My Project

Generated by Doxygen 1.11.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

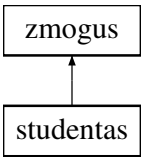
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 studentas Class Reference

Inheritance diagram for studentas:



Public Member Functions

- **studentas** (string Var, string Pav, VEKTORIUS< int > tarp, double egz)
- virtual std::string get_Vardas () const override
- ~**studentas** ()
  destruktorius
- **studentas** (const studentas &other)
  kopijavimo konstruktorius
- studentas & **operator=** (const studentas &other)
  kopijavimo assignment operatorius
- **studentas** (studentas &&other) noexcept
  perkelimo konstruktorius
- studentas & **operator=** (studentas &&other) noexcept
  perkelimo assignment operatorius
- bool operator== (const studentas stud) const
- void **set_egz** (int egz)
- double **get_egz** ()
- void **set_tarpiniai** (VEKTORIUS< int > tarp)
- VEKTORIUS< int > **get_tarpiniai** () const
- void **set_paz_kiekis** (int paz)
- int **get_paz_kieki** ()
- void **set_vidurkis** (double vidur)
- void **set_vidurkis** ()
- double **get_vidurkis** ()
- void **set_mediana** (double medi)
- double **get_mediana** ()
- void **set_gal_v_m** ()
- double **get_gal_v_m** (int pasirinkimas) const

Public Member Functions inherited from zmogus

- **zmogus** (const std::string &Var, const std::string &Pav)
- void **set_Vardas** (string var)
- void **set_Pavarde** (string pav)
- virtual std::string **get_Pavarde** () const

Protected Attributes

- double **egz_rez**
- int **pazKiekis**
- VEKTORIUS< int > **tarpiniai**
- double **vidurkis**
- double **mediana**
- double **gal_vid**
- double **gal_med**

Protected Attributes inherited from zmogus

- string **Vardas**
- string **Pavarde**

Friends

- ostream & operator<< (ostream &out, const studentas &stud)
- ostream & **su_mediana** (ostream &out, const studentas &stud)
- ostream & **su_vidurkiu** (ostream &out, const studentas &stud)
- istream & **operator**>> (istream &in, studentas &stud)

4.1.1 Member Function Documentation

4.1.1.1 get_Vardas()

```
virtual std::string studentas::get_Vardas () const  [inline], [override], [virtual]
```

Implements zmogus.

4.1.1.2 operator==()

```
bool studentas::operator== (
            const studentas stud) const  [inline]
```

ivesties operatorius

4.1.2 Friends And Related Symbol Documentation

4.1.2.1 operator<<

```
ostream & operator<< (
            ostream & out,
            const studentas & stud)  [friend]
```

isvesties operatorius

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

- funkcijos3.h

4.2 VEKTORIUS< T > Class Template Reference

Public Types

- typedef T * **rodykle**
  kiek elementu gali buti
- typedef T * **iterator**
- typedef size_t **size_type**
- typedef T **reiksme_type**
- typedef const T * **const_iterator**
- typedef std::reverse_iterator< iterator > **reverse_iterator**
- typedef std::reverse_iterator< const_iterator > **const_reverse_iterator**

Public Member Functions

- **dydis** (n_dydis)
- **kiekis** (n_dydis)
- **VEKTORIUS** (std::initializer_list< T > sarasas)
- **VEKTORIUS** (const VEKTORIUS &kitas)
- VEKTORIUS & **operator=** (const VEKTORIUS &naujas)
- VEKTORIUS & **operator=** (VEKTORIUS &&naujas)
- **VEKTORIUS** (VEKTORIUS &&naujas) noexcept
- T & **at** (size_t indeksas)
- const T & **at** (size_type indeksas) const
- T & **operator[ ]** (size_t indeksas)
- const T & **operator[ ]** (size_t indeksas) const
- T & **front** ()
- const T & **front** () const
- T & **back** ()
- const T & **back** () const
- T * **data** () noexcept
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- const_iterator **cbegin** () const noexcept
- const_iterator **cend** () const noexcept

- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () 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
- void **reserve** (size_type naujsKiekis)
- size_type **capacity** () const noexcept
- void **shrink_to_fit** ()
- void **clear** () noexcept
- iterator **insert** (const_iterator pos, const T &reiksme)
- iterator **erase** (const_iterator pos)
- iterator **erase** (const_iterator pirmas, const_iterator paskutinis)
- void **push_back** (const T &pradziaa)
- void **pop_back** ()
- void **resize** (size_type new_size)
- void **swap** (VEKTORIUS &naujas) noexcept
- bool **operator==** (const VEKTORIUS< T > &naujas) const
- bool **operator!=** (const VEKTORIUS< T > &naujas) const
- bool **operator**< (const VEKTORIUS< T > &naujas) const
- bool **operator**<= (const VEKTORIUS< T > &naujas) const
- bool **operator**> (const VEKTORIUS< T > &naujas) const
- bool **operator**>= (const VEKTORIUS< T > &naujas) const
- T **suma** () const

Public Attributes

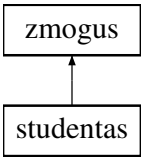
- std::allocator< T > **alloc**
- iterator **pradzia**
- iterator **pab**
- : kiekis(new T[n_dydis])

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

- vektorius.h

4.3 zmogus Class Reference

Inheritance diagram for zmogus:



Public Member Functions

- **zmogus** (const std::string &Var, const std::string &Pav)
- void **set_Vardas** (string var)
- void **set_Pavarde** (string pav)
- virtual std::string **get_Vardas** () const =0
- virtual std::string **get_Pavarde** () const

Protected Attributes

- string **Vardas**
- string **Pavarde**

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

- funkcijos3.h

Chapter 5

File Documentation

5.1 funkcijos3.h

```
00001 #ifndef FUNKCIJOS3_H
00002 #define FUNKCIJOS3_H
00003 #include <numeric>
00004 #include "VEKTORIUS.h"
00005
00006 #include "manoBiblioteka.h"
00007
00008 class zmogus {
00009     protected:
00010         string Vardas;
00011         string Pavarde;
00012
00013     public:
00014         zmogus() : Vardas(""), Pavarde("") {}
00015         zmogus(const std::string& Var, const std::string& Pav) : Vardas(Var), Pavarde(Pav) {}
00016
00017         virtual ~zmogus() {}
00018
00019         void set_Vardas(string var){
00020             Vardas = var;
00021         }
00022         void set_Pavarde(string pav){
00023             Pavarde = pav;
00024         }
00025
00026         virtual std::string get_Vardas() const = 0;
00027
00028         virtual std::string get_Pavarde() const {
00029             return Pavarde;
00030         }
00031
00032
00033 };
00034
00035 class studentas : public zmogus {
00036     protected:
00037
00038         double egz_rez;
00039         int pazKiekis;
00040         VEKTORIUS <int> tarpiniai;
00041         double vidurkis;
00042         double mediana;
00043         double gal_vid;
00044         double gal_med;
00045
00046     public:
00047         studentas () : tarpiniai (), pazKiekis (0), egz_rez (0){}
00048
00049         studentas(string Var, string Pav, VEKTORIUS <int> tarp, double egz ) : zmogus(Var, Pav) {
00050
00051             tarpiniai = tarp;
00052             pazKiekis = tarp.size();
00053             egz_rez = egz;
00054             pazKiekis = tarpiniai.size();
00055
00056         }
00057
00058         virtual std::string get_Vardas() const override {
```

```
00059             return Vardas;
00060         }
00061
00063         ~studentas ( ){tarpiniai.clear();}
00064
00066         studentas(const studentas& other) : zmogus(other.get_Vardas(), other.get_Pavarde()) {
00067             egz_rez = other.egz_rez;
00068             tarpiniai = other.tarpiniai;
00069             vidurkis = other.vidurkis;
00070             mediana = other.mediana;
00071             gal_vid = other.gal_vid;
00072             gal_med = other.gal_med;
00073         }
00074
00075
00077         studentas& operator=(const studentas& other) {
00078             if (this != &other) {
00079                 zmogus :: set_Vardas (other.get_Vardas());
00080                 zmogus :: set_Pavarde (other.get_Pavarde());
00081                 egz_rez = other.egz_rez;
00082                 tarpiniai = other.tarpiniai;
00083                 vidurkis = other.vidurkis;
00084                 mediana = other.mediana;
00085                 gal_vid = other.gal_vid;
00086                 gal_med = other.gal_med;
00087             }
00088             return *this;
00089         }
00090
00092        studentas(studentas&& other) noexcept {
00093             Vardas = move(other.Vardas);
00094             Pavarde = move(other.Pavarde);
00095             egz_rez = other.egz_rez;
00096             tarpiniai = move(other.tarpiniai);
00097             vidurkis = other.vidurkis;
00098             mediana = other.mediana;
00099             gal_vid = other.gal_vid;
00100             gal_med = other.gal_med;
00101             other.vidurkis = 0;
00102             other.mediana = 0;
00103             other.gal_vid = 0;
00104             other.gal_med = 0;
00105         }
00106
00108         studentas& operator=(studentas&& other) noexcept {
00109             if (this != &other) {
00110                 Vardas = move(other.Vardas);
00111                 Pavarde = move(other.Pavarde);
00112                 egz_rez = other.egz_rez;
00113                 tarpiniai = move(other.tarpiniai);
00114                 vidurkis = other.vidurkis;
00115                 mediana = other.mediana;
00116                 gal_vid = other.gal_vid;
00117                 gal_med = other.gal_med;
00118                 other.vidurkis = 0;
00119                 other.mediana = 0;
00120                 other.gal_vid = 0;
00121                 other.gal_med = 0;
00122             }
00123             return *this;
00124         }
00125
00126         friend ostream& operator«(ostream& out, const studentas &stud) {
00127         out « left « setw(15) « stud.get_Vardas() « setw(15) « stud.get_Pavarde();
00128         out « setw(15) « fixed « setprecision(2) « stud.get_gal_v_m(0);
00129         out « setw(20) « setprecision(2) « stud.get_gal_v_m(1) « "\n";
00130         return out;
00131         }
00132
00133         friend ostream& su_mediana(ostream& out, const studentas& stud) {
00134             out « left « setw(15) « stud.get_Vardas() « setw(15) « stud.get_Pavarde();
00135             out « setw(20) « setprecision(2) « stud.gal_med « "\n";
00136             return out;
00137         }
00138
00139         friend ostream& su_vidurkiu(ostream& out, const studentas& stud) {
00140             out « left « setw(15) « stud.get_Vardas() « setw(15) « stud.get_Pavarde();
00141             out « setw(20) « setprecision(2) « stud.gal_vid« "\n";
00142             return out;
00143         }
00144
00145
00146         friend istream& operator»(istream& in, studentas &stud){
00147             in » stud.Vardas » stud.Pavarde;
00148             int paz;
00149             VEKTORIUS <int> pzm;
00150             while(in » paz)
```

```
00151             {
00152                 stud.tarpiniai.push_back(paz);
00153             }
00154             if(!stud.tarpiniai.empty())
00155             {
00156                 stud.egz_rez = stud.tarpiniai.back();
00157                 stud.tarpiniai.pop_back();
00158             }
00159 
00160             return in;
00161         }
00162 
00163         bool operator==(const studentas stud) const {
00164 
00165             if (get_Vardas() == stud.get_Vardas() && get_Pavarde() == stud.get_Pavarde()  )
00166             true;
00167             else false;
00168 
00169         }
00170 
00171         void set_egz(int egz){            egz_rez = egz;        }
00172 
00173         double get_egz(){            return egz_rez;        }
00174 
00175         void set_tarpiniai(VEKTORIUS <int> tarp){            tarpiniai = tarp;        }
00176 
00177         VEKTORIUS <int> get_tarpiniai( ) const{       return tarpiniai;        }
00178 
00179         void set_paz_kiekis(int paz){            pazKiekis = paz;                }
00180 
00181         int get_paz_kieki(){            return pazKiekis;        }
00182 
00183         void set_vidurkis(double vidur){            vidurkis = vidur;        }
00184 
00185         // void set_vidurkis(){            vidurkis = accumulate(tarpiniai.begin(), tarpiniai.end(),
      0.0) / tarpiniai.size();        }
00186         void set_vidurkis() {
00187             double sum = 0.0;
00188             for (size_t i = 0; i < tarpiniai.size(); ++i) {
00189                 sum += tarpiniai[i];
00190             }
00191             vidurkis = (tarpiniai.size() > 0) ? (sum / tarpiniai.size()) : 0;
00192         }
00193 
00194 
00195         double get_vidurkis(){     return vidurkis;        }
00196 
00197         void set_mediana(double medi){  mediana = medi;    }
00198 
00199         double get_mediana(){  return mediana; }
00200 
00201         void set_gal_v_m (){
00202 
00203             gal_vid = vidurkis * 0.4 + 0.6 *egz_rez;
00204             pazKiekis = tarpiniai.size();
00205 
00206             sort(tarpiniai.begin(), tarpiniai.end());
00207 
00208             if ((pazKiekis % 2) == 0)
00209                 {
00210                     mediana = (double(tarpiniai[pazKiekis / 2 - 1]) + (tarpiniai[pazKiekis / 2])) / 2;
00211                     if(mediana >10) mediana = gal_vid;
00212                 }
00213             else
00214                 {
00215                     mediana = (tarpiniai[pazKiekis / 2]);
00216                     if(mediana >10) mediana = gal_vid;
00217                 }
00218 
00219                 gal_med = mediana * 0.4 + 0.6 *egz_rez;
00220 
00221 
00222         }
00223         double get_gal_v_m (int pasirinkimas) const{
00224             if (pasirinkimas == 0) return gal_vid;
00225             if (pasirinkimas == 1) return gal_med ;
00226         }
00227 
00228 };
00229 
00230 
00232 const VEKTORIUS <string> vyrV = {"Jonas", "Petras", "Antanas", "Juozas","Arnas","Dziugas", "Mantas",
      "Stasius","Vilius", "Kazimieras", "Algirdas", "Rimas", "Mindaugas", "Rokas", "Paulius","Kajus",
      "Pijus", "Titas"};
00233 const VEKTORIUS <string> vyrP = {"Kazlauskas", "Petrauskas", "Jonaitis", "Antanaitis", "Rimkus",
      "Grybauskas", "Brazauskas", "Vaitkevicius", "Statkus", "Sutkaitis","Baciuska", "Zulkus"};
00234 
```

```
00236 const VEKTORIUS <string> motV = {"Ona", "Marija", "Lina", "Gabija", "Jurga", "Egle",
      "Ruta","Aida","Karolina","Austeja","Karina","Meda","Jorune", "Gintare", "Deimante", "Aiste", "Kamile",
      "Rugile", "Ugne","Selina","Monika", "Paulina", "Adriana"};
00237 const VEKTORIUS <string> motP = {"Kazlauskiene", "Petrauskiene", "Jonaite", "Antanaite", "Rimkute",
      "Grybauskiene", "Brazauskiene", "Vaitkeviciute", "Zobelaite","Macaite","Mockute"};
00238
00239 void spausdinimas(vector<studentas>& var);
00240 double galutinis(vector<studentas>& var, int &k,int &pasirinkimas);
00241 double mediana(vector<studentas>& var,int &k);
00242 double vidurkis(vector<studentas>& var,int &k);
00243 void atsitiktiniaiPazymiai(vector<studentas>& var, int &studSk);
00244 void tikrinimas(int &pasirinkimas);
00245 void ivedimasRanka(vector<studentas>& var, int &studSk);
00246 void atsitiktiniaiPazVar (vector<studentas>& var, int &studSk);
00247 void su_duomenimis_is_failu(vector<studentas>& kursiokai, vector<studentas>& var);
00248 void skaitymas(vector<studentas>& var, VEKTORIUS <string>&failoPav, int indeksas, double &laikas,int
      laboras);
00249 void failoKurimas(VEKTORIUS <string>&failoPav, int indeksas, int &kiekND,int &studSk, double &laikas);
00250 void rusiavimas(vector<studentas>& var);
00251 void spausdinami_surikiuoti(vector<studentas>& var,double &laikas);
00252 bool rikiuotiVarda(const studentas &a, const studentas &b);
00253 bool rikiuotiPavarde(const studentas &a, const studentas &b);
00254 bool rikiuotiVid(const studentas &a, const studentas &b);
00255 bool rikiuotiMed(const studentas &a, const studentas &b);
00256 void didejimo (vector<studentas>& var, double &laikas);
00257
00258 void testavimui(vector<studentas>& var);
00259 void rusiavimasTest(vector<studentas>& var, vector<studentas>& vargsai,vector<studentas>& galvociai,
      double &laikas, int &pasirinkimas,int indeksas);
00260 void spausdinimasTest(vector<studentas>& vargsai, vector<studentas>& galvociai, VEKTORIUS <string>
      pav, double &laikas,int &pasirinkimas, int indeksas);
00261 void testFail(vector<studentas>& var);
00262 void testFail_3strategija(vector<studentas> &var);
00263 void rusiavimasTest_3strategija(vector<studentas> &var, vector<studentas> &vargsai, double
      &galutinisLaikas, int &pasirinkimas, int indeksas);
00264 void testFail_2strategija(vector<studentas> &var);
00265 void rusiavimasTest_2strategija(vector<studentas> &var, vector<studentas> &vargsai, double
      &galutinisLaikas, int &pasirinkimas, int indeksas);
00266
00267 void testuoti_clase();
00268
00269 #endif
```

5.2 manoBiblioteka.h

```
00001 #ifndef MANOBIBLIOTEKA_H
00002 #define MANOBIBLIOTEKA_H
00003
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <fstream>
00007 #include <vector>
00008 #include <math.h>
00009 #include <string>
00010 #include <algorithm>
00011 #include <random>
00012 #include <ctime>
00013 #include <sstream>
00014 #include <chrono>
00015 #include <stdexcept>
00016 #include <limits>
00017 #include <list>
00018 #include <deque>
00019 #include <iterator>
00020 #include <utility>
00021 #include <numeric>
00022
00023
00024 using std::cout;
00025 using std::cin;
00026 using std::setw;
00027 using std::string;
00028 using std::left;
00029 using std::endl;
00030 using std::vector;
00031 using std::invalid_argument;
00032 using std::cerr;
00033 using std::numeric_limits;
00034 using std::streamsize;
00035 using std::ofstream;
00036 using std::fixed;
00037 using std::setprecision;
00038 using std::ifstream;
```

```
00039 using std::istringstream;
00040 using std::list;
00041 using std::deque;
00042 using std::sort;
00043 using std::move;
00044 using std::ostream;
00045 using std::istream;
00046
00047
00048 #endif
```

5.3 vektorius.h

```
00001 #ifndef VEKTORIUS_H
00002 #define VEKTORIUS_H
00003 #include "manoBiblioteka.h"
00004
00005 #include <iostream>
00006 #include <algorithm>
00007 #include <sstream>
00008 #include <cassert>
00009 #include <chrono>
00010 #include <iomanip>
00011 #include <numeric>
00012
00013 template <typename T>
00014     class VEKTORIUS
00015     {
00016         private:
00017             T *r_masyvas;
00018             size_t dydis;
00019             size_t kiekis;
00020
00021         public:
00022
00023             typedef T *rodykle;
00024             typedef T *iterator;
00025             typedef size_t size_type;
00026             typedef T reiksme_type;
00027             std::allocator<T> alloc;
00028             typedef const T* const_iterator;
00029             typedef std::reverse_iterator<iterator> reverse_iterator;
00030             typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
00031             iterator pradzia;
00032             iterator pab;
00033
00034
00035             // Konstruktorius
00036             VEKTORIUS() : r_masyvas(nullptr), dydis(0), kiekis(0) {}
00037
00038             // VEKTORIUS(std::initializer_list<T> init_list) {
00039             //     dydis = init_list.size();
00040             //     kiekis = dydis;
00041             //     r_masyvas = new T[kiekis];
00042             //     std::copy(init_list.begin(), init_list.end(), r_masyvas);
00043             // }
00044
00045             VEKTORIUS(size_type n_dydis, const T& value = T())
00046                 : kiekis(new T[n_dydis]), dydis(n_dydis), kiekis(n_dydis)
00047             {
00048                 std::fill(r_masyvas, r_masyvas + dydis, value);
00049             }
00050
00051             // VEKTORIUS(size_t initial_size) : dydis(initial_size), kiekis(0) {
00052             //     r_masyvas = new T[initial_size];
00053             // }
00054
00055             VEKTORIUS(std::initializer_list<T> sarasas)
00056                 : r_masyvas(new T[sarasas.size()]), dydis(sarasas.size()), kiekis(sarasas.size())
00057             {
00058                 std::copy(sarasas.begin(), sarasas.end(), r_masyvas);
00059             }
00060
00061
00062             // Kopijavimo konstruktorius
00063             VEKTORIUS(const VEKTORIUS& kitas)
00064                 : r_masyvas(new T[kitas.dydis]), dydis(kitas.dydis), kiekis(kitas.kiekis)
00065             {
00066                 for(int i = 0; i !=dydis; ++i)
00067                 {
00068                     r_masyvas[i]=kitas.r_masyvas[i];
00069                 }
00070             }
```

```
00071
00072           // Kopijavimo priskyrimo operatorius
00073           VEKTORIUS &operator=(const VEKTORIUS& naujas)
00074           {
00075                if (this != &naujas) {
00076                   T* naujiDuom = new T[naujas.dydis];
00077                   for(int i = 0; i !=dydis; ++i)
00078                   {
00079                       naujiDuom[i]=naujas.r_masyvas[i];
00080                   }
00081                   delete[] r_masyvas;    // atlaisvinama sena atmintis
00082                   r_masyvas = naujiDuom; // mduom pointina i nauja atminti
00083                   dydis = naujas.dydis;
00084                   kiekis = naujas.kiekis;
00085               }
00086               return *this;
00087           }
00088
00089           // Perkėlimo priskyrimo operatorius
00090           VEKTORIUS &operator=(VEKTORIUS &&naujas)
00091           {
00092               if (&naujas == this) return *this;
00093               delete[] r_masyvas; // atlaisviname seną atmintį
00094               r_masyvas = naujas.r_masyvas; // elem point'ina į v.elem atmintį
00095               dydis = naujas.dydis; // atnaujiname size
00096               kiekis = naujas.kiekis;
00097               naujas.r_masyvas = nullptr; // v neturi jokių elementų
00098               naujas.dydis = 0;
00099               naujas.kiekis= 0;
00100
00101               return *this; // grąžiname objektą
00102           }
00103
00104           // Perkėlimo konstruktorius
00105           VEKTORIUS(VEKTORIUS && naujas) noexcept
00106               : r_masyvas(naujas.r_masyvas), dydis(naujas.dydis), kiekis(naujas.kiekis)
00107           {
00108               naujas.r_masyvas = nullptr;
00109               naujas.dydis = 0;
00110               naujas.kiekis = 0;
00111           }
00112
00113           // Dekstruktorius
00114           ~VEKTORIUS()
00115           {
00116               delete[] r_masyvas;
00117           }
00118
00119           // Pasiekia elemeta tam tikrame indekse
00120           T &at(size_t indeksas)
00121           {
00122               if (indeksas >= kiekis)
00123                   throw std::out_of_range("Indeksas uz ribu");
00124               return r_masyvas[indeksas];
00125           }
00126
00127           const T& at(size_type indeksas) const
00128           {
00129               if (indeksas >= dydis)
00130                   throw std::out_of_range("Index out of range");
00131               return r_masyvas[indeksas];
00132           }
00133
00134           // Elementas pagal indeksą []
00135           T &operator[](size_t indeksas)
00136           {
00137               if (indeksas >= dydis)
00138                   throw std::out_of_range("Indeksas uz ribu");
00139               return r_masyvas[indeksas];
00140           }
00141
00142           const T &operator[](size_t indeksas) const
00143           {
00144               if (indeksas >= dydis)
00145                   throw std::out_of_range("Indeksas uz ribu");
00146               return r_masyvas[indeksas];
00147           }
00148
00149           // Pirmas elementas
00150           T &front()
00151           {
00152               if (dydis == 0)
00153                   throw std::out_of_range("VEKTORIUS yra tuščias");
00154               return r_masyvas[0];
00155           }
00156
00157           const T &front() const
```

```
00158             {
00159                 if (dydis == 0)
00160                     throw std::out_of_range("VEKTORIUS yra tuščias");
00161                 return r_masyvas[0];
00162             }
00163 
00164             // Paskutinis elementas
00165             T &back()
00166             {
00167                 if (dydis == 0)
00168                     throw std::out_of_range("VEKTORIUS tuscias");
00169                 return r_masyvas[dydis - 1];
00170             }
00171 
00172             const T &back() const
00173             {
00174                 if (dydis == 0)
00175                     throw std::out_of_range("VEKTORIUS tuscias");
00176                 return r_masyvas[dydis - 1];
00177             }
00178 
00179             T* data() noexcept { return r_masyvas; }
00180 
00181             // grazina iteratoriu i pradzia
00182             iterator begin() noexcept { return r_masyvas; }
00183             const_iterator begin() const noexcept { return r_masyvas; }
00184             iterator end() noexcept { return r_masyvas + dydis; }
00185             const_iterator end() const noexcept { return r_masyvas + dydis; }
00186 
00187             const_iterator cbegin() const noexcept { return r_masyvas; }
00188             const_iterator cend() const noexcept { return r_masyvas + dydis; }
00189             reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
00190             const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
00191 
00192             reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
00193             const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
00194 
00195             const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }
00196             const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }
00197 
00198             // Ar VEKTORIUS yra tuščias
00199             bool empty() const noexcept
00200             {
00201                 return dydis == 0;
00202             }
00203 
00204             // grazina kiek yra elementu
00205             size_type size() const noexcept { return dydis; }
00206 
00207             // kiek daugiausiai elementu VEKTORIUS gali tureti
00208             size_type max_size() const noexcept{ return std::numeric_limits<size_t>::max(); }
00209 
00210             // Rezervuoti vietą elementams
00211             void reserve(size_type naujsKiekis)
00212             {
00213                 if (naujsKiekis <= kiekis)
00214                     return;
00215 
00216                 T* naujiDuom = new T[naujsKiekis];
00217                 for (size_type k = 0; k < dydis; ++k)
00218                     naujiDuom[k] = std::move(r_masyvas[k]);
00219 
00220                 delete[] r_masyvas;
00221                 r_masyvas = naujiDuom;
00222                 kiekis = naujsKiekis;
00223             }
00224 
00225             size_type capacity() const noexcept{ return kiekis; }
00226 
00227             // funkcija sumažina vektoriaus talpą, kad ji atitiktų jo dydį
00228             void shrink_to_fit() {
00229                 if (dydis < kiekis) {
00230                     T* naujiDuom = new T[dydis];
00231                     std::copy(r_masyvas, r_masyvas + dydis, naujiDuom);
00232                     delete[] r_masyvas;
00233                     r_masyvas = naujiDuom;
00234                     kiekis = dydis;
00235                 }
00236             }
00237 
00238             void clear() noexcept
00239             {
00240                 dydis = 0;
00241             }
00242 
00243             // Elemento iterpimas į vektorių
00244             iterator insert(const_iterator pos, const T& reiksme) {
```

```
00245                 size_type index = pos - begin();
00246                 if (dydis == kiekis) {
00247                     size_type new_capacity = (kiekis == 0) ? 1 : kiekis * 2;
00248                     reserve(new_capacity);
00249                 }
00250 
00251                 // Perstumiam visus elementus nuo įterpimo vietos į dešinę per viena
00252                 for (size_type i = dydis; i > index; --i) {
00253                     r_masyvas[i] = std::move(r_masyvas[i - 1]);
00254                 }
00255 
00256                 // Įterpiame naują elementą į vietą 'pos'
00257                 r_masyvas[index] = reiksme;
00258                 ++dydis;
00259 
00260                 return begin() + index;
00261             }
00262 
00263             iterator erase(const_iterator pos) {
00264                 size_type index = pos - begin();
00265                 if (index >= dydis) {
00266                     throw std::out_of_range("Index out of range");
00267                 }
00268 
00269                 // Perstumiam visus elementus nuo vienetu i kaire
00270                 for (size_type i = index; i < dydis - 1; ++i) {
00271                     r_masyvas[i] = std::move(r_masyvas[i + 1]);
00272                 }
00273 
00274                 --dydis;
00275                 return begin() + index;
00276             }
00277 
00278             iterator erase(const_iterator pirmas, const_iterator paskutinis)
00279             {
00280                 size_type pirmas_indeks = pirmas - begin();
00281                 size_type paskutinis_indeks = paskutinis - begin();
00282                 if (pirmas_indeks > paskutinis_indeks || paskutinis_indeks > dydis) {
00283                     throw std::out_of_range("Invalid range");
00284                 }
00285 
00286                 size_type istrinamas = paskutinis_indeks - pirmas_indeks;
00287                 for (size_type i = pirmas_indeks; i < dydis - istrinamas; ++i) {
00288                     r_masyvas[i] = std::move(r_masyvas[i + istrinamas]);
00289                 }
00290 
00291                 dydis -= istrinamas;
00292                 return begin() + pirmas_indeks;
00293             }
00294 
00295             void push_back(const T& pradziaa) {
00296                 if (kiekis == dydis) {
00297                     reserve(kiekis == 0 ? 1 : kiekis * 2);
00298                 }
00299                 r_masyvas[dydis++] = pradziaa;
00300             }
00301 
00302             void pop_back()
00303             {
00304                 if (dydis > 0)
00305                 --dydis;
00306             }
00307 
00308             // iterator resize(size_t n_dydis) {
00309             //     if (n_dydis == 0) {
00310             //         dydis = 0;
00311             //         kiekis = 2;
00312             //         T *naujas = new T[kiekis];
00313             //         delete[] r_masyvas;
00314             //         r_masyvas = naujas;
00315             //         return 0;
00316             //     }
00317             //     size_t lim;
00318             //     T *naujas = new T[n_dydis];
00319             //     if (n_dydis < dydis) {
00320             //         dydis = n_dydis;
00321             //         lim = n_dydis;
00322             //     }
00323             //     else lim = dydis;
00324             //     std::move(&r_masyvas[0], &r_masyvas[lim], naujas);
00325             //     kiekis = n_dydis;
00326             //     delete[] r_masyvas;
00327             //     r_masyvas = naujas;
00328             // }
00329 
00330             void resize(size_type new_size)
00331             {
```

```
00332                 if (new_size < dydis) {
00333                     dydis = new_size;
00334                 } else if (new_size > dydis) {
00335                     reserve(new_size);
00336                     dydis = new_size;
00337                 }
00338             }
00339 
00340             void swap(VEKTORIUS& naujas) noexcept {
00341                 std::swap(r_masyvas, naujas.r_masyvas);
00342                 std::swap(dydis, naujas.dydis);
00343                 std::swap(kiekis, naujas.kiekis);
00344             }
00345 
00346 
00347             bool operator== (const VEKTORIUS<T>& naujas) const {
00348                 if (size() != naujas.size()) {
00349                     return false;
00350                 }
00351                 return std::equal(begin(), end(), naujas.begin());
00352             }
00353             bool operator!= (const VEKTORIUS<T>& naujas) const {
00354                 return !(*this == naujas);
00355             }
00356             bool operator < (const VEKTORIUS<T> & naujas) const {
00357                 return std::lexicographical_compare(  begin(), end(), naujas.begin(), naujas.end());
00358             }
00359             bool operator <= (const VEKTORIUS<T> & naujas) const {
00360                 return !(naujas < *this);
00361             }
00362             bool operator > (const VEKTORIUS<T> & naujas)const  {
00363                 return std::lexicographical_compare( naujas.begin(), naujas.end(), begin(), end());
00364             }
00365             bool operator >= (const VEKTORIUS<T> & naujas) const {
00366                 return !(naujas > *this);
00367             }
00368 
00369 
00370         T suma() const {
00371             return std::accumulate(begin(), end(), T());
00372         }
00373 
00374     };
00375 
00376 
00377 
00378 #endif // VEKTORIUS_H
```

Index