**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INformatikos fakultetas**

****

**Lygiagretusis programavimas**

SEMESTRO DARBŲ ATASKAITA

**Vadovas**

Lek. Dominykas Barisas

Lek. Mindaugas Vasiljevas

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Studentas**

Lukas Gužauskas,

IFF-5/1 grupė

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**KAUNAS, 2017-2018**

**Turinys:**

1. L1. Procesai – gijos 3

L1a. Gijos – objektai 3

L1b. Gijos – klonai: C++ & OpenMP 9

L1c. Gijos – klonai: C & CUDA 16

2. L2. Bendra atmintis 24

L2a. Java(C++, C#) 24

L2b. C++ & OpenMP 38

3. L3. Paskirstyta atmintis 48

L3a. Any2One 48

L3b. One2One 65

4. L4. Lygiagretusis programavimas CUDA 82

L4a. CUDA gijos 82

L4b. Thrust CPU ir GPU vektoriai 90

1. L1. Procesai – gijos
   * L1a. Gijos – objektai

**Užduotis:**

Duomenų faile Grupe\_PavardeV\_L1a\_dat.txt (Grupe – jūsų grupė, PavardeV – jūsų pavardė, v.) yra jūsų pasirinkto tipo struktūrų (bent trys skirtingų tipų laukai: *string*, *int*, *double*) penki duomenų rinkiniai.

Įvesti šiuos duomenis į penkis masyvus *P1(k1), P2(k2), ..., P5(k5)* (*4 < k1, ..., k5 < 10*). Spausdinti masyvų duomenis lentelėmis į failą Grupe\_PavardeV\_L1a\_rez.txt.

Sukurti ir įvykdyti gijas ***gija\_1***, ***gija\_2***, …, ***gija\_5***. Kiekvienos gijos kūrimo metu jai perduodami atitinkamo masyvo (*P1, P2, ..., P5*) duomenys.

Vykdymo metu gijos savo masyvų elementus rašo į visoms gijoms bendro masyvo *P* pabaigą neatsižvelgdamos į kitų gijų darbą. Kiekvienam gijos masyvo elementui į masyvą *P* įrašyti gijos pavadinimą, masyvo elemento numerį ir laukus. Baigus gijų darbą į failą Grupe\_PavardeV\_L1a\_rez.txt spausdinti masyvą *P* lentele.

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L1a.cpp : Lukas Gužauskas IFF-5/1 gr.

//

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <fstream>

#include <sstream>

#include <iterator>

#include <vector>

#include <thread>

#include <mutex>

#include <cstdio>

namespace A{

int count; }

using namespace std;

using namespace A;

/\*\* Klasė studento duomenis saugoti

@class Student\*/

class Student

{

public: string name; // vardas ir pavardė

public: int course; // kursas

public: double average; // pažymių vidurkis

public: Student() {}

/\* Studento duomenys

@param nam - nauja vardo ir pavardės reikšmė

@param cour - nauja kurso reikšmė

@param ave - nauja vidurkio reikšmė\*/

public: Student(string nam, int cour, double ave)

{

name = nam;

course = cour;

average = ave;

}

};

/\*\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty

{

public: string tilte; // fakulteto pavadinimas

int stucout; // studentu skaicius

vector<Student> Students;

/\* Grąžina nurodyto indekso studentų objektą.

@param i - studentų indeksas\*/

public: Student getStudent(int i) {

return Students[i];

}

/\* Dėti į studentų objektų masyvą naują studentą ir masyvo dydį.

@param i - indeksas

@param s - studentų objektas\*/

void setStudent(int i, Student s) {

Students[i] = s;

}

/\* Dėti į studentų objektų masyvą naują studentą.

@param s - studentų objektas\*/

void setStudent(Student s) {

Students.push\_back(s);

}

/\* Grąžina visą studentų objektą.\*/

vector<Student> getStudent() {

return Students;

}

void setStudent(vector<Student> s) {

Students = s;

}

/\* Fakulteto duomenys

@param p - nauja fauklteto pavadinimo reikšmė

@param sk - nauja studentu skaiciaus reikšmė

@param s - nauja objektu reikšmė\*/

Faculty(string p, int sk, vector<Student> s) {

tilte = p;

stucout = sk;

Students = s;

}

/\* Fakulteto numatytas duomenys\*/

Faculty() {

tilte = "";

stucout = 0;

Students = vector<Student>();

}

};

FILE \* File;

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(ifstream & File, Faculty & P)

{

string faculty, name, line;

int studCout, course;

double average;

File >> P.tilte >> P.stucout;

for (int i = 0; i < P.stucout; i++) {

File >> name >> course >> average;

Student s = Student(name, course, average);

P.setStudent(s);

}

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void WriteData(FILE \* File, Faculty P)

{

fprintf(File, " \*\*\* %s \*\*\* \n", P.tilte.c\_str());

fprintf(File, " Vardas kursas vid\n");

for (int i = 0; i < P.stucout; i++) {

Student S = P.getStudent(i);

fprintf(File, " %i) %s \t %i \t %g \n", i + 1, S.name.c\_str(), S.course, S.average);

//cout << S.name << " " << S.course << endl;

}

}

/\* Spausdina rez į failą

@param File - failo vardas

@param P - masyvo vardas\*/

void RezPrint(FILE \*File, string \*P)

{

if (File == NULL)

fprintf(stdout, "Nera failo");

else

for (int i = 0; i < sizeof(\*P); i++)

fprintf(File, P[i].c\_str());

return;

}

int count = 0;

void Execute(Faculty P, string \*L, string j);

int cnt;

mutex theLock;

int main()

{

string filename = "IFF-5-1\_GuzauskasL\_L1a\_dat.txt";

string printname = "IFF-5-1\_GuzauskasL\_L1a\_rez.txt";

Faculty L1 = Faculty();

Faculty L2 = Faculty();

Faculty L3 = Faculty();

Faculty L4 = Faculty();

Faculty L5 = Faculty();

ifstream infile;

infile.open(filename);

ReadData(infile, L1);

ReadData(infile, L2);

ReadData(infile, L3);

ReadData(infile, L4);

ReadData(infile, L5);

ReadData(infile, L5);

infile.close();

File = freopen(printname.c\_str(), "w", stderr);

WriteData(File, L1);

WriteData(File, L2);

WriteData(File, L3);

WriteData(File, L4);

WriteData(File, L5);

fprintf(File,"-----------------------------------------\n");

cout << "Pradiniai duomenys" << endl;

string \*H;

H = new string[50];

cnt = 0;

thread g1(Execute, L1, H, "gija\_1 ");

thread g2(Execute, L2, H, "gija\_2 ");

thread g3(Execute, L3, H, "gija\_3 ");

thread g4(Execute, L4, H, "gija\_4 ");

thread g5(Execute, L5, H, "gija\_5 ");

g1.join();

g2.join();

g3.join();

g4.join();

g5.join();

RezPrint(File, H);

fprintf(File, "\nPabaiga");

fclose(File);

return 0;

}

void Execute(Faculty P, string \*L, string j)

{

for (int i = 0; i < P.stucout; i++){

lock\_guard<mutex> lock(theLock);

Student S = P.getStudent(i);

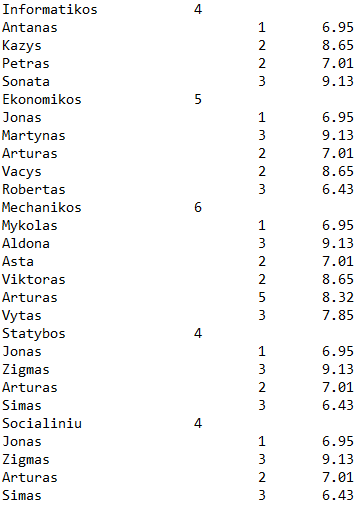
L[cnt++] = j + to\_string(i + 1) + " " + S.name + " \t" + to\_string(S.course) + "\t" + to\_string(S.average) + "\n";

this\_thread::sleep\_for(chrono::milliseconds(0));

}

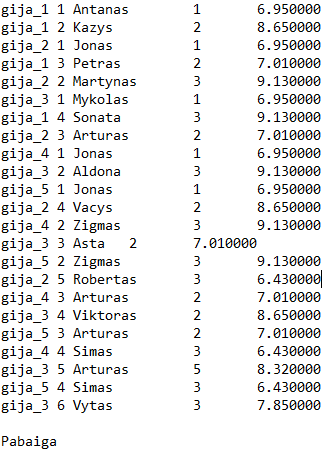
}

**Duomenų failas:**



**Rezultatų failas:**





* + L1b. Gijos – klonai: C++ & OpenMP

**Užduotis:**

Duomenų faile Grupe\_PavardeV\_L1b\_dat.txt (Grupe – jūsų grupė, PavardeV – jūsų pavardė, v.) yra jūsų pasirinkto tipo struktūrų (trys skirtingų tipų laukai: *string*, *int*, *double*) duomenų rinkinys. *string* tipo lauko reikšmės surikiuotos pagal abėcėlę, *int*, *double* - didėjimo kryptimi.

Įvesti šiuos duomenis į tris masyvus *S(n), I(n),* ir *D(n)* (*10 < n < 20*). Kiekvieno duomenų lauko reikšmės įvedamos į atskirą masyvą. Spausdinti masyvų duomenis vienoje lentelėje į failą Grupe\_PavardeV\_L1b\_rez.txt. Kiekvieną masyvą spausdinti atskirame lentelės stulpelyje.

Sukurti ir įvykdyti ***n gijų***.

Gijų vykdymo metu kuriamas bendras struktūrų (*int, string*, *int*, *double*) masyvas *P* tokiu būdu: *i*-oji gija (*i = 0, 1, 2, ...,* *n-1*)į masyvo P pabaigą (į naujo elemento laukus) rašo *i*, *Si*, *I i*, *Di* reikšmes.Gijos dirbaneatsižvelgdamos į kitų gijųdarbą.

Baigus gijų darbą į failą Grupe\_PavardeV\_L1b\_rez.txt spausdinti masyvą *P* lentele.

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L1b.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <fstream>

#include <sstream>

#include <iterator>

#include <vector>

#include <thread>

#include <mutex>

#include <cstdio>

#include <algorithm>

#include <omp.h>

using namespace std;

/\*\* Klasė studento duomenis saugoti

@class Student\*/

class Student

{

public: string name; // vardas ir pavardė

public: string tilte; // fakulteto pavadinimas

public: int course; // kursas

int length; // studentų skaičius

public: double average; // pažymių vidurkis

public: Student() {}

/\* Studento duomenys

@param til - nauja fakulteto reikšmė

@param len - nauja skaičiaus reikšmė\*/

public: Student(string til, int len)

{

tilte = til;

length = len;

}

/\* Studento duomenys

@param nam - nauja vardo ir pavardės reikšmė

@param cour - nauja kurso reikšmė

@param ave - nauja vidurkio reikšmė\*/

public: Student(string nam, int cour, double ave)

{

name = nam;

course = cour;

average = ave;

}

};

/\*\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty

{

public: string tilte; // fakulteto pavadinimas

int stucout; // studentu skaicius

vector<Student> Students;

/\* Grąžina nurodyto indekso studentų objektą.

@param i - studentų indeksas\*/

public: Student getStudent(int i) {

return Students[i];

}

/\* Dėti į studentų objektų masyvą naują studentą

ir masyvo dydį.

@param i - indeksas

@param s - studentų objektas\*/

void setStudent(int i, Student s) {

Students[i] = s;

}

/\* Dėti į studentų objektų masyvą naują studentą.

@param s - studentų objektas\*/

void setStudent(Student s) {

Students.push\_back(s);

}

/\* Grąžina visą studentų objektą.\*/

vector<Student> getStudent() {

return Students;

}

void setStudent(vector<Student> s) {

Students = s;

}

/\* Fakulteto duomenys

@param p - nauja fauklteto pavadinimo reikšmė

@param sk - nauja studentu skaiciaus reikšmė

@param s - nauja objektu reikšmė\*/

Faculty(string p, int sk, vector<Student> s) {

tilte = p;

stucout = sk;

Students = s;

}

/\* Fakulteto numatytas duomenys\*/

Faculty() {

tilte = "";

stucout = 0;

Students = vector<Student>();

}

};

/\*\* Klasė bendras saugoti strukturu duomenis

@class Overall\*/

class Overall

{

public: int i, course;

public: string name;

public: double average;

/\* Bendro numatytas duomenys\*/

Overall() {

i = 0;

name = "";

course = 0;

average = 0.00;

}

/\* Bendro duomenys

@param ii - nauja gijos pavadinimo reikšmė

@param nam - nauja studento vardo reikšmė

@param cour - nauja studijos kurso reikšmė

@param aver - nauja studijos vidurkio reikšmė\*/

Overall(int ii, string nam, int cour, double aver)

{

i = ii; name = nam; course = cour; average = aver;

}

};

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(ifstream & File, Faculty & P)

{

string faculty, name, line;

int studCout, course;

double average;

File >> P.tilte >> P.stucout;

for (int i = 0; i < P.stucout; i++) {

File >> name >> course >> average;

Student s = Student(name, course, average);

P.setStudent(s);

}

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void WriteData(FILE \* File, string \*S, int \*I, double \*D, int length)

{

fprintf(File, " Vardas kursas vid\n");

for (int i = 0; i < length; i++) {

fprintf(File, " %i) %s \t %i \t %g \n", i + 1, S[i].c\_str(), I[i], D[i]);

}

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void RezWriteData(FILE \* File, Overall \* P, int length)

{

for (int i = 0; i < length; i++) {

fprintf(File, " gija\_%i %s \t %i \t %g \n", P[i].i + 1, P[i].name.c\_str(), P[i].course, P[i].average);

}

}

bool sortbyName(Student &a, Student &b) {return (a.name < b.name || (a.name == b.name && a.course < b.course && a.average < b.average));}

void PutDataIntoArray(Faculty F, string \*P, int \*I, double \*D, int length);

void Execute(Overall \* P, string S, int I, double D);

FILE \* File;

int countsk;

mutex theLock;

omp\_lock\_t mylock;

int main()

{

string filename = "IFF-5-1\_GuzauskasL\_L1b\_dat.txt";

string printname = "IFF-5-1\_GuzauskasL\_L1b\_rez.txt";

Faculty P1 = Faculty();

vector<Student> Stud;

ifstream infile;

infile.open(filename);

ReadData(infile, P1);

Stud = P1.getStudent();

sort(Stud.begin(), Stud.end(), sortbyName);

P1.setStudent(Stud);

string S[50];

int I[50];

double D[50];

PutDataIntoArray(P1, S, I, D, P1.stucout);

File = freopen(printname.c\_str(), "w", stderr);

WriteData(File, S, I, D, P1.stucout);

fprintf(File, "-----------------------------------------\n");

int gijuSk = P1.stucout;

Overall P[50];

int gijosNr = 0;

countsk = 0;

omp\_set\_num\_threads(gijuSk);

#pragma omp parallel private (gijosNr)

{

gijosNr = omp\_get\_thread\_num();

Execute(P, S[countsk],I[countsk], D[countsk]);

}

RezWriteData(File, P, P1.stucout);

fprintf(File, "Pabaiga\n");

infile.close();

return 0;

}

void PutDataIntoArray(Faculty F,string \*P, int \*I, double \*D, int length)

{

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

{

Student S = F.getStudent(i);

P[i] = S.name; I[i] = S.course; D[i] = S.average;

}

}

void Execute(Overall \* P, string S, int I, double D)

{

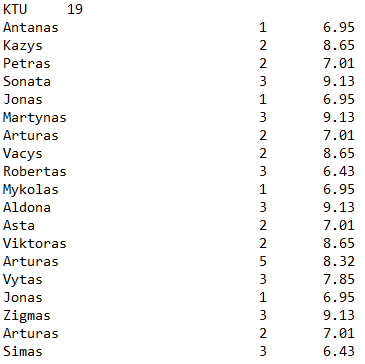
Overall L = Overall(countsk, S, I, D);

P[countsk] = L;

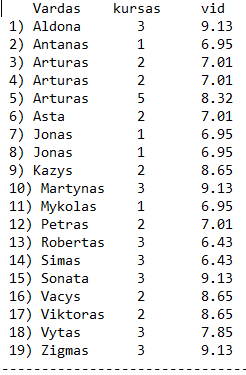
countsk++;

}

**Duomenų failas:**



**Rezultatų failas:**





* + L1c. Gijos – klonai: C & CUDA

**Užduotis:**

Duomenų faile Grupe\_PavardeV\_L1c\_dat.txt (Grupe – jūsų grupė, PavardeV – jūsų pavardė, v.) yra jūsų pasirinkto tipo struktūrų (trys skirtingų tipų laukai: *string*, *int*, *double*) duomenų rinkinys. *string* tipo lauko reikšmės surikiuotos pagal abėcėlę, *int*, *double* - didėjimo kryptimi.

Įvesti šiuos duomenis į tris masyvus *S(n), I(n),* ir *D(n)* (*10 < n < 20*). Kiekvieno duomenų lauko reikšmės įvedamos į atskirą masyvą. Spausdinti masyvų duomenis vienoje lentelėje į failą Grupe\_PavardeV\_L1c\_rez.txt. Kiekvieną masyvą spausdinti atskirame lentelės stulpelyje.

Sukurti ir įvykdyti ***n gijų***.

Gijų vykdymo metu kuriamas bendras struktūrų (*int, string*, *int*, *double*) masyvas *P* tokiu būdu: *i*-oji gija (*i = 0, 1, 2, ...,* *n-1*)į masyvo P pabaigą (į naujo elemento laukus) rašo *i*, *Si*, *I i*, *Di* reikšmes.Gijos dirbaneatsižvelgdamos į kitų gijųdarbą.

Baigus gijų darbą į failą Grupe\_PavardeV\_L1c\_rez.txt spausdinti masyvą *P* lentele.

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L1b.cpp : Lukas Gužauskas IFF-5/1 gr.

// ---------------------------------------------------------------------------------

// Kokia tvarka startuoja gijos?

// Tokia, kokia užrašyti.

// Kokia tvarka vykdomi gijos?

// Tokia, kokia startuoja.

// Kiek iteracijų iš eilės padaro viena gija?

// Visas;

// Kokia tvarka to paties duomenų masyvo elementai surašomi į rezultatų masyvą?

// Tokia, kokia surašyti duomenų masyve.

// Kurioje programoje trumpiausias vienos gijos kodas?

// ??

// Kokiu kompiuteriu vykdėte savo programas?

// Branduoliai skaičiai - 5 ir dažnis: 3.00 GHz, OA apimtis: 6107 MB, OS: Windows 10, NVIDIA plokštės tipas: GeForce GTX 1050.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <cuda.h>

#include <stdio.h>

#include <string>

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

#include <stdio.h>

using namespace std;

const char \* FileName = "IFF-5-1\_GuzauskasL\_L1c\_dat.txt";

const char \* printname = "IFF-5-1\_GuzauskasL\_L1c\_rez.txt";

int countsk;

/\*\* Klasė studento duomenis saugoti

@class Student\*/

class Student

{

public: char name[20]; // vardas ir pavardė

public: int course; // kursas

public: double average; // pažymių vidurkis

/\* Studento duomenys

@param nam - nauja vardo ir pavardės reikšmė

@param cour - nauja kurso reikšmė

@param ave - nauja vidurkio reikšmė\*/

Student(string nam, int cour, double ave)

{

    strcpy(name, nam.c\_str());

    course = cour;

    average = ave;

}

};

/\*\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty

{

public: char tilte[20]; // fakulteto pavadinimas

        int stucout; // studentu skaicius

        vector<Student> Students;

public:

    \_\_device\_\_ \_\_host\_\_ Student getStudent(int i) {

        return Students[i];

    }

    vector<Student> getStudent() {

        return Students;

    }

    void setStudent(int i, Student s) {

        Students[i] = s;

    }

    void setStudent(Student s) {

        Students.push\_back(s);

    }

    void setStudent(vector<Student> s) {

        Students = s;

    }

    /\* Fakulteto duomenys

        @param p - nauja studento vardo reikšmė

        @param sk - nauja studento skaičių reikšmė

        @param s - nauja objetų masyvo reikšmė\*/

    Faculty(string p, int sk, vector<Student> s) {

        strcpy(tilte, p.c\_str());

        stucout = sk;

        Students = s;

    }

    /\* Bendro numatytas duomenys\*/

    Faculty() {

        //tilte = "";

        stucout = 0;

        Students = vector<Student>();

    }

};

/\*\* Klasė bendro masyvo saugoti studentų duomenis

@class Overall\*/

class Overall

{

public: int i, course;

public: char \*name;

public: double average;

        /\* Bendro numatytas duomenys\*/

        Overall() {

            i = 0;

            //name = "";

            course = 0;

            average = 0.00;

        }

        /\* Bendro duomenys

        @param ii - nauja gijos pavadinimo reikšmė

        @param nam - nauja studento vardo reikšmė

        @param cour - nauja studijos kurso reikšmė

        @param aver - nauja studijos vidurkio reikšmė\*/

        \_\_device\_\_ \_\_host\_\_ Overall(int ii, char \*nam, int cour, double aver)

        {

            i = ii; name = nam; course = cour; average = aver;

        }

};

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(ifstream & File, Faculty & P)

{

    string faculty, name, line;

    int studCout, course;

    double average;

    File >> P.tilte >> P.stucout;

    for (int i = 0; i < P.stucout; i++) {

        File >> name >> course >> average;

        Student s = Student(name, course, average);

        P.setStudent(s);

    }

}

//\* Spausdina duomenis į failą

//@param File - failo vardas

//@param S - dvi masyvo vardas

//@param I - masyvo vardas

//@param D - masyvo vardas

//@param length - studentų skaičiai vardas\*/

void WriteData(FILE \* File, const char \*\*S, int \*I, double \*D, int length)

{

    fprintf(File, " Vardas kursas vid\n");

    for (int i = 0; i < length; i++) {

        fprintf(File, " %i) %s \t %i \t %g \n", i + 1, S[i], I[i], D[i]);

    }

}

//\* Spausdina duomenis į failą

//@param File - failo vardas

//@param P - objektų vardas\*/

void WriteData1(FILE \* File, Overall \*P)

{

    fprintf(File, " Vardas kursas vid\n");

    for (int i = 0; i < 19; i++) {

        fprintf(File, " %i) %s \t %i \t %g \n", i + 1, P[i].name, P[i].course, P[i].average);

    }

}

/\* Metodas, vykdymo gijos skirtas pridėti duomenims į bendrą masyvą

@param d\_P - objektų masyvo vardas

@param d\_S - [][] masyvo vardas

@param d\_I - masyvo vardas

@param d\_D - masyvo vardas \*/

\_\_global\_\_ void ExecuteGPU(Overall \*d\_P, char \*\*d\_S, int \*d\_I, double \*d\_D)

{

    int nr = threadIdx.x;

    int i = 0;

    d\_P[nr].i = nr; d\_P[nr].course = d\_I[nr]; d\_P[nr].average = d\_D[nr]; d\_P[nr].name = d\_S[nr];

    i++;

    printf(" %i %i %g\n", d\_P[nr].i, d\_P[nr].course, d\_P[nr].average);

    printf(" %i\n", i);

}

bool sortbyName(Student &a, Student &b) { return (a.name < b.name || (a.name == b.name && a.course < b.course && a.average < b.average)); }

void PutDataIntoArray(Faculty F, string \*P, int \*I, double \*D, int length);

FILE \*File;

int main()

{

    Faculty P1 = Faculty();

    Faculty P2 = Faculty();

    vector<Student> Stud;

    ifstream infile;

    infile.open(FileName);

    ReadData(infile, P1);

    Stud = P1.getStudent();

    sort(Stud.begin(), Stud.end(), sortbyName);

    P2.setStudent(Stud);

    for (int i = 0; i < 19; i++)

        printf(" %s \n", P2.getStudent(i).name);

    string S[50];

    const char\* ss[50];

    int I[50];

    double D[50];

    PutDataIntoArray(P1, S, I, D, P1.stucout);

    for (int i = 0; i < 20; i++)

        ss[i] = S[i].c\_str();

    File = freopen(printname, "w", stderr);

    WriteData(File, ss, I, D, P1.stucout);

    fprintf(File, "-----------------------------------------\n");

    fprintf(File, "Po gijos darbą\n");

    Overall P[20];

    Overall \*d\_P;

    char \*\*d\_S;

    int \*d\_I;

    double \*d\_D;

    int size = 20 \* sizeof(Overall);

    cudaMalloc((void\*\*)&d\_S, 800);

    cudaMalloc((void\*\*)&d\_I, 800);

    cudaMalloc((void\*\*)&d\_D, 800);

    cudaMalloc((void\*\*)&d\_P, 800);

    cudaMemcpy(d\_S, ss, 800, cudaMemcpyHostToDevice);

    cudaMemcpy(d\_I, I, 800, cudaMemcpyHostToDevice);

    cudaMemcpy(d\_D, D, 800, cudaMemcpyHostToDevice);

    ExecuteGPU<<<1,19>>>(d\_P, d\_S, d\_I, d\_D);

    cudaMemcpy(P, d\_P, size, cudaMemcpyDeviceToHost);

    cudaDeviceSynchronize();

    cudaFree(d\_P);

    WriteData1(File,P);

    fprintf(File, "\nPabaiga\n");

    infile.close();

}

void PutDataIntoArray(Faculty F, string \*P, int \*I, double \*D, int length)

{

    for (int i = 0; i < length; i++)

    {

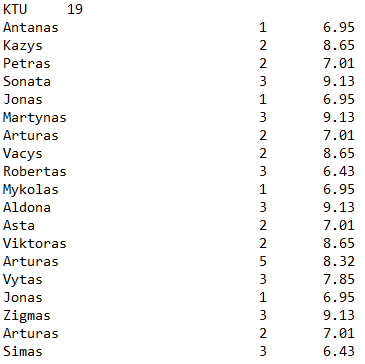
        Student S = F.getStudent(i);

        P[i] = S.name; I[i] = S.course; D[i] = S.average;

    }

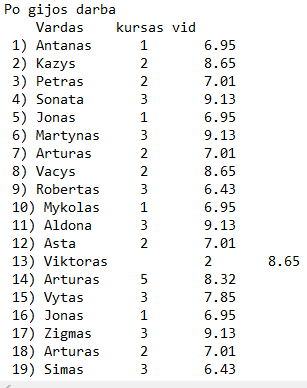
}

**Duomenų failas:**



**Rezultatų failas:**





1. L2. Bendra atmintis

**Užduotis:**

Procesų ***procesas\_1***, ***procesas\_2***, …, ***procesas\_n*** duomenys – pasirinkto tipo struktūrų (bent trys laukai: *string*, *int*, *double*) masyvai *P1(k1), P2(k2), ..., Pn(kn)* atitinkamai (***n***–pasirinktasprocesų skaičius, *4 < n, k1, ..., kn < 10, žr. LD1*).Masyvai yra surikiuoti pagal vieną iš laukų (pasirinkti pačiam).

Kitų procesų ***procesas\_11***, ***procesas\_12***,…, ***procesas\_1m*** duomenimis yra struktūrų tipo „*rikiavimo laukas, kiekis*“ masyvai *V1(r1), V2(r2), ..., Vm(rm)* atitinkamai (***m*** – šio tipo procesų skaičius, *3 < m, r1, ..., rm < 10*). *Rikiavimo laukas* – tai 1-ojo tipo masyvų (*P1, P2, ..., Pn*) rikiavimo laukas.

Prieš procesų (gijų) darbo pradžią įvesti ir spausdinti į failą Grupe\_PavardeV\_L2a\_rez.txt (Grupe\_PavardeV\_L2b\_rez.txt) lentelėmis duomenų masyvus.

Duomenų masyvai perduodami procesams jų sukūrimo metu.

Procesai *procesas\_1*, *procesas\_2*, …, *procesas\_n* iš masyvų *P1(k1), P2(k2), ..., Pn(kn)* duomenų formuoja vieną bendrą **surikiuotą** masyvą ***B(k)***. Masyvo*B(k)*laukai: pasirinktas rikiavimo laukas, kiekis.Kiekvienas procesas, rašantis įbendrąjį masyvą, cikle į **reikiamą** šio masyvo vietą užrašo rikiavimo lauko reikšmę iš savo duomenų masyvo, o kiekiui priskiria 1. Elementai su ta pačia rikiavimo lauko reikšme sujungiami, padidinant lauko „*kiekis*” reikšmę masyve B.

Procesų *procesas\_11*, *procesas\_12*,…, *procesas\_1m* masyvuose *V1(r1), V2(r2), ..., Vm(rm)* surašyti požymiai ir kiekiai, kuriuos turintys duomenys procesų darbo metu iš **bendrojo masyvo** yra naudojami (atimant kiekius ir pašalinant, jei *kiekis=0*).

Baigus darbą visiems procesams, spausdinti į failą Grupe\_PavardeV\_L2a\_rez.txt

(Grupe\_PavardeV\_L2b\_rez.txt) lentele bendrąjį masyvą (procesų darbo metu nieko spausdinti nereikia).

Visi procesai pradeda darbą tuo pačiu metu. Formavimo procesai baigia darbą po to, kai į bendrą masyvą surašo visus savo duomenis. **Naudojimo procesai turi kartoti šalinimo ciklą tol, kol dar galimi pasikeitimai** bendrame masyve *B(k)*.

* + L2a. Java(C++, C#)

**Programų tekstai:**

package main;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author IFF-5-1\_GuzauskasL\_L2a.java : Lukas Gužauskas IFF-5/1 gr.

\*/

public class Main {

private static final String FileName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\IFF-5-1\_GuzauskasL\_L2a\\IFF-5-1\_GuzauskasL\_L2a\_dat.txt";

private static final String FileName2 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\IFF-5-1\_GuzauskasL\_L2a\\IFF-5-1\_GuzauskasL\_L2a\_dat2.txt";

private static final String FileName3 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\IFF-5-1\_GuzauskasL\_L2a\\IFF-5-1\_GuzauskasL\_L2a\_dat3.txt";

private static final String PrintName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\IFF-5-1\_GuzauskasL\_L2a\\IFF-5-1\_GuzauskasL\_L2a\_rez.txt";

private static volatile Monitor B = new Monitor();

/\* Skaito duomenis iš failo

@param br - buferio skaitymo vardas

@param P - objektų vardas\*/

private static void ReadData(BufferedReader br, Faculty P) throws IOException {

try {

String line = br.readLine();

String fakultetas, vardas;

int studSk, kursas;

double vidurkis;

Scanner ed;

ed = new Scanner(line);

fakultetas = ed.next();

studSk = ed.nextInt();

P.tilte = fakultetas;

P.studcout = studSk;

for (int i = 0; i < studSk; i++) {

line = br.readLine();

ed = new Scanner(line);

vardas = ed.next();

kursas = ed.nextInt();

vidurkis = ed.nextDouble();

Student s = new Student(vardas, kursas, vidurkis);

P.setStudent(s);

}

P.Sort();

} catch (NullPointerException e) {

System.out.print("NullPointerException caught");

System.exit(1);

}

}

/\* Spausdina duomenis į failą

@param out - failo vardas

@param P - objektų vardas\*/

private static void WriteTable(BufferedWriter out, Faculty P) throws IOException {

PrintWriter outFile = new PrintWriter (out);

try{

outFile.println("\*\*\*\*" + P.tilte + "\*\*\*\*");

outFile.println(" Vardas Kursas Vidurkis");

for (int i = 0; i < P.studcout; i++) {

Student S = P.getStudent(i);

outFile.printf("%d) %-10s %-5d %-3.2f %n", i + 1, S.name, S.course, S.grade);

}

} catch (NullPointerException e1) {

System.out.print("Error during writing");

System.exit(1);

}

}

/\* Spausdina bendro masyvo duomenys į failą

@param out - failo vardas\*/

private static void PrintArray(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

if (B.B.isEmpty())

outFile.println("Bendras masyvas tuščias");

else {

outFile.println("Bendras masyvas:");

outFile.println("Vardas kiekis");

for (int i = 0; i < B.B.size(); i++)

outFile.printf("%-2d) %-10s %-2d %n", i + 1, B.getB(i).vardas, B.getB(i).kiekis);

}

}

public static void main(String[] args) throws IOException, InterruptedException {

BufferedReader br = new BufferedReader(new FileReader(FileName)); // salinimo visu bendras masyvas

BufferedReader br2 = new BufferedReader(new FileReader(FileName2)); // is bendro masyvo nepasalina ne vieno elemento

BufferedReader br3 = new BufferedReader(new FileReader(FileName3)); // is bendro masyvo pasalinima dalis elementu

BufferedWriter wr = new BufferedWriter(new FileWriter(PrintName));

Faculty P1 = new Faculty();

Faculty P2 = new Faculty();

Faculty P3 = new Faculty();

Faculty P4 = new Faculty();

Faculty P5 = new Faculty();

ReadData(br2, P1);

ReadData(br2, P2);

ReadData(br2, P3);

ReadData(br2, P4);

ReadData(br2, P5);

try {

br.close();

} catch (IOException ex) {

Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);

}

WriteTable(wr, P1);

WriteTable(wr, P2);

WriteTable(wr, P3);

WriteTable(wr, P4);

WriteTable(wr, P5);

wr.write("-------------------------------");

wr.newLine();

List<TempClass> T1 = new ArrayList(Arrays.asList(

new TempClass("Antanas", 1),

new TempClass("Kazys", 1),

new TempClass("Petras", 1),

new TempClass("Sonata", 1)

));

List<TempClass> T2 = new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Martynas", 1),

new TempClass("Lukas", 1),

new TempClass("Vacys", 1),

new TempClass("Robertas", 1)

));

List<TempClass> T3 = new ArrayList(Arrays.asList(

new TempClass("Mykolas", 1),

new TempClass("Aldona", 1),

new TempClass("Asta", 1),

new TempClass("Viktoras", 1),

new TempClass("Arturas", 1),

new TempClass("Vytas", 1)

));

List<TempClass> T4 = new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Zigmas", 1),

new TempClass("Arturas", 1),

new TempClass("Simas", 1)

));

List<TempClass> T5 = new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Mantas", 1),

new TempClass("Vytas", 1),

new TempClass("Robertas", 1)

));

Runnable r1;

r1 = new AddingThread(P1);

Thread t1 = new Thread(r1);

t1.setName("procesas\_1");

Runnable r2;

r2 = new AddingThread(P2);

Thread t2 = new Thread(r2);

t2.setName("procesas\_2");

Runnable r3;

r3 = new AddingThread(P3);

Thread t3 = new Thread(r3);

t3.setName("procesas\_3");

Runnable r4;

r4 = new AddingThread(P4);

Thread t4 = new Thread(r4);

t4.setName("procesas\_4");

Runnable r5;

r5 = new AddingThread(P5);

Thread t5 = new Thread(r5);

t5.setName("procesas\_5");

Runnable r11;

r11 = new RemovingThread(T1);

Thread t11 = new Thread(r11);

t11.setName("procesas\_11");

Runnable r22;

r22 = new RemovingThread(T2);

Thread t22 = new Thread(r22);

t22.setName("procesas\_22");

Runnable r33;

r33 = new RemovingThread(T3);

Thread t33 = new Thread(r33);

t33.setName("procesas\_33");

Runnable r44;

r44 = new RemovingThread(T4);

Thread t44 = new Thread(r44);

t44.setName("procesas\_44");

Runnable r55;

r55 = new RemovingThread(T5);

Thread t55 = new Thread(r55);

t55.setName("procesas\_55");

t1.start();

t2.start();

t3.start();

t4.start();

t5.start();

t1.join();

t2.join();

t3.join();

t4.join();

t5.join();

PrintArray(wr);

t11.start();

t22.start();

t33.start();

t44.start();

t55.start();

t11.join();

t22.join();

t33.join();

t44.join();

t55.join();

wr.write("-------------------------------");

wr.newLine();

wr.write("Po salinimo metoda");

wr.newLine();

PrintArray(wr);

wr.write("Pabaiga");

try {

wr.close();

} catch (IOException ex) {

Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*

@class AddingThread - klasė duomenims ą bendrą masyvà dėti

\*/

static class AddingThread implements Runnable {

private Faculty P;

AddingThread(Faculty P){

this.P = P;

}

@Override

public void run() {

for (int i = 0; i < P.studcout; i++)

B.set(new TempClass(P.getStudent(i).name, 1)); // Deti metodas

}

}

/\*

@class RemovingThread - klasė, skirta pašalinti duomenims iš bendro masyvo

\*/

static class RemovingThread implements Runnable {

private List<TempClass> V = new ArrayList<>();

RemovingThread(List<TempClass> V) {

this.V = V;

}

@Override

public void run(){

for (int i = 0; i < V.size(); i++)

B.get(V.get(i)); // imti metodas

}

}

/\* Klasė sudaro bendro masyvo klasę (Monitroius)

@class Monitor\*/

private static class Monitor{

private volatile List<TempClass> B = new ArrayList<>();

private volatile int threadCounter = 0;

private int yra = 0;

public synchronized TempClass getB(int i){

return B.get(i);

}

public void detiyra(int size){

yra = size;

}

public synchronized void increaseCounter(){

threadCounter++;

}

public synchronized int getThreadCounter(){

return threadCounter;

}

    /\* Dėti į bendrą masyvą.

        @param val - laikino klasės objektas\*/

public synchronized void set(TempClass val){

for (TempClass B1 : B){

if (B1.vardas == null ? val.vardas == null : B1.vardas.equals(val.vardas)){

B1.kiekis++;

return;

}

}

if (B.isEmpty())

B.add(val);

else if (B.get(0).vardas.compareToIgnoreCase(val.vardas) > 0)

B.add(0, val);

else if (B.get(B.size() - 1).vardas.compareToIgnoreCase(val.vardas) < 0)

B.add(B.size(), val);

else {

int i = 0;

while (B.get(i).vardas.compareToIgnoreCase(val.vardas) < 0)

i++;

B.add(i, val);

}

notifyAll();

}

/\* Grąžina iš bendro masyvo kai po pašalinimą.

        @param V - laikino klasės objektas\*/

public synchronized void get(TempClass V) {

try{

while (B.isEmpty())

wait();

} catch (InterruptedException e){}

for (int i = 0; i < B.size(); i++){

if(B.get(i).vardas.equalsIgnoreCase(V.vardas)){

B.get(i).kiekis--;

if (B.get(i).kiekis == 0)

B.remove(i);

return;

}

}

notifyAll();

}

}

/\* Klasė išrinkimo saugoti studentų rikiavimo lauko ir kiekio duomenis

@class TempClass\*/

static class TempClass {

public String vardas; // Rikiavimo 1 laukas

public int kiekis; // kiekis

/\* Išrinkimo duomenys

    @param vardas - nauja rikiavimo lauko reikšmė

    @param kiekis - nauja kiekio reikšmė\*/

public TempClass(String vardas, int kiekis) {

this.vardas = vardas;

this.kiekis = kiekis;

}

/\* Išrinkimo numatytas duomenys\*/

public TempClass() {}

}

/\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

static class Faculty{

public String tilte; // fakulteto pavadinimas

public int studcout; // studentų skaičius

List<Student> Stud; // studentų rinkinys

public void Sort() {

Stud.sort(new NameComparator());

}

/\* Grąžina nurodyto indekso studentų objektą.

        @param i - studentų indeksas\*/

public Student getStudent(int i){

return Stud.get(i);

}

/\* Dėti į studentų objektų masyvą naują studentą

        ir masyvo dydį.

        @param i - indeksas

        @param s - studentų objektas\*/

public void setStudent(int i, Student s){

Stud.add(i, s);

}

/\* Dėti į studentų objektų masyvą naują studentą.

        @param s - studentų objektas\*/

public void setStudent(Student s){

Stud.add(s);

}

/\* Grąžina visą studentų objektą.\*/

public List<Student> getStudent() {

return Stud;

}

public void setStudent(List<Student> s){

this.Stud = s;

}

/\* Fakulteto duomenys

        @param tilte - nauja fauklteto pavadinimo reikšmė

        @param studcout - nauja studentu skaiciaus reikšmė

        @param stud - nauja objektu reikšmė\*/

public Faculty(String tilte, int studcout, List Stud){

this.tilte = tilte;

this.studcout = studcout;

this.Stud = Stud;

}

/\* Fakulteto numatytas duomenys\*/

public Faculty(){

this.tilte = "";

this.studcout = -1;

this.Stud = new ArrayList();

}

}

static class NameComparator implements Comparator<Student> {

@Override

public int compare(Student a, Student b) {

return a.name.compareToIgnoreCase(b.name);

}

}

/\* Klasė studento saugoti duomenis

@class Student\*/

static class Student{

public String name; // vardas

public int course; // kursas

public double grade; // pažymių vidurkis

/\* Studento duomenys

@param nam - nauja vardo reikšmė

@param course - nauja kurso reikšme

@param grade - nauja vidurkio reikšmė\*/

public Student(String name, int course, double grade){

this.name = name;

this.course = course;

this.grade = grade;

}

/\* Studento numatytas duomenys \*/

public Student(){

this.name = "";

this.course = -1;

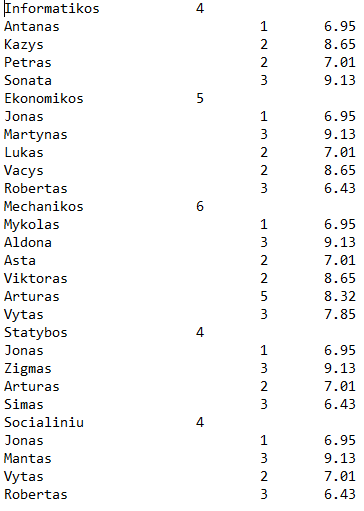
this.grade = -1;

}

}

}

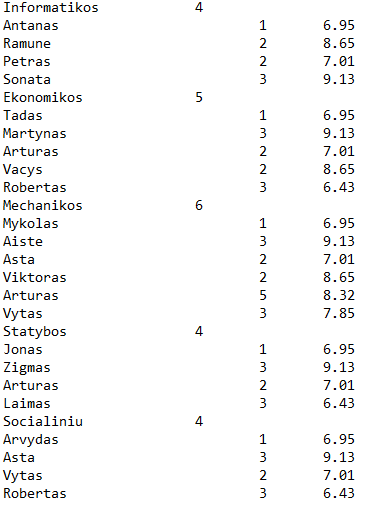
**Duomenų failas (po šalinimo bendras masyvas tuščias):**



**Duomenų failas(iš bendro masyvo nepašalina nė vieno elemento):**



**Duomenų failas(iš bendro masyvo pašalinama dalis elementų):**



**Rezultatų failas:**





* + L2b. C++ & OpenMP

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L2b.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <fstream>

#include <sstream>

#include <iterator>

#include <vector>

#include <thread>

#include <mutex>

#include <cstdio>

#include <algorithm>

#include <omp.h>

#include <iomanip>

using namespace std;

const string FileName1 = "IFF-5-1\_GuzauskasL\_L2b\_dat1.txt";

const string FileName2 = "IFF-5-1\_GuzauskasL\_L2b\_dat2.txt";

const string FileName3 = "IFF-5-1\_GuzauskasL\_L2b\_dat3.txt";

const string PrintName = "IFF-5-1\_GuzauskasL\_L2b\_rez.txt";

class Student

{

public: string name; // vardas ir pavardė

public: int course; // kursas

public: double average; // pažymių vidurkis

public: Student() {}

/\* Studento duomenys

@param nam - nauja vardo ir pavardės reikšmė

@param cour - nauja kurso reikšmė

@param ave - nauja vidurkio reikšmė\*/

public: Student(string nam, int cour, double ave)

{

name = nam;

course = cour;

average = ave;

}

bool operator < (const Student & other) { return name < other.name; }

};

/\*\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty

{

public: string tilte; // fakulteto pavadinimas

int stucout; // studentu skaicius

vector<Student> Students;

/\* Grąžina nurodyto indekso studentų objektą.

@param i - studentų indeksas\*/

public: Student getStudent(int i) {

return Students[i];

}

/\* Dėti į studentų objektų masyvą naują studentą

ir masyvo dydį.

@param i - indeksas

@param s - studentų objektas\*/

void setStudent(int i, Student s) {

Students[i] = s;

}

/\* Dėti į studentų objektų masyvą naują studentą.

@param s - studentų objektas\*/

void setStudent(Student s) {

Students.push\_back(s);

}

/\* Grąžina visą studentų objektą.\*/

vector<Student> getStudent() {

return Students;

}

void setStudent(vector<Student> s) {

Students = s;

}

/\* Fakulteto duomenys

@param p - nauja fauklteto pavadinimo reikšmė

@param sk - nauja studentu skaiciaus reikšmė

@param s - nauja objektu reikšmė\*/

Faculty(string p, int sk, vector<Student> s) {

tilte = p;

stucout = sk;

Students = s;

}

/\* Fakulteto numatytas duomenys\*/

Faculty() {

tilte = "";

stucout = 0;

Students = vector<Student>();

}

};

/\* Klasė išrinkimo saugoti studentų rikiavimo lauko ir kiekio duomenis

@class Overall\*/

class Overall {

private: vector<Overall> B;

public: string name;

public: int countt;

public:

/\* Išrinkimo duomenys

@param t - nauja objekto reikšmė\*/

Overall(const Overall & t) {

name = t.name;

countt = t.countt;

}

/\* Išrinkimo duomenys

@param vardas - nauja rikiavimo lauko reikšmė

@param kiekis - nauja kiekio reikšmė\*/

Overall(string nam, int cou) {

this->name = nam;

this->countt = cou;

}

/\* Išrinkimo numatytas duomenys\*/

Overall() {

name = "";

countt = 0;

}

};

/\* Klasė sudaro bendro masyvo klasę (Monitroius)

@class Monitor\*/

class Monitor {

private: vector<Overall> B = vector<Overall>();

public:

/\* Grąžina bendro masyvo elementą.

@param i - indeksas\*/

Overall getB(int i){

return B[i];

}

/\* Grąžina true ar bendro masyvo tuščias\*/

bool Bempty() { if (B.empty()) return true;}

/\* Grąžina bendro masyvo dydį\*/

int Bsize() { return B.size();}

/\* Dėti į bendrą masyvą.

@param val - laikino klasės objektas\*/

void set(Overall val) {

for (int i = 0; i < B.size(); i++)

if (B[i].name == val.name) {

B[i].countt++;

return;

}

if (B.empty())

B.push\_back(val);

else if (B[0].name.compare(val.name) > 0)

B.insert(B.begin(), val);

else if (B[B.size() - 1].name.compare(val.name) < 0)

B.insert(B.end(), val);

else {

int i = 0;

while (B[i].name.compare(val.name) < 0)

i++;

B.insert(B.begin() + i, val);

}

}

/\* Grąžina iš bendro masyvo kai po pašalinimą.

@param V - laikino klasės objektas\*/

bool get(Overall val){

for (int i = 0; i < B.size(); i++)

if (B[i].name == val.name) {

B[i].countt--;

if (B[i].countt == 0)

B.erase(B.begin() + i);

return true;

}

return false;

}

};

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(ifstream & File, Faculty & P)

{

string faculty, name, line;

int course;

double average;

File >> P.tilte >> P.stucout;

for (int i = 0; i < P.stucout; i++) {

File >> name >> course >> average;

Student s = Student(name, course, average);

P.setStudent(s);

}

vector<Student> Stud;

Stud = P.getStudent();

sort(Stud.begin(), Stud.end());

P.setStudent(Stud);

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void WriteData(FILE \* File, Faculty P)

{

fprintf(File, " \*\*\* %s \*\*\* \n", P.tilte.c\_str());

fprintf(File, " Vardas kursas vid\n");

for (int i = 0; i < P.stucout; i++) {

Student S = P.getStudent(i);

fprintf(File, " %i) %s \t %i \t %g \n", i + 1, S.name.c\_str(), S.course, S.average);

}

}

/\* Metodas bendro masyvo spausdinimui

@param B - bendras masyvas\*/

void PrintArray(FILE \*File, Monitor B) {

if (B.Bempty())

fprintf(File, "Bendras masyvas tušcias\n");

else {

fprintf(File, "Bendras masyvas:\n");

fprintf(File, "Nr. Vardas kiekis\n");

for (int i = 0; i < B.Bsize(); i++)

fprintf(File, " %-2i %-10s %-2d\n", i + 1, B.getB(i).name.c\_str(), B.getB(i).countt);

}

}

FILE \*File;

int main()

{

setlocale(LC\_ALL, "Lithuanian");

Faculty P1 = Faculty();

Faculty P2 = Faculty();

Faculty P3 = Faculty();

Faculty P4 = Faculty();

Faculty P5 = Faculty();

ifstream infile;

infile.open(FileName1);

ReadData(infile, P1);

ReadData(infile, P2);

ReadData(infile, P3);

ReadData(infile, P4);

ReadData(infile, P5);

infile.close();

File = freopen(PrintName.c\_str(), "w", stderr);

WriteData(File, P1);

WriteData(File, P2);

WriteData(File, P3);

WriteData(File, P4);

WriteData(File, P5);

fprintf(File, "-----------------------------------\n");

vector<Overall> T1 = {

Overall("Antanas", 1),

Overall("Kazys", 1),

Overall("Petras", 1),

Overall("Sonata", 1)

};

vector<Overall> T2 = {

Overall("Jonas", 1),

Overall("Martynas", 1),

Overall("Lukas", 1),

Overall("Vacys", 1),

Overall("Robertas", 1)

};

vector<Overall> T3 = {

Overall("Mykolas", 1),

Overall("Aldona", 1),

Overall("Asta", 1),

Overall("Viktoras", 1),

Overall("Arturas", 1),

Overall("Vytas", 1)

};

vector<Overall> T4 = {

Overall("Jonas", 1),

Overall("Zigmas", 1),

Overall("Arturas", 1),

Overall("Simas", 1)

};

vector<Overall> T5 = {

Overall("Jonas", 1),

Overall("Mantas", 1),

Overall("Vytas", 1),

Overall("Robertas", 1)

};

vector<Overall> T6 = {

Overall("gfb", 1),

Overall("Mas", 1),

Overall("nn", 1),

Overall("gg", 1)

};

Monitor B;

int threadCounter = 0;

int threadCt = 10;

omp\_set\_num\_threads(threadCt);

//--------- Lygiagretus sritis -------------------

#pragma omp parallel shared(B, threadCounter)

{

vector <Overall> V;

Faculty P;

bool check = false;

bool removed = false;

bool pass = false;

int nr = omp\_get\_thread\_num();

int b = 0;

switch (nr)

{

case 0:

P = P1;

break;

case 1:

P = P2;

break;

case 2:

P = P3;

break;

case 3:

P = P4;

break;

case 4:

P = P5;

break;

case 5:

V = T1;

break;

case 6:

V = T2;

break;

case 7:

V = T3;

break;

case 8:

V = T4;

break;

case 9:

V = T5;

break;

}

if (nr < 5)

{

for (int i = 0; i < P.stucout; i++)

{

#pragma omp critical

{

B.set(Overall(P.getStudent(i).name, 1));

}

}

#pragma omp flush (threadCounter, B)

#pragma omp critical

{

threadCounter+= 1;

}

#pragma omp flush (threadCounter, B)

}

else {

int a;

#pragma omp flush (threadCounter, B)

#pragma omp critical

{

a = threadCounter;

}

for (vector<Overall>::iterator iter = V.begin(); iter != V.end();) {

#pragma omp critical

{

removed = B.get(\*iter);

}

if (removed){

iter = V.erase(iter);

}

else {

++iter;

}

}

#pragma omp critical

{

a = threadCounter;

}

}

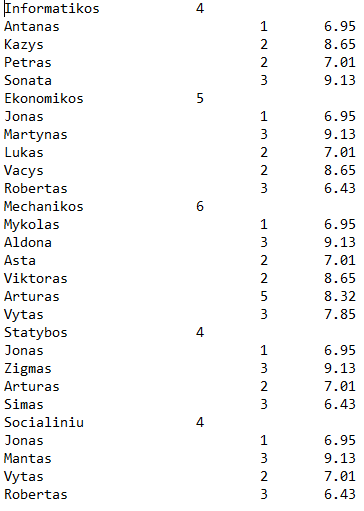
}

PrintArray(File,B);

return 0;

}

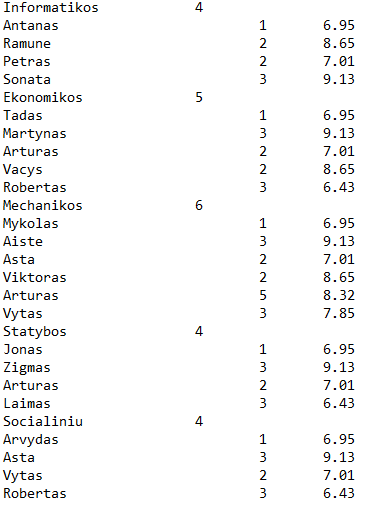
**Duomenų failas (po šalinimo bendras masyvas tuščias):**



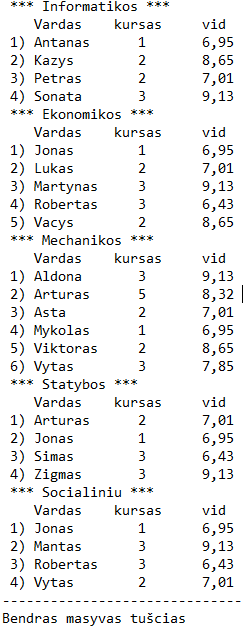
**Duomenų failas (iš bendro masyvo nepašalina nė vieno elemento):**



**Duomenų failas (iš bendro masyvo pašalinama dalis elementų):**



**Rezultatų failas:**



1. L3. Paskirstyta atmintis

**Užduotis:**

Procesų ***procesas\_1***, ***procesas\_2***, …, ***procesas\_n*** duomenys – pasirinkto tipo struktūrų (bent trys laukai: *string*, *int*, *double*) masyvai *P1(k1), P2(k2), ..., Pn(kn)* atitinkamai (***n***–pasirinktasprocesų skaičius, *4 < n, k1, ..., kn < 10*).Masyvai yra surikiuoti pagal vieną iš laukų (pasirinkti pačiam).

Kitų procesų ***procesas\_11***, ***procesas\_12***,…, ***procesas\_1m*** duomenimis yra struktūrų tipo „*rikiavimo laukas, kiekis*“ masyvai *V1(r1), V2(r2), ..., Vm(rm)* atitinkamai (***m*** – šio tipo procesų skaičius, *3 < m, r1, ..., rm < 10*). *Rikiavimo laukas* – tai 1-ojo tipo masyvų (*P1, P2, ..., Pn*) rikiavimo laukas.

Masyvus *P1(k1), P2(k2), ..., Pn(kn)* ir *V1(r1), V2(r2), ..., Vm(rm)* įveda iš duomenų failo ir spausdina į failą (Grupe\_PavardeV\_L3a\_rez.txt (Grupe\_PavardeV\_L3b\_rez.txt) lentelėmis papildomas ***procesas\_00***. Po to šis procesas persiunčia masyvus atitinkamiems procesams ***procesas\_1***, ***procesas\_2***, …, ***procesas\_n***, ***procesas\_11***, ***procesas\_12***,…, ***procesas\_1m***.

Procesų *procesas\_1*, *procesas\_2*, …, *procesas\_n* masyvuose *P1(k1), P2(k2), ..., Pn(kn)* esančius duomenis reikia surašyti į vieną bendrą **surikiuotą** masyvą ***B(k)*** (rikiavimo raktas – *pasirinktas laukas*). Rašymo metu „panašūs” elementai sujungiami, visiems elementams įvedus papildomą lauką „*kiekis*”, „nepanašūs“ – įterpiami į **reikiamą** vietą. Masyve *B(k)*: pasirinktas laukas, kiekis.

Procesų *procesas\_11*, *procesas\_12*,…, *procesas\_1m* masyvuose *V1(r1), V2(r2), ..., Vm(rm)* surašyti požymiai ir kiekiai, kuriuos turintys duomenys procesų darbo metu iš **bendrojo masyvo** yra naudojami (atimant kiekius ir pašalinant, jei *kiekis=0*).

Tiek pildymo procesai *procesas\_1*, *procesas\_2*, …, *procesas\_n*, tiek naudojimo procesai *procesas\_11*, *procesas\_12*,…, *procesas\_1m* su masyvu *B(k)* dirba ne tiesiogiai, bet siunčiasavo duomenis valdymo procesui***valdytojas***, kuris saugobendrąjį masyvą ir papildo arba šalina iš jo.

Visi procesai pradeda darbą tuo pačiu metu. Pildymo procesai baigia darbą, kai į valdymo procesą persiunčia visus savo duomenis. **Naudojimo procesai turi kartoti** **duomenų siuntimo ciklą tol, kol dar galimi pasikeitimai bendrame** **masyve *B(k)*.** Pildymo ir naudojimo procesams baigusdarbą, valdymo procesas turi persiųsti masyvą B papildomamprocesui *procesas\_00*, kuris jį turi spausdinti į failą lentele.

* + L3a. Any2One

**Programų tekstai:**

package guzauskasld3\_a;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.jcsp.lang.\*;

/\*\*

\*

\* @author IFF-5-1\_GuzauskasL\_L3a.java : Lukas Gužauskas IFF-5/1 gr.

\*/

/\* Klasė studento saugoti duomenis

@class Student\*/

class Student{

public String name; // vardas

public int course; // kursas

public double grade; // pažymių vidurkis

/\* Studento duomenys

@param nam - nauja vardo reikšmė

@param course - nauja kurso reikšme

@param grade - nauja vidurkio reikšmė\*/

public Student(String name, int course, double grade){

this.name = name;

this.course = course;

this.grade = grade;

}

/\* Studento numatytas duomenys \*/

public Student(){

this.name = "";

this.course = -1;

this.grade = -1;

}

}

/\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty{

public String tilte; // fakulteto pavadinimas

public int studcout; // studentų skaičius

List<Student> Stud; // studentų rinkinys

public void Sort() {

Stud.sort(new NameComparator());

}

/\* Grąžina nurodyto indekso studentų objektą.

    @param i - studentų indeksas\*/

public Student getStudent(int i){

return Stud.get(i);

}

/\* Dėti į studentų objektų masyvą naują studentą

    ir masyvo dydį.

    @param i - indeksas

    @param s - studentų objektas\*/

public void setStudent(int i, Student s){

Stud.add(i, s);

}

/\* Dėti į studentų objektų masyvą naują studentą.

    @param s - studentų objektas\*/

public void setStudent(Student s){

Stud.add(s);

}

/\* Grąžina visą studentų objektą.\*/

public List<Student> getStudent() {

return Stud;

}

public void setStudent(List<Student> s){

this.Stud = s;

}

/\* Fakulteto duomenys

    @param tilte - nauja fauklteto pavadinimo reikšmė

    @param studcout - nauja studentu skaiciaus reikšmė

    @param stud - nauja objektu reikšmė\*/

public Faculty(String tilte, int studcout, List Stud){

this.tilte = tilte;

this.studcout = studcout;

this.Stud = Stud;

}

/\* Fakulteto numatytas duomenys\*/

public Faculty(){

this.tilte = "";

this.studcout = -1;

this.Stud = new ArrayList();

}

}

class NameComparator implements Comparator<Student> {

@Override

public int compare(Student a, Student b) {

return a.name.compareToIgnoreCase(b.name);

}

}

/\* Klasė išrinkimo saugoti studentų rikiavimo lauko ir kiekio duomenis

@class TempClass\*/

class TempClass {

public String vardas; // Rikiavimo 1 laukas

public int kiekis; // kiekis

/\* Išrinkimo duomenys

    @param vardas - nauja rikiavimo lauko reikšmė

    @param kiekis - nauja kiekio reikšmė\*/

public TempClass(String vardas, int kiekis) {

this.vardas = vardas;

this.kiekis = kiekis;

}

/\* Išrinkimo numatytas duomenys\*/

public TempClass() {}

}

/\* Klasė pildymo procesas, kuris reikia vykdyti rašymo metodu į bendrą masyvą

@class Writer\*/

class Writer implements CSProcess{

private Faculty P;

private final ChannelOutput out;

private final AltingChannelInput in;

/\* Writer duomenys

    @param out - nauja kanalo išvesčio reikšmė

    @param in - nauja kanalo įvesčio reikšmė\*/

Writer(ChannelOutput out, AltingChannelInput in){

this.out = out;

this.in = in;

}

@Override

public void run(){

P = (Faculty) in.read(); // skaityti obejkto klasę iš Process00

if (P != null){

for (int i = 0; i < P.studcout; i++)

out.write(new TempClass(P.getStudent(i).name,1)); // persiusti į bendrą masyvą(B)

}

out.write(null);

}

}

/\* Klasė naudojimo procesas, kuris reikia vykdyti trinimą metodą iš bendros masyvos

@class Remover\*/

class Remover implements CSProcess{

private List<TempClass> V;

private final ChannelOutput out;

private final AltingChannelInput in;

/\* Remover duomenys

    @param out - nauja kanalo išvesčio reikšmė

    @param in - nauja kanalo įvesčio reikšmė\*/

Remover(ChannelOutput out, AltingChannelInput in){

this.out = out;

this.in = in;

}

@Override

public void run(){

V = (List<TempClass>) in.read(); // skaityti obejkto klase is Process00

if (V != null){

for (int i = 0; i < V.size(); i++)

out.write(V.get(i)); // naudoti is bendros masyvos(B)

}

out.write(null);

}

}

/\* Klasė, skirta saugoti pagrindiniam masyvui

@class Remover\*/

class Monitor {

private final List<TempClass> B;

public Monitor() {

this.B = new ArrayList<>();

}

public TempClass get(int i){

return B.get(i);

}

public void remove(int i){

B.remove(i);

}

public void add(TempClass V){

B.add(V);

}

public void add(int i, TempClass V){

B.add(i, V);

}

public int size(){

return B.size();

}

public boolean isEmpty(){

return B.isEmpty();

}

public List<TempClass> B(){

return B;

}

}

/\* Klasė valdytojo saugoti bendrą masyvą(B)

@class ProcessManager\*/

class ProcessManager implements CSProcess {

Monitor B = new Monitor();

private final AltingChannelInput removerInput;

private final AltingChannelInput writerInput;

private final ChannelOutput mainarrayOutput;

private int finishedW = 0;

private int finishedR = 0;

/\* ProcessManager duomenys

    @param in0 - nauja kanalo įvesčio reikšmė

    @param in0 - nauja kanalo įvesčio reikšmė

@param out - nauja kanalo išvesčio reikšmė\*/

public ProcessManager(AltingChannelInput in0, AltingChannelInput in1, ChannelOutput out){

this.writerInput = in0;

this.removerInput = in1;

this.mainarrayOutput = out;

}

@Override

public void run(){

while(finishedW != 5 || finishedR != 5){

if(finishedW != 5)

Writer(); // skaityti is pildymo proceso

else

Remover(); // skaityti is naudojimo proceso

}

mainarrayOutput.write(B);

PrintArray1();

}

/\* Grąžina iš bendro masyvo kai po pašalinimą.

    @param V - laikino klasės objektas\*/

public void removeValue(TempClass V) {

for (int i = 0; i < B.size(); i++){

if(B.get(i).vardas.equalsIgnoreCase(V.vardas)){

B.get(i).kiekis--;

if (B.get(i).kiekis == 0)

B.remove(i);

return;

}

}

}

/\* Dėti į bendrą masyvą.

    @param val - laikino klasės objektas\*/

public boolean addValue(TempClass val){

for (TempClass B1 : B.B()){

if (B1.vardas == null ? val.vardas == null : B1.vardas.equals(val.vardas)){

B1.kiekis++;

return true;

}

}

if (B.isEmpty())

B.add(val);

else if (B.get(0).vardas.compareToIgnoreCase(val.vardas) > 0)

B.add(0, val);

else if (B.get(B.size() - 1).vardas.compareToIgnoreCase(val.vardas) < 0)

B.add(B.size(), val);

else {

int i = 0;

while (B.get(i).vardas.compareToIgnoreCase(val.vardas) < 0)

i++;

B.add(i, val);

}

return true;

}

/\* Spausdina bendro masyvo duomenius\*/

public void PrintArray1(){

if (B.isEmpty())

System.out.println("Bendras masyvas tuščias");

else {

System.out.println("Bendras masyvas:");

System.out.println("Vardas kiekis");

for (int i = 0; i < B.size(); i++)

System.out.printf("%-2d) %-10s %-2d %n", i + 1, B.get(i).vardas, B.get(i).kiekis);

}

}

/\* Writer metodas, skaityti iš Writer klasės(papildymo procesas), jei data nelygus null tad elementai dėti į bendrą masyvą(B)\*/

public void Writer(){

TempClass data;

data = (TempClass) writerInput.read();

if (data == null)

finishedW++;

else

addValue(data);

}

/\* Remover metodas, skaityti iš Remover klasės(naudojimo procesas), jei data nelygus null tad elementai trinti iš bendros masyvos(B)\*/

public void Remover(){

TempClass data;

data = (TempClass) removerInput.read();

if (data != null)

removeValue(data);

else

finishedR++;

}

}

/\* Klasė process00 vykdyta per skaitymą iš failo ir spaudinta į failą.

@class ProcessManager\*/

class Process00 implements CSProcess{

private List<Faculty> P = new ArrayList<>();

private List<TempClass>[] V;

Monitor B = new Monitor();

private final ChannelOutput[] Poutput;

private final ChannelOutput[] Voutput;

private final AltingChannelInput mainarrayInput;

private final BufferedWriter out;

private final BufferedReader br;

private boolean check = true;

private boolean check1 = true;

/\* Process00 duomenys

    @param br - nauja failo skaitymo reikšmė

@param out - nauja buferio rašymo reikšmė

@param outP - nauja kanalo išvesčio masyvo reikšmė

    @param outV - nauja kanalo išvesčio masyvo reikšmė

@param in - nauja kanalo įvesčio reikšmė\*/

public Process00(String br, BufferedWriter out, ChannelOutput[] outP, ChannelOutput[] outV, AltingChannelInput in) throws FileNotFoundException, IOException

{

this.br = new BufferedReader(new FileReader(br));

this.out = out;

this.Poutput = outP;

this.Voutput = outV;

this.mainarrayInput = in;

}

@Override

public void run(){

try {

while(check){

if(check1){

ReadData(br);

try {

br.close();

} catch (IOException ex) {

Logger.getLogger(Process00.class.getName()).log(Level.SEVERE, null, ex);

}

ReadDataV();

WriteTableP(out);

WriteTableV(out);

PVtoWriter(); // P ir V masyvams persiusti i pildymo ir naudojimo procesus.

check1 = false;

}else{

WriteTableB(out);

check = false;

}

}

} catch (IOException ex) {

Logger.getLogger(Process00.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\* Skaito duomenis iš failo

@param br - buferio skaitymo vardas\*/

public void ReadData(BufferedReader br) throws IOException {

try {

for (int i = 0; i < 5; i++){

String line = br.readLine();

String fakultetas, vardas;

int studSk, kursas;

double vidurkis;

Scanner ed;

ed = new Scanner(line);

fakultetas = ed.next();

studSk = ed.nextInt();

List<Student> S = new ArrayList<>();

for (int j = 0; j < studSk; j++) {

line = br.readLine();

ed = new Scanner(line);

vardas = ed.next();

kursas = ed.nextInt();

vidurkis = ed.nextDouble();

Student s = new Student(vardas, kursas, vidurkis);

S.add(s);

}

Faculty PP = new Faculty(fakultetas, studSk, S);

P.add(PP);

P.get(i).Sort();

}

} catch (NullPointerException e) {

System.out.print("NullPointerException caught");

System.exit(1);

}

}

public void ReadDataV(){

V = new List[]{new ArrayList(Arrays.asList(

new TempClass("Antanas", 1),

new TempClass("Kazys", 1),

new TempClass("Petras", 1),

new TempClass("Sonata", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Martynas", 1),

new TempClass("Lukas", 1),

new TempClass("Vacys", 1),

new TempClass("Robertas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Mykolas", 1),

new TempClass("Aldona", 1),

new TempClass("Asta", 1),

new TempClass("Viktoras", 1),

new TempClass("Arturas", 1),

new TempClass("Vytas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Zigmas", 1),

new TempClass("Arturas", 1),

new TempClass("Simas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Mantas", 1),

new TempClass("Vytas", 1),

new TempClass("Robertas", 1)

))};

}

/\* Spausdina fakulteto duomenis į failą

@param out - failo vardas\*/

public void WriteTableP(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

try{

outFile.println("######## P(n) masyvai ########");

outFile.println("-------------------------------");

for (Faculty P1 : P) {

outFile.println("\*\*\*\*" + P1.tilte + "\*\*\*\*");

outFile.println(" Vardas Kursas Vidurkis");

for (int i = 0; i < P1.studcout; i++) {

Student S = P1.getStudent(i);

outFile.printf("%d) %-10s %-5d %-3.2f %n", i + 1, S.name, S.course, S.grade);

}

}

outFile.println("-------------------------------");

} catch (NullPointerException e1) {

System.out.print("Error during writing");

System.exit(1);

}

}

/\* Spausdina TempClass duomenys į failą

@param out - failo vardas\*/

public void WriteTableV(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

int j = 1;

outFile.println("######## V(m) masyvai ########");

outFile.println("-------------------------------");

for(List<TempClass> T1 : V){

if (T1.isEmpty())

outFile.println("Bendras masyvas tuščias");

else {

outFile.printf("V%d masyvas: %n", j); j++;

outFile.println("Vardas kiekis");

for (int i = 0; i < T1.size(); i++)

outFile.printf("%d) %-10s %-2d %n", i + 1, T1.get(i).vardas, T1.get(i).kiekis);

}

}

outFile.println("-------------------------------");

}

/\* Spausdina bendro masyvo duomenys į failą

@param out - failo vardas\*/

public void WriteTableB(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

B = (Monitor) mainarrayInput.read();

outFile.println("######## Po valdymo procesus ########");

outFile.println("-------------------------------");

if (B.isEmpty())

outFile.println("Bendras masyvas tuščias");

else {

outFile.println("Bendras masyvas:");

outFile.println("Vardas kiekis");

for (int i = 0; i < B.size(); i++)

outFile.printf("%-2d) %-10s %-2d %n", i + 1, B.get(i).vardas, B.get(i).kiekis);

}

}

/\* Fakulteto ir TempClass masyvo elementų persiųsti į valdytojo(ProcessManager class) procesą.

@param out - failo vardas\*/

public void PVtoWriter(){

for (int i = 0; i < P.size(); i++) {

Poutput[i].write(P.get(i));

}

for (int i = 0; i < V.length; i++){

Voutput[i].write(V[i]);

}

}

}

public class GuzauskasLD3\_a {

private static final String FileName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_a\\IFF-5-1\_GuzauskasL\_L3a\_dat.txt";

private static final String FileName2 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_a\\IFF-5-1\_GuzauskasL\_L3a\_dat2.txt";

private static final String FileName3 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_a\\IFF-5-1\_GuzauskasL\_L3a\_dat3.txt";

private static final String PrintName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_a\\IFF-5-1\_GuzauskasL\_L3a\_rez.txt";

public static Any2OneChannel[] P = new Any2OneChannel[5];

public static Any2OneChannel[] V = new Any2OneChannel[5];

public static Any2OneChannel Vv = Channel.any2one();

public static Any2OneChannel W = Channel.any2one();

public static Any2OneChannel R = Channel.any2one();

public static One2OneChannel B = Channel.one2one();

public static void main(String[] args) throws FileNotFoundException, IOException {

BufferedReader br = new BufferedReader(new FileReader(FileName)); // salinimo visu bendras masyvas

BufferedReader br2 = new BufferedReader(new FileReader(FileName2)); // is bendro masyvo nepasalina ne vieno elemento

BufferedReader br3 = new BufferedReader(new FileReader(FileName3)); // is bendro masyvo pasalinima dalis elementu

BufferedWriter wr = new BufferedWriter(new FileWriter(PrintName));

for (int i = 0; i < 5; i++){

P[i] = Channel.any2one();

V[i] = Channel.any2one();

}

ChannelOutput[] Pin = new ChannelOutput[5];

ChannelOutput[] Vin = new ChannelOutput[5];

for (int i = 0; i < 5; i++){

Pin[i] = P[i].out();

Vin[i] = V[i].out();

}

new Parallel(new CSProcess[]{new Process00(FileName3, wr, Pin, Vin , B.in()),

new Writer(W.out(), P[0].in()),

new Writer(W.out(), P[1].in()),

new Writer(W.out(), P[2].in()),

new Writer(W.out(), P[3].in()),

new Writer(W.out(), P[4].in()),

new Remover(R.out(), V[0].in()),

new Remover(R.out(), V[1].in()),

new Remover(R.out(), V[2].in()),

new Remover(R.out(), V[3].in()),

new Remover(R.out(), V[4].in()),

new ProcessManager(W.in(), R.in(), B.out())

}

).run();

try {

wr.close();

} catch (IOException ex) {

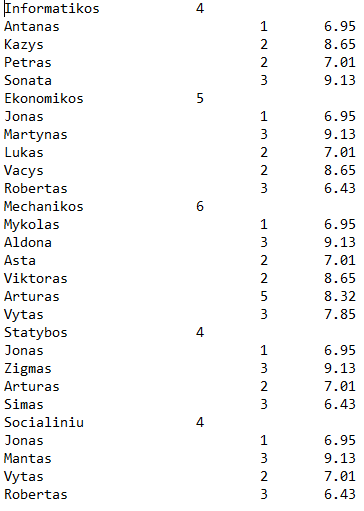
Logger.getLogger(GuzauskasLD3\_a.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

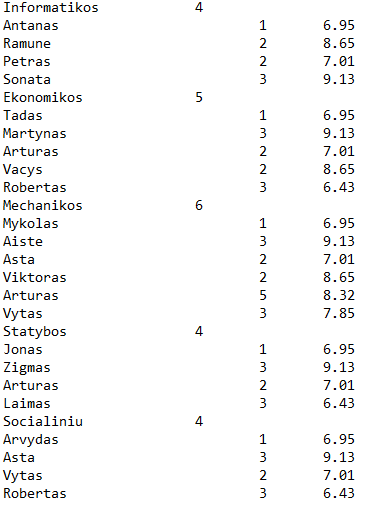
**Duomenų failas(po šalinimo bendras masyvas tuščias):**



**Duomenų failas(iš bendro masyvo nepašalina nė vieno elemento):**



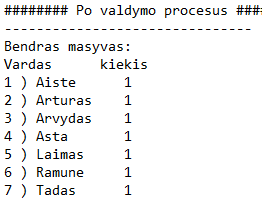
**Duomenų failas(iš bendro masyvo pašalinama dalis elementų):**



**Rezultatų failas:**







* + L3b. One2One

**Programų tekstai:**

package guzauskasld3\_b;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.jcsp.lang.\*;

/\*\*

\*

\* @author IFF-5-1\_GuzauskasL\_L3b.java : Lukas Gužauskas IFF-5/1 gr.

\*/

/\* Klasė studento saugoti duomenis

@class Student\*/

class Student{

public String name; // vardas

public int course; // kursas

public double grade; // pažymių vidurkis

/\* Studento duomenys

@param nam - nauja vardo reikšmė

@param course - nauja kurso reikšme

@param grade - nauja vidurkio reikšmė\*/

public Student(String name, int course, double grade){

this.name = name;

this.course = course;

this.grade = grade;

}

/\* Studento numatytas duomenys \*/

public Student(){

this.name = "";

this.course = -1;

this.grade = -1;

}

}

/\* Klasė fakulteto saugoti studentų duomenis

@class Faculty\*/

class Faculty{

public String tilte; // fakulteto pavadinimas

public int studcout; // studentų skaičius

List<Student> Stud; // studentų rinkinys

public void Sort() {

Stud.sort(new NameComparator());

}

/\* Grąžina nurodyto indekso studentų objektą.

    @param i - studentų indeksas\*/

public Student getStudent(int i){

return Stud.get(i);

}

/\* Dėti į studentų objektų masyvą naują studentą

    ir masyvo dydį.

    @param i - indeksas

    @param s - studentų objektas\*/

public void setStudent(int i, Student s){

Stud.add(i, s);

}

/\* Dėti į studentų objektų masyvą naują studentą.

    @param s - studentų objektas\*/

public void setStudent(Student s){

Stud.add(s);

}

/\* Grąžina visą studentų objektą.\*/

public List<Student> getStudent() {

return Stud;

}

public void setStudent(List<Student> s){

this.Stud = s;

}

/\* Fakulteto duomenys

    @param tilte - nauja fauklteto pavadinimo reikšmė

    @param studcout - nauja studentu skaiciaus reikšmė

    @param stud - nauja objektu reikšmė\*/

public Faculty(String tilte, int studcout, List Stud){

this.tilte = tilte;

this.studcout = studcout;

this.Stud = Stud;

}

/\* Fakulteto numatytas duomenys\*/

public Faculty(){

this.tilte = "";

this.studcout = -1;

this.Stud = new ArrayList();

}

}

class NameComparator implements Comparator<Student> {

@Override

public int compare(Student a, Student b) {

return a.name.compareToIgnoreCase(b.name);

}

}

/\* Klasė išrinkimo saugoti studentų rikiavimo lauko ir kiekio duomenis

@class TempClass\*/

class TempClass {

public String vardas; // Rikiavimo 1 laukas

public int kiekis; // kiekis

/\* Išrinkimo duomenys

    @param vardas - nauja rikiavimo lauko reikšmė

    @param kiekis - nauja kiekio reikšmė\*/

public TempClass(String vardas, int kiekis) {

this.vardas = vardas;

this.kiekis = kiekis;

}

/\* Išrinkimo numatytas duomenys\*/

public TempClass() {}

}

/\* Klasė pildymo procesas, kuris reikia vykdyti rašymo metodu į bendrą masyvą

@class Writer\*/

class Writer implements CSProcess{

private Faculty P;

private final ChannelOutput out;

private final AltingChannelInput in;

/\* Writer duomenys

    @param out - nauja kanalo išvesčio reikšmė

    @param in - nauja kanalo įvesčio reikšmė\*/

Writer(ChannelOutput out, AltingChannelInput in){

this.out = out;

this.in = in;

}

@Override

public void run(){

P = (Faculty) in.read(); // skaityti obejkto klasę iš Process00

if (P != null){

for (int i = 0; i < P.studcout; i++)

out.write(new TempClass(P.getStudent(i).name,1)); // persiusti į bendrą masyvą(B)

}

out.write(null);

}

}

/\* Klasė naudojimo procesas, kuris reikia vykdyti trinimą metodą iš bendros masyvos

@class Remover\*/

class Remover implements CSProcess{

private List<TempClass> V;

private final ChannelOutput out;

private final AltingChannelInput in;

/\* Remover duomenys

    @param out - nauja kanalo išvesčio reikšmė

    @param in - nauja kanalo įvesčio reikšmė\*/

Remover(ChannelOutput out, AltingChannelInput in){

this.out = out;

this.in = in;

}

@Override

public void run(){

V = (List<TempClass>) in.read(); // skaityti obejkto klase is Process00

if (V != null){

for (int i = 0; i < V.size(); i++)

out.write(V.get(i)); // naudoti is bendros masyvos(B)

}

out.write(null);

} // suprantu, kad salina viena karta ir baigia darba? Taip. Pagal salyga turetu salinimo ciklas vykti daug kartu, kol P baigia deti ir po to dar kol nebelieka ka salinti. Be sito butu 2,2/2,5. Apsispresk kaip pasirinkti, ar taisysi, ar lieka taip

}

/\* Klasė, skirta saugoti pagrindiniam masyvui

@class Remover\*/

class Monitor {

private final List<TempClass> B;

public Monitor() {

this.B = new ArrayList<>();

}

public TempClass get(int i){

return B.get(i);

}

public void remove(int i){

B.remove(i);

}

public void add(TempClass V){

B.add(V);

}

public void add(int i, TempClass V){

B.add(i, V);

}

public int size(){

return B.size();

}

public boolean isEmpty(){

return B.isEmpty();

}

public List<TempClass> B(){

return B;

}

}

/\* Klasė valdytojo saugoti bendrą masyvą(B)

@class ProcessManager\*/

class ProcessManager implements CSProcess {

Monitor B = new Monitor();

private final AltingChannelInput[] removerInput;

private final AltingChannelInput[] writerInput;

private final ChannelOutput mainarrayOutput;

private boolean[] preselect = new boolean[10];

private int finishedW = 0;

private int finishedR = 0;

/\* ProcessManager duomenys

    @param in0 - nauja kanalo įvesčio reikšmė

    @param in0 - nauja kanalo įvesčio reikšmė

@param out - nauja kanalo išvesčio reikšmė\*/

public ProcessManager(One2OneChannel[] in0, One2OneChannel[] in1, ChannelOutput out){

writerInput = new AltingChannelInput[5];

for (int i = 0; i < 5; i++)

writerInput[i] = in0[i].in();

removerInput = new AltingChannelInput[5];

for (int i = 0; i < 5; i++)

removerInput[i] = in1[i].in();

this.mainarrayOutput = out;

}

@Override

public void run(){

Alternative alternative = createGuards();

while(finishedW != 5 || finishedR != 5){

int index = alternative.fairSelect(isAlive());

switch (index){

case 0:

Writer(index); break; // skaityti is pildymo proceso

case 1:

Writer(index); break;

case 2:

Writer(index); break;

case 3:

Writer(index); break;

case 4:

Writer(index); break;

case 5:

Remover(index - 5); break; // skaityti is naudojimo proceso

case 6:

Remover(index - 5); break;

case 7:

Remover(index - 5); break;

case 8:

Remover(index - 5); break;

case 9:

Remover(index - 5); break;

}

}

mainarrayOutput.write(B);

PrintArray1();

}

/\* Grąžina iš bendro masyvo kai po pašalinimą.

    @param V - laikino klasės objektas\*/

public void removeValue(TempClass V) {

for (int i = 0; i < B.size(); i++){

if(B.get(i).vardas.equalsIgnoreCase(V.vardas)){

B.get(i).kiekis--;

if (B.get(i).kiekis == 0)

B.remove(i);

return;

}

}

}

/\* Dėti į bendrą masyvą.

    @param val - laikino klasės objektas\*/

public boolean addValue(TempClass val){

for (TempClass B1 : B.B()){

if (B1.vardas == null ? val.vardas == null : B1.vardas.equals(val.vardas)){

B1.kiekis++;

return true;

}

}

if (B.isEmpty())

B.add(val);

else if (B.get(0).vardas.compareToIgnoreCase(val.vardas) > 0)

B.add(0, val);

else if (B.get(B.size() - 1).vardas.compareToIgnoreCase(val.vardas) < 0)

B.add(B.size(), val);

else {

int i = 0;

while (B.get(i).vardas.compareToIgnoreCase(val.vardas) < 0)

i++;

B.add(i, val);

}

return true;

}

/\* Spausdina bendro masyvo duomenius\*/

public void PrintArray1(){

if (B.isEmpty())

System.out.println("Bendras masyvas tuščias");

else {

System.out.println("Bendras masyvas:");

System.out.println("Vardas kiekis");

for (int i = 0; i < B.size(); i++)

System.out.printf("%-2d) %-10s %-2d %n", i + 1, B.get(i).vardas, B.get(i).kiekis);

}

}

/\* Writer metodas, skaityti iš Writer klasės(papildymo procesas), jei data nelygus null tad elementai dėti į bendrą masyvą(B)\*/

public void Writer(int index){ //java metodu varus iprasta pradeti mazosiomis raidemis

TempClass data;

data = (TempClass) writerInput[index].read();

if (data == null)

finishedW++;

else

addValue(data);

}

/\* Remover metodas, skaityti iš Remover klasės(naudojimo procesas), jei data nelygus null tad elementai trinti iš bendros masyvos(B)\*/

public void Remover(int index){ // atrodo visai gerai, vienAS klausimas, kada baigia salinti?

TempClass data;

data = (TempClass) removerInput[index].read();

if (data != null)

removeValue(data);

else

finishedR++;

}

public Alternative createGuards(){

Guard guards[] = new Guard[10];

for (int i = 0; i < 5; i++){

guards[i] = this.writerInput[i];

guards[i + 5] = this.removerInput[i];

}

Alternative alternative = new Alternative(guards);

return alternative;

}

public boolean[] isAlive(){

int i;

for (i = 0; i < preselect.length / 2; i++)

preselect[i] = (finishedW != 5);

for (int j = i; j < preselect.length; j++)

// preselect[j] = (!preselect[0]);

preselect[j] = (!B.isEmpty() || finishedR != 5);

return preselect;

}

}

/\* Klasė process00 vykdyta per skaitymą iš failo ir spaudinta į failą.

@class ProcessManager\*/

class Process00 implements CSProcess{

private List<Faculty> P = new ArrayList<>();

private List<TempClass>[] V;

Monitor B = new Monitor();

private final ChannelOutput[] Poutput;

private final ChannelOutput[] Voutput;

private final AltingChannelInput mainarrayInput;

private final BufferedWriter out;

private final BufferedReader br;

private boolean check = true;

private boolean check1 = true;

/\* Process00 duomenys

    @param br - nauja failo skaitymo reikšmė

@param out - nauja buferio rašymo reikšmė

@param outP - nauja kanalo išvesčio masyvo reikšmė

    @param outV - nauja kanalo išvesčio masyvo reikšmė

@param in - nauja kanalo įvesčio reikšmė\*/

public Process00(String br, BufferedWriter out, ChannelOutput[] outP, ChannelOutput[] outV, AltingChannelInput in) throws FileNotFoundException, IOException

{

this.br = new BufferedReader(new FileReader(br));

this.out = out;

this.Poutput = outP;

this.Voutput = outV;

this.mainarrayInput = in;

}

@Override

public void run(){

try {

while(check){

if(check1){

ReadData(br);

try {

br.close();

} catch (IOException ex) {

Logger.getLogger(Process00.class.getName()).log(Level.SEVERE, null, ex);

}

ReadDataV();

WriteTableP(out);

WriteTableV(out);

PVtoWriter(); // P ir V masyvams persiusti i pildymo ir naudojimo procesus.

check1 = false;

}else{

WriteTableB(out);

check = false;

}

}

} catch (IOException ex) {

Logger.getLogger(Process00.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\* Skaito duomenis iš failo

@param br - buferio skaitymo vardas\*/

public void ReadData(BufferedReader br) throws IOException {

try {

for (int i = 0; i < 5; i++){

String line = br.readLine();

String fakultetas, vardas;

int studSk, kursas;

double vidurkis;

Scanner ed;

ed = new Scanner(line);

fakultetas = ed.next();

studSk = ed.nextInt();

List<Student> S = new ArrayList<>();

for (int j = 0; j < studSk; j++) {

line = br.readLine();

ed = new Scanner(line);

vardas = ed.next();

kursas = ed.nextInt();

vidurkis = ed.nextDouble();

Student s = new Student(vardas, kursas, vidurkis);

S.add(s);

}

Faculty PP = new Faculty(fakultetas, studSk, S);

P.add(PP);

P.get(i).Sort();

}

} catch (NullPointerException e) {

System.out.print("NullPointerException caught");

System.exit(1);

}

}

public void ReadDataV(){

V = new List[]{new ArrayList(Arrays.asList(

new TempClass("Antanas", 1),

new TempClass("Kazys", 1),

new TempClass("Petras", 1),

new TempClass("Sonata", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Martynas", 1),

new TempClass("Lukas", 1),

new TempClass("Vacys", 1),

new TempClass("Robertas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Mykolas", 1),

new TempClass("Aldona", 1),

new TempClass("Asta", 1),

new TempClass("Viktoras", 1),

new TempClass("Arturas", 1),

new TempClass("Vytas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Zigmas", 1),

new TempClass("Arturas", 1),

new TempClass("Simas", 1)

)),

new ArrayList(Arrays.asList(

new TempClass("Jonas", 1),

new TempClass("Mantas", 1),

new TempClass("Vytas", 1),

new TempClass("Robertas", 1)

))};

}

/\* Spausdina fakulteto duomenis į failą

@param out - failo vardas\*/

public void WriteTableP(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

try{

outFile.println("######## P(n) masyvai ########");

outFile.println("-------------------------------");

for (Faculty P1 : P) {

outFile.println("\*\*\*\*" + P1.tilte + "\*\*\*\*");

outFile.println(" Vardas Kursas Vidurkis");

for (int i = 0; i < P1.studcout; i++) {

Student S = P1.getStudent(i);

outFile.printf("%d) %-10s %-5d %-3.2f %n", i + 1, S.name, S.course, S.grade);

}

}

outFile.println("-------------------------------");

} catch (NullPointerException e1) {

System.out.print("Error during writing");

System.exit(1);

}

}

/\* Spausdina TempClass duomenys į failą

@param out - failo vardas\*/

public void WriteTableV(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

int j = 1;

outFile.println("######## V(m) masyvai ########");

outFile.println("-------------------------------");

for(List<TempClass> T1 : V){

if (T1.isEmpty())

outFile.println("Bendras masyvas tuščias");

else {

outFile.printf("V%d masyvas: %n", j); j++;

outFile.println("Vardas kiekis");

for (int i = 0; i < T1.size(); i++)

outFile.printf("%d) %-10s %-2d %n", i + 1, T1.get(i).vardas, T1.get(i).kiekis);

}

}

outFile.println("-------------------------------");

}

/\* Spausdina bendro masyvo duomenys į failą

@param out - failo vardas\*/

public void WriteTableB(BufferedWriter out) throws IOException {

PrintWriter outFile = new PrintWriter (out);

B = (Monitor) mainarrayInput.read();

outFile.println("######## Po valdymo procesus ########");

outFile.println("-------------------------------");

if (B.isEmpty())

outFile.println("Bendras masyvas tuščias");

else {

outFile.println("Bendras masyvas:");

outFile.println("Vardas kiekis");

for (int i = 0; i < B.size(); i++)

outFile.printf("%-2d) %-10s %-2d %n", i + 1, B.get(i).vardas, B.get(i).kiekis);

}

}

/\* Fakulteto ir TempClass masyvo elementų persiųsti į valdytojo(ProcessManager class) procesą.

@param out - failo vardas\*/

public void PVtoWriter(){

for (int i = 0; i < P.size(); i++) {

Poutput[i].write(P.get(i));

}

for (int i = 0; i < V.length; i++){

Voutput[i].write(V[i]);

}

}

}

public class GuzauskasLD3\_b {

private static final String FileName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_b\\IFF-5-1\_GuzauskasL\_L3b\_dat.txt";

private static final String FileName2 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_b\\IFF-5-1\_GuzauskasL\_L3b\_dat2.txt";

private static final String FileName3 = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_b\\IFF-5-1\_GuzauskasL\_L3b\_dat3.txt";

private static final String PrintName = "C:\\Users\\Lukas\\Documents\\NetBeansProjects\\GuzauskasLD3\_b\\IFF-5-1\_GuzauskasL\_L3b\_rez.txt";

public static One2OneChannel[] P = new One2OneChannel[5];

public static One2OneChannel[] V = new One2OneChannel[5];

public static One2OneChannel[] W = new One2OneChannel[5];

public static One2OneChannel[] R = new One2OneChannel[5];

public static One2OneChannel B = Channel.one2one();

public static void main(String[] args) throws FileNotFoundException, IOException {

BufferedReader br = new BufferedReader(new FileReader(FileName)); // salinimo visu bendras masyvas

BufferedReader br2 = new BufferedReader(new FileReader(FileName2)); // is bendro masyvo nepasalina ne vieno elemento

BufferedReader br3 = new BufferedReader(new FileReader(FileName3)); // is bendro masyvo pasalinima dalis elementu

BufferedWriter wr = new BufferedWriter(new FileWriter(PrintName));

for (int i = 0; i < 5; i++){

P[i] = Channel.one2one();

V[i] = Channel.one2one();

W[i] = Channel.one2one();

R[i] = Channel.one2one();

}

ChannelOutput[] Pin = new ChannelOutput[5];

ChannelOutput[] Vin = new ChannelOutput[5];

for (int i = 0; i < 5; i++){

Pin[i] = P[i].out();

Vin[i] = V[i].out();

}

new Parallel(new CSProcess[]{new Process00(FileName3, wr, Pin, Vin , B.in()),

new Writer(W[0].out(), P[0].in()),

new Writer(W[1].out(), P[1].in()),

new Writer(W[2].out(), P[2].in()),

new Writer(W[3].out(), P[3].in()),

new Writer(W[4].out(), P[4].in()),

new Remover(R[0].out(), V[0].in()),

new Remover(R[1].out(), V[1].in()),

new Remover(R[2].out(), V[2].in()),

new Remover(R[3].out(), V[3].in()),

new Remover(R[4].out(), V[4].in()),

new ProcessManager(W, R, B.out())

}

).run();

try {

wr.close();

} catch (IOException ex) {

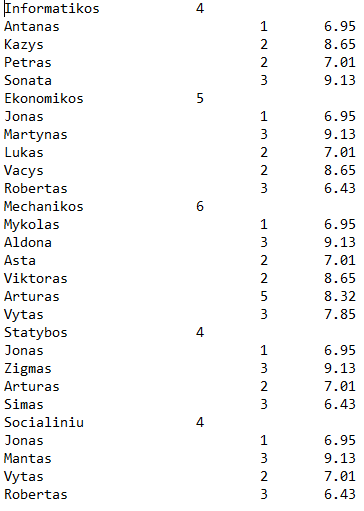
Logger.getLogger(GuzauskasLD3\_b.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

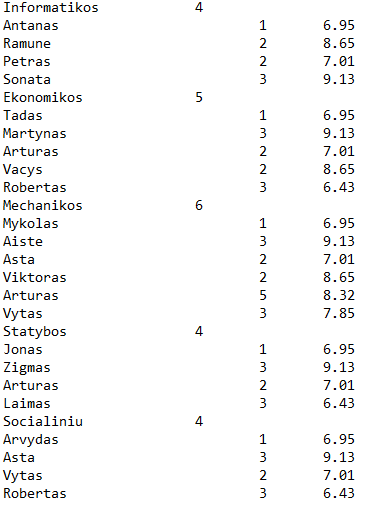
**Duomenų failas(po šalinimo bendras masyvas tuščias):**



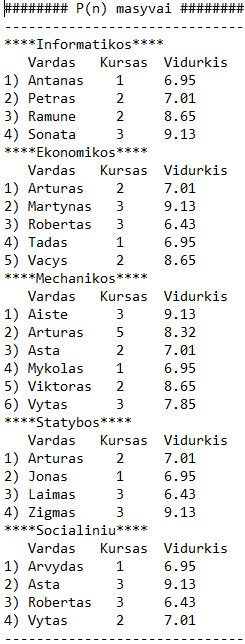
**Duomenų failas(iš bendro masyvo nepašalina nė vieno elemento):**



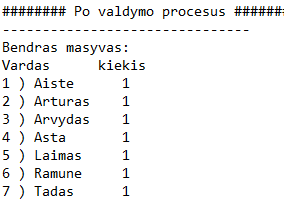
**Duomenų failas(iš bendro masyvo pašalinama dalis elementų):**



**Rezultatų failas:**







1. L4. Lygiagretusis programavimas CUDA

**Užduotis:**

Programos duomenys – pasirinkto tipo struktūrų (bent trys laukai: *string*, *int*, *double*) masyvai *P1(k), P2(k), ..., Pn(k)* atitinkamai (*4 < n < 10, 10 < k < 30*). Masyvai yra surikiuoti pagal vieną iš laukų (pasirinkti pačiam).

Sudaryti naują to paties tipo struktūrų masyvą *P(k)*, kur *Pi* *= P1i* *+ P2i* *+ ... Pni* (skaitiniai laukai sudedami, eilutės sujungiamos).

Spausdinti duomenis ir rezultatus atitinkamai į failus Grupe\_PavardeV\_L4a\_rez.txt, Grupe\_PavardeV\_L4b\_rez.txt.

Programos duomenys surašyti faile Grupe\_PavardeV\_L4.txt (Grupe, PavardeV – jūsų grupė ir pavardė, v.). Abiems programoms naudojamas tas pats duomenų failas.

* + L4a. CUDA gijos

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L4a.cu : Lukas Gužauskas IFF-5/1 gr.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <cuda.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

#include <vector>

#include <algorithm>

using namespace std;

const int THREAD\_COUNT = 5;

const int ARRAY\_SIZE = 15;

const string FileName = "IFF-5-1\_GuzauskasL\_L4dat.txt";

const string printname = "IFF-5-1\_GuzauskasL\_L4a\_rez.txt";

struct Student{

char name[75];

int course;

double average;

};

struct Faculty{

char tilte[15];

int countSk;

     Student Stud2[ARRAY\_SIZE];

};

cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size);

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(string filename, Faculty (& P)[THREAD\_COUNT])

{

    string faculty, title, line;

    int count, j;

    j = 0;

    ifstream fin(filename.c\_str());

    if (!fin) {

        cerr << "Couldn't open file! \n";

    } else{

        while (!fin.eof()){

            fin >> title >> count;

            strcpy(P[j].tilte, title.c\_str());

            P[j].countSk = count;

            Student Stud2[ARRAY\_SIZE];

            for (int i = 0; i < count; i++){

                Student s;

                fin >> s.name >> s.course >> s.average;

                P[j].Stud2[i] = s;

            }

            j++;

        }

        fin.close();

    }

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void WriteData(FILE \* File, Faculty (& P)[THREAD\_COUNT])

{

    for (int i = 0; i < THREAD\_COUNT; i++){

        fprintf(File, " %s \t %d \n", P[i].tilte, P[i].countSk);

        fprintf(File, "---------------------------------------------\n");

        for (int j = 0; j < P[i].countSk; j++) {

            fprintf(File, " %i) %s \t %i \t %g \n", j + 1, P[i].Stud2[j].name, P[i].Stud2[j].course, P[i].Stud2[j].average);

        }

        fprintf(File, "---------------------------------------------\n");

    }

}

void WriteRData(FILE \* File, Student (& S)[ARRAY\_SIZE]){

    fprintf(File, "\n");

    fprintf(File, "Po GPU vykdymą\n");

    fprintf(File, "----------------------------------------------------------\n");

    fprintf(File, " Studento pavadinimas Kiekis Kaina\n");

    fprintf(File, "----------------------------------------------------------\n");

    for (int i = 0; i < ARRAY\_SIZE; i++){

        fprintf(File, " %i) %s\t %i\t %g \n", i + 1, S[i].name, S[i].course, S[i].average);

    }

}

bool sortbyName(Student &a, Student &b) { string c = a.name; string d = b.name; return c < d; }

\_\_global\_\_ void addKernel(Faculty \*P, Student \*S);

\_\_device\_\_ char \*my\_strcat(char \*dest, const char \*src);

\_\_device\_\_ char \* my\_strcpy(char \*dest, const char \*src);

FILE \* File;

int main()

{

    Faculty Pn[THREAD\_COUNT];

    Student r[ARRAY\_SIZE];

    vector<Student> Stud;

    File = freopen(printname.c\_str(), "w", stderr);

    ReadData(FileName, Pn);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        Student s;

        strcpy(s.name, "");

        s.course = 0;

        s.average = 0.0;

        r[i] = s;

    }

    WriteData(File, Pn);

    //Paruosiame GPU

    Faculty \*P;

    Student \*result;

    int i = THREAD\_COUNT \* sizeof(Faculty);

    int j = ARRAY\_SIZE \* sizeof(Student);

    cudaMalloc((void\*\*)&P, i);

    cudaMalloc((void\*\*)&result, j);

    cudaMemcpy(P, Pn, i, cudaMemcpyHostToDevice);

    addKernel<<<1, ARRAY\_SIZE>>>(P, result);

    cudaDeviceSynchronize();

    cudaMemcpy(r, result, j, cudaMemcpyDeviceToHost);

    WriteRData(File, r);

    cudaFree(P);

    cudaFree(result);

    printf("%d \n", r[1].course);

    printf("%d \n", r[2].course);

return 0;

}

\_\_global\_\_ void addKernel(Faculty \*P, Student \*S)

{

int thread = threadIdx.x;

    int j = blockIdx.x \* blockDim.x \* threadIdx.x;

    for (int i = 0; i < THREAD\_COUNT; i++){

        my\_strcat(S[thread].name, P[i].Stud2[thread].name);

        S[thread].course += P[i].Stud2[thread].course;

        S[thread].average += P[i].Stud2[thread].average;

    }

}

\_\_device\_\_ char \* my\_strcpy(char \*dest, const char \*src){

    int i = 0;

    do {

        dest[i] = src[i];

    } while (src[i++] != 0);

    return dest;

}

\_\_device\_\_ char \*my\_strcat(char \*dest, const char \*src){

    int i = 0;

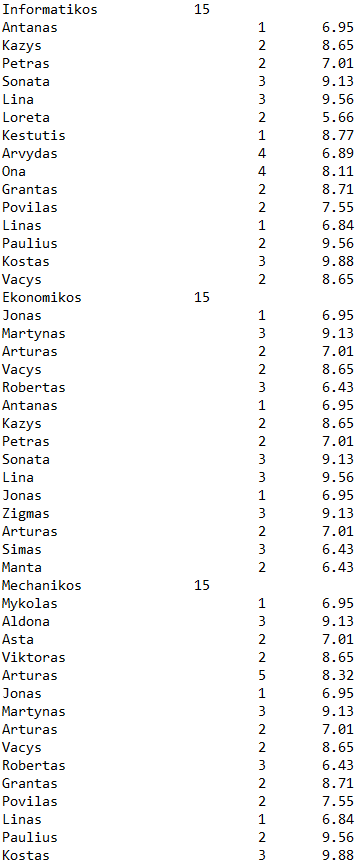
    while(dest[i] != 0) i++;

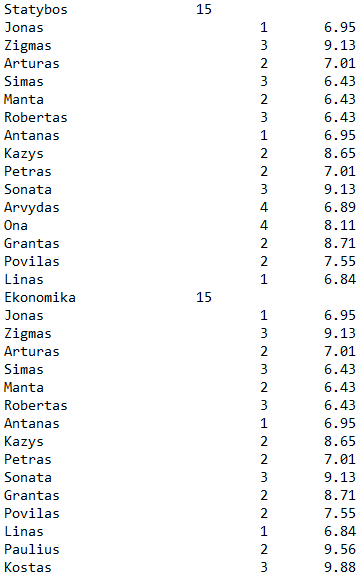
    my\_strcpy(dest + i, src);

    return dest;

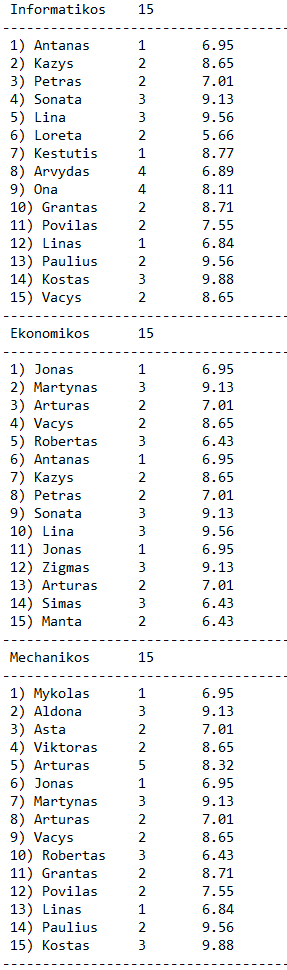
}

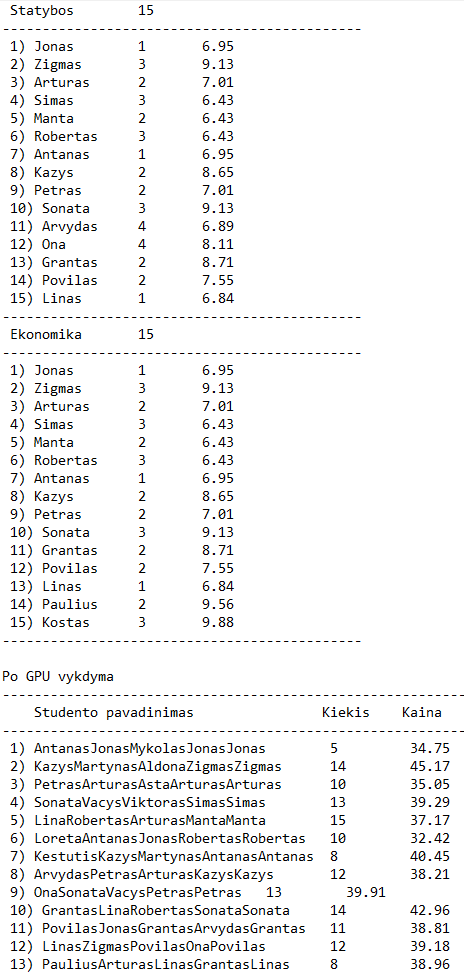
**Duomenų failas:**





**Rezultatų failas:**







* + L4b. Thrust CPU ir GPU vektoriai

**Programų tekstai:**

// IFF-5-1\_GuzauskasL\_L4b.cu : Lukas Gužauskas IFF-5/1 gr.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <cuda.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

#include <vector>

#include <algorithm>

#include <thrust/host\_vector.h>

#include <thrust/device\_vector.h>

#include <thrust/generate.h>

#include <thrust/transform.h>

#include <thrust/functional.h>

#include <thrust/iterator/zip\_iterator.h>

using namespace std;

const int THREAD\_COUNT = 5;

const int ARRAY\_SIZE = 15;

const string FileName = "IFF-5-1\_GuzauskasL\_L4dat.txt";

const string printname = "IFF-5-1\_GuzauskasL\_L4b\_rez.txt";

/\*\*

\* @struct Student - struktura, skirta studento duomenims saugoti

\*/

struct Student{

char name[75];

int course;

double average;

};

/\*\*

\* @struct Faculty - struktura, skirta fakulteto duomenims saugoti

\*/

struct Faculty{

char tilte[15];

int countSk;

     Student Stud2[ARRAY\_SIZE];

};

/\*\*

\* @struct vectorAdd - struktura, skirta operacijos vykdymui

\*/

struct vectorAdd {

    \_\_host\_\_ \_\_device\_\_

        Student operator()(Student stud1, Student stud2){

            Student stud;

            stud.course = stud1.course + stud2.course;

            stud.average = stud1.average + stud2.average;

            return stud;

    }

};

cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size);

/\* Skaito duomenis iš failo

@param File - failo vardas

@param P - objektų vardas\*/

void ReadData(string filename, Faculty (& P)[THREAD\_COUNT])

{

    string faculty, title, line;

    int count, j;

    j = 0;

    ifstream fin(filename.c\_str());

    if (!fin) {

        cerr << "Couldn't open file! \n";

    } else{

        while (!fin.eof()){

            fin >> title >> count;

            strcpy(P[j].tilte, title.c\_str());

            P[j].countSk = count;

            Student Stud2[ARRAY\_SIZE];

            for (int i = 0; i < count; i++){

                Student s;

                fin >> s.name >> s.course >> s.average;

                P[j].Stud2[i] = s;

            }

            j++;

        }

        fin.close();

    }

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param P - objektų vardas\*/

void WriteData(FILE \* File, Faculty (& P)[THREAD\_COUNT])

{

    for (int i = 0; i < THREAD\_COUNT; i++){

        fprintf(File, " %s \t %d \n", P[i].tilte, P[i].countSk);

        fprintf(File, "---------------------------------------------\n");

        for (int j = 0; j < P[i].countSk; j++) {

            fprintf(File, " %i) %s \t %i \t %g \n", j + 1, P[i].Stud2[j].name, P[i].Stud2[j].course, P[i].Stud2[j].average);

        }

        fprintf(File, "---------------------------------------------\n");

    }

}

/\* Spausdina duomenis į failą

@param File - failo vardas

@param H - objektų vardas\*/

void WriteRData(FILE \* File, thrust::host\_vector<Student> H)

{

    fprintf(File, "\n");

    fprintf(File, "Po GPU vykdymą\n");

    fprintf(File, "-------------------------------------------------------------------------------\n");

    fprintf(File, " Studento pavadinimas Kiekis Kaina\n");

    fprintf(File, "-------------------------------------------------------------------------------\n");

    for (int i = 0; i < H.size(); i++){

        fprintf(File, " %i) %-50s\t %i\t %g \n", i + 1, H[i].name, H[i].course, H[i].average);

    }

}

FILE \*File;

/\* Vykdymas eilutės sujungiamos

@param D1 - objektų vardas

@param D2 - objektų vardas

@param D3 - objektų vardas

@param D4 - objektų vardas

@param D5 - objektų vardas\*/

Student Plus(Student D1, Student D2, Student D3, Student D4, Student D5, Student Res){

    Student result;

    strcpy(result.name, D1.name);

    strcat(result.name, D2.name);

    strcat(result.name, D3.name);

    strcat(result.name, D4.name);

    strcat(result.name, D5.name);

    result.course = Res.course;

    result.average = Res.average;

    return result;

}

void addCUDA(FILE \*File, Faculty \*P){

    thrust::device\_vector<Student> Res(ARRAY\_SIZE);

    Student Stud[ARRAY\_SIZE];

    thrust::host\_vector<Student> H1(ARRAY\_SIZE);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        H1[i] = P[0].Stud2[i];

    }

    thrust::device\_vector<Student> D1 = H1;

    thrust::host\_vector<Student> H2(ARRAY\_SIZE);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        H2[i] = P[1].Stud2[i];

    }

    thrust::device\_vector<Student> D2 = H2;

    thrust::host\_vector<Student> H3(ARRAY\_SIZE);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        H3[i] = P[2].Stud2[i];

    }

    thrust::device\_vector<Student> D3 = H3;

    thrust::host\_vector<Student> H4(ARRAY\_SIZE);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        H4[i] = P[3].Stud2[i];

    }

    thrust::device\_vector<Student> D4 = H4;

    thrust::host\_vector<Student> H5(ARRAY\_SIZE);

    for (int i = 0; i < ARRAY\_SIZE; i++){

        H5[i] = P[4].Stud2[i];

    }

    thrust::device\_vector<Student> D5 = H5;

    thrust::device\_vector<Student> D(ARRAY\_SIZE);

    thrust::transform(Res.begin(), Res.end(), D1.begin(), Res.begin(), vectorAdd());

    thrust::transform(Res.begin(), Res.end(), D2.begin(), Res.begin(), vectorAdd());

    thrust::transform(Res.begin(), Res.end(), D3.begin(), Res.begin(), vectorAdd());

    thrust::transform(Res.begin(), Res.end(), D4.begin(), Res.begin(), vectorAdd());

    thrust::transform(Res.begin(), Res.end(), D5.begin(), Res.begin(), vectorAdd());

    for(int i = 0; i < ARRAY\_SIZE; i++){

        Stud[i] = Plus(D1[i], D2[i], D3[i], D4[i], D5[i], Res[i]);

        D[i] = Stud[i];

    }

    thrust::host\_vector<Student> H = D;

    WriteRData(File, H);

}

int main()

{

    Faculty Pn[THREAD\_COUNT];

    File = freopen(printname.c\_str(), "w", stderr);

    ReadData(FileName, Pn);

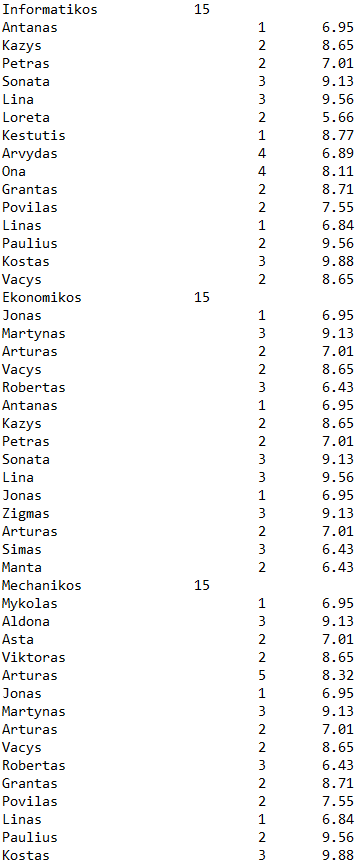
    WriteData(File, Pn);

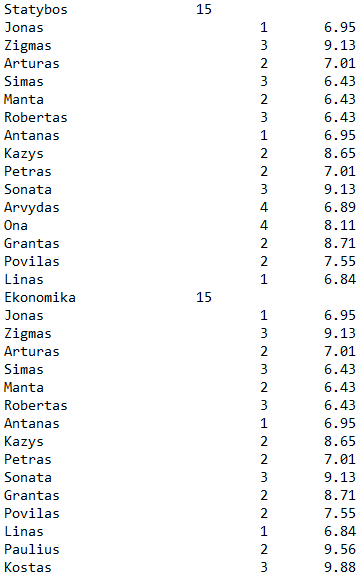
    addCUDA(File, Pn);

return 0;

}

**Duomenų failas:**





**Rezultatų failas:**

