Objektinio Programavimo 3.0

Generated by Doxygen 1.13.2

Chapter 1

Objektinio-programavimo-uzd

Ši programa del skirtingu konteineriu laikų testavimų naudojant struktūras Kad paleist programa reikia tik paleisti run.bat faila

CPU: AMD Ryzen 7 8845, 8 Cores, 3,8 GHZ
RAM: 16 GB, 5600 MT/s
SSD NVMe

3.0

Šioje Programoje sukurta ir naudojama Vektoriaus klase Joje implementuota Construktoriai: Vektorius<typename> v //Sukuria tuščia vektoriu Vektorius<typename> v(n) //Sukuria tuščia vektoriu n dydžio Vektorius<typename> v(n,k) //Sukuria vektoriu n dydžio, užpildyta k Vektorius<typename> v({1,2,3}) //Sukuria vektoriu kuriame yra elementai 1,2,3

Destruktorius copy construktorius ir operacija move consturktorius ir operacija

swap assign shrink_to_fit erase insert emplace push_back pop_back size capacity empty clear reserve resize at [] operatorius ==, !=, <, <=, >, >= begin, cbegin, rbegin, rcbegin end, cend, rend, rcend, front, back, įprasta vektoriaus data() funckija pervardinta į storage(),

Daugiau dokumentuotas klase yra pdf formatu

std::vector ir Vektoriaus užpildymo laiko testavimas

	std:vector	Vektorius	std Reallocations	Vektoriaus Reallocations
10000	0.000517	0.000272	15	15
100000	0.003779	0.001009	18	18
1000000	0.035981	0.010968	21	21
10000000	0.287432	0.107743	25	25
100000000	3.30311	1.23185	28	28

Sukurto vektoriaus spartumas

Vektorius				
	Nuskaitymas	Išrušiavimas	Išrikiavimas	Iš viso

1000	0.043964	0.00224	0.0056341	0.051769
10000	0.444661	0.0309179	0.049972	0.525462
100000	4.5369	0.22554	0.558844	5.32118
1000000	44.3652	2.31514	5.91125	52.5915
10000000	490.645	47.8489	61.81	600.3

std::vector spartumas

std Vector				
	Nuskaitymas	Išrušiavimas	Išrikiavimas	Iš viso
1000	0.0644308	0.0022129	0.0074612	0.074028
10000	0.649687	0.0372766	0.0914448	0.776228
100000	6.62527	0.302673	0.828665	7.75646
1000000	65.8765	2.76999	9.14181	77.7871

| 10000000 | 697.445 | 33.2529 | 95.7009 | 826.395 |

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 files with brief descriptions:

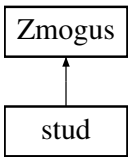
Chapter 5

Class Documentation

5.1 stud Class Reference

`#include <stud.h>`

Inheritance diagram for stud:



Public Member Functions

- stud ()
- stud (const string &vardas, const string &pavarde)
- stud (std::istringstream &line)
- stud (const stud &kitas)
- stud (stud &&kitas) noexcept
- ~stud ()
- stud & operator= (const stud &kitas)
- stud & operator= (stud &&kitas) noexcept
- void addTarpPazymys (int paz)
- void setEgzaminas (double egzaminas)
- void setVid (double vid)
- void setMed (double med)
- Vektorius< int > & getTarp ()
- double getEgz () const
- double getGalutinisVid () const
- double getGalutinisMed () const
- void calculateGalutinis ()

Public Member Functions inherited from Zmogus

- Zmogus ()=default
- Zmogus (const string &vard, const string &pava)
- virtual ~Zmogus ()=default
- Zmogus (const Zmogus &other)
- Zmogus & operator= (const Zmogus &other)
- Zmogus (Zmogus &&other) noexcept
- Zmogus & operator= (Zmogus &&other) noexcept
- void setvard (const string &vardas)
- void setpava (const string &pavard)
- string getVardas () const
- string getPavarde () const

Friends

- std::ostream & operator<< (std::ostream &out, const stud &a)
- std::istream & operator>> (std::istream &in, stud &a)

Additional Inherited Members

Protected Attributes inherited from Zmogus

- string vard
- string pava

5.1.1 Detailed Description

Definition at line 9 of file stud.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 stud() [1/5]

```
stud::stud ()  [inline]
```

Definition at line 20 of file stud.h.

5.1.2.2 stud() [2/5]

```
stud::stud (
            const string & vardas,
            const string & pavarde)  [inline]
```

Definition at line 22 of file stud.h.

5.1.2.3 stud() [3/5]

```
stud::stud (
            std::istringstream & line)  [inline]
```

Definition at line 24 of file stud.h.

5.1.2.4 stud() [4/5]

```
stud::stud (
            const stud & kitas)  [inline]
```

Definition at line 37 of file stud.h.

5.1.2.5 stud() [5/5]

```
stud::stud (
            stud && kitas)  [inline], [noexcept]
```

Definition at line 45 of file stud.h.

5.1.2.6 ~stud()

```
stud::~stud ()  [inline]
```

Definition at line 57 of file stud.h.

5.1.3 Member Function Documentation

5.1.3.1 addTarpPazymys()

```
void stud::addTarpPazymys (
            int paz)  [inline]
```

Definition at line 91 of file stud.h.

5.1.3.2 calculateGalutinis()

```
void stud::calculateGalutinis ()  [inline]
```

Definition at line 106 of file stud.h.

5.1.3.3 getEgz()

```
double stud::getEgz () const  [inline]
```

Definition at line 101 of file stud.h.

5.1.3.4 getGalutinisMed()

```
double stud::getGalutinisMed () const  [inline]
```

Definition at line 103 of file stud.h.

5.1.3.5 getGalutinisVid()

```
double stud::getGalutinisVid () const  [inline]
```

Definition at line 102 of file stud.h.

5.1.3.6 getTarp()

```
Vektorius< int > & stud::getTarp ()  [inline]
```

Definition at line 100 of file stud.h.

5.1.3.7 operator=() [1/2]

```
stud & stud::operator= (
            const stud & kitas)  [inline]
```

Definition at line 63 of file stud.h.

5.1.3.8 operator=() [2/2]

```
stud & stud::operator= (
            stud && kitas)  [inline], [noexcept]
```

Definition at line 75 of file stud.h.

5.1.3.9 setEgzaminas()

```
void stud::setEgzaminas (
            double egzaminas)  [inline]
```

Definition at line 95 of file stud.h.

5.1.3.10 setMed()

```
void stud::setMed (
            double med)  [inline]
```

Definition at line 97 of file stud.h.

5.1.3.11 setVid()

```
void stud::setVid (
            double vid)  [inline]
```

Definition at line 96 of file stud.h.

5.1.4 Friends And Related Symbol Documentation

5.1.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & out,
            const stud & a)  [friend]
```

Definition at line 121 of file stud.h.

5.1.4.2 operator>>

```
std::istream & operator>> (
            std::istream & in,
            stud & a)  [friend]
```

Definition at line 127 of file stud.h.

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

- stud.h

5.2 Timer Class Reference

```
#include <Timer.h>
```

Public Member Functions

- Timer ()
- void reset ()
- double elapsed () const

5.2.1 Detailed Description

Definition at line 6 of file Timer.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Timer()

```
Timer::Timer ()  [inline]
```

Definition at line 13 of file Timer.h.

5.2.3 Member Function Documentation

5.2.3.1 elapsed()

```
double Timer::elapsed () const  [inline]
```

Definition at line 17 of file Timer.h.

5.2.3.2 reset()

```
void Timer::reset ()  [inline]
```

Definition at line 14 of file Timer.h.

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

- Timer.h

5.3 Vektorius< T > Class Template Reference

```
#include <Vektorius.h>
```

Public Types

- using iterator = T*
- using const_iterator = const T*
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using reference = T&
- using const_reference = const T&
- using reverse_iterator = std::reverse_iterator<iterator>
- using const_reverse_iterator = std::reverse_iterator<const_iterator>

Public Member Functions

- Vektorius ()

  Default construktorius, sukuria tuscia vektoriu.
- Vektorius (size_t size)

  Konstruktorius, sukuria vektoriu su nurodytu dydziu.
- Vektorius (size_t size, const T &value)

  Konstruktorius, sukuria vektoriu su nurodytu dydziu ir uzpildo jį nurodyta reiksme.
- Vektorius (std::initializer_list< T > ilist)

  Konstruktorius su initializer_list.
- ~Vektorius ()

  Destruktorius.
- Vektorius (const Vektorius &other)

  Copy constructorius.
- Vektorius & operator= (const Vektorius &other)

  Copy assignment operatorius.
- Vektorius (Vektorius &&other) noexcept

  Move constructorius.
- Vektorius & operator= (Vektorius &&other) noexcept

  Move assignment operatorius.
- void swap (Vektorius &other)

  Swap funkcija.
- void assign (size_t count, const T &value)

  Priskiria vektoriui nauja reiksme.
- template<typename InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt>>>
  void assign (InputIt first, InputIt last)

  Priskiria vektoriui nauja reiksme.
- void assign (std::initializer_list< T > ilist)

  Priskiria vektoriui nauja reiksme.
- void shrink_to_fit ()

  Sumažina vektoriaus talpą iki jo dydžio.
- iterator erase (const_iterator pos)

  Ištrina elementą iš vektoriaus.
- iterator erase (const_iterator first, const_iterator last)

  Ištrina elementus iš vektoriaus.
- iterator insert (const_iterator pos, const T &value)

  Įterpia elementą į vektorių
- iterator insert (const_iterator pos, T &&value)

  Įterpia elementą į vektorių naudodamas move semantika.
- iterator insert (const_iterator pos, size_type count, const T &value)

  Įterpia kelis elementus į vektorių
- template<class InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt>>>
  iterator insert (const_iterator pos, InputIt first, InputIt last)

  Įterpia kelis elementus į vektorių per pointerius.
- template<class... Args>
  iterator emplace (const_iterator pos, Args &&... args)

  sukuriamas elementą į vektorių su emplace
- void push_back (const T &value)

  Prideda elementą į vektoriaus gala.
- void pop_back ()

  Ištrina paskutinį elementą iš vektoriaus.

- size_t size () const

  Gražina vektoriaus dydį
- size_t capacity () const

  Gražina vektoriaus talpą
- bool empty () const

  Gražina ar vektorius yra tuscias.
- void clear ()

  Ištrina visus elementus iš vektoriaus.
- void reserve (size_t new_cap)

  Rezervuoja talpą vektoriui.
- void resize (size_t count)

  Pakeičia vektoriaus dydį
- reference at (size_type pos)

  Gražina elementą pagal nurodytą indeksą
- const_reference at (size_type pos) const

  Gražina elementą pagal nurodytą indeksą
- T & operator[ ] (size_t index)

  Gražina elementą pagal nurodytą indeksą
- const T & operator[ ] (size_t index) const

  Gražina elementą pagal nurodytą indeksą
- bool operator== (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- bool operator!= (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- bool operator< (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- bool operator<= (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- bool operator> (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- bool operator>= (const Vektorius< T > &other) const

  Operatorius, lyginantis du vektorius.
- iterator begin ()

  Gražina iteratorių, nurodantį į pirmą vektoriaus elementą
- const_iterator begin () const
- const_iterator cbegin () const
- reverse_iterator rbegin ()

  Gražina atbula iteratorių, nurodantį į paskutinį vektoriaus elementą
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const noexcept
- iterator end ()

  Gražina iteratorių, nurodantį į vektoriaus pabaigą
- const_iterator end () const
- const_iterator cend () const
- reverse_iterator rend ()

  Gražina atbula iteratorių, nurodantį į vektoriaus pabaigą
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const noexcept
- reference front ()

  Gražina pirmą elementą
- const_reference front () const

- reference back ()

  Gražina paskutinį elementą
- const_reference back () const
- T * storage ()

  Gražina vektoriaus duomenų masyvą (taspats kaip std::vector::data())
- const T * storage () const

5.3.1 Detailed Description

template<typename T>
class Vektorius< T >

Definition at line 10 of file Vektorius.h.

5.3.2 Member Typedef Documentation

5.3.2.1 const_iterator

```
template<typename T>
using Vektorius< T >::const_iterator = const T*
```

Definition at line 30 of file Vektorius.h.

5.3.2.2 const_reference

```
template<typename T>
using Vektorius< T >::const_reference = const T&
```

Definition at line 34 of file Vektorius.h.

5.3.2.3 const_reverse_iterator

```
template<typename T>
using Vektorius< T >::const_reverse_iterator = std::reverse_iterator<const_iterator>
```

Definition at line 36 of file Vektorius.h.

5.3.2.4 difference_type

```
template<typename T>
using Vektorius< T >::difference_type = std::ptrdiff_t
```

Definition at line 32 of file Vektorius.h.

5.3.2.5 iterator

```
template<typename T>
using Vektorius< T >::iterator = T*
```

Definition at line 29 of file Vektorius.h.

5.3.2.6 reference

```
template<typename T>
using Vektorius< T >::reference = T&
```

Definition at line 33 of file Vektorius.h.

5.3.2.7 reverse_iterator

```
template<typename T>
using Vektorius< T >::reverse_iterator = std::reverse_iterator<iterator>
```

Definition at line 35 of file Vektorius.h.

5.3.2.8 size_type

```
template<typename T>
using Vektorius< T >::size_type = std::size_t
```

Definition at line 31 of file Vektorius.h.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 Vektorius() [1/6]

```
template<typename T>
Vektorius< T >::Vektorius ()  [inline]
```

Default construktorius, sukuria tuscia vektoriu.

Definition at line 42 of file Vektorius.h.

5.3.3.2 Vektorius() [2/6]

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

Konstruktorius, sukuria vektoriu su nurodytu dydziu.

Parameters

size	- norimas dydis

Definition at line 48 of file Vektorius.h.

5.3.3.3 Vektorius() [3/6]

```
template<typename T>
Vektorius< T >::Vektorius (
            size_t size,
            const T & value)  [inline]
```

Konstruktorius, sukuria vektoriu su nurodytu dydziu ir uzpildo jį nurodyta reiksme.

Parameters

size	- norimas dydis
value	- reiksme, kuria uzpildomas vektorius

Definition at line 60 of file Vektorius.h.

5.3.3.4 Vektorius() [4/6]

```
template<typename T>
Vektorius< T >::Vektorius (
            std::initializer_list< T > ilist)  [inline]
```

Konstruktorius su initializer_list.

Parameters

ilist	- sąrašas pradinių reikšmių

Definition at line 71 of file Vektorius.h.

5.3.3.5 ~Vektorius()

```
template<typename T>
Vektorius< T >::~Vektorius ()  [inline]
```

Destruktorius.

Definition at line 83 of file Vektorius.h.

5.3.3.6 Vektorius() [5/6]

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

Copy constructorius.

Definition at line 90 of file Vektorius.h.

5.3.3.7 Vektorius() [6/6]

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

Move constructorius.

Definition at line 118 of file Vektorius.h.

5.3.4 Member Function Documentation

5.3.4.1 assign() [1/3]

```
template<typename T>
template<typename InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt>>>
void Vektorius< T >::assign (
            InputIt first,
            InputIt last)  [inline]
```

Priskiria vektoriui nauja reiksme.

Parameters

first	- iteratorius, nurodantis pradzia
last	- iteratorius, nurodantis pabaiga

Definition at line 173 of file Vektorius.h.

5.3.4.2 assign() [2/3]

```
template<typename T>
void Vektorius< T >::assign (
            size_t count,
            const T & value)  [inline]
```

Priskiria vektoriui nauja reiksme.

Parameters

count	- norimas dydis
value	- reiksme, kuria uzpildomas vektorius

Definition at line 156 of file Vektorius.h.

5.3.4.3 assign() [3/3]

```
template<typename T>
void Vektorius< T >::assign (
            std::initializer_list< T > ilist)  [inline]
```

Priskiria vektoriui nauja reiksme.

Parameters

ilist	- inicializavimo sarasas

Definition at line 191 of file Vektorius.h.

5.3.4.4 at() [1/2]

```
template<typename T>
reference Vektorius< T >::at (
            size_type pos)  [inline]
```

Gražina elementą pagal nurodytą indeksą

Parameters

pos	- indeksas

Returns

- elementas

Exceptions

std::out_of_range	- jei indeksas yra už ribų

Definition at line 449 of file Vektorius.h.

5.3.4.5 at() [2/2]

```
template<typename T>
const_reference Vektorius< T >::at (
            size_type pos) const  [inline]
```

Gražina elementą pagal nurodytą indeksą

Parameters

pos	- indeksas

Returns

- elementas

Exceptions

std::out_of_range	- jei indeksas yra už ribų

Definition at line 462 of file Vektorius.h.

5.3.4.6 back() [1/2]

```
template<typename T>
reference Vektorius< T >::back ()  [inline]
```

Gražina paskutinį elementą

Returns

- paskutinis elementas

Definition at line 615 of file Vektorius.h.

5.3.4.7 back() [2/2]

```
template<typename T>
const_reference Vektorius< T >::back () const  [inline]
```

Definition at line 621 of file Vektorius.h.

5.3.4.8 begin() [1/2]

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

Gražina iteratorių, nurodantį į pirmą vektoriaus elementą

Returns

- iteratorius, nurodantis į pirmą elementą

Definition at line 566 of file Vektorius.h.

5.3.4.9 begin() [2/2]

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

Definition at line 567 of file Vektorius.h.

5.3.4.10 capacity()

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

Gražina vektoriaus talpą

Definition at line 402 of file Vektorius.h.

5.3.4.11 cbegin()

```
template<typename T>
const_iterator Vektorius< T >::cbegin () const  [inline]
```

Definition at line 568 of file Vektorius.h.

5.3.4.12 cend()

```
template<typename T>
const_iterator Vektorius< T >::cend () const  [inline]
```

Definition at line 584 of file Vektorius.h.

5.3.4.13 clear()

```
template<typename T>
void Vektorius< T >::clear ()  [inline]
```

Ištrina visus elementus iš vektoriaus.

Definition at line 412 of file Vektorius.h.

5.3.4.14 crbegin()

```
template<typename T>
const_reverse_iterator Vektorius< T >::crbegin () const  [inline], [noexcept]
```

Definition at line 576 of file Vektorius.h.

5.3.4.15 crend()

```
template<typename T>
const_reverse_iterator Vektorius< T >::crend () const  [inline], [noexcept]
```

Definition at line 592 of file Vektorius.h.

5.3.4.16 emplace()

```
template<typename T>
template<class...  Args>
iterator Vektorius< T >::emplace (
            const_iterator pos,
            Args &&...  args)  [inline]
```

sukuriamas elementą į vektorių su emplace

Parameters

pos	- iteratorius, nurodantis vietą, kur reikia įterpti
args	- argumentai, kuriuos reikia perduoti elementui

Returns

iteratorius, nurodantis į įterptą elementą

Definition at line 358 of file Vektorius.h.

5.3.4.17 empty()

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

Gražina ar vektorius yra tuscias.

Definition at line 407 of file Vektorius.h.

5.3.4.18 end() [1/2]

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

Gražina iteratorių, nurodantį į vektoriaus pabaigą

Returns

- iteratorius, nurodantis į vektoriaus pabaigą

Definition at line 582 of file Vektorius.h.

5.3.4.19 end() [2/2]

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

Definition at line 583 of file Vektorius.h.

5.3.4.20 erase() [1/2]

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

Ištrina elementus iš vektoriaus.

Parameters

first	- iteratorius, nurodantis nuo kur trinti
last	- iteratorius, nurodantis iki kur trinti

Returns

iteratorius, nurodantis į elementą po ištrinto

Definition at line 241 of file Vektorius.h.

5.3.4.21 erase() [2/2]

```
template<typename T>
iterator Vektorius< T >::erase (
            const_iterator pos)  [inline]
```

Ištrina elementą iš vektoriaus.

Parameters

pos	- iteratorius, nurodantis elementą, kurį reikia ištrinti

Returns

iteratorius, nurodantis į elementą po ištrinto

Definition at line 223 of file Vektorius.h.

5.3.4.22 front() [1/2]

```
template<typename T>
reference Vektorius< T >::front ()  [inline]
```

Gražina pirmą elementą

Returns

- pirmas elementas

Definition at line 598 of file Vektorius.h.

5.3.4.23 front() [2/2]

```
template<typename T>
const_reference Vektorius< T >::front () const  [inline]
```

Definition at line 604 of file Vektorius.h.

5.3.4.24 insert() [1/4]

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

Įterpia elementą į vektorių

Parameters

pos	- iteratorius, nurodantis vietą, kur reikia įterpti
value	- reiksme, kurią reikia įterpti

Returns

iteratorius, nurodantis į įterptą elementą

Definition at line 261 of file Vektorius.h.

5.3.4.25 insert() [2/4]

```
template<typename T>
template<class InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt>>>
iterator Vektorius< T >::insert (
            const_iterator pos,
            InputIt first,
            InputIt last)  [inline]
```

Įterpia kelis elementus į vektorių per pointerius.

Parameters

pos	- iteratorius, nurodantis vietą, kur reikia įterpti
first	- iteratorius, nurodantis duomenu pradzia
last	- iteratorius, nurodantis duomenu pabaiga

Returns

iteratorius, nurodantis į pirmą įterptą elementą

Definition at line 332 of file Vektorius.h.

5.3.4.26 insert() [3/4]

```
template<typename T>
iterator Vektorius< T >::insert (
            const_iterator pos,
            size_type count,
            const T & value)  [inline]
```

Įterpia kelis elementus į vektorių

Parameters

pos	- iteratorius, nurodantis vietą, kur reikia įterpti
count	- kiek elementų reikia įterpti
value	- reiksme, kurią reikia įterpti

Returns

iteratorius, nurodantis į pirmą įterptą elementą

Definition at line 306 of file Vektorius.h.

5.3.4.27 insert() [4/4]

```
template<typename T>
iterator Vektorius< T >::insert (
            const_iterator pos,
            T && value)  [inline]
```

Įterpia elementą į vektorių naudodamas move semantika.

Parameters

pos	- iteratorius, nurodantis vietą, kur reikia įterpti
value	- reiksme, kurią reikia įterpti

Returns

iteratorius, nurodantis į įterptą elementą

Definition at line 283 of file Vektorius.h.

5.3.4.28 operator"!=()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei vektoriai yra nelygūs

Definition at line 506 of file Vektorius.h.

5.3.4.29 operator<()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei pirmas vektorius mažesnis už antrą

Definition at line 515 of file Vektorius.h.

5.3.4.30 operator<=()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei pirmas vektorius mažesnis arba lygus antram

Definition at line 528 of file Vektorius.h.

5.3.4.31 operator=() [1/2]

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

Copy assignment operatorius.

Definition at line 102 of file Vektorius.h.

5.3.4.32 operator=() [2/2]

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

Move assignment operatorius.

Definition at line 129 of file Vektorius.h.

5.3.4.33 operator==()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei vektoriai yra lygūs

Definition at line 493 of file Vektorius.h.

5.3.4.34 operator>()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei pirmas vektorius didesnis už antrą

Definition at line 541 of file Vektorius.h.

5.3.4.35 operator>=()

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

Operatorius, lyginantis du vektorius.

Parameters

other	- kitas vektorius

Returns

- true, jei pirmas vektorius didesnis arba lygus antram

Definition at line 554 of file Vektorius.h.

5.3.4.36 operator[]() [1/2]

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

Gražina elementą pagal nurodytą indeksą

Parameters

index	- indeksas

Returns

- elementas

Definition at line 475 of file Vektorius.h.

5.3.4.37 operator[]() [2/2]

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

Gražina elementą pagal nurodytą indeksą

Parameters

index	- indeksas

Returns

- elementas

Definition at line 484 of file Vektorius.h.

5.3.4.38 pop_back()

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

Ištrina paskutinį elementą iš vektoriaus.

Returns

- paskutinio elemento reiksme

Definition at line 390 of file Vektorius.h.

5.3.4.39 push_back()

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

Prideda elementą į vektoriaus gala.

Parameters

value	- reiksme, kurią prides

Definition at line 379 of file Vektorius.h.

5.3.4.40 rbegin() [1/2]

```
template<typename T>
reverse_iterator Vektorius< T >::rbegin ()  [inline]
```

Gražina atbula iteratorių, nurodantį į paskutinį vektoriaus elementą

Returns

- atbulas iteratorius, nurodantis į paskutinį elementą

Definition at line 574 of file Vektorius.h.

5.3.4.41 rbegin() [2/2]

```
template<typename T>
const_reverse_iterator Vektorius< T >::rbegin () const  [inline]
```

Definition at line 575 of file Vektorius.h.

5.3.4.42 rend() [1/2]

```
template<typename T>
reverse_iterator Vektorius< T >::rend ()  [inline]
```

Gražina atbula iteratorių, nurodantį į vektoriaus pabaigą

Returns

- atbulas iteratorius, nurodantis į vektoriaus pabaigą

Definition at line 590 of file Vektorius.h.

5.3.4.43 rend() [2/2]

```
template<typename T>
const_reverse_iterator Vektorius< T >::rend () const  [inline]
```

Definition at line 591 of file Vektorius.h.

5.3.4.44 reserve()

```
template<typename T>
void Vektorius< T >::reserve (
            size_t new_cap)  [inline]
```

Rezervuoja talpą vektoriui.

Parameters

new_cap	- nauja talpa

Definition at line 423 of file Vektorius.h.

5.3.4.45 resize()

```
template<typename T>
void Vektorius< T >::resize (
            size_t count)  [inline]
```

Pakeičia vektoriaus dydį

Parameters

count	- naujas dydis

Definition at line 432 of file Vektorius.h.

5.3.4.46 shrink_to_fit()

```
template<typename T>
void Vektorius< T >::shrink_to_fit ()  [inline]
```

Sumažina vektoriaus talpą iki jo dydžio.

Definition at line 206 of file Vektorius.h.

5.3.4.47 size()

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

Gražina vektoriaus dydį

Definition at line 397 of file Vektorius.h.

5.3.4.48 storage() [1/2]

```
template<typename T>
T * Vektorius< T >::storage ()  [inline]
```

Gražina vektoriaus duomenų masyvą (taspats kaip std::vector::data())

Returns

- duomenų masyvas

Definition at line 632 of file Vektorius.h.

5.3.4.49 storage() [2/2]

```
template<typename T>
const T * Vektorius< T >::storage () const  [inline]
```

Definition at line 635 of file Vektorius.h.

5.3.4.50 swap()

```
template<typename T>
void Vektorius< T >::swap (
            Vektorius< T > & other)  [inline]
```

Swap funkcija.

Parameters

other	- kitas vektorius

Definition at line 145 of file Vektorius.h.

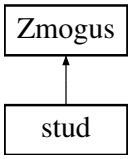
The documentation for this class was generated from the following file:

- Vektorius.h

5.4 Zmogus Class Reference

`#include <Zmogus.h>`

Inheritance diagram for Zmogus:



Public Member Functions

- Zmogus ()=default
- Zmogus (const string &vard, const string &pava)
- virtual ~Zmogus ()=default
- Zmogus (const Zmogus &other)
- Zmogus & operator= (const Zmogus &other)
- Zmogus (Zmogus &&other) noexcept
- Zmogus & operator= (Zmogus &&other) noexcept
- void setvard (const string &vardas)
- void setpava (const string &pavard)
- string getVardas () const
- string getPavarde () const

Protected Attributes

- string vard
- string pava

5.4.1 Detailed Description

Definition at line 7 of file Zmogus.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Zmogus() [1/4]

```
Zmogus::Zmogus ()  [default]
```

5.4.2.2 Zmogus() [2/4]

```
Zmogus::Zmogus (
            const string & vard,
            const string & pava)  [inline]
```

Definition at line 12 of file Zmogus.h.

5.4.2.3 ~Zmogus()

```
virtual Zmogus::~Zmogus ()  [virtual], [default]
```

5.4.2.4 Zmogus() [3/4]

```
Zmogus::Zmogus (
            const Zmogus & other)  [inline]
```

Definition at line 18 of file Zmogus.h.

5.4.2.5 Zmogus() [4/4]

```
Zmogus::Zmogus (
            Zmogus && other)  [inline], [noexcept]
```

Definition at line 31 of file Zmogus.h.

5.4.3 Member Function Documentation

5.4.3.1 getPavarde()

```
string Zmogus::getPavarde () const  [inline]
```

Definition at line 47 of file Zmogus.h.

5.4.3.2 getVardas()

```
string Zmogus::getVardas () const  [inline]
```

Definition at line 46 of file Zmogus.h.

5.4.3.3 operator=() [1/2]

```
Zmogus & Zmogus::operator= (
            const Zmogus & other)  [inline]
```

Definition at line 22 of file Zmogus.h.

5.4.3.4 operator=() [2/2]

```
Zmogus & Zmogus::operator= (
            Zmogus && other)  [inline], [noexcept]
```

Definition at line 35 of file Zmogus.h.

5.4.3.5 setpava()

```
void Zmogus::setpava (
            const string & pavard)  [inline]
```

Definition at line 44 of file Zmogus.h.

5.4.3.6 setvard()

```
void Zmogus::setvard (
            const string & vardas)  [inline]
```

Definition at line 43 of file Zmogus.h.

5.4.4 Member Data Documentation

5.4.4.1 pava

```
string Zmogus::pava  [protected]
```

Definition at line 51 of file Zmogus.h.

5.4.4.2 vard

```
string Zmogus::vard  [protected]
```

Definition at line 50 of file Zmogus.h.

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

- Zmogus.h

Chapter 6

File Documentation

6.1 automatiskas.cpp File Reference

```
#include "bibl.h"
```

Functions

- void automatiskas (Vektorius< stud > &A)

Variables

- string Vard [5] {"Jonas", "Vytautas", "Antanas", "Tomas", "Juozas"}
- string Pava [5] {"Kazlauskas", "Stankevicius", "Petrauskas", "Janauskas", "Zukauskas"}

6.1.1 Function Documentation

6.1.1.1 automatiskas()

```
void automatiskas (
            Vektorius< stud > & A)
```

Definition at line 6 of file automatiskas.cpp.

6.1.2 Variable Documentation

6.1.2.1 Pava

```
string Pava[5] {"Kazlauskas", "Stankevicius", "Petrauskas", "Janauskas", "Zukauskas"}
```

Definition at line 4 of file automatiskas.cpp.

6.1.2.2 Vard

```
string Vard[5] {"Jonas", "Vytautas", "Antanas", "Tomas", "Juozas"}
```

Definition at line 3 of file automatiskas.cpp.

6.2 automatiskas.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 string Vard[5] {"Jonas", "Vytautas", "Antanas", "Tomas", "Juozas"};
00004 string Pava[5] {"Kazlauskas", "Stankevicius", "Petrauskas", "Janauskas", "Zukauskas"};
00005
00006 void automatiskas (Vektorius <stud> &A){
00007     std::random_device rd;
00008     std::mt19937 mt(rd());
00009     std::uniform_int_distribution<int> vardui(0,4);
00010     std::uniform_int_distribution<int> pazymiui(0,10);
00011
00012     stud temp(Vard[vardui(mt)],Pava[vardui(mt)]);
00013
00014
00015     for (int i=0;i<pazymiui(mt);i++){
00016         int paz=pazymiui(mt);
00017         temp.addTarpPazymys(paz);
00018     }
00019     temp.setEgzaminas(pazymiui(mt));
00020
00021     temp.calculateGalutinis();
00022     A.push_back(temp);
00023 }
```

6.3 bibl.h File Reference

```
#include "stud.h"
#include "std.h"
#include "Vektorius.h"
#include "Zmogus.h"
#include "Timer.h"
```

Functions

- void spausdina (Vektorius< stud >)
- void rankinis (Vektorius< stud > &)
- void pusrankis (Vektorius< stud > &)
- void automatiskas (Vektorius< stud > &)
- void compare (Vektorius< stud > &, double &)
- void compare (Vektorius< stud > &)
- void failoNusk (Vektorius< stud > &)
- void failoGen ()
- void rusiavimas ()
- void spausdinaFaila (Vektorius< stud > &A, string)
- void test ()
- int vectorTest ()
- int run_unit_tests ()
- bool compVardas (stud &, stud &)
- bool compPavard (stud &, stud &)
- bool compVid (stud &, stud &)
- bool compMed (stud &, stud &)

Variables

- string Vard [5]
- string Pava [5]

6.3.1 Function Documentation

6.3.1.1 automatiskas()

```
void automatiskas (
            Vektorius< stud > & A)
```

Definition at line 6 of file automatiskas.cpp.

6.3.1.2 compare() [1/2]

```
void compare (
            Vektorius< stud > & A)
```

Definition at line 3 of file compare.cpp.

6.3.1.3 compare() [2/2]

```
void compare (
            Vektorius< stud > & A,
            double & time)
```

Definition at line 43 of file compare.cpp.

6.3.1.4 compMed()

```
bool compMed (
            stud & a,
            stud & b)
```

Definition at line 98 of file compare.cpp.

6.3.1.5 compPavard()

```
bool compPavard (
            stud & a,
            stud & b)
```

Definition at line 90 of file compare.cpp.

6.3.1.6 compVardas()

```
bool compVardas (
            stud & a,
            stud & b)
```

Definition at line 86 of file compare.cpp.

6.3.1.7 compVid()

```
bool compVid (
            stud & a,
            stud & b)
```

Definition at line 94 of file compare.cpp.

6.3.1.8 failoGen()

```
void failoGen ()
```

Definition at line 3 of file failoGen.cpp.

6.3.1.9 failoNusk()

```
void failoNusk (
            Vektorius< stud > & A)
```

Definition at line 3 of file failoNusk.cpp.

6.3.1.10 pusrankis()

```
void pusrankis (
            Vektorius< stud > & A)
```

Definition at line 3 of file pusrankis.cpp.

6.3.1.11 rankinis()

```
void rankinis (
            Vektorius< stud > & A)
```

Definition at line 3 of file rankinis.cpp.

6.3.1.12 run_unit_tests()

```
int run_unit_tests ()
```

Definition at line 4 of file unit_test.cpp.

6.3.1.13 rusiavimas()

```
void rusiavimas ()
```

Definition at line 3 of file rusiavimas.cpp.

6.3.1.14 spausdina()

```
void spausdina (
            Vektorius< stud > A)
```

Definition at line 3 of file spausdina.cpp.

6.3.1.15 spausdinaFaila()

```
void spausdinaFaila (
            Vektorius< stud > & A,
            string failas)
```

Definition at line 12 of file spausdina.cpp.

6.3.1.16 test()

```
void test ()
```

Definition at line 99 of file test.cpp.

6.3.1.17 vectorTest()

```
int vectorTest ()
```

Definition at line 7 of file vectortest.cpp.

6.3.2 Variable Documentation

6.3.2.1 Pava

```
string Pava[5]  [extern]
```

Definition at line 4 of file automatiskas.cpp.

6.3.2.2 Vard

```
string Vard[5]  [extern]
```

Definition at line 3 of file automatiskas.cpp.

6.4 bibl.h

Go to the documentation of this file.

```
00001 # ifndef BIBL_H
00002 # define BIBL_H
00003
00004
00005 #include "stud.h"
00006 #include "std.h"
00007 #include "Vektorius.h"
00008 #include "Zmogus.h"
00009 #include "Timer.h"
00010
00011
00012 void spausdina(Vektorius <stud> );
00013 void rankinis(Vektorius <stud> &);
00014 void pusrankis(Vektorius <stud> &);
00015 void automatiskas (Vektorius <stud> &);
00016 void compare(Vektorius <stud> &, double &);
00017 void compare(Vektorius <stud> &);
00018 void failoNusk (Vektorius <stud> &);
00019 void failoGen();
00020 void rusiavimas();
00021 void spausdinaFaila(Vektorius <stud> &A, string);
00022 void test();
00023 int vectorTest();
00024 int run_unit_tests();
00025
00026 bool compVardas(stud &, stud &);
00027 bool compPavard(stud &, stud &);
00028 bool compVid(stud &, stud &);
00029 bool compMed(stud &, stud &);
00030
00031 extern string Vard[5];
00032 extern string Pava[5];
00033
00034
00035 # endif
```

6.5 compare.cpp File Reference

```
#include "bibl.h"
```

Functions

- void compare (Vektorius< stud > &A)
- void compare (Vektorius< stud > &A, double &time)
- bool compVardas (stud &a, stud &b)
- bool compPavard (stud &a, stud &b)
- bool compVid (stud &a, stud &b)
- bool compMed (stud &a, stud &b)

6.5.1 Function Documentation

6.5.1.1 compare() [1/2]

```
void compare (
            Vektorius< stud > & A)
```

Definition at line 3 of file compare.cpp.

6.5.1.2 compare() [2/2]

```
void compare (
            Vektorius< stud > & A,
            double & time)
```

Definition at line 43 of file compare.cpp.

6.5.1.3 compMed()

```
bool compMed (
            stud & a,
            stud & b)
```

Definition at line 98 of file compare.cpp.

6.5.1.4 compPavard()

```
bool compPavard (
            stud & a,
            stud & b)
```

Definition at line 90 of file compare.cpp.

6.5.1.5 compVardas()

```
bool compVardas (
            stud & a,
            stud & b)
```

Definition at line 86 of file compare.cpp.

6.5.1.6 compVid()

```
bool compVid (
            stud & a,
            stud & b)
```

Definition at line 94 of file compare.cpp.

6.6 compare.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void compare(Vektorius <stud> &A){
00004     while(true){
00005         cout « "1 - Pagal Varda " « endl;
00006         cout « "2 - Pagal Pavarde " « endl;
00007         cout « "3 - Pagal pazymiu vidurki " « endl;
00008         cout « "4 - Pagal pazymiu mediana " « endl;
00009         int input;
00010         try {
00011             if (!(cin»input)||input<1 || input>4){
00012                 cin.clear();
00013                 cin.ignore();
00014                 throw "Ivestas neteisingas simbolis";
00015             }
00016             switch(input){
00017                 case 1:
00018                     std::sort(A.begin(),A.end(), compVardas);
00019                     break;
00020
00021                 case 2:
00022                     std::sort(A.begin(),A.end(), compPavard);
00023                     break;
00024
00025                 case 3:
00026                     std::sort(A.begin(),A.end(), compVid);
00027                     break;
00028
00029                 case 4:
00030                     std::sort(A.begin(),A.end(), compMed);
00031                     break;
00032             }
00033             break;
00034         }
00035         catch (char const *x){
00036             cout « x « endl;
00037             continue;
00038             cout « "Pagal ka isrusiuoti duomenis?" « endl;
00039         }
00040     }
00041 }
00042
00043 void compare(Vektorius <stud> &A,double &time){
00044     Timer laik;
00045     while(true){
00046         cout « "1 - Pagal Varda " « endl;
00047         cout « "2 - Pagal Pavarde " « endl;
00048         cout « "3 - Pagal pazymiu vidurki " « endl;
00049         cout « "4 - Pagal pazymiu mediana " « endl;
00050         int input;
00051         try {
00052             if (!(cin»input)||input<1 || input>4){
00053                 cin.clear();
00054                 cin.ignore();
00055                 throw "Ivestas neteisingas simbolis";
00056             }
00057             laik.reset();
00058             switch(input){
00059                 case 1:
00060                     std::sort(A.begin(),A.end(), compVardas);
00061                     break;
00062
00063                 case 2:
00064                     std::sort(A.begin(),A.end(), compPavard);
00065                     break;
00066
00067                 case 3:
00068                     std::sort(A.begin(),A.end(), compVid);
00069                     break;
00070
00071                 case 4:
00072                     std::sort(A.begin(),A.end(), compMed);
00073                     break;
00074             }
00075             time+=laik.elapsed();
00076             break;
00077         }
00078         catch (char const *x){
00079             cout « x « endl;
00080             continue;
00081             cout « "Pagal ka isrusiuoti duomenis?" « endl;
00082         }
```

```
00083     }
00084 }
00085
00086 bool compVardas(stud &a, stud &b){
00087     return a.getVardas()<b.getVardas();
00088   }
00089
00090   bool compPavard(stud &a, stud &b){
00091     return a.getPavarde()<b.getPavarde();
00092   }
00093
00094   bool compVid(stud &a, stud &b){
00095     return a.getGalutinisVid()<b.getGalutinisVid();
00096   }
00097
00098   bool compMed(stud &a, stud &b){
00099     return a.getGalutinisMed()<b.getGalutinisMed();
00100   }
```

6.7 failoGen.cpp File Reference

```
#include "bibl.h"
```

Functions

- void failoGen ()

6.7.1 Function Documentation

6.7.1.1 failoGen()

```
void failoGen ()
```

Definition at line 3 of file failoGen.cpp.

6.8 failoGen.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void failoGen(){
00004     string failas;
00005     int kiek;
00006     int pazkiek;
00007     std::random_device rd;
00008     std::mt19937 mt(rd());
00009     std::uniform_int_distribution <int> pazymiui(0,10);
00010     Timer t;
00011
00012     cout « "Iveskite failo pavadinima (pvz. kursiokai)" « endl;
00013     cin » failas;
00014     cout « "Iveskite kiek sugeneruoti studentu" « endl;
00015     cin » kiek;
00016     cout « "Iveskite kiek pazymiu tures studentai (neskaiciuojant egzamino)" « endl;
00017     cin » pazkiek;
00018
00019     t.reset();
00020     std::stringstream eil;
00021
00022     std::ofstream rf(failas+".txt");
00023
00024     eil «std::left «setw(15)« "Vardas"« setw(15) « "Pavarde" ;
```

```
00025     for (int i=1;i<=pazkiek;i++){
00026         eil  « "ND"« setw(5) «std::to_string(i);
00027     }
00028     eil «  "Egz." « "\n";
00029
00030     rf « eil.str();
00031
00032     eil.str("");
00033
00034     for (int i=1;i<=kiek;i++){
00035         eil«setw(15) «"Vardas" + to_string(i) «setw(15)« "Pavarde" + to_string(i);
00036         for (int j=0;j<pazkiek;j++){
00037             eil « setw(7) « pazymiui(mt);
00038         }
00039         eil « setw(7) « pazymiui(mt) « "\n";
00040         rf « eil.str();
00041         eil.str("");
00042     }
00043     rf.close();
00044     cout « "failu kurimas ir jo uzdarymas uztruko " « t.elapsed() « endl;
00045 }
```

6.9 failoNusk.cpp File Reference

```
#include "bibl.h"
```

Functions

- void failoNusk (Vektorius< stud > &A)

6.9.1 Function Documentation

6.9.1.1 failoNusk()

```
void failoNusk (
            Vektorius< stud > & A)
```

Definition at line 3 of file failoNusk.cpp.

6.10 failoNusk.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void failoNusk (Vektorius <stud> &A){
00004     string failas;
00005
00006     cout « "Iveskite failo pavadinima (pvz. kursiokai.txt)" « endl;
00007     while(true){
00008         cin » failas;
00009         if (!(std::filesystem::exists(failas))){
00010             cout « "Toks failas neegzistuoja, pabandykite vel" « endl;
00011             continue;
00012         }
00013         break;
00014     }
00015
00016     string eil;
00017     Timer t;
00018
00019     std::ifstream df(failas);
```

```
00020     getline(df,eil);
00021
00022     while(getline(df,eil)){
00023         std::istringstream line(eil);
00024         stud temp(line);
00025         temp.calculateGalutinis();
00026         A.push_back(temp);
00027     }
00028     cout « "Perskaityt ir suskaiciuot vidurkius uztruko " « t.elapsed() « endl;
00029     df.close();
00030 }
```

6.11 main.cpp File Reference

```
#include "bibl.h"
```

Functions

- int main ()

Variables

- Vektorius< stud > A

6.11.1 Function Documentation

6.11.1.1 main()

```
int main ()
```

Definition at line 5 of file main.cpp.

6.11.2 Variable Documentation

6.11.2.1 A

```
Vektorius<stud> A
```

Definition at line 3 of file main.cpp.

6.12 main.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 Vektorius <stud> A;
00004
00005 int main(){
00006     int input;
00007     string failas;
00008     while ((true)){
00009         cout « "Iveskite skaiciu kokiu budu norite ivesti duomenis " « endl;
00010         cout « "1 - Iveskite visus duomenis rankiniu budu " « endl;
00011         cout « "2 - Iveskite varda ir pavarde rankniu budu " « endl;
00012         cout « "3 - Sugeneruoti visus duomenis automatiskai " « endl;
00013         cout « "4 - Paiimti duomenis is failo " « endl;
00014         cout « "5 - Sugeneruoti nauja duomenu faila " « endl;
00015         cout « "6 - Surusiuoti faila i vargsiukus ir kietiakus " « endl;
00016         cout « "7 - Baigti darba ir spausdinti " « endl;
00017         cout « "8 - Vektoriaus laiko testavimas atvejai " « endl;
00018         cout « "9 - Vektoriaus unit testai " « endl;
00019         try {
00020             if (!(cin»input)||input<1 || input>9){
00021                 cin.clear();
00022                 cin.ignore();
00023                 throw "Ivestas neteisingas simbolis";
00024             }
00025
00026             switch(input){
00027                 case 1:
00028                     rankinis(A);
00029                     break;
00030
00031                 case 2:
00032                     pusrankis(A);
00033                     break;
00034
00035                 case 3:
00036                     int n;
00037                     cout « "Iveskite kiek mokiniu generuoti" « endl;
00038                     cin » n;
00039                     for (int i=0;i<n;i++){
00040                         automatiskas(A);
00041                     }
00042                     break;
00043
00044                 case 4:
00045                     failoNusk(A);
00046                     break;
00047
00048                 case 5:
00049                     failoGen();
00050                     break;
00051
00052                 case 6:
00053                     rusiavimas();
00054                     break;
00055
00056                 case 7:
00057                     cout « "Pagal ka isrusiuoti duomenis?" « endl;
00058                     compare(A);
00059                     spausdina(A);
00060                     cout « "Spauskite Enter, kad uzdaryti programa..." « endl;
00061                     cin.ignore();
00062                     cin.get();
00063                     return 0;
00064                 case 8:
00065                     vectorTest();
00066                     cout « "Spauskite Enter, kad uzdaryti programa..." « endl;
00067                     cin.ignore();
00068                     cin.get();
00069                     return 0;
00070                 case 9:
00071                     run_unit_tests();
00072                     cout « "Spauskite Enter, kad uzdaryti programa..." « endl;
00073                     cin.ignore();
00074                     cin.get();
00075                     return 0;
00076                 default:
00077                     cout « "Ivedete neteisinga simobli, pabandykit vel! :)" « endl;
00078                     break;
00079         }
00080         }
00081         catch (char const *x){
00082             cout « x « endl;
```

```
00083             continue;
00084         }
00085
00086     }
00087 cout « "Spauskite Enter, kad uzdaryti programa..." « endl;
00088 cin.ignore();
00089 cin.get();
00090  return 0;
00091 }
```

6.13 pusrankis.cpp File Reference

```
#include "bibl.h"
```

Functions

- void pusrankis (Vektorius< stud > &A)

6.13.1 Function Documentation

6.13.1.1 pusrankis()

```
void pusrankis (
             Vektorius< stud > & A)
```

Definition at line 3 of file pusrankis.cpp.

6.14 pusrankis.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void pusrankis(Vektorius <stud> &A){
00004     std::random_device rd;
00005     std::mt19937 mt(rd());
00006     std::uniform_int_distribution <int> dist(0,10);
00007     string vardas;
00008     string pavard;
00009
00010     cout « "Iveskite studento Varda ir pavarde ";
00011     cin » vardas » pavard;
00012     stud temp(vardas,pavard);
00013     for (int i=0;i<dist(mt);i++){
00014         int paz=dist(mt);
00015         temp.addTarpPazymys(paz);
00016     }
00017     temp.setEgzaminas(dist(mt));
00018
00019     temp.calculateGalutinis();
00020     A.push_back(temp);
00021 }
```

6.15 rankinis.cpp File Reference

```
#include "bibl.h"
```

Functions

- void rankinis (Vektorius< stud > &A)

6.15.1 Function Documentation

6.15.1.1 rankinis()

```
void rankinis (
            Vektorius< stud > & A)
```

Definition at line 3 of file rankinis.cpp.

6.16 rankinis.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void rankinis(Vektorius <stud> &A){
00004     string input;
00005     string vardas;
00006     string pavard;
00007     stud temp;
00008     cout « "Iveskite studento Varda ir pavarde ";
00009     cin » temp;
00010     temp.calculateGalutinis();
00011     A.push_back(temp);
00012 }
```

6.17 README.md File Reference

6.18 rusiavimas.cpp File Reference

```
#include "bibl.h"
```

Functions

- void rusiavimas ()

6.18.1 Function Documentation

6.18.1.1 rusiavimas()

```
void rusiavimas ()
```

Definition at line 3 of file rusiavimas.cpp.

6.19 rusiavimas.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void rusiavimas(){
00004     string failas;
00005     Vektorius <stud> visi;
00006     Vektorius <stud> nuskriausti;
00007     bool vid;
00008     double visaTrukme=0;
00009     cout « "Iveskite failo pavadinima (pvz. kursiokai.txt)" « endl;
00010     while(true){
00011         cin » failas;
00012         if (!(std::filesystem::exists(failas))){
00013             cout « "Toks failas neegzistuoja, pabandykite vel" « endl;
00014             continue;
00015         }
00016         break;
00017     }
00018
00019     while(true){
00020         cout « "Pagal ka atrinkti studentus?" « endl;
00021         cout « "1 - Pagal pazymiu vidurki " « endl;
00022         cout « "2 - Pagal pazymiu mediana " « endl;
00023         int input;
00024         try {
00025             if (!(cin»input)||input<1 || input>2){
00026                 cin.clear();
00027                 cin.ignore();
00028                 throw "Ivestas neteisingas simbolis";
00029             }
00030             switch(input){
00031                 case 1:
00032                     vid=1;
00033                     break;
00034
00035                 case 2:
00036                     vid=0;
00037                     break;
00038             }
00039         break;
00040         }
00041         catch (char const *x){
00042             cout « x « endl;
00043             continue;
00044         }
00045     }
00046
00047     string eil;
00048     Timer t;
00049     std::ifstream df(failas);
00050     getline(df,eil);
00051
00052     while(getline(df,eil)){
00053         std::istringstream line(eil);
00054         stud temp(line);
00055         temp.calculateGalutinis();
00056         visi.push_back(std::move(temp));
00057
00058     }
00059     df.close();
00060     visaTrukme+=t.elapsed();
00061     cout « "Duomenis nuskaityti uztruko " « visaTrukme « endl;
00062     t.reset();
00063     if (vid==1){
00064         auto it = std::partition(visi.begin(), visi.end(), [](const stud &s)
00065         {
00066             return s.getGalutinisVid() >= 5.0; //
00067         });
00068         nuskriausti.insert(nuskriausti.end(), it, visi.end());
00069         visi.erase(it, visi.end());
00070         visi.shrink_to_fit();
00071     }
00072
00073     else if(vid==0){
00074         auto it = std::partition(visi.begin(), visi.end(), [](const stud &s)
00075         {
00076             return s.getGalutinisMed() >= 5.0; //
00077         });
00078         nuskriausti.insert(nuskriausti.end(), it, visi.end());
00079         visi.erase(it, visi.end());
00080         visi.shrink_to_fit();
00081     }
00082
```

```
00083     visaTrukme+=t.elapsed();
00084     cout « "Mokinius isrusiuoti i atskirus konteinerius uztruko " « t.elapsed() « endl;
00085     double trukme=0;
00086
00087
00088     cout « "Pagal ka isrikiuoti nuskriaustu duomenis?" « endl;
00089     compare(nuskriausti,trukme);
00090     cout « "Pagal ka isrikiuoti kietiaku duomenis?" « endl;
00091     compare(visi,trukme);
00092     visaTrukme+=trukme;
00093     cout « "Duomenis isrikiuoti uztruko " « trukme « endl;
00094
00095     t.reset();
00096     //spausdinaFaila(nuskriausti,"nuskriausti "+failas);
00097     //spausdinaFaila(visi,"kietiakai "+failas);
00098     //visaTrukme+=t.elapsed();
00099     //cout « "Duomenis atspausdinti uztruko "« t.elapsed()« endl;
00100     cout « "Isviso uztruko: "« visaTrukme« endl;
00101 }
00102
```

6.20 spausdina.cpp File Reference

```
#include "bibl.h"
```

Functions

- void spausdina (Vektorius< stud > A)
- void spausdinaFaila (Vektorius< stud > &A, string failas)

6.20.1 Function Documentation

6.20.1.1 spausdina()

```
void spausdina (
            Vektorius< stud > A)
```

Definition at line 3 of file spausdina.cpp.

6.20.1.2 spausdinaFaila()

```
void spausdinaFaila (
            Vektorius< stud > & A,
            string failas)
```

Definition at line 12 of file spausdina.cpp.

6.21 spausdina.cpp

Go to the documentation of this file.

```
00001 #include "bibl.h"
00002
00003 void spausdina(Vektorius <stud> A){
00004     cout « "Vardas              Pavarde        Galutinis(vid.) / Galutinis(med.)" « endl;
00005     cout « "------------------------------------------------------------------------" « endl;
00006 for (int i=0;i<A.size();i++){
00007     cout « A[i];
00008 }
00009
00010 }
00011
00012 void spausdinaFaila(Vektorius <stud> &A, string failas){
00013     std::ofstream rf (failas);
00014     rf « "Vardas              Pavarde        Galutinis(vid.) / Galutinis(med.)" « endl;
00015     rf « "------------------------------------------------------------------------" « endl;
00016 for (int i=0;i<A.size();i++){
00017     rf « A[i];
00018 }
00019
00020 }
```

6.22 std.h File Reference

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <ctime>
#include <random>
#include <stdlib.h>
#include <fstream>
#include <chrono>
#include <sstream>
#include <filesystem>
#include <string>
#include <list>
#include <deque>
```

6.23 std.h

Go to the documentation of this file.

```
00001
00002 # ifndef STD_H
00003 # define STD_H
00004
00005 #include <iostream>
00006 #include <iomanip>
00007 #include <vector>
00008 #include <algorithm>
00009 #include <ctime>
00010 #include <random>
00011 #include <stdlib.h>
00012 #include <fstream>
00013 #include <chrono>
00014 #include <sstream>
00015 #include <filesystem>
00016 #include <string>
00017 #include <list>
00018 #include <deque>
00019
00020
```

```
00021 using std::cin;
00022 using std::cout;
00023 using std::endl;
00024 using std::string;
00025 using std::vector;
00026 using std::setw;
00027 using std::to_string;
00028 using std::setw;
00029
00030 # endif
```

6.24 stud.h File Reference

```
#include "std.h"
#include <numeric>
#include "Zmogus.h"
#include "Vektorius.h"
```

Classes

- class stud

6.25 stud.h

Go to the documentation of this file.

```
00001 #ifndef STUD_H
00002 #define STUD_H
00003
00004 #include "std.h"
00005 #include <numeric>
00006 #include "Zmogus.h"
00007 #include "Vektorius.h"
00008
00009 class stud : public Zmogus {
00010 private:
00011     Vektorius<int> tarp;
00012     double tarpvid = 0;
00013     double tarpmed = 0;
00014     double egz = 0;
00015     double galutinisvid = 0;
00016     double galutinismed = 0;
00017
00018 public:
00019     // Constructors
00020     stud() : Zmogus() {}
00021
00022     stud(const string& vardas, const string& pavarde) : Zmogus(vardas, pavarde) {}
00023
00024     stud(std::istringstream& line) {
00025         int paz;
00026         line » vard » pava;
00027         while (line » paz) {
00028             addTarpPazymys(paz);
00029         }
00030         if (!tarp.empty()) {
00031             egz = tarp.back();
00032             tarp.pop_back();
00033         }
00034     }
00035
00036     // Copy constructor
00037     stud(const stud& kitas) : Zmogus(kitas) {
00038         tarp = kitas.tarp;
00039         egz = kitas.egz;
00040         galutinisvid = kitas.galutinisvid;
00041         galutinismed = kitas.galutinismed;
00042     }
00043
```

```
00044     // Move constructor
00045     stud(stud&& kitas) noexcept : Zmogus(std::move(kitas)) {
00046         tarp = std::move(kitas.tarp);
00047         egz = kitas.egz;
00048         galutinisvid = kitas.galutinisvid;
00049         galutinismed = kitas.galutinismed;
00050 
00051         kitas.egz = 0;
00052         kitas.galutinisvid = 0.0;
00053         kitas.galutinismed = 0.0;
00054     }
00055 
00056     // Destructor
00057     ~stud() {
00058         tarp.clear();
00059         tarp.shrink_to_fit();
00060     }
00061 
00062     // Copy assignment operator
00063     stud& operator=(const stud& kitas) {
00064         if (this != &kitas) {
00065             Zmogus::operator=(kitas); // Call base class assignment operator
00066             tarp = kitas.tarp;
00067             egz = kitas.egz;
00068             galutinisvid = kitas.galutinisvid;
00069             galutinismed = kitas.galutinismed;
00070         }
00071         return *this;
00072     }
00073 
00074     // Move assignment operator
00075     stud& operator=(stud&& kitas) noexcept {
00076         if (this != &kitas) {
00077             Zmogus::operator=(std::move(kitas)); // Call base class move assignment operator
00078             tarp = std::move(kitas.tarp);
00079             egz = kitas.egz;
00080             galutinisvid = kitas.galutinisvid;
00081             galutinismed = kitas.galutinismed;
00082 
00083             kitas.egz = 0;
00084             kitas.galutinisvid = 0.0;
00085             kitas.galutinismed = 0.0;
00086         }
00087         return *this;
00088     }
00089 
00090     // Setter methods
00091     void addTarpPazymys(int paz) {
00092         if (tarp.capacity() == 0) tarp.reserve(10);
00093         tarp.push_back(paz);
00094     }
00095     inline void setEgzaminas(double egzaminas) { egz = egzaminas; }
00096     inline void setVid(double vid) { galutinisvid = vid; }
00097     inline void setMed(double med) { galutinismed = med; }
00098 
00099     // Getter methods
00100     Vektorius<int>& getTarp() { return tarp; }
00101     double getEgz() const { return egz; }
00102     double getGalutinisVid() const { return galutinisvid; }
00103     double getGalutinisMed() const { return galutinismed; }
00104 
00105     // Calculate final grades
00106     void calculateGalutinis() {
00107         if (!tarp.empty()) {
00108             tarpvid = std::accumulate(tarp.begin(), tarp.end(), 0) / double(tarp.size());
00109             std::sort(tarp.begin(), tarp.end());
00110             if (tarp.size() % 2 == 0) {
00111                 tarpmed = (tarp[tarp.size() / 2 - 1] + tarp[tarp.size() / 2]) / 2.0;
00112             } else {
00113                 tarpmed = tarp[tarp.size() / 2];
00114             }
00115         }
00116         galutinisvid = (tarpvid * 0.4) + (egz * 0.6);
00117         galutinismed = (tarpmed * 0.4) + (egz * 0.6);
00118     }
00119 
00120     // Overloaded operators
00121     friend std::ostream& operator<<(std::ostream& out, const stud& a) {
00122         out << std::left << setw(20) << a.getVardas() << setw(15) << a.getPavarde()
00123             << setw(18) << std::fixed << std::setprecision(2) << a.getGalutinisVid() << " " <<
      a.getGalutinisMed() << endl;
00124         return out;
00125     }
00126 
00127     friend std::istream& operator>>(std::istream& in, stud& a) {
00128         in >> a.vard >> a.pava;
00129         cout << "Veskite studento namu darbo pazymius arba N, kad sustoti ";
```

```
00130         string input;
00131         while (true) {
00132             try {
00133                 in » input;
00134                 if (input == "N" || input == "n") break;
00135                 int paz = std::stoi(input);
00136                 if (paz < 0 || paz > 10) {
00137                     throw "Ivestas neteisingas simbolis";
00138                 }
00139                 a.addTarpPazymys(paz);
00140             } catch (const std::invalid_argument&) {
00141                 cout « "Ivestas neteisingas simbolis" « endl;
00142                 continue;
00143             } catch (const char* x) {
00144                 cout « x « endl;
00145                 continue;
00146             }
00147         }
00148 
00149         cout « "Iveskite studento egzamino rezultata ";
00150         while (true) {
00151             try {
00152                 in » input;
00153                 int egz = std::stoi(input);
00154                 if (egz < 0 || egz > 10) {
00155                     throw "Ivestas neteisingas simbolis";
00156                 }
00157                 a.setEgzaminas(egz);
00158                 break;
00159             } catch (const std::invalid_argument&) {
00160                 cout « "Ivestas neteisingas simbolis" « endl;
00161                 continue;
00162             } catch (const char* x) {
00163                 cout « x « endl;
00164                 continue;
00165             }
00166         }
00167         return in;
00168     }
00169 };
00170 
00171 #endif
```

6.26 test.cpp File Reference

```
#include <cassert>
#include "bibl.h"
#include "Vektorius.h"
#include <iostream>
#include "stud.h"
#include <stdexcept>
```

Functions

- void testDefaultConstructor ()
- void testSettersAndGetters ()
- void testCopyConstructor ()
- void testMoveConstructor ()
- void testCopyAssignment ()
- void testMoveAssignment ()
- void testOutput ()
- void testInput ()
- void test ()

6.26.1 Function Documentation

6.26.1.1 test()

```
void test ()
```

Definition at line 99 of file test.cpp.

6.26.1.2 testCopyAssignment()

```
void testCopyAssignment ()
```

Definition at line 57 of file test.cpp.

6.26.1.3 testCopyConstructor()

```
void testCopyConstructor ()
```

Definition at line 33 of file test.cpp.

6.26.1.4 testDefaultConstructor()

```
void testDefaultConstructor ()
```

Definition at line 8 of file test.cpp.

6.26.1.5 testInput()

```
void testInput ()
```

Definition at line 88 of file test.cpp.

6.26.1.6 testMoveAssignment()

```
void testMoveAssignment ()
```

Definition at line 67 of file test.cpp.

6.26.1.7 testMoveConstructor()

```
void testMoveConstructor ()
```

Definition at line 46 of file test.cpp.

6.26.1.8 testOutput()

```
void testOutput ()
```

Definition at line 77 of file test.cpp.

6.26.1.9 testSettersAndGetters()

```
void testSettersAndGetters ()
```

Definition at line 17 of file test.cpp.

6.27 test.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002 #include "bibl.h"
00003 #include "Vektorius.h"
00004 #include <iostream>
00005 #include "stud.h"
00006 #include <stdexcept>
00007
00008 void testDefaultConstructor() {
00009     stud s;
00010     assert(s.getVardas() == "");
00011     assert(s.getPavarde() == "");
00012     assert(s.getGalutinisVid() == 0);
00013     assert(s.getGalutinisMed() == 0);
00014     std::cout « "Default konstruktoriaus testas sekmingas!.\n";
00015 }
00016
00017 void testSettersAndGetters() {
00018     stud s;
00019     s.setvard("Jonas");
00020     s.setpava("Jonaitis");
00021     s.setEgzaminas(9.5);
00022     s.setVid(8.0);
00023     s.setMed(8.5);
00024
00025     assert(s.getVardas() == "Jonas");
00026     assert(s.getPavarde() == "Jonaitis");
00027     assert(s.getEgz() == 9.5);
00028     assert(s.getGalutinisVid() == 8);
00029     assert(s.getGalutinisMed() == 8.5);
00030     std::cout « "Setters ir getters testas sekmingas!\n";
00031 }
00032
00033 void testCopyConstructor() {
00034     stud original;
00035     original.setvard("Petras");
00036     original.setpava("Petraitis");
00037     original.getTarp().push_back(10);
00038     original.setEgzaminas(9);
00039
00040     stud copy(original);
00041     assert(copy.getVardas() == "Petras");
00042     assert(copy.getTarp()[0] == 10);
00043     std::cout « "Copy konstruktoriaus testas sekmingas!\n";
00044 }
00045
00046 void testMoveConstructor() {
00047     stud temp;
00048     temp.setvard("Move");
00049     temp.getTarp().push_back(7);
00050
00051     stud moved(std::move(temp));
00052     assert(moved.getVardas() == "Move");
00053     assert(moved.getTarp()[0] == 7);
00054     std::cout « "Move konstruktoriaus testas sekmingas!\n";
00055 }
00056
00057 void testCopyAssignment() {
```

```
00058     stud s1;
00059     s1.setpava("Kebabas");
00060
00061     stud s2;
00062     s2 = s1;
00063     assert(s2.getPavarde() == "Kebabas");
00064     std::cout « "Copy assignment testas sekmingas!\n";
00065 }
00066
00067 void testMoveAssignment() {
00068     stud s1;
00069     s1.setpava("Moved");
00070
00071     stud s2;
00072     s2 = std::move(s1);
00073     assert(s2.getPavarde() == "Moved");
00074     std::cout « "Move assignment testas sekmingas!\n";
00075 }
00076
00077 void testOutput() {
00078     stud original;
00079     original.setvard("Petras");
00080     original.setpava("Petraitis");
00081     original.getTarp().push_back(10);
00082     original.setEgzaminas(9);
00083     original.calculateGalutinis();
00084     cout « original;
00085     std::cout « "Output testas sekmingas!\n";
00086 }
00087
00088 void testInput() {
00089     stud original;
00090     cout « "iveskite varda ir pavarde\n";
00091     cin » original;
00092     original.calculateGalutinis();
00093     cout « original;
00094     std::cout « "Input testas sekmingas!\n";
00095 }
00096
00097
00098
00099 void test() {
00100     testDefaultConstructor();
00101     testSettersAndGetters();
00102     testCopyConstructor();
00103     testMoveConstructor();
00104     testCopyAssignment();
00105     testMoveAssignment();
00106     testOutput();
00107     //testInput();          //Užkomentuotas kad butu automatizuoti testai
00108     std::cout « "\nAll tests passed!\n";
00109 }
```

6.28 Timer.h File Reference

```
#include "std.h"
```

Classes

- class Timer

6.29 Timer.h

Go to the documentation of this file.

```
00001 # ifndef TIMER_H
00002 # define TIMER_H
00003
00004 #include "std.h"
00005
```

```
00006 class Timer {
00007     private:
00008       // panaudojame using
00009       using hrClock = std::chrono::high_resolution_clock;
00010       using durationDouble = std::chrono::duration<double>;
00011       std::chrono::time_point<hrClock> start;
00012     public:
00013       Timer() : start{ hrClock::now() } {}
00014       void reset() {
00015         start = hrClock::now();
00016       }
00017       double elapsed() const {
00018         return durationDouble (hrClock::now() - start).count();
00019       }
00020   };
00021
00022   # endif
```

6.30 unit_test.cpp File Reference

```
#include <gtest/gtest.h>
#include "Vektorius.h"
```

Functions

- int run_unit_tests ()
- TEST (VektoriusTest, DefaultConstructor)
- TEST (VektoriusTest, SizeConstructor)
- TEST (VektoriusTest, SizeValueConstructor)
- TEST (VektoriusTest, InitializerListConstructor)
- TEST (VektoriusTest, CopyConstructor)
- TEST (VektoriusTest, CopyAssignment)
- TEST (VektoriusTest, MoveConstructor)
- TEST (VektoriusTest, MoveAssignment)
- TEST (VektoriusTest, PushPopFrontBack)
- TEST (VektoriusTest, AtAndBracket)
- TEST (VektoriusTest, AssignCountValue)
- TEST (VektoriusTest, AssignInitializerList)
- TEST (VektoriusTest, ClearEmptyShrinkReserveResize)
- TEST (VektoriusTest, Insert)
- TEST (VektoriusTest, Erase)
- TEST (VektoriusTest, Emplace)
- TEST (VektoriusTest, Storage)
- TEST (VektoriusTest, Iterators)
- TEST (VektoriusTest, ComparisonOperators)
- TEST (VektoriusTest, FrontBackException)

6.30.1 Function Documentation

6.30.1.1 run_unit_tests()

```
int run_unit_tests ()
```

Definition at line 4 of file unit_test.cpp.

6.30.1.2 TEST() [1/20]

```
TEST (
            VektoriusTest ,
            AssignCountValue )
```

Definition at line 105 of file unit_test.cpp.

6.30.1.3 TEST() [2/20]

```
TEST (
            VektoriusTest ,
            AssignInitializerList )
```

Definition at line 114 of file unit_test.cpp.

6.30.1.4 TEST() [3/20]

```
TEST (
            VektoriusTest ,
            AtAndBracket )
```

Definition at line 97 of file unit_test.cpp.

6.30.1.5 TEST() [4/20]

```
TEST (
            VektoriusTest ,
            ClearEmptyShrinkReserveResize )
```

Definition at line 124 of file unit_test.cpp.

6.30.1.6 TEST() [5/20]

```
TEST (
            VektoriusTest ,
            ComparisonOperators )
```

Definition at line 197 of file unit_test.cpp.

6.30.1.7 TEST() [6/20]

```
TEST (
            VektoriusTest ,
            CopyAssignment )
```

Definition at line 54 of file unit_test.cpp.

6.30.1.8 TEST() [7/20]

```
TEST (
            VektoriusTest ,
            CopyConstructor )
```

Definition at line 45 of file unit_test.cpp.

6.30.1.9 TEST() [8/20]

```
TEST (
            VektoriusTest ,
            DefaultConstructor )
```

Definition at line 12 of file unit_test.cpp.

6.30.1.10 TEST() [9/20]

```
TEST (
            VektoriusTest ,
            Emplace )
```

Definition at line 165 of file unit_test.cpp.

6.30.1.11 TEST() [10/20]

```
TEST (
            VektoriusTest ,
            Erase )
```

Definition at line 156 of file unit_test.cpp.

6.30.1.12 TEST() [11/20]

```
TEST (
            VektoriusTest ,
            FrontBackException )
```

Definition at line 216 of file unit_test.cpp.

6.30.1.13 TEST() [12/20]

```
TEST (
            VektoriusTest ,
            InitializerListConstructor )
```

Definition at line 35 of file unit_test.cpp.

6.30.1.14 TEST() [13/20]

```
TEST (
            VektoriusTest ,
            Insert )
```

Definition at line 138 of file unit_test.cpp.

6.30.1.15 TEST() [14/20]

```
TEST (
            VektoriusTest ,
            Iterators )
```

Definition at line 183 of file unit_test.cpp.

6.30.1.16 TEST() [15/20]

```
TEST (
            VektoriusTest ,
            MoveAssignment )
```

Definition at line 73 of file unit_test.cpp.

6.30.1.17 TEST() [16/20]

```
TEST (
            VektoriusTest ,
            MoveConstructor )
```

Definition at line 64 of file unit_test.cpp.

6.30.1.18 TEST() [17/20]

```
TEST (
            VektoriusTest ,
            PushPopFrontBack )
```

Definition at line 83 of file unit_test.cpp.

6.30.1.19 TEST() [18/20]

```
TEST (
            VektoriusTest ,
            SizeConstructor )
```

Definition at line 20 of file unit_test.cpp.

6.30.1.20 TEST() [19/20]

```
TEST (
            VektoriusTest ,
            SizeValueConstructor )
```

Definition at line 28 of file unit_test.cpp.

6.30.1.21 TEST() [20/20]

```
TEST (
            VektoriusTest ,
            Storage )
```

Definition at line 174 of file unit_test.cpp.

6.31 unit_test.cpp

Go to the documentation of this file.

```
00001 #include <gtest/gtest.h>
00002 #include "Vektorius.h"
00003
00004 int run_unit_tests() {
00005     int argc = 1;
00006     char* argv[] = { (char*)"program" };
00007     ::testing::InitGoogleTest(&argc, argv);
00008     return RUN_ALL_TESTS();
00009 }
00010
00011 // Test default constructor
00012 TEST(VektoriusTest, DefaultConstructor) {
00013     Vektorius<int> v;
00014     EXPECT_EQ(v.size(), 0);
00015     EXPECT_EQ(v.capacity(), 0);
00016     EXPECT_TRUE(v.empty());
00017 }
00018
00019 // Test size constructor
00020 TEST(VektoriusTest, SizeConstructor) {
00021     Vektorius<int> v(5);
00022     EXPECT_EQ(v.size(), 5);
00023     for (size_t i = 0; i < 5; ++i)
00024         EXPECT_EQ(v[i], 0);
00025 }
00026
00027 // Test size and value constructor
00028 TEST(VektoriusTest, SizeValueConstructor) {
00029     Vektorius<int> v(3, 7);
00030     EXPECT_EQ(v.size(), 3);
00031     for (size_t i = 0; i < 3; ++i)
00032         EXPECT_EQ(v[i], 7);
00033 }
00034
00035 TEST(VektoriusTest, InitializerListConstructor) {
00036     Vektorius<int> v{10, 20, 30, 40};
00037     EXPECT_EQ(v.size(), 4);
00038     EXPECT_EQ(v[0], 10);
00039     EXPECT_EQ(v[1], 20);
00040     EXPECT_EQ(v[2], 30);
00041     EXPECT_EQ(v[3], 40);
00042 }
00043
00044 // Test copy constructor
00045 TEST(VektoriusTest, CopyConstructor) {
00046     Vektorius<int> v1(2, 9);
00047     Vektorius<int> v2(v1);
00048     EXPECT_EQ(v2.size(), 2);
00049     EXPECT_EQ(v2[0], 9);
00050     EXPECT_EQ(v2[1], 9);
00051 }
```

```
00052
00053 // Test copy assignment
00054 TEST(VektoriusTest, CopyAssignment) {
00055     Vektorius<int> v1(2, 5);
00056     Vektorius<int> v2;
00057     v2 = v1;
00058     EXPECT_EQ(v2.size(), 2);
00059     EXPECT_EQ(v2[0], 5);
00060     EXPECT_EQ(v2[1], 5);
00061 }
00062
00063 // Test move constructor
00064 TEST(VektoriusTest, MoveConstructor) {
00065     Vektorius<int> v1(2, 3);
00066     Vektorius<int> v2(std::move(v1));
00067     EXPECT_EQ(v2.size(), 2);
00068     EXPECT_EQ(v2[0], 3);
00069     EXPECT_EQ(v2[1], 3);
00070 }
00071
00072 // Test move assignment
00073 TEST(VektoriusTest, MoveAssignment) {
00074     Vektorius<int> v1(2, 4);
00075     Vektorius<int> v2;
00076     v2 = std::move(v1);
00077     EXPECT_EQ(v2.size(), 2);
00078     EXPECT_EQ(v2[0], 4);
00079     EXPECT_EQ(v2[1], 4);
00080 }
00081
00082 // Test push_back, pop_back, front, back
00083 TEST(VektoriusTest, PushPopFrontBack) {
00084     Vektorius<int> v;
00085     v.push_back(1);
00086     v.push_back(2);
00087     v.push_back(3);
00088     EXPECT_EQ(v.size(), 3);
00089     EXPECT_EQ(v.front(), 1);
00090     EXPECT_EQ(v.back(), 3);
00091     v.pop_back();
00092     EXPECT_EQ(v.size(), 2);
00093     EXPECT_EQ(v.back(), 2);
00094 }
00095
00096 // Test at() and operator[]
00097 TEST(VektoriusTest, AtAndBracket) {
00098     Vektorius<int> v(2, 10);
00099     EXPECT_EQ(v.at(0), 10);
00100     EXPECT_EQ(v[1], 10);
00101     EXPECT_THROW(v.at(2), std::out_of_range);
00102 }
00103
00104 // Test assign (count, value)
00105 TEST(VektoriusTest, AssignCountValue) {
00106     Vektorius<int> v;
00107     v.assign(4, 8);
00108     EXPECT_EQ(v.size(), 4);
00109     for (size_t i = 0; i < 4; ++i)
00110         EXPECT_EQ(v[i], 8);
00111 }
00112
00113 // Test assign (initializer_list)
00114 TEST(VektoriusTest, AssignInitializerList) {
00115     Vektorius<int> v;
00116     v.assign({1, 2, 3});
00117     EXPECT_EQ(v.size(), 3);
00118     EXPECT_EQ(v[0], 1);
00119     EXPECT_EQ(v[1], 2);
00120     EXPECT_EQ(v[2], 3);
00121 }
00122
00123 // Test clear, empty, shrink_to_fit, reserve, resize
00124 TEST(VektoriusTest, ClearEmptyShrinkReserveResize) {
00125     Vektorius<int> v(5, 2);
00126     v.clear();
00127     EXPECT_EQ(v.size(), 0);
00128     EXPECT_TRUE(v.empty());
00129     v.reserve(10);
00130     EXPECT_GE(v.capacity(), 10);
00131     v.resize(3);
00132     EXPECT_EQ(v.size(), 3);
00133     v.shrink_to_fit();
00134     EXPECT_EQ(v.capacity(), v.size());
00135 }
00136
00137 // Test insert (single, multiple, range)
00138 TEST(VektoriusTest, Insert) {
```

```
00139     Vektorius<int> v;
00140     v.push_back(1);
00141     v.push_back(3);
00142     v.insert(v.begin() + 1, 2);
00143     EXPECT_EQ(v[1], 2);
00144
00145     v.insert(v.end(), 2, 4);
00146     EXPECT_EQ(v[3], 4);
00147     EXPECT_EQ(v[4], 4);
00148
00149     int arr[] = {5, 6};
00150     v.insert(v.end(), arr, arr + 2);
00151     EXPECT_EQ(v[5], 5);
00152     EXPECT_EQ(v[6], 6);
00153 }
00154
00155 // Test erase (single, range)
00156 TEST(VektoriusTest, Erase) {
00157     Vektorius<int> v = {1, 2, 3, 4, 5};
00158     v.erase(v.begin() + 1);
00159     EXPECT_EQ(v[1], 3);
00160     v.erase(v.begin(), v.begin() + 2);
00161     EXPECT_EQ(v[0], 4);
00162 }
00163
00164 // Test emplace
00165 TEST(VektoriusTest, Emplace) {
00166     Vektorius<std::string> v;
00167     v.emplace(v.begin(), "hello");
00168     v.emplace(v.end(), "world");
00169     EXPECT_EQ(v[0], "hello");
00170     EXPECT_EQ(v[1], "world");
00171 }
00172
00173 // Test storage
00174 TEST(VektoriusTest, Storage) {
00175     Vektorius<int> v = {1, 2, 3};
00176     int* ptr = v.storage();
00177     EXPECT_EQ(ptr[0], 1);
00178     EXPECT_EQ(ptr[1], 2);
00179     EXPECT_EQ(ptr[2], 3);
00180 }
00181
00182 // Test iterators
00183 TEST(VektoriusTest, Iterators) {
00184     Vektorius<int> v = {1, 2, 3};
00185     int sum = 0;
00186     for (auto it = v.begin(); it != v.end(); ++it)
00187         sum += *it;
00188     EXPECT_EQ(sum, 6);
00189
00190     sum = 0;
00191     for (auto it = v.rbegin(); it != v.rend(); ++it)
00192         sum += *it;
00193     EXPECT_EQ(sum, 6);
00194 }
00195
00196 // Test comparison operators
00197 TEST(VektoriusTest, ComparisonOperators) {
00198     Vektorius<int> v1 = {1, 2, 3};
00199     Vektorius<int> v2 = {1, 2, 3};
00200     Vektorius<int> v3 = {1, 2, 4};
00201     Vektorius<int> v4 = {1, 2};
00202
00203     EXPECT_TRUE(v1 == v2);
00204     EXPECT_FALSE(v1 != v2);
00205     EXPECT_TRUE(v1 < v3);
00206     EXPECT_TRUE(v3 > v1);
00207     EXPECT_TRUE(v4 < v1);
00208     EXPECT_TRUE(v1 > v4);
00209     EXPECT_TRUE(v1 <= v2);
00210     EXPECT_TRUE(v1 >= v2);
00211     EXPECT_TRUE(v1 <= v3);
00212     EXPECT_TRUE(v3 >= v1);
00213 }
00214
00215 // Test front/back exception
00216 TEST(VektoriusTest, FrontBackException) {
00217     Vektorius<int> v;
00218     EXPECT_THROW(v.front(), std::out_of_range);
00219     EXPECT_THROW(v.back(), std::out_of_range);
00220 }
```

6.32 vectortest.cpp File Reference

```
#include <iostream>
#include "bibl.h"
#include <vector>
```

Functions

- int vectorTest ()

6.32.1 Function Documentation

6.32.1.1 vectorTest()

```
int vectorTest ()
```

Definition at line 7 of file vectortest.cpp.

6.33 vectortest.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "bibl.h"
00003 #include <vector>
00004 using namespace std;
00005
00006
00007 int vectorTest(){
00008
00009 unsigned int sz = 10000; // 100000, 1000000, 10000000, 100000000
00010 cout « sz « endl;
00011 Timer t;
00012 int count = 0;
00013 std::vector<unsigned int> v1;
00014
00015 for (unsigned int i = 1; i <= sz; ++i) {
00016     if(v1.capacity() == v1.size()) {
00017         count++;
00018     }
00019     v1.push_back(i);
00020 }
00021 cout « "std::vector uzpildymo laikas: " « t.elapsed() « " sek." « endl;
00022 cout « "std::vector resize kartu: " « count « endl;
00023 t.reset();
00024 count = 0;
00025
00026 Vektorius <unsigned int> v2;
00027
00028 for (unsigned int  i = 1; i <= sz; ++i){
00029     if(v2.capacity() == v2.size()) {
00030         count++;
00031     }
00032     v2.push_back(i);
00033 }
00034
00035 cout « "Vektorius uzpildymo laikas: " « t.elapsed() « " sek." « endl;
00036 cout « "Vektoriaus resize kartu: " « count « endl;
00037  return 0;
00038 }
```

6.34 Vektorius.h File Reference

```
#include "std.h"
#include <iostream>
#include <iterator>
```

Classes

- class Vektorius< T >

6.35 Vektorius.h

Go to the documentation of this file.

```
00001 #include "std.h"
00002
00003 #ifndef VEKTORIUS_H
00004 #define VEKTORIUS_H
00005
00006 #include <iostream>
00007 #include <iterator>
00008
00009 template <typename T>
00010 class Vektorius {
00011 private:
00012
00013     T* data;
00014     size_t sz;
00015     size_t cap;
00016
00017     void reallocate(size_t new_cap) {
00018         T* new_data = new T[new_cap];
00019         for (size_t i = 0; i < sz; ++i) {
00020             new_data[i] = data[i];
00021         }
00022         delete[] data;
00023         data = new_data;
00024         cap = new_cap;
00025     }
00026
00027 public:
00028
00029     using iterator        = T*;
00030     using const_iterator  = const T*;
00031     using size_type       = std::size_t;
00032     using difference_type = std::ptrdiff_t;
00033     using reference       = T&;
00034     using const_reference = const T&;
00035     using reverse_iterator = std::reverse_iterator<iterator>;
00036     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00037
00038     // Constructor
00042     Vektorius() : data(nullptr), sz(0), cap(0) {}
00043
00048     Vektorius(size_t size) : sz(size), cap(size) {
00049         data = new T[cap];
00050         for (size_t i = 0; i < sz; ++i) {
00051             data[i] = T();
00052         }
00053     }
00054
00060     Vektorius(size_t size, const T& value) : sz(size), cap(size) {
00061         data = new T[cap];
00062         for (size_t i = 0; i < sz; ++i) {
00063             data[i] = value;
00064         }
00065     }
00066
00071     Vektorius(std::initializer_list<T> ilist) : sz(ilist.size()), cap(ilist.size()) {
00072         data = new T[cap];
00073         size_t i = 0;
00074         for (const auto& val : ilist) {
00075             data[i++] = val;
```

```
00076         }
00077     }
00078
00079
00083     ~Vektorius() {
00084         delete[] data;
00085     }
00086
00090     Vektorius(const Vektorius& other) {
00091         sz = other.sz;
00092         cap = other.cap;
00093         data = new T[cap];
00094         for (size_t i = 0; i < sz; ++i) {
00095             data[i] = other.data[i];
00096         }
00097     }
00098
00102     Vektorius& operator=(const Vektorius& other) {
00103         if (this != &other) {
00104             delete[] data;
00105             sz = other.sz;
00106             cap = other.cap;
00107             data = new T[cap];
00108             for (size_t i = 0; i < sz; ++i) {
00109                 data[i] = other.data[i];
00110             }
00111         }
00112         return *this;
00113     }
00114
00118     Vektorius(Vektorius&& other) noexcept {
00119         data = other.data;
00120         sz = other.sz;
00121         cap = other.cap;
00122         other.data = nullptr;
00123         other.sz = other.cap = 0;
00124     }
00125
00129     Vektorius& operator=(Vektorius&& other) noexcept {
00130         if (this != &other) {
00131             delete[] data;
00132             data = other.data;
00133             sz = other.sz;
00134             cap = other.cap;
00135             other.data = nullptr;
00136             other.sz = other.cap = 0;
00137         }
00138         return *this;
00139     }
00140
00145     void swap(Vektorius& other) {
00146         std::swap(data, other.data);
00147         std::swap(sz, other.sz);
00148         std::swap(cap, other.cap);
00149     }
00150
00156     void assign(size_t count, const T& value ){
00157
00158         if (count > cap) {
00159             reallocate(count);
00160         }
00161         for (int i = 0; i < count; ++i) {
00162             data[i] = value;
00163         }
00164         sz = count;
00165     }
00166
00172     template< typename InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt» >
00173     void assign( InputIt first, InputIt last ){
00174         size_t count = last - first;
00175         if (count < 0) {
00176             throw std::out_of_range("Iterator out of range");
00177         }
00178         if (count > cap) {
00179             reallocate(count);
00180         }
00181         for (size_t i = 0; i < count; ++i) {
00182             data[i] = *first++;
00183         }
00184         sz = count;
00185     }
00186
00191     void assign( std::initializer_list<T> ilist ) {
00192         size_t count = ilist.size();
00193         if (count > cap) {
00194             reallocate(count);
00195         }
```

```
00196         size_t i = 0;
00197         for (const auto& item : ilist) {
00198             data[i++] = item;
00199         }
00200         sz = count;
00201     }
00202
00206     void shrink_to_fit() {
00207         if (sz < cap) {
00208             T* new_data = new T[sz];
00209             for (size_t i = 0; i < sz; ++i) {
00210                 new_data[i] = data[i];
00211             }
00212             delete[] data;
00213             data = new_data;
00214             cap = sz;
00215         }
00216     }
00217
00223     iterator erase( const_iterator pos ){
00224         if (pos < data || pos >= data + sz) {
00225             throw std::out_of_range("Iterator out of range");
00226         }
00227         size_t index = pos - data;
00228         for (size_t i = index; i < sz - 1; ++i) {
00229             data[i] = data[i + 1];
00230         }
00231         --sz;
00232         return data + index;
00233     }
00234
00241     iterator erase( const_iterator first, const_iterator last ){
00242         if (first < data || last > data + sz) {
00243             throw std::out_of_range("Iterator out of range");
00244         }
00245         size_t start = first - data;
00246         size_t end = last - data;
00247         size_t count = end - start;
00248         for (size_t i = start; i < sz - count; ++i) {
00249             data[i] = data[i + count];
00250         }
00251         sz -= count;
00252         return data + start;
00253     }
00254
00261     iterator insert( const_iterator pos, const T& value ){
00262         if (pos < data || pos > data + sz) {
00263             throw std::out_of_range("Iterator out of range");
00264         }
00265         size_t index = pos - data;
00266         if (sz >= cap) {
00267             reallocate(cap == 0 ? 1 : cap * 2);
00268         }
00269         for (size_t i = sz; i > index; --i) {
00270             data[i] = data[i - 1];
00271         }
00272         data[index] = value;
00273         ++sz;
00274         return data + index;
00275     }
00276
00283     iterator insert( const_iterator pos, T&& value ){
00284         if (pos < data || pos > data + sz) {
00285             throw std::out_of_range("Iterator out of range");
00286         }
00287         size_t index = pos - data;
00288         if (sz >= cap) {
00289             reallocate(cap == 0 ? 1 : cap * 2);
00290         }
00291         for (size_t i = sz; i > index; --i) {
00292             data[i] = std::move(data[i - 1]);
00293         }
00294         data[index] = std::move(value);
00295         ++sz;
00296         return data + index;
00297     }
00298
00306     iterator insert( const_iterator pos, size_type count, const T& value ){
00307         if (pos < data || pos > data + sz) {
00308             throw std::out_of_range("Iterator out of range");
00309         }
00310         size_t index = pos - data;
00311         if (sz + count > cap) {
00312             reallocate(cap == 0 ? count : cap + count);
00313         }
00314         for (size_t i = sz + count - 1; i >= index + count; --i) {
00315             data[i] = data[i - count];
```

```
00316         }
00317         for (size_t i = index; i < index + count; ++i) {
00318             data[i] = value;
00319         }
00320         sz += count;
00321         return data + index;
00322     }
00323 
00331     template< class InputIt, typename = std::enable_if_t<!std::is_integral_v<InputIt» >
00332     iterator insert( const_iterator pos, InputIt first, InputIt last ){
00333         if (pos < data || pos > data + sz) {
00334             throw std::out_of_range("Iterator out of range");
00335         }
00336         size_t index = pos - data;
00337         size_t count = last - first;
00338         if (sz + count > cap) {
00339             reallocate(cap == 0 ? count : cap + count);
00340         }
00341         for (size_t i = sz + count - 1; i >= index + count; --i) {
00342             data[i] = data[i - count];
00343         }
00344         for (size_t i = index; i < index + count; ++i) {
00345             data[i] = *first++;
00346         }
00347         sz += count;
00348         return data + index;
00349     }
00350 
00357     template< class... Args >
00358     iterator emplace( const_iterator pos, Args&&... args ){
00359         if (pos < data || pos > data + sz) {
00360             throw std::out_of_range("Iterator out of range");
00361         }
00362         size_t index = pos - data;
00363         if (sz >= cap) {
00364             reallocate(cap == 0 ? 1 : cap * 2);
00365         }
00366         for (size_t i = sz; i > index; --i) {
00367             data[i] = std::move(data[i - 1]);
00368         }
00369         new (&data[index]) T(std::forward<Args>(args)...);
00370         ++sz;
00371         return data + index;
00372     }
00373 
00374 
00379     void push_back(const T& value) {
00380         if (sz >= cap) {
00381             reallocate(cap == 0 ? 1 : cap * 2);
00382         }
00383         data[sz++] = value;
00384     }
00385 
00390     void pop_back() {
00391         if (sz > 0) --sz;
00392     }
00393 
00397     size_t size() const { return sz; }
00398 
00402     size_t capacity() const { return cap; }
00403 
00407     bool empty() const { return sz == 0; }
00408 
00412     void clear() {
00413         for (size_t i = 0; i < sz; ++i) {
00414             data[i].~T();
00415         }
00416         sz = 0;
00417     }
00418 
00423     void reserve(size_t new_cap) {
00424         if (new_cap > cap)
00425             reallocate(new_cap);
00426     }
00427 
00432     void resize( size_t count ){
00433         if (count > cap) {
00434             reallocate(count);
00435         }
00436         for (size_t i = sz; i < count; ++i) {
00437             data[i] = T();
00438         }
00439         sz = count;
00440     }
00441 
00442 
00449     reference at( size_type pos ){
```

```
00450         if (pos >= sz) {
00451             throw std::out_of_range("Index out of range");
00452         }
00453         return data[pos];
00454     }
00455
00462     const_reference at( size_type pos ) const {
00463         if (pos >= sz) {
00464             throw std::out_of_range("Index out of range");
00465         }
00466         return data[pos];
00467     }
00468
00469
00475     T& operator[](size_t index) {
00476         return data[index];
00477     }
00478
00484     const T& operator[](size_t index) const {
00485         return data[index];
00486     }
00487
00493     bool operator==( const Vektorius<T>& other ) const {
00494         if (sz != other.sz) return false;
00495         for (size_t i = 0; i < sz; ++i) {
00496             if (data[i] != other.data[i]) return false;
00497         }
00498         return true;
00499     }
00500
00506     bool operator!=( const Vektorius<T>& other ) const {
00507         return !(*this == other);
00508     }
00509
00515     bool operator<( const Vektorius<T>& other ) const {
00516         for (size_t i = 0; i < std::min(sz, other.sz); ++i) {
00517             if (data[i] < other.data[i]) return true;
00518             if (data[i] > other.data[i]) return false;
00519         }
00520         return sz < other.sz;
00521     }
00522
00528     bool operator<=( const Vektorius<T>& other ) const {
00529         for (size_t i = 0; i < std::min(sz, other.sz); ++i) {
00530             if (data[i] < other.data[i]) return true;
00531             if (data[i] > other.data[i]) return false;
00532         }
00533         return sz <= other.sz;
00534     }
00535
00541     bool operator>( const Vektorius<T>& other ) const {
00542         for (size_t i = 0; i < std::min(sz, other.sz); ++i) {
00543             if (data[i] < other.data[i]) return false;
00544             if (data[i] > other.data[i]) return true;
00545         }
00546         return sz > other.sz;
00547     }
00548
00554     bool operator>=( const Vektorius<T>& other ) const {
00555         for (size_t i = 0; i < std::min(sz, other.sz); ++i) {
00556             if (data[i] < other.data[i]) return false;
00557             if (data[i] > other.data[i]) return true;
00558         }
00559         return sz >= other.sz;
00560     }
00561
00566     iterator begin() { return data; }
00567     const_iterator begin() const { return data; }
00568     const_iterator cbegin() const { return data; }
00569
00574     reverse_iterator rbegin(){ return reverse_iterator(end()); }
00575     const_reverse_iterator rbegin() const { return const_reverse_iterator(end()); }
00576     const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }
00577
00582     iterator end() { return data + sz; }
00583     const_iterator end() const { return data + sz; }
00584     const_iterator cend() const { return data + sz; }
00585
00590     reverse_iterator rend() { return reverse_iterator(begin()); }
00591     const_reverse_iterator rend() const { return const_reverse_iterator(begin()); }
00592     const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }
00593
00598     reference front() {
00599         if (sz == 0) {
00600             throw std::out_of_range("Vector is empty");
00601         }
00602         return data[0];
```

```
00603     }
00604     const_reference front() const {
00605         if (sz == 0) {
00606             throw std::out_of_range("Vector is empty");
00607         }
00608         return data[0];
00609     }
00610
00615     reference back() {
00616         if (sz == 0) {
00617             throw std::out_of_range("Vector is empty");
00618         }
00619         return data[sz - 1];
00620     }
00621     const_reference back() const {
00622         if (sz == 0) {
00623             throw std::out_of_range("Vector is empty");
00624         }
00625         return data[sz - 1];
00626     }
00627
00632     T* storage() {
00633         return data;
00634     }
00635     const T* storage() const {
00636         return data;
00637     }
00638
00639
00640 };
00641
00642 #endif // VEKTORIUS_H
```

6.36 Zmogus.h File Reference

```
#include "std.h"
#include <numeric>
```

Classes

- class Zmogus

6.37 Zmogus.h

Go to the documentation of this file.

```
00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003
00004 #include "std.h"
00005 #include <numeric>
00006
00007 class Zmogus {
00008 public:
00009     // Default constructor
00010     Zmogus() = default;
00011
00012     Zmogus(const string& vard, const string& pava) : vard(vard), pava(pava) {}
00013
00014     // Virtual destructor
00015     virtual ~Zmogus() = default;
00016
00017     // Copy constructor
00018     Zmogus(const Zmogus& other)
00019         : vard(other.vard), pava(other.pava) {}
00020
00021     // Copy assignment operator
00022     Zmogus& operator=(const Zmogus& other) {
00023         if (this != &other) {
00024             vard = other.vard;
```

```
00025             pava = other.pava;
00026         }
00027         return *this;
00028     }
00029
00030     // Move constructor
00031     Zmogus(Zmogus&& other) noexcept
00032         : vard(std::move(other.vard)), pava(std::move(other.pava)) {}
00033
00034     // Move assignment operator
00035     Zmogus& operator=(Zmogus&& other) noexcept {
00036         if (this != &other) {
00037             vard = std::move(other.vard);
00038             pava = std::move(other.pava);
00039         }
00040         return *this;
00041     }
00042
00043     inline void setvard(const string& vardas) { vard = vardas; }
00044     inline void setpava(const string& pavard) { pava = pavard; }
00045
00046     inline string getVardas() const { return vard; }
00047     inline string getPavarde() const { return pava; }
00048 protected:
00049
00050     string vard;
00051     string pava;
00052 };
00053
00054 #endif // ZMOGUS_H
```

Index