My Project V3.0

Generated by Doxygen 1.10.0

Chapter 1

V3 Versija

1.1 Kas nauja 3 versjoje

1. Sukurta nuosava vektoriaus klase - myVector
2. Patikrintos funkcijos
3. Atlikti testavimai

1.2 Funkciju testavimas

1.2.1 Vektoriu palyginimai

1.2.2 Clear, pop, erase

1.2.3 empty, getsize, get capacity

1.2.4 first, last, begin, end

1.2.5 [], at, pop

1.3 myVector ir std::vector spartos testavimas

Elementu kiekis	std::vector	myVector
10000	0.0001257	0.0001122
100000	0.0008885	0.000937
1000000	0.0077361	0.0071594
10000000	0.078231	0.0986152
100000000	0.725096	0.785338

1. myVector veikia greičiau su mažesniais skaičaiais (Vidutiniškai 10% - 15% greičiau)

2. Su dideliais skaičiais myVector veikia lėčiau (Vidutiniškai 5% - 10% lėčiau)

1.4 Resize Kiekis

1.4.1 Atminties perskirstymas myVector konteineryje yra 1 mažesnins del to, kad default myVector sukuriamas su 1 capacity

1.5 Programos veikimo laikas su konteineriais

1.5.1 100.000.txt

Darbas	std::vector	myVector
Duomenu nuskaitymas	0.2695445	0.2869592
skirstymas i dvi grupes	0.0452824	0.1115644
Rusiavimas	0.2883375	0.3545991
Visas laikas	0.6031644	0.7531227

1.5.2 1.000.000.txt

Darbas	std::vector	myVector
Duomenu nuskaitymas	2.609202	2.738449
skirstymas i dvi grupes	0.5331666	1.227727
Rusiavimas	3.764685	4.563468
Visas laikas	6.907054	8.529645

1.5.3 10.000.000.txt

Darbas	std::vector	myVector
Duomenu nuskaitymas	27.49252	30.6472
skirstymas i dvi grupes	4.522078	11.05496
Rusiavimas	46.87415	58.14309
Visas laikas	78.88875	99.84526

1.5.4 Papildomas testavimas

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 myVector< T > Class Template Reference

`#include <myVector.h>`

Public Member Functions

- myVector ()
- myVector (size_t value, T data)
- myVector (std::initializer_list< T > ilist)
- myVector (T data)
- myVector (const myVector &other)
- myVector (myVector &&other) noexcept
- myVector & operator= (const myVector &other)
- myVector & operator= (myVector &&other) noexcept
- ~myVector ()
- void reserve (size_t new_capacity)
- void push_back (T duomenys)
- void pop_back ()
- T at (size_t index)
- int size () const
- int getCapacity ()
- void print ()
- bool empty ()
- void clear ()
- T & first ()
- T & last ()
- T * begin ()
- T * end ()
- T & operator[ ] (size_t index) const
- void assign (size_t count, const T &value)
- template<typename InputIterator >
  void assign (InputIterator first, InputIterator last)
- void erase (T *position)
- void erase (T *first, T *last)
- void swap (myVector &other)
- void resize (size_t n)

- T * insert (T *position, const T &val)
- void insert (T *position, size_t n, const T &val)
- template<typename InputIterator >
  void insert (T *position, InputIterator first, InputIterator last)
- bool operator== (const myVector &other) const
- bool operator!= (const myVector &other) const
- bool operator< (const myVector &other) const
- bool operator<= (const myVector &other) const
- bool operator> (const myVector &other) const
- bool operator>= (const myVector &other) const

5.1.1 Constructor & Destructor Documentation

5.1.1.1 myVector() [1/6]

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

5.1.1.2 myVector() [2/6]

```
template<typename T >
myVector< T >::myVector (
            size_t value,
            T data )  [inline]
```

5.1.1.3 myVector() [3/6]

```
template<typename T >
myVector< T >::myVector (
            std::initializer_list< T > ilist )  [inline]
```

5.1.1.4 myVector() [4/6]

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

5.1.1.5 myVector() [5/6]

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

5.1.1.6 myVector() [6/6]

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

5.1.1.7 ~myVector()

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

5.1.2 Member Function Documentation

5.1.2.1 assign() [1/2]

```
template<typename T >
template<typename InputIterator >
void myVector< T >::assign (
            InputIterator first,
            InputIterator last )  [inline]
```

5.1.2.2 assign() [2/2]

```
template<typename T >
void myVector< T >::assign (
            size_t count,
            const T & value )  [inline]
```

5.1.2.3 at()

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

5.1.2.4 begin()

```
template<typename T >
T * myVector< T >::begin ( )  [inline]
```

5.1.2.5 clear()

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

5.1.2.6 empty()

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

5.1.2.7 end()

```
template<typename T >
T * myVector< T >::end ( )  [inline]
```

5.1.2.8 erase() [1/2]

```
template<typename T >
void myVector< T >::erase (
            T * first,
            T * last )  [inline]
```

5.1.2.9 erase() [2/2]

```
template<typename T >
void myVector< T >::erase (
            T * position )  [inline]
```

5.1.2.10 first()

```
template<typename T >
T & myVector< T >::first ( )  [inline]
```

5.1.2.11 getCapacity()

```
template<typename T >
int myVector< T >::getCapacity ( )  [inline]
```

5.1.2.12 insert() [1/3]

```
template<typename T >
T * myVector< T >::insert (
            T * position,
            const T & val )  [inline]
```

5.1.2.13 insert() [2/3]

```
template<typename T >
template<typename InputIterator >
void myVector< T >::insert (
            T * position,
            InputIterator first,
            InputIterator last )  [inline]
```

5.1.2.14 insert() [3/3]

```
template<typename T >
void myVector< T >::insert (
            T * position,
            size_t n,
            const T & val )  [inline]
```

5.1.2.15 last()

```
template<typename T >
T & myVector< T >::last ( )  [inline]
```

5.1.2.16 operator"!=()

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

5.1.2.17 operator<()

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

5.1.2.18 operator<=()

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

5.1.2.19 operator=() [1/2]

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

5.1.2.20 operator=() [2/2]

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

5.1.2.21 operator==()

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

5.1.2.22 operator>()

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

5.1.2.23 operator>=()

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

5.1.2.24 operator[]()

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

5.1.2.25 pop_back()

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

5.1.2.26 print()

```
template<typename T >
void myVector< T >::print ( )  [inline]
```

5.1.2.27 push_back()

```
template<typename T >
void myVector< T >::push_back (
            T duomenys )  [inline]
```

5.1.2.28 reserve()

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

5.1.2.29 resize()

```
template<typename T >
void myVector< T >::resize (
            size_t n )  [inline]
```

5.1.2.30 size()

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

5.1.2.31 swap()

```
template<typename T >
void myVector< T >::swap (
            myVector< T > & other )  [inline]
```

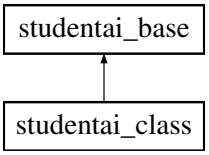
The documentation for this class was generated from the following file:

- myVector.h

5.2 studentai_base Class Reference

`#include <funkcijuBazeVektoriai.h>`

Inheritance diagram for studentai_base:



Public Member Functions

- virtual void setMasyvas (int a, int b)
- virtual void testav ()=0
- studentai_base ()
- virtual ~studentai_base ()

Protected Attributes

- string vardas_
- string pavarde_
- myVector< int > balai_
- int egzaminas_
- double vidurkis_
- double mediana_
- int * rodykle_
- int dydis_

5.2.1 Constructor & Destructor Documentation

5.2.1.1 studentai_base()

```
studentai_base::studentai_base ( )  [inline]
```

5.2.1.2 ~studentai_base()

```
virtual studentai_base::~studentai_base ( )  [inline], [virtual]
```

5.2.2 Member Function Documentation

5.2.2.1 setMasyvas()

```
virtual void studentai_base::setMasyvas (
            int a,
            int b )  [inline], [virtual]
```

5.2.2.2 testav()

```
virtual void studentai_base::testav ( )  [pure virtual]
```

Implemented in studentai_class.

5.2.3 Member Data Documentation

5.2.3.1 balai_

```
myVector<int> studentai_base::balai_  [protected]
```

5.2.3.2 dydis_

```
int studentai_base::dydis_  [protected]
```

5.2.3.3 egzaminas_

```
int studentai_base::egzaminas_  [protected]
```

5.2.3.4 mediana_

```
double studentai_base::mediana_  [protected]
```

5.2.3.5 pavarde_

```
string studentai_base::pavarde_  [protected]
```

5.2.3.6 rodykle_

```
int* studentai_base::rodykle_  [protected]
```

5.2.3.7 vardas_

```
string studentai_base::vardas_  [protected]
```

5.2.3.8 vidurkis_

```
double studentai_base::vidurkis_  [protected]
```

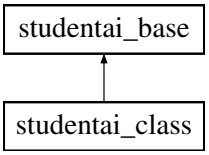
The documentation for this class was generated from the following file:

- funkcijuBazeVektoriai.h

5.3 studentai_class Class Reference

```
#include <funkcijuBazeVektoriai.h>
```

Inheritance diagram for studentai_class:



Public Member Functions

- studentai_class ()
- studentai_class (string vardas, string pavarde, myVector< int > balai, int egzaminas, double vidurkis, double mediana)
- ~studentai_class ()
- studentai_class (int dydis)
- void testav ()
- void setVardas (string a)
- void setPavarde (string a)
- void setBalai (myVector< int > a)
- void setEgzaminas (int a)
- void setVidurkis (double a)
- void setMediana (double a)
- string getVardas ()
- string getPavarde ()
- myVector< int > getBalai ()
- int getEgzaminas ()
- double getVidurkis ()
- double getMediana ()
- studentai_class (const studentai_class &a)
- studentai_class operator= (const studentai_class &a)
- studentai_class (studentai_class &&a)
- studentai_class operator= (studentai_class &&a)

Public Member Functions inherited from studentai_base

- virtual void setMasyvas (int a, int b)
- studentai_base ()
- virtual ~studentai_base ()

Friends

- ostream & operator<< (ostream &out, studentai_class a)
- istream & operator>> (istream &in, studentai_class &a)

Additional Inherited Members

Protected Attributes inherited from studentai_base

- string vardas_
- string pavarde_
- myVector< int > balai_
- int egzaminas_
- double vidurkis_
- double mediana_
- int * rodykle_
- int dydis_

5.3.1 Constructor & Destructor Documentation

5.3.1.1 studentai_class() [1/5]

```
studentai_class::studentai_class ( )
```

5.3.1.2 studentai_class() [2/5]

```
studentai_class::studentai_class (
            string vardas,
            string pavarde,
            myVector< int > balai,
            int egzaminas,
            double vidurkis,
            double mediana )
```

5.3.1.3 ~studentai_class()

```
studentai_class::~studentai_class ( )  [inline]
```

5.3.1.4 studentai_class() [3/5]

```
studentai_class::studentai_class (
            int dydis )
```

5.3.1.5 studentai_class() [4/5]

```
studentai_class::studentai_class (
            const studentai_class & a )
```

5.3.1.6 studentai_class() [5/5]

```
studentai_class::studentai_class (
            studentai_class && a )
```

5.3.2 Member Function Documentation

5.3.2.1 getBalai()

```
myVector< int > studentai_class::getBalai ( )  [inline]
```

5.3.2.2 getEgzaminas()

```
int studentai_class::getEgzaminas ( )  [inline]
```

5.3.2.3 getMediana()

```
double studentai_class::getMediana ( )  [inline]
```

5.3.2.4 getPavarde()

```
string studentai_class::getPavarde ( )  [inline]
```

5.3.2.5 getVardas()

```
string studentai_class::getVardas ( )  [inline]
```

5.3.2.6 getVidurkis()

```
double studentai_class::getVidurkis ( )  [inline]
```

5.3.2.7 operator=() [1/2]

```
studentai_class studentai_class::operator= (
            const studentai_class & a )
```

5.3.2.8 operator=() [2/2]

```
studentai_class studentai_class::operator= (
            studentai_class && a )
```

5.3.2.9 setBalai()

```
void studentai_class::setBalai (
            myVector< int > a )  [inline]
```

5.3.2.10 setEgzaminas()

```
void studentai_class::setEgzaminas (
            int a )  [inline]
```

5.3.2.11 setMediana()

```
void studentai_class::setMediana (
            double a )  [inline]
```

5.3.2.12 setPavarde()

```
void studentai_class::setPavarde (
            string a )  [inline]
```

5.3.2.13 setVardas()

```
void studentai_class::setVardas (
            string a )  [inline]
```

5.3.2.14 setVidurkis()

```
void studentai_class::setVidurkis (
            double a )  [inline]
```

5.3.2.15 testav()

```
void studentai_class::testav ( )  [virtual]
```

Implements studentai_base.

5.3.3 Friends And Related Symbol Documentation

5.3.3.1 operator<<

```
ostream & operator<< (
            ostream & out,
            studentai_class a )  [friend]
```

5.3.3.2 operator>>

```
istream & operator>> (
            istream & in,
            studentai_class & a )  [friend]
```

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

- funkcijuBazeVektoriai.h
- klasesRealizacija.cpp

Chapter 6

File Documentation

6.1 bendrosFunkcijos.cpp File Reference

```
#include "funkcijuBaze.h"
```

Functions

- string tarpai (string a, int tarpuDydis)
- string extraSpace (string a, int b)
- mt19937 mt (rd())
- uniform_int_distribution< int > distribution (1, 10)
- void failoGeneracija ()
- void laikoSpausdinimas (duration< double > readTime, duration< double > sortTime, duration< double > typeTime)

Variables

- random_device rd

6.1.1 Function Documentation

6.1.1.1 distribution()

```
uniform_int_distribution< int > distribution (
            1 ,
            10  )
```

6.1.1.2 extraSpace()

```
string extraSpace (
            string a,
            int b )
```

6.1.1.3 failoGeneracija()

```
void failoGeneracija ( )
```

6.1.1.4 laikoSpausdinimas()

```
void laikoSpausdinimas (
            duration< double > readTime,
            duration< double > sortTime,
            duration< double > typeTime )
```

6.1.1.5 mt()

```
mt19937 mt (
            rd()  )
```

6.1.1.6 tarpai()

```
string tarpai (
            string a,
            int tarpuDydis )
```

6.1.2 Variable Documentation

6.1.2.1 rd

```
random_device rd
```

6.2 funkcijosVektoriai.cpp File Reference

```
#include "funkcijuBazeVektoriai.h"
#include "myVector.h"
```

Functions

- void darbasSuVektoriais ()
- void vektoriuTestavimas ()
- void klasiuTestavimas ()
- double medianosApsk (myVector< int > a, int egzaminas)
- double vidurkioApsk (myVector< int > a, int egzaminas)
- void pirmasPasirinkimas ()
- void antrasPasirinkimas ()
- void treciasPasirinkimas ()
- void NuskaitymasFailo (string fileName)
- void spausdinimasFaile ()
- void spausdinimasTerminale ()
- void spausdinimasFaileSkirstymas ()
- void spausdinimasTerminaleSkirstymas ()
- bool rusiavimasVardas (studentai_class &a, studentai_class &b)
- bool rusiavimasPavarde (studentai_class &a, studentai_class &b)
- bool rusiavimasMediana (studentai_class &a, studentai_class &b)
- bool rusiavimasVidurkis (studentai_class &a, studentai_class &b)
- void rusiavimoMenu ()
- void rusiavimoMenuSkirstymas ()
- bool Less (studentai_class a)
- bool LessM (studentai_class a)
- void skirstymas ()

6.2.1 Function Documentation

6.2.1.1 antrasPasirinkimas()

```
void antrasPasirinkimas ( )
```

6.2.1.2 darbasSuVektoriais()

```
void darbasSuVektoriais ( )
```

6.2.1.3 klasiuTestavimas()

```
void klasiuTestavimas ( )
```

6.2.1.4 Less()

```
bool Less (
            studentai_class a )
```

6.2.1.5 LessM()

```
bool LessM (
            studentai_class a )
```

6.2.1.6 medianosApsk()

```
double medianosApsk (
            myVector< int > a,
            int egzaminas )
```

6.2.1.7 NuskaitymasFailo()

```
void NuskaitymasFailo (
            string fileName )
```

6.2.1.8 pirmasPasirinkimas()

```
void pirmasPasirinkimas ( )
```

6.2.1.9 rusiavimasMediana()

```
bool rusiavimasMediana (
            studentai_class & a,
            studentai_class & b )
```

6.2.1.10 rusiavimasPavarde()

```
bool rusiavimasPavarde (
            studentai_class & a,
            studentai_class & b )
```

6.2.1.11 rusiavimasVardas()

```
bool rusiavimasVardas (
            studentai_class & a,
            studentai_class & b )
```

6.2.1.12 rusiavimasVidurkis()

```
bool rusiavimasVidurkis (
            studentai_class & a,
            studentai_class & b )
```

6.2.1.13 rusiavimoMenu()

```
void rusiavimoMenu ( )
```

6.2.1.14 rusiavimoMenuSkirstymas()

```
void rusiavimoMenuSkirstymas ( )
```

6.2.1.15 skirstymas()

```
void skirstymas ( )
```

6.2.1.16 spausdinimasFaile()

```
void spausdinimasFaile ( )
```

6.2.1.17 spausdinimasFaileSkirstymas()

```
void spausdinimasFaileSkirstymas ( )
```

6.2.1.18 spausdinimasTerminale()

```
void spausdinimasTerminale ( )
```

6.2.1.19 spausdinimasTerminaleSkirstymas()

```
void spausdinimasTerminaleSkirstymas ( )
```

6.2.1.20 treciasPasirinkimas()

```
void treciasPasirinkimas ( )
```

6.2.1.21 vektoriuTestavimas()

```
void vektoriuTestavimas ( )
```

6.2.1.22 vidurkioApsk()

```
double vidurkioApsk (
            myVector< int > a,
            int egzaminas )
```

6.3 FunkcijuBaze.h File Reference

```
#include "includes.h"
```

Functions

- void darbasSuVektoriais ()
- void darbasSuDekais ()
- void darbasSuListais ()
- string tarpai (string a, int tarpuDydis)
- string extraSpace (string a, int b)
- void failoGeneracija ()

6.3.1 Function Documentation

6.3.1.1 darbasSuDekais()

```
void darbasSuDekais ( )
```

6.3.1.2 darbasSuListais()

```
void darbasSuListais ( )
```

6.3.1.3 darbasSuVektoriais()

```
void darbasSuVektoriais ( )
```

6.3.1.4 extraSpace()

```
string extraSpace (
            string a,
            int b )
```

6.3.1.5 failoGeneracija()

```
void failoGeneracija ( )
```

6.3.1.6 tarpai()

```
string tarpai (
            string a,
            int tarpuDydis )
```

6.4 FunkcijuBaze.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "includes.h"
00003
00004 //funkcijos main.cpp
00005 void darbasSuVektoriais();
00006 void darbasSuDekais();
00007 void darbasSuListais();
00008
00009 //papildomos funkcijos darbui
00010 string tarpai(string a, int tarpuDydis);
00011 string extraSpace (string a, int b);
00012
00013 void failoGeneracija();
```

6.5 funkcijuBazeVektoriai.h File Reference

```
#include "includes.h"
#include "myVector.h"
```

Classes

- class studentai_base
- class studentai_class

Functions

- string tarpai (string a, int tarpuDydis)
- double vidurkioApsk (myVector< int > a, int egzaminas)
- double medianosApsk (myVector< int > a, int egzaminas)
- string extraSpace (string a, int b)
- void vektoriuTestavimas ()
- void klasiuTestavimas ()
- void pirmasPasirinkimas ()
- void antrasPasirinkimas ()
- void treciasPasirinkimas ()
- void NuskaitymasFailo (string fileName)
- void failoGeneracija ()
- void skirstymas ()
- void rusiavimoMenu ()
- void rusiavimoMenuSkirstymas ()
- bool rusiavimasVardas (studentai_class &a, studentai_class &b)
- bool rusiavimasPavarde (studentai_class &a, studentai_class &b)
- bool rusiavimasVidurkis (studentai_class &a, studentai_class &b)
- bool rusiavimasMediana (studentai_class &a, studentai_class &b)
- void spausdinimasFaile ()
- void spausdinimasTerminale ()
- void spausdinimasFaileSkirstymas ()
- void spausdinimasTerminaleSkirstymas ()
- void laikoSpausdinimas (duration< double > readTime, duration< double > sortTime, duration< double > typeTime)

6.5.1 Function Documentation

6.5.1.1 antrasPasirinkimas()

```
void antrasPasirinkimas ( )
```

6.5.1.2 extraSpace()

```
string extraSpace (
            string a,
            int b )
```

6.5.1.3 failoGeneracija()

```
void failoGeneracija ( )
```

6.5.1.4 klasiuTestavimas()

```
void klasiuTestavimas ( )
```

6.5.1.5 laikoSpausdinimas()

```
void laikoSpausdinimas (
            duration< double > readTime,
            duration< double > sortTime,
            duration< double > typeTime )
```

6.5.1.6 medianosApsk()

```
double medianosApsk (
            myVector< int > a,
            int egzaminas )
```

6.5.1.7 NuskaitymasFailo()

```
void NuskaitymasFailo (
            string fileName )
```

6.5.1.8 pirmasPasirinkimas()

```
void pirmasPasirinkimas ( )
```

6.5.1.9 rusiavimasMediana()

```
bool rusiavimasMediana (
            studentai_class & a,
            studentai_class & b )
```

6.5.1.10 rusiavimasPavarde()

```
bool rusiavimasPavarde (
            studentai_class & a,
            studentai_class & b )
```

6.5.1.11 rusiavimasVardas()

```
bool rusiavimasVardas (
            studentai_class & a,
            studentai_class & b )
```

6.5.1.12 rusiavimasVidurkis()

```
bool rusiavimasVidurkis (
            studentai_class & a,
            studentai_class & b )
```

6.5.1.13 rusiavimoMenu()

```
void rusiavimoMenu ( )
```

6.5.1.14 rusiavimoMenuSkirstymas()

```
void rusiavimoMenuSkirstymas ( )
```

6.5.1.15 skirstymas()

```
void skirstymas ( )
```

6.5.1.16 spausdinimasFaile()

```
void spausdinimasFaile ( )
```

6.5.1.17 spausdinimasFaileSkirstymas()

```
void spausdinimasFaileSkirstymas ( )
```

6.5.1.18 spausdinimasTerminale()

```
void spausdinimasTerminale ( )
```

6.5.1.19 spausdinimasTerminaleSkirstymas()

```
void spausdinimasTerminaleSkirstymas ( )
```

6.5.1.20 tarpai()

```
string tarpai (
            string a,
            int tarpuDydis )
```

6.5.1.21 treciasPasirinkimas()

```
void treciasPasirinkimas ( )
```

6.5.1.22 vektoriuTestavimas()

```
void vektoriuTestavimas ( )
```

6.5.1.23 vidurkioApsk()

```
double vidurkioApsk (
            myVector< int > a,
            int egzaminas )
```

6.6 funkcijuBazeVektoriai.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "includes.h"
00003 #include "myVector.h"
00004 // struct studentai{
00005 //     string vardas;
00006 //     string pavarde;
00007 //     myVector<int> balai;
00008 //     int egzaminas;
00009 //     double vidurkis;
00010 //     double mediana;
00011 // };
00012 string tarpai(string a, int tarpuDydis);// turi buti pirmiau uz klase del perdengto operatoriaus «
00013 double vidurkioApsk(myVector<int> a, int egzaminas);
00014 double medianosApsk(myVector<int> a, int egzaminas);
00015
00016
00017 class studentai_base{
00018     protected:
00019         string vardas_;
00020         string pavarde_;
00021         myVector<int> balai_;
00022         int egzaminas_;
00023         double vidurkis_;
```

```
00024         double mediana_;
00025         int* rodykle_;
00026         int dydis_;
00027     public:
00028         virtual void setMasyvas(int a, int b){rodykle_[a] = b;};
00029         virtual void testav() = 0;
00030         studentai_base(){};//default konstruktorius
00031         virtual ~studentai_base(){};
00032 };
00033
00034
00035
00036
00037 class studentai_class : public studentai_base{
00038     public:
00039         studentai_class();
00040         studentai_class(string vardas, string pavarde, myVector<int> balai, int egzaminas, double
      vidurkis, double mediana);
00041         ~studentai_class(){delete[] rodykle_;};
00042         //testavimui
00043         studentai_class(int dydis);
00044
00045         void testav();
00046         void setVardas(string a){vardas_ = a;};
00047         void setPavarde(string a){pavarde_ = a;};
00048         void setBalai(myVector<int> a){balai_ = a;};
00049         void setEgzaminas(int a){egzaminas_ = a;};
00050         void setVidurkis(double a){vidurkis_ = a;};
00051         void setMediana(double a){mediana_ = a;};
00052
00053         string getVardas(){return vardas_;};
00054         string getPavarde(){return pavarde_;};
00055         myVector<int> getBalai(){return balai_;};
00056         int getEgzaminas(){return egzaminas_;};
00057         double getVidurkis(){return vidurkis_;};
00058         double getMediana(){return mediana_;};
00059         friend ostream& operator<<(ostream& out, studentai_class a);//output overloading
00060         friend istream& operator>>(istream& in, studentai_class &a);//input
00061
00062         studentai_class (const studentai_class& a);  // kopiviavimo konstruktorius
00063         studentai_class operator=(const studentai_class& a); // kopiviavimo operatorius / priskirimas
00064         studentai_class (studentai_class&& a); // move konstruktorius
00065         studentai_class operator=(studentai_class&& a); // move operatorius
00066 };
00067 string extraSpace (string a, int b);
00068
00069 void vektoriuTestavimas();
00070 void klasiuTestavimas();
00071 //Vektorius
00072 void pirmasPasirinkimas();
00073 void antrasPasirinkimas();
00074 void treciasPasirinkimas();
00075
00076 void NuskaitymasFailo(string fileName);
00077 void failoGeneracija();
00078
00079 void skirstymas();
00080
00081 void rusiavimoMenu();
00082 void rusiavimoMenuSkirstymas();
00083
00084 bool rusiavimasVardas(studentai_class &a, studentai_class &b);
00085 bool rusiavimasPavarde(studentai_class &a, studentai_class &b);
00086 bool rusiavimasVidurkis(studentai_class &a, studentai_class &b);
00087 bool rusiavimasMediana(studentai_class &a, studentai_class &b);
00088
00089 void spausdinimasFaile();
00090 void spausdinimasTerminale();
00091 void spausdinimasFaileSkirstymas();
00092 void spausdinimasTerminaleSkirstymas();
00093
00094 void laikoSpausdinimas(duration<double> readTime, duration<double> sortTime ,duration<double>
      typeTime);
00095
```

6.7 Includes.h File Reference

```
#include <iostream>
#include <iomanip>
#include <algorithm>
```

```
#include <ctime>
#include <fstream>
#include <chrono>
#include <random>
#include <sstream>
```

Variables

- random_device rd
- mt19937 mt
- uniform_int_distribution< int > distribution

6.7.1 Variable Documentation

6.7.1.1 distribution

```
uniform_int_distribution<int> distribution  [extern]
```

6.7.1.2 mt

```
mt19937 mt  [extern]
```

6.7.1.3 rd

```
random_device rd  [extern]
```

6.8 Includes.h

Go to the documentation of this file.

```
00001 #ifndef include
00002 #define include
00003 #include <iostream>
00004 #include <iomanip>
00005 #include <algorithm>
00006 #include <ctime>
00007 #include <fstream>
00008 #include <chrono>
00009 #include <random>
00010 #include <sstream>
00011
00012 using namespace std;
00013 using namespace std::chrono;
00014
00015 extern random_device rd;
00016 extern mt19937 mt;
00017 extern uniform_int_distribution<int> distribution;
00018 #endif
00019 // vidurkio ir medianos funkcijos visos tos pacios, gal reiks pakeisti
```

6.9 klasesRealizacija.cpp File Reference

```
#include "funkcijuBazeVektoriai.h"
#include "myVector.h"
```

Functions

- ostream & operator<< (ostream &out, studentai_class a)
- istream & operator>> (istream &in, studentai_class &a)

6.9.1 Function Documentation

6.9.1.1 operator<<()

```
ostream & operator<< (
            ostream & out,
            studentai_class a )
```

6.9.1.2 operator>>()

```
istream & operator>> (
            istream & in,
            studentai_class & a )
```

6.10 main.cpp File Reference

```
#include "funkcijuBaze.h"
```

Functions

- int main ()

6.10.1 Function Documentation

6.10.1.1 main()

```
int main ( )
```

6.11 myVector.h File Reference

```
#include "Includes.h"
```

Classes

- class myVector< T >

6.12 myVector.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Includes.h"
00003 template <typename T> class myVector{
00004     private:
00005         T* arr;
00006         int capacity;
00007         int currentSize;
00008 
00009     public:
00010     //creates an empty vector with capacity of 1
00011     myVector(){
00012         arr = new T[1];
00013         capacity = 1;
00014         currentSize = 0;
00015     }
00016 
00017     //creates a vector and fills it
00018     myVector(size_t value, T data){
00019         arr = new T[value];
00020         for (int i = 0; i < value; i++){
00021             arr[i] = data;
00022         }
00023         capacity = value;
00024         currentSize = value;
00025     }
00026     // Initializer list constructor
00027     myVector(std::initializer_list<T> ilist) : arr(new T[1]), capacity(1), currentSize(0) {
00028         reserve(ilist.size());
00029         for (const auto& elem : ilist) {
00030             push_back(elem);
00031         }
00032     }
00033     myVector(T data){
00034         arr = new T[1];
00035         arr[0] = data;
00036         capacity = 1;
00037         currentSize = 1;
00038     }
00039 
00040     // Copy constructor
00041     myVector(const myVector& other) : arr(nullptr), capacity(0), currentSize(0) {
00042         reserve(other.currentSize);
00043         std::copy(other.arr, other.arr + other.currentSize, arr);
00044         currentSize = other.currentSize;
00045     }
00046 
00047     // Move constructor
00048     myVector(myVector&& other) noexcept : arr(other.arr), capacity(other.capacity),
      currentSize(other.scurrentSizeize) {
00049         other.arr = nullptr;
00050         other.capacity = 0;
00051         other.currentSize = 0;
00052     }
00053 
00054   // Copy assignment operator
00055     myVector& operator=(const myVector& other) {
00056         if (this == &other) {
00057             return *this;
00058         }
00059         T* new_arr = new T[other.capacity];
00060         std::copy(other.arr, other.arr + other.currentSize, new_arr);
00061         delete[] arr;
00062         arr = new_arr;
00063         capacity = other.capacity;
00064         currentSize = other.currentSize;
00065         return *this;
00066     }
00067 
00068     // Move assignment operator
00069     myVector& operator=(myVector&& other) noexcept {
00070         if (this == &other) {
00071             return *this;
00072         }
00073         delete[] arr;
00074         arr = other.arr;
00075         capacity = other.capacity;
00076         currentSize = other.currentSize;
00077         other.arr = nullptr;
00078         other.capacity = 0;
00079         other.currentSize = 0;
00080         return *this;
00081     }
```

```
00082
00083     ~myVector(){
00084         delete [] arr;
00085     }
00086     void reserve(size_t new_capacity) {
00087         if (new_capacity > capacity) {
00088             T* new_arr = new T[new_capacity];
00089             for (size_t i = 0; i < currentSize; ++i) {
00090                 new_arr[i] = arr[i];
00091             }
00092             delete[] arr;
00093             arr = new_arr;
00094             capacity = new_capacity;
00095         }
00096     }
00097     void push_back(T duomenys){
00098
00099         // if capacity reached, double size
00100         if (currentSize == capacity) {
00101             T* temp = new T[2 * capacity];
00102
00103             // copy elements
00104             for (int i = 0; i < capacity; i++) {
00105                 temp[i] = arr[i];
00106             }
00107
00108             // deleting previous array
00109             delete[] arr;
00110             capacity *= 2;
00111             arr = temp;
00112         }
00113
00114         // Adding data
00115         arr[currentSize] = duomenys;
00116         currentSize++;
00117     }
00118
00119     // function to delete last element
00120     void pop_back(){
00121         currentSize--;
00122     }
00123
00124
00125     T at(size_t index){
00126         if (index < currentSize){
00127             return arr[index];
00128         }
00129         else{
00130             throw std::out_of_range("Vector out of range");
00131         }
00132     }
00133
00134
00135     // function to get size of the vector
00136     int size() const{
00137         return currentSize;
00138     }
00139     // function to get capacity of the vector
00140     int getCapacity(){
00141         return capacity;
00142     }
00143
00144
00145     // function to print out all array elements
00146     void print()
00147     {
00148         for (int i = 0; i < currentSize; i++) {
00149             cout « arr[i] « " ";
00150         }
00151         cout « endl;
00152     }
00153     // check if empty
00154     bool empty(){
00155         if (currentSize == 0){
00156             return true;
00157         }else{
00158             return false;
00159         }
00160     }
00161
00162
00163     // clears vector
00164     void clear(){
00165         currentSize = 0;
00166     }
00167
00168  // Access the first element
```

```
00169     T& first() {
00170         if (currentSize == 0) {
00171             throw std::out_of_range("Vector is empty");
00172         }
00173         return arr[0];
00174     }
00175 
00176     // Access the last element
00177     T& last() {
00178         if (currentSize == 0) {
00179             throw std::out_of_range("Vector is empty");
00180         }
00181         return arr[currentSize - 1];
00182     }
00183 
00184     // Iterator to the beginning
00185     T* begin() {
00186         return arr;
00187     }
00188 
00189     // Iterator to the end
00190     T* end() {
00191         return arr + currentSize;
00192     }
00193 
00194 
00195 
00196     T& operator[] (size_t index) const{
00197         if (index >= capacity) {
00198             throw std::out_of_range("Index out of rangee");
00199         }
00200         return arr[index];
00201     }
00202     // Assign function to assign multiple copies of a value
00203     void assign( size_t count, const T& value ){
00204         clear();
00205         for (int i = 0; i < count; i++){
00206             arr[i] = value;
00207         }
00208     }
00209 
00210    // Assign function to assign a range of elements
00211     template <typename InputIterator>
00212     void assign(InputIterator first, InputIterator last) {
00213         size_t new_size = std::distance(first, last);
00214         if (new_size > capacity) {
00215             reserve(new_size);
00216         }
00217         currentSize = new_size;
00218         std::copy(first, last, arr);
00219     }
00220 
00221     // Erase a single element at position
00222     void erase(T* position) {
00223         if (position < arr || position >= arr + currentSize) {
00224             throw std::out_of_range("Position out of range");
00225         }
00226         std::copy(position + 1, arr + currentSize, position);
00227         --currentSize;
00228     }
00229 
00230     // Erase elements in range
00231     void erase(T* first, T* last) {
00232         if (first < arr || last > arr + currentSize || first > last) {
00233             throw std::out_of_range("Range out of range");
00234         }
00235         std::copy(last, arr + currentSize, first);
00236         currentSize -= (last - first);
00237     }
00238 
00239 
00240     // Swap the contents with another vector
00241     void swap(myVector& other) {
00242         std::swap(arr, other.arr);
00243         std::swap(capacity, other.capacity);
00244         std::swap(currentSize, other.currentSize);
00245     }
00246 
00247     // Resize the vector to contain n elements
00248     void resize(size_t n) {
00249         if (n > capacity) {
00250             reserve(n);
00251         }
00252         if (n > currentSize) {
00253             std::fill(arr + currentSize, arr + n, T());
00254         }
00255         currentSize = n;
```

```
00256     }
00257         // Insert a single element at position
00258     T* insert(T* position, const T& val) {
00259         size_t index = position - arr;
00260         if (currentSize == capacity) {
00261             reserve(capacity == 0 ? 1 : 2 * capacity);
00262         }
00263         position = arr + index;
00264         std::copy_backward(position, arr + currentSize, arr + currentSize + 1);
00265         *position = val;
00266         ++currentSize;
00267         return position;
00268     }
00269 
00270     // Insert multiple elements at position
00271     void insert(T* position, size_t n, const T& val) {
00272         size_t index = position - arr;
00273         if (currentSize + n > capacity) {
00274             reserve(currentSize + n);
00275         }
00276         position = arr + index;
00277         std::copy_backward(position, arr + currentSize, arr + currentSize + n);
00278         std::fill(position, position + n, val);
00279         currentSize += n;
00280     }
00281 
00282     // Insert elements from range [first, last) at position
00283     template <typename InputIterator>
00284     void insert(T* position, InputIterator first, InputIterator last) {
00285         size_t index = position - arr;
00286         size_t n = std::distance(first, last);
00287         if (currentSize + n > capacity) {
00288             reserve(currentSize + n);
00289         }
00290         position = arr + index;
00291         std::copy_backward(position, arr + currentSize, arr + currentSize + n);
00292         std::copy(first, last, position);
00293         currentSize += n;
00294     }
00295 
00296 
00297     // Comparison operators
00298     bool operator==(const myVector& other) const {
00299         if (currentSize != other.currentSize) return false;
00300         for (size_t i = 0; i < currentSize; ++i) {
00301             if (arr[i] != other.arr[i]) return false;
00302         }
00303         return true;
00304     }
00305 
00306     bool operator!=(const myVector& other) const {
00307         return !(*this == other);
00308     }
00309 
00310     bool operator<(const myVector& other) const {
00311         return std::lexicographical_compare(arr, arr + currentSize, other.arr, other.arr +
    other.currentSize);
00312     }
00313 
00314     bool operator<=(const myVector& other) const {
00315         return !(other < *this);
00316     }
00317 
00318     bool operator>(const myVector& other) const {
00319         return other < *this;
00320     }
00321 
00322     bool operator>=(const myVector& other) const {
00323         return !(*this < other);
00324     }
00325 };
```

6.13 readMe.md File Reference

Index