v3.0

Beta

Generated by Doxygen 1.10.0

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

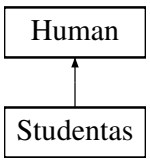
Class Documentation

4.1 Human Class Reference

Bazinė klasė Human reprezentuoja žmogų.

`#include <zmogus.h>`

Inheritance diagram for Human:



Public Member Functions

- virtual ~**Human** ()=default

  Virtualus destruktorius.
- virtual void setVardas (const std::string &vardas)=0

  Nustato žmogaus vardą.
- virtual std::string getVardas () const =0

  Gauna žmogaus vardą.
- virtual void setPavarde (const std::string &pavarde)=0

  Nustato žmogaus pavardę.
- virtual std::string getPavarde () const =0

  Gauna žmogaus pavardę.

4.1.1 Detailed Description

Bazinė klasė Human reprezentuoja žmogų.

Ši klasė apibrėžia abstrakčius metodus, kurie turi būti implementuoti vaikų klasėse.

4.1.2 Member Function Documentation

4.1.2.1 getPavarde()

```
virtual std::string Human::getPavarde ( ) const  [pure virtual]
```

Gauna žmogaus pavardę.

Returns

Žmogaus pavardė.

Implemented in Studentas.

4.1.2.2 getVardas()

```
virtual std::string Human::getVardas ( ) const  [pure virtual]
```

Gauna žmogaus vardą.

Returns

Žmogaus vardas.

Implemented in Studentas.

4.1.2.3 setPavarde()

```
virtual void Human::setPavarde (
            const std::string & pavarde )  [pure virtual]
```

Nustato žmogaus pavardę.

Parameters

pavarde	Nustatoma pavardė.

Implemented in Studentas.

4.1.2.4 setVardas()

```
virtual void Human::setVardas (
            const std::string & vardas )  [pure virtual]
```

Nustato žmogaus vardą.

Parameters

vardas	Nustatomas vardas.

Implemented in Studentas.

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

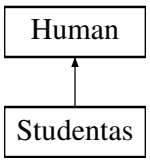
- zmogus.h

4.2 Studentas Class Reference

Klasė Studentas reprezentuoja studentą.

`#include <studentas.h>`

Inheritance diagram for Studentas:



Public Member Functions

- **Studentas** ()
  Numatytasis konstruktorius.
- Studentas (const std::string &vardas, const std::string &pavarde)
  Konstruktorius su parametrais vardui ir pavardėi.
- **Studentas** (const Studentas &other)
  Kopijavimo konstruktorius.
- Studentas & **operator=** (const Studentas &other)
  Priskyrimo operatorius.
- **Studentas** (Studentas &&other) noexcept
  Perkėlimo konstruktorius.
- Studentas & **operator=** (Studentas &&other) noexcept
  Perkėlimo priskyrimo operatorius.
- ~**Studentas** () override
  Destruktorius.
- void setVardas (const std::string &vardas) override
  Nustato studento vardą.
- std::string getVardas () const override
  Gauna studento vardą.
- void setPavarde (const std::string &pavarde) override
  Nustato studento pavardę.
- std::string getPavarde () const override
  Gauna studento pavardę.
- void **setNamuDarbai** (const std::vector< int > &nd)

Nustato studento namų darbų įvertinimus.

- std::vector< int > **getNamuDarbai** () const

  Gauna studento namų darbų įvertinimus.
- void **addNamuDarbas** (int pazymys)

  Prideda namų darbų įvertinimą studentui.
- void **setEgzaminas** (int egzaminas)

  Nustato studento egzamino įvertinimą.
- int **getEgzaminas** () const

  Gauna studento egzamino įvertinimą.
- double **skaiciuotiVidurki** () const

  Skaičiuoja studento vidurkį.
- double **skaiciuotiMediana** () const

  Skaičiuoja studento medianą.
- double skaiciuotiGalutini (bool naudotiVidurki) const

  Skaičiuoja studento galutinį įvertinimą.
- void **atsitiktiniai** ()

  Sugeneruoja atsitiktinius įvertinimus studento namų darbams ir egzaminui.
- void **atsitiktiniaiStudentai** ()

  Sugeneruoja atsitiktinius įvertinimus keliems studentų namų darbams ir egzaminams.

Public Member Functions inherited from Human

- virtual ~**Human** ()=default

  Virtualus destruktorius.

Friends

- std::ostream & operator<< (std::ostream &os, const Studentas &student)

  Išvesti studento duomenis į srautą.
- std::istream & operator>> (std::istream &is, Studentas &student)

  Įvesti studento duomenis iš srauto.

4.2.1 Detailed Description

Klasė Studentas reprezentuoja studentą.

Ši klasė paveldi funkcijas iš žmogaus klasės ir prideda funkcionalumą, specifinį studentams, kaip namų darbų ir egzaminų įvertinimų valdymas.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Studentas()

```
Studentas::Studentas (
            const std::string & vardas,
            const std::string & pavarde )
```

Konstruktorius su parametrais vardui ir pavardėi.

Parameters

vardas	Studento vardas.
pavarde	Studento pavardė.

4.2.3 Member Function Documentation

4.2.3.1 getPavarde()

```
std::string Studentas::getPavarde ( ) const  [override], [virtual]
```

Gauna studento pavardę.

Implements Human.

4.2.3.2 getVardas()

```
std::string Studentas::getVardas ( ) const  [override], [virtual]
```

Gauna studento vardą.

Implements Human.

4.2.3.3 setPavarde()

```
void Studentas::setPavarde (
            const std::string & pavarde )  [override], [virtual]
```

Nustato studento pavardę.

Implements Human.

4.2.3.4 setVardas()

```
void Studentas::setVardas (
            const std::string & vardas )  [override], [virtual]
```

Nustato studento vardą.

Implements Human.

4.2.3.5 skaiciuotiGalutini()

```
double Studentas::skaiciuotiGalutini (
            bool naudotiVidurki ) const
```

Skaičiuoja studento galutinį įvertinimą.

Parameters

naudotiVidurki	Ar naudoti vidurkį (true) ar medianą (false).

4.2.4 Friends And Related Symbol Documentation

4.2.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Studentas & student )  [friend]
```

Išvesti studento duomenis į srautą.

Parameters

os	Išvesties srautas.
student	Studento objektas, kurio duomenys išvedami.

Returns

Išvesties srauto nuoroda.

4.2.4.2 operator>>

```
std::istream & operator>> (
            std::istream & is,
            Studentas & student )  [friend]
```

Įvesti studento duomenis iš srauto.

Parameters

is	Įvesties srautas.
student	Studento objektas, į kurį įvedami duomenys.

Returns

Įvesties srauto nuoroda.

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

- studentas.h
- studentas.cpp

4.3 Vector< T > Class Template Reference

Dinaminio masyvo realizacija, panaši į std::vector.

`#include <vector.h>`

Public Types

- typedef T **value_type**

  Saugomų elementų tipas vektoriuje.
- typedef T & **reference**

  Nuoroda į vektoriaus elementą.
- typedef const T & **const_reference**

  Konstantinė nuoroda į vektoriaus elementą.
- typedef T * **iterator**

  Iteratorius, skirtas peržiūrėti vektoriaus elementus.
- typedef const T * **const_iterator**

  Konstantinė iteratoriaus versija, skirta peržiūrėti vektoriaus elementus.
- typedef std::ptrdiff_t **difference_type**

  Ženklintas sveikasis skaičius, kuris nurodo skirtumą tarp dviejų iteratorių.
- typedef size_t **size_type**

  Nenulinis sveikasis skaičius, nurodantis vektoriaus dydį.

Public Member Functions

- **Vector** ()

  Sukuria tuščią vektorių.
- ~**Vector** ()

  Sunaikina vektorių, atlaisvindamas dinamiškai priskirtą atmintį.
- Vector (const Vector &other)

  Sukuria vektorių su kitų vektoriaus turiniu.
- Vector & operator= (const Vector &other)

  Priskiria kito vektoriaus turinį šiam vektoriui.
- Vector (Vector &&other) noexcept

  Sukuria vektorių perkeliant kitų vektoriaus turinį.
- Vector & operator= (Vector &&other) noexcept

  Priskiria kito vektoriaus turinį šiam vektoriui perkeliant.
- void push_back (const T &value)

  Prideda elementą į vektoriaus galą.
- void pop_back ()

  Pašalinti paskutinį elementą.
- void **clear** ()

  Išvalo vektorių, pašalindamas visus jo elementus.
- void reserve (size_t new_capacity)

  Rezervuoja vietą vektoriuje tam tikram elemento skaičiui.
- size_t capacity () const

  Gražina vektoriaus talpumą.
- void resize (size_t new_size)

  Keičia vektoriaus dydį.

- size_t size () const

  Gražina vektoriaus dydį.
- bool empty () const

  Patikrina, ar vektorius yra tuščias.
- T & operator[ ] (size_t index)

  Pasiekia elementą pagal indeksą naudodami [] operatorių.
- const T & operator[ ] (size_t index) const

  Pasiekia elementą pagal indeksą naudodami [] operatorių (konstantinė versija).
- T * begin ()

  Gražina iteratorių į vektoriaus pradžią.
- const T * begin () const

  Gražina konstantinį iteratorių į vektoriaus pradžią.
- T * end ()

  Gražina iteratorių į vektoriaus pabaigą.
- const T * end () const

  Gražina konstantinį iteratorių į vektoriaus pabaigą.
- T & front ()

  Pasiekia vektoriaus pirmąjį elementą.
- const T & front () const

  Pasiekia vektoriaus pirmąjį elementą (konstantinė versija).
- T & back ()

  Pasiekia vektoriaus paskutinį elementą.
- const T & back () const

  Pasiekia vektoriaus paskutinį elementą (konstantinė versija).
- T * insert (iterator pos, const T &value)

  Iterpkite elementą į vektorių.
- T * erase (iterator pos)

  Pašalinkite elementą iš vektoriaus.
- T * erase (iterator first, iterator last)

  Pašalinkite elementus iš vektoriaus.
- void **sort** ()

  Surikiuokite vektorių.
- template<typename Compare >
  void sort (Compare comp)

  Surikiuokite vektorių naudodami palyginimo funkciją.
- void **reverse** ()

  Atvirkštinė tvarka vektoriuje.
- T & first_duplicate ()

  Pirmas pasikartojantis elementas vektoriuje.
- template<typename Predicate >
  size_t count_if (Predicate pred)

  Gražina skaičių elementų, kurie atitinka sąlygą.
- template<typename Predicate >
  T & first_duplicate_if (Predicate pred)

  Gražina pirmą pasikartojantį elementą, kuris atitinka sąlygą.
- template<typename Func >
  void sort_by (Func func)

  Surikiuoti vektorių pagal funkcijos rezultatą.
- template<typename Func , typename... Args>
  double benchmark (Func func, Args &&...args)

  Gauti laiką, reikalingą funkcijos vykdymui.

- void rotate (iterator start, iterator middle, iterator end)

  Sukeičia vektoriaus elementų tvarką tarp nurodytų indeksų.
- bool is_sorted () const

  Patikrina, ar vektorius yra surikiuotas didėjančia tvarka.
- size_t index_of (const T &value) const

  Gražina indeksą, kuriame pirmą kartą pasitaiko elementas.
- void swap_elements (size_t index1, size_t index2)

  Sukeičia du elementus pagal jų indeksus.

Friends

- std::ostream & operator<< (std::ostream &out, const Vector &vec)

  Spausdinti vektorių į srautą.

4.3.1 Detailed Description

template<typename T>
class Vector< T >

Dinaminio masyvo realizacija, panaši į std::vector.

Template Parameters

T	Saugomų elementų tipas vektoriuje.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Vector() [1/2]

```
template<typename T >
Vector< T >::Vector (
            const Vector< T > & other )  [inline]
```

Sukuria vektorių su kitų vektoriaus turiniu.

Parameters

other	Kopijuojamas vektorius.

4.3.2.2 Vector() [2/2]

```
template<typename T >
Vector< T >::Vector (
            Vector< T > && other )  [inline], [noexcept]
```

Sukuria vektorių perkeliant kitų vektoriaus turinį.

Parameters

other	Perkeliamas vektorius.

4.3.3 Member Function Documentation

4.3.3.1 back() [1/2]

```
template<typename T >
T & Vector< T >::back ( )  [inline]
```

Pasiekia vektoriaus paskutinį elementą.

Returns

T& Nuoroda į vektoriaus paskutinį elementą.

Exceptions

std::out_of_range	Jei vektorius yra tuščias.

4.3.3.2 back() [2/2]

```
template<typename T >
const T & Vector< T >::back ( ) const  [inline]
```

Pasiekia vektoriaus paskutinį elementą (konstantinė versija).

Returns

const T& Konstantinė nuoroda į vektoriaus paskutinį elementą.

Exceptions

std::out_of_range	Jei vektorius yra tuščias.

4.3.3.3 begin() [1/2]

```
template<typename T >
Vector< T >::iterator Vector< T >::begin ( )  [inline]
```

Gražina iteratorių į vektoriaus pradžią.

Returns

T∗ Iteratorius į vektoriaus pradžią.

4.3.3.4 begin() [2/2]

```
template<typename T >
Vector< T >::const_iterator Vector< T >::begin ( ) const  [inline]
```

Gražina konstantinį iteratorių į vektoriaus pradžią.

Returns

const T* Konstantinis iteratorius į vektoriaus pradžią.

4.3.3.5 benchmark()

```
template<typename T >
template<typename Func , typename...  Args>
double Vector< T >::benchmark (
            Func func,
            Args &&...  args )  [inline]
```

Gauti laiką, reikalingą funkcijos vykdymui.

Template Parameters

Func	Funkcijos tipas.
Args	Funkcijos parametrų tipų aibė.

Parameters

func	Funkcija, kurios vykdymo laikas yra matuojamas.
args	Funkcijos parametrai.

Returns

double Funkcijos vykdymo laikas sekundėmis.

4.3.3.6 capacity()

```
template<typename T >
size_t Vector< T >::capacity ( ) const  [inline]
```

Gražina vektoriaus talpumą.

Returns

size_t Vektoriaus talpumas.

4.3.3.7 count_if()

```
template<typename T >
template<typename Predicate >
size_t Vector< T >::count_if (
            Predicate pred )  [inline]
```

Gražina skaičių elementų, kurie atitinka sąlygą.

Parameters

pred	Sąlygos funkcija.

Returns

size_t Skaičius elementų, kurie atitinka sąlygą.

4.3.3.8 empty()

```
template<typename T >
bool Vector< T >::empty ( ) const  [inline]
```

Patikrina, ar vektorius yra tuščias.

Returns

true, jei vektorius tuščias, false, jei vektorius netuščias.

4.3.3.9 end() [1/2]

```
template<typename T >
Vector< T >::iterator Vector< T >::end ( )  [inline]
```

Gražina iteratorių į vektoriaus pabaigą.

Returns

T∗ Iteratorius į vektoriaus pabaigą.

4.3.3.10 end() [2/2]

```
template<typename T >
Vector< T >::const_iterator Vector< T >::end ( ) const  [inline]
```

Gražina konstantinį iteratorių į vektoriaus pabaigą.

Returns

const T∗ Konstantinis iteratorius į vektoriaus pabaigą.

4.3.3.11 erase() [1/2]

```
template<typename T >
T * Vector< T >::erase (
            iterator first,
            iterator last )  [inline]
```

Pašalinkite elementus iš vektoriaus.

Parameters

first	Iteratorius, rodantis į pirmąjį ištrinamą elementą.
last	Iteratorius, rodantis į vieną elementą už paskutinio ištrinamojo.

Returns

T* Iteratorius, rodantis į vietą, į kurią buvo perkeltas iteratorius.

4.3.3.12 erase() [2/2]

```
template<typename T >
T * Vector< T >::erase (
            iterator pos )  [inline]
```

Pašalinkite elementą iš vektoriaus.

Parameters

pos	Iteratorius, rodantis į vietą, iš kurios norima pašalinti elementą.

Returns

T* Iteratorius, rodantis į vietą, į kurią buvo perkeltas iteratorius.

4.3.3.13 first_duplicate()

```
template<typename T >
T & Vector< T >::first_duplicate ( )  [inline]
```

Pirmas pasikartojantis elementas vektoriuje.

Returns

T& Nuoroda į pirmą pasikartojantį elementą vektoriuje.

Exceptions

std::logic_error	Jei vektorius neturi pasikartojančių elementų.

4.3.3.14 first_duplicate_if()

```
template<typename T >
template<typename Predicate >
T & Vector< T >::first_duplicate_if (
            Predicate pred )  [inline]
```

Gražina pirmą pasikartojantį elementą, kuris atitinka sąlygą.

Parameters

pred	Sąlygos funkcija.

Returns

T& Nuoroda į pirmą pasikartojantį elementą, kuris atitinka sąlygą.

Exceptions

std::logic_error	Jei vektorius neturi pasikartojančio elemento, kuris atitinka sąlygą.

4.3.3.15 front() [1/2]

```
template<typename T >
T & Vector< T >::front ( )  [inline]
```

Pasiekia vektoriaus pirmąjį elementą.

Returns

T& Nuoroda į vektoriaus pirmąjį elementą.

Exceptions

std::out_of_range	Jei vektorius yra tuščias.

4.3.3.16 front() [2/2]

```
template<typename T >
const T & Vector< T >::front ( ) const  [inline]
```

Pasiekia vektoriaus pirmąjį elementą (konstantinė versija).

Returns

const T& Konstantinė nuoroda į vektoriaus pirmąjį elementą.

Exceptions

std::out_of_range	Jei vektorius yra tuščias.

4.3.3.17 index_of()

```
template<typename T >
size_t Vector< T >::index_of (
            const T & value ) const  [inline]
```

Gražina indeksą, kuriame pirmą kartą pasitaiko elementas.

Parameters

value	Ieškomas elementas.

Returns

size_t Indeksas, kuriame pirmą kartą pasitaiko elementas.

Exceptions

std::out_of_range	Jei elementas neegzistuoja vektoriuje.

4.3.3.18 insert()

```
template<typename T >
T * Vector< T >::insert (
            iterator pos,
            const T & value )  [inline]
```

Iterpkite elementą į vektorių.

Parameters

pos	Iteratorius, rodantis į vietą, į kurią norima įterpti elementą.
value	Įterpiamas elementas.

Returns

T∗ Iteratorius, rodantis į įterpto elemento poziciją.

4.3.3.19 is_sorted()

```
template<typename T >
bool Vector< T >::is_sorted ( ) const  [inline]
```

Patikrina, ar vektorius yra surikiuotas didėjančia tvarka.

Returns

true, jei vektorius yra surikiuotas didėjančia tvarka, false, jei ne.

4.3.3.20 operator=() [1/2]

```
template<typename T >
Vector & Vector< T >::operator= (
            const Vector< T > & other )  [inline]
```

Priskiria kito vektoriaus turinį šiam vektoriui.

Parameters

other	Kopijuojamas vektorius.

Returns

Vector& Nuoroda į šį vektorių po priskyrimo.

4.3.3.21 operator=() [2/2]

```
template<typename T >
Vector & Vector< T >::operator= (
            Vector< T > && other )  [inline], [noexcept]
```

Priskiria kito vektoriaus turinį šiam vektoriui perkeliant.

Parameters

other	Perkeliamas vektorius.

Returns

Vector& Nuoroda į šį vektorių po perkelimo priskyrimo.

4.3.3.22 operator[]() [1/2]

```
template<typename T >
T & Vector< T >::operator[] (
            size_t index )  [inline]
```

Pasiekia elementą pagal indeksą naudodami [] operatorių.

Parameters

index	Indeksas, pagal kurį norima pasiekti elementą.

Returns

T& Nuoroda į elementą pagal nurodytą indeksą.

Exceptions

std::out_of_range	Jei indeksas nepatenka į ribas.

4.3.3.23 operator[]() [2/2]

```
template<typename T >
const T & Vector< T >::operator[] (
            size_t index ) const  [inline]
```

Pasiekia elementą pagal indeksą naudodami [] operatorių (konstantinė versija).

Parameters

index	Indeksas, pagal kurį norima pasiekti elementą.

Returns

const T& Konstantinė nuoroda į elementą pagal nurodytą indeksą.

Exceptions

std::out_of_range	Jei indeksas nepatenka į ribas.

4.3.3.24 pop_back()

```
template<typename T >
void Vector< T >::pop_back ( )  [inline]
```

Pašalinti paskutinį elementą.

Exceptions

std::logic_error	Jei vektorius yra tuščias.

4.3.3.25 push_back()

```
template<typename T >
void Vector< T >::push_back (
            const T & value )  [inline]
```

Prideda elementą į vektoriaus galą.

Jei vektorius pilnas, jo talpumas padidinamas dvigubai prieš pridedant elementą.

Parameters

value	Pridedamas elemento vertė.

4.3.3.26 reserve()

```
template<typename T >
void Vector< T >::reserve (
            size_t new_capacity )  [inline]
```

Rezervuoja vietą vektoriuje tam tikram elemento skaičiui.

Parameters

new_capacity	Naujas vektorius talpumas.

4.3.3.27 resize()

```
template<typename T >
void Vector< T >::resize (
            size_t new_size )  [inline]
```

Keičia vektoriaus dydį.

Parameters

new_size	Naujas vektoriaus dydis.

4.3.3.28 rotate()

```
template<typename T >
void Vector< T >::rotate (
            iterator start,
            iterator middle,
            iterator end )  [inline]
```

Sukeičia vektoriaus elementų tvarką tarp nurodytų indeksų.

Parameters

start	Pradinis indeksas.
middle	Vidurinis indeksas.
end	Paskutinis indeksas.

4.3.3.29 size()

```
template<typename T >
size_t Vector< T >::size ( ) const  [inline]
```

Gražina vektoriaus dydį.

Returns

size_t Vektoriaus dydis.

4.3.3.30 sort()

```
template<typename T >
template<typename Compare >
void Vector< T >::sort (
            Compare comp )  [inline]
```

Surikiuokite vektorių naudodami palyginimo funkciją.

Parameters

comp	Funkcija, naudojama lyginant elementus.

4.3.3.31 sort_by()

```
template<typename T >
template<typename Func >
void Vector< T >::sort_by (
            Func func )  [inline]
```

Surikiuoti vektorių pagal funkcijos rezultatą.

Template Parameters

Func	Funkcijos tipas, kuris priima vektoriaus elemento tipą ir grąžina bool reikšmę.

Parameters

func	Funkcija, kurios rezultatas naudojamas rikiuojant elementus.

4.3.3.32 swap_elements()

```
template<typename T >
void Vector< T >::swap_elements (
            size_t index1,
            size_t index2 )  [inline]
```

Sukeičia du elementus pagal jų indeksus.

Parameters

index1	Indeksas pirmam elementui.
index2	Indeksas antram elementui.

4.3.4 Friends And Related Symbol Documentation

4.3.4.1 operator<<

```
template<typename T >
std::ostream & operator<< (
            std::ostream & out,
            const Vector< T > & vec )  [friend]
```

Spausdinti vektorių į srautą.

Parameters

out	Srautas, į kurį spausdinamas vektorius.
vec	Spausdinamas vektorius.

Returns

std::ostream& Nuoroda į srautą, į kurį spausdinama.

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

- vector.h
- vector.cpp

Chapter 5

File Documentation

5.1 app.h

```
00001 #ifndef FUNCTIONS_H
00002 #define FUNCTIONS_H
00003
00004 #include <vector>
00005 #include <string>
00006 #include "studentas.h"
00007 #include "funkcijos.h"
00008
00012 enum class ContainerType { None, Vector };
00013
00017 enum class Action { None, Generate, Sort };
00018
00024 ContainerType getContainerChoice();
00025
00031 Action getActionChoice();
00032
00040 void performAction(ContainerType containerChoice, Action actionChoice, const std::vector<int>& sizes);
00041
00045 void runApp();
00046
00047 #endif // FUNCTIONS_H
```

5.2 funkcijos.h

```
00001 #ifndef FUNKCIJOSVECTOR_H
00002 #define FUNKCIJOSVECTOR_H
00003
00004 #include "studentas.h"
00005 #include <vector>
00006
00007 void manualInput(std::vector<Studentas> &studentai);
00008 void spausdintiGalutiniusBalus(const std::vector<Studentas> &studentai, const std::string
      &isvedimoFailoVardas, int rusiavimoTipas);
00009 void generateGradesOnly(std::vector<Studentas> &studentai);
00010 void readFileDataFromFile(std::vector<Studentas> &studentai, const std::string &failoVardas);
00011
00012
00019 void readDataVector(std::vector<Studentas>& studentai, const std::string& failoVardas);
00020
00026 void generateStudentFilesVector(int size);
00027
00033 void rusiuotStudentusVector(const std::string& failoVardas);
00034
00040 void rusiuotStudentusVector2(const std::string& failoVardas);
00041
00047 void rusiuotStudentusVector3(const std::string& failoVardas);
00048
00049 #endif // FUNKCIJOSVECTOR_H
```

5.3 funkcijosVECTOR.h

```
00001 #ifndef FUNKCIJOSVECTOR_H
00002 #define FUNKCIJOSVECTOR_H
00003
00004 #include "studentas.h"
00005 #include <vector>
00006
00013 void readDataVector(std::vector<Studentas>& studentai, const std::string& failoVardas);
00014
00020 void generateStudentFilesVector(int size);
00021
00027 void rusiuotStudentusVector(const std::string& failoVardas);
00028
00034 void rusiuotStudentusVector2(const std::string& failoVardas);
00035
00041 void rusiuotStudentusVector3(const std::string& failoVardas);
00042
00043 #endif // FUNKCIJOSVECTOR_H
```

5.4 studentas.h

```
00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
00003
00004 #include "zmogus.h"
00005 #include <vector>
00006 #include <iostream>
00007
00014 class Studentas : public Human {
00015 public:
00019     Studentas();
00020
00027     Studentas(const std::string &vardas, const std::string &pavarde);
00028
00029     // Rule of Five
00033     Studentas(const Studentas &other);                // Kopijavimo konstruktorius
00034
00038     Studentas &operator=(const Studentas &other);     // Priskyrimo operatorius
00039
00043     Studentas(Studentas &&other) noexcept;            // Perkėlimo konstruktorius
00044
00048     Studentas &operator=(Studentas &&other) noexcept; // Perkėlimo priskyrimo operatorius
00049
00050     // Destruktorius
00054     ~Studentas() override;
00055
00056     // Implementacija abstrakčių klasės funkcijų
00060     void setVardas(const std::string &vardas) override;
00061
00065     std::string getVardas() const override;
00066
00070     void setPavarde(const std::string &pavarde) override;
00071
00075     std::string getPavarde() const override;
00076
00080     void setNamuDarbai(const std::vector<int> &nd);
00081
00085     std::vector<int> getNamuDarbai() const;
00086
00090     void addNamuDarbas(int pazymys);
00091
00095     void setEgzaminas(int egzaminas);
00096
00100     int getEgzaminas() const;
00101
00105     double skaiciuotiVidurki() const;
00106
00110     double skaiciuotiMediana() const;
00111
00117     double skaiciuotiGalutini(bool naudotiVidurki) const;
00118
00122     void atsitiktiniai();
00123
00127     void atsitiktiniaiStudentai();
00128
00136     friend std::ostream &operator<<(std::ostream &os, const Studentas &student);
00137
00145     friend std::istream &operator>>(std::istream &is, Studentas &student);
00146
00147
00148 private:
00149     std::string vardas;
```

```
00150     std::string pavarde;
00151     std::vector<int> nd_rezultatai;
00152     int egzaminas;
00153 };
00154
00155 #endif
```

5.5 vector.h

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <iostream>
00005 #include <algorithm> // Include std::swap, std::sort, std::find, std::reverse
00006 #include <sstream>
00007 #include <cassert>
00008 #include <chrono>  // Include for time measurement
00009 #include <iomanip> // Include for output formatting
00010
00016 template <typename T>
00017 class Vector
00018 {
00019 private:
00020     T *m_data;
00021     size_t capacity_;
00022     size_t length;
00023
00024 public:
00025     // Narių tipai
00026     typedef T value_type;
00027     typedef T &reference;
00028     typedef const T &const_reference;
00029     typedef T *iterator;
00030     typedef const T *const_iterator;
00031     typedef std::ptrdiff_t difference_type;
00032     typedef size_t size_type;
00033
00034     // Konstruktorius
00038     Vector() : m_data(nullptr), capacity_(0), length(0) {}
00039
00040     // Dekstruktorius
00044     ~Vector()
00045     {
00046         delete[] m_data;
00047     }
00048
00049     // Kopijavimo konstruktorius
00055     Vector(const Vector &other) : m_data(nullptr), capacity_(0), length(0)
00056     {
00057         *this = other;
00058     }
00059
00060     // Kopijavimo priskyrimo operatorius
00067     Vector &operator=(const Vector &other)
00068     {
00069         if (this != &other)
00070         {
00071             clear();
00072             reserve(other.size());
00073             for (size_t i = 0; i < other.size(); ++i)
00074             {
00075                 push_back(other.m_data[i]);
00076             }
00077         }
00078         return *this;
00079     }
00080
00081     // Perkėlimo konstruktorius
00087     Vector(Vector &&other) noexcept : m_data(nullptr), capacity_(0), length(0)
00088     {
00089         *this = std::move(other);
00090     }
00091
00092     // Perkėlimo priskyrimo operatorius
00099     Vector &operator=(Vector &&other) noexcept
00100     {
00101         if (this != &other)
00102         {
00103             clear();
00104             m_data = other.m_data;
00105             capacity_ = other.capacity_;
00106             length = other.length;
00107             other.m_data = nullptr;
```

```
00108             other.capacity_ = 0;
00109             other.length = 0;
00110         }
00111         return *this;
00112     }
00113
00114     // Pridėti elementą į galą
00122     void push_back(const T &value)
00123     {
00124         if (length >= capacity_)
00125         {
00126             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00127         }
00128         m_data[length++] = value;
00129     }
00130
00131     // Pašalinti paskutinį elementą
00137     void pop_back()
00138     {
00139         if (length == 0)
00140             throw std::logic_error("Vektorius yra tuščias");
00141         --length;
00142     }
00143
00144     // Išvalyti vektorių
00148     void clear()
00149     {
00150         length = 0;
00151     }
00152
00153     // Rezervuoti vietą elementams
00159     void reserve(size_t new_capacity)
00160     {
00161         if (new_capacity > capacity_)
00162         {
00163             T *new_data = new T[new_capacity];
00164             for (size_t i = 0; i < length; ++i)
00165             {
00166                 new_data[i] = std::move(m_data[i]);
00167             }
00168             delete[] m_data;
00169             m_data = new_data;
00170             capacity_ = new_capacity;
00171         }
00172     }
00173
00174     // Gauk vektoriaus talpumą
00180     size_t capacity() const
00181     {
00182         return capacity_;
00183     }
00184
00185     // Pakeisti vektoriaus dydį
00191     void resize(size_t new_size)
00192     {
00193         reserve(new_size);
00194         length = new_size;
00195     }
00196
00197     // Gauk vektoriaus dydį
00203     size_t size() const
00204     {
00205         return length;
00206     }
00207
00208     // Patikrink, ar vektorius yra tuščias
00214     bool empty() const
00215     {
00216         return length == 0;
00217     }
00218
00219     // Pasiekite elementą pagal indeksą naudodami []
00227     T &operator[](size_t index)
00228     {
00229         if (index >= length)
00230             throw std::out_of_range("Indeksas už ribų");
00231         return m_data[index];
00232     }
00233
00234     // Pasiekite elementą pagal indeksą naudodami [] operatorių (konstantinė versija)
00242     const T &operator[](size_t index) const
00243     {
00244         if (index >= length)
00245             throw std::out_of_range("Indeksas už ribų");
00246         return m_data[index];
00247     }
00248
```

```
00249     // Gauk iteratorių į vektoriaus pradžią
00255     T *begin()
00256     {
00257         return m_data;
00258     }
00259
00260     // Gauk konstantinį iteratorių į vektoriaus pradžią
00266     const T *begin() const
00267     {
00268         return m_data;
00269     }
00270
00271     // Gauk iteratorių į vektoriaus pabaigą
00277     T *end()
00278     {
00279         return m_data + length;
00280     }
00281
00282     // Gauk konstantinį iteratorių į vektoriaus pabaigą
00288     const T *end() const
00289     {
00290         return m_data + length;
00291     }
00292
00293     // Pasiekite pirmąjį elementą
00300     T &front()
00301     {
00302         if (length == 0)
00303             throw std::out_of_range("Vektorius yra tuščias");
00304         return m_data[0];
00305     }
00306
00307     // Pasiekite pirmąjį elementą (konstantinė versija)
00314     const T &front() const
00315     {
00316         if (length == 0)
00317             throw std::out_of_range("Vektorius yra tuščias");
00318         return m_data[0];
00319     }
00320
00321     // Pasiekite paskutinį elementą
00328     T &back()
00329     {
00330         if (length == 0)
00331             throw std::out_of_range("Vektorius yra tuščias");
00332         return m_data[length - 1];
00333     }
00334
00335     // Pasiekite paskutinį elementą (konstantinė versija)
00342     const T &back() const
00343     {
00344         if (length == 0)
00345             throw std::out_of_range("Vektorius yra tuščias");
00346         return m_data[length - 1];
00347     }
00348
00349     // Iterpkite elementą į vektorių
00357     T *insert(iterator pos, const T &value)
00358     {
00359         size_t index = pos - begin();
00360         if (length >= capacity_)
00361         {
00362             size_t new_capacity = capacity_ == 0 ? 1 : capacity_ * 2;
00363             reserve(new_capacity);
00364         }
00365         std::move_backward(begin() + index, end(), end() + 1);
00366         m_data[index] = value;
00367         ++length;
00368         return begin() + index;
00369     }
00370
00371     // Pašalinkite elementą iš vektoriaus
00378     T *erase(iterator pos)
00379     {
00380         size_t index = pos - begin();
00381         std::move(begin() + index + 1, end(), begin() + index);
00382         --length;
00383         return begin() + index;
00384     }
00385
00386     // Pašalinkite elementus iš vektoriaus
00394     T *erase(iterator first, iterator last)
00395     {
00396         size_t start = first - begin();
00397         size_t end = last - begin();
00398         std::move(begin() + end, end, begin() + start);
00399         length -= end - start;
```

```
00400         return begin() + start;
00401     }
00402
00403     // Surikiuokite vektorių
00407     void sort()
00408     {
00409         std::sort(begin(), end());
00410     }
00411
00412     // Surikiuokite vektorių naudodami palyginimo funkciją
00418     template <typename Compare>
00419     void sort(Compare comp)
00420     {
00421         std::sort(begin(), end(), comp);
00422     }
00423
00424     // Atvirkštinė tvarka vektoriuje
00428     void reverse()
00429     {
00430         std::reverse(begin(), end());
00431     }
00432
00433     // Pirmas pasikartojantis elementas vektoriuje
00440     T &first_duplicate()
00441     {
00442         for (size_t i = 0; i < length; ++i)
00443         {
00444             for (size_t j = i + 1; j < length; ++j)
00445             {
00446                 if (m_data[i] == m_data[j])
00447                 {
00448                     return m_data[i];
00449                 }
00450             }
00451         }
00452         throw std::logic_error("Vektorius neturi pasikartojančių elementų");
00453     }
00454
00455     // Gauk skaičių elementų, kurie atitinka sąlygą
00462     template <typename Predicate>
00463     size_t count_if(Predicate pred)
00464     {
00465         size_t count = 0;
00466         for (const auto &element : *this)
00467         {
00468             if (pred(element))
00469             {
00470                 ++count;
00471             }
00472         }
00473         return count;
00474     }
00475
00476     // Gauti pirmą pasikartojantį elementą, kuris atitinka sąlygą
00484     template <typename Predicate>
00485     T &first_duplicate_if(Predicate pred)
00486     {
00487         for (size_t i = 0; i < length; ++i)
00488         {
00489             for (size_t j = i + 1; j < length; ++j)
00490             {
00491                 if (m_data[i] == m_data[j] && pred(m_data[i]))
00492                 {
00493                     return m_data[i];
00494                 }
00495             }
00496         }
00497         throw std::logic_error("Vektorius neturi pasikartojančio elemento, kuris atitinka sąlygą");
00498     }
00499
00500     // Surikiuoti vektorių pagal funkcijos rezultatą
00507     template <typename Func>
00508     void sort_by(Func func)
00509     {
00510         std::sort(begin(), end(), [&](const T &a, const T &b)
00511                   { return func(a) < func(b); });
00512     }
00513
00514     // Gauti laiką, reikalingą funkcijos vykdymui
00524     template <typename Func, typename... Args>
00525     double benchmark(Func func, Args &&...args)
00526     {
00527         auto start = std::chrono::high_resolution_clock::now();
00528         func(std::forward<Args>(args)...);
00529         auto end = std::chrono::high_resolution_clock::now();
00530         std::chrono::duration<double> duration = end - start;
00531         return duration.count();
```

```
00532     }
00533
00534     // Spausdinti vektorių į srautą
00542     friend std::ostream &operator«(std::ostream &out, const Vector &vec)
00543     {
00544         out « "[";
00545         if (!vec.empty())
00546         {
00547             out « vec[0];
00548             for (size_t i = 1; i < vec.size(); ++i)
00549             {
00550                 out « ", " « vec[i];
00551             }
00552         }
00553         out « "]";
00554         return out;
00555     }
00556
00557     // Sukurti funkciją, kuri sukeičia vektoriaus elementų tvarką tarp nurodytų indeksų
00565     void rotate(iterator start, iterator middle, iterator end)
00566     {
00567         size_t leftLength = middle - start;
00568         size_t rightLength = end - middle;
00569         Vector<T> temp(leftLength + rightLength); // Temporary vector to store rotated elements
00570
00571         // Copy elements from the left side to temporary vector
00572         for (size_t i = 0; i < leftLength; ++i)
00573         {
00574             temp[i] = std::move(start[i]);
00575         }
00576
00577         // Move elements from the right side to the original vector's left side
00578         for (size_t i = 0; i < rightLength; ++i)
00579         {
00580             start[i] = std::move(middle[i]);
00581         }
00582
00583         // Move elements from the temporary vector to the original vector's right side
00584         for (size_t i = 0; i < leftLength; ++i)
00585         {
00586             start[rightLength + i] = std::move(temp[i]);
00587         }
00588     }
00589
00590     // Patikrinkite, ar vektorius yra surikiuotas didėjančia tvarka
00596     bool is_sorted() const
00597     {
00598         return std::is_sorted(begin(), end());
00599     }
00600
00601     // Gauti indeksą, kuriame pirmą kartą pasitaiko elementas
00609     size_t index_of(const T &value) const
00610     {
00611         const_iterator it = std::find(begin(), end(), value);
00612         if (it == end())
00613         {
00614             throw std::out_of_range("Elementas neegzistuoja vektoriuje");
00615         }
00616         return it - begin();
00617     }
00618
00619     // Sukurti funkciją, kuri sukeičia du elementus pagal jų indeksus
00626     void swap_elements(size_t index1, size_t index2)
00627     {
00628         if (index1 >= length || index2 >= length)
00629         {
00630             throw std::out_of_range("Indeksas už ribų");
00631         }
00632         std::swap(m_data[index1], m_data[index2]);
00633     }
00634 };
00635
00636 #endif // VECTOR_H
```

5.6 zmogus.h

```
00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003
00004 #include <string>
00005 #include <vector>
00006
00012 class Human
```

```
00013 {
00014 public:
00018     virtual ~Human() = default;
00019
00025     virtual void setVardas(const std::string &vardas) = 0;
00026
00032     virtual std::string getVardas() const = 0;
00033
00039     virtual void setPavarde(const std::string &pavarde) = 0;
00040
00046     virtual std::string getPavarde() const = 0;
00047
00048     // Abstraktūs metodai, kurie gali būti įgyvendinti vaikinėse klasėse:
00049     // virtual void setNamuDarbai(const std::vector<int> &nd) = 0;
00050     // virtual std::vector<int> getNamuDarbai() const = 0;
00051     // virtual void addNamuDarbas(int pazymys) = 0;
00052     // virtual void setEgzaminas(int egzaminas) = 0;
00053     // virtual int getEgzaminas() const = 0;
00054     // virtual double skaiciuotiVidurki() const = 0;
00055     // virtual double skaiciuotiMediana() const = 0;
00056     // virtual double skaiciuotiGalutini(bool naudotiVidurki) const = 0;
00057     // virtual void atsitiktiniai() = 0;
00058     // virtual void atsitiktiniaiStudentai() = 0;
00059 };
00060
00061 #endif
```

Index