My Project

Generated by Doxygen 1.10.0

Chapter 1

README

v2.0

Makefile set-up: **Makefile idiegimas naudojant Chocolatey (Windows)**

Isitikinkite, kad turite Chocolatey idiegima: patikrinkite, ar jusu kompiuteryje yra idiegta Chocolatey. Jei ne, idiekite pagal instrukcijas https://chocolatey.org/install.

Idiekite Makefile: atidarykite PowerShell kaip administratorius ir ivykdykite sia komanda:

choco install make

Patikrinkite idiegima: patikrinkite, ar Makefile sekmingai idiegtas, vykdydami komanda:

make –version

Jei viskas sekminga, turetumete pamatyti Make versijos informacija

Makefile idiegimas naudojant kitus metodus **MacOS:** Makefile iprastai yra idiegtas standartinėje MacOS distribucijoje, todel papildomu veiksmu paprastai nereikia

Linux: Daugumoje Linux distribuciju Makefile taip pat yra idiegtas is anksto. Jei reikia, naudokite savo paketu tvarkykle, pvz., apt-get, yum, dnf, arba kita pagal distribucija

Windows (be Chocolatey): Noredami idiegti Makefile Windows sistemoje be Chocolatey, galite naudoti rankinius idiegimo failus, kuriuos galite rasti internete. Paprastai tie failai turi .exe pletini ir gali buti lengvai idiegti, sekdamie pridedamas instrukcijas

Norint pradeti, i terminala reikia ivesti "make", kai viskas bus sukompiliuota, galima testi su programa, jei norima, galima rasyti "make clean" norint istrinti .o ir .exe failus

Programos naudojimas veikimo metu:

Vartotojas pasirenka, su kokiu konteineriu norima vygdyti programa ir pasirinktinai i terminala parasoma: ./vektoriai , ./list arba ./deque

Vos paleidus programa atsiras pasirinkimas ar pratestuoti musu turimus klases metodus , jei paspaudziame 't', tada pasirenkame numeri nuo 1-5 ir gauname testo rezultata

Toliau musu bus klausiama, ar norime ivesti duomenis ar skaityti is failo

1. Jei bus pasirenkamas duomenu ivedimas, bus reikalaujama pasirinkti ar norima ivesti/generuoti duomenis

1.1 Ar vienu, ar kitu budu reikes ivesti studentu vardus ir pavardes, toliau reikes ivesti studentu namu darbu ir egzamino pazymius

1.2 Jei bus pasirinktas duomenu generavimas, po vardu ir pavardziu irasymo nieko daryti nebereikes

1.3 Galiausiai reikes pasirinkti kur norime matyti duomenis ekrane ar faile

1. Jei pacioje pradzioje bus pasirinktas skaitymas, jusu bus klausiama ar norite generuoti naujus failus, jei ivesite 't'(taip), bus generuojami nauji failai, jei ivesite bet koki kita simboli, programa veiks toliau

2.1 Toliau, jusu bus klausiama ar norite skaityti naujai sukurtus failus, ar jau turimus

2.2 Bus prasoma ivesti, pasirinktinai, turimu/nauju failu kieki, jie bus nuskaitomi, isvedami apytiksliai testavimu laikai ekrane bei sukuriami nauji failai, kuriuose yra surusiuoti studentai pagal vidurki (nuskriaustieji/mokslinciai)

2.3 Galiausiai, kaip ir anksciau, bus isvedami apytiksliai testavimu laikai ekrane bei sukuriami nauji failai, kuriuose yra surusiuoti studentai pagal vidurki (nuskriaustieji/mokslinciai)

RELEASAI

0.1

Sukurta nauja repozitorija, realizuotos elementarios funkcijos, kaip vidurkio ir medianos skaiciavimas. Rezultate gavome, kad vektorius naudoti yra zymiai efektyviau atminties atzvilgiu.

0.2

Programa padaryta prieinamesne vartotojui, galima ne tik irasyti, bet ir skaityti is failo. Testuojama su 10000, 100000 ir 1000000 dydzio failais.

0.3

Prideti header failai, try/catch blokai. Rezultate programa tapo labiau strukturizuota bei klaidu gaudymas uzdraus-davo programos luzima.

0.4

Programa pagal vartotojo pasirinkima sukuria naujus failus, isskirsto mokinius i vargsiukus ir mokslincius, isveda i failus. Padaryti tikslus laiko matavimai.

1.0

Programa padaryta veikti su atskiro tipo konteineriais: deque, list ir vector. Kiekvienas pagal tris strategijas. Pagal matavimo rezultatus greiciausiai buvo vykdoma vector programa naudojant 3 strategija.

1.1

Atliktas repozitorijos kopijavimas. Programoje is strukturu pereinama i klases. Rezultate, programa veikia nasiau naudojant klases.

1.2

Igyvendinti visi "Rule of Five" ir isviesties bei ivesties operatoriai savai klasei.

1.5

Sukurta dar viena **bazine**, abstrakcioji klase, kuriai priklauso klase derived. Prideti konstruktoriaus, copy konstruk-toriaus... testavimai.

2.0

Per Doxygen HTML formatu sukurta dokumentacija bei padaryti Unit testai naudojant patogu C++ framework'a supratimui.

Kaip atrodo Doxygen dokumentacija:

Unit testai:

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

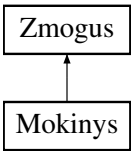
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Mokinys Class Reference

Inheritance diagram for Mokinys:



Public Member Functions

- **Mokinys** (string vard="", string pav="", Vector< int > nd={}, int e=0, double vid=0.0, double med=0.0)
- void patikrinimas () const override
- **Mokinys** (const Mokinys &other)
- **Mokinys** (Mokinys &&other) noexcept
- Mokinys & **operator=** (const Mokinys &other)
- void **clear** ()
- Mokinys & **operator=** (Mokinys &&other) noexcept
- string **getVardas** () const
- string **getPavarde** () const
- Vector< int > **getND** () const
- int **getEgzaminas** () const
- double **getVID** () const
- double **getMED** () const
- void **setVardas** (const string &name)
- void **setPavarde** (const string &surname)
- void **addND** (int nd)
- void **clearND** ()
- void **setEgzaminas** (int exam)
- void **setVID** (double vid)
- void **setMED** (double med)
- void **Vidurkis** (Vector< Mokinys > &A)
- void **Isvedimas** (const Vector< Mokinys > &A, int MOK_kiekis, string isvedimas)
- void **Isvedimas2** (const Vector< Mokinys > &A, int MOK_kiekis, string isvedimas)

- void **Skaitymas** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< int > &IrasuSk, string failas, Vector< Mokinys > &A, int &temp, char strategija)
- void **StudentuRusiavimas** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **StudentuRusiavimas2** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **StudentuRusiavimas3** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **Rikiavimas** (Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &Nuskriaustieji, Vector< int > &IrasuSk)

Public Member Functions inherited from Zmogus

- **Zmogus** (string vard="", string pav="")

Static Public Member Functions

- static bool **PagalVidurki** (const Mokinys &a, const Mokinys &b)
- static bool **PagalMediana** (const Mokinys &a, const Mokinys &b)
- static bool **PagalVarda** (const Mokinys &a, const Mokinys &b)
- static bool **PagalPavarde** (const Mokinys &a, const Mokinys &b)

Friends

- std::ostream & **operator**<< (std::ostream &fr, const Mokinys &temp1)
- istream & **operator**>> (istream &fd, Mokinys &temp1)

Additional Inherited Members

Protected Attributes inherited from Zmogus

- string **vardas**
- string **pavarde**

5.1.1 Member Function Documentation

5.1.1.1 patikrinimas()

```
void Mokinys::patikrinimas ( ) const  [inline], [override], [virtual]
```

Implements Zmogus.

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

- mokinys.h
- mokinys.cpp

5.2 Vector< T > Class Template Reference

Public Types

- using **value_type** = T

Public Member Functions

- **Vector** (size_t size, size_t capacity, const T &defaultValue)
- **Vector** (size_t initialCapacity)
- **Vector** (size_t size, const T &defaultValue)
- **Vector** (std::initializer_list< T > initList)
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- void **push_back** (const T &value)
- void **push_back** (T &&value)
- void **pop_back** ()
- size_t **size** () const
- size_t **capacity** () const
- bool **empty** () const
- T & **operator[ ]** (size_t index)
- const T & **operator[ ]** (size_t index) const
- void **clear** ()
- void **reserve** (size_t newCapacity)
- T * **begin** ()
- T * **end** ()
- const T * **begin** () const
- const T * **end** () const
- T & **back** ()
- const T & **back** () const
- T & **front** ()
- const T & **front** () const
- T * **data_ptr** ()
- const T * **data_ptr** () const
- void **erase** (size_t index)
- void **resize** (size_t newSize, const T &defaultValue=T())
- void **swap** (Vector &other)
- template<typename InputIterator >
  void **assign** (InputIterator first, InputIterator last)
- void **assign** (size_t count, const T &value)
- void **assign** (std::initializer_list< T > ilist)
- void **insert** (size_t index, const T &value)
- template<typename InputIterator >
  void **insert_range** (size_t index, InputIterator first, InputIterator last)
- void **append_range** (std::initializer_list< T > ilist)
- T & **at** (size_t index)
- const T & **at** (size_t index) const
- std::reverse_iterator< T * > **rbegin** ()
- std::reverse_iterator< T * > **rend** ()
- std::reverse_iterator< const T * > **rbegin** () const
- std::reverse_iterator< const T * > **rend** () const

Friends

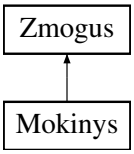
- bool **operator==** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator!=** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator**< (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator**<**=** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator**> (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator**>**=** (const Vector< T > &lhs, const Vector< T > &rhs)

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

- mokinys.h

5.3 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** (string vard="", string pav="")
- virtual void **patikrinimas** () const =0

Protected Attributes

- string **vardas**
- string **pavarde**

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

- mokinys.h

Chapter 6

File Documentation

6.1 funkcijos.h

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003 #include "mokinys.h"
00004
00005 bool Patikrinimas(string kint);
00006 void GeneruotiFailus(Vector<Mokinys>& Nuskriaustieji, Vector<Mokinys>& Mokslinciai, Vector<int>&
      IrasuSk, Vector<Mokinys>& A);
00007 void testConstructor();
00008 void testCopyConstructor();
00009 void testMoveConstructor();
00010 void testCopyAssignment();
00011 void testMoveAssignment();
00012 void Palyginimas();
00013
00014 #endif
```

6.2 mokinys.h

```
00001 #ifndef MOKINYS_H
00002 #define MOKINYS_H
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <iomanip>
00007 #include <string>
00008 #include <Vector>
00009 #include <sstream>
00010 #include <algorithm>
00011 #include <chrono>
00012 #include <cstring>
00013 #include <stdexcept>
00014 #include <list>
00015 #include <deque>
00016 #include <cassert>
00017 #include <utility>
00018 #include <initializer_list>
00019 #include <iterator>
00020
00021
00022 using namespace std;
00023
00024 const char CRfv[] = "rezultatai.txt";
00025 const char CRfv2[] = "naujas_failas.txt";
00026 const char CRfv3[] = "mokslinciai.txt";
00027 const char CRfv4[] = "nuskriaustieji.txt";
00028
00029 // const char CDfv[] = "kursiokai.txt";
00030 const char CDfv0[] = "studentai10000.txt";
00031 const char CDfv1[] = "studentai100000.txt";
00032 const char CDfv2[] = "studentai1000000.txt";
00033
00034 template<typename T>
00035 class Vector {
00036 private:
```

```
00037     T* data;
00038     size_t _size;
00039     size_t _capacity;
00040 
00041     void reallocate(size_t newCapacity);
00042 
00043     template<typename InputIterator>
00044     void assign_impl(InputIterator first, InputIterator last, std::input_iterator_tag);
00045 
00046     template<typename RandomAccessIterator>
00047     void assign_impl(RandomAccessIterator first, RandomAccessIterator last,
      std::random_access_iterator_tag);
00048 
00049 public:
00050     Vector(size_t size, size_t capacity, const T& defaultValue)
00051     : _size(size), _capacity(capacity) {
00052     data = new T[_capacity];
00053     for (size_t i = 0; i < _size; ++i) {
00054         data[i] = defaultValue;
00055     }
00056 }
00057     Vector();
00058     explicit Vector(size_t initialCapacity);
00059     Vector(size_t size, const T& defaultValue);
00060     Vector(std::initializer_list<T> initList);
00061     Vector(const Vector& other);
00062     Vector(Vector&& other) noexcept;
00063     Vector& operator=(const Vector& other);
00064     Vector& operator=(Vector&& other) noexcept;
00065     using value_type = T;
00066     ~Vector();
00067 
00068     void push_back(const T& value);
00069     void push_back(T&& value);
00070     void pop_back();
00071     size_t size() const;
00072     size_t capacity() const;
00073     bool empty() const;
00074     T& operator[](size_t index);
00075     const T& operator[](size_t index) const;
00076     void clear();
00077     void reserve(size_t newCapacity);
00078     T* begin();
00079     T* end();
00080     const T* begin() const;
00081     const T* end() const;
00082     T& back();
00083     const T& back() const;
00084     T& front();
00085     const T& front() const;
00086     T* data_ptr();
00087     const T* data_ptr() const;
00088     void erase(size_t index);
00089     void resize(size_t newSize, const T& defaultValue = T());
00090     void swap(Vector& other);
00091 
00092     template<typename InputIterator>
00093     void assign(InputIterator first, InputIterator last);
00094     void assign(size_t count, const T& value);
00095     void assign(std::initializer_list<T> ilist);
00096 
00097     void insert(size_t index, const T& value);
00098     template<typename InputIterator>
00099     void insert_range(size_t index, InputIterator first, InputIterator last);
00100     void append_range(std::initializer_list<T> ilist);
00101 
00102     T& at(size_t index);
00103     const T& at(size_t index) const;
00104 
00105     typename std::reverse_iterator<T*> rbegin();
00106     typename std::reverse_iterator<T*> rend();
00107     typename std::reverse_iterator<const T*> rbegin() const;
00108     typename std::reverse_iterator<const T*> rend() const;
00109 
00110     friend bool operator==(const Vector<T>& lhs, const Vector<T>& rhs) {
00111         if (lhs._size != rhs._size) {
00112             return false;
00113         }
00114         for (size_t i = 0; i < lhs._size; ++i) {
00115             if (lhs.data[i] != rhs.data[i]) {
00116                 return false;
00117             }
00118         }
00119         return true;
00120     }
00121 
00122     friend bool operator!=(const Vector<T>& lhs, const Vector<T>& rhs) {
```

```
00123         return !(lhs == rhs);
00124     }
00125
00126     friend bool operator<(const Vector<T>& lhs, const Vector<T>& rhs) {
00127         return std::lexicographical_compare(lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
00128     }
00129
00130     friend bool operator<=(const Vector<T>& lhs, const Vector<T>& rhs) {
00131         return !(rhs < lhs);
00132     }
00133
00134     friend bool operator>(const Vector<T>& lhs, const Vector<T>& rhs) {
00135         return rhs < lhs;
00136     }
00137
00138     friend bool operator>=(const Vector<T>& lhs, const Vector<T>& rhs) {
00139         return !(lhs < rhs);
00140     }
00141 };
00142
00143 template<typename T>
00144 Vector<T>::Vector() : data(nullptr), _size(0), _capacity(0) {}
00145
00146 template<typename T>
00147 Vector<T>::Vector(size_t initialCapacity) : _size(0), _capacity(initialCapacity) {
00148     data = new T[_capacity];
00149 }
00150
00151 template<typename T>
00152 Vector<T>::Vector(size_t size, const T& defaultValue) : _size(size), _capacity(size) {
00153     data = new T[_capacity];
00154     std::fill(data, data + _size, defaultValue);
00155 }
00156
00157 template<typename T>
00158 Vector<T>::Vector(std::initializer_list<T> initList) : _size(initList.size()),
      _capacity(initList.size()) {
00159     data = new T[_capacity];
00160     std::copy(initList.begin(), initList.end(), data);
00161 }
00162
00163 template<typename T>
00164 Vector<T>::Vector(const Vector& other) : _size(other._size), _capacity(other._capacity) {
00165     data = new T[_capacity];
00166     std::copy(other.data, other.data + _size, data);
00167 }
00168
00169 template<typename T>
00170 Vector<T>::Vector(Vector&& other) noexcept : data(other.data), _size(other._size),
      _capacity(other._capacity) {
00171     other.data = nullptr;
00172     other._size = 0;
00173     other._capacity = 0;
00174 }
00175
00176 template<typename T>
00177 Vector<T>& Vector<T>::operator=(const Vector& other) {
00178     if (this != &other) {
00179         T* newData = new T[other._capacity];
00180         std::copy(other.data, other.data + other._size, newData);
00181         delete[] data;
00182         data = newData;
00183         _size = other._size;
00184         _capacity = other._capacity;
00185     }
00186     return *this;
00187 }
00188
00189 template<typename T>
00190 Vector<T>& Vector<T>::operator=(Vector&& other) noexcept {
00191     if (this != &other) {
00192         delete[] data;
00193         data = other.data;
00194         _size = other._size;
00195         _capacity = other._capacity;
00196         other.data = nullptr;
00197         other._size = 0;
00198         other._capacity = 0;
00199     }
00200     return *this;
00201 }
00202
00203 template<typename T>
00204 Vector<T>::~Vector() {
00205     delete[] data;
00206 }
00207
```

```
00208 template<typename T>
00209 void Vector<T>::push_back(const T& value) {
00210     if (_size == _capacity) {
00211         reallocate(_capacity == 0 ? 1 : _capacity * 2);
00212     }
00213     data[_size++] = value;
00214 }
00215
00216 template<typename T>
00217 void Vector<T>::push_back(T&& value) {
00218     if (_size == _capacity) {
00219         reallocate(_capacity == 0 ? 1 : _capacity * 2);
00220     }
00221     data[_size++] = std::move(value);
00222 }
00223
00224 template<typename T>
00225 void Vector<T>::pop_back() {
00226     if (_size > 0) {
00227         --_size;
00228     }
00229 }
00230
00231 template<typename T>
00232 size_t Vector<T>::size() const {
00233     return _size;
00234 }
00235
00236 template<typename T>
00237 size_t Vector<T>::capacity() const {
00238     return _capacity;
00239 }
00240
00241 template<typename T>
00242 bool Vector<T>::empty() const {
00243     return _size == 0;
00244 }
00245
00246 template<typename T>
00247 T& Vector<T>::operator[](size_t index) {
00248     return data[index];
00249 }
00250
00251 template<typename T>
00252 const T& Vector<T>::operator[](size_t index) const {
00253     return data[index];
00254 }
00255
00256 template<typename T>
00257 void Vector<T>::clear() {
00258     _size = 0;
00259 }
00260
00261 template<typename T>
00262 void Vector<T>::reserve(size_t newCapacity) {
00263     if (newCapacity > _capacity) {
00264         reallocate(newCapacity);
00265     }
00266 }
00267
00268 template<typename T>
00269 T* Vector<T>::begin() {
00270     return data;
00271 }
00272
00273 template<typename T>
00274 T* Vector<T>::end() {
00275     return data + _size;
00276 }
00277
00278 template<typename T>
00279 const T* Vector<T>::begin() const {
00280     return data;
00281 }
00282
00283 template<typename T>
00284 const T* Vector<T>::end() const {
00285     return data + _size;
00286 }
00287
00288 template<typename T>
00289 T& Vector<T>::back() {
00290     if (_size == 0) {
00291         throw std::out_of_range("Kvieciamas back() tusciam vektoriui");
00292     }
00293     return data[_size - 1];
00294 }
```

```
00295
00296 template<typename T>
00297 const T& Vector<T>::back() const {
00298     if (_size == 0) {
00299         throw std::out_of_range("Kvieciamas back() tusciam vektoriui");
00300     }
00301     return data[_size - 1];
00302 }
00303
00304 template<typename T>
00305 T& Vector<T>::front() {
00306     if (_size == 0) {
00307         throw std::out_of_range("Kvieciamas front() tusciam vektoriui");
00308     }
00309     return data[0];
00310 }
00311
00312 template<typename T>
00313 const T& Vector<T>::front() const {
00314     if (_size == 0) {
00315         throw std::out_of_range("Kvieciamas front() tusciam vektoriui");
00316     }
00317     return data[0];
00318 }
00319
00320 template<typename T>
00321 T* Vector<T>::data_ptr() {
00322     return data;
00323 }
00324
00325 template<typename T>
00326 const T* Vector<T>::data_ptr() const {
00327     return data;
00328 }
00329
00330 template<typename T>
00331 void Vector<T>::erase(size_t index) {
00332     if (index < _size) {
00333         std::move(data + index + 1, data + _size, data + index);
00334         --_size;
00335     }
00336 }
00337
00338 template<typename T>
00339 void Vector<T>::resize(size_t newSize, const T& defaultValue) {
00340     if (newSize > _capacity) {
00341         reallocate(newSize);
00342     }
00343     if (newSize > _size) {
00344         std::fill(data + _size, data + newSize, defaultValue);
00345     }
00346     _size = newSize;
00347 }
00348
00349 template<typename T>
00350 void Vector<T>::swap(Vector& other) {
00351     std::swap(data, other.data);
00352     std::swap(_size, other._size);
00353     std::swap(_capacity, other._capacity);
00354 }
00355
00356 template<typename T>
00357 template<typename InputIterator>
00358 void Vector<T>::assign(InputIterator first, InputIterator last) {
00359     using category = typename std::iterator_traits<InputIterator>::iterator_category;
00360     assign_impl(first, last, category());
00361 }
00362
00363 template<typename T>
00364 template<typename InputIterator>
00365 void Vector<T>::assign_impl(InputIterator first, InputIterator last, std::input_iterator_tag) {
00366     clear();
00367     for (; first != last; ++first) {
00368         push_back(*first);
00369     }
00370 }
00371
00372 template<typename T>
00373 template<typename RandomAccessIterator>
00374 void Vector<T>::assign_impl(RandomAccessIterator first, RandomAccessIterator last,
      std::random_access_iterator_tag) {
00375     size_t newSize = std::distance(first, last);
00376     if (newSize > _capacity) {
00377         reallocate(newSize);
00378     }
00379     std::copy(first, last, data);
00380     _size = newSize;
```

```
00381 }
00382
00383 template<typename T>
00384 void Vector<T>::assign(std::initializer_list<T> ilist) {
00385     size_t newSize = ilist.size();
00386     if (newSize > _capacity) {
00387         reallocate(newSize);
00388     }
00389     std::copy(ilist.begin(), ilist.end(), data);
00390     _size = newSize;
00391 }
00392
00393 template<typename T>
00394 void Vector<T>::assign(size_t count, const T& value) {
00395     size_t newSize = count;
00396     if (newSize > _capacity) {
00397         reallocate(newSize);
00398     }
00399     std::fill_n(data, count, value);
00400     _size = count;
00401 }
00402
00403 template<typename T>
00404 void Vector<T>::insert(size_t index, const T& value) {
00405     if (_size == _capacity) {
00406         reallocate(_capacity == 0 ? 1 : _capacity * 2);
00407     }
00408     if (index < _size) {
00409         std::move_backward(data + index, data + _size, data + _size + 1);
00410     }
00411     data[index] = value;
00412     ++_size;
00413 }
00414
00415 template<typename T>
00416 template<typename InputIterator>
00417 void Vector<T>::insert_range(size_t index, InputIterator first, InputIterator last) {
00418     size_t count = std::distance(first, last);
00419     if (_size + count > _capacity) {
00420         reallocate(_size + count);
00421     }
00422     if (index < _size) {
00423         std::move_backward(data + index, data + _size, data + _size + count);
00424     }
00425     std::copy(first, last, data + index);
00426     _size += count;
00427 }
00428
00429 template<typename T>
00430 void Vector<T>::append_range(std::initializer_list<T> ilist) {
00431     if (_size + ilist.size() > _capacity) {
00432         reallocate(_size + ilist.size());
00433     }
00434     std::copy(ilist.begin(), ilist.end(), data + _size);
00435     _size += ilist.size();
00436 }
00437
00438 template<typename T>
00439 T& Vector<T>::at(size_t index) {
00440     if (index >= _size) {
00441         throw std::out_of_range("Index out of range");
00442     }
00443     return data[index];
00444 }
00445
00446 template<typename T>
00447 const T& Vector<T>::at(size_t index) const {
00448     if (index >= _size) {
00449         throw std::out_of_range("Index out of range");
00450     }
00451     return data[index];
00452 }
00453
00454 template<typename T>
00455 typename std::reverse_iterator<T*> Vector<T>::rbegin() {
00456     return std::reverse_iterator<T*>(end());
00457 }
00458
00459 template<typename T>
00460 typename std::reverse_iterator<T*> Vector<T>::rend() {
00461     return std::reverse_iterator<T*>(begin());
00462 }
00463
00464 template<typename T>
00465 typename std::reverse_iterator<const T*> Vector<T>::rbegin() const {
00466     return std::reverse_iterator<const T*>(end());
00467 }
```

```
00468
00469 template<typename T>
00470 typename std::reverse_iterator<const T*> Vector<T>::rend() const {
00471     return std::reverse_iterator<const T*>(begin());
00472 }
00473
00474 template<typename T>
00475 void Vector<T>::reallocate(size_t newCapacity) {
00476     T* newData = new T[newCapacity];
00477     if (data) {
00478         std::move(data, data + _size, newData);
00479         delete[] data;
00480     }
00481     data = newData;
00482     _capacity = newCapacity;
00483 }
00484
00485 // Bazine ir Derived klases
00486 class Zmogus
00487 {
00488 protected:
00489     string vardas;
00490     string pavarde;
00491
00492 public:
00493     Zmogus(string vard = "", string pav = "") : vardas(move(vard)), pavarde(move(pav)) {}
00494     virtual ~Zmogus() = default;
00495     virtual void patikrinimas() const = 0;
00496 };
00497
00498 class Mokinys : public Zmogus
00499 {
00500 private:
00501     /* string vardas;
00502     string pavarde; */
00503     Vector<int> ND;
00504     int egzaminas;
00505     double VID;
00506     double MED;
00507
00508 public:
00509     // Constructor
00510     Mokinys(string vard = "", string pav = "", Vector<int> nd = {}, int e = 0, double vid = 0.0,
    double med = 0.0)
00511         : Zmogus(move(vard), move(pav)), ND(nd), egzaminas(e), VID(vid), MED(med) {}
00512
00513     // Destructor
00514     ~Mokinys() = default;
00515
00516     void patikrinimas() const override{};
00517
00518     // Copy constructor
00519     Mokinys(const Mokinys &other) : Zmogus(other), ND(other.ND), egzaminas(other.egzaminas),
    VID(other.VID), MED(other.MED) {}
00520
00521     // Move contructor
00522     Mokinys(Mokinys &&other) noexcept
00523         : Zmogus((move(other.vardas)), (move(other.pavarde))),
00524           ND(move(other.ND)), egzaminas(exchange(other.egzaminas, 0)),
00525           VID(exchange(other.VID, 0)), MED(exchange(other.MED, 0)) {}
00526
00527     // Copy Assignment Operator
00528     Mokinys &operator=(const Mokinys &other)
00529     {
00530         Zmogus::operator=(other);
00531         ND = other.ND;
00532         egzaminas = other.egzaminas;
00533         VID = other.VID;
00534         MED = other.MED;
00535         return *this;
00536     }
00537
00538     void clear() {
00539         vardas.clear();
00540         pavarde.clear();
00541         ND.clear();
00542         egzaminas = 0;
00543         VID = 0;
00544         MED = 0;
00545     }
00546
00547     // Move Assignment Operator
00548         Mokinys& operator=(Mokinys&& other) noexcept {
00549         if (this != &other) {
00550             // Copy data from 'other' to 'this'
00551             vardas = std::move(other.vardas);
00552             pavarde = std::move(other.pavarde);
```

```
00553             ND = std::move(other.ND);
00554             egzaminas = other.egzaminas;
00555             VID = other.VID;
00556             MED = other.MED;
00557 
00558             // Clear 'other'
00559             other.clear();
00560         }
00561         return *this;
00562     }
00563 
00564 
00565     friend std::ostream &operator«(std::ostream &fr, const Mokinys &temp1)
00566     {
00567         fr « "Vardas: " « temp1.vardas « endl;
00568         fr « "Pavarde: " « temp1.pavarde « endl;
00569         fr « "Namu darbai: ";
00570         for (int pazymys : temp1.ND)
00571         {
00572             fr « pazymys « " ";
00573         }
00574         cout « endl;
00575         fr « "Egzamino pazymys: " « temp1.egzaminas « endl;
00576         fr « "Mediana: " « temp1.MED « endl;
00577         fr « "Vidurkis: " « temp1.VID « endl;
00578         return fr;
00579     }
00580 
00581     friend istream &operator»(istream &fd, Mokinys &temp1)
00582     {
00583         cout « "Iveskite varda: ";
00584         fd » temp1.vardas;
00585         cout « "Iveskite pavarde: ";
00586         fd » temp1.pavarde;
00587         cout « "Iveskite namu darbus: ";
00588         int pazymys;
00589         temp1.ND.clear();
00590         while (fd » pazymys && pazymys != 0)
00591         {
00592             temp1.ND.push_back(pazymys);
00593         }
00594         cout « "Iveskite egzamino pazymi: ";
00595         fd » temp1.egzaminas;
00596         cout « "Iveskite mediana: ";
00597         fd » temp1.MED;
00598         cout « "Iveskite vidurki: ";
00599         fd » temp1.VID;
00600         return fd;
00601     }
00602 
00603 
00604     // Getter functions
00605     string getVardas() const { return vardas; }
00606     string getPavarde() const { return pavarde; }
00607     Vector<int> getND() const { return ND; }
00608     int getEgzaminas() const { return egzaminas; }
00609     double getVID() const { return VID; }
00610     double getMED() const { return MED; }
00611 
00612     // Setter functions
00613     void setVardas(const string &name) { vardas = name; }
00614     void setPavarde(const string &surname) { pavarde = surname; }
00615     void addND(int nd) { ND.push_back(nd); }
00616     void clearND() { ND.clear(); }
00617     void setEgzaminas(int exam) { egzaminas = exam; }
00618     void setVID(double vid) { VID = vid; }
00619     void setMED(double med) { MED = med; }
00620 
00621     // Utility functions
00622     void Vidurkis(Vector<Mokinys> &A);
00623     void Isvedimas(const Vector<Mokinys> &A, int MOK_kiekis, string isvedimas);
00624     void Isvedimas2(const Vector<Mokinys> &A, int MOK_kiekis, string isvedimas);
00625     static bool PagalVidurki(const Mokinys &a, const Mokinys &b);
00626     static bool PagalMediana(const Mokinys &a, const Mokinys &b);
00627     static bool PagalVarda(const Mokinys &a, const Mokinys &b);
00628     static bool PagalPavarde(const Mokinys &a, const Mokinys &b);
00629     void Skaitymas(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai, Vector<int>
    &IrasuSk, string failas, Vector<Mokinys> &A, int &temp, char strategija);
00630     void StudentuRusiavimas(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai,
    Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
00631     void StudentuRusiavimas2(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai,
    Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
00632     void StudentuRusiavimas3(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai,
    Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
00633     void Rikiavimas(Vector<Mokinys> &Mokslinciai, Vector<Mokinys> &Nuskriaustieji, Vector<int>
    &IrasuSk);
00634 };
```

```
00635
00636 #endif
```

Index