opProject

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 MyVector< T > Class Template Reference

Nuosavo konteinerio klasė

`#include <vector.h>`

Public Member Functions

- int max_size () const

  Maksimalaus galimo vektoriaus dydžio grąžinimas.
- **MyVector** (std::initializer_list< T > init)
- MyVector (const MyVector< T > &vector_)

  Kopijavimo konstruktorius.
- MyVector (MyVector &&vector_) noexcept

  Move konstuktorius.
- MyVector & operator= (const MyVector &vector_)

  Kopijavimo operatorius.
- MyVector & operator= (MyVector &&vector_)

  Move operatorius.
- T & operator[ ] (unsigned int index)

  Nurodyto elemento grąžinimas.
- const T & operator[ ] (unsigned int index) const

  Nurodyto elemento grąžinimas.
- T & front ()

  Pirmo elemeneto grąžinimas.
- const T & front () const

  Pirmo elemeneto grąžinimas.
- T & back ()

  Paskutinio elemeneto grąžinimas.
- const T & back () const

  Paskutinio elemeneto grąžinimas.
- T * data () noexcept

  Rodyklės į duomenis grąžinimas.
- T * begin () noexcept

Rodyklės į pirmąjį elementą grąžinimas.

- const T * begin () const noexcept

  Rodyklės į pirmąjį elementą grąžinimas.
- T * end () noexcept

  Rodyklės į vektoriaus pabaigą grąžinimas.
- const T * end () const noexcept

  Rodyklės į vektoriaus pabaigą grąžinimas.
- T & at (unsigned int index)

  Nurodyto elemento grąžinimas.
- const T & at (unsigned int index) const

  Nurodyto elemento grąžinimas.
- unsigned int size () const

  Vektoriaus dydžio grąžinimas.
- unsigned int capacity () const

  Vektoriaus talpos grąžinimas.
- bool empty () const

  Patikrinimas, ar vektorius tuščias.
- void reserve (unsigned int reserve_)

  Vektoriuje vietos rezervavimas.
- void **shrink_to_fit** ()

  Vektoriaus talpos sumažinimas iki jo dydžio.
- void assign (unsigned int n, const T &value)

  Elemento priskyrimas nurodytam kiekiui.
- void **clear** ()

  Vektoriaus valymas.
- void push_back (const T &value)

  Elemento pridėjimas į vektoriaus pabaigą
- T * insert (unsigned int index, const T &value)

  Elemento pridėjimas į nurodytą poziciją
- T * emplace (unsigned int index, T &&value)

  Elemento pridėjimas į nurodytą poziciją
- T & emplace_back (T &&value)

  Elemento pridėjimas į vektoriaus pabaigą
- void **pop_back** ()

  Elemento šalinimas iš vektoriaus pabaigos.
- void resize (unsigned int newsize_)

  Vektoriaus dydžio nurodymas.
- void swap (MyVector &vector_)

  Vektorių sukeitimas vietomis.
- T * erase (unsigned int index)

  Nurodyto elemento išmetimas.
- T * erase (T *begin_, T *end_)

  Elementų išmetimas tarp nurodytų elementų

4.1.1 Detailed Description

template<**typename T**>
class MyVector< **T** >

Nuosavo konteinerio klasė

4.1.2 Constructor & Destructor Documentation

4.1.2.1 MyVector() [1/2]

```
template<typename T >
MyVector< T >::MyVector (
            const MyVector< T > & vector_ )  [inline]
```

Kopijavimo konstruktorius.

Parameters

*vector_*	Vektorius, iš kurio kopijuojami duomenys

4.1.2.2 MyVector() [2/2]

```
template<typename T >
MyVector< T >::MyVector (
            MyVector< T > && vector_ )  [inline], [noexcept]
```

Move konstuktorius.

Parameters

*vector_*	Vektorius, iš kurio kopijuojami duomenys

4.1.3 Member Function Documentation

4.1.3.1 assign()

```
template<typename T >
void MyVector< T >::assign (
            unsigned int n,
            const T & value )  [inline]
```

Elemento priskyrimas nurodytam kiekiui.

Parameters

n	Kiekis
value	Elementas

4.1.3.2 at() [1/2]

```
template<typename T >
```

```
T & MyVector< T >::at (
            unsigned int index )  [inline]
```

Nurodyto elemento grąžinimas.

Parameters

index	Elemento indeksas

Returns

Elementas

Exceptions

std::out_of_range	Jei indeksas už ribų

4.1.3.3 at() [2/2]

```
template<typename T >
const T & MyVector< T >::at (
            unsigned int index ) const  [inline]
```

Nurodyto elemento grąžinimas.

Parameters

index	Elemento indeksas

Returns

Elementas

Exceptions

std::out_of_range	Jei indeksas už ribų

4.1.3.4 back() [1/2]

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

Paskutinio elemeneto grąžinimas.

Returns

Paskutinis elementas

4.1.3.5 back() [2/2]

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

Paskutinio elemeneto grąžinimas.

Returns

Paskutinis elementas

4.1.3.6 begin() [1/2]

```
template<typename T >
const T * MyVector< T >::begin ( ) const  [inline], [noexcept]
```

Rodyklės į pirmąjį elementą grąžinimas.

Returns

Rodyklė į pirmąjį elementą

4.1.3.7 begin() [2/2]

```
template<typename T >
T * MyVector< T >::begin ( )  [inline], [noexcept]
```

Rodyklės į pirmąjį elementą grąžinimas.

Returns

Rodyklė į pirmąjį elementą

4.1.3.8 capacity()

```
template<typename T >
unsigned int MyVector< T >::capacity ( ) const  [inline]
```

Vektoriaus talpos grąžinimas.

Returns

Vektoriaus talpa

4.1.3.9 data()

```
template<typename T >
T * MyVector< T >::data ( )  [inline], [noexcept]
```

Rodyklės į duomenis grąžinimas.

Returns

Rodyklė į duomenis

4.1.3.10 emplace()

```
template<typename T >
T * MyVector< T >::emplace (
            unsigned int index,
            T && value )  [inline]
```

Elemento pridėjimas į nurodytą poziciją

Parameters

index	Indeksas
value	Elementas

4.1.3.11 emplace_back()

```
template<typename T >
T & MyVector< T >::emplace_back (
            T && value )  [inline]
```

Elemento pridėjimas į vektoriaus pabaigą

Parameters

value	Elementas

4.1.3.12 empty()

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

Patikrinimas, ar vektorius tuščias.

Return values

TRUE	vektorius tuščias
FALSE	vektorius netuščias

4.1.3.13 end() [1/2]

```
template<typename T >
const T * MyVector< T >::end ( ) const  [inline], [noexcept]
```

Rodyklės į vektoriaus pabaigą grąžinimas.

Returns

Rodyklė į vektoriaus pabaigą

4.1.3.14 end() [2/2]

```
template<typename T >
T * MyVector< T >::end ( )  [inline], [noexcept]
```

Rodyklės į vektoriaus pabaigą grąžinimas.

Returns

Rodyklė į vektoriaus pabaigą

4.1.3.15 erase() [1/2]

```
template<typename T >
T * MyVector< T >::erase (
            T * begin_,
            T * end_ )  [inline]
```

Elementų išmetimas tarp nurodytų elementų

Parameters

*begin_*	Pirmas elementas
*end_*	Paskutinis elementas

Returns

Rodyklė į paskutinį elementą

4.1.3.16 erase() [2/2]

```
template<typename T >
T * MyVector< T >::erase (
            unsigned int index )  [inline]
```

Nurodyto elemento išmetimas.

Parameters

index	Elementas

4.1.3.17 front() [1/2]

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

Pirmo elemeneto grąžinimas.

Returns

Pirmas elementas

4.1.3.18 front() [2/2]

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

Pirmo elemeneto grąžinimas.

Returns

Pirmas elementas

4.1.3.19 insert()

```
template<typename T >
T * MyVector< T >::insert (
            unsigned int index,
            const T & value )  [inline]
```

Elemento pridėjimas į nurodytą poziciją

Parameters

index	Indeksas
value	Elementas

4.1.3.20 max_size()

```
template<typename T >
int MyVector< T >::max_size ( ) const  [inline]
```

Maksimalaus galimo vektoriaus dydžio grąžinimas.

Returns

Maksimalus vektoriaus dydis

4.1.3.21 operator=() [1/2]

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

Kopijavimo operatorius.

Parameters

*vector_*	Vektorius, iš kurio kopijuojami duomenys

4.1.3.22 operator=() [2/2]

```
template<typename T >
MyVector & MyVector< T >::operator= (
            MyVector< T > && vector_ )  [inline]
```

Move operatorius.

Parameters

*vector_*	Vektorius, iš kurio perkeliami duomenys

4.1.3.23 operator[]() [1/2]

```
template<typename T >
T & MyVector< T >::operator[] (
            unsigned int index )  [inline]
```

Nurodyto elemento grąžinimas.

Parameters

index	Elemento indeksas

4.1.3.24 operator[]() [2/2]

```
template<typename T >
const T & MyVector< T >::operator[] (
            unsigned int index ) const  [inline]
```

Nurodyto elemento grąžinimas.

Parameters

index	Elemento indeksas

4.1.3.25 push_back()

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

Elemento pridėjimas į vektoriaus pabaigą

Parameters

value	Pridėdamas elementas

4.1.3.26 reserve()

```
template<typename T >
void MyVector< T >::reserve (
            unsigned int reserve_ )  [inline]
```

Vektoriuje vietos rezervavimas.

Parameters

*reserve_*	Nauja vektoriaus talpa

Exceptions

std::length_error	Jei vektorius neturi tiek vietos

4.1.3.27 resize()

```
template<typename T >
void MyVector< T >::resize (
            unsigned int newsize_ )  [inline]
```

Vektoriaus dydžio nurodymas.

Parameters

*newsize_*	Dydis

4.1.3.28 size()

```
template<typename T >
unsigned int MyVector< T >::size ( ) const  [inline]
```

Vektoriaus dydžio grąžinimas.

Returns

Vektoriaus dydis

4.1.3.29 swap()

```
template<typename T >
void MyVector< T >::swap (
            MyVector< T > & vector_ )  [inline]
```

Vektorių sukeitimas vietomis.

Parameters

*vector_*	Kitas vektorius

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

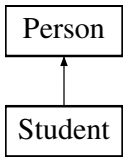
- code/vector.h

4.2 Person Class Reference

Abstrakti žmogaus klasė

`#include <person.h>`

Inheritance diagram for Person:



Public Member Functions

- std::string get_Name () const
  Vardo gavimas.
- std::string get_Surname () const
  Pavardės gavimas.
- void set_Name (std::string name)
  Vardo nustatymas.
- void set_Surname (std::string surname)
  Pavardės nustatymas.

Protected Member Functions

- **Person** ()

  Default konstruktorius.
- Person (const std::string &name, const std::string &surname)

  Žmogaus klasės konstruktorius.
- virtual ~**Person** ()

  Žmogus klasės destruktorius.

Protected Attributes

- std::string **name_**
- std::string **surname_**

4.2.1 Detailed Description

Abstrakti žmogaus klasė

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Person()

```
Person::Person (
            const std::string & name,
            const std::string & surname )  [inline], [protected]
```

Žmogaus klasės konstruktorius.

Parameters

name	Vardas
surname	Pavardė

4.2.3 Member Function Documentation

4.2.3.1 get_Name()

```
std::string Person::get_Name ( ) const  [inline]
```

Vardo gavimas.

Returns

Vardas

4.2.3.2 get_Surname()

```
std::string Person::get_Surname ( ) const  [inline]
```

Pavardės gavimas.

Returns

Pavardė

4.2.3.3 set_Name()

```
void Person::set_Name (
            std::string name )  [inline]
```

Vardo nustatymas.

Parameters

name	

4.2.3.4 set_Surname()

```
void Person::set_Surname (
            std::string surname )  [inline]
```

Pavardės nustatymas.

Parameters

surname	

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

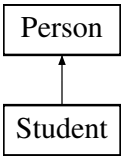
- code/person.h

4.3 Student Class Reference

Studento klasė

`#include <func.h>`

Inheritance diagram for Student:



Public Member Functions

- **Student** ()

  Default konstruktorius.
- Student (const std::string name, const std::string surname)

  Studento klasės konstruktorius.
- MyVector< int > get_HwRes () const

  Namų darbų gavimas.
- double get_exRes () const

  Egzamino rezultato gavimas.
- double get_Avg () const

  Galutinio pagal vidurkį gavimas.
- double get_Med () const

  Galutinio pagal medianą gavimas.
- bool hwRes_Empty () const

  Patikrinimas ar namų darbų vektorius yra tuščias.
- int hwRes_Size () const

  Namų darbų vektoriaus didžio gavimas.
- void **hw_Sort** ()

  Namų darbų rūšiavimas didėjimo tvarka.
- int hw_Sum ()

  Namų darbų sumos apskaičiavimas.
- int hw_Last ()

  Paskutinio namų darbų elemento gavimas.
- ~**Student** ()

  Studento klasės destruktorius.
- void set_ExRes (double exRes)

  Egzamino rezultato priskyrimas.
- void set_Avg (double avg)

  Galutnio pagal vidurkį priskyrimas.
- void set_Med (double med)

  Galutinio pagal medianą priskyrimas.
- void set_Hw (int hw)

  Pažymio pridėjimas į namų darbų vektoriu.
- void **del_LastHw** ()

  Paskutinio namų darbų elemento ištrynimas.
- void **clear_Hw** ()

  Namų darbų vektoriaus išvalymas.
- void **clear_All** ()

  Viso studento išvalymas.
- double Average ()

  Galutinio pagal vidurkį skaičiavimas.
- double Median ()

  Galutinio pagal medianą skaičiavimas.
- Student (const Student &Student_)

  Kopijavimo konstruktorius.
- Student (Student &&Student_) noexcept

  Perkėlimo konstruktorius.
- Student & operator= (const Student &Student_)

  Kopijavimo priskyrimo operatorius.
- Student & operator= (Student &&Student_) noexcept

  Perkėlimo priskyrimo operatorius.

Public Member Functions inherited from Person

- std::string get_Name () const
  Vardo gavimas.
- std::string get_Surname () const
  Pavardės gavimas.
- void set_Name (std::string name)
  Vardo nustatymas.
- void set_Surname (std::string surname)
  Pavardės nustatymas.

Friends

- std::istringstream & operator>> (std::istringstream &input, Student &Student_)
  Įvesties operatorius darbui su failais.
- std::istream & operator>> (std::istream &input, Student &Student_)
  Įvesties operatorius darbui su vartotoju per konsolę
- std::ostream & operator<< (std::ostream &output, const Student &Student_)
  Išvesties operatorius į konsolę
- std::ofstream & operator<< (std::ofstream &output, const Student &Student_)
  Išvesties operatorius į failą

Additional Inherited Members

Protected Member Functions inherited from Person

- **Person** ()
  Default konstruktorius.
- Person (const std::string &name, const std::string &surname)
  Žmogaus klasės konstruktorius.
- virtual ~**Person** ()
  Žmogus klasės destruktorius.

Protected Attributes inherited from Person

- std::string **name_**
- std::string **surname_**

4.3.1 Detailed Description

Studento klasė

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Student() [1/3]

```
Student::Student (
            const std::string name,
            const std::string surname )
```

Studento klasės konstruktorius.

Parameters

name	Studento vardas
surname	Studento pavardė

4.3.2.2 Student() [2/3]

```
Student::Student (
            const Student & Student_ )
```

Kopijavimo konstruktorius.

Parameters

*Student_*	Kopijuojamas objektas

4.3.2.3 Student() [3/3]

```
Student::Student (
            Student && Student_ )  [noexcept]
```

Perkėlimo konstruktorius.

Parameters

*Student_*	Perkialamas objektas

4.3.3 Member Function Documentation

4.3.3.1 Average()

```
double Student::Average ( )
```

Galutinio pagal vidurkį skaičiavimas.

Returns

Galutinis pagal vidurkį

4.3.3.2 get_Avg()

```
double Student::get_Avg ( ) const  [inline]
```

Galutinio pagal vidurkį gavimas.

Returns

Galutinis pagal vidurkį

4.3.3.3 get_exRes()

```
double Student::get_exRes ( ) const  [inline]
```

Egzamino rezultato gavimas.

Returns

Egzamino rezultatas

4.3.3.4 get_HwRes()

```
MyVector< int > Student::get_HwRes ( ) const  [inline]
```

Namų darbų gavimas.

Returns

Namų darbų vektorius

4.3.3.5 get_Med()

```
double Student::get_Med ( ) const  [inline]
```

Galutinio pagal medianą gavimas.

Returns

Galutinis pagal medianą

4.3.3.6 hw_Last()

```
int Student::hw_Last ( )  [inline]
```

Paskutinio namų darbų elemento gavimas.

Returns

Paskutinis namų darbų elementas

4.3.3.7 hw_Sum()

```
int Student::hw_Sum ( )  [inline]
```

Namų darbų sumos apskaičiavimas.

Returns

Namų darbų elementų suma

4.3.3.8 hwRes_Empty()

```
bool Student::hwRes_Empty ( ) const  [inline]
```

Patikrinimas ar namų darbų vektorius yra tuščias.

Return values

TRUE	vektorius tuščias
FALSE	vektorius turi elementų

4.3.3.9 hwRes_Size()

```
int Student::hwRes_Size ( ) const  [inline]
```

Namų darbų vektoriaus didžio gavimas.

Returns

Namų darbų vektoriaus dydis

4.3.3.10 Median()

```
double Student::Median ( )
```

Galutinio pagal medianą skaičiavimas.

Returns

Galutinis pagal mediana

4.3.3.11 operator=() [1/2]

```
Student & Student::operator= (
            const Student & Student_ )
```

Kopijavimo priskyrimo operatorius.

Parameters

*Student_*	Kopijuojamas objektas

4.3.3.12 operator=() [2/2]

```
Student & Student::operator= (
            Student && Student_ )  [noexcept]
```

Perkėlimo priskyrimo operatorius.

Parameters

Student↩ _	Perkeliamas objektas

4.3.3.13 set_Avg()

```
void Student::set_Avg (
            double avg )  [inline]
```

Galutnio pagal vidurkį priskyrimas.

Parameters

avg	Vidurkis

4.3.3.14 set_ExRes()

```
void Student::set_ExRes (
            double exRes )  [inline]
```

Egzamino rezultato priskyrimas.

Parameters

exRes	Egzamino rezultatas

4.3.3.15 set_Hw()

```
void Student::set_Hw (
            int hw )  [inline]
```

Pažymio pridėjimas į namų darbų vektoriu.

Parameters

hw	Pažymys

4.3.3.16 set_Med()

```
void Student::set_Med (
            double med )  [inline]
```

Galutinio pagal medianą priskyrimas.

Parameters

med	Mediana

4.3.4 Friends And Related Symbol Documentation

4.3.4.1 operator<< [1/2]

```
std::ofstream & operator<< (
            std::ofstream & output,
            const Student & Student_ )  [friend]
```

Išvesties operatorius į failą

Parameters

output	Išvesties objektas
*Student_*	Išvedamas objektas

Returns

Išvesties ofstream objektas

4.3.4.2 operator<< [2/2]

```
std::ostream & operator<< (
            std::ostream & output,
            const Student & Student_ )  [friend]
```

Išvesties operatorius į konsolę

Parameters

output	Išvesties objektas
*Student_*	Išvedamas objektas

Returns

Išvesties ostream objektas

4.3.4.3 operator>> [1/2]

```
std::istream & operator>> (
            std::istream & input,
            Student & Student_ )  [friend]
```

Įvesties operatorius darbui su vartotoju per konsolę

Parameters

input	Įvesties objektas
*Student_*	Objektas, į kurį skaitomi duomenys

Returns

Įvesties istream objektas

4.3.4.4 operator>> [2/2]

```
std::istringstream & operator>> (
            std::istringstream & input,
            Student & Student_ )  [friend]
```

Įvesties operatorius darbui su failais.

Parameters

input	Įvesties objektas
*Student_*	Objektas, į kurį skaitomi duomenys

Returns

Įvesties istringstream objektas

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

- code/func.h
- code/func.cpp

Chapter 5

File Documentation

5.1 func.h

```
00001 #ifndef func_h
00002 #define func_h
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <sstream>
00007 #include <iomanip>
00008 #include <algorithm>
00009 #include <vector>
00010 #include <random>
00011 #include <string>
00012 #include <chrono>
00013 #include "person.h"
00014 #include "vector.h"
00015
00018 class Student : public Person
00019 {
00020 private:
00021     double exRes_;
00022     MyVector<int> hwRes_;
00023     double avg_, med_;
00024
00025 public:
00027     Student();
00031     Student(const std::string name, const std::string surname);
00034     MyVector<int> get_HwRes() const { return hwRes_; }
00037     double get_exRes() const { return exRes_; }
00040     double get_Avg() const { return avg_; }
00043     double get_Med() const { return med_; }
00047     bool hwRes_Empty() const { return hwRes_.empty(); }
00050     int hwRes_Size() const { return hwRes_.size(); }
00052     void hw_Sort() { std::sort(hwRes_.begin(), hwRes_.end()); }
00055     int hw_Sum() { return std::accumulate(hwRes_.begin(), hwRes_.end(), 0); }
00058     int hw_Last() { return hwRes_.back(); }
00060     ~Student();
00061
00064     void set_ExRes(double exRes) { this->exRes_ = exRes; }
00067     void set_Avg(double avg) { this->avg_ = avg; }
00070     void set_Med(double med) { this->med_ = med; }
00073     void set_Hw(int hw) { this->hwRes_.push_back(hw); }
00075     void del_LastHw() { this->hwRes_.pop_back(); }
00077     void clear_Hw() { this->hwRes_.clear(); }
00079     void clear_All()
00080     {
00081         this->name_.clear();
00082         this->surname_.clear();
00083         this->hwRes_.clear();
00084         this->exRes_ = 0;
00085         this->avg_ = 0.0;
00086         this->med_ = 0.0;
00087     }
00090     double Average();
00093     double Median();
00094
00097     Student(const Student &Student_);
00100     Student(Student &&Student_) noexcept;
00103     Student &operator=(const Student &Student_);
00106     Student &operator=(Student &&Student_) noexcept;
```

```
00111     friend std::istringstream &operator»(std::istringstream &input, Student &Student_);
00116     friend std::istream &operator»(std::istream &input, Student &Student_);
00121     friend std::ostream &operator«(std::ostream &output, const Student &Student_);
00126     friend std::ofstream &operator«(std::ofstream &output, const Student &Student_);
00127 };
00128
00134 bool compareName(const Student &a, const Student &b);
00140 bool compareSurname(const Student &a, const Student &b);
00146 bool compareAvg(const Student &a, const Student &b);
00152 bool compareMed(const Student &a, const Student &b);
00155 int RandGrade();
00157 void CinError();
00161 void GenFile(int size, int hw);
00164 void ReadFile(MyVector<Student> &studVector);
00169 void Selection(MyVector<Student> &studVector, MyVector<Student> &best, int choice);
00172 void Results(MyVector<Student> studVector);
00175 void ReadUser(MyVector<Student> &studVector);
00180 void GenUser(MyVector<Student> &studVector, int size, int hw);
00181 void VectorTest();
00182
00183 #endif
```

5.2 person.h

```
00001 #ifndef PERSON_H
00002 #define PERSON_H
00003
00004 #include <string>
00005
00008 class Person
00009 {
00010 protected:
00011     std::string name_, surname_;
00013     Person() : name_(""), surname_(""){};
00017     Person(const std::string &name, const std::string &surname) : name_(name), surname_(surname){};
00019     virtual ~Person(){};
00020
00021 public:
00024     inline std::string get_Name() const { return name_; }
00027     inline std::string get_Surname() const { return surname_; }
00030     void set_Name(std::string name) { this->name_ = name; }
00033     void set_Surname(std::string surname) { this->surname_ = surname; }
00034 };
00035
00036 #endif
```

5.3 vector.h

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <iostream>
00005 #include <initializer_list>
00006 #include <limits>
00007 #include <algorithm>
00008 #include <stdexcept>
00009
00010 template <typename T>
00013 class MyVector
00014 {
00015 private:
00016     size_t size_;
00017     size_t capacity_;
00018     T *data_;
00019
00020 public:
00023     int max_size() const { return std::numeric_limits<unsigned int>::max() / sizeof(T); }
00024     MyVector() : size_(0), capacity_(0), data_(new T[capacity_]) {}
00025     MyVector(std::initializer_list<T> init) : size_(init.size()), capacity_(init.size()) {
      std::copy(init.begin(), init.end(), data_); }
00026     ~MyVector() { delete[] data_; }
00027
00030     MyVector(const MyVector<T> &vector_) : size_(vector_.size_), capacity_(vector_.capacity_),
      data_(new T[capacity_])
00031     {
00032         for (unsigned int i = 0; i < size_; i++)
00033             data_[i] = vector_.data_[i];
00034     }
00035
```

```
00038     MyVector(MyVector &&vector_) noexcept : size_(vector_.size_), capacity_(vector_.capacity_),
      data_(vector_.data_)
00039     {
00040         vector_.size_ = 0;
00041         vector_.capacity_ = 0;
00042         vector_.data_ = nullptr;
00043     }
00044
00047     MyVector &operator=(const MyVector &vector_)
00048     {
00049         if (this == &vector_)
00050             return *this;
00051         delete[] data_;
00052         size_ = vector_.size_;
00053         capacity_ = vector_.capacity_;
00054         data_ = new T[capacity_];
00055         for (unsigned int i = 0; i < size_; i++)
00056             data_[i] = vector_.data_[i];
00057         return *this;
00058     }
00059
00062     MyVector &operator=(MyVector &&vector_)
00063     {
00064         if (this == &vector_)
00065             return *this;
00066         delete[] data_;
00067         size_ = vector_.size_;
00068         capacity_ = vector_.capacity_;
00069         data_ = vector_.data_;
00070         vector_.size_ = 0;
00071         vector_.capacity_ = 0;
00072         vector_.data_ = nullptr;
00073         return *this;
00074     }
00075
00076     //Element access
00077
00080     T &operator[](unsigned int index) { return data_[index]; }
00083     const T &operator[](unsigned int index) const { return data_[index]; }
00086     T &front() { return data_[0]; }
00089     const T &front() const { return data_[0]; }
00092     T &back() { return data_[size_ - 1]; }
00095     const T &back() const { return data_[size_ - 1]; }
00098     T *data() noexcept { return data_; }
00099
00100     //Iterators
00101
00104     T *begin() noexcept { return data_; }
00107     const T *begin() const noexcept { return data_; }
00110     T *end() noexcept { return data_ + size_; }
00113     const T *end() const noexcept { return data_ + size_; }
00118     T &at(unsigned int index)
00119     {
00120         if (index >= size_)
00121             throw std::out_of_range("Indeksas uz ribu!");
00122         return data_[index];
00123     }
00128     const T &at(unsigned int index) const
00129     {
00130         if (index >= size_)
00131             throw std::out_of_range("Indeksas uz ribu!");
00132         return data_[index];
00133     }
00134
00135     //Capacity
00136
00139     unsigned int size() const { return size_; }
00142     unsigned int capacity() const { return capacity_; }
00146     bool empty() const { return size_ == 0; }
00147
00151     void reserve(unsigned int reserve_)
00152     {
00153         if (reserve_ <= capacity_)
00154             return;
00155         if (reserve_ > max_size())
00156             throw std::length_error("Vektorius neturi tiek vietos!");
00157         T *newdata_ = new T[reserve_];
00158         for (unsigned int i = 0; i < size_; i++)
00159             newdata_[i] = data_[i];
00160         delete[] data_;
00161         data_ = newdata_;
00162         capacity_ = reserve_;
00163     }
00164
00166     void shrink_to_fit()
00167     {
00168         if (size_ == capacity_)
```

```
00169             return;
00170         T *newdata_ = new T[size_];
00171         for (unsigned int i = 0; i < size_; i++)
00172             newdata_[i] = data_[i];
00173         delete[] data_;
00174         data_ = newdata_;
00175         capacity_ = size_;
00176     }
00177
00181     void assign(unsigned int n, const T &value)
00182     {
00183         if (n > capacity_)
00184             reserve(n);
00185         for (unsigned int i = 0; i < n; i++)
00186             data_[i] = value;
00187         size_ = n;
00188     }
00189
00190     //Modifiers
00191
00193     void clear() { size_ = 0; }
00194
00197     void push_back(const T &value)
00198     {
00199         if (size_ >= capacity_)
00200             reserve(capacity_ == 0 ? 1 : size_ * 2);
00201         data_[size_++] = value;
00202     }
00203
00207     T *insert(unsigned int index, const T &value)
00208     {
00209         if (size_ >= capacity_)
00210             reserve(capacity_ == 0 ? 1 : size_ * 2);
00211         for (unsigned int i = size_; i > index; i--)
00212             data_[i] = data_[i - 1];
00213         data_[index] = value;
00214         size_++;
00215         return &data_[index];
00216     }
00217
00221     T *emplace(unsigned int index, T &&value)
00222     {
00223         if (size_ == capacity_)
00224             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00225         for (unsigned int i = size_; i > index; i--)
00226             data_[i] = data_[i - 1];
00227         data_[index] = std::move(value);
00228         size_++;
00229         return &data_[index];
00230     }
00231
00234     T &emplace_back(T &&value)
00235     {
00236         if (size_ >= capacity_)
00237             reserve(capacity_ == 0 ? 1 : size_ * 2);
00238         data_[size_++] = std::move(value);
00239         return data_[size_ - 1];
00240     }
00241
00243     void pop_back()
00244     {
00245         if (size_ > 0)
00246         {
00247             --size_;
00248         }
00249     }
00250
00253     void resize(unsigned int newsize_)
00254     {
00255         if (newsize_ > capacity_)
00256             reserve(newsize_);
00257         for (unsigned int i = size_; i < newsize_; i++)
00258             data_[i] = T();
00259         size_ = newsize_;
00260     }
00261
00264     void swap(MyVector &vector_)
00265     {
00266         std::swap(size_, vector_.size_);
00267         std::swap(capacity_, vector_.capacity_);
00268         std::swap(data_, vector_.data_);
00269     }
00270
00273     T *erase(unsigned int index)
00274     {
00275         for (unsigned int i = index; i < size_ - 1; i++)
00276             data_[i] = data_[i + 1];
```

```
00277         size_--;
00278         return &data_[index];
00279     }
00280 
00285     T *erase(T *begin_, T *end_)
00286     {
00287         if (begin_ >= data_ && end_ <= data_ + size_)
00288         {
00289             size_t deletedVal = end_ - begin_;
00290             size_t movedVal = data_ + size_ - end_;
00291 
00292             for (size_t i = 0; i < movedVal; i++)
00293                 *(begin_ + i) = *(end_ + i);
00294 
00295             size_ -= deletedVal;
00296 
00297             if (end_ == data_ + size_)
00298                 return end_;
00299         }
00300         return end_;
00301     }
00302 };
00303 
00304 #endif
```

Index