ObjektinisProgramavimas

v3.0

Generated by Doxygen 1.10.0

Chapter 1

ObjektinisProgramavimas

Programa apskaičiuojanti studentų galutinius balus iš pateiktų duomenų. Projekto `dokumentacija`.

1.1 Release'ai

1. V.pradinė: Sukurtas programos karkasas. Naudotojas gali įvesti studentų kiekį, jų duomenis (vardą, pavardę, pažymius) ir ekrane matyti atspausdintus studento duomenis su apskaičiuotu galutiniu balu.
2. V0.1: Programa papildyta taip, kad studentų skaičius ir namų darbų skaičius nėra žinomi iš anksto. Pridėtas dar vienas programos failas (viename faile studentams saugoti naudojame C masyvus, kitame - std::vector konteinerius).
3. V0.2: Programoje atsirado galimybė nuskaityti duomenis iš failo, bei juos išrykiuoti.
4. V0.3: Funkcijos ir jų antraštės perkeltos į atskirus .cpp ir .h failus. Pridėtas išimčių valdymas.
5. V0.4: Pridėta failų generavimo funkcija. Pridėtas studentų rūšiavimas į atskirus konteinerius ir failus, atsižvelgiant į jų galutinius balus. Atlikti du programos spartos tyrimai.
6. V1.0: Atliktas programos testavimas su skirtingais konteineriais (Vector, List ir Deque). Taip pat naudojant skirtingus algoritmus, atliktas studentų skirstymas į dvi grupes testavimas.

1.2 Naudojimosi instrukcija

Norint naudoti 5 arba 6 parinktį (darbą su failas), pirmiausia turite susigeneruoti failus naudodami funkciją generuotiFaila().

1. Paleisti programą
2. Sekti programoje nurodomus žingsnius priklausomai nuo to, kaip jūs norite vykdyti programą.
3. Gauti studentų rezultatus ekrane arba faile (priklausomai nuo to, kokį išvedimo būdą jūs pasirinkote).

1.3 Programos diegimo ir paleidimo instrukcija

1. Privaloma turėti įsidiegus "MinGW" kompiliatorių ir "Make" - automatizavimo įrankį, kuris kuria vykdomąsias programas (Šis įrankis dažniausiai būna automatiškai instaliuotas Linux ir MacOS sistemose). Atsisiųsti MinGW galite čia: MinGW Pamoka, kaip atsisiųsti "Make" Windows naudotojams: Make
2. Atsisiųskite programos šaltinio kodą iš mūsų repozitorijos.
3. Atsidarę terminalą, naviguokite į atsisiųstos programos aplanką.
4. Įvykdykite komandą: make "konteineris" (vietoj "konteineris" įrašykite, su kokio tipo konteineriu norite testuoti programą: Vector, List ar Deque).
5. Tuomet terminale įrašykite ./mainVector, ./mainList arba ./mainDeque, kad paleistumėte norimą programą Linux sistemoje arba mainVector.exe, mainList.exe ar mainDeque.exe Windows sistemoje.

1.4 Testavimo parametrai

CPU: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz RAM: 16GB SSD: Micron NVMe 512GB

1.5 Darbo su vektoriais rezultatai, naudojant 1 rūšiavimo strategiją

Studentu skaicius	Failo generavimo trukme	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Duomenu isvedimo i failus trukme	Viso testo trukme
1000	0.015606	0.003097	0.000917	0.000369	0.006659	0.011042
10000	0.066734	0.021904	0.007164	0.003322	0.047934	0.080324
100000	0.579735	0.194111	0.096998	0.020499	0.384902	0.69651
1000000	5.70716	1.79173	1.33658	0.207118	3.42451	6.75994
10000000	56.1039	17.711	16.2768	1.79619	34.9862	70.7702

1.6 Darbo su list'ais rezultatai, naudojant 1 rūšiavimo strategiją

Studentu skaicius	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme
1000	0.012964	0.000793	0.003106	0.031365
10000	0.079438	0.003847	0.011004	0.145519
100000	0.284524	0.056278	0.10717	0.837474
1000000	3.03823	0.999937	1.05213	9.01883
10000000	29.4128	15.8629	14.6962	101.946

1.7 Darbo su deque'ais rezultatai, naudojant 1 rūšiavimo strategiją

Studentu skaicius	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme
1000	0.004163	0.002462	0.000821	0.017254
10000	0.024412	0.021921	0.006072	0.090725
100000	0.173054	0.348351	0.067596	1.04723
1000000	1.91702	4.24411	0.651649	11.0898
10000000	16.7831	54.2263	15.4799	124.884

1.8 Vektoriuje esanciu studentu rikiavimo rezultatai naudojant 2 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.009835
10000	0.876002
100000	91.3894
1000000	1000+
10000000	10000+

Testuojant faila su 1 000 000 studentu skirstymo laikas toks ilgas, jog tiesiog neverta laukti pabaigos

1.9 Vektoriuje esanciu studentu rikiavimo rezultatai naudojant 3 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.000356
10000	0.003718
100000	0.02288
1000000	0.236657
10000000	2.38154

1.10 List'e esanciu studentu rikiavimo rezultatai naudojant 2 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.000423
10000	0.007395
100000	0.046986
1000000	0.483659
10000000	4.88548

Neapsakomai greičiau, nei naudojant 2 strategiją su vektoriais

1.11 List'e esanciu studentu rikiavimo rezultatai naudojant 3 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.00587
10000	0.008309
100000	0.079064
1000000	0.826309
10000000	8.46289

Programa vykdoma lėčiau, nei naudojant 2 strategiją

1.12 Deque esanciu studentu rikiavimo rezultatai naudojant 2 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.006382
10000	0.562682
100000	57.5754
1000000	1000+
10000000	10000+

Testuojant faila su 1 000 000 studentu skirstymo laikas toks ilgas, jog tiesiog neverta laukti pabaigos

1.13 Deque esanciu studentu rikiavimo rezultatai naudojant 3 strategija

Studentu skaicius	Duomenu skirstymo trukme
1000	0.001138
10000	0.010341
100000	0.099007
1000000	1.18494
10000000	30.9919

Vykdymo laikas ženkliai sutrumpėja, lyginant su 2 strategija

1.14 Programos veikimo laikų palyginimas naudojant Class ir Struct

1.14.1 Struct

Studentu skaicius	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme
100000	0.184327	0.067112	0.023777	0.624784
1000000	1.74853	0.918207	0.24031	5.9056

1.14.2 Class

Studentu skaicius	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme
100000	0.201868	0.119731	0.036706	0.601262
1000000	1.84009	1.63147	0.372546	6.44111

1.15 Programos veikimo laikų palyginimas naudojant optimizavimo flag'us

1.15.1 Studentų kiekis: 100000

	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme	.exe failo dydis
Struct -O1	0.140682	0.013065	0.012014	0.393933	3233 kB
Struct -O2	0.138285	0.013137	0.011664	0.403994	3216 kB
Struct -O3	0.137578	0.011896	0.011614	0.400341	3206 kB
Class -O1	0.146403	0.061518	0.024968	0.492828	3225 kB
Class -O2	0.15007	0.050086	0.021801	0.461427	3208 kB
Class -O3	0.152853	0.048282	0.022229	0.462433	3205 kB

1.15.2 Studentų kiekis: 1000000

	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Viso testo trukme
Struct -O1	1.28276	0.186754	0.165579	4.07632
Struct -O2	1.22172	0.184022	0.126522	4.02027
Struct -O3	1.26782	0.169268	0.12368	4.10841
Class -O1	1.34906	0.709025	0.235499	4.71324
Class -O2	1.31023	0.698598	0.224778	4.59237
Class -O3	1.35358	0.711891	0.241283	4.60437

1.16 Rule Of Five Pritaikymas

1.16.1 Testavimo funkcijos rezultatai

1.16.2 Pridėtų dalykų aprašas

1. Copy konstruktorius - naujo "studentas" objekto kūrimo metu mes nukopijuojame visus duomenis į naują objektą iš kažkurio seno objekto.
2. Copy Assignment operatorius - naudodami lygybės ženklą mes galime nukopijuoti visus vieno objekto duomenis kitam objektui.
3. Move konstruktorius - naujo "studentas" objekto kūrimo metu mes perkeliame visus duomenis iš senesnio objekto į naujai kuriamą (senasis objektas lieka galioti, bet jo būsena nėra tiksliai žinoma).
4. Move Assignment operatorius - naudodami lygybės ženklą ir "move" raktažodį, mes galime jau sukurtam objektui perkelti visus duomenis iš seno objekto (senasis objektas lieka galioti, bet jo būsena nėra tiksliai žinoma).

1.16.3 Perdengtų metodų aprašas

1.16.3.1 Įvestis

1. Rankinis būdas: programoje parašius, tarkim, cin >> studentas, vartotojas turės galimybę ranka įvesti visus objekto duomenis, jei parinktis (gauta programos pradžioje, bus lygi 1).
2. Automatinis būdas: jei parinktis bus lygi 2 arba 3, tuomet vartotojas galės įvesti tik vardą ir pavardę arba apskritai visi duomenys bus generuojami. (Programoje šios įvesties užrašymas taip pat atrodo cin >> studentas).
3. Nuskaitymas iš failo: liko nepakitęs.

1.16.3.2 Išvestis

1. Į ekraną: panaudojant operatorių <<, tarkim cout << studentas, visi "studentas" klasės duomenys bus išvesti į ekraną.
2. Į failą: panaudojant operatorių <<, tarkim vargsiukai << studentas, visi "studentas" klasės duomenys bus išvesti į failą "vargsiukai".

1.17 Paveldėjimo pritaikymas

1.17.1 Pridėtų dalykų aprašas

1. Nauja klasė "zmogus", iš kurios išvedame mūsų senąją klasę "studentas".
2. Į naująją klasę "zmogus" iš klasės "studentas" mes perkėlėme kintamuosius "vardas" ir "pavarde", taip pat naujoji klasė turi konstruktorių, destruktorių, get'erius bei keturias virtualias funkcijas.

1.18 Savo Vector klasės implementacija

1.18.1 Penkių funkcijų aprašymai

1. Operatorius == : Pirmiausia ši bool funkcija tikrina ar abejose lygybės pusėse esančių vektorių dydžiai ir talpos yra vienodi. Jei ne, tuomet funkcija iškart grąžina reikšmę false, reiškia vektoriai nėra lygūs. Priešingu atveju vyksta ciklas, kuris lygina abiejų vektorių elementus, esančius tose pačiose vietose, jei kažkurie du elementai nesutampa, funkcija grąžina reikšmę false. Jei ciklas sėkmingai užbaigiamas, grąžinama reikšmė true.
2. push_back: Ši funkcija pirmiausia patikrina, ar vektoriaus dydis yra toks pat, kokia yra ir talpa, jei taip, tuomet talpa padvigubinama. Tuomet yra sukonstruojamas naujas pateiktoje vietoje, o tiksliau - vektoriaus gale.
3. reserve: Ši funkcija atlieka savo darbą tik tuomet, jei perduotas argumentas n yra didesnis už dabartinę vektoriaus talpą. Jei taip ir yra, tuomet funkcija pirmiausia paskiria atminties, kuri gali sutalpinti n elementų. Paskui užvedamas ciklas, einantis per jau egzistuojančio vektoriaus dydį. Kiekvienas egzistuojančio vektoriaus elementas yra perkeliamas į naujai priskirtą atmintį ir ištrinamas iš pirminio vektoriaus. Tuomet yra atlaisvinama visa pirminio vektoriaus atmintis, bei atnaujinami vektoriaus dydžio ir talpos kintamieji.
4. shrink_to_fit: Funkcija vykdoma tik jei vektoriaus dydis yra mažesnis nei jo talpa. Jei taip ir yra, tuomet vektoriaus talpai yra priskiriama vektoriaus dydžio reikšmė.
5. pop_back: Funkcija vykdoma, jei vektoriaus dydis yra didesnis už nulį. Tuomet yra atlaisvinama paskutinio vektoriaus elemento vieta, ištrinant tą elementą iš atminties.

1.18.2 std::vector vs. Vector konteineriu uzpildymo spartos palyginimas

El. kiekis	std::vector pildymo laikas	Vector pildymo laikas
10000	69 mikrosekundes	46 mikrosekundes
100000	945 mikrosekundes	773 mikrosekundes
1000000	6816 mikrosekundes	5803 mikrosekundes
10000000	59936 mikrosekundes	60396 mikrosekundes
100000000	575138 mikrosekundes	565729 mikrosekundes

1.18.3 std::vector vs. Vector konteineriu darbo su studentu failais spartos palyginimas

	Duomenu nuskaitymo trukme	Duomenu rikiavimo trukme	Duomenu skirstymo trukme	Isvedimo trukme	Viso testo trukme
Vector 100000	0.173407	0.071158	0.029485	0.243146	0.517196
Vector 1000000	1.62093	0.97565	0.298364	2.4385	5.33345
Vector 10000000	17.2497	12.6539	2.98685	25.5777	58.4681
std::vector 100000	0.209535	0.101484	0.032385	0.2511	0.594504
std::vector 1000000	1.89343	1.34222	0.326025	2.39114	5.95281
std::vector 10000000	19.1226	16.9183	3.39667	26.2336	65.6712

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

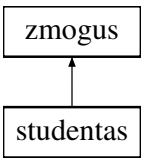
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 studentas Class Reference

Inheritance diagram for studentas:



Public Member Functions

- **studentas** (std::string vardas="", std::string pavarde="", std::Vector< int > nd={}, int egz=0)
- **studentas** (std::istream &is)
- **studentas** (const studentas &other)
- **studentas** (studentas &&other)
- studentas & **operator=** (const studentas &other)
- studentas & **operator=** (studentas &&other)
- double **galutinis** () const
- std::string **getVardas** () const
- std::string **getPavarde** () const
- int **getEgz** () const
- const std::Vector< int > & **getNd** () const
- void **clearNd** ()
- int **gautiPaskutiniPazymi** ()
- void **generuotiEgzPazymi** ()
- void **generuotiNdPazymi** ()
- void **baloSkaiciavimas** (std::string)
- void didziosiosVardas ()
- void **didziosiosPavarde** ()
- void generuotiVarda (int i) override
- void generuotiPavarde (int i) override

Public Member Functions inherited from zmogus

- std::string **vardas** () const
- std::string **pavarde** () const

Friends

- bool **palygintiMazejant** (const studentas &, const studentas &)
- bool **palygintiDidejant** (const studentas &, const studentas &)
- std::ostream & **operator**<< (std::ostream &, const studentas &)
- std::istream & **operator**>> (std::istream &, studentas &)
- bool **operator==** (const studentas &, const studentas &)
- bool **operator==** (const std::string &, const std::string &)

Additional Inherited Members

Protected Member Functions inherited from zmogus

- **zmogus** (std::string vardas="", std::string pavarde="")

Protected Attributes inherited from zmogus

- std::string **vardas_**
- std::string **pavarde_**

5.1.1 Member Function Documentation

5.1.1.1 didziosiosVardas()

```
void studentas::didziosiosVardas ( )  [inline], [virtual]
```

Implements zmogus.

5.1.1.2 generuotiPavarde()

```
void studentas::generuotiPavarde (
            int i )  [inline], [override], [virtual]
```

Reimplemented from zmogus.

5.1.1.3 generuotiVarda()

```
void studentas::generuotiVarda (
            int i )  [inline], [override], [virtual]
```

Reimplemented from zmogus.

The documentation for this class was generated from the following files:

- funkcijos.h
- funkcijos.cpp

5.2 std::Vector< T, Allocator > Class Template Reference

Public Types

- using **value_type** = T
- using **allocator_type** = Allocator
- using **pointer** = typename allocator_traits<Allocator>::pointer
- using **const_pointer** = typename allocator_traits<Allocator>::const_pointer
- using **reference** = value_type&
- using **const_reference** = const value_type&
- using **size_type** = size_t
- using **difference_type** = ptrdiff_t
- using **iterator** = pointer
- using **const_iterator** = const_pointer
- using **reverse_iterator** = std::reverse_iterator<iterator>
- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>

Public Member Functions

- **Vector** (const Allocator &) noexcept
- **Vector** (size_type n, const Allocator &=Allocator())
- **Vector** (size_type n, const T &value, const Allocator &=Allocator())
- **Vector** (initializer_list< T > il, const Allocator &=Allocator())
- **Vector** (const Vector &x)
- **Vector** (Vector &&x) noexcept
- Vector & **operator=** (const Vector &x)
- Vector & **operator=** (Vector &&x) noexcept(allocator_traits< Allocator >::propagate_on_container_move_↩ assignment::value||allocator_traits< Allocator >::is_always_equal::value)
- bool **operator==** (const Vector &x) const
- Vector & **operator=** (initializer_list< T > il)
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () const noexcept
- const_iterator **cbegin** () const noexcept
- const_iterator **cend** () const noexcept
- const_reverse_iterator **crbegin** () const noexcept
- const_reverse_iterator **crend** () const noexcept
- bool **empty** () const noexcept
- size_type **size** () const noexcept
- size_type **max_size** () const noexcept
- size_type **capacity** () const noexcept
- void **resize** (size_type new_size)
- void **resize** (size_type newsz, const T &c)
- void **reserve** (size_type n)
- void **shrink_to_fit** ()
- reference **operator[ ]** (size_type n)
- const_reference **operator[ ]** (size_type n) const
- const_reference **at** (size_type n) const

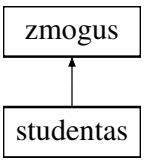
- reference **at** (size_type n)
- reference **front** ()
- const_reference **front** () const
- reference **back** ()
- const_reference **back** () const
- T ∗ **data** () noexcept
- const T ∗ **data** () const noexcept
- template<class... Args>
  reference **emplace_back** (Args &&... args)
- void **push_back** (const T &x)
- void **push_back** (T &&x)
- void **pop_back** ()
- template<class... Args>
  iterator **emplace** (const_iterator position, Args &&... args)
- iterator **insert** (const_iterator position, const T &x)
- iterator **insert** (const_iterator position, T &&x)
- iterator **insert** (const_iterator position, size_type n, const value_type &x)
- template<class InputIt , typename = std::_RequireInputIter<InputIt>>
  iterator **insert** (const_iterator position, InputIt first, InputIt last)
- iterator **insert** (const_iterator position, initializer_list< T > il)
- iterator **erase** (const_iterator position)
- iterator **erase** (const_iterator first, const_iterator last)
- void **swap** (Vector &other) noexcept(allocator_traits< Allocator >::propagate_on_container_swap↩
  ::value||allocator_traits< Allocator >::is_always_equal::value)
- void **clear** () noexcept

The documentation for this class was generated from the following file:

- vector.h

5.3 zmogus Class Reference

Inheritance diagram for zmogus:



Public Member Functions

- std::string **vardas** () const
- std::string **pavarde** () const
- virtual void **didziosiosVardas** ()=0
- virtual void **generuotiVarda** (int i)
- virtual void **generuotiPavarde** (int i)

Protected Member Functions

- **zmogus** (std::string vardas="", std::string pavarde="")

Protected Attributes

- std::string **vardas_**
- std::string **pavarde_**

The documentation for this class was generated from the following file:

- funkcijos.h

Chapter 6

File Documentation

6.1 funkcijos.h

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003
00004 #include <iostream>
00005 #include <numeric>
00006 #include <functional>
00007 #include <iomanip>
00008 #include <string>
00009 #include <limits>
00010 #include <cmath>
00011 #include <random>
00012 #include <ctime>
00013 #include <fstream>
00014 #include <sstream>
00015 #include <algorithm>
00016 #include <chrono>
00017 #include <list>
00018 #include <deque>
00019 #include <utility>
00020 #include "vector.h"
00021
00022 using namespace std::chrono;
00023
00024 extern std::string skaiciavimoBudas;
00025 extern int pazymiuKiekis, parinktis, papildymas, k, i, randomPazymiuKiekis;
00026
00027 // Zmogaus duomenis sauganti klase
00028 class zmogus
00029 {
00030     protected:
00031         std::string vardas_, pavarde_; // Klases kintamieji: zmogaus vardas ir pavarde
00032     protected:
00033         zmogus(std::string vardas = "", std::string pavarde = "") : vardas_(vardas), pavarde_(pavarde)
    {} // Default konstruktorius
00034         ~zmogus() { vardas_.clear(), pavarde_.clear(); } // Destruktorius
00035     public:
00036         inline std::string vardas() const { return vardas_; }    // get'eriai, inline
00037         inline std::string pavarde() const { return pavarde_; }  // get'eriai, inline
00038         virtual void didziosiosVardas() = 0; // Visiskai virtuali funkcija
00039         virtual void generuotiVarda(int i) // Funkcija, skirta generuoti zmogaus vardui
00040         {
00041             std::string vardas;
00042             vardas = "Vardas" + std::to_string(i + 1);
00043             vardas_ = vardas;
00044         }
00045         virtual void generuotiPavarde(int i) // Funkcija, skirta generuoti zmogaus pavardei
00046         {
00047             std::string pavarde;
00048             pavarde = "Pavarde" + std::to_string(i + 1);
00049             pavarde_ = pavarde;
00050         }
00051 };
00052
00053 // Studento duomenis sauganti klase
00054 class studentas : public zmogus
00055 {
00056     private:
00057         std::Vector<int> nd_; // Studento namu darbu pazymiu vektorius
```

```
00058         int egz_; // Studento egzamino pazymys
00059         double vidurkis_, mediana_, galutinis_; // Studento pazymiu vidurkis, mediana ir galutinis
      balas
00060     public:
00061         studentas(std::string vardas = "", std::string pavarde = "", std::Vector<int> nd = {}, int egz
      = 0) : zmogus(vardas, pavarde), nd_(nd), egz_(egz) {} // Default konstruktorius
00062         ~studentas() { clearNd(); } // destruktorius
00063         studentas(std::istream& is); // Konstruktorius su nuoroda i istream objekta, kaip parametru
00064         studentas(const studentas& other) : // Copy konstruktorius
00065             zmogus(other.vardas_, other.pavarde_),
00066             nd_(other.nd_),
00067             egz_(other.egz_),
00068             vidurkis_(other.vidurkis_),
00069             mediana_(other.mediana_),
00070             galutinis_(other.galutinis_) {}
00071         studentas(studentas&& other) : // Move konstruktorius
00072             zmogus(other.vardas_, other.pavarde_),
00073             nd_(std::move(other.nd_)),
00074             egz_(other.egz_),
00075             vidurkis_(other.vidurkis_),
00076             mediana_(other.mediana_),
00077             galutinis_(other.galutinis_)
00078             {
00079                 other.vardas_ = "";
00080                 other.pavarde_ = "";
00081                 other.nd_ = {};
00082                 other.egz_ = 0;
00083             }
00084         studentas& operator=(const studentas& other) // Copy assignment operatorius
00085         {
00086             if (&other == this) return *this;
00087             vardas_ = other.vardas_;
00088             pavarde_ = other.pavarde_;
00089             nd_ = other.nd_;
00090             egz_ = other.egz_;
00091             vidurkis_ = other.vidurkis_;
00092             mediana_ = other.mediana_;
00093             galutinis_ = other.galutinis_;
00094             return *this;
00095         }
00096         studentas& operator=(studentas&& other) // Move assignment operatorius
00097         {
00098             if (&other == this) return *this;
00099             vardas_ = other.vardas_;
00100             pavarde_ = other.pavarde_;
00101             nd_ = std::move(other.nd_);
00102             egz_ = other.egz_;
00103             vidurkis_ = other.vidurkis_;
00104             mediana_ = other.mediana_;
00105             galutinis_ = other.galutinis_;
00106             other.vardas_ = "";
00107             other.pavarde_ = "";
00108             other.nd_ = {};
00109             other.egz_ = 0;
00110             return *this;
00111         }
00112         double galutinis() const { return galutinis_; } // Galutinio balo get'eris
00113         std::string getVardas() const { return vardas_; } // Vardo get'eris
00114         std::string getPavarde() const { return pavarde_; } // Pavardes get'eris
00115         int getEgz() const { return egz_; } // Egzamino pazymio get'eris
00116         const std::Vector<int>& getNd() const { return nd_; } // Namu darbu pazymiu vektoriaus
      get'eris
00117         void clearNd() { nd_.clear(); } // Funkcija, isvalanti namu darbu pazymiu vektoriu
00118         int gautiPaskutiniPazymi();
00119         void generuotiEgzPazymi();
00120         void generuotiNdPazymi();
00121         void baloSkaiciavimas(std::string);
00122         void didziosiosVardas() { for(char &c : vardas_) c = toupper(c); } // Funkcija, skirta visas
      vardo raides paversti i didziasias
00123         void didziosiosPavarde() { for(char &c : pavarde_) c = toupper(c); } // Funkcija, skirta visas
      pavardes raides paversti i didziasias
00124         void generuotiVarda(int i) override { zmogus::generuotiVarda(i); } // Funkcija, skirta
      generuoti varda
00125         void generuotiPavarde(int i) override { zmogus::generuotiPavarde(i); } // Funkcija, skirta
      generuoti pavarde
00126         friend bool palygintiMazejant(const studentas&, const studentas&);
00127         friend bool palygintiDidejant(const studentas&, const studentas&);
00128         friend std::ostream& operator«(std::ostream&, const studentas&);
00129         friend std::istream& operator»(std::istream&, studentas&);
00130         friend bool operator==(const studentas&, const studentas&);
00131         friend bool operator==(const std::string&, const std::string&);
00132 };
00133
00134 int generuotiPazymi();
00135 std::string didziosios(std::string&);
00136 bool tikRaides(std::string);
00137 int tarpuSkaicius(std::string);
```

```
00138 void printHeader(std::ostream&);
00139 void testas(studentas&);
00140 void generuotiFaila(int, int, std::string);
00141 template <typename Container>
00142 void failoSkaitymas(std::ifstream&, Container&);
00143 void strategija3(std::Vector<studentas>&, std::Vector<studentas>&);
00144 void rikiuotiDidejant(std::Vector<studentas>&);
00145 void rikiuotiMazejant(std::Vector<studentas>&);
00146
00147 #endif
```

6.2 vector.h

```
00001
00002 #ifndef VECTOR_H
00003 #define VECTOR_H
00004
00005 #include <memory>
00006 #include <iterator>
00007 #include <algorithm>
00008 #include <initializer_list>
00009 #include <stdexcept>
00010 #include <utility>
00011
00012 namespace std
00013 {
00014 template<class T, class Allocator = allocator<T»
00015 class Vector
00016 {
00017     typename allocator_traits<Allocator>::pointer elem;
00018     size_t sz;
00019     size_t cap;
00020     Allocator alloc;
00021 public:
00022     // Member types
00023     using value_type             = T;
00024     using allocator_type         = Allocator;
00025     using pointer                = typename allocator_traits<Allocator>::pointer;
00026     using const_pointer          = typename allocator_traits<Allocator>::const_pointer;
00027     using reference              = value_type&;
00028     using const_reference        = const value_type&;
00029     using size_type              = size_t;
00030     using difference_type        = ptrdiff_t;
00031     using iterator               = pointer;
00032     using const_iterator         = const_pointer;
00033     using reverse_iterator       = std::reverse_iterator<iterator>;
00034     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00035
00036     // Konstruktoriai
00037     Vector() noexcept(noexcept(Allocator())) : Vector(Allocator()) {}
00038     Vector(const Allocator&) noexcept : elem(nullptr), sz(0), cap(0), alloc(Allocator())
00039     {
00040         reserve(1);
00041     };
00042     Vector(size_type n, const Allocator& = Allocator()) : elem(nullptr), sz(0), cap(0),
      alloc(Allocator())
00043     {
00044         resize(n);
00045     };
00046     Vector(size_type n, const T& value, const Allocator& = Allocator()): elem(nullptr), sz(0), cap(0),
      alloc(Allocator())
00047     {
00048         resize(n, value);
00049     };
00050
00051     // Initializer list'as
00052     Vector(initializer_list<T> il, const Allocator& = Allocator()) : elem(nullptr), sz(0), cap(0),
      alloc(Allocator())
00053     {
00054         insert(begin(), il);
00055     }
00056
00057     // Destruktorius
00058     ~Vector()
00059     {
00060         if(elem)
00061             alloc.deallocate(elem, cap);
00062     };
00063
00064     // Copy konstruktorius
00065     Vector(const Vector& x)
00066         : elem(nullptr), sz(0),
00067         cap(0),
```

```
00068         alloc(allocator_traits<Allocator>::select_on_container_copy_construction(x.alloc))
00069     {
00070         reserve(x.sz);
00071         for (size_type i = 0; i < x.sz; ++i)
00072         {
00073             emplace_back(x.elem[i]);
00074         }
00075     }
00076 
00077     // Move konstruktorius
00078     Vector(Vector&& x) noexcept
00079         : elem(std::exchange(x.elem, nullptr)),
00080         sz(std::exchange(x.sz, 0)),
00081         cap(std::exchange(x.cap, 0)),
00082         alloc(std::move(x.alloc)) {}
00083 
00084     // Copy assignment operatorius
00085     Vector& operator=(const Vector& x)
00086     {
00087         if (this != &x)
00088         {
00089             if (allocator_traits<Allocator>::propagate_on_container_copy_assignment::value && alloc !=
      x.alloc)
00090             {
00091                 if (elem)
00092                 {
00093                     clear();
00094                     alloc.deallocate(elem, cap);
00095                 }
00096                 alloc = x.alloc;
00097                 elem = nullptr;
00098                 cap = 0;
00099             }
00100             reserve(x.sz);
00101             if (x.sz <= sz)
00102             {
00103                 std::copy(x.elem, x.elem + x.sz, elem);
00104                 for (size_type i = x.sz; i < sz; ++i)
00105                 {
00106                     alloc.destroy(&elem[i]);
00107                 }
00108             }
00109             else
00110             {
00111                 for (size_type i = 0; i < sz; ++i)
00112                 {
00113                     alloc.construct(&elem[i], x.elem[i]);
00114                 }
00115                 for (size_type i = sz; i < x.sz; ++i)
00116                 {
00117                     alloc.construct(&elem[i], x.elem[i]);
00118                 }
00119             }
00120             sz = x.sz;
00121         }
00122         return *this;
00123     }
00124 
00125     // Move assignment operatorius
00126     Vector& operator=(Vector&& x)
00127       noexcept(
00128         allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
00129         allocator_traits<Allocator>::is_always_equal::value
00130       )
00131       {
00132         if (this != &x)
00133         {
00134             if (allocator_traits<Allocator>::propagate_on_container_move_assignment::value)
00135             {
00136                 if (elem)
00137                 {
00138                     clear();
00139                     alloc.deallocate(elem, cap);
00140                 }
00141                 alloc = std::move(x.alloc);
00142                 elem = std::exchange(x.elem, nullptr);
00143                 sz = std::exchange(x.sz, 0);
00144                 cap = std::exchange(x.cap, 0);
00145             }
00146             else
00147             {
00148                 if (elem)
00149                 {
00150                     clear();
00151                     alloc.deallocate(elem, cap);
00152                 }
00153 
```

```
00154                 elem = std::exchange(x.elem, nullptr);
00155                 sz = std::exchange(x.sz, 0);
00156                 cap = std::exchange(x.cap, 0);
00157             }
00158         }
00159         return *this;
00160     }
00161
00162     // Operatorius, tikrinantis, ar du vektoriai yra lygus
00163     bool operator==(const Vector& x) const
00164     {
00165         if (sz != x.sz || cap != x.cap)
00166         {
00167             return false;
00168         }
00169         for (size_type i = 0; i < sz; ++i)
00170         {
00171             if (elem[i] != x.elem[i])
00172             {
00173                 return false;
00174             }
00175         }
00176         return true;
00177     }
00178
00179     // Priskyrimo operatorius su initializer list'u
00180     Vector& operator=(initializer_list<T> il)
00181     {
00182         clear();
00183         reserve(il.size());
00184         for (auto& elem : il)
00185         {
00186             emplace_back(elem);
00187         }
00188         return *this;
00189     }
00190
00191     // Iteratoriai
00192     iterator begin() noexcept
00193     {
00194         return elem;
00195     }
00196
00197     const_iterator begin() const noexcept
00198     {
00199         return elem;
00200     }
00201
00202     iterator end() noexcept
00203     {
00204         return elem + sz;
00205     }
00206
00207     const_iterator end() const noexcept
00208     {
00209         return elem + sz;
00210     }
00211
00212     reverse_iterator rbegin() noexcept
00213     {
00214         return reverse_iterator(end());
00215     }
00216
00217     const_reverse_iterator rbegin() const noexcept
00218     {
00219         return const_reverse_iterator(end());
00220     }
00221
00222     reverse_iterator rend() noexcept
00223     {
00224         return reverse_iterator(begin());
00225     }
00226
00227     const_reverse_iterator rend() const noexcept
00228     {
00229         return const_reverse_iterator(begin());
00230     }
00231
00232     const_iterator cbegin() const noexcept
00233     {
00234         return elem;
00235     }
00236
00237     const_iterator cend() const noexcept
00238     {
00239         return elem + sz;
00240     }
```

```
00241
00242     const_reverse_iterator crbegin() const noexcept
00243     {
00244         return const_reverse_iterator(end());
00245     }
00246
00247     const_reverse_iterator crend() const noexcept
00248     {
00249         return const_reverse_iterator(begin());
00250     }
00251
00252     // Funkcija, nustatanti vektoriaus dydi i nuli
00253     bool empty() const noexcept
00254     {
00255         return sz==0;
00256     }
00257
00258     // Funkcija, grazinanti vektoriaus dydi
00259     size_type size() const noexcept
00260     {
00261         return sz;
00262     }
00263
00264     // Funkcija, grazinanti maksimalu galima vektoriaus elementu kieki
00265     size_type max_size() const noexcept
00266     {
00267         return alloc.max_size();
00268     }
00269
00270     // Funkcija, grazinanti vektoriaus talpa
00271     size_type capacity() const noexcept
00272     {
00273         return cap;
00274     }
00275
00276     // Funkcija, pakeicianti vektoriaus dydi
00277     void resize(size_type new_size)
00278     {
00279         if (new_size > cap)
00280         {
00281             reserve(new_size);
00282         }
00283
00284         if (new_size > sz)
00285         {
00286             for (size_type i = sz; i < new_size; ++i)
00287             {
00288                 alloc.construct(&elem[i]);
00289             }
00290         }
00291         else
00292         {
00293             for (size_type i = new_size; i < sz; ++i)
00294             {
00295                 alloc.destroy(&elem[i]);
00296             }
00297         }
00298
00299         sz = new_size;
00300     }
00301
00302     // Funkcija, pakeicianti vektoriaus dydi ir priskirianti naujiems elementams tam tikra reiksme
00303     void resize(size_type newsz, const T& c)
00304     {
00305         if (newsz > cap)
00306         {
00307             reserve(newsz);
00308         }
00309
00310         if (newsz > sz)
00311         {
00312             for (size_type i = sz; i < newsz; ++i)
00313             {
00314                 alloc.construct(&elem[i], c);
00315             }
00316         }
00317         else
00318         {
00319             for (size_type i = newsz; i < sz; ++i)
00320             {
00321                 alloc.destroy(&elem[i]);
00322             }
00323         }
00324
00325         sz = newsz;
00326     }
00327
```

```
00328     // Funkcija, keicianti vektoriaus talpa
00329     void reserve(size_type n)
00330     {
00331         if (n > cap)
00332         {
00333             pointer new_data = alloc.allocate(n);
00334             for (size_type i = 0; i < sz; ++i)
00335             {
00336                 alloc.construct(&new_data[i], std::move(elem[i]));
00337                 alloc.destroy(&elem[i]);
00338             }
00339             if (elem)
00340             {
00341                 alloc.deallocate(elem, cap);
00342             }
00343             elem = new_data;
00344             cap = n;
00345         }
00346     }
00347
00348     // Funkcija, pakeicianti vektoriaus talpos reiksme i lygia vektoriaus dydziui
00349     void shrink_to_fit()
00350     {
00351         if (sz < cap)
00352         {
00353             pointer new_data = alloc.allocate(sz);
00354             for (size_type i = 0; i < sz; ++i)
00355             {
00356                 alloc.construct(&new_data[i], std::move(elem[i]));
00357                 alloc.destroy(&elem[i]);
00358             }
00359             if (elem)
00360             {
00361                 alloc.deallocate(elem, cap);
00362             }
00363             elem = new_data;
00364             cap = sz;
00365         }
00366     }
00367
00368     // Operatoriai elementu pasiekimui
00369     reference operator[](size_type n)
00370     {
00371         return elem[n];
00372     }
00373     const_reference operator[](size_type n) const
00374     {
00375         return elem[n];
00376     }
00377     const_reference at(size_type n) const
00378     {
00379         if (n >= sz)
00380         {
00381             throw std::out_of_range("Vector::at: index out of range");
00382         }
00383         return elem[n];
00384     }
00385     reference at(size_type n)
00386     {
00387         if (n >= sz)
00388         {
00389             throw std::out_of_range("Vector::at: index out of range");
00390         }
00391         return elem[n];
00392     }
00393     reference front()
00394     {
00395         return elem[0];
00396     }
00397     const_reference front() const
00398     {
00399         return elem[0];
00400     }
00401     reference back()
00402     {
00403         return elem[sz-1];
00404     }
00405     const_reference back() const
00406     {
00407         return elem[sz-1];
00408     }
00409
00410     T* data() noexcept
00411     {
00412         return elem;
00413     }
00414     const T* data() const noexcept
```

```
00415     {
00416         return elem;
00417     }
00418 
00419     // Funkcija, vektoriaus gale sukonstruojanti elementa
00420     template<class... Args> reference emplace_back(Args&&... args)
00421     {
00422         if (sz == cap)
00423         {
00424             reserve(cap * 2);
00425         }
00426         alloc.construct(&elem[sz], std::forward<Args>(args)...);
00427         return elem[sz++];
00428     }
00429 
00430     // Funkcija, sukonstruojanti elementa vektoriaus gale
00431     void push_back(const T& x)
00432     {
00433         if (sz == cap)
00434         {
00435             reserve(cap* 2);
00436         }
00437         alloc.construct(&elem[sz++], x);
00438     }
00439 
00440     // Funkcija, perkelianti elementa i vektoriaus gala
00441     void push_back(T&& x)
00442     {
00443         emplace_back(std::move(x));
00444     }
00445 
00446     // Funkcija, pasalinanti paskutini vektoriaus elementa
00447     void pop_back()
00448     {
00449         if (sz > 0)
00450         {
00451             alloc.destroy(&elem[--sz]);
00452         }
00453     }
00454 
00455     // Funkcija, sukonstruojanti nauja elementa vartotojo parinktoje vektoriaus vietoje
00456     template<class... Args> iterator emplace(const_iterator position, Args&&... args)
00457     {
00458         size_type pos_index = position - cbegin();
00459         if (sz == cap)
00460         {
00461             reserve(2 *cap);
00462         }
00463         if (pos_index < sz)
00464         {
00465             for (size_type i = sz; i > pos_index; --i)
00466             {
00467                 alloc.construct(&elem[i], std::move(elem[i - 1]));
00468                 alloc.destroy(&elem[i - 1]);
00469             }
00470         }
00471         alloc.construct(&elem[pos_index], std::forward<Args>(args)...);
00472         ++sz;
00473         return begin() + pos_index;
00474     }
00475 
00476     // Funkcija, ikopijuojanti nauja elementa i vektoriaus tam tikra vieta
00477     iterator insert(const_iterator position, const T& x)
00478     {
00479         return emplace(position, x);
00480     }
00481 
00482     // Funkcija, perkelianti nauja elementa i vektoriaus tam tikra vieta
00483     iterator insert(const_iterator position, T&& x)
00484     {
00485         return emplace(position, std::move(x));
00486     }
00487 
00488     // Funkcija, ikopijuojanti n elementu su reiksme x i pasirinkta vektoriaus vieta
00489     iterator insert(const_iterator position, size_type n, const value_type& x)
00490     {
00491         size_type pos_index = position - cbegin();
00492         if (sz + n > cap)
00493         {
00494             reserve(sz + n);
00495         }
00496         for (size_type i = sz; i > pos_index; --i)
00497         {
00498             alloc.construct(&elem[i + n - 1], std::move(elem[i - 1]));
00499             alloc.destroy(&elem[i - 1]);
00500         }
00501         for (size_type i = 0; i < n; ++i)
```

```
00502         {
00503             alloc.construct(&elem[pos_index + i], x);
00504         }
00505         sz += n;
00506         return begin() + pos_index;
00507     }
00508 
00509     // Funkcija, iterpianti intervala elementu i nurodyta vektoriaus vieta
00510     template<class InputIt, typename = std::_RequireInputIter<InputIt»
00511     iterator insert(const_iterator position, InputIt first, InputIt last)
00512     {
00513         size_type pos_index = position - cbegin();
00514         size_type n = std::distance(first, last);
00515         if (sz + n > cap)
00516         {
00517             reserve(sz + n);
00518         }
00519         for (size_type i = sz; i > pos_index; --i)
00520         {
00521             alloc.construct(&elem[i + n - 1], std::move(elem[i - 1]));
00522             alloc.destroy(&elem[i - 1]);
00523         }
00524         for (size_type i = 0; first != last; ++i, ++first)
00525         {
00526             alloc.construct(&elem[pos_index + i], *first);
00527         }
00528         sz += n;
00529         return begin() + pos_index;
00530     }
00531 
00532     // Funkcija, iterpianti duota elementu sarasa i paskirta vektoriaus vieta
00533     iterator insert(const_iterator position, initializer_list<T> il)
00534     {
00535         return insert(position, il.begin(), il.end());
00536     }
00537 
00538     // Funkcija, istrinanti elementa is nurodytos vektoriaus pozicijos
00539     iterator erase(const_iterator position)
00540     {
00541         size_type pos_index = position - cbegin();
00542         if (pos_index >= sz)
00543         {
00544             throw std::out_of_range("Vector::erase: position out of range");
00545         }
00546         alloc.destroy(&elem[pos_index]);
00547         for (size_type i = pos_index; i < sz - 1; ++i)
00548         {
00549             alloc.construct(&elem[i], std::move(elem[i + 1]));
00550             alloc.destroy(&elem[i + 1]);
00551         }
00552         --sz;
00553         return begin() + pos_index;
00554     }
00555 
00556     // Funkcija, istrinanti nurodyta intervala vektoriaus elementu
00557     iterator erase(const_iterator first, const_iterator last)
00558     {
00559         size_type start_index = first - cbegin();
00560         size_type end_index = last - cbegin();
00561         if (start_index >= sz || end_index > sz || start_index > end_index)
00562         {
00563             throw std::out_of_range("Vector::erase: range out of range");
00564         }
00565 
00566         for (size_type i = start_index; i < end_index; ++i)
00567         {
00568             alloc.destroy(&elem[i]);
00569         }
00570         for (size_type i = end_index; i < sz; ++i)
00571         {
00572             alloc.construct(&elem[i - (end_index - start_index)], std::move(elem[i]));
00573             alloc.destroy(&elem[i]);
00574         }
00575         sz -= (end_index - start_index);
00576         return begin() + start_index;
00577     }
00578 
00579     // Funkcija, apkeicianti dvieju vektoriu reiksmes vietomis
00580     void swap(Vector& other) noexcept (allocator_traits<Allocator>::propagate_on_container_swap::value
    ||
00581             allocator_traits<Allocator>::is_always_equal::value)
00582     {
00583         std::swap(elem, other.elem);
00584         std::swap(sz, other.sz);
00585         std::swap(cap, other.cap);
00586         std::swap(alloc, other.alloc);
00587     }
```

```
00588
00589     // Funkcija, isvalanti vektoriu
00590     void clear() noexcept
00591     {
00592         for (size_type i = 0; i < sz; ++i)
00593         {
00594             alloc.destroy(&elem[i]);
00595         }
00596         sz = 0;
00597     }
00598 };
00599
00600 }
00601
00602 #endif
```

Index