

UniversityCourseV3

v3.0

Generated by Doxygen 1.10.0

1 UniversityCourseV3	1
1.1 Efektyvumo/spartos analizė	1
1.2 Atminties perskirstymai	1
1.3 Spartos analizė	1
1.3.0.1 Failo nuskaitymas	1
1.3.0.2 Studentų rušiavimas į dvi grupes	2
1.3.0.3 Studentų rikiavimas didejimo tvarka	2
1.3.0.4 Studentų išvedimas į failą	2
1.4 System	2
1.5 vectorClass Functions	2
1.5.1 1. push_back	2
1.5.2 2. operator[]	3
1.5.3 3. resize	3
1.5.4 4. at	3
1.5.5 5. insert	4
2 Hierarchical Index	5
2.1 Class Hierarchy	5
3 Class Index	7
3.1 Class List	7
4 File Index	9
4.1 File List	9
5 Class Documentation	11
5.1 RandInt Class Reference	11
5.2 Studentas Class Reference	11
5.2.1 Member Function Documentation	12
5.2.1.1 doSomething()	12
5.2.1.2 Pavarde()	12
5.2.1.3 setPavarde()	12
5.2.1.4 setVardas()	12
5.2.1.5 Vardas()	13
5.3 vectorClass< T > Class Template Reference	13
5.4 Zmogus Class Reference	14
6 File Documentation	15
6.1 MainHeader.h	15
6.2 MainIncludes.h	15
6.3 RandInt.hpp	16
6.4 Studentas.h	16
6.5 vectorClass.h	16
6.6 Zmogus.h	17

Chapter 1

UniversityCourseV3

Objektinio programavimo paskutinioji užduotis. Sukurtas `vectorClass` konteineris padengiantis `std::vector` konteinerio metodus ir pilnai veikiantis su v1.5 programos versija.

1.1 Efektyvumo/spartos analizė

Number of `int` type elements filling vectors using `push_back()` function.

Number of elements	<code>std::vector</code>	<code>vectorClass</code>
10000	0.000152 seconds.	0.00009 seconds.
100000	0.0010375 seconds.	0.0007579 seconds.
1000000	0.0086524 seconds.	0.0059382 seconds.
10000000	0.0767485 seconds.	0.0648773 seconds.
100000000	0.717261 seconds.	0.543378 seconds.

1.2 Atminties perskirstymai

`vectorClass` ir `std::vector` užpildant vektorių 10.000.000 elementų įvyksta 27 atminties perskirstymai. Perskirstymas įvyksta tada, kai yra patenkinama sąlyga: `capacity() == size()`, t.y. kai nelieta vietos `capacity()` naujiems elementams.

1.3 Spartos analizė

1.3.0.1 Failo nuskaitymas

Studentų įrašų skaičius	<code>std::vector</code>	<code>vectorClass</code>
100.000	2.32839 seconds.	2.34903 seconds.
1.000.000	22.8893 seconds.	23.1926 seconds.
10.000.000	218.92 seconds.	233.481 seconds.

1.3.0.2 Studentų rušiavimas į dvi grupes

Studentų įrašų skaičius	std::vector	vectorClass
100.000	0.0363991 seconds.	0.0307202 seconds.
1.000.000	0.364148 seconds.	0.40457 seconds.
10.000.000	3.49419 seconds.	3.98185 seconds.

1.3.0.3 Studentų rikiavimas didėjimo tvarka

Studentų įrašų skaičius	std::vector	vectorClass
100.000	0.221569 seconds.	0.207835 seconds.
1.000.000	2.61153 seconds.	2.56135 seconds.
10.000.000	34.2261 seconds.	32.7805 seconds.

1.3.0.4 Studentų išvedimas į failą

Studentų įrašų skaičius	std::vector	vectorClass
100.000	0.736306 seconds.	0.882708 seconds.
1.000.000	7.33968 seconds.	7.32049 seconds.
10.000.000	72.7425 seconds.	70.5485 seconds.

1.4 System

All the test were ran on ryzen 5 5600x, 32gb ddr4 3600mhz, rtx 4060, 1m.2 samsung ssd.

1.5 vectorClass Functions

This part outlines some of the functions used in my custom `vectorClass`.

1.5.1 1. push_back

Code Example:

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

void test_push_back() {
    vectorClass<int> myVec;
    myVec.push_back(1);
    myVec.push_back(2);
    myVec.push_back(3);
    std::cout << "vectorClass: ";
    for (int i = 0; i < myVec.size(); ++i) {
        std::cout << myVec[i] << " ";
    }
    std::cout << std::endl;
}
```

```
int main() {
    test_push_back();
    system("pause");
}
```

Output:
std::vector: 1 2 3
vectorClass: 1 2 3

1.5.2 2. operator[]

Code Example:

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

void test_operator_index() {
    vectorClass<int> myVec;
    myVec.push_back(10);
    myVec.push_back(20);
    myVec.push_back(30);
    std::cout << "vectorClass: " << myVec[0] << " " << myVec[1] << " " << myVec[2] << std::endl;
}

int main() {
    test_operator_index();
    system("pause");
}

Output:
std::vector: 10 20 30
vectorClass: 10 20 30
```

1.5.3 3. resize

Code Example:

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

void test_resize() {
    vectorClass<int> myVec;
    myVec.push_back(1);
    myVec.push_back(2);
    myVec.push_back(3);
    myVec.resize(5);
    std::cout << "vectorClass (resize to 5): ";
    for (int i = 0; i < myVec.size(); ++i) {
        std::cout << myVec[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    test_resize();
    system("pause");
}

Output:
std::vector (resize to 5): 1 2 3 0 0
vectorClass (resize to 5): 1 2 3 0 0
```

1.5.4 4. at

Code Example:

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

void test_at() {
    vectorClass<int> myVec;
    myVec.push_back(100);
    myVec.push_back(200);
    myVec.push_back(300);
    try {
        std::cout << "vectorClass: " << myVec.at(1) << std::endl;
        std::cout << "vectorClass: " << myVec.at(3) << std::endl; // This will throw
    } catch (const std::out_of_range& e) {
        std::cout << "vectorClass out_of_range exception: " << e.what() << std::endl;
    }
}

int main() {
    test_at();
    system("pause");
}

Output:
std::vector: 200
std::vector out_of_range exception: vector::_M_range_check: __n (which is 3) >= this->size() (which is 3)
vectorClass: 200
vectorClass out_of_range exception: Index out of range
```

1.5.5 5. insert

Code Example:

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

void test_insert() {
    vectorClass<int> myVec;
    myVec.push_back(1);
    myVec.push_back(2);
    myVec.push_back(4);
    myVec.insert(2, 3);
    std::cout << "vectorClass: ";
    for (int i = 0; i < myVec.size(); ++i) {
        std::cout << myVec[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    test_insert();
    return 0;
}

Output:
std::vector: 1 2 3 4
vectorClass: 1 2 3 4
```


Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

RandInt	11
vectorClass< T >	13
Zmogus	14
Studentas	11

Chapter 3

Class Index

3.1 Class List

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

RandInt	11
Studentas	11
vectorClass< T >	13
Zmogus	14

Chapter 4

File Index

4.1 File List

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

MainHeader.h	15
MainIncludes.h	15
RandInt.hpp	16
Studentas.h	16
vectorClass.h	16
Zmogus.h	17

Chapter 5

Class Documentation

5.1 RandInt Class Reference

Public Member Functions

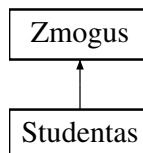
- **RandInt** (int low, int high)
- int **operator()** ()

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

- RandInt.hpp

5.2 Studentas Class Reference

Inheritance diagram for Studentas:



Public Member Functions

- **Studentas** (const std::string &vardas, const std::string &pavarde, double egzaminas, const std::vector< double > &namudarbass)
- **Studentas** (const **Studentas** &other)
- **Studentas** (**Studentas** &&other) noexcept
- **Studentas** & **operator=** (const **Studentas** &other)
- **Studentas** & **operator=** (**Studentas** &&other) noexcept
- std::string **Vardas** () const
- std::string **Pavarde** () const
- double **Egzaminas** () const
- std::vector< double > **Namudarbass** () const
- void **setVardas** (const std::string &vardas)
- void **setPavarde** (const std::string &pavarde)
- void **setEgzaminas** (double egzaminas)
- void **setNamudarbass** (const std::vector< double > &namudarbass)
- std::istream & **readStudent** (std::istream &)
- virtual void **doSomething** ()

Public Member Functions inherited from [Zmogus](#)

- **Zmogus** (const std::string &vardas, const std::string &pavarde)

Static Public Member Functions

- static double **Vidurkis** (const std::vector< double > &namudarbas)
- static double **Mediana** (const std::vector< double > &namudarbas)

Friends

- double **GalutinisVid** (const [Studentas](#) &duom)
- double **GalutinisMed** (const [Studentas](#) &duom)

Additional Inherited Members

Protected Attributes inherited from [Zmogus](#)

- std::string **Vardas_**
- std::string **Pavarde_**

5.2.1 Member Function Documentation

5.2.1.1 doSomething()

```
void Studentas::doSomething ( ) [virtual]
```

Implements [Zmogus](#).

5.2.1.2 Pavarde()

```
std::string Studentas::Pavarde ( ) const [inline], [virtual]
```

Reimplemented from [Zmogus](#).

5.2.1.3 setPavarde()

```
void Studentas::setPavarde (
    const std::string & pavarde ) [virtual]
```

Reimplemented from [Zmogus](#).

5.2.1.4 setVardas()

```
void Studentas::setVardas (
    const std::string & vardas ) [virtual]
```

Reimplemented from [Zmogus](#).

5.2.1.5 Vardas()

```
std::string Studentas::Vardas ( ) const [inline], [virtual]
```

Reimplemented from [Zmogus](#).

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

- Studentas.h
- Studentas.cpp

5.3 vectorClass< T > Class Template Reference

Public Types

- typedef T * **iterator**
- typedef const T * **const_iterator**
- typedef std::reverse_iterator< iterator > **reverse_iterator**
- typedef std::reverse_iterator< const_iterator > **const_reverse_iterator**

Public Member Functions

- **vectorClass** (const [vectorClass](#) &other)
- [vectorClass](#) & **operator=** (const [vectorClass](#) &other)
- **vectorClass** ([vectorClass](#) &&other) noexcept
- [vectorClass](#) & **operator=** ([vectorClass](#) &&other) noexcept
- void **assign** (int n, const T &val)
- T & **at** (int index)
- const T & **at** (int index) const
- T & **operator[]** (int index)
- const T & **operator[]** (int index) const
- T & **front** ()
- const T & **front** () const
- T & **back** ()
- const T & **back** () const
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () const noexcept
- bool **empty** () const
- int **size** () const
- int **max_size** () const
- void **reserve** (int new_capacity)
- int **get_capacity** () const
- void **shrink_to_fit** ()
- void **clear** ()
- void **insert** (int index, T data)

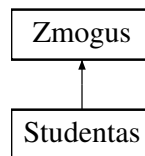
- void **erase** (int index)
- void **push_back** (T data)
- void **pop_back** ()
- void **resize** (int new_size)
- void **swap** (vectorClass< T > &other)

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

- vectorClass.h
- vectorClass.cpp

5.4 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** (const std::string &vardas, const std::string &pavarde)
- virtual std::string **Vardas** () const
- virtual std::string **Pavarde** () const
- virtual void **setVardas** (const std::string &vardas)
- virtual void **setPavarde** (const std::string &pavarde)
- virtual void **doSomething** ()=0

Protected Attributes

- std::string **Vardas_**
- std::string **Pavarde_**

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

- Zmogus.h

Chapter 6

File Documentation

6.1 MainHeader.h

```
00001 #ifndef HEADER_H // redefinition apsauga
00002 #define HEADER_H
00003
00004 #include "MainIncludes.h"
00005 #include "studentas.h"
00006
00007 void input1(Studentas& duom);
00008 void input2(Studentas& duom);
00009 void input3(Studentas& duom, int n);
00010 void OutputBy(const vector<Studentas>& student);
00011 void manualmode();
00012 void readingmode(const string& fileName);
00013 void filegeneration();
00014 void SplitVector(const vector<Studentas>& student);
00015 void SplitVector2(vector<Studentas>& student);
00016 void SplitVector3(vector<Studentas>& student);
00017 int NumberVerification(const string& prompt, int minValue, int maxValue);
00018 int NumberVerification(const string& prompt, int minValue);
00019 int YesNoVerification(const string& prompt);
00020
00021 #endif // HEADER_H
```

6.2 MainIncludes.h

```
00001 #ifndef INCLUDES_H // redefinition apsauga
00002 #define INCLUDES_H
00003
00004 // Standard Libraries
00005 #include <fstream>
00006 #include <iostream>
00007 #include <sstream>
00008 #include <vector>
00009 #include <iomanip>
00010 #include <string>
00011 #include <algorithm>
00012 #include <chrono>
00013 #include <cstdlib>
00014 #include <ctime>
00015 #include <random>
00016
00017 using namespace std;
00018 using namespace std::chrono;
00019
00020 // Custom Libraries
00021 #include "RandInt.hpp"
00022
00023 #endif // INCLUDES_H
```

6.3 RandInt.hpp

```

00001 #pragma once
00002
00003 class RandInt {
00004     public:
00005         RandInt(int low, int high) : mt{rd()}, dist{low, high} { }
00006         int operator()() { return dist(mt); } // generuok int'a
00007     private:
00008         std::random_device rd;
00009         std::mt19937 mt;
00010         std::uniform_int_distribution<int> dist;
00011 };

```

6.4 Studentas.h

```

00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
00003
00004 #include <algorithm>
00005 #include <string>
00006 #include <vector>
00007 #include <iostream>
00008 #include "Zmogus.h"
00009
00010 class Studentas : public Zmogus{
00011     private:
00012         double egzaminas_;
00013         std::vector<double> namudarbas_;
00014     public:
00015         // Default konstruktoriai
00016         Studentas() : Zmogus(), egzaminas_(0), namudarbas_({}) {}
00017         Studentas(const std::string& vardas, const std::string& pavarde, double egzaminas, const
00018             std::vector<double>& namudarbas) : Zmogus(vardas, pavarde), egzaminas_(egzaminas),
00019             namudarbas_(namudarbas) {}
00020
00021         // Rule of Five
00022         ~Studentas();
00023         Studentas(const Studentas& other); // Copy constructor
00024         Studentas(Studentas&& other) noexcept; // Move constructor
00025         Studentas& operator=(const Studentas& other); // Copy assignment
00026         Studentas& operator=(Studentas&& other) noexcept; // Move assignment
00027
00028         // Outputeriai
00029         std::string Vardas() const { return Vardas_; }
00030         std::string Pavarde() const { return Pavarde_; }
00031         double Egzaminas() const { return egzaminas_; }
00032         std::vector<double> Namudarbas() const { return namudarbas_; }
00033
00034         // Set'eriai
00035         void setVardas(const std::string& vardas);
00036         void setPavarde(const std::string& pavarde);
00037         void setEgzaminas(double egzaminas);
00038         void setNamudarbas(const std::vector<double>& namudarbas);
00039         std::istream& readStudent(std::istream&);
00040
00041         // Calculations
00042         friend double GalutinisVid(const Studentas& duom);
00043         friend double GalutinisMed(const Studentas& duom);
00044
00045         // Static functions for calculations
00046         static double Vidurkis(const std::vector<double>& namudarbas);
00047         static double Mediana(const std::vector<double>& namudarbas);
00048
00049         // Virtualios funkcijos deklaracija, kad klase Zmogus butu abstrakti
00050         virtual void doSomething();
00051 };
00052 #endif

```

6.5 vectorClass.h

```

00001 #ifndef VECTORCLASS_H
00002 #define VECTORCLASS_H
00003
00004 #include <iostream>
00005 #include <stdexcept>
00006 #include <limits>
00007

```

```

00008 template <typename T>
00009 class vectorClass {
00010     T* arr;
00011     int capacity;
00012     int current;
00013
00014 public:
00015     // Destructor and Constructor
00016     vectorClass();
00017     ~vectorClass();
00018
00019     // Member functions
00020     vectorClass(const vectorClass& other); // Copy constructor
00021     vectorClass& operator=(const vectorClass& other); // Copy assignment
00022     vectorClass(vectorClass&& other) noexcept; // Move constructor
00023     vectorClass& operator=(vectorClass&& other) noexcept; // Move assignment
00024     void assign(int n, const T& val);
00025     /* Truksta assing_range, get_allocator */
00026
00027     // Element access
00028     T& at(int index);
00029     const T& at(int index) const;
00030     T& operator[](int index);
00031     const T& operator[](int index) const;
00032     T& front();
00033     const T& front() const;
00034     T& back();
00035     const T& back() const;
00036     /* Truksta data */
00037
00038     // Iterators
00039     typedef T* iterator;
00040     typedef const T* const_iterator;
00041     typedef std::reverse_iterator<iterator> reverse_iterator;
00042     typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
00043     iterator begin() noexcept { return arr; }
00044     const_iterator begin() const noexcept { return arr; }
00045     iterator end() noexcept { return arr + current; }
00046     const_iterator end() const noexcept { return arr + current; }
00047     reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
00048     const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
00049     reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
00050     const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
00051
00052     // Capacity
00053     bool empty() const;
00054     int size() const;
00055     int max_size() const;
00056     void reserve(int new_capacity);
00057     int get_capacity() const;
00058     void shrink_to_fit();
00059
00060     // Modifiers
00061     void clear();
00062     void insert(int index, T data);
00063     void erase(int index);
00064     void push_back(T data);
00065     void pop_back();
00066     void resize(int new_size);
00067     void swap(vectorClass<T>& other);
00068     /* Truksta insert_range, append_range, emplace, emplace_back */
00069
00070     /* Isviso truksta: assing_range, get_allocator, data, insert_range, append_range, emplace,
    emplace_back */
00071     /* Realizuota 25 is 32 funkciju, kas yra 78% */
00072 };
00073 #endif // VECTORCLASS_H

```

6.6 Zmogus.h

```

00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003
00004 class Zmogus {
00005 protected:
00006     std::string Vardas_;
00007     std::string Pavarde_;
00008
00009 public:
00010     // Default konstruktorius
00011     Zmogus() : Vardas_(""), Pavarde_("") {}
00012     Zmogus(const std::string& vardas, const std::string& pavarde) : Vardas_(vardas), Pavarde_(pavarde)
    {}

```

```
00013
00014     // Virtualus destruktorius
00015     virtual ~Zmogus() {}
00016
00017     // Virtualus outputeriai
00018     virtual std::string Vardas() const { return Vardas_; }
00019     virtual std::string Pavarde() const { return Pavarde_; }
00020
00021     // Virtualus set'eriai
00022     virtual void setVardas(const std::string& vardas) { Vardas_ = vardas; }
00023     virtual void setPavarde(const std::string& pavarde) { Pavarde_ = pavarde; }
00024
00025     // Virtuali funkcija be realizacijos tam, kad klase Zmogus butu abstrakti
00026     virtual void doSomething() = 0;
00027 };
00028
00029 #endif
```

Index

- doSomething
 - Studentas, [12](#)
- Pavarde
 - Studentas, [12](#)
- RandInt, [11](#)
- setPavarde
 - Studentas, [12](#)
- setVardas
 - Studentas, [12](#)
- Studentas, [11](#)
 - doSomething, [12](#)
 - Pavarde, [12](#)
 - setPavarde, [12](#)
 - setVardas, [12](#)
 - Vardas, [12](#)
- UniversityCourseV3, [1](#)
- Vardas
 - Studentas, [12](#)
- vectorClass< T >, [13](#)
- Zmogus, [14](#)