Objektinio programavimo projektas

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 files with brief descriptions:

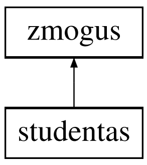
Chapter 4

Class Documentation

4.1 studentas Class Reference

`#include <studentas.h>`

Inheritance diagram for studentas:



Public Member Functions

- studentas ()
- studentas (const string &v, const string &p, const Vector< int > &nd, int e, double g)
- ∼studentas ()
- studentas (const studentas &kit)
- studentas & operator= (const studentas &kit)
- studentas (studentas &&kit) noexcept
- studentas & operator= (studentas &&kit) noexcept
- string getVardas () const
- string getPavarde () const
- Vector< int > getNdrez () const
- int getErez () const
- double getGbalas () const
- void setVardas (const string &v)
- void setPavarde (const string &p)
- void setNdrez (const Vector< int > &nd)
- void setErez (int e)
- void setGbalas (double g)
- void sortNdrez ()

Public Member Functions inherited from zmogus

- virtual ∼zmogus ()
- virtual void setVardas (string v)
- virtual void setPavarde (string p)

Public Attributes

- string line
- char budas

Private Attributes

- Vector< int > ndrez
- int erez
- double gbalas

Friends

- std::istream & operator>> (std::istream &in, studentas &kit)
- std::ostream & operator<< (std::ostream &out, const studentas &kit)

Additional Inherited Members

Protected Attributes inherited from zmogus

- string vardas
- string pavarde

4.1.1 Constructor & Destructor Documentation

4.1.1.1 studentas() [1/4]

```
studentas::studentas ( )
```

4.1.1.2 studentas() [2/4]

```
studentas::studentas (
            const string & v,
            const string & p,
            const Vector< int > & nd,
            int e,
            double g )
```

4.1.1.3 ∼studentas()

```
studentas::∼studentas ( )
```

4.1.1.4 studentas() [3/4]

```
studentas::studentas (
            const studentas & kit )
```

4.1.1.5 studentas() [4/4]

```
studentas::studentas (
            studentas && kit )  [noexcept]
```

4.1.2 Member Function Documentation

4.1.2.1 getErez()

```
int studentas::getErez ( ) const  [inline]
```

4.1.2.2 getGbalas()

```
double studentas::getGbalas ( ) const  [inline]
```

4.1.2.3 getNdrez()

```
Vector< int > studentas::getNdrez ( ) const  [inline]
```

4.1.2.4 getPavarde()

```
string studentas::getPavarde ( ) const  [inline], [virtual]
```

Implements zmogus.

4.1.2.5 getVardas()

```
string studentas::getVardas ( ) const  [inline], [virtual]
```

Implements zmogus.

4.1.2.6 operator=() [1/2]

```
studentas & studentas::operator= (
            const studentas & kit )
```

4.1.2.7 operator=() [2/2]

```
studentas & studentas::operator= (
             studentas && kit )  [noexcept]
```

4.1.2.8 setErez()

```
void studentas::setErez (
             int e )  [inline]
```

4.1.2.9 setGbalas()

```
void studentas::setGbalas (
             double g )  [inline]
```

4.1.2.10 setNdrez()

```
void studentas::setNdrez (
             const Vector< int > & nd )  [inline]
```

4.1.2.11 setPavarde()

```
void studentas::setPavarde (
             const string & p )  [inline]
```

4.1.2.12 setVardas()

```
void studentas::setVardas (
             const string & v )  [inline]
```

4.1.2.13 sortNdrez()

```
void studentas::sortNdrez ( )  [inline]
```

4.1.3 Friends And Related Symbol Documentation

4.1.3.1 operator<<

```
std::ostream & operator<< (
             std::ostream & out,
             const studentas & kit )  [friend]
```

4.1.3.2 operator>>

```
std::istream & operator>> (
             std::istream & in,
             studentas & kit )  [friend]
```

4.1.4 Member Data Documentation

4.1.4.1 budas

```
char studentas::budas
```

4.1.4.2 erez

```
int studentas::erez  [private]
```

4.1.4.3 gbalas

```
double studentas::gbalas  [private]
```

4.1.4.4 line

```
string studentas::line
```

4.1.4.5 ndrez

```
Vector<int> studentas::ndrez  [private]
```

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

- studentas.h
- studentas.cpp

4.2 Vector< T > Class Template Reference

```
#include <vector.h>
```

Public Types

- using value_type = T
- using reference = T&
- using const_reference = const T&
- using iterator = T∗
- using const_iterator = const T∗

Public Member Functions

- Vector ()
- Vector (std::initializer_list< T > init)
- Vector (const Vector &other)
- Vector (Vector &&other) noexcept
- ∼Vector ()
- Vector & operator= (const Vector &other)
- Vector & operator= (Vector &&other) noexcept
- size_t size () const
- size_t max_size () const
- size_t capacity () const
- bool empty () const
- void reserve (size_t new_capacity)
- void resize (size_t new_size, const T &value=T())
- void shrink_to_fit ()
- reference operator[ ] (size_t index)
- const_reference operator[ ] (size_t index) const
- reference at (size_t index)
- const_reference at (size_t index) const
- reference front ()
- const_reference front () const
- reference back ()
- const_reference back () const
- void push_back (const T &value)
- void pop_back ()
- iterator erase (iterator position)
- iterator erase (iterator first, iterator last)
- void clear ()
- iterator begin ()
- const_iterator begin () const
- iterator end ()
- const_iterator end () const
- Vector (T ∗data, const size_t &capacity_, const size_t &length)
- bool operator== (const Vector &other) const
- size_t getReallocationCount () const

Private Attributes

- T ∗ data = nullptr
- size_t capacity_ = 0
- size_t length = 0
- size_t reallocations

4.2.1 Member Typedef Documentation

4.2.1.1 const_iterator

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

4.2.1.2 const_reference

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

4.2.1.3 iterator

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

4.2.1.4 reference

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

4.2.1.5 value_type

```
template<typename T >
using Vector< T >::value_type = T
```

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Vector() [1/5]

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

4.2.2.2 Vector() [2/5]

```
template<typename T >
Vector< T >::Vector (
             std::initializer_list< T > init )  [inline]
```

4.2.2.3 Vector() [3/5]

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

4.2.2.4 Vector() [4/5]

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

4.2.2.5 ∼Vector()

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

4.2.2.6 Vector() [5/5]

```
template<typename T >
Vector< T >::Vector (
            T * data,
            const size_t & capacity_,
            const size_t & length )  [inline]
```

4.2.3 Member Function Documentation

4.2.3.1 at() [1/2]

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

4.2.3.2 at() [2/2]

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

4.2.3.3 back() [1/2]

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

4.2.3.4 back() [2/2]

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

4.2.3.5 begin() [1/2]

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

4.2.3.6 begin() [2/2]

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

4.2.3.7 capacity()

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

4.2.3.8 clear()

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

4.2.3.9 empty()

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

4.2.3.10 end() [1/2]

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

4.2.3.11 end() [2/2]

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

4.2.3.12 erase() [1/2]

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

4.2.3.13 erase() [2/2]

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

4.2.3.14 front() [1/2]

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

4.2.3.15 front() [2/2]

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

4.2.3.16 getReallocationCount()

```
template<typename T >
size_t Vector< T >::getReallocationCount ( ) const  [inline]
```

4.2.3.17 max_size()

```
template<typename T >
size_t Vector< T >::max_size ( ) const  [inline]
```

4.2.3.18 operator=() [1/2]

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

4.2.3.19 operator=() [2/2]

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

4.2.3.20 operator==()

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

4.2.3.21 operator[]() [1/2]

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

4.2.3.22 operator[]() [2/2]

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

4.2.3.23 pop_back()

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

4.2.3.24 push_back()

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

4.2.3.25 reserve()

```
template<typename T >
void Vector< T >::reserve (
             size_t new_capacity )  [inline]
```

4.2.3.26 resize()

```
template<typename T >
void Vector< T >::resize (
             size_t new_size,
             const T & value = T() )  [inline]
```

4.2.3.27 shrink_to_fit()

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

4.2.3.28 size()

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

4.2.4 Member Data Documentation

4.2.4.1 capacity_

```
template<typename T >
size_t Vector< T >::capacity_ = 0  [private]
```

4.2.4.2 data

```
template<typename T >
T* Vector< T >::data = nullptr  [private]
```

4.2.4.3 length

```
template<typename T >
size_t Vector< T >::length = 0  [private]
```

4.2.4.4 reallocations

```
template<typename T >
size_t Vector< T >::reallocations  [private]
```

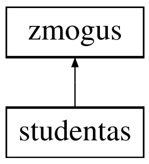
The documentation for this class was generated from the following file:

- vector.h

4.3 zmogus Class Reference

```
#include <zmogus.h>
```

Inheritance diagram for zmogus:



Public Member Functions

- virtual ∼zmogus ()
- virtual string getVardas () const =0
- virtual void setVardas (string v)
- virtual string getPavarde () const =0
- virtual void setPavarde (string p)

Protected Attributes

- string vardas
- string pavarde

4.3.1 Constructor & Destructor Documentation

4.3.1.1 ∼zmogus()

```
virtual zmogus::∼zmogus ( )  [inline], [virtual]
```

4.3.2 Member Function Documentation

4.3.2.1 getPavarde()

```
virtual string zmogus::getPavarde ( ) const  [pure virtual]
```

Implemented in studentas.

4.3.2.2 getVardas()

```
virtual string zmogus::getVardas ( ) const  [pure virtual]
```

Implemented in studentas.

4.3.2.3 setPavarde()

```
virtual void zmogus::setPavarde (
             string p )  [inline], [virtual]
```

4.3.2.4 setVardas()

```
virtual void zmogus::setVardas (
             string v )  [inline], [virtual]
```

4.3.3 Member Data Documentation

4.3.3.1 pavarde

```
string zmogus::pavarde  [protected]
```

4.3.3.2 vardas

```
string zmogus::vardas  [protected]
```

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

- zmogus.h

Chapter 5

File Documentation

5.1 errorfinder.cpp File Reference

```
#include "errorfinder.h"
#include "studentas.h"
```

Functions

- int ivedbudpatikra ()
- char budaspatikra ()
- char dskaitpatikra ()
- int studskpatikra ()
- char isvedbudpatikra ()
- int erezpatikra ()
- char rikbudpatikra ()
- int pazymiopatikra ()
- char skirstymopatikra ()
- char fgeneravimopatikra ()
- int skirststratpat ()

5.1.1 Function Documentation

5.1.1.1 budaspatikra()

```
char budaspatikra ( )
```

5.1.1.2 dskaitpatikra()

```
char dskaitpatikra ( )
```

5.1.1.3 erezpatikra()

```
int erezpatikra ( )
```

5.1.1.4 fgeneravimopatikra()

```
char fgeneravimopatikra ( )
```

5.1.1.5 isvedbudpatikra()

```
char isvedbudpatikra ( )
```

5.1.1.6 ivedbudpatikra()

```
int ivedbudpatikra ( )
```

5.1.1.7 pazymiopatikra()

```
int pazymiopatikra ( )
```

5.1.1.8 rikbudpatikra()

```
char rikbudpatikra ( )
```

5.1.1.9 skirststratpat()

```
int skirststratpat ( )
```

5.1.1.10 skirstymopatikra()

```
char skirstymopatikra ( )
```

5.1.1.11 studskpatikra()

```
int studskpatikra ( )
```

5.2 errorfinder.h File Reference

```
#include <iostream>
#include <limits>
```

Functions

- int ivedbudpatikra ()
- char budaspatikra ()
- char dskaitpatikra ()
- int studskpatikra ()
- char isvedbudpatikra ()
- int erezpatikra ()
- char rikbudpatikra ()
- int pazymiopatikra ()
- char skirstymopatikra ()
- int skirststratpat ()
- char fgeneravimopatikra ()

5.2.1 Function Documentation

5.2.1.1 budaspatikra()

```
char budaspatikra ( )
```

5.2.1.2 dskaitpatikra()

```
char dskaitpatikra ( )
```

5.2.1.3 erezpatikra()

```
int erezpatikra ( )
```

5.2.1.4 fgeneravimopatikra()

```
char fgeneravimopatikra ( )
```

5.2.1.5 isvedbudpatikra()

```
char isvedbudpatikra ( )
```

5.2.1.6 ivedbudpatikra()

```
int ivedbudpatikra ( )
```

5.2.1.7 pazymiopatikra()

```
int pazymiopatikra ( )
```

5.2.1.8 rikbudpatikra()

```
char rikbudpatikra ( )
```

5.2.1.9 skirststratpat()

```
int skirststratpat ( )
```

5.2.1.10 skirstymopatikra()

```
char skirstymopatikra ( )
```

5.2.1.11 studskpatikra()

```
int studskpatikra ( )
```

5.3 errorfinder.h

Go to the documentation of this file.

```
#include <iostream>
#include <limits>

using namespace std;

int ivedbudpatikra();
char budaspatikra();
char dskaitpatikra();
int studskpatikra();
char isvedbudpatikra();
int erezpatikra();
char rikbudpatikra();
int pazymiopatikra();
char skirstymopatikra();
int skirststratpat();
char fgeneravimopatikra();
```

5.4 filegenerator.cpp File Reference

```
#include "filegenerator.h"
#include "errorfinder.h"
#include "functions.h"
#include "studentas.h"
```

Functions

- int failugeneravimas ()

5.4.1 Function Documentation

5.4.1.1 failugeneravimas()

```
int failugeneravimas ( )
```

5.5 filegenerator.h File Reference

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <chrono>
```

Functions

- int failugeneravimas ()

5.5.1 Function Documentation

5.5.1.1 failugeneravimas()

```
int failugeneravimas ( )
```

5.6 filegenerator.h

Go to the documentation of this file.

```
#include<iostream>
#include<iomanip>
#include<fstream>
#include <sstream>
#include <chrono>


using namespace std;
using namespace std::chrono;

int failugeneravimas();
```

5.7 functions.cpp File Reference

```
#include "functions.h"
#include "errorfinder.h"
#include "studentas.h"
```

Functions

- void skaitymasisfailo (Vector< studentas > &A, char budas, char ivedbudas)
- void irasymasifaila (Vector< studentas > &A, char budas)
- void isvedimas (Vector< studentas > &A, char budas)
- void pazymiuived (studentas &new_studentas, char budas, int ivedbudas)
- void skaiciavimas (studentas &new_studentas, char budas)
- bool rikiavimasgbalas (const studentas &a, const studentas &b)
- bool rikiavimasvardas (const studentas &a, const studentas &b)
- bool rikiavimaspavarde (const studentas &a, const studentas &b)
- void rikiavimas (Vector< studentas > &A)
- void skirstymas1 (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V)
- void skirstymas2 (Vector< studentas > &A, Vector< studentas > &V)
- void skirstymas3 (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V)
- void irasymasifailaK (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V, char budas, int skistr)

Variables

- int tlaikas = 0

5.7.1 Function Documentation

5.7.1.1 irasymasifaila()

```
void irasymasifaila (
            Vector< studentas > & A,
            char budas )
```

5.7.1.2 irasymasifailaK()

```
void irasymasifailaK (
            Vector< studentas > & A,
            Vector< studentas > & K,
            Vector< studentas > & V,
            char budas,
            int skistr )
```

5.7.1.3 isvedimas()

```
void isvedimas (
            Vector< studentas > & A,
            char budas )
```

5.7.1.4 pazymiuived()

```
void pazymiuived (
            studentas & new_studentas,
            char budas,
            int ivedbudas )
```

5.7.1.5 rikiavimas()

```
void rikiavimas (
            Vector< studentas > & A )
```

5.7.1.6 rikiavimasgbalas()

```
bool rikiavimasgbalas (
            const studentas & a,
            const studentas & b )
```

5.7.1.7 rikiavimaspavarde()

```
bool rikiavimaspavarde (
            const studentas & a,
            const studentas & b )
```

5.7.1.8 rikiavimasvardas()

```
bool rikiavimasvardas (
            const studentas & a,
            const studentas & b )
```

5.7.1.9 skaiciavimas()

```
void skaiciavimas (
            studentas & new_studentas,
            char budas )
```

5.7.1.10 skaitymasisfailo()

```
void skaitymasisfailo (
            Vector< studentas > & A,
            char budas,
            char ivedbudas )
```

5.7.1.11 skirstymas1()

```
void skirstymas1 (
            Vector< studentas > & A,
            Vector< studentas > & K,
            Vector< studentas > & V )
```

5.7.1.12 skirstymas2()

```
void skirstymas2 (
            Vector< studentas > & A,
            Vector< studentas > & V )
```

5.7.1.13 skirstymas3()

```
void skirstymas3 (
            Vector< studentas > & A,
            Vector< studentas > & K,
            Vector< studentas > & V )
```

5.7.2 Variable Documentation

5.7.2.1 tlaikas

```
int tlaikas = 0
```

5.8 functions.h File Reference

```
#include <iostream>
#include <iomanip>
#include <limits>
#include <algorithm>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <string>
#include <fstream>
#include <sstream>
#include <chrono>
#include "studentas.h"
#include "vector.h"
```

Functions

- void skaitymasisfailo (Vector< studentas > &A, char budas, char ivedbudas)
- void isvedimas (Vector< studentas > &A, char budas)
- void pazymiuived (studentas &new_studentas, char budas, int ivedbudas)
- void skaiciavimas (studentas &new_studentas, char budas)
- void irasymasifaila (Vector< studentas > &A, char budas)
- bool rikiavimasgbalas (const studentas &a, const studentas &b)
- bool rikiavimasvardas (const studentas &a, const studentas &b)
- bool rikiavimaspavarde (const studentas &a, const studentas &b)
- void rikiavimas (Vector< studentas > &A)
- void skirstymas1 (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V)
- void skirstymas2 (Vector< studentas > &A, Vector< studentas > &V)
- void skirstymas3 (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V)
- void irasymasifailaK (Vector< studentas > &A, Vector< studentas > &K, Vector< studentas > &V, char budas, int skistr)

Variables

- int tlaikas

5.8.1 Function Documentation

5.8.1.1 irasymasifaila()

```
void irasymasifaila (
             Vector< studentas > & A,
             char budas )
```

5.8.1.2 irasymasifailaK()

```
void irasymasifailaK (
             Vector< studentas > & A,
             Vector< studentas > & K,
             Vector< studentas > & V,
             char budas,
             int skistr )
```

5.8.1.3 isvedimas()

```
void isvedimas (
             Vector< studentas > & A,
             char budas )
```

5.8.1.4 pazymiuived()

```
void pazymiuived (
             studentas & new_studentas,
             char budas,
             int ivedbudas )
```

5.8.1.5 rikiavimas()

```
void rikiavimas (
             Vector< studentas > & A )
```

5.8.1.6 rikiavimasgbalas()

```
bool rikiavimasgbalas (
             const studentas & a,
             const studentas & b )
```

5.8.1.7 rikiavimaspavarde()

```
bool rikiavimaspavarde (
            const studentas & a,
            const studentas & b )
```

5.8.1.8 rikiavimasvardas()

```
bool rikiavimasvardas (
            const studentas & a,
            const studentas & b )
```

5.8.1.9 skaiciavimas()

```
void skaiciavimas (
            studentas & new_studentas,
            char budas )
```

5.8.1.10 skaitymasisfailo()

```
void skaitymasisfailo (
            Vector< studentas > & A,
            char budas,
            char ivedbudas )
```

5.8.1.11 skirstymas1()

```
void skirstymas1 (
            Vector< studentas > & A,
            Vector< studentas > & K,
            Vector< studentas > & V )
```

5.8.1.12 skirstymas2()

```
void skirstymas2 (
            Vector< studentas > & A,
            Vector< studentas > & V )
```

5.8.1.13 skirstymas3()

```
void skirstymas3 (
            Vector< studentas > & A,
            Vector< studentas > & K,
            Vector< studentas > & V )
```

5.8.2 Variable Documentation

5.8.2.1 tlaikas

```
int tlaikas [extern]
```

5.9 functions.h

Go to the documentation of this file.

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <iomanip>
#include <limits>
#include <algorithm>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <string>
#include <fstream>
#include <sstream>
#include <chrono>
#include "studentas.h"
#include "vector.h"

using namespace std;
using namespace std::chrono;

// struct studentas
// {
//     string vardas;
//     string pavarde;
//     vector<int> ndrez; //sudaromas vektorius
//     int erez;
//     double gbalas;
// };

extern int tlaikas;

void skaitymasisfailo(Vector<studentas> &A, char budas, char ivedbudas);
void isvedimas(Vector<studentas> &A, char budas);
void pazymiuived(studentas &new_studentas, char budas, int ivedbudas);
void skaiciavimas(studentas &new_studentas, char budas);
void irasymasifaila(Vector<studentas> &A, char budas);
bool rikiavimasgbalas(const studentas &a, const studentas &b);
bool rikiavimasvardas(const studentas &a, const studentas &b);
bool rikiavimaspavarde(const studentas &a, const studentas &b);
void rikiavimas(Vector<studentas> &A);
void skirstymas1(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V);
void skirstymas2(Vector<studentas> &A, Vector<studentas> &V);
void skirstymas3(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V);
void irasymasifailaK(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V, char budas, int
      skistr);

#endif // FUNCTIONS_H
```

5.10 main.cpp File Reference

```
#include "functions.h"
#include "errorfinder.h"
#include "filegenerator.h"
#include "studentas.h"
#include "vector.h"
#include <chrono>
```

Functions

- void testCopyConstruction ()
- void testMoveConstruction ()
- void testCopyAssignment ()
- void testMoveAssignment ()
- int main ()

5.10.1 Function Documentation

5.10.1.1 main()

```
int main ( )
```

5.10.1.2 testCopyAssignment()

```
void testCopyAssignment ( )
```

5.10.1.3 testCopyConstruction()

```
void testCopyConstruction ( )
```

5.10.1.4 testMoveAssignment()

```
void testMoveAssignment ( )
```

5.10.1.5 testMoveConstruction()

```
void testMoveConstruction ( )
```

5.11 studentas.cpp File Reference

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

5.12 studentas.h File Reference

```
#include "zmogus.h"
#include "vector.h"
#include "errorfinder.h"
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <algorithm>
#include <sstream>
```

Classes

- class studentas

5.13 studentas.h

Go to the documentation of this file.

```
#ifndef STUDENTAS_H
#define STUDENTAS_H

#include "zmogus.h"
#include "vector.h"
#include "errorfinder.h"
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <algorithm>
#include <sstream>

using namespace std;

class studentas : public zmogus {
private:
  Vector<int> ndrez;
  int erez;
  double gbalas;
  // interfeisas
  public:
    string line;
    char budas;
    studentas(); // default konstruktorius
    studentas(const string &v, const string &p, const Vector<int> &nd, int e, double g);

    ~studentas(); // destruktorius

    studentas(const studentas &kit); // copy konstruktorius

    studentas &operator=(const studentas &kit); // priskyrimo operatorius

    studentas(studentas &&kit) noexcept; // move konstruktorius

    studentas &operator=(studentas &&kit) noexcept;

friend std::istream &operator»(std::istream &in, studentas &kit){
  kit.ndrez.clear();
  int sum = 0;

    if(kit.budas == 'f'){
      istringstream my_buffer(kit.line);

      my_buffer » kit.vardas » kit.pavarde;
      int pazymys;
      while (my_buffer » pazymys)
      {
        kit.ndrez.push_back(pazymys); // prisikiriamas elSementas
        sum += pazymys;
      }

      if (!kit.ndrez.empty()) {
        kit.erez = kit.ndrez.back();
        kit.ndrez.pop_back();
        sum -= kit.erez;
    }
    kit.gbalas = sum;}
    if(kit.budas == 'r')
    {
      cout « "Iveskite studento varda ir pavarde arba „11‟, jeigu norite uzbaigti studentu vedima: ";
      in » kit.vardas;
      if (kit.vardas != "11")
      {
        in » kit.pavarde;
      }
    }
    return in;
  }

    friend std::ostream &operator«(std::ostream &out, const studentas &kit)
   {
        out « setw(25) « left « kit.vardas « setw(25) « left « kit.pavarde « setprecision(3) « left «
kit.gbalas « '\n';
```

```
        return out;
  }

    string getVardas() const { return vardas; }   // get'eriai
    string getPavarde() const { return pavarde; } // get'eriai
    Vector<int> getNdrez() const { return ndrez; }
    int getErez() const { return erez; }
    double getGbalas() const { return gbalas; } // get'eriai

    void setVardas(const string &v) { vardas = v; }
    void setPavarde(const string &p) { pavarde = p; }
    void setNdrez(const Vector<int> &nd) { ndrez = nd; }
    void setErez(int e) { erez = e; }
    void setGbalas(double g) { gbalas = g; } // set'eriai

    void sortNdrez() { sort(ndrez.begin(), ndrez.end()); }


};

#endif // STUDENTAS_H
```

5.14 test.cpp File Reference

```
#include "catch2/catch.hpp"
#include "vector.h"
```

Macros

- #define CATCH_CONFIG_MAIN

Functions

- TEST_CASE ("Default Constructor", "[Default Constructor]")
- TEST_CASE ("Initializer List Constructor", "[Initializer List Constructor]")
- TEST_CASE ("Copy Constructor", "[Copy Constructor]")
- TEST_CASE ("Move Constructor", "[Move Constructor]")
- TEST_CASE ("Copy Assignment Operator", "[Copy Assignment Operator]")
- TEST_CASE ("Move Assignment Operator", "[Move Assignment Operator]")
- TEST_CASE ("Element Access", "[Element Access]")
- TEST_CASE ("Modifiers", "[Modifiers]")
- TEST_CASE ("Iterators", "[Iterators]")

5.14.1 Macro Definition Documentation

5.14.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

5.14.2 Function Documentation

5.14.2.1 TEST_CASE() [1/9]

```
TEST_CASE (
            "Copy Assignment Operator" ,
            "" [Copy Assignment Operator] )
```

5.14.2.2 TEST_CASE() [2/9]

```
TEST_CASE (
             "Copy Constructor" ,
             "" [Copy Constructor] )
```

5.14.2.3 TEST_CASE() [3/9]

```
TEST_CASE (
             "Default Constructor" ,
             "" [Default Constructor] )
```

5.14.2.4 TEST_CASE() [4/9]

```
TEST_CASE (
             "Element Access" ,
             "" [Element Access] )
```

5.14.2.5 TEST_CASE() [5/9]

```
TEST_CASE (
             "Initializer List Constructor" ,
             "" [Initializer List Constructor] )
```

5.14.2.6 TEST_CASE() [6/9]

```
TEST_CASE (
             "Iterators" ,
             "" [Iterators] )
```

5.14.2.7 TEST_CASE() [7/9]

```
TEST_CASE (
             "Modifiers" ,
             "" [Modifiers] )
```

5.14.2.8 TEST_CASE() [8/9]

```
TEST_CASE (
             "Move Assignment Operator" ,
             "" [Move Assignment Operator] )
```

5.14.2.9 TEST_CASE() [9/9]

```
TEST_CASE (
             "Move Constructor" ,
             "" [Move Constructor] )
```

5.15 vector.h File Reference

```
#include <iostream>
#include <stdexcept>
#include <limits>
#include <initializer_list>
```

Classes

- class Vector< T >

5.16 vector.h

Go to the documentation of this file.

```
#ifndef VECTOR_H
#define VECTOR_H
#include <iostream>
#include <stdexcept>
#include <limits>
#include <initializer_list>

template <typename T>
class Vector {
private:
    T* data = nullptr; // Pointer to the dynamically allocated array
    size_t capacity_ = 0; // Capacity of the array
    size_t length = 0; // Number of elements in the array
    size_t reallocations;

public:
    // Member types
    using value_type = T;
    using reference = T&;
    using const_reference = const T&;
    using iterator = T*;
    using const_iterator = const T*;

    // Constructor
    Vector() : data(nullptr), capacity_(0), length(0) {}

    // Constructor with initializer list
    Vector(std::initializer_list<T> init) : data(nullptr), capacity_(0), length(0) {
        reserve(init.size());
        for (const T& value : init) {
            push_back(value);
        }
    }

    // Copy constructor
    Vector(const Vector& other) : data(nullptr), capacity_(0), length(0) {
        reserve(other.length);
        for (size_t i = 0; i < other.length; ++i) {
            push_back(other.data[i]);
        }
    }

    // Move constructor
    Vector(Vector&& other) noexcept : data(other.data), capacity_(other.capacity_),
length(other.length) {
        other.data = nullptr;
        other.capacity_ = 0;
        other.length = 0;
    }

    // Destructor
    ~Vector() {
        delete[] data;
    }

    // Copy assignment operator
    Vector& operator=(const Vector& other) {
        if (this != &other) {
            delete[] data;
            data = nullptr;
            capacity_ = 0;
            length = 0;
            reserve(other.length);
            for (size_t i = 0; i < other.length; ++i) {
                push_back(other.data[i]);
            }
        }
        return *this;
    }

    // Move assignment operator
    Vector& operator=(Vector&& other) noexcept {
        if (this != &other) {
            delete[] data;
            data = other.data;
            capacity_ = other.capacity_;
            length = other.length;
            other.data = nullptr;
            other.capacity_ = 0;
            other.length = 0;
        }
        return *this;
    }

    // Member functions

    // Capacity
    size_t size() const {
        return length;
    }

    size_t max_size() const {
        return std::numeric_limits<size_t>::max() / sizeof(T);
    }

    size_t capacity() const {
        return capacity_;
    }

    bool empty() const {
        return length == 0;
    }

    void reserve(size_t new_capacity) {
            if (new_capacity <= capacity_) {
                return;
            }
            ++reallocations; // Increment reallocations count
            T* new_data = new T[new_capacity];
            std::copy(data, data + length, new_data);
            delete[] data;
            data = new_data;
            capacity_ = new_capacity;
        }

    void resize(size_t new_size, const T& value = T()) {
        if (new_size > length) {
            reserve(new_size);
            std::fill(data + length, data + new_size, value);
        }
        length = new_size;
    }

    void shrink_to_fit() {
        if (length < capacity_) {
            T* new_data = new T[length];
            std::copy(data, data + length, new_data);
            delete[] data;
            data = new_data;
            capacity_ = length;
        }
    }

    // Element access
    reference operator[](size_t index) {
        if (index >= length) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    const_reference operator[](size_t index) const {
        if (index >= length) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    reference at(size_t index) {
        if (index >= length) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    const_reference at(size_t index) const {
        if (index >= length) {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
    }

    reference front() {
        if (length == 0) {
            throw std::out_of_range("Vector is empty");
        }
        return data[0];
    }

    const_reference front() const {
        if (length == 0) {
            throw std::out_of_range("Vector is empty");
        }
        return data[0];
    }

    reference back() {
        if (length == 0) {
            throw std::out_of_range("Vector is empty");
        }
        return data[length - 1];
    }

    const_reference back() const {
        if (length == 0) {
            throw std::out_of_range("Vector is empty");
        }
        return data[length - 1];
    }

    // Modifiers
    void push_back(const T& value) {
        if (length >= capacity_) {
            reserve((capacity_ == 0) ? 1 : capacity_ * 2);
        }
        data[length++] = value;
    }

    void pop_back() {
        if (length == 0) {
            throw std::out_of_range("Vector is empty");
        }
        --length;
    }

    iterator erase(iterator position) {
        if (position < data || position >= data + length) {
            throw std::out_of_range("Iterator out of range");
        }
        std::copy(position + 1, data + length, position);
        --length;
        return position;
    }

    iterator erase(iterator first, iterator last) {
        if (first < data || first >= data + length || last < data || last > data + length || first >
last) {
            throw std::out_of_range("Iterator out of range");
        }
        std::copy(last, data + length, first);
        length -= last - first;
        return first;
    }

    void clear() {
        length = 0;
    }

    // Iterators
    iterator begin() {
        return data;
    }
    const_iterator begin() const {
        return data;
    }

    iterator end() {
        return data + length;
    }

    const_iterator end() const {
        return data + length;
    }

    Vector(T* data, const size_t& capacity_, const size_t& length)
        : data(data), capacity_(capacity_), length(length)
    {
    }

    bool operator==(const Vector& other) const
    {
        return false;
    }

    // Function to get the number of reallocations
    size_t getReallocationCount() const {
        return reallocations;
    }
};

#endif // VECTOR_H
```

5.17 zmogus.h File Reference

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

Classes

- class zmogus

5.18 zmogus.h

Go to the documentation of this file.

```
#ifndef ZMOGUS_H
#define ZMOGUS_H

#include <iostream>
#include <vector>
#include <string>

using namespace std;

class zmogus {
    protected:
      string vardas;
        string pavarde;
public:
    virtual ~zmogus(){};
    virtual string getVardas() const = 0;
    virtual void setVardas(string v) { vardas = v; }

    virtual string getPavarde() const = 0;
    virtual void setPavarde(string p) { pavarde = p; }
};

#endif // ZMOGUS_H
```

Index