniuose failuose Knyga1.txt ir Knyga2.txt duotas tekstas sudarytas iš žodžių, atskirtų skyrikliais. Skyriklių aibė žinoma ir abejuose failuose yra ta pati. Analizuojant tekstus, didžiosios ir mažosios raidės nesvarbios. Raskite, spausdinkite faile Analizė.txt ir išveskite ekrane teksto analizės rezultatus:

• ilgiausių žodžių, kurie yra tik faile Knyga1.txt, bet nėra faile Knyga2.txt, sąrašą (ne daugiau nei 10 žodžių) ir jų pasikartojimo skaičių;

• ilgiausią teksto fragmentą, sudarytą iš žodžių ir juos skiriančių skyriklių, kuris yra abejuose failuose ir jo eilutės numerius pirmame ir antrame faile; Spausdinkite faile ManoKnyga.txt apjungtą tekstą, sudarytą pagal tokias taisykles:

• kopijuojamas pirmojo failo tekstas tol, kol sutinkamas pirmasis antrojo failo žodis arba pasiekiama failo pabaiga;

• kopijuojamas antrojo failo tekstas tol, kol sutinkamas pirmasis nenukopijuotas pirmojo failo žodis arba pasiekiama failo pabaiga;

• kartojama tol, kol pasiekiama abiejų failų pabaiga.

***4.2. Programos tekstas***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace L5Tekstas

{

class Program

{

const string file1 = @"..\..\Knyga1.txt";

const string file2 = @"..\..\Knyga2.txt";

const string analysis = @"..\..\Analize.txt";

const string results = @"..\..\ManoKnyga.txt";

const int maxNumberOfWords = 95829;

static void Main(string[] args)

{

char[] punctuation = { '.', ',', ' ', ';', '(', ')', '-', '?', '!', '"', '\n', ':', '/', '}', '{', '[', ']', '&', '\*' };

int count;

string[] words1 = GetWordsFromText(file1, punctuation);

string[] words2 = GetWordsFromText(file2, punctuation);

words1 = words1.OrderBy(aux => aux.Length).ToArray();

string[] filteredWords = FilteredWordArray(words1, words2, out count);

filteredWords = filteredWords.Where(aux => !string.IsNullOrEmpty(aux)).ToArray();

string[] longestWords = LongestWordArray(filteredWords);

longestWords = longestWords.Where(aux => !string.IsNullOrEmpty(aux)).ToArray();

int[] repeatings = LongestWordRepeatings(longestWords, filteredWords);

string[] fragmentArray = FragmentArray(file1, file2);

WriteAnalysis(analysis, longestWords, repeatings, fragmentArray);

string[] fullText = MyBook(file1, file2);

fullText = fullText.Where(aux => !string.IsNullOrEmpty(aux)).ToArray();

WriteMyBookIntoFile(results, fullText);

}

/// <summary>

/// Israso visus zodzius i zodziu masyva is teksto

/// </summary>

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

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

/// <returns>grazina zodziu be skirikliu masyva</returns>

static string[] GetWordsFromText(string file, char[] punctuation)

{

string lines = File.ReadAllText(file, Encoding.GetEncoding(1257));

string[] words = lines.Split(punctuation, StringSplitOptions.RemoveEmptyEntries);

return words;

}

/// <summary>

/// Isfiltruoja zodzius (palieka tik tuos zodzius kurie yra pirmame faile bet nera antrame)

/// </summary>

/// <param name="words1"></param>

/// <param name="words2"></param>

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

/// <returns>grazina filtruotu zodziu masyva</returns>

static string[] FilteredWordArray(string[] words1, string[] words2, out int count)

{

string[] filteredWords = new string[words1.Length];

count = 0;

for (int i = 0; i < words1.Length; i++)

{

if (!words2.Contains(words1[i]))

{

if (!filteredWords.Contains(words1[i]))

{

filteredWords[count] = words1[i];

count++;

}

}

}

return filteredWords;

}

/// <summary>

/// Sudaro ilgiausiu zodziu masyva

/// </summary>

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

/// <returns>grazina ilgiausiu zodziu masyva</returns>

static string[] LongestWordArray(string[] filteredWords)

{

string[] longestWords = new string[filteredWords.Length];

int count = 0;

for (int i = filteredWords.Length - 1; i >= filteredWords.Length - 10; i--)

{

longestWords[count] = filteredWords[i];

count++;

}

return longestWords;

}

/// <summary>

/// Suskaiciuoja kiek kartu karojasi kiekvienas ilgiausias zodis

/// </summary>

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

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

/// <returns> grazina ilgiausiu zodziu pasikartojimu masyva</returns>

static int[] LongestWordRepeatings(string[] longestWords, string[] filteredWords)

{

int[] repeatings = new int[longestWords.Length];

for (int i = 0; i < longestWords.Length; i++)

{

if (filteredWords.Contains(longestWords[i]))

{

repeatings[i]++;

}

else

{

repeatings[i] = 0;

}

}

return repeatings;

}

/// <summary>

/// suraso tarpinius rezultatus i analizes faila "Analize.txt"

/// </summary>

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

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

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

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

static void WriteAnalysis(string file, string[] longestWords, int[] repeatings, string[] fragmentArray)

{

using (StreamWriter writer = new StreamWriter(file))

{

writer.Write("Ilgiausi žodžiai mažėjimo tvarka: ");

for (int i = 0; i < longestWords.Length; i++)

{

writer.WriteLine("Zodis nr.{0} \"{1}\" kartojasi {2} kartu.", i + 1, longestWords[i], repeatings[i]);

}

for (int i = 0; i < fragmentArray.Length; i++)

{

writer.Write("{0} ", fragmentArray[i]);

}

}

}

/// <summary>

/// Israso ilgiausia fragmenta kuris yra abiejuose tekstiniuose failuose

/// </summary>

/// <param name="file1"></param>

/// <param name="file2"></param>

/// <returns>grazina fragmentu masyva</returns>

static string[] FragmentArray(string file1, string file2)

{

string text1 = File.ReadAllText(file1, Encoding.GetEncoding(1257));

string text2 = File.ReadAllText(file2, Encoding.GetEncoding(1257));

string[] splitText1 = text1.Split(' ');

string[] splitText2 = text2.Split(' ');

string[] fragmentArray = new string[splitText1.Length];

int count = 0;

int[] index = new int[fragmentArray.Length];

foreach (string fragment in splitText1)

{

if (splitText2.Contains(fragment))

{

fragmentArray[count] = fragment;

count++;

}

}

for (int i = 0; i < fragmentArray.Length; i++)

{

index[i] = Array.IndexOf(splitText2, fragmentArray[i]);

}

//int Count = 0;

//int str = Array.IndexOf(fragmentArray, fragmentArray[1]);

//foreach(int indx in index)

//{

// if(indx == str)

// {

// }

//}

return fragmentArray;

}

/// <summary>

/// tikrina ar fragmentai eina nuosekliai

/// </summary>

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

/// <param name="file2"></param>

/// <returns></returns>

static bool Follows(string[] fullText, string file2)

{

string text = File.ReadAllText(file2);

string[] textFragments = text.Split(' ');

bool follows = false;

int indexFr = 0;

for(int i = 0; i < textFragments.Length; i++)

{

if ((textFragments[i] == fullText[indexFr]) && (fullText[indexFr] == null))

{

indexFr++;

follows = true;

}

else if(i == textFragments.Length -1)

{

if(follows == true)

{

follows = true;

return follows;

}

else if(follows == false)

{

follows = false;

return follows;

}

}

}

return follows;

}

/// <summary>

/// sudaro viso teksto isskirstyto ir skaityto pagal reikalavimus masyva

/// </summary>

/// <param name="file1"></param>

/// <param name="file2"></param>

/// <returns>grazina viso teksto sudaryto pagal reikalavimus masyva</returns>

static string[] MyBook(string file1, string file2)

{

string txt1 = File.ReadAllText(file1);

string txt2 = File.ReadAllText(file2);

string[] text1 = txt1.Split(' ');

string[] text2 = txt2.Split(' ');

string[] fullText = new string[text1.Length + text2.Length];

bool text1End = false;

bool text2End = false;

bool contains = false;

int count = 0;

int index1 = 0;

int index2 = 0;

while ((text2End != true) || (text1End != true))

{

Console.WriteLine("55");

while (contains == false)

{

for (int i = index1; i < text1.Length; i++)

{

if ((!text2.Contains(text1[i])) && (text1End = false))

{

fullText[count] = text1[i];

index1 = i;

count++;

}

else if (text2.Contains(text1[i]))

{

index1 = i;

}

else if (text1[i] == null)

{

text1End = true;

contains = false;

}

}

}

contains = false;

while (contains == false)

{

for (int i = index2; i < text2.Length; i++)

{

if ((!text1.Contains(text2[i])) && (text2End = false))

{

fullText[count] = text2[i];

index1 = i;

count++;

}

else if (text1.Contains(text2[i]))

{

contains = true;

index1 = i;

}

else if (text2[i] == null)

{

text2End = true;

}

}

}

}

if (text1End == true)

{

while (text2End != true)

{

if (text2[index2] != null)

{

fullText[count] = text2[index2];

index2++;

count++;

}

else if (text2[index2] == null)

{

text2End = true;

}

}

}

if (text2End == true)

{

while (text1End != true)

{

if (text1[index1] != null)

{

fullText[count] = text1[index1];

index1++;

count++;

}

else if (text2[index1] == null)

{

text1End = true;

}

}

}

return fullText;

}

/// <summary>

/// suraso duomenis is fullText masyvo i faila "ManoKnyga.txt"

/// </summary>

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

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

static void WriteMyBookIntoFile(string results, string[] fullText)

{

using (StreamWriter writer = new StreamWriter(results))

{

writer.WriteLine("Tekstas sudarytas is abieju tekstu, pagal reikalavimus: \n");

for (int i = 0; i < fullText.Length; i++)

{

writer.Write("{0} ", fullText[i]);

}

}

}

}

}

***4.3. Pradiniai duomenys ir rezultatai***

Failas nr 1: Reikia teksto žodžius sulygiuoti, kad kiekvienos eilutės kiekvienas žodis prasidėtų fiksuotoje toje pačioje pozicijoje.

Galima įterpti tik minimalų būtiną tarpų skaičių.

Galima šalinti kelis iš eilės einančius vienodus skyriklius, paliekant tik vieną jų atstovą.

Įterpimo ir šalinimo taisykles taikome, siekdami gauti lygiuotą minimalų tekstą.

Šalinimo taisyklės netaikome, jei nėra poreikio.

Pradinio teksto eilutės ilgis neviršija 80 simbolių.

If you really wanted to do it this way, something like this would be clearer

Failas nr 2: Initialize If queue to null, which is really just a compiler trick that says "I'll figure out my own uninitialized variables, thank you very much".

It's a useful trick, but I don't like it in this case - you have too many if branches to easily check that you're doing it properly.

If you really wanted to do it this way, something like this would be clearer

Rezultatu failas nr 1:

Ilgiausi žodžiai mažėjimo tvarka:

Zodis nr.1 "skyriklius" kartojasi 1 kartu.

Zodis nr.2 "pozicijoje" kartojasi 1 kartu.

Zodis nr.3 "fiksuotoje" kartojasi 1 kartu.

Zodis nr.4 "kiekvienas" kartojasi 1 kartu.

Zodis nr.5 "kiekvienos" kartojasi 1 kartu.

Zodis nr.6 "sulygiuoti" kartojasi 1 kartu.

Zodis nr.7 "neviršija" kartojasi 1 kartu.

Zodis nr.8 "netaikome" kartojasi 1 kartu.

Zodis nr.9 "taisyklės" kartojasi 1 kartu.

Zodis nr.10 "taisykles" kartojasi 1 kartu.

you really wanted to do it this way, something like this would be clearer

# 5. Sudėtingesnis konteineris

***5.1. Darbo užduotis***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace U4WCG19

{

class Program

{

const int maxNumberOfPlayers = 90;

const int numberOfBranches = 3;

static void Main(string[] args)

{

int indexB;

int indexE;

int indexB2;

int indexE2;

Branch[] branches = Branches();

Read(branches);

BestLoLPlayer(branches, out indexB, out indexE);

Console.WriteLine("Geriausias LOL žaidėjas yra: {0} {1} iš {2} komandos", branches[indexB].AllPlayers.GetPlayer(indexE).name,

branches[indexB].AllPlayers.GetPlayer(indexE).surname, branches[indexB].AllPlayers.GetPlayer(indexE).team);

BestCSGOPlayer(branches, out indexB2, out indexE2);

Console.WriteLine("Geriausias CSGO žaidėjas yra: {0} {1} iš {2} komandos.", branches[indexB2].AllPlayers.GetPlayer(indexE2).name,

branches[indexB2].AllPlayers.GetPlayer(indexE2).surname, branches[indexB2].AllPlayers.GetPlayer(indexE2).team);

FindAndPrintTeams(branches, @"..\..\Komandos.csv");

PlayersContainer allplayers = allPlayers(branches);

allplayers = Sort(allplayers);

PrintPlayersToFile(allplayers, @"..\..\Dalyviai.csv");

//Console.WriteLine(branches[1].AllPlayers.Count);

//int index1;

//int index2;

//int index3;

//int index4;

//string[] teams = new string[maxNumberOfPlayers];

//PlayersContainer Lolplayers = new PlayersContainer(maxNumberOfPlayers);

//int teamCount = GetTeams(branches, teams);

//PrintTeams(teams, teamCount, @"..\..\Komandos.csv");

//BestLOLPlayer(branches, out index1, out index2);

//LOLPlayers player = branches[index1].LOLplayers.GetPlayer(index2) as LOLPlayers;

//Console.WriteLine("Geriausias Lol zaidejas yra: {0} {1} is {2} komandos", player.name, player.surname, player.team);

//BestCSPlayer(branches, out index3, out index4);

//CSGOPlayers players = branches[index3].CSGOplayers.GetPlayer(index4) as CSGOPlayers;

//Console.WriteLine("Geriausias Lol zaidejas yra: {0} {1} is {2} komandos", players.name, players.surname, players.team);

//PlayersContainer participants = AllParticipants(branches);

//PrintSortedParticipants(participants, @"..\..\Visi.csv");

}

/// <summary>

/// padaro visus branches

/// </summary>

/// <returns></returns>

private static Branch[] Branches()

{

Branch[] branches = new Branch[numberOfBranches];

for(int i = 0; i < numberOfBranches; i++)

{

branches[i] = new Branch(i + 1);

}

return branches;

}

/// <summary>

/// priskiria branches tourui

/// </summary>

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

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

/// <returns></returns>

private static Branch GetBranchByTour(Branch[] branches, int tour)

{

for (int i = 0; i < numberOfBranches; i++)

{

if (branches[i].tour == tour)

{

return branches[i];

}

}

return null;

}

/// <summary>

/// perskaito zaideju duomenis is failo

/// </summary>

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

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

private static void ReadPlayerData(string file, Branch[] branches)

{

int tour = 0;

using (StreamReader reader = new StreamReader(@file))

{

string line = null;

line = reader.ReadLine();

if (line != null)

{

tour = int.Parse(line);

}

Branch branch = GetBranchByTour(branches, tour);

line = reader.ReadLine();

string date = line;

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

{

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

string game = values[0];

if (game == "LOL")

{

//string position = values[4];

//string champion = values[5];

//int kills = int.Parse(values[6]);

//int deaths = int.Parse(values[7]);

//int assists = int.Parse(values[8]);

LOLPlayers LOLPlayer = new LOLPlayers();

LOLPlayer.SetData(line);

branch.AllPlayers.AddPlayer(LOLPlayer);

//Console.WriteLine(LOLPlayer.game);

}

else if (game == "CS")

{

//int kills = int.Parse(values[4]);

//int deaths = int.Parse(values[5]);

//string favouriteWeapon = values[6];

CSGOPlayers CSGOPlayer = new CSGOPlayers();

CSGOPlayer.SetData(line);

branch.AllPlayers.AddPlayer(CSGOPlayer);

//Console.WriteLine(CSGOPlayer.game);

}

}

}

}

/// <summary>

/// perskaito visus failus

/// </summary>

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

private static void Read(Branch[] branches)

{

string[] filePath = Directory.GetFiles(Directory.GetCurrentDirectory(), "\*.csv");

foreach (string file in filePath)

{

ReadPlayerData(file, branches);

}

}

/// <summary>

/// randa geriausia lolzaideja

/// </summary>

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

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

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

private static void BestLoLPlayer(Branch[] branches, out int indexB, out int indexE)

{

int maxKDA = 0;

indexB = 0;

indexE = 0;

for (int i = 0; i < branches.Length; i++)

{

for (int j = 0; j < branches[i].AllPlayers.Count; j++)

{

LOLPlayers players = branches[i].AllPlayers.GetPlayer(j) as LOLPlayers;

if (branches[i].AllPlayers.GetPlayer(j).game == "LOL")

{

if(((players.kills + players.assists)/players.deaths) > maxKDA)

{

maxKDA = ((players.kills + players.assists) / players.deaths);

indexB = i;

indexE = j;

}

}

}

}

}

/// <summary>

/// randa geriausia csgo zaideja

/// </summary>

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

/// <param name="indexB2"></param>

/// <param name="indexE2"></param>

private static void BestCSGOPlayer(Branch[] branches, out int indexB2, out int indexE2)

{

int maxKDA = 0;

indexB2 = 0;

indexE2 = 0;

for (int i = 0; i < branches.Length; i++)

{

for (int j = 0; j < branches[i].AllPlayers.Count; j++)

{

CSGOPlayers players = branches[i].AllPlayers.GetPlayer(j) as CSGOPlayers;

if (branches[i].AllPlayers.GetPlayer(j).game == "CS")

{

if (players.kills / players.deaths > maxKDA)

{

maxKDA = players.kills / players.deaths;

indexB2 = i;

indexE2 = j;

}

}

}

}

}

/// <summary>

/// printina komandas i faila

/// </summary>

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

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

private static void FindAndPrintTeams(Branch[] branches, string file)

{

using (StreamWriter writer = new StreamWriter(@file))

{

writer.WriteLine("Komandų sąrašas: ");

for (int i = 0; i < branches.Length; i++)

{

for (int j = 1; j < branches[i].AllPlayers.Count; j++)

{

if (branches[i].AllPlayers.GetPlayer(j).team != branches[i].AllPlayers.GetPlayer(j - 1).team)

{

writer.WriteLine("{0}", branches[i].AllPlayers.GetPlayer(j-1).team);

writer.WriteLine("{0}", branches[i].AllPlayers.GetPlayer(j).team);

}

}

}

}

}

//private static List<object> SortObjectList(Branch[] branches)

//{

// List<object> allPlayers = new List<object>();

// int count = 0;

// for(int i = 0; i < branches.Length; i++)

// {

// for(int j = 0; j < branches[i].AllPlayers.Count; j++)

// {

// Players players = branches[i].AllPlayers.GetPlayer(j) as Players;

// allPlayers[count] = players;

// }

// }

//}

/// <summary>

/// visu zaideju konteineris

/// </summary>

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

/// <returns></returns>

private static PlayersContainer allPlayers(Branch[] branches)

{

PlayersContainer allplayas = new PlayersContainer(maxNumberOfPlayers);

for (int i = 0; i < branches.Length; i++)

{

for (int j = 0; j < branches[i].AllPlayers.Count; j++)

{

allplayas.AddPlayer(branches[i].AllPlayers.GetPlayer(j));

}

}

return allplayas;

}

/// <summary>

/// isrikiuoja zaidejus pagal pavarde

/// </summary>

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

/// <returns></returns>

static PlayersContainer Sort(PlayersContainer allplayers)

{

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

{

for (int j = 1; j < allplayers.Count; j++)

{

if (allplayers.GetPlayer(j - 1) <= allplayers.GetPlayer(j))

{

Players a = allplayers.GetPlayer(j - 1);

allplayers.SetPlayer(j - 1, allplayers.GetPlayer(j));

allplayers.SetPlayer(j, a);

}

}

}

return allplayers;

}

/// <summary>

/// israso surikiuotus zaidejus i faila

/// </summary>

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

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

private static void PrintPlayersToFile(PlayersContainer allplayers, string file)

{

using (StreamWriter writer = new StreamWriter(@file))

{

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

{

writer.WriteLine("{0} {1} iš komandos {2}", allplayers.GetPlayer(i).surname, allplayers.GetPlayer(i).name, allplayers.GetPlayer(i).team);

}

}

}

## 5.2. Programos tekstas

U6\_19. WCG turnyras. Turite trijų turnyro ratų duomenis. Keičiasi duomenų formatas. Pirmoje eilutėje rato numeris, antroje – data. Toliau pateikta informacija apie to rato rezultatus. Turnyre varžosi kelių skirtingų žaidimų („League of Legends“ ir „Counter Strike“) žaidėjai. Sukurkite klasę „Žaidėjas“ (laukai - P175B118 Objektinis programavimas I 18 vardas, pavardė, komanda), kurią paveldės klasės “LOLŽaidėjas” (papildomi laukai - pozicija, čempionas, nužudymai(K), mirtys(D), dalyvavimai nužudymuose(A)) ir “CSŽaidėjas” (papildomi laukai - nužudymai(K), mirtys(D), mėgstamiausias ginklas).

• Raskite LOL ir CS žaidėją, pademonstravusį geriausią bendrą (per visus tris ratus) asmeninį rezultatą. LOL žaidėjų palyginimui naudokite vadinamąjį KDA santykį (nužudymai + dalyvavimai nužudymuose)/mirtys t.y. (K+A)/D, o CS žaidėjams palyginti - KD santykį (K/D). Ekrane atspausdinkite jų vardus, pavardes ir komandos pavadinimą.

• Sudarykite visų turnyre dalyvaujančių komandų sąrašą, ir atspausdinkite faile „Komandos.csv“.

• Sudarykite visų turnyro dalyvių sąrašą, išrikiuokite juos pagal pavardę ir vardą, ir įrašykite į failą „Visi.csv“.

***5.3. Pradiniai duomenys ir rezultatai***

Rezultatai:

Dalyviai.csv

Won-seok Heo iš komandos SSW

Winneche Jacob iš komandos Liquid

Wang Vincent iš komandos TeamSoloMid

Toledo Gabriel iš komandos SK

Szkudlarek Mateusz iš komandos Fnatic

Sun-gu Kang iš komandos SKT1

Se-hoon Byun iš komandos SSW

Se-hyoun Cho iš komandos SSW

Sang-hyeok Lee iš komandos SKT1

Pessoa Epitacio iš komandos SK

Peng Yiliang iš komandos TeamSoloMid

Marzano Joshua iš komandos Liquid

Martin Spencer iš komandos Liquid

Lau Lincoln iš komandos SK

Larsson Martin iš komandos Fnatic

Kim Bora iš komandos Fnatic

Jun-sik Bae iš komandos SKT1

Johnsen Dennis iš komandos TeamSoloMid

Jae-wan Lee iš komandos SKT1

Jablonowski Jonathan iš komandos Liquid

In-kyu Choi iš komandos SSW

Yarnell Kevin iš komandos TeamSoloMid

Ho-Seong Lee iš komandos SKT1

Hyeong-seok Jang iš komandos SSW

Diepstraten Fabian iš komandos Fnatic

David Marcelo iš komandos SK

Da-yoon Lee iš komandos Fnatic

Cannella Nicholas iš komandos Liquid

Bjerg Soren iš komandos TeamSoloMid

Alvarenga Fernando iš komandos SK

Komandos.csv

Komandų sąrašas:

TeamSoloMid

Fnatic

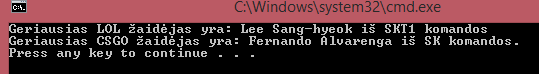
SK

Liquid

SKT1

SSW

Išvesties Langas:



Pradiniai duomenys:

Skiriklis – ‚;‘, Game;Name;Surname;Team;Position;Kills;Assists;Deaths;

Tour1.csv

1

Data

LOL;Kevin;Yarnell;TeamSoloMid;Top;Gangplank;13;18;8;

LOL;Dennis;Johnsen;TeamSoloMid;Jungle;Kha'zix;7;24;9;

LOL;Soren;Bjerg;TeamSoloMid;Mid;Ahri;21;8;7;

LOL;Yiliang;Peng;TeamSoloMid;ADC;Vayne;19;9;6;

LOL;Vincent;Wang;TeamSoloMid;Support;Thresh;2;29;16;

LOL;Mateusz;Szkudlarek;Fnatic;Top;Gnar;5;16;4;

LOL;Lee;Da-yoon;Fnatic;Jungle;Sejuani;4;22;18;

LOL;Fabian;Diepstraten;Fnatic;Mid;Lux;15;17;25;

LOL;Martin;Larsson;Fnatic;ADC;Jhin;22;7;13;

LOL;Bora;Kim;Fnatic;Support;Lulu;7;19;14;

Tour2.csv

2

DATA

CS;Gabriel;Toledo;SK;11;11;AK;

CS;Fernando;Alvarenga;SK;14;1;M4;

CS;Marcelo;David;SK;18;3;AWP;

CS;Lincoln;Lau;SK;2;10;AK;

CS;Epitacio;Pessoa;SK;1;6;AWP;

CS;Nicholas;Cannella;Liquid;14;14;AK;

CS;Jonathan;Jablonowski;Liquid;20;4;AK;

CS;Spencer;Martin;Liquid;17;2;AWP;

CS;Joshua;Marzano;Liquid;5;5;M4;

CS;Jacob;Winneche;Liquid;12;3;AWP;

Tour3.csv

3

Data

LOL;Lee;Ho-Seong;SKT1;Top;Kled;14;14;17;

LOL;Kang;Sun-gu;SKT1;Jungle;Shaco;24;16;10;

LOL;Lee;Sang-hyeok;SKT1;Mid;Fizz;15;12;2;

LOL;Bae;Jun-sik;SKT1;ADC;Vayne;14;14;3;

LOL;Lee;Jae-wan;SKT1;Support;Braum;1;33;4;

LOL;Byun;Se-hoon;SSW;ADC;Jinx;22;22;5;

LOL;Jang;Hyeong-seok;SSW;Top;Orianna;10;13;7;

LOL;Choi;In-kyu;SSW;Jungle;Elise;11;11;9;

LOL;Heo;Won-seok;SSW;Mid;Wukong;17;17;11;

LOL;Cho;Se-hyoun;SSW;Support;Blitzcrank;14;14;13;