OOPFinalinis

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

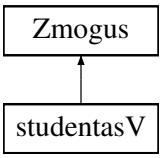
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 studentasV Class Reference

Inheritance diagram for studentasV:



Public Member Functions

- studentasV ()

  Default studento objekto konstruktorius.
- void setVardas (std::string vardas) override

  Ovveride'inta Zmogus klases funkcija setVardas - nustato studento varda.
- std::string getVardas () const override

  Ovveride'inta Zmogus klases funkcija getVardas - grazina studento varda.
- void setPavarde (std::string pavarde) override

  Ovveride'inta Zmogus klases funkcija setPavarde - nustato studento pavarde.
- std::string getPavarde () const override

  Ovveride'inta Zmogus klases funkcija getPavarde - grazina studento pavarde.
- void setPazymiaiVector (const Vector< int > &pazVector)

  funkcija setPazymiaiVector - priskiriami pazymiai (vector<int> tipo).
- void setPazymiai (int paz)

  funkcija setPazymiai - priskiriami pazymiai (int tipo).
- Vector< int > getPazymiai () const

  funkcija getPazymiai - grazina pazymius.
- void resizePazymiai (int n)

  funkcija resizePazymiai - pakeicia std::vector<int> pazymiai dydi.
- void setEgzaminas (int egzaminas)

  funkcija setEgzaminas - iraso studento egzamino rezultata.
- int getEgzaminas () const

  funkcija getEgzaminas - grazina mokinio egzamino rezultata.

- void **setVidurkis** ()

  funkcija setVidurkis - apskaiciuoja studento vidurki.
- void **setMediana** ()

  funkcija setMediana - apskaiciuoja studento mediana.
- float getVidurkis () const

  funkcija getVidurkis - grazina studento vidurki.
- float getMediana () const

  funkcija getMediana - grazina studento mediana.
- void **setAtsitiktiniaiPazymiai** ()

  funkcija setAtsitiktiniaiPazymiai - ivestiems studentu vardams ranka, sugeneruoja pazymius ir egzamina.
- void setAtsitiktiniaiDuomenys ()
- studentasV (const studentasV &kita)

  rule of five kopijavimo konstruktorius - kopijuoja visa objekta su jo duomenimis i kita objekta.
- studentasV & operator= (const studentasV &kita)

  rule of five priskyrimo konstruktorius - priskiria vieno objekto duomenis kitam objektui
- studentasV (studentasV &&kita) noexcept

  rule of five perkelimo konstruktorius - perkelia objekta is vieno i kita pasalindamas duomenis is pirmojo
- studentasV & operator= (studentasV &&kita) noexcept

  rule of five perkelimo priskyrimo operatorius - perkelia objekta is vieno i kita pasalindamas duomenis is pirmojo

Public Member Functions inherited from Zmogus

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

  Virtualus objekto destruktorius.

Friends

- std::ostream & operator<< (std::ostream &os, const studentasV &studentas)

  Overloadinamas ostream operatorius << skirtas darbui su objektu studentas.
- std::istream & operator>> (std::istream &is, studentasV &studentas)

  Overloadinamas istream operatorius >> skirtas darbui su objektu studentas.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 studentasV() [1/3]

```
studentasV::studentasV ( )  [inline]
```

Default studento objekto konstruktorius.

>Studento egzamino ivertinimas.

4.1.1.2 studentasV() [2/3]

```
studentasV::studentasV (
            const studentasV & kita )
```

rule of five kopijavimo konstruktorius - kopijuoja visa objekta su jo duomenimis i kita objekta.

Parameters

kita	

4.1.1.3 studentasV() [3/3]

```
studentasV::studentasV (
            studentasV && kita )  [noexcept]
```

rule of five perkelimo konstruktorius - perkelia objekta is vieno i kita pasalindamas duomenis is pirmojo

Parameters

kita	

4.1.2 Member Function Documentation

4.1.2.1 getEgzaminas()

```
int studentasV::getEgzaminas ( ) const
```

funkcija getEgzaminas - grazina mokinio egzamino rezultata.

Returns

egzaminas

4.1.2.2 getMediana()

```
float studentasV::getMediana ( ) const
```

funkcija getMediana - grazina studento mediana.

Returns

mediana

4.1.2.3 getPavarde()

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

Ovveride'inta Zmogus klases funkcija getPavarde - grazina studento pavarde.

Returns

pavarde

Implements Zmogus.

4.1.2.4 getPazymiai()

```
Vector< int > studentasV::getPazymiai ( ) const
```

funkcija getPazymiai - grazina pazymius.

Returns

std::vector<int> pazymiai

4.1.2.5 getVardas()

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

Ovveride'inta Zmogus klases funkcija getVardas - grazina studento varda.

Returns

vardas

Implements Zmogus.

4.1.2.6 getVidurkis()

```
float studentasV::getVidurkis ( ) const
```

funkcija getVidurkis - grazina studento vidurki.

Returns

vidurkis

4.1.2.7 operator=() [1/2]

```
studentasV & studentasV::operator= (
            const studentasV & kita )
```

rule of five priskyrimo konstruktorius - priskiria vieno objekto duomenis kitam objektui

Parameters

kita	

4.1.2.8 operator=() [2/2]

```
studentasV & studentasV::operator= (
            studentasV && kita )  [noexcept]
```

rule of five perkelimo priskyrimo operatorius - perkelia objekta is vieno i kita pasalindamas duomenis is pirmojo

Parameters

kita	

4.1.2.9 resizePazymiai()

```
void studentasV::resizePazymiai (
            int n )
```

funkcija resizePazymiai - pakeicia std::vector<int> pazymiai dydi.

Parameters

n	

4.1.2.10 setAtsitiktiniaiDuomenys()

```
void studentasV::setAtsitiktiniaiDuomenys ( )
```

funkcija setAtsitiktiniaiDuomenys - sugeneruoja studentus ir ju pazymius ir egzamina.

4.1.2.11 setEgzaminas()

```
void studentasV::setEgzaminas (
            int egzaminas )
```

funkcija setEgzaminas - iraso studento egzamino rezultata.

Parameters

egzaminas	

4.1.2.12 setPavarde()

```
void studentasV::setPavarde (
            std::string pavarde )  [override], [virtual]
```

Ovveride'inta Zmogus klases funkcija setPavarde - nustato studento pavarde.

Parameters

pavarde	

Implements Zmogus.

4.1.2.13 setPazymiai()

```
void studentasV::setPazymiai (
            int paz )
```

funkcija setPazymiai - priskiriami pazymiai (int tipo).

Parameters

paz	

4.1.2.14 setPazymiaiVector()

```
void studentasV::setPazymiaiVector (
            const Vector< int > & pazVector )
```

funkcija setPazymiaiVector - priskiriami pazymiai (vector<int> tipo).

Parameters

pazVector	

4.1.2.15 setVardas()

```
void studentasV::setVardas (
            std::string vardas )  [override], [virtual]
```

Ovveride'inta Zmogus klases funkcija setVardas - nustato studento varda.

Parameters

vardas	

Implements Zmogus.

4.1.3 Friends And Related Symbol Documentation

4.1.3.1 operator<<

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

Overloadinamas ostream operatorius << skirtas darbui su objektu studentas.

Parameters

os	outputstream kintamasis
studentas	

4.1.3.2 operator>>

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

Overloadinamas istream operatorius >> skirtas darbui su objektu studentas.

Parameters

is	
studentas	

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

- studentas.h
- studentas.cpp

4.2 Vector< T > Class Template Reference

Šabloninė Vector klasė

```
#include <vectorClass.h>
```

Public Types

- typedef T value_type
- typedef T & reference
- typedef const T & const_reference
- typedef T * iterator
- typedef const T * const_iterator
- typedef std::ptrdiff_t difference_type
- typedef size_t size_type

Public Member Functions

- **Vector** ()

  Default konstruktorius.
- Vector (size_type n, const_reference value)

  Konstruktorius su parametrais.
- Vector (const Vector &rhs)

  Kopijavimo konstruktorius.
- ~**Vector** ()

  Destruktorius.
- size_type Size () const

  Grąžina elementų skaičių
- size_type Capacity () const

  Grąžina vietos kiekį atmintyje.
- Vector (Vector &&other) noexcept

  Konstruktorius su perkeliamaisiais parametrais.
- Vector & operator= (Vector other) noexcept

  Priskyrimo operatorius su perkeliamaisiais parametrais.
- bool isEmpty () const

  Patikrina ar vektorius yra tuščias.
- void PushBack (const_reference object)

  Prideda elementą į vektorių
- void **PopBack** ()

  Išmeta paskutinį elementą iš vektoriaus.
- void **Clear** ()

  Išvalo vektorių
- void Resize (size_type n)

  Keičia vektoriaus dydį
- void Reserve (size_type n)

  Rezervuoja vietos atmintyje.
- void Swap (Vector &rhs) noexcept

  Apkeičia vektorius.
- void **ShrinkToFit** ()

  Sumažina vektoriaus vietos atmintyje kiekį iki elemento skaičiaus.
- void Erase (size_type index)

  Ištrina elementą pagal indeksą
- void Erase (iterator position, iterator last)

  Ištrina elementus intervalo viduje.
- void Insert (size_type index, const_reference object)

  Įterpia elementą į vektorių
- void Insert (iterator pos, const_iterator first, const_iterator last)

  Įterpia elementus į vektorių
- reference operator[ ] (size_t index)

  Grąžina elementą pagal indeksą
- const_reference operator[ ] (size_t index) const

  Grąžina elementą pagal indeksą
- reference At (size_type index)

  Grąžina elementą pagal indeksą
- reference Front ()

  Grąžina pirmą elementą
- reference Back ()

Grąžina paskutinį elementą

- iterator begin ()

  Grąžina iteratorių į pirmą elementą
- const_iterator begin () const

  Grąžina iteratorių į pirmą elementą
- iterator end ()

  Grąžina iteratorių į paskutinį elementą
- const_iterator end () const

  Grąžina iteratorių į paskutinį elementą
- void EmplaceBack (T &&object)

  Prideda elementą į vektorių

4.2.1 Detailed Description

template<typename T>
class Vector< T >

Šabloninė Vector klasė

Template Parameters

T	Elementų tipas

Vector klasė

Ši klasė realizuoja vektorių, kuris yra dinaminis masyvas.

4.2.2 Member Typedef Documentation

4.2.2.1 const_iterator

```
template<typename T >
typedef const T* Vector< T >::const_iterator
```

Konstantinis iteratorius

4.2.2.2 const_reference

```
template<typename T >
typedef const T& Vector< T >::const_reference
```

Konstantine elemento referencija

4.2.2.3 difference_type

```
template<typename T >
typedef std::ptrdiff_t Vector< T >::difference_type
```

Skirtumas tarp dvieju iteratoriu

4.2.2.4 iterator

```
template<typename T >
typedef T* Vector< T >::iterator
```

Iteratorius

4.2.2.5 reference

```
template<typename T >
typedef T& Vector< T >::reference
```

Elemento referencija

4.2.2.6 size_type

```
template<typename T >
typedef size_t Vector< T >::size_type
```

Elementu skaicius

4.2.2.7 value_type

```
template<typename T >
typedef T Vector< T >::value_type
```

Elemento tipas

4.2.3 Constructor & Destructor Documentation

4.2.3.1 Vector() [1/3]

```
template<typename T >
Vector< T >::Vector (
            size_type n,
            const_reference value )  [inline]
```

Konstruktorius su parametrais.

Parameters

n	Elementų skaičius
value	Pradinė reikšmė

4.2.3.2 Vector() [2/3]

```
template<typename T >
```

```
Vector< T >::Vector (
            const Vector< T > & rhs )  [inline]
```

Kopijavimo konstruktorius.

Parameters

rhs	Kopijuojamas vektorius

4.2.3.3 Vector() [3/3]

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

Konstruktorius su perkeliamaisiais parametrais.

Parameters

other	Perkeliamas vektorius

4.2.4 Member Function Documentation

4.2.4.1 At()

```
template<typename T >
reference Vector< T >::At (
            size_type index )  [inline]
```

Grąžina elementą pagal indeksą

Parameters

index	Indeksas

Returns

Elementas

4.2.4.2 Back()

```
template<typename T >
reference Vector< T >::Back ( )  [inline]
```

Grąžina paskutinį elementą

Returns

Paskutinis elementas

4.2.4.3 begin() [1/2]

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

Grąžina iteratorių į pirmą elementą

Returns

Iteratorius į pirmą elementą

4.2.4.4 begin() [2/2]

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

Grąžina iteratorių į pirmą elementą

Returns

Iteratorius į pirmą elementą

4.2.4.5 Capacity()

```
template<typename T >
size_type Vector< T >::Capacity ( ) const  [inline]
```

Grąžina vietos kiekį atmintyje.

Returns

Vietos kiekis atmintyje

4.2.4.6 EmplaceBack()

```
template<typename T >
void Vector< T >::EmplaceBack (
            T && object )  [inline]
```

Prideda elementą į vektorių

Parameters

object	Pridedamas elementas

4.2.4.7 end() [1/2]

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

Grąžina iteratorių į paskutinį elementą

Returns

Iteratorius į paskutinį elementą

4.2.4.8 end() [2/2]

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

Grąžina iteratorių į paskutinį elementą

Returns

Iteratorius į paskutinį elementą

4.2.4.9 Erase() [1/2]

```
template<typename T >
void Vector< T >::Erase (
            iterator position,
            iterator last )  [inline]
```

Ištrina elementus intervalo viduje.

Parameters

position	Pradžios iteratorius
last	Pabaigos iteratorius

4.2.4.10 Erase() [2/2]

```
template<typename T >
void Vector< T >::Erase (
            size_type index )  [inline]
```

Ištrina elementą pagal indeksą

Parameters

index	Indeksas

4.2.4.11 Front()

```
template<typename T >
reference Vector< T >::Front ( )  [inline]
```

Grąžina pirmą elementą

Returns

Pirmas elementas

4.2.4.12 Insert() [1/2]

```
template<typename T >
void Vector< T >::Insert (
            iterator pos,
            const_iterator first,
            const_iterator last )  [inline]
```

Įterpia elementus į vektorių

Parameters

pos	Įterpimo vieta
first	Pradžios iteratorius
last	Pabaigos iteratorius

4.2.4.13 Insert() [2/2]

```
template<typename T >
void Vector< T >::Insert (
            size_type index,
            const_reference object )  [inline]
```

Įterpia elementą į vektorių

Parameters

index	Įterpimo vieta
object	Įterpiamas elementas

4.2.4.14 isEmpty()

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

Patikrina ar vektorius yra tuščias.

Returns

Ar tuščias

4.2.4.15 operator=()

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

Priskyrimo operatorius su perkeliamaisiais parametrais.

Parameters

other	Perkeliamas vektorius

Returns

Šis vektorius

4.2.4.16 operator[]() [1/2]

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

Grąžina elementą pagal indeksą

Parameters

index	Indeksas

Returns

Elementas

4.2.4.17 operator[]() [2/2]

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

Grąžina elementą pagal indeksą

Parameters

index	Indeksas

Returns

Elementas

4.2.4.18 PushBack()

```
template<typename T >
void Vector< T >::PushBack (
            const_reference object )  [inline]
```

Prideda elementą į vektorių

Parameters

object	Pridedamas elementas

4.2.4.19 Reserve()

```
template<typename T >
void Vector< T >::Reserve (
            size_type n )  [inline]
```

Rezervuoja vietos atmintyje.

Parameters

n	Vietos kiekis

4.2.4.20 Resize()

```
template<typename T >
void Vector< T >::Resize (
            size_type n )  [inline]
```

Keičia vektoriaus dydį

Parameters

n	Naujasis dydis

4.2.4.21 Size()

```
template<typename T >
size_type Vector< T >::Size ( ) const  [inline]
```

Grąžina elementų skaičių

Returns

Elementų skaičius

4.2.4.22 Swap()

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

Apkeičia vektorius.

Parameters

rhs	Apkeičiamas vektorius

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

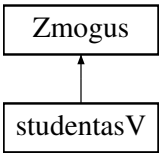
- vectorClass.h

4.3 Zmogus Class Reference

Base klasė Zmogus kuria zmogaus objektus - vardas, pavarde.

```
#include <studentas.h>
```

Inheritance diagram for Zmogus:



Public Member Functions

- virtual void setVardas (std::string vardas)=0
  Objektui priskiria varda.
- virtual std::string getVardas () const =0
  Grazina objekto varda.
- virtual void setPavarde (std::string pavarde)=0
  Objektui priskiria pavarde.
- virtual std::string getPavarde () const =0
  Grazina objekto pavarde.
- virtual ~**Zmogus** ()=default
  Virtualus objekto destruktorius.

4.3.1 Detailed Description

Base klasė Zmogus kuria zmogaus objektus - vardas, pavarde.

Ši klasė yra abstrakti, t.y. apibrezia abstrakcius metodus, kurie turi buti perkrauti paveldetose klasese

4.3.2 Member Function Documentation

4.3.2.1 getPavarde()

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

Grazina objekto pavarde.

Returns

pavarde Grazinama pavarde.

Implemented in studentasV.

4.3.2.2 getVardas()

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

Grazina objekto varda.

Returns

vardas Grazinamas objekto vardas.

Implemented in studentasV.

4.3.2.3 setPavarde()

```
virtual void Zmogus::setPavarde (
            std::string pavarde )  [pure virtual]
```

Objektui priskiria pavarde.

Parameters

pavarde	Nustatoma pavarde.

Implemented in studentasV.

4.3.2.4 setVardas()

```
virtual void Zmogus::setVardas (
            std::string vardas )  [pure virtual]
```

Objektui priskiria varda.

Parameters

vardas	Nustatomas vardas.

Implemented in studentasV.

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

- studentas.h

Chapter 5

File Documentation

5.1 Common.h

```
00001 #pragma once
00002 #include "PazymiaiVectors.h"
00003
00008 void meniu(string& pasirinkimas);
00009
00014 void sortChoice(string& choice);
00015
00020 void pasirinkimasVidMed(string& vidMed);
00021
00026 void pasirinkimasIvedimas(string& ivedimas);
00027
00032 void failoGeneravimasIsvedimas(int iteracija);
00033
00037 void pazymiuFailoGeneravimas();
```

5.2 PazymiaiVectors.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <string>
00004 #include <iomanip>
00005 #include <cstring>
00006 #include <utility>
00007 #include <vector>
00008 #include <numeric>
00009 #include "studentas.h"
00010 #include "vectorClass.h"
00011 using namespace std;
00012
00018 void ivedimas(Vector<studentasV>& grupeVector, int n);
00019
00026 void ivedimasV(Vector<studentasV>& grupeVector, studentasV& stud, int studentoNr, int pazymiuKiekis);
00031 void ivedimasNoSize(Vector<studentasV>& grupeVector);
00032
00037 void ivedimasCaseTwo(Vector<studentasV>& grupeVector);
00038
00047 void fileReading(Vector<studentasV>& grupeVector, const string& failas, double & laikasSkaitymas, int&
      fakePazymiai, double& laikasSkaiciavimas);
00048
00053 void generateRandomGrades(studentasV &stud);
00054
00059 void generateRandomNames(studentasV &stud);
00060
00065 void generalVidurkisCalculate(Vector<studentasV>& grupeVector);
00066
00071 void generalMedianaCalculate(Vector<studentasV>& grupeVector);
00072
00082 void isvedimas(Vector<studentasV> grupeVector, double laikasSkaitymas, double laikasSkaiciavimas,
      double laikasRusiavimas, int fakePazymiai, int iteracija);
00083
00089 void readNumbersV(studentasV& stud, int maxItems);
00090
00096 void sortInput(string& choice, Vector<studentasV>& grupeVector);
00097
```

```
00108 void failoNuskaitymasRusiavimas(Vector<studentasV>& grupeVector, Vector<studentasV>& grupeBad,
      Vector<studentasV>& grupeGood, int i, string vidMed, string ivedimasKonteineris, string choice);
00109
00118 void isvedimasFailai(Vector<studentasV>& grupeVector, Vector<studentasV>& grupeBad, int i, string&
      vidMed, string& choice);
00119
00124 void pasirinkimasVidMed(string& vidMed);
00125
00135 void vektoriaiMain(string vidMed, string choice, Vector<studentasV>& grupeVector, Vector<studentasV>&
      grupeBad, Vector<studentasV>& grupeGood, string ivedimasKonteineris);
00136
00144 void vectorPartition(string vidMed, Vector<studentasV>& grupeVector, Vector<studentasV>& grupeGood,
      Vector<studentasV>& grupeBad);
00145
00152 void vectorPartition2(string vidMed, Vector<studentasV>& grupeVector, Vector<studentasV>& grupeBad);
```

5.3 studentas.h

```
00001 //
00002 // Created by Adomas on 15/04/2024.
00003 //
00004
00005 #ifndef OOPUZD_STUDENTAS_H
00006 #define OOPUZD_STUDENTAS_H
00007
00008 #include <iostream>
00009 #include <string>
00010 #include <vector>
00011 #include "numeric"
00012 #include <algorithm>
00013 #include "vectorClass.h"
00019 class Zmogus {
00020 public:
00021
00027     virtual void setVardas(std::string vardas) = 0;
00028
00034     virtual std::string getVardas() const = 0;
00035
00041     virtual void setPavarde(std::string pavarde) = 0;
00042
00048     virtual std::string getPavarde() const = 0;
00049
00053     virtual ~Zmogus() = default;
00054 };
00055
00056 class studentasV : public Zmogus {
00057 private:
00058     std::string vardas = " ";
00059     std::string pavarde = " ";
00060     float vidurkis = 0.0;
00061     float mediana = 0.0;
00062     Vector<int> pazymiai;
00063     double egzaminas{};
00064 public:
00065
00069     studentasV() : egzaminas(0.0) {}
00070
00076     void setVardas(std::string vardas) override;
00077
00083     std::string getVardas() const override;
00084
00090     void setPavarde(std::string pavarde) override;
00091
00097     std::string getPavarde() const override;
00098
00104     void setPazymiaiVector(const Vector<int>& pazVector);
00105
00111     void setPazymiai(int paz);
00112
00117     Vector<int> getPazymiai() const;
00118
00123     void resizePazymiai(int n);
00128     void setEgzaminas(int egzaminas);
00129
00134     int getEgzaminas() const;
00135
00139     void setVidurkis();
00140
00144     void setMediana();
00145
00150     float getVidurkis() const;
00151
00156     float getMediana() const;
```

```
00157
00161     void setAtsitiktiniaiPazymiai();
00162
00166     void setAtsitiktiniaiDuomenys();
00167
00168 //    //Rule of Five headers
00173     studentasV(const studentasV &kita);
00174
00179     studentasV &operator=(const studentasV &kita);
00180
00185     studentasV(studentasV &&kita) noexcept;
00186
00191     studentasV &operator=(studentasV &&kita) noexcept; //Perkelimo priskyrimo konstruktorius
00192     ~studentasV(); //Destruktorius
00193
00199     friend std::ostream &operator«(std::ostream &os, const studentasV &studentas);
00200
00206     friend std::istream &operator»(std::istream &is, studentasV &studentas);
00207 };
00208
00209 #endif //OOPUZD_STUDENTAS_H
```

5.4 vectorClass.h

```
00001 //
00002 // Created by adoma on 5/13/2024.
00003 //
00004
00005 #ifndef OOPUZD_VECTORCLASS_H
00006 #define OOPUZD_VECTORCLASS_H
00007
00008 #include <cstddef>
00009 #include <stdexcept>
00010 #include <algorithm> // for std::move and std::move_backward
00011
00017 template<typename T>
00023 class Vector{
00024 private:
00025     int size;
00026     int capacity;
00027     T* elements;
00028
00029 public:
00030
00031     typedef T value_type;
00032     typedef T& reference;
00033     typedef const T& const_reference;
00034     typedef T* iterator;
00035     typedef const T* const_iterator;
00036     typedef std::ptrdiff_t difference_type;
00037     typedef size_t size_type;
00042     Vector() : size(0), capacity(10), elements(new value_type[capacity]) {}
00043
00050     Vector(size_type n, const_reference value) : size(n), capacity(n), elements(new
      value_type[capacity]) {
00051         std::fill(elements, elements + size, value);
00052     }
00053
00059     Vector(const Vector &rhs) : size(rhs.size), capacity(rhs.capacity), elements(new
      value_type[capacity]) {
00060         std::copy(rhs.elements, rhs.elements + size, elements);
00061     }
00062
00066     ~Vector() {
00067         delete[] elements;
00068     }
00069
00075     [[nodiscard]] size_type Size() const {
00076         return size;
00077     }
00078
00084     [[nodiscard]] size_type Capacity() const {
00085         return capacity;
00086     }
00087
00093     Vector(Vector&& other) noexcept : size(other.size), capacity(other.capacity),
      elements(other.elements) {
00094         other.size = 0;
00095         other.capacity = 0;
00096         other.elements = nullptr;
00097     }
00098
00105     Vector& operator=(Vector other) noexcept {
```

```
00106         Swap(other);
00107         return *this;
00108     }
00109
00115     [[nodiscard]] bool isEmpty() const {
00116         return size == 0;
00117     }
00118
00124     void PushBack(const_reference object) {
00125         if (size == capacity) {
00126             Reserve(capacity == 0 ? 1 : capacity * 2);
00127         }
00128         elements[size++] = object;
00129     }
00130
00134     void PopBack() {
00135         if (size == 0) {
00136             throw std::out_of_range("Out of range");
00137         }
00138         --size;
00139     }
00140
00144     void Clear() {
00145         size = 0;
00146     }
00147
00153     void Resize(size_type n) {
00154         if (n < size) {
00155             size = n;
00156         } else {
00157             Reserve(n);
00158             std::fill(elements + size, elements + n, value_type());
00159             size = n;
00160         }
00161     }
00162
00168     void Reserve(size_type n) {
00169         if (n > capacity) {
00170             T* newElements = new value_type[n];
00171             std::move(elements, elements + size, newElements);
00172             delete[] elements;
00173             elements = newElements;
00174             capacity = n;
00175         }
00176     }
00177
00183     void Swap(Vector &rhs) noexcept {
00184         std::swap(size, rhs.size);
00185         std::swap(capacity, rhs.capacity);
00186         std::swap(elements, rhs.elements);
00187     }
00188
00192     void ShrinkToFit() {
00193         if (size < capacity) {
00194             T* newElements = new value_type[size];
00195             std::move(elements, elements + size, newElements);
00196             delete[] elements;
00197             elements = newElements;
00198             capacity = size;
00199         }
00200     }
00201
00207     void Erase(size_type index) {
00208         if (index >= size) {
00209             throw std::out_of_range("Out of range");
00210         }
00211         std::move(elements + index + 1, elements + size, elements + index);
00212         --size;
00213     }
00214
00221     void Erase(iterator position, iterator last) {
00222         if (position < begin() || last > end() || position > last) {
00223             throw std::out_of_range("Out of range");
00224         }
00225         size_t numElements = last - position;
00226         std::move(last, end(), position);
00227         size -= numElements;
00228     }
00229
00236     void Insert(size_type index, const_reference object) {
00237         if (index > size) {
00238             throw std::out_of_range("Out of range");
00239         }
00240         if (size == capacity) {
00241             Reserve(capacity == 0 ? 1 : capacity * 2);
00242         }
00243         std::move_backward(elements + index, elements + size, elements + size + 1);
```

```
00244         elements[index] = object;
00245         ++size;
00246     }
00247
00255     void Insert(iterator pos, const_iterator first, const_iterator last) {
00256         size_t index = pos - begin();
00257         size_t numNewElements = last - first;
00258
00259         if (size + numNewElements > capacity) {
00260             Reserve((size + numNewElements) * 2);
00261         }
00262
00263         std::move_backward(elements + index, elements + size, elements + size + numNewElements);
00264         std::copy(first, last, elements + index);
00265         size += numNewElements;
00266     }
00267
00274     reference operator[](size_t index) {
00275         if (index >= size) {
00276             throw std::out_of_range("Index out of range");
00277         }
00278         return elements[index];
00279     }
00280
00287     const_reference operator[](size_t index) const {
00288         if (index >= size) {
00289             throw std::out_of_range("Index out of range");
00290         }
00291         return elements[index];
00292     }
00293
00300     reference At(size_type index) {
00301         if (index >= size) {
00302             throw std::out_of_range("Out of range");
00303         }
00304         return elements[index];
00305     }
00306
00312     reference Front() {
00313         if (size == 0) {
00314             throw std::out_of_range("Out of range");
00315         }
00316         return elements[0];
00317     }
00318
00324     reference Back() {
00325         if (size == 0) {
00326             throw std::out_of_range("Out of range");
00327         }
00328         return elements[size - 1];
00329     }
00330
00336     iterator begin() {
00337         return elements;
00338     }
00339
00345     const_iterator begin() const {
00346         return elements;
00347     }
00348
00354     iterator end() {
00355         return elements + size;
00356     }
00357
00363     const_iterator end() const {
00364         return elements + size;
00365     }
00366
00372     void EmplaceBack(T&& object) {
00373         if (size == capacity) {
00374             Reserve(capacity == 0 ? 1 : capacity * 2);
00375         }
00376         elements[size++] = std::forward<T>(object);
00377     }
00378 };
00379
00380 #endif //OOPUZD_VECTORCLASS_H
```

Index