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:

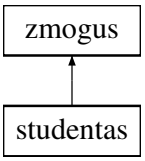
Chapter 4

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]
```

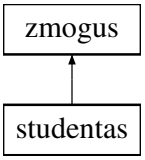
The documentation for this class was generated from the following file:

- 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.

```
00001 #include <iostream>
00002 #include <limits>
00003
00004 using namespace std;
00005
00006 int ivedbudpatikra();
00007 char budaspatikra();
00008 char dskaitpatikra();
00009 int studskpatikra();
00010 char isvedbudpatikra();
00011 int erezpatikra();
00012 char rikbudpatikra();
00013 int pazymiopatikra();
00014 char skirstymopatikra();
00015 int skirststratpat();
00016 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.

```
00001 #include<iostream>
00002 #include<iomanip>
00003 #include<fstream>
00004 #include <sstream>
00005 #include <chrono>
00006
00007
00008 using namespace std;
00009 using namespace std::chrono;
00010
00011 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.

```
00001 #ifndef FUNCTIONS_H
00002 #define FUNCTIONS_H
00003
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <limits>
00007 #include <algorithm>
00008 #include <vector>
00009 #include <cstdlib>
00010 #include <ctime>
00011 #include <string>
00012 #include <fstream>
00013 #include <sstream>
00014 #include <chrono>
00015 #include "studentas.h"
00016 #include "vector.h"
00017
00018 using namespace std;
00019 using namespace std::chrono;
00020
00021 // struct studentas
00022 // {
00023 //     string vardas;
00024 //     string pavarde;
00025 //     vector<int> ndrez; //sudaromas vektorius
00026 //     int erez;
00027 //     double gbalas;
00028 // };
00029
00030 extern int tlaikas;
00031
00032 void skaitymasisfailo(Vector<studentas> &A, char budas, char ivedbudas);
00033 void isvedimas(Vector<studentas> &A, char budas);
00034 void pazymiuived(studentas &new_studentas, char budas, int ivedbudas);
00035 void skaiciavimas(studentas &new_studentas, char budas);
00036 void irasymasifaila(Vector<studentas> &A, char budas);
00037 bool rikiavimasgbalas(const studentas &a, const studentas &b);
00038 bool rikiavimasvardas(const studentas &a, const studentas &b);
00039 bool rikiavimaspavarde(const studentas &a, const studentas &b);
00040 void rikiavimas(Vector<studentas> &A);
00041 void skirstymas1(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V);
00042 void skirstymas2(Vector<studentas> &A, Vector<studentas> &V);
00043 void skirstymas3(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V);
00044 void irasymasifailaK(Vector<studentas> &A, Vector<studentas> &K, Vector<studentas> &V, char budas, int
      skistr);
00045
00046 #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.

```
00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
00003
00004 #include "zmogus.h"
00005 #include "vector.h"
00006 #include "errorfinder.h"
00007 #include <iostream>
00008 #include <iomanip>
00009 #include <vector>
00010 #include <string>
00011 #include <algorithm>
00012 #include <sstream>
00013
00014 using namespace std;
00015
00016 class studentas : public zmogus {
00017 private:
00018   Vector<int> ndrez;
00019   int erez;
00020   double gbalas;
00021   // interfeisas
00022   public:
00023     string line;
00024     char budas;
00025     studentas(); // default konstruktorius
00026     studentas(const string &v, const string &p, const Vector<int> &nd, int e, double g);
00027
00028     ~studentas(); // destruktorius
00029
00030     studentas(const studentas &kit); // copy konstruktorius
00031
00032     studentas &operator=(const studentas &kit); // priskyrimo operatorius
00033
00034     studentas(studentas &&kit) noexcept; // move konstruktorius
00035
00036     studentas &operator=(studentas &&kit) noexcept;
00037
00038 friend std::istream &operator»(std::istream &in, studentas &kit){
00039   kit.ndrez.clear();
00040   int sum = 0;
00041
00042     if(kit.budas == 'f'){
00043       istringstream my_buffer(kit.line);
00044
00045       my_buffer » kit.vardas » kit.pavarde;
00046       int pazymys;
00047       while (my_buffer » pazymys)
00048       {
00049         kit.ndrez.push_back(pazymys); // prisikiriamas elSementas
00050         sum += pazymys;
00051       }
00052
00053       if (!kit.ndrez.empty()) {
00054         kit.erez = kit.ndrez.back();
00055         kit.ndrez.pop_back();
00056         sum -= kit.erez;
00057     }
00058     kit.gbalas = sum;}
00059     if(kit.budas == 'r')
00060     {
00061       cout « "Iveskite studento varda ir pavarde arba „11“, jeigu norite uzbaigti studentu vedima: ";
00062       in » kit.vardas;
00063       if (kit.vardas != "11")
00064       {
00065         in » kit.pavarde;
00066       }
00067     }
00068     return in;
00069   }
00070
00071     friend std::ostream &operator«(std::ostream &out, const studentas &kit)
00072    {
00073         out « setw(25) « left « kit.vardas « setw(25) « left « kit.pavarde « setprecision(3) « left «
      kit.gbalas « '\n';
```

```
00074         return out;
00075   }
00076
00077     string getVardas() const { return vardas; }   // get'eriai
00078     string getPavarde() const { return pavarde; } // get'eriai
00079     Vector<int> getNdrez() const { return ndrez; }
00080     int getErez() const { return erez; }
00081     double getGbalas() const { return gbalas; } // get'eriai
00082
00083     void setVardas(const string &v) { vardas = v; }
00084     void setPavarde(const string &p) { pavarde = p; }
00085     void setNdrez(const Vector<int> &nd) { ndrez = nd; }
00086     void setErez(int e) { erez = e; }
00087     void setGbalas(double g) { gbalas = g; } // set'eriai
00088
00089     void sortNdrez() { sort(ndrez.begin(), ndrez.end()); }
00090
00091
00092 };
00093
00094 #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.

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003 #include <iostream>
00004 #include <stdexcept>
00005 #include <limits>
00006 #include <initializer_list>
00007
00008 template <typename T>
00009 class Vector {
00010 private:
00011     T* data = nullptr; // Pointer to the dynamically allocated array
00012     size_t capacity_ = 0; // Capacity of the array
00013     size_t length = 0; // Number of elements in the array
00014     size_t reallocations;
00015
00016 public:
00017     // Member types
00018     using value_type = T;
00019     using reference = T&;
00020     using const_reference = const T&;
00021     using iterator = T*;
00022     using const_iterator = const T*;
00023
00024     // Constructor
00025     Vector() : data(nullptr), capacity_(0), length(0) {}
00026
00027     // Constructor with initializer list
00028     Vector(std::initializer_list<T> init) : data(nullptr), capacity_(0), length(0) {
00029         reserve(init.size());
00030         for (const T& value : init) {
00031             push_back(value);
00032         }
00033     }
00034
00035     // Copy constructor
00036     Vector(const Vector& other) : data(nullptr), capacity_(0), length(0) {
00037         reserve(other.length);
00038         for (size_t i = 0; i < other.length; ++i) {
00039             push_back(other.data[i]);
00040         }
00041     }
00042
00043     // Move constructor
00044     Vector(Vector&& other) noexcept : data(other.data), capacity_(other.capacity_),
      length(other.length) {
00045         other.data = nullptr;
00046         other.capacity_ = 0;
00047         other.length = 0;
00048     }
00049
00050     // Destructor
00051     ~Vector() {
00052         delete[] data;
00053     }
00054
00055     // Copy assignment operator
00056     Vector& operator=(const Vector& other) {
00057         if (this != &other) {
```

```
00058             delete[] data;
00059             data = nullptr;
00060             capacity_ = 0;
00061             length = 0;
00062             reserve(other.length);
00063             for (size_t i = 0; i < other.length; ++i) {
00064                 push_back(other.data[i]);
00065             }
00066         }
00067         return *this;
00068     }
00069
00070     // Move assignment operator
00071     Vector& operator=(Vector&& other) noexcept {
00072         if (this != &other) {
00073             delete[] data;
00074             data = other.data;
00075             capacity_ = other.capacity_;
00076             length = other.length;
00077             other.data = nullptr;
00078             other.capacity_ = 0;
00079             other.length = 0;
00080         }
00081         return *this;
00082     }
00083
00084     // Member functions
00085
00086     // Capacity
00087     size_t size() const {
00088         return length;
00089     }
00090
00091     size_t max_size() const {
00092         return std::numeric_limits<size_t>::max() / sizeof(T);
00093     }
00094
00095     size_t capacity() const {
00096         return capacity_;
00097     }
00098
00099     bool empty() const {
00100         return length == 0;
00101     }
00102
00103     void reserve(size_t new_capacity) {
00104             if (new_capacity <= capacity_) {
00105                 return;
00106             }
00107             ++reallocations; // Increment reallocations count
00108             T* new_data = new T[new_capacity];
00109             std::copy(data, data + length, new_data);
00110             delete[] data;
00111             data = new_data;
00112             capacity_ = new_capacity;
00113         }
00114
00115     void resize(size_t new_size, const T& value = T()) {
00116         if (new_size > length) {
00117             reserve(new_size);
00118             std::fill(data + length, data + new_size, value);
00119         }
00120         length = new_size;
00121     }
00122
00123     void shrink_to_fit() {
00124         if (length < capacity_) {
00125             T* new_data = new T[length];
00126             std::copy(data, data + length, new_data);
00127             delete[] data;
00128             data = new_data;
00129             capacity_ = length;
00130         }
00131     }
00132
00133     // Element access
00134     reference operator[](size_t index) {
00135         if (index >= length) {
00136             throw std::out_of_range("Index out of range");
00137         }
00138         return data[index];
00139     }
00140
00141     const_reference operator[](size_t index) const {
00142         if (index >= length) {
00143             throw std::out_of_range("Index out of range");
00144         }
```

```
00145         return data[index];
00146     }
00147
00148     reference at(size_t index) {
00149         if (index >= length) {
00150             throw std::out_of_range("Index out of range");
00151         }
00152         return data[index];
00153     }
00154
00155     const_reference at(size_t index) const {
00156         if (index >= length) {
00157             throw std::out_of_range("Index out of range");
00158         }
00159         return data[index];
00160     }
00161
00162     reference front() {
00163         if (length == 0) {
00164             throw std::out_of_range("Vector is empty");
00165         }
00166         return data[0];
00167     }
00168
00169     const_reference front() const {
00170         if (length == 0) {
00171             throw std::out_of_range("Vector is empty");
00172         }
00173         return data[0];
00174     }
00175
00176     reference back() {
00177         if (length == 0) {
00178             throw std::out_of_range("Vector is empty");
00179         }
00180         return data[length - 1];
00181     }
00182
00183     const_reference back() const {
00184         if (length == 0) {
00185             throw std::out_of_range("Vector is empty");
00186         }
00187         return data[length - 1];
00188     }
00189
00190     // Modifiers
00191     void push_back(const T& value) {
00192         if (length >= capacity_) {
00193             reserve((capacity_ == 0) ? 1 : capacity_ * 2);
00194         }
00195         data[length++] = value;
00196     }
00197
00198     void pop_back() {
00199         if (length == 0) {
00200             throw std::out_of_range("Vector is empty");
00201         }
00202         --length;
00203     }
00204
00205     iterator erase(iterator position) {
00206         if (position < data || position >= data + length) {
00207             throw std::out_of_range("Iterator out of range");
00208         }
00209         std::copy(position + 1, data + length, position);
00210         --length;
00211         return position;
00212     }
00213
00214     iterator erase(iterator first, iterator last) {
00215         if (first < data || first >= data + length || last < data || last > data + length || first >
      last) {
00216             throw std::out_of_range("Iterator out of range");
00217         }
00218         std::copy(last, data + length, first);
00219         length -= last - first;
00220         return first;
00221     }
00222
00223     void clear() {
00224         length = 0;
00225     }
00226
00227     // Iterators
00228     iterator begin() {
00229         return data;
00230     }
```

```
00231
00232     const_iterator begin() const {
00233         return data;
00234     }
00235
00236     iterator end() {
00237         return data + length;
00238     }
00239
00240     const_iterator end() const {
00241         return data + length;
00242     }
00243
00244     Vector(T* data, const size_t& capacity_, const size_t& length)
00245         : data(data), capacity_(capacity_), length(length)
00246     {
00247     }
00248
00249     bool operator==(const Vector& other) const
00250     {
00251         return false;
00252     }
00253
00254     // Function to get the number of reallocations
00255     size_t getReallocationCount() const {
00256         return reallocations;
00257     }
00258 };
00259
00260 #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.

```
00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <string>
00007
00008 using namespace std;
00009
00010 class zmogus {
00011     protected:
00012       string vardas;
00013         string pavarde;
00014 public:
00015     virtual ~zmogus(){};
00016     virtual string getVardas() const = 0;
00017     virtual void setVardas(string v) { vardas = v; }
00018
00019     virtual string getPavarde() const = 0;
00020     virtual void setPavarde(string p) { pavarde = p; }
00021 };
00022
00023 #endif // ZMOGUS_H
```

Index