Objektinio Programavimo Labaratorinis

v3.0

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

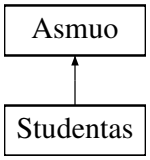
Class Documentation

4.1 Asmuo Class Reference

Bazine klase, kuri atstovauja asmeniui.

`#include <asmuo.h>`

Inheritance diagram for Asmuo:



Public Member Functions

- virtual ~**Asmuo** ()=default
  Virtualus destruktorius.
- virtual std::string getV () const =0
  Gauti varda.
- virtual std::string getP () const =0
  Gauti pavarde.
- virtual const std::vector< double > & getNd_rez () const =0
  Gauti namu darbu rezultatus.
- virtual double getEgz_rez () const =0
  Gauti egzamino rezultata.
- virtual double getNd_sum () const =0
  Gauti namu darbu rezultatu suma.
- virtual double getRez () const =0
  Gauti galutini rezultata.

4.1.1 Detailed Description

Bazine klase, kuri atstovauja asmeniui.

4.1.2 Member Function Documentation

4.1.2.1 getEgz_rez()

```
virtual double Asmuo::getEgz_rez ( ) const  [pure virtual]
```

Gauti egzamino rezultata.

Returns

Asmens egzamino rezultatas.

Implemented in Studentas.

4.1.2.2 getNd_rez()

```
virtual const std::vector< double > & Asmuo::getNd_rez ( ) const  [pure virtual]
```

Gauti namu darbu rezultatus.

Returns

Asmens namu darbu rezultatai.

Implemented in Studentas.

4.1.2.3 getNd_sum()

```
virtual double Asmuo::getNd_sum ( ) const  [pure virtual]
```

Gauti namu darbu rezultatu suma.

Returns

Asmens namu darbu rezultatu suma.

Implemented in Studentas.

4.1.2.4 getP()

```
virtual std::string Asmuo::getP ( ) const  [pure virtual]
```

Gauti pavarde.

Returns

Asmens pavarde.

Implemented in Studentas.

4.1.2.5 getRez()

```
virtual double Asmuo::getRez ( ) const  [pure virtual]
```

Gauti galutini rezultata.

Returns

Asmens galutinis rezultatas.

Implemented in Studentas.

4.1.2.6 getV()

```
virtual std::string Asmuo::getV ( ) const  [pure virtual]
```

Gauti varda.

Returns

Asmens vardas.

Implemented in Studentas.

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

- C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_vektoriai/v3.0_vektoriai/asmuo.h

4.2 ManoVektorius< T, Allocator > Class Template Reference

Klasė ManoVektorius.

```
#include <ManoVektorius.h>
```

Public Types

- using **value_type** = T
- using **allocator_type** = Allocator
- using **size_type** = std::size_t
- using **difference_type** = std::ptrdiff_t
- using **reference** = value_type&
- using **const_reference** = const value_type&
- using **pointer** = typename std::allocator_traits<Allocator>::pointer
- using **const_pointer** = typename std::allocator_traits<Allocator>::const_pointer
- using **iterator** = pointer
- using **const_iterator** = const_pointer
- using **reverse_iterator** = std::reverse_iterator<iterator>
- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>

Public Member Functions

- ManoVektorius ()
- ~ManoVektorius ()
- ManoVektorius (const ManoVektorius &other)
- ManoVektorius & operator= (const ManoVektorius &other)
- ManoVektorius (ManoVektorius &&other) noexcept
- ManoVektorius & operator= (ManoVektorius &&other) noexcept
- reference at (size_type pos)
- const_reference at (size_type pos) const
- reference operator[ ] (size_type pos)
- const_reference operator[ ] (size_type pos) const
- allocator_type get_allocator () const noexcept
- reference front ()
- const_reference front () const
- reference back ()
- const_reference back () const
- T * data () noexcept
- const T * data () const noexcept
- iterator begin () noexcept
- const_iterator begin () const noexcept
- const_iterator cbegin () const noexcept
- iterator end () noexcept
- const_iterator end () const noexcept
- const_iterator cend () const noexcept
- reverse_iterator rbegin () noexcept
- const_reverse_iterator crbegin () const noexcept
- reverse_iterator rend () noexcept
- const_reverse_iterator rend () const noexcept
- const_reverse_iterator crend () const noexcept
- bool empty () const noexcept
- size_type size () const noexcept
- size_type max_size () const noexcept
- void reserve (size_type new_cap)
- size_type getCapacity () const noexcept
- void shrink_to_fit ()
- void clear () noexcept
- void push_back (const T &value)
- void pop_back ()
- void resize (size_type count, T value=T())
- void swap (ManoVektorius &other) noexcept
- iterator insert (const_iterator pos, const T &value)
- template<typename... Args>
  iterator emplace (const_iterator pos, Args &&...args)
- iterator erase (const_iterator pos)
- template<typename InputIt >
  void assign (InputIt first, InputIt last)
- void assign (size_type count, const T &value)
- template<typename... Args>
  void emplace_back (Args &&...args)
- template<typename InputIt >
  void append_range (InputIt first, InputIt last)

Private Member Functions

- void **destroy_elements** ()

  Sunaikina visus elementus masyve.

Private Attributes

- allocator_type **allocator**
- pointer **arr**
- size_type **capacity**
- size_type **current**

4.2.1 Detailed Description

template<typename T, typename Allocator = std::allocator<T>>
class ManoVektorius< T, Allocator >

Klasė ManoVektorius.

Tai yra vektorių klasė, kuri realizuoja dinaminio masyvo funkcionalumą.

Template Parameters

T	Elemento tipas
Allocator	Alokatoriaus tipas

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ManoVektorius() [1/3]

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius< T, Allocator >::ManoVektorius ( )  [inline]
```

Konstruktorius be parametru. Sukuria tuscia ManoVektorius objekta.

4.2.2.2 ~ManoVektorius()

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius< T, Allocator >::~ManoVektorius ( )  [inline]
```

Destruktorius. Sunaikina ManoVektorius objekta ir atlaisvina atminti.

4.2.2.3 ManoVektorius() [2/3]

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius< T, Allocator >::ManoVektorius (
            const ManoVektorius< T, Allocator > & other )  [inline]
```

Kopijavimo konstruktorius. Sukuria nauja ManoVektorius objekta, kuris yra identiskas kitam objektui.

Parameters

other	- objektas, kurio kopija bus sukurta.

4.2.2.4 ManoVektorius() [3/3]

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius< T, Allocator >::ManoVektorius (
            ManoVektorius< T, Allocator > && other )  [inline], [noexcept]
```

Perkelimo konstruktorius. Perkelia duomenis is vieno ManoVektorius objekto i kita.

Parameters

other	- objektas, is kurio duomenys bus perkelti.

4.2.3 Member Function Documentation

4.2.3.1 append_range()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename InputIt >
void ManoVektorius< T, Allocator >::append_range (
            InputIt first,
            InputIt last )  [inline]
```

Prideda elementu intervala i vektoriaus gala.

Parameters

first,last	- intervalo pradzia ir pabaiga.

4.2.3.2 assign() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename InputIt >
void ManoVektorius< T, Allocator >::assign (
            InputIt first,
            InputIt last )  [inline]
```

Priskiria vektoriui naujas reiksmes is intervalo.

Parameters

first,last	- intervalo pradzia ir pabaiga.

4.2.3.3 assign() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::assign (
            size_type count,
            const T & value )  [inline]
```

Priskiria vektoriui naujas reiksmes.

Parameters

count	- kiek reiksmiu bus priskirta.
value	- reiksme, kuri bus priskirta.

4.2.3.4 at() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference ManoVektorius< T, Allocator >::at (
            size_type pos )  [inline]
```

Grazina elemento nuoroda pagal nurodyta pozicija.

Parameters

pos	- elemento pozicija.

Returns

Elemento nuoroda.

Exceptions

std::out_of_range	jei pos yra didesne arba lygi dabartiniam dydziui.

4.2.3.5 at() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference ManoVektorius< T, Allocator >::at (
            size_type pos ) const  [inline]
```

Grazina konstanta elemento nuoroda pagal nurodyta pozicija.

Parameters

pos	- elemento pozicija.

Returns

Konstanta elemento nuoroda.

Exceptions

std::out_of_range	jei pos yra didesne arba lygi dabartiniam dydziui.

4.2.3.6 back() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference ManoVektorius< T, Allocator >::back ( )  [inline]
```

Grazina nuoroda i paskutini elementa.

Returns

Nuoroda i paskutini elementa.

4.2.3.7 back() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference ManoVektorius< T, Allocator >::back ( ) const  [inline]
```

Grazina konstanta nuoroda i paskutini elementa.

Returns

Konstanta nuoroda i paskutini elementa.

4.2.3.8 begin() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator ManoVektorius< T, Allocator >::begin ( ) const  [inline], [noexcept]
```

Grazina konstanta iteratoriu i pradzia.

Returns

Konstanta iteratorius i pradzia.

4.2.3.9 begin() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
iterator ManoVektorius< T, Allocator >::begin ( )  [inline], [noexcept]
```

Grazina iteratoriu i pradzia.

Returns

Iteratorius i pradzia.

4.2.3.10 cbegin()

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator ManoVektorius< T, Allocator >::cbegin ( ) const  [inline], [noexcept]
```

Grazina konstanta iteratoriu i pradzia.

Returns

Konstanta iteratorius i pradzia.

4.2.3.11 cend()

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator ManoVektorius< T, Allocator >::cend ( ) const  [inline], [noexcept]
```

Grazina konstanta iteratoriu i pabaiga.

Returns

Konstanta iteratorius i pabaiga.

4.2.3.12 clear()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::clear ( )  [inline], [noexcept]
```

Isvalo vektoriu.

4.2.3.13 crbegin()

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator ManoVektorius< T, Allocator >::crbegin ( ) const  [inline], [noexcept]
```

Grazina konstanta apversta iteratoriu i pradzia.

Returns

Konstanta apverstas iteratorius i pradzia. Grazina konstanta apversta iteratoriu i pradzia.

Konstanta apverstas iteratorius i pradzia.

4.2.3.14 crend()

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator ManoVektorius< T, Allocator >::crend ( ) const  [inline], [noexcept]
```

Grazina konstanta apversta iteratoriu i pabaiga.

Returns

Konstanta apverstas iteratorius i pabaiga.

4.2.3.15 data() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const T * ManoVektorius< T, Allocator >::data ( ) const  [inline], [noexcept]
```

Grazina konstanta nuoroda i duomenu masyva.

Returns

Konstanta nuoroda i duomenu masyva.

4.2.3.16 data() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
T * ManoVektorius< T, Allocator >::data ( )  [inline], [noexcept]
```

Grazina nuoroda i duomenu masyva.

Returns

Nuoroda i duomenu masyva.

4.2.3.17 emplace()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename...  Args>
iterator ManoVektorius< T, Allocator >::emplace (
            const_iterator pos,
            Args &&...  args )  [inline]
```

Sukuria elementa vietoje nurodytoje vietoje.

Parameters

pos	- vieta, kurioje bus sukurtas elementas.
args	- argumentai, reikalingi elementui sukurti.

Returns

Iteratorius i sukurtą elementą.

4.2.3.18 emplace_back()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename...  Args>
void ManoVektorius< T, Allocator >::emplace_back (
            Args &&...  args )  [inline]
```

Sukuria ir prideda elementa i vektoriaus gala.

Parameters

args	- argumentai, reikalingi elementui sukurti.

4.2.3.19 empty()

```
template<typename T , typename Allocator = std::allocator<T>>
bool ManoVektorius< T, Allocator >::empty ( ) const  [inline], [noexcept]
```

Patikrina ar vektorius yra tuscias.

Returns

true jei vektorius yra tuscias, false priesingu atveju.

4.2.3.20 end() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator ManoVektorius< T, Allocator >::end ( ) const  [inline], [noexcept]
```

Grazina konstanta iteratoriu i pabaiga.

Returns

Konstanta iteratorius i pabaiga.

4.2.3.21 end() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
iterator ManoVektorius< T, Allocator >::end ( )  [inline], [noexcept]
```

Grazina iteratoriu i pabaiga.

Returns

Iteratorius i pabaiga.

4.2.3.22 erase()

```
template<typename T , typename Allocator = std::allocator<T>>
iterator ManoVektorius< T, Allocator >::erase (
            const_iterator pos )  [inline]
```

Pasalina elementa is nurodytos vietos.

Parameters

pos	- vieta, is kurios bus pasalintas elementas.

Returns

Iteratorius i sekanti elementa po pasalintą.

4.2.3.23 front() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference ManoVektorius< T, Allocator >::front ( )  [inline]
```

Grazina nuoroda i pirmaji elementa.

Returns

Nuoroda i pirmaji elementa.

4.2.3.24 front() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference ManoVektorius< T, Allocator >::front ( ) const  [inline]
```

Grazina konstanta nuoroda i pirmaji elementa.

Returns

Konstanta nuoroda i pirmaji elementa.

4.2.3.25 get_allocator()

```
template<typename T , typename Allocator = std::allocator<T>>
allocator_type ManoVektorius< T, Allocator >::get_allocator ( ) const  [inline], [noexcept]
```

Grazina alokatoriaus kopija.

Returns

Alokatoriaus kopija.

4.2.3.26 getCapacity()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type ManoVektorius< T, Allocator >::getCapacity ( ) const  [inline], [noexcept]
```

Grazina vektoriaus talpa.

Returns

Vektoriaus talpa.

4.2.3.27 insert()

```
template<typename T , typename Allocator = std::allocator<T>>
iterator ManoVektorius< T, Allocator >::insert (
            const_iterator pos,
            const T & value )  [inline]
```

Iterpia elementa i nurodyta vieta.

Parameters

pos	- vieta, kurioje bus iterptas elementas.
value	- iterpiamas elementas.

Returns

Iteratorius i iterpta elementa.

4.2.3.28 max_size()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type ManoVektorius< T, Allocator >::max_size ( ) const  [inline], [noexcept]
```

Grazina maksimalu vektoriaus dydi.

Returns

Maksimalus vektoriaus dydis.

4.2.3.29 operator=() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius & ManoVektorius< T, Allocator >::operator= (
            const ManoVektorius< T, Allocator > & other )  [inline]
```

Priskyrimo operatorius. Priskiria viena ManoVektorius objekta kitam.

Parameters

other	- objektas, kuris bus priskirtas.

Returns

∗this

4.2.3.30 operator=() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
ManoVektorius & ManoVektorius< T, Allocator >::operator= (
            ManoVektorius< T, Allocator > && other )  [inline], [noexcept]
```

Perkelimo priskyrimo operatorius. Perkelia duomenis is vieno ManoVektorius objekto i kita.

Parameters

other	- objektas, is kurio duomenys bus perkelti.

Returns

∗this

4.2.3.31 operator[]() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference ManoVektorius< T, Allocator >::operator[] (
            size_type pos )  [inline]
```

Perkrovimo operatorius []. Grazina elemento nuoroda pagal nurodyta pozicija.

Parameters

pos	- elemento pozicija.

Returns

Elemento nuoroda.

4.2.3.32 operator[]() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference ManoVektorius< T, Allocator >::operator[] (
            size_type pos ) const  [inline]
```

Perkrovimo operatorius []. Grazina konstanta elemento nuoroda pagal nurodyta pozicija.

Parameters

pos	- elemento pozicija.

Returns

Konstanta elemento nuoroda.

4.2.3.33 pop_back()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::pop_back ( )  [inline]
```

Pasalina elementa is vektoriaus galo.

Exceptions

std::out_of_range	jei vektorius yra tuscias.

4.2.3.34 push_back()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::push_back (
            const T & value )  [inline]
```

Prideda elementa i vektoriaus gala.

Parameters

value	- pridedamas elementas.

4.2.3.35 rbegin()

```
template<typename T , typename Allocator = std::allocator<T>>
reverse_iterator ManoVektorius< T, Allocator >::rbegin ( )  [inline], [noexcept]
```

Grazina apversta iteratoriu i pradzia.

Returns

Apverstas iteratorius i pradzia.

4.2.3.36 rend() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator ManoVektorius< T, Allocator >::rend ( ) const  [inline], [noexcept]
```

Grazina konstanta apversta iteratoriu i pabaiga.

Returns

Konstanta apverstas iteratorius i pabaiga.

4.2.3.37 rend() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reverse_iterator ManoVektorius< T, Allocator >::rend ( )  [inline], [noexcept]
```

Grazina apversta iteratoriu i pabaiga.

Returns

Apverstas iteratorius i pabaiga.

4.2.3.38 reserve()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::reserve (
            size_type new_cap )  [inline]
```

Rezervuoja atminti vektoriui.

Parameters

new_cap	- naujas talpinimo dydis.

4.2.3.39 resize()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::resize (
            size_type count,
            T value = T() )  [inline]
```

Keicia vektoriaus dydi.

Parameters

count	- naujas vektoriaus dydis.
value	- elementas, kuris bus naudojamas uzpildyti naujai sukurta vieta.

4.2.3.40 shrink_to_fit()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::shrink_to_fit ( )  [inline]
```

Sumazina vektoriaus talpa iki dabartinio dydzio.

4.2.3.41 size()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type ManoVektorius< T, Allocator >::size ( ) const  [inline], [noexcept]
```

Grazina vektoriaus dydi.

Returns

Vektoriaus dydis.

4.2.3.42 swap()

```
template<typename T , typename Allocator = std::allocator<T>>
void ManoVektorius< T, Allocator >::swap (
            ManoVektorius< T, Allocator > & other )  [inline], [noexcept]
```

Apkeicia du vektorius.

Parameters

other	- vektorius, su kuriuo bus apkeiciamas dabartinis vektorius.

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

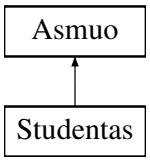
- C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_vektoriai/v3.0_vektoriai/ManoVektorius.h

4.3 Studentas Class Reference

Representuoja studenta ir jo asmenine informacija bei akademinius rezultatus.

`#include <studentas.h>`

Inheritance diagram for Studentas:



Public Member Functions

- **Studentas** ()

  Konstruktorius be parametru.
- Studentas (const std::string &v_, const std::string &p_, double egz_rez_, double rez_, const std::vector< double > &nd_rez_)

  Konstruktorius su parametrais.
- Studentas (const Studentas &other)

  Copy konstruktorius.
- Studentas (Studentas &&other) noexcept

  Move konstruktorius.
- Studentas & operator= (const Studentas &other)

  Copy priskyrimo operatorius.
- Studentas & operator= (Studentas &&other) noexcept

  Move priskyrimo operatorius.
- ~**Studentas** ()

  Destruktorius.
- std::string getV () const

  Grazina varda.
- std::string getP () const

  Grazina pavarde.
- std::string getName () const

  Grazina varda ir pavarde.
- const std::vector< double > & getNd_rez () const

  Grazina namu darbu rezultatus.
- double getEgz_rez () const

  Grazina egzamino rezultata.
- double getNd_sum () const

  Grazina namu darbu rezultatu suma.
- double getRez () const

  Grazina galutini rezultata.

- void setV (const std::string &v)

  Nustato varda.
- void setP (const std::string &p)

  Nustato pavarde.
- void setRez (double rez)

  Nustato galutini rezultata.
- void setEgz_rez (double egz_rez)

  Nustato egzamino rezultata.
- void setNd_rez (const std::vector< double > &nd_rez)

  Nustato namu darbu rezultatus ir atnaujina suma.
- void setNd_sum (double nd_sum)

  Nustato namu darbu rezultatu suma.
- double ApskaiciuotiMediana () const

  Apskaiciuoja namu darbu rezultatu mediana.
- double ApskaiciuotiVidurki () const

  Apskaiciuoja namu darbu rezultatu vidurki.
- double ApskaiciuotiGalutini (bool mediana)

  Apskaiciuoja galutini rezultata.
- void addNd_rez (double result)

  Prideda namu darbu rezultata.
- void resizeNd_rez (int n)

  Pakeicia namu darbu rezultatu dydi.
- void **popNd_rez** ()

  Pasalina paskutini namu darbu rezultata.
- void **clearNdRez** ()

  Isvalo namu darbu rezultatus.

Public Member Functions inherited from Asmuo

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

  Virtualus destruktorius.

Private Member Functions

- void **UpdateNdSum** ()

  Atnaujina namu darbu rezultatu suma.

Private Attributes

- std::string v
- std::string p
- double egz_rez
- std::vector< double > nd_rez
- double nd_sum = 0
- double rez

Friends

- std::istream & operator>> (std::istream &is, Studentas &studentas)

  Ivesties operatorius.
- std::ostream & operator<< (std::ostream &os, const Studentas &studentas)

  Isvesties operatorius.

4.3.1 Detailed Description

Representuoja studenta ir jo asmenine informacija bei akademinius rezultatus.

Si klase suteikia funkcionaluma saugoti ir manipuliuoti studentu duomenimis, iskaitant ju vardus, pavardes, egzamino rezultatus, namu darbu rezultatus. Taip pat si klase suteikia galimybe apskaiciuoti galutinius rezultatus.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Studentas() [1/3]

```
Studentas::Studentas (
            const std::string & v_,
            const std::string & p_,
            double egz_rez_,
            double rez_,
            const std::vector< double > & nd_rez_ )
```

Konstruktorius su parametrais.

Parameters

*v_*	Vardas
*p_*	Pavarde
*egz_rez_*	Egzamino rezultatas
*rez_*	Galutinis rezultatas
*nd_rez_*	Namu darbu rezultatai

4.3.2.2 Studentas() [2/3]

```
Studentas::Studentas (
            const Studentas & other )
```

Copy konstruktorius.

Parameters

other	Kito objekto kopija

4.3.2.3 Studentas() [3/3]

```
Studentas::Studentas (
            Studentas && other )  [noexcept]
```

Move konstruktorius.

Parameters

other	Kito objekto perkelimas

4.3.3 Member Function Documentation

4.3.3.1 addNd_rez()

```
void Studentas::addNd_rez (
            double result )  [inline]
```

Prideda namu darbu rezultata.

Parameters

result	Namu darbu rezultatas

4.3.3.2 ApskaiciuotiGalutini()

```
double Studentas::ApskaiciuotiGalutini (
            bool mediana )
```

Apskaiciuoja galutini rezultata.

Parameters

mediana	Ar naudoti mediana

Returns

Galutinis rezultatas

4.3.3.3 ApskaiciuotiMediana()

```
double Studentas::ApskaiciuotiMediana ( ) const
```

Apskaiciuoja namu darbu rezultatu mediana.

Returns

Namu darbu rezultatu mediana

4.3.3.4 ApskaiciuotiVidurki()

```
double Studentas::ApskaiciuotiVidurki ( ) const
```

Apskaiciuoja namu darbu rezultatu vidurki.

Returns

Namu darbu rezultatu vidurkis

4.3.3.5 getEgz_rez()

```
double Studentas::getEgz_rez ( ) const  [inline], [virtual]
```

Grazina egzamino rezultata.

Returns

Egzamino rezultatas

Implements Asmuo.

4.3.3.6 getName()

```
std::string Studentas::getName ( ) const  [inline]
```

Grazina varda ir pavarde.

Returns

Vardas ir pavarde

4.3.3.7 getNd_rez()

```
const std::vector< double > & Studentas::getNd_rez ( ) const  [inline], [virtual]
```

Grazina namu darbu rezultatus.

Returns

Namu darbu rezultatai

Implements Asmuo.

4.3.3.8 getNd_sum()

```
double Studentas::getNd_sum ( ) const  [inline], [virtual]
```

Grazina namu darbu rezultatu suma.

Returns

Namu darbu rezultatu suma

Implements Asmuo.

4.3.3.9 getP()

```
std::string Studentas::getP ( ) const  [inline], [virtual]
```

Grazina pavarde.

Returns

Pavarde

Implements Asmuo.

4.3.3.10 getRez()

```
double Studentas::getRez ( ) const  [inline], [virtual]
```

Grazina galutini rezultata.

Returns

Galutinis rezultatas

Implements Asmuo.

4.3.3.11 getV()

```
std::string Studentas::getV ( ) const  [inline], [virtual]
```

Grazina varda.

Returns

Vardas

Implements Asmuo.

4.3.3.12 operator=() [1/2]

```
Studentas & Studentas::operator= (
            const Studentas & other )
```

Copy priskyrimo operatorius.

Parameters

other	Kito objekto kopija

Returns

Sis objektas

4.3.3.13 operator=() [2/2]

```
Studentas & Studentas::operator= (
            Studentas && other )  [noexcept]
```

Move priskyrimo operatorius.

Parameters

other	Kito objekto perkelimas

Returns

Sis objektas

4.3.3.14 resizeNd_rez()

```
void Studentas::resizeNd_rez (
            int n )  [inline]
```

Pakeicia namu darbu rezultatu dydi.

Parameters

n	Naujas dydis

4.3.3.15 setEgz_rez()

```
void Studentas::setEgz_rez (
            double egz_rez )  [inline]
```

Nustato egzamino rezultata.

Parameters

egz_rez	Egzamino rezultatas

4.3.3.16 setNd_rez()

```
void Studentas::setNd_rez (
            const std::vector< double > & nd_rez )  [inline]
```

Nustato namu darbu rezultatus ir atnaujina suma.

Parameters

nd_rez	Namu darbu rezultatai

4.3.3.17 setNd_sum()

```
void Studentas::setNd_sum (
            double nd_sum )  [inline]
```

Nustato namu darbu rezultatu suma.

Parameters

nd_sum	Namu darbu rezultatu suma

4.3.3.18 setP()

```
void Studentas::setP (
            const std::string & p )  [inline]
```

Nustato pavarde.

Parameters

p	Pavarde

4.3.3.19 setRez()

```
void Studentas::setRez (
            double rez )  [inline]
```

Nustato galutini rezultata.

Parameters

rez	Galutinis rezultatas

4.3.3.20 setV()

```
void Studentas::setV (
            const std::string & v )  [inline]
```

Nustato varda.

Parameters

v	Vardas

4.3.4 Friends And Related Symbol Documentation

4.3.4.1 operator<<

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

Isvesties operatorius.

Parameters

os	Isvesties srautas
studentas	Studento objektas

Returns

Isvesties srautas

4.3.4.2 operator>>

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

Ivesties operatorius.

Parameters

is	Ivesties srautas
studentas	Studento objektas

Returns

Ivesties srautas

4.3.5 Member Data Documentation

4.3.5.1 egz_rez

```
double Studentas::egz_rez  [private]
```

Egzamino rezultatas

4.3.5.2 nd_rez

```
std::vector<double> Studentas::nd_rez  [private]
```

Namu darbu rezultatai

4.3.5.3 nd_sum

```
double Studentas::nd_sum = 0  [private]
```

Namu darbu rezultatu suma

4.3.5.4 p

```
std::string Studentas::p  [private]
```

Pavarde

4.3.5.5 rez

```
double Studentas::rez  [private]
```

Galutinis rezultatas

4.3.5.6 v

```
std::string Studentas::v  [private]
```

Vardas

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

- C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_vektoriai/v3.0_vektoriai/studentas.h
- C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_vektoriai/v3.0_vektoriai/studentas.cpp

Chapter 5

File Documentation

5.1 C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_↩ vektoriai/v3.0_vektoriai/asmuo.h File Reference

Sis failas apima Asmens klases deklaracija.

```
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>
#include <numeric>
#include <istream>
#include <ostream>
#include <iomanip>
#include <sstream>
```

Classes

- class Asmuo

  Bazine klase, kuri atstovauja asmeniui.

5.1.1 Detailed Description

Sis failas apima Asmens klases deklaracija.

5.2 asmuo.h

Go to the documentation of this file.

```
00001
00006 #ifndef ASMUO_H
00007 #define ASMUO_H
00008
00009 #include <string>
00010 #include <vector>
00011 #include <iostream>
00012 #include <algorithm>
00013 #include <numeric>
00014 #include <istream>
00015 #include <ostream>
00016 #include <iomanip>
00017 #include <sstream>
00018
00023 class Asmuo {
00024 public:
00025
00029     virtual ~Asmuo() = default;
00030
00035     virtual std::string getV() const = 0;
00036
00041     virtual std::string getP() const = 0;
00042
00047     virtual const std::vector<double>& getNd_rez() const = 0;
00048
00053     virtual double getEgz_rez() const = 0;
00054
00059     virtual double getNd_sum() const = 0;
00060
00065     virtual double getRez() const = 0;
00066
00067 };
00068
00069 #endif // ASMUO_H
```

5.3 funkcijos.h

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003
00004 #include <vector>
00005 #include "studentas.h"
00006
00007 void GeneruotiPazymius(std::vector<Studentas> & Duomenys);
00008 void GeneruotiPazymiusVardus();
00009 double Skaitymas(ManoVektorius<Studentas>& Duomenys1);
00010 void RusiuotiSpausdinti(ManoVektorius<Studentas>& Duomenys1, double laikas, const std::string&
      pavadinimas);
00011 void RankinisIvedimas(std::vector<Studentas>& Duomenys);
00012 void GeneruotiFailus();
00013 void RusiuotiGeraisBlogais(ManoVektorius<Studentas>& Duomenys1);
00014 std::vector<double> Testavimas(ManoVektorius<Studentas>& Duomenys1);
00015 void TestavimasRuleOfFive();
00016 void Vector_VS_ManoVektorius();
00017
00018 #endif
```

5.4 ManoVektorius.h

```
00001 #ifndef MANOVEKTORIUS_H
00002 #define MANOVEKTORIUS_H
00003
00004 #include <cstddef>
00005 #include <iterator>
00006 #include <memory>
00007 #include <stdexcept>
00008 #include <algorithm>
00009 #include <utility>
00010
00019 template <typename T, typename Allocator = std::allocator<T»
00020 class ManoVektorius
00021 {
00022 public:
00023
```

```
00024     using value_type = T;
00025     using allocator_type = Allocator;
00026     using size_type = std::size_t;
00027     using difference_type = std::ptrdiff_t;
00028     using reference = value_type&;
00029     using const_reference = const value_type&;
00030     using pointer = typename std::allocator_traits<Allocator>::pointer;
00031     using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;
00032     using iterator = pointer;
00033     using const_iterator = const_pointer;
00034     using reverse_iterator = std::reverse_iterator<iterator>;
00035     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00036
00037 private:
00038
00039     allocator_type allocator;
00040     pointer arr;
00041     size_type capacity;
00042     size_type current;
00043
00048     void destroy_elements()
00049     {
00050         for (size_type i = 0; i < current; ++i)
00051         {
00052             std::allocator_traits<Allocator>::destroy(allocator, arr + i);
00053         }
00054         current = 0;
00055     }
00056
00057 public:
00058
00063     ManoVektorius() : arr(nullptr), capacity(0), current(0) {}
00064
00065
00070     ~ManoVektorius()
00071     {
00072         destroy_elements();
00073         if (arr)
00074         {
00075             allocator.deallocate(arr, capacity);
00076         }
00077     }
00078
00079
00085     ManoVektorius(const ManoVektorius& other) : allocator(other.allocator), arr(nullptr), capacity(0),
      current(0)
00086     {
00087         if (other.current > 0)
00088         {
00089             arr = allocator.allocate(other.capacity);
00090             try
00091             {
00092                 for (current = 0; current < other.current; ++current)
00093                 {
00094                     std::allocator_traits<Allocator>::construct(allocator, arr + current,
      other.arr[current]);
00095                 }
00096                 capacity = other.capacity;
00097             }
00098             catch (...)
00099             {
00100                 destroy_elements();
00101                 allocator.deallocate(arr, other.capacity);
00102                 throw;
00103             }
00104         }
00105     }
00106
00113     ManoVektorius& operator=(const ManoVektorius& other)
00114     {
00115         if (this != &other)
00116         {
00117             ManoVektorius temp(other);
00118             swap(temp);
00119         }
00120         return *this;
00121     }
00122
00128     ManoVektorius(ManoVektorius&& other) noexcept : allocator(std::move(other.allocator)),
      arr(other.arr), capacity(other.capacity), current(other.current)
00129     {
00130         other.arr = nullptr;
00131         other.capacity = 0;
00132         other.current = 0;
00133     }
00134
00141     ManoVektorius& operator=(ManoVektorius&& other) noexcept
```

```
00142     {
00143         if (this != &other)
00144         {
00145             destroy_elements();
00146             if (arr)
00147             {
00148                 allocator.deallocate(arr, capacity);
00149             }
00150 
00151             allocator = std::move(other.allocator);
00152             arr = other.arr;
00153             capacity = other.capacity;
00154             current = other.current;
00155 
00156             other.arr = nullptr;
00157             other.capacity = 0;
00158             other.current = 0;
00159         }
00160         return *this;
00161     }
00162 
00169     reference at(size_type pos)
00170     {
00171         if (pos >= current)
00172         {
00173             throw std::out_of_range("ManoVektorius::at");
00174         }
00175         return arr[pos];
00176     }
00177 
00184     const_reference at(size_type pos) const
00185     {
00186         if (pos >= current)
00187         {
00188             throw std::out_of_range("ManoVektorius::at");
00189         }
00190         return arr[pos];
00191     }
00192 
00199     reference operator[](size_type pos)
00200     {
00201         return arr[pos];
00202     }
00203 
00210     const_reference operator[](size_type pos) const
00211     {
00212         return arr[pos];
00213     }
00214 
00219     allocator_type get_allocator() const noexcept
00220     {
00221         return allocator_type();
00222     }
00223 
00228     reference front()
00229     {
00230         return arr[0];
00231     }
00232 
00237     const_reference front() const
00238     {
00239         return arr[0];
00240     }
00241 
00246     reference back()
00247     {
00248         return arr[current - 1];
00249     }
00250 
00255     const_reference back() const
00256     {
00257         return arr[current - 1];
00258     }
00259 
00264     T* data() noexcept
00265     {
00266         return arr;
00267     }
00268 
00273     const T* data() const noexcept
00274     {
00275         return arr;
00276     }
00277 
00282     iterator begin() noexcept
00283     {
00284         return arr;
```

```
00285     }
00286
00291     const_iterator begin() const noexcept
00292     {
00293         return arr;
00294     }
00295
00300     const_iterator cbegin() const noexcept
00301     {
00302         return arr;
00303     }
00304
00309     iterator end() noexcept
00310     {
00311         return arr + current;
00312     }
00313
00318     const_iterator end() const noexcept
00319     {
00320         return arr + current;
00321     }
00322
00327     const_iterator cend() const noexcept
00328     {
00329         return arr + current;
00330     }
00331
00336     reverse_iterator rbegin() noexcept
00337     {
00338         return reverse_iterator(end());
00339     }
00340
00350     const_reverse_iterator crbegin() const noexcept
00351     {
00352         return const_reverse_iterator(end());
00353     }
00354
00359     reverse_iterator rend() noexcept
00360     {
00361         return reverse_iterator(begin());
00362     }
00363
00368     const_reverse_iterator rend() const noexcept
00369     {
00370         return const_reverse_iterator(begin());
00371     }
00372
00377     const_reverse_iterator crend() const noexcept
00378     {
00379         return const_reverse_iterator(begin());
00380     }
00381
00386     bool empty() const noexcept
00387     {
00388         return current == 0;
00389     }
00390
00395     size_type size() const noexcept
00396     {
00397         return current;
00398     }
00399
00404     size_type max_size() const noexcept
00405     {
00406         return std::numeric_limits<size_t>::max() / sizeof(T);
00407     }
00408
00413     void reserve(size_type new_cap)
00414     {
00415         if (new_cap > capacity)
00416         {
00417             pointer new_arr = allocator.allocate(new_cap);
00418             try
00419             {
00420                 for (size_type i = 0; i < current; ++i)
00421                 {
00422                     std::allocator_traits<Allocator>::construct(allocator, new_arr + i,
      std::move_if_noexcept(arr[i]));
00423                 }
00424             }
00425             catch (...)
00426             {
00427                 for (size_type i = 0; i < current; ++i)
00428                 {
00429                     std::allocator_traits<Allocator>::destroy(allocator, new_arr + i);
00430                 }
00431                 allocator.deallocate(new_arr, new_cap);
```

```
00432                 throw;
00433             }
00434             destroy_elements();
00435             allocator.deallocate(arr, capacity);
00436             arr = new_arr;
00437             capacity = new_cap;
00438         }
00439     }
00440
00445     size_type getCapacity() const noexcept
00446     {
00447         return capacity;
00448     }
00449
00453     void shrink_to_fit()
00454     {
00455         if (capacity > current)
00456         {
00457             pointer new_arr = allocator.allocate(current);
00458             try
00459             {
00460                 for (size_type i = 0; i < current; ++i)
00461                 {
00462                     std::allocator_traits<Allocator>::construct(allocator, new_arr + i,
      std::move_if_noexcept(arr[i]));
00463                 }
00464             }
00465             catch (...)
00466             {
00467                 for (size_type i = 0; i < current; ++i)
00468                 {
00469                     std::allocator_traits<Allocator>::destroy(allocator, new_arr + i);
00470                 }
00471                 allocator.deallocate(new_arr, current);
00472                 throw;
00473             }
00474             destroy_elements();
00475             allocator.deallocate(arr, capacity);
00476             arr = new_arr;
00477             capacity = current;
00478         }
00479     }
00480
00484     void clear() noexcept
00485     {
00486         destroy_elements();
00487     }
00488
00493     void push_back(const T& value)
00494     {
00495         if (current == capacity)
00496         {
00497             reserve(capacity == 0 ? 1 : capacity * 2);
00498         }
00499         std::allocator_traits<Allocator>::construct(allocator, arr + current, value);
00500         ++current;
00501     }
00502
00507     void pop_back()
00508     {
00509         if (current > 0)
00510         {
00511             --current;
00512             std::allocator_traits<Allocator>::destroy(allocator, arr + current);
00513         }
00514         else
00515         {
00516             throw std::out_of_range("Cannot pop_back from an empty ManoVektorius");
00517         }
00518     }
00519
00525     void resize(size_type count, T value = T())
00526     {
00527         if (count > current)
00528         {
00529             if (count > capacity)
00530             {
00531                 reserve(count);
00532             }
00533             for (size_type i = current; i < count; ++i)
00534             {
00535                 std::allocator_traits<Allocator>::construct(allocator, arr + i, value);
00536             }
00537         }
00538         else
00539         {
00540             for (size_type i = count; i < current; ++i)
```

```
00541             {
00542                 std::allocator_traits<Allocator>::destroy(allocator, arr + i);
00543             }
00544         }
00545         current = count;
00546     }
00547
00552     void swap(ManoVektorius& other) noexcept
00553     {
00554         std::swap(arr, other.arr);
00555         std::swap(capacity, other.capacity);
00556         std::swap(current, other.current);
00557         std::swap(allocator, other.allocator);
00558     }
00559
00566     iterator insert(const_iterator pos, const T& value)
00567     {
00568         size_type index = std::distance(cbegin(), pos);
00569         if (current == capacity)
00570         {
00571             reserve(capacity == 0 ? 1 : capacity * 2);
00572         }
00573         if (index < current)
00574         {
00575             std::move_backward(arr + index, arr + current, arr + current + 1);
00576         }
00577         std::allocator_traits<Allocator>::construct(allocator, arr + index, value);
00578         ++current;
00579         return arr + index;
00580     }
00581
00588     template <typename... Args>
00589     iterator emplace(const_iterator pos, Args &&...args)
00590     {
00591         size_type index = std::distance(cbegin(), pos);
00592         if (current == capacity)
00593         {
00594             reserve(capacity == 0 ? 1 : capacity * 2);
00595         }
00596         std::move_backward(arr + index, arr + current, arr + current + 1);
00597         arr[index] = T(std::forward<Args>(args)...);
00598         ++current;
00599         return arr + index;
00600     }
00601
00607     iterator erase(const_iterator pos)
00608     {
00609         size_type index = std::distance(cbegin(), pos);
00610         std::allocator_traits<Allocator>::destroy(allocator, arr + index);
00611         std::move(arr + index + 1, arr + current, arr + index);
00612         --current;
00613         return arr + index;
00614     }
00615
00620     template <typename InputIt>
00621     void assign(InputIt first, InputIt last)
00622     {
00623         size_type count = std::distance(first, last);
00624         if (count > capacity)
00625         {
00626             clear();
00627             allocator.deallocate(arr, capacity);
00628             arr = allocator.allocate(count);
00629             capacity = count;
00630         }
00631         for (current = 0; current < count; ++current, ++first)
00632         {
00633             std::allocator_traits<Allocator>::construct(allocator, arr + current, *first);
00634         }
00635     }
00636
00642     void assign(size_type count, const T& value)
00643     {
00644         if (count > capacity)
00645         {
00646             clear();
00647             allocator.deallocate(arr, capacity);
00648             arr = allocator.allocate(count);
00649             capacity = count;
00650         }
00651         for (current = 0; current < count; ++current)
00652         {
00653             std::allocator_traits<Allocator>::construct(allocator, arr + current, value);
00654         }
00655     }
00656
00661     template <typename... Args>
```

```
00662     void emplace_back(Args &&...args)
00663     {
00664         if (current == capacity)
00665         {
00666             reserve(capacity == 0 ? 1 : capacity * 2);
00667         }
00668         std::allocator_traits<Allocator>::construct(allocator, arr + current,
      std::forward<Args>(args)...);
00669         ++current;
00670     }
00671
00672 template <typename InputIt>
00677     void append_range(InputIt first, InputIt last)
00678     {
00679         size_type count = std::distance(first, last);
00680         if (current + count > capacity)
00681         {
00682             reserve(current + count);
00683         }
00684         for (; first != last; ++first, ++current)
00685         {
00686             std::allocator_traits<Allocator>::construct(allocator, arr + current, *first);
00687         }
00688     }
00689 };
00690
00691 template <typename T, typename Allocator>
00697 bool operator==(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00698 {
00699     return lhs.size() == rhs.size() && std::equal(lhs.begin(), lhs.end(), rhs.begin());
00700 }
00701
00702 template <typename T, typename Allocator>
00708 bool operator!=(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00709 {
00710     return !(lhs == rhs);
00711 }
00712
00713 template <typename T, typename Allocator>
00719 bool operator<(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00720 {
00721     return std::lexicographical_compare(lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
00722 }
00723
00724 template <typename T, typename Allocator>
00730 bool operator<=(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00731 {
00732     return !(rhs < lhs);
00733 }
00734
00735 template <typename T, typename Allocator>
00741 bool operator>(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00742 {
00743     return rhs < lhs;
00744 }
00745
00746 template <typename T, typename Allocator>
00752 bool operator>=(const ManoVektorius<T, Allocator>& lhs, const ManoVektorius<T, Allocator>& rhs)
00753 {
00754     return !(lhs < rhs);
00755 }
00756
00757 namespace std
00758 {
00759     template <typename T, typename Allocator>
00764     void swap(ManoVektorius<T, Allocator>& lhs, ManoVektorius<T, Allocator>& rhs) noexcept
00765     {
00766         lhs.swap(rhs);
00767     }
00768 }
00769
00770 template <typename T, typename Allocator, typename Pred>
00776 void erase(ManoVektorius<T, Allocator>& vec, Pred pred)
00777 {
00778     vec.erase(std::remove_if(vec.begin(), vec.end(), pred), vec.end());
00779 }
00780
00781 template <typename T, typename Allocator, typename Pred>
00787 void erase_if(ManoVektorius<T, Allocator>& vec, Pred pred)
00788 {
00789     vec.erase(std::remove_if(vec.begin(), vec.end(), pred), vec.end());
00790 }
00791
00792 #endif
```

5.5 C:/Users/matas/OneDrive/Desktop/vu/Objektinis/v3.0/v3.0_↩ vektoriai/v3.0_vektoriai/studentas.h File Reference

Sis failas apima Studento klases deklaracija.

```
#include "Asmuo.h"
#include "ManoVektorius.h"
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>
#include <numeric>
#include <iomanip>
#include <limits>
#include <random>
#include <fstream>
#include <sstream>
```

Classes

- class Studentas

  Representuoja studenta ir jo asmenine informacija bei akademinius rezultatus.

5.5.1 Detailed Description

Sis failas apima Studento klases deklaracija.

5.6 studentas.h

Go to the documentation of this file.

```
00001
00006 #ifndef STUDENTAS_H
00007 #define STUDENTAS_H
00008
00009 #include "Asmuo.h"
00010 #include "ManoVektorius.h"
00011
00012 #include <string>
00013 #include <vector>
00014 #include <algorithm>
00015 #include <iostream>
00016 #include <numeric>
00017 #include <iomanip>
00018 #include <limits>
00019 #include <random>
00020 #include <fstream>
00021 #include <sstream>
00022
00030 class Studentas : public Asmuo {
00031
00032 private:
00033
00034     std::string v;
00035     std::string p;
00036     double egz_rez;
00037     std::vector<double> nd_rez;
00038     double nd_sum = 0;
00039     double rez;
00044     void UpdateNdSum() { nd_sum = std::accumulate(nd_rez.begin(), nd_rez.end(), 0.0); }
00045
```

```
00046 public:
00047
00051     Studentas();
00052
00061     Studentas(const std::string& v_, const std::string& p_, double egz_rez_, double rez_, const
      std::vector<double>& nd_rez_);
00062
00067     Studentas(const Studentas& other);
00068
00073     Studentas(Studentas&& other) noexcept;
00074
00080     Studentas& operator=(const Studentas& other);
00081
00087     Studentas& operator=(Studentas&& other) noexcept;
00088
00095     friend std::istream& operator>>(std::istream& is, Studentas& studentas);
00096
00103     friend std::ostream& operator<<(std::ostream& os, const Studentas& studentas);
00104
00108     ~Studentas() {
00109         v.clear();
00110         p.clear();
00111         nd_rez.clear();
00112     }
00113
00118     inline std::string getV() const { return v; }
00119
00124     inline std::string getP() const { return p; }
00125
00130     std::string getName() const { return getV() + " " + getP(); }
00131
00136     const std::vector<double>& getNd_rez() const { return nd_rez; }
00137
00142     double getEgz_rez() const { return egz_rez; }
00143
00148     double getNd_sum() const { return nd_sum; }
00149
00154     double getRez() const { return rez; }
00155
00160     void setV(const std::string& v) { this->v = v; }
00161
00166     void setP(const std::string& p) { this->p = p; }
00167
00172     void setRez(double rez) { this->rez = rez; }
00173
00178     void setEgz_rez(double egz_rez) { this->egz_rez = egz_rez; }
00179
00184     void setNd_rez(const std::vector<double>& nd_rez) { this->nd_rez = nd_rez; UpdateNdSum(); }
00185
00190     void setNd_sum(double nd_sum) { this->nd_sum = nd_sum; }
00191
00196     double ApskaiciuotiMediana() const;
00197
00202     double ApskaiciuotiVidurki() const;
00203
00209     double ApskaiciuotiGalutini(bool mediana);
00210
00215     void addNd_rez(double result) { nd_rez.push_back(result); }
00216
00221     void resizeNd_rez(int n) { nd_rez.resize(n); }
00222
00226     void popNd_rez() { nd_rez.pop_back(); }
00227
00231     void clearNdRez() { nd_rez.clear(); }
00232
00233 };
00234
00235 #endif
```

Index