My Project

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

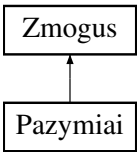
Class Documentation

4.1 Pazymiai Class Reference

Klase Pazymiai saugo informacija apie studentu pazymius.

`#include <pagalbinesvector.h>`

Inheritance diagram for Pazymiai:



Public Member Functions

- **Pazymiai** ()

  Numatytasis konstruktorius, inicializuoja narius nuliais.
- Pazymiai (const std::string &var, const std::string &pav, double vid, int egz, const Vector< int > &paz, double galutinis, double med)

  Parametrizuotas konstruktorius, inicializuoja narius su duotomis reiksmemis.
- ~**Pazymiai** ()

  Desktruktorius, isvalo vektoriu.
- **Pazymiai** (const Pazymiai &other)

  Kopijavimo konstruktorius.
- **Pazymiai** (Pazymiai &&other) noexcept

  Perkelimo konstruktorius.
- Pazymiai & **operator=** (const Pazymiai &other)

  Kopijavimo priskyrimo operatorius.
- Pazymiai & **operator=** (Pazymiai &&other) noexcept

  Perkelimo priskyrimo operatorius.
- int **getPazN** (const Vector< int > &newPaz, int pos) const

  Grazina nurodyta pazymi is saraso pagal pozicija.
- void **clearPaz** ()

  Isvalo vektoriu.

- void **setVid** (double newVid)

  Nustato studento vidurki.
- void **setEgz** (int newEgz)

  Nustato egzamino rezultata.
- void **setOnePaz** (int newPaz)

  Iterpia viena pazymi i sarasa..
- void **setPazymiai** (const Vector< int > &pazymiai)

  Nustato pazymiu sarasa.
- void **setGalutinis** (double newGalutinis)

  Nustato galutini pazymi.
- void **setMed** (double newMed)

  Nustato mediana.
- void **sortPaz** (Pazymiai &C)

  Rikiuoja pazymiu sarasa.
- double **getVid** () const

  Grazina vidurki.
- int **getEgz** () const

  Grazina egzamino rezultata.
- Vector< int > **getPaz** () const

  Grazina pazymiu sarasa.
- double **getGalutinis** () const

  Grazina galutini pazymi.
- double **getMed** () const

  Grazina mediana.
- std::string getVar () const override

  Grazina zmogaus varda.
- std::string getPav () override

  Grazina zmogaus pavarde.

Public Member Functions inherited from Zmogus

- **Zmogus** ()=default

  Numatytasis konstruktorius.
- Zmogus (const std::string &var, const std::string &pav)

  Parametrizuotas konstruktorius, inicializuoja varda ir pavarde.
- virtual ~**Zmogus** ()

  Virtualus destruktorius.
- virtual void **setVar** (const std::string &newVar)

  Nustato studento varda.
- virtual void **setPav** (const std::string &newPav)

  Nustato studento pavarde.

Private Attributes

- double vid_
- int egz_
- Vector< int > paz_
- double galutinis_
- double med_

Friends

- double mediana (int u, const Pazymiai h)

  Skaiciuoja mediana.
- std::istream & operator>> (std::istream &is, Pazymiai &obj)

  Ivedimo operatorius, skirtas nuskaityti objekto duomenis is ivesties srauto.
- std::ostream & operator<< (std::ostream &os, const Pazymiai &obj)

  Isvedimo operatorius, skirtas isvesti objekto duomenis i isvesties srauta.

Additional Inherited Members

Protected Attributes inherited from Zmogus

- std::string var_
- std::string pav_

4.1.1 Detailed Description

Klase Pazymiai saugo informacija apie studentu pazymius.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Pazymiai()

```
Pazymiai::Pazymiai (
            const std::string & var,
            const std::string & pav,
            double vid,
            int egz,
            const Vector< int > & paz,
            double galutinis,
            double med )  [inline]
```

Parametrizuotas konstruktorius, inicializuoja narius su duotomis reiksmemis.

Parameters

var	Studento vardas
pav	Studento pavarde
vid	Vidurkis
egz	Egzamino rezultatas
paz	Pazymiu sarasas
galutinis	Galutinis pazymys
med	Mediana

4.1.3 Member Function Documentation

4.1.3.1 getPav()

```
std::string Pazymiai::getPav ( )  [inline], [override], [virtual]
```

Grazina zmogaus pavarde.

Implements Zmogus.

4.1.3.2 getVar()

```
std::string Pazymiai::getVar ( ) const  [inline], [override], [virtual]
```

Grazina zmogaus varda.

Implements Zmogus.

4.1.4 Friends And Related Symbol Documentation

4.1.4.1 mediana

```
double mediana (
            int u,
            const Pazymiai h )  [friend]
```

Skaiciuoja mediana.

Parameters

u	Pazymiu skaicius
h	Pazymiai objektas

Returns

Mediana

4.1.4.2 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Pazymiai & obj )  [friend]
```

Isvedimo operatorius, skirtas isvesti objekto duomenis i isvesties srauta.

Parameters

os	Isvesties srautas
obj	Pazymiai objektas

Returns

Isvesties srautas

4.1.4.3 operator>>

```
std::istream & operator>> (
            std::istream & is,
            Pazymiai & obj )  [friend]
```

Ivedimo operatorius, skirtas nuskaityti objekto duomenis is ivesties srauto.

Parameters

is	Ivesties srautas
obj	Pazymiai objektas, i kuri nuskaitomi duomenys

Returns

Ivesties srautas

4.1.5 Member Data Documentation

4.1.5.1 egz_

```
int Pazymiai::egz_  [private]
```

Egzamino rezultatas

4.1.5.2 galutinis_

```
double Pazymiai::galutinis_  [private]
```

Galutinis pazymys

4.1.5.3 med_

```
double Pazymiai::med_  [private]
```

Mediana

4.1.5.4 paz_

```
Vector<int> Pazymiai::paz_  [private]
```

Pazymiu sarasas

4.1.5.5 vid_

```
double Pazymiai::vid_  [private]
```

Vidurkis

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

- C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/pagalbinesvector.h

4.2 Vector< T > Class Template Reference

Public Types

- typedef size_t **size_type**
- typedef T **value_type**
- typedef T & **reference**
- typedef const T & **const_reference**
- typedef T * **iterator**
- typedef const T * **const_iterator**

Public Member Functions

- **Vector** ()

  Konstruktoriai /default.
- **Vector** (size_type n, const T &t=T{})
- **Vector** (const Vector &v)
- template<class InputIterator >
  Vector (InputIterator first, InputIterator last)
- **Vector** (Vector &&v)
- **Vector** (const std::initializer_list< T > il)
- ~**Vector** ()

  Destruktorius.
- Vector & **operator=** (const Vector &other)

  Operator = /copy assignment.
- Vector & **operator=** (Vector &&other)
- template<class InputIterator >
  void **assign** (InputIterator first, InputIterator last)

  Assign.
- void **assign** (size_type n, const value_type &val)
- void **assign** (std::initializer_list< value_type > il)
- const_reference **at** (size_type n) const
- T & **operator[ ]** (size_type n)
- const T & **operator[ ]** (size_type n) const
- reference **at** (size_type n)
- reference **front** ()
- const_reference **front** () const
- reference **back** ()
- const_reference **back** () const
- value_type * **data** () noexcept
- const value_type * **data** () const noexcept

- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- size_type **size** () const
- size_type **max_size** () const
- void **resize** (size_type sz)
- void **resize** (size_type sz, const value_type &value)
- size_type **capacity** () const
- bool **empty** () const noexcept
- void **reserve** (size_type n)
- void **shrink_to_fit** ()
- void **clear** () noexcept
- iterator **insert** (const_iterator position, const value_type &val)
- iterator **insert** (iterator position, size_type n, const value_type &val)
- iterator **erase** (iterator position)
- iterator **erase** (iterator first, iterator last)
- void **push_back** (const value_type &t)
- void **push_back** (value_type &&val)
- void **pop_back** ()
- void **swap** (Vector &x)
- bool **operator==** (const Vector< T > &other) const
- bool **operator!=** (const Vector< T > &other) const
- bool **operator**< (const Vector< T > &other) const
- bool **operator**<**=** (const Vector< T > &other) const
- bool **operator**> (const Vector< T > &other) const
- bool **operator**>**=** (const Vector< T > &other) const
- void **swap** (Vector< T > &x, Vector< T > &y)

Private Member Functions

- void **create** ()
- void **create** (size_type n, const T &val)
- void **create** (const_iterator i, const_iterator j)
- void **uncreate** ()
- void **grow** (size_type new_capacity=1)
- void **unchecked_append** (const T &val)

Private Attributes

- iterator **dat**
- iterator **avail**
- iterator **limit**
- std::allocator< T > **alloc**

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

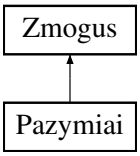
- C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/vector.h

4.3 Zmogus Class Reference

Klase Zmogus saugo informacija apie studento varda ir pavarde.

`#include <pagalbinesvector.h>`

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** ()=default
  Numatytasis konstruktorius.
- Zmogus (const std::string &var, const std::string &pav)
  Parametrizuotas konstruktorius, inicializuoja varda ir pavarde.
- virtual ~**Zmogus** ()
  Virtualus destruktorius.
- virtual void **setVar** (const std::string &newVar)
  Nustato studento varda.
- virtual void **setPav** (const std::string &newPav)
  Nustato studento pavarde.
- virtual std::string getVar () const =0
  Grazina zmogaus varda.
- virtual std::string getPav ()=0
  Grazina zmogaus pavarde.

Protected Attributes

- std::string var_
- std::string pav_

4.3.1 Detailed Description

Klase Zmogus saugo informacija apie studento varda ir pavarde.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Zmogus()

```
Zmogus::Zmogus (
            const std::string & var,
            const std::string & pav )  [inline]
```

Parametrizuotas konstruktorius, inicializuoja varda ir pavarde.

Parameters

var	Zmogaus vardas
pav	Zmogaus pavarde

4.3.3 Member Function Documentation

4.3.3.1 getPav()

```
virtual std::string Zmogus::getPav ( )  [pure virtual]
```

Grazina zmogaus pavarde.

Implemented in Pazymiai.

4.3.3.2 getVar()

```
virtual std::string Zmogus::getVar ( ) const  [pure virtual]
```

Grazina zmogaus varda.

Implemented in Pazymiai.

4.3.4 Member Data Documentation

4.3.4.1 pav_

```
std::string Zmogus::pav_  [protected]
```

Zmogaus pavarde

4.3.4.2 var_

```
std::string Zmogus::var_  [protected]
```

Zmogaus vardas

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

- C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/pagalbinesvector.h

Chapter 5

File Documentation

5.1 C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/mainvector.cpp File Reference

Pagrindinio failo vykdymas.

```
#include "pagalbinesvector.h"
#include "vector.h"
```

Functions

- int main ()

  Vartotojas is meniu pasirenka norima veiksma ir pagal pasirinkima, veda arba generuoja studentu duomenis.

Variables

- int **pasirinkimas**
- int **c**
- int **x**
- int **s**
- int **i** =0
- Vector< Pazymiai > **S**
- Vector< Pazymiai > **P**
- Vector< Pazymiai > **Z**

5.1.1 Detailed Description

Pagrindinio failo vykdymas.

5.1.2 Function Documentation

5.1.2.1 main()

```
int main ( )
```

Vartotojas is meniu pasirenka norima veiksma ir pagal pasirinkima, veda arba generuoja studentu duomenis.

Meniu:

Atvejis 1: Ivesti duomenis ranka Si dalis leidzia vartotojui ivesti studentu duomenis rankiniu budu. Vartotojas gali vesti studentu duomenis, kol ives "4" i studento vardo arba pavardes lauka. Kai duomenu ivedimas baigtas, atvaizduojami ivesti studentu duomenys ekrane

Parameters

P	Vektorius, i kuri pridedami ivesti duomenys.
Pazymiai	C; Objektas, kuri laikinai saugomi ivesti duomenys.

Atvejis 2: Generuoja tik pazymius. Si dalis leidzia vartotojui generuoti studentu duomenis su atsitiktiniais pazymiais. Vartotojui leidziama ivesti varda ir pavarde Vartotojas pasirenka, ar nori generuoti atsitiktinius pazymius arba pasirinkti kieki Atvaizduojami ivesti studentu duomenys ekrane

Parameters

P	Vektorius, i kuri pridedami studentu duomenys.
int	i; Skaiciuoja kiek studentu iraso vartotojas.
Pazymiai	C; Studento objektas, i kuri laikinai saugomi studento duomenys.
string	xx, yy; Laikini kintamieji, kuriuose saugomi studento vardas ir pavarde.
int	egg; Kintamasis, kuriame laikinai saugomas studento egzamino rezultatas.
double	suma = 0.0; Kintamasis, kuriame laikinai saugoma pazymiu suma.
x	Laikinas kintamasis, kuriame saugomas vartotojo atsakymas apie generavima.
string	y; Laikinas kintamasis, kuriame saugomas vartotojo atsakymas apie generavimo tesima.
int	h = 0; Skaiciuoja pazymiu kieki.
int	k; Kintamasis, kuriame saugomas pasirinktu pazymiu skaicius.
int	w; Kintamasis, kuriame saugomas vartotojo pasirinkimas apie egzamino rezultata.

Atvejis 3: Generuoja studentu vardus, pavardes ir pazymius. Si dalis leidzia automatiskai generuoti studentu duomenis su atsitiktiniais pazymiais. Vartotojui atsitiktinai bus generuojamas studeno vardas ir pavarde kol nebus pasirinkta stabdyti generavima. Vartotojui leidziama pasirinkti, ar generuoti atsitiktinius ar pasirinktus pazymius. Vartotojui leidziama pasirinkti, ar generuoti daugiau studentu duomenu. Atvaizduojami studentu rezultatai ekrane.

Parameters

P	Vektorius, i kuri pridedami studentu duomenys.
Pazymiai	C; Studento objektas, i kuri laikinai saugomi studento duomenys.
string	xx, yy; Laikini kintamieji, kuriuose saugomi studento vardas ir pavarde.
x	Laikinas kintamasis, kuriame saugomas vartotojo atsakymas apie generavimo buda.
string	y; Laikinas kintamasis, kuriame saugomas vartotojo atsakymas apie generavimo tesima.
int	h = 0; Skaiciuoja pazymiu kieki.
int	k; Kintamasis, kuriame saugomas pasirinktu pazymiu skaicius.
int	w; Kintamasis, kuriame saugomas vartotojo pasirinkimas apie egzamino rezultata.
int	qq=0; Indikuoja, ar norima testi studentu generavima. Jei qq == 1, baigiama studentu generavimo procedura.
int	egg; Saugomas gzamino rezultatas, jei pasirinkta ivesti ranka.
string	vardai[] = {"Paulina", "Adriana", "Gitanas", "Donald", "Ugne", "Kamile", "Rugile", "Roberta", "Valdemaras", "Jurgis"}; Masyvas, kuriame saugomi studentu vardai, is kuriu bus atsitiktinai generuojami duomenys.
string	pavardes[] = {"Podgaiska", "Obama", "Trump", "Nauseda", "Sirokyte", "Mockute", "Zobelaite", "Macaite", "Jurpalyte", "Jankauskas"}; Masyvas, kuriame saugomos studentu pavardes, is kuriu bus atsitiktinai generuojami duomenys.

Atvejis 4: Nuskaito duomenis is pasirinktu failu. Vartotojui leidziama pasirinkti, is kurio failo nuskaityti duomenis. Nustatomas pasirinktas failo pavadinimas pagal vartotojo pasirinkima. Vartotojui leidziama pasirinkti, kiek studentu

duomenu nuskaityti is failo. Vartotojui leidziama pasirinkti, kaip surikiuoti studentus. Vartotojui leidziama pasirinkti, kur isvesti surikiuotus duomenis.

Parameters

P	Vektorius, i kuri pridedami studentu duomenys.
Pazymiai	C; Studento objektas, i kuri laikinai saugomi generuojami duomenys.
int	z; Laikinas kintamasis pazymiui saugoti.
int	o; Vartotojo pasirinkimas, kur isvesti rezultatus (ekranas ar failas).
int	stud; Vartotojo pasirinkimas, kiek studentu duomenu nuskaityti is failo.
string	zodziai; Laikinas kintamasis pirmos eilutes su pavadinimais nuskaitymui.
int	xyz; Vartotojo pasirinkimas, kaip rusiuoti studentus.
int	pv = 0; Kintamasis saugantis nuskaitytu pazymiu kieki.
int	numeris; Kintamasis saugantis vartotojo pasirinkta failo numeri.
string	wp; Kintamasis saugantis failo pavadinima.
string	xx, yy; Laikini kintamieji studento vardo ir pavardes saugojimui.
int	egg; Laikinas kintamasis egzamino rezultatui saugoti.

Atvejis 5: Generuoja failus su atsitiktiniais duomenimis duomenimis. Generuojami failai yra skirtingu dydziu(1000, 10 000, 100 000, 1 000 000, 10 000 000).

Parameters

Pazymiai	C; Studento objektas, i kuri laikinai saugomi generuojami duomenys.

Atvejis 6: Sugeneruotu failu skaitymas ir apdorojimas. Vartotojas pasirenka strategijas, pagal kurias failai bus skaiciuojami. Rezultatai isspausdinami i failus.

Parameters

Pazymiai	C; Studento objektas, i kuri laikinai saugomi generuojami duomenys.
S	Vektorius, i kuri pridedami visu studentu duomenys.
P	Vektorius, i kuri pridedami studentu duomenys, kuriu vidurkis didesnis nei 5.
Z	Vektorius, i kuri pridedami studentu duomenys, kuriu vidurkis mazesnis uz 5.
int	strategy; Pasirinkta strategija, pagal kuria vykdomas failu apdorojimas.
const	Vector<string> filenames = {"1k.txt", "10k.txt", "100k.txt", "1kk.txt", "10kk.txt"}; Vektorius, saugantis failu pavadinimus, kurie bus apdorojami.
const	Vector<string> lopukaiFilenames = {"lopukai1.txt", "lopukai2.txt", "lopukai3.txt", "lopukai4.txt", "lopukai5.txt"}; Vektorius, saugantis failu pavadinimus, kuriuose bus isvedami rezultatai.
const	Vector<string> saunuoliukaiFilenames = {"saunuoliukai1.txt", "saunuoliukai2.txt", "saunuoliukai3.txt", "saunuoliukai4.txt", "saunuoliukai5.txt"}; Vektorius, saugantis failu pavadinimus, kuriuose bus isvedami rezultatai.

Atvejis 7: Klasiu testavimas. Iskvieciamas funkcijos testai() ir ekrane matome testu rezultata.

Atvejis 8: Programos pabaiga

Returns

Grazinamas nulis, nurodantis, kad programa sekmingai baige darba

5.2 C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/pagalbinesvector.cpp File Reference

Funkciju deklaracija.

```
#include "pagalbinesvector.h"
#include "vector.h"
```

Functions

- void rezultatai (Vector< Pazymiai > hh)

  Spausdina studentu rezultatus i konsole.
- void rezultataifailas (Vector< Pazymiai > hh, std::string failiukas)

  Spausdina studentu rezultatus i faila.
- double mediana (int u, Pazymiai h)

  Skaiciuoja mediana.
- void generuojam (int studentusk, std::string failopav)

  Generuoja atsitiktinius studentu duomenis ir iraso i faila.
- void failuskaick (std::string wp, Pazymiai hi, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)

  Nuskaito duomenis is failo ir apdoroja juos.
- void failuskaickstrategija1 (string wp, Pazymiai hi, Vector< Pazymiai > &S, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)

  Nuskaito duomenis is failo ir apdoroja juos, taikant strategija 1.
- void failuskaickstrategija2 (string wp, Pazymiai hi, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)

  Funkcija nuskaito duomenis is failo, apdoroja juos ir isskirsto i du konteinerius.
- void failuskaickstrategija3 (string wp, Pazymiai hi, Vector< Pazymiai > &S, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)

  Funkcija nuskaito duomenis is failo, apdoroja juos ir isskirsto i tris konteinerius.
- void spausdintuvas (std::string zekai, std::string malaciai, Vector< Pazymiai > P, Vector< Pazymiai > Z)

  Funkcija isveda studentu rezultatus i du atskirus failus.
- void **testai** ()

  Funkcija skirta programos testavimui.

5.2.1 Detailed Description

Funkciju deklaracija.

5.2.2 Function Documentation

5.2.2.1 failuskaick()

```
void failuskaick (
            std::string wp,
            Pazymiai hi,
            Vector< Pazymiai > & P,
            Vector< Pazymiai > & Z )
```

Nuskaito duomenis is failo ir apdoroja juos.

Parameters

wp	Failo pavadinimas
hi	Pazymiai objektas
P	Studentai, kuriu galutinis pazymys >= 5
Z	Studentai, kuriu galutinis pazymys < 5

5.2.2.2 failuskaickstrategija1()

```
void failuskaickstrategija1 (
            string wp,
            Pazymiai hi,
            Vector< Pazymiai > & S,
            Vector< Pazymiai > & P,
            Vector< Pazymiai > & Z )
```

Nuskaito duomenis is failo ir apdoroja juos, taikant strategija 1.

Parameters

wp	Failo pavadinimas
hi	Pazymiai objektas
S	Visi nuskaityti studentai
P	Studentai, kuriu galutinis pazymys >= 5
Z	Studentai, kuriu galutinis pazymys < 5

5.2.2.3 failuskaickstrategija2()

```
void failuskaickstrategija2 (
            string wp,
            Pazymiai hi,
            Vector< Pazymiai > & P,
            Vector< Pazymiai > & Z )
```

Funkcija nuskaito duomenis is failo, apdoroja juos ir isskirsto i du konteinerius.

Parameters

wp	Failo pavadinimas, is kurio nuskaitomi duomenys.
hi	Objektas, kuris naudojamas saugoti viena studento informacijos irasa.
P	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra didesnis nei 5.
Z	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra mazesnis nei 5.

5.2.2.4 failuskaickstrategija3()

```
void failuskaickstrategija3 (
            string wp,
```

```
            Pazymiai hi,
            Vector< Pazymiai > & S,
            Vector< Pazymiai > & P,
            Vector< Pazymiai > & Z )
```

Funkcija nuskaito duomenis is failo, apdoroja juos ir isskirsto i tris konteinerius.

Parameters

wp	Failo pavadinimas, is kurio nuskaitomi duomenys.
hi	Objektas, kuris naudojamas saugoti viena studento informacijos irasa.
S	Konteineris, kuriame saugomi visi studentai.
P	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra didesnis nei 5.
Z	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra mazesnis nei 5.

5.2.2.5 generuojam()

```
void generuojam (
            int studentusk,
            std::string failopav )
```

Generuoja atsitiktinius studentu duomenis ir iraso i faila.

Parameters

studentusk	Studentu skaicius
failopav	Failo pavadinimas

5.2.2.6 mediana()

```
double mediana (
            int u,
            Pazymiai h )
```

Skaiciuoja mediana.

Parameters

u	Pazymiu skaicius
h	Pazymiai objektas

Returns

Mediana

5.2.2.7 rezultatai()

```
void rezultatai (
            Vector< Pazymiai > hh )
```

Spausdina studentu rezultatus i konsole.

Parameters

hh	Studentu sarasas

5.2.2.8 rezultataifailas()

```
void rezultataifailas (
            Vector< Pazymiai > hh,
            std::string failiukas )
```

Spausdina studentu rezultatus i faila.

Parameters

hh	Studentu sarasas
failiukas	Failo pavadinimas

5.2.2.9 spausdintuvas()

```
void spausdintuvas (
            std::string zekai,
            std::string malaciai,
            Vector< Pazymiai > P,
            Vector< Pazymiai > Z )
```

Funkcija isveda studentu rezultatus i du atskirus failus.

Parameters

zekai	Failo pavadinimas, i kuri isvedami studentai, kuriu galutinis rezultatas yra mazesnis nei 5.
malaciai	Failo pavadinimas, i kuri isvedami studentai, kuriu galutinis rezultatas yra didesnis nei 5.
P	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra didesnis nei 5.
Z	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra mazesnis nei 5.

5.3 C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/pagalbinesvector.h File Reference

Pazymiai ir Zmogus klasiu deklaracija ir funkciju reiksmiu priskyrimas.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cstdlib>
#include <vector>
```

```
#include <algorithm>
#include <fstream>
#include <chrono>
#include <stdexcept>
#include <cassert>
#include "vector.h"
```

Classes

- class Zmogus

  Klase Zmogus saugo informacija apie studento varda ir pavarde.
- class Pazymiai

  Klase Pazymiai saugo informacija apie studentu pazymius.

Functions

- void rezultatai (Vector< Pazymiai > hh)

  Spausdina studentu rezultatus i konsole.
- void rezultataifailas (Vector< Pazymiai > hh, std::string failiukas)

  Spausdina studentu rezultatus i faila.
- double mediana (int u, Pazymiai h)

  Skaiciuoja mediana.
- void generuojam (int studentusk, std::string failopav)

  Generuoja atsitiktinius studentu duomenis ir iraso i faila.
- void failuskaick (std::string wp, Pazymiai hi, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)

  Nuskaito duomenis is failo ir apdoroja juos.
- void **failuskaickstrategija1** (std::string wp, Pazymiai hi, Vector< Pazymiai > &S, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)
- void **failuskaickstrategija2** (std::string wp, Pazymiai hi, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)
- void **failuskaickstrategija3** (std::string wp, Pazymiai hi, Vector< Pazymiai > &S, Vector< Pazymiai > &P, Vector< Pazymiai > &Z)
- void spausdintuvas (std::string zekai, std::string malaciai, Vector< Pazymiai > P, Vector< Pazymiai > Z)

  Funkcija isveda studentu rezultatus i du atskirus failus.
- void **testai** ()

  Funkcija skirta programos testavimui.

5.3.1 Detailed Description

Pazymiai ir Zmogus klasiu deklaracija ir funkciju reiksmiu priskyrimas.

5.3.2 Function Documentation

5.3.2.1 failuskaick()

```
void failuskaick (
            std::string wp,
            Pazymiai hi,
            Vector< Pazymiai > & P,
            Vector< Pazymiai > & Z )
```

Nuskaito duomenis is failo ir apdoroja juos.

Parameters

wp	Failo pavadinimas
hi	Pazymiai objektas
P	Studentai, kuriu galutinis pazymys >= 5
Z	Studentai, kuriu galutinis pazymys < 5

5.3.2.2 generuojam()

```
void generuojam (
            int studentusk,
            std::string failopav )
```

Generuoja atsitiktinius studentu duomenis ir iraso i faila.

Parameters

studentusk	Studentu skaicius
failopav	Failo pavadinimas

5.3.2.3 mediana()

```
double mediana (
            int u,
            Pazymiai h )
```

Skaiciuoja mediana.

Parameters

u	Pazymiu skaicius
h	Pazymiai objektas

Returns

Mediana

5.3.2.4 rezultatai()

```
void rezultatai (
            Vector< Pazymiai > hh )
```

Spausdina studentu rezultatus i konsole.

Parameters

hh	Studentu sarasas

5.3.2.5 rezultataifailas()

```
void rezultataifailas (
            Vector< Pazymiai > hh,
            std::string failiukas )
```

Spausdina studentu rezultatus i faila.

Parameters

hh	Studentu sarasas
failiukas	Failo pavadinimas

5.3.2.6 spausdintuvas()

```
void spausdintuvas (
            std::string zekai,
            std::string malaciai,
            Vector< Pazymiai > P,
            Vector< Pazymiai > Z )
```

Funkcija isveda studentu rezultatus i du atskirus failus.

Parameters

zekai	Failo pavadinimas, i kuri isvedami studentai, kuriu galutinis rezultatas yra mazesnis nei 5.
malaciai	Failo pavadinimas, i kuri isvedami studentai, kuriu galutinis rezultatas yra didesnis nei 5.
P	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra didesnis nei 5.
Z	Konteineris, kuriame saugomi studentai, kuriu galutinis rezultatas yra mazesnis nei 5.

5.4 pagalbinesvector.h

Go to the documentation of this file.

```
00001
00005 #ifndef PAGALBINESVECTOR_H_INCLUDED
00006 #define PAGALBINESVECTOR_H_INCLUDED
00007 #include <iostream>
00008 #include <iomanip>
00009 #include <string>
00010 #include <cstdlib>
00011 #include <vector>
00012 #include <algorithm>
00013 #include <fstream>
00014 #include <chrono>
00015 #include <string>
00016 #include <stdexcept>
00017 #include <cassert>
00018 #include "vector.h"
00023 class Zmogus {
00024 protected:
00025     std::string var_;
00026     std::string pav_;
00028 public:
00032     Zmogus() = default;
00033
00039     Zmogus(const std::string& var, const std::string& pav) : var_(var), pav_(pav) {}
00040
00044     virtual ~Zmogus() {}
```

```
00045
00049     virtual void setVar(const std::string& newVar) { var_ = newVar; }
00050
00054     virtual void setPav(const std::string& newPav) { pav_ = newPav; }
00055
00059     virtual std::string getVar() const = 0;
00060
00064     virtual std::string getPav() = 0;
00065 };
00066
00071 class Pazymiai : public Zmogus {
00072 private:
00073     double vid_;
00074     int egz_;
00075     Vector<int> paz_;
00076     double galutinis_;
00077     double med_;
00079 public:
00083     Pazymiai() : vid_(0), egz_(0), galutinis_(0), med_(0) {}
00084
00095     Pazymiai(const std::string& var, const std::string& pav, double vid, int egz, const Vector<int>&
    paz,
00096              double galutinis, double med)
00097         : Zmogus(var, pav), vid_(vid), egz_(egz), paz_(paz), galutinis_(galutinis), med_(med) {}
00098
00102     ~Pazymiai() { paz_.clear(); }
00106     Pazymiai(const Pazymiai& other)
00107         : Zmogus(other.getVar(), other.getPav()), vid_(other.vid_), egz_(other.egz_),
00108           paz_(other.paz_), galutinis_(other.galutinis_), med_(other.med_) {}
00112     Pazymiai(Pazymiai&& other) noexcept
00113         : Zmogus(std::move(other.var_), std::move(other.pav_)),
00114           vid_(other.vid_), egz_(other.egz_), paz_(std::move(other.paz_)),
00115           galutinis_(other.galutinis_), med_(other.med_) {}
00119     Pazymiai& operator=(const Pazymiai& other) {
00120         if (this != &other) {
00121             Zmogus::setVar(other.getVar());
00122             Zmogus::setPav(other.getPav());
00123             vid_ = other.vid_;
00124             egz_ = other.egz_;
00125             paz_ = other.paz_;
00126             galutinis_ = other.galutinis_;
00127             med_ = other.med_;
00128         }
00129         return *this;
00130     }
00134     Pazymiai& operator=(Pazymiai&& other) noexcept {
00135         if (this != &other) {
00136             Zmogus::setVar(std::move(other.var_));
00137             Zmogus::setPav(std::move(other.pav_));
00138             vid_ = other.vid_;
00139             egz_ = other.egz_;
00140             paz_ = std::move(other.paz_);
00141             galutinis_ = other.galutinis_;
00142             med_ = other.med_;
00143         }
00144         return *this;
00145     }
00146
00149     int getPazN(const Vector<int>& newPaz, int pos) const { return newPaz[pos]; }
00150
00154     void clearPaz() { paz_.clear(); }
00157     void setVid(double newVid) { vid_ = newVid; }
00158
00161     void setEgz(int newEgz) { egz_ = newEgz; }
00162
00165     void setOnePaz(int newPaz) { paz_.push_back(newPaz); }
00166
00169     void setPazymiai(const Vector<int>& pazymiai) { paz_ = pazymiai; }
00170
00173     void setGalutinis(double newGalutinis) { galutinis_ = newGalutinis; }
00174
00177     void setMed(double newMed) { med_ = newMed; }
00178
00181     void sortPaz(Pazymiai& C) { std::sort(C.paz_.begin(), C.paz_.end()); }
00182
00185     double getVid() const { return vid_; }
00186
00189     int getEgz() const { return egz_; }
00190
00193     Vector<int> getPaz() const { return paz_; }
00194
00197     double getGalutinis() const { return galutinis_; }
00198
00201     double getMed() const { return med_; }
00202
00206     std::string getVar() const override { return var_; }
00207
```

```
00211     std::string getPav() override { return pav_; }
00212
00218     friend double mediana(int u, const Pazymiai h);
00219
00225     friend std::istream& operator»(std::istream& is, Pazymiai& obj) {
00226         std::cout « "Iveskite studento varda (noredami baigti spauskite 4):" « std::endl;
00227         is » obj.var_;
00228         if (obj.var_ == "4" || obj.pav_ == "4")
00229             return is;
00230         std::cout « "Iveskite studento pavarde (noredami baigti spauskite 4):" « std::endl;
00231         is » obj.pav_;
00232         if (obj.var_ == "4" || obj.pav_ == "4")
00233             return is;
00234
00235         double suma = 0.0;
00236         int pazymys;
00237         int j = 0;
00238
00239         do {
00240             std::cout « "Iveskite " « j + 1 « " pazymi (norint baigti spauskite 11): ";
00241             is » pazymys;
00242
00243             if (pazymys == 11)
00244                 break;
00245
00246             while (pazymys < 1 || pazymys > 10 || is.fail()) {
00247                 std::cout « "Klaida. Iveskite skaiciu nuo 1 iki 10: ";
00248                 is.clear();
00249                 is.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00250                 is » pazymys;
00251             }
00252
00253             obj.paz_.push_back(pazymys);
00254             suma += pazymys;
00255             j++;
00256         } while (true);
00257
00258         obj.vid_ = suma / j;
00259
00260         std::cout « "Iveskite egzamino rezultata : ";
00261         is » obj.egz_;
00262
00263         while (obj.egz_ < 1 || obj.egz_ > 10 || is.fail()) {
00264             std::cout « "Klaida. Iveskite skaiciu nuo 1 iki 10: ";
00265             is.clear();
00266             is.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00267             is » obj.egz_;
00268         }
00269
00270         obj.galutinis_ = (obj.vid_ * 0.4) + (obj.egz_ * 0.6);
00271
00272         std::sort(obj.paz_.begin(), obj.paz_.end());
00273
00274         obj.med_ = mediana(j, obj);
00275
00276         return is;
00277     }
00284     friend std::ostream& operator«(std::ostream& os, const Pazymiai& obj) {
00285         os « std::left « std::setw(15) « obj.var_ « std::setw(15) « obj.pav_ « std::setw(17)
00286            « std::fixed « std::setprecision(2) « obj.galutinis_ « std::setw(17) « std::fixed
00287            « std::setprecision(2) « obj.med_ « std::endl;
00288         return os;
00289     }
00290 };
00291
00292 void rezultatai(Vector<Pazymiai> hh);
00293
00294 void rezultataifailas(Vector<Pazymiai> hh, std::string failiukas);
00295
00296 double mediana(int u, Pazymiai h);
00297
00298 void generuojam(int studentusk, std::string failopav);
00299
00300 void failuskaick(std::string wp, Pazymiai hi, Vector<Pazymiai>& P, Vector<Pazymiai>& Z);
00301
00302 void failuskaickstrategija1(std::string wp, Pazymiai hi,Vector<Pazymiai>& S, Vector<Pazymiai>& P,
      Vector<Pazymiai>& Z);
00303
00304 void failuskaickstrategija2(std::string wp, Pazymiai hi, Vector<Pazymiai>& P, Vector<Pazymiai>& Z);
00305
00306 void failuskaickstrategija3(std::string wp, Pazymiai hi,Vector<Pazymiai>& S, Vector<Pazymiai>& P,
      Vector<Pazymiai>& Z);
00307
00308 void spausdintuvas(std::string zekai, std::string malaciai, Vector<Pazymiai> P, Vector<Pazymiai> Z );
00309
00310 void testai();
00311
```

```
00312 #endif // PAGALBINES_H_INCLUDED
```

5.5 C:/Users/Paulina/Documents/Objektinis programavimas 1 kursas/vectorcontainer/vector.h File Reference

Vector konteineris.

```
#include <iostream>
#include <memory>
#include <algorithm>
#include <limits>
```

Classes

- class Vector< T >

5.5.1 Detailed Description

Vector konteineris.

5.6 vector.h

Go to the documentation of this file.

```
00001 #pragma once
00006 #include <iostream>
00007 #include <memory>
00008 #include <algorithm>
00009 #include <limits>
00010
00011 template <typename T>
00012 class Vector{
00013     public:
00014
00015     // MEMBER TYPE
00016         typedef size_t size_type;
00017         typedef T value_type;
00018         typedef T& reference;
00019         typedef const T& const_reference;
00020         typedef T* iterator;
00021         typedef const T* const_iterator;
00022
00023     // MEMBER FUNCTIONS
00024
00027         Vector() {create();}
00029         explicit Vector(size_type n, const T& t = T{}) { create (n,t); }
00031         Vector(const Vector& v) { create(v.begin(), v.end()); }
00033         template <class InputIterator>
00034         Vector (InputIterator first, InputIterator last) { create(first,last); }
00036         Vector (Vector&& v) {
00037             create();
00038             swap(v);
00039             v.uncreate();
00040         }
00042         Vector(const std::initializer_list<T> il) { create(il.begin(), il.end()); }
00043
00045         ~Vector() {uncreate();}
00046
00049         Vector& operator = (const Vector& other) {
00050             if (this != &other) {
00051                 uncreate();
00052                 create(other.begin(), other.end());
```

```
00053             }
00054             return *this;
00055         };
00056 
00058         Vector& operator = (Vector&& other) {
00059             if (this != &other) {
00060                 create(other.begin(), other.end());
00061                 uncreate();
00062             }
00063             return *this;
00064         }
00065 
00067         template <class InputIterator>
00068         void assign (InputIterator first, InputIterator last) {
00069             uncreate();
00070             create(first, last);
00071         }
00072         void assign (size_type n, const value_type& val) {
00073             uncreate();
00074             create(n, val);
00075         }
00076         void assign (std::initializer_list<value_type> il) {
00077             uncreate();
00078             create(il);
00079         }
00080 
00081         // Element access
00082         const_reference at (size_type n) const {
00083             if (n >= size() || n < 0)
00084                 throw std::out_of_range("Index out of range");
00085             return dat[n];
00086         }
00087 
00088         T& operator[] (size_type n) {return dat[n];}
00089         const T& operator[] (size_type n) const {return dat[n];}
00090         reference at (size_type n) {
00091             if (n >= size() || n < 0)
00092                 throw std::out_of_range("Index out of range");
00093             return dat[n];
00094         }
00095 
00096         reference front() {
00097             return dat[0];
00098         };
00099 
00100         const_reference front() const {
00101             return dat[0];
00102         }
00103 
00104         reference back() {
00105             return dat[size() - 1];
00106         }
00107 
00108         const_reference back() const {
00109             return dat[size() - 1];
00110         }
00111 
00112         value_type* data() noexcept {
00113             return dat;
00114         }
00115 
00116         const value_type* data() const noexcept {
00117             return dat;
00118         }
00119 
00120     // Iterators
00121         iterator begin() {return dat;}
00122         const_iterator begin() const {return dat;}
00123         iterator end() {return avail;}
00124         const_iterator end() const {return avail;}
00125 
00126     // Capacity
00127         size_type size() const {return avail-dat;}
00128         size_type max_size() const {return std::numeric_limits<size_type>::max();}
00129         void resize(size_type sz) {
00130             if (sz < size()) {
00131                 iterator it = dat + sz;
00132                 while (it != avail) {
00133                     alloc.destroy(it++);
00134                 }
00135                 avail = dat + sz;
00136             }
00137             else if (sz > capacity()) {
00138                 grow(sz);
00139                 std::uninitialized_fill(avail, dat + sz, value_type());
00140                 avail = dat + sz;
00141             }
```

```
00142             else if (sz > size()) {
00143                 std::uninitialized_fill(avail, dat + sz, value_type());
00144                 avail = dat + sz;
00145             }
00146         }
00147 
00148         void resize(size_type sz, const value_type& value) {
00149             if (sz > capacity()) {
00150                 grow(sz);
00151             }
00152 
00153             if (sz > size()) {
00154                 insert(end(), sz - size(), value);
00155             } else if (sz < size()) {
00156                 avail = dat + sz;
00157             }
00158         }
00159 
00160         size_type capacity() const {return limit-dat;}
00161         bool empty() const noexcept { return size() == 0;}
00162         void reserve (size_type n) {
00163             if (n > capacity()) {
00164                 grow(n);
00165             }
00166         }
00167         void shrink_to_fit(){
00168             if (limit > avail)
00169             limit = avail;
00170         }
00171 
00172     // Modifiers
00173         void clear() noexcept {
00174             uncreate();
00175         }
00176 
00177         iterator insert (const_iterator position, const value_type& val) {
00178             return insert(position, 1, val);
00179         }
00180         iterator insert(iterator position, size_type n, const value_type& val) {
00181             if (position < begin() || position > end()) {
00182                 throw std::out_of_range("Index out of range");
00183             }
00184             if (avail + n > limit) {
00185                 size_type index = position - begin();
00186                 grow(n);
00187                 position = begin() + index;
00188             }
00189             for (iterator it = end() + n - 1; it != position + n - 1; --it) {
00190                 *it = std::move(*(it - n));
00191             }
00192             std::uninitialized_fill(position, position + n, val);
00193             avail += n;
00194 
00195             return position;
00196         }
00197 
00198         iterator erase(iterator position) {
00199             if (position < dat || position > avail) {
00200                 throw std::out_of_range("Index out of range");
00201             }
00202             std::move(position + 1, avail, position);
00203             alloc.destroy(avail - 1);
00204             --avail;
00205 
00206             return position;
00207         }
00208 
00209         iterator erase(iterator first, iterator last) {
00210             iterator new_available = std::uninitialized_copy(last, avail, first);
00211 
00212             iterator it = avail;
00213             while (it != new_available) {
00214                 alloc.destroy(--it);
00215             }
00216 
00217             avail= new_available;
00218             return last;
00219         }
00220 
00221         void push_back (const value_type& t) {
00222             if (avail==limit)
00223                 grow();
00224             unchecked_append(t);
00225         }
00226 
00227         void push_back (value_type&& val) {
00228             if (avail == limit)
```

```
00229                 grow();
00230             unchecked_append(val);
00231         }
00232 
00233         void pop_back() {
00234             if (avail != dat)
00235                 alloc.destroy(--avail);
00236         }
00237 
00238         void swap(Vector& x) {
00239             std::swap(dat, x.dat);
00240             std::swap(avail, x.avail);
00241             std::swap(limit, x.limit);
00242         }
00243 
00244 
00245     // NON-MEMBER FUNCTIONS
00246 
00247         bool operator== (const Vector<T>& other) const {
00248             if (size() != other.size()) {
00249                 return false;
00250             }
00251 
00252             return std::equal(begin(), end(), other.begin());
00253         }
00254         bool operator!= (const Vector<T>& other) const {
00255             return !(*this == other);
00256         }
00257         bool operator < (const Vector<T> & other) const {
00258             return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
00259         }
00260         bool operator <= (const Vector<T> & other) const {
00261             return !(other < *this);
00262         }
00263         bool operator > (const Vector<T> & other) const {
00264             return std::lexicographical_compare(other.begin(), other.end(), begin(), end());
00265         }
00266         bool operator >= (const Vector<T> & other) const {
00267             return !(other > *this);
00268         }
00269 
00270         void swap (Vector<T>& x, Vector<T>& y) {
00271             std::swap(x,y);
00272         }
00273 
00274     private:
00275         iterator dat;
00276         iterator avail;
00277         iterator limit;
00278         std::allocator<T> alloc;
00279         void create() {dat = avail = limit = nullptr;}
00280         void create (size_type n, const T& val) {
00281             dat = alloc.allocate(n);
00282             limit = avail = dat + n;
00283             std::uninitialized_fill(dat, limit, val);
00284         }
00285         void create(const_iterator i, const_iterator j) {
00286             dat = alloc.allocate(j - i);
00287             limit = avail = std::uninitialized_copy(i, j, dat);
00288         }
00289         void uncreate(){
00290             if (dat) {
00291                 iterator it = avail;
00292                 while (it != dat) {
00293                     alloc.destroy(--it);
00294                 }
00295             alloc.deallocate(dat, limit - dat);
00296             }
00297             dat = limit = avail = nullptr;
00298         }
00299         void grow(size_type new_capacity = 1) {
00300             size_type new_size = std::max(new_capacity, 2 * capacity());
00301             iterator new_data = alloc.allocate(new_size);
00302             iterator new_avail = std::uninitialized_copy(dat, avail, new_data);
00303             uncreate();
00304             dat = new_data;
00305             avail = new_avail;
00306             limit = dat + new_size;
00307         }
00308         void unchecked_append(const T& val) {
00309             alloc.construct(avail++, val);
00310         }
00311 };
```

Index