UniversityCourseV3

v3.0

Generated by Doxygen 1.10.0

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()` funkction.

Number of elements	std::vector	vectorClass
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 nelieka vietos `capacity()` naujiems elementams.

1.3 Spartos analizė

1.3.0.1 Failo nuskaitymas

Studentų įrašų skaičius	std::vector	vectorClass
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 didejimo 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 i 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:

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 documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 RandInt Class Reference

Public Member Functions

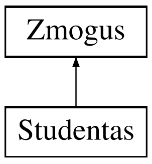
- **RandInt** (int low, int high)
- int **operator()** ()

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 > &namudarbas)
- **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 > **Namudarbas** () const
- void setVardas (const std::string &vardas)
- void setPavarde (const std::string &pavarde)
- void **setEgzaminas** (double egzaminas)
- void **setNamudarbas** (const std::vector< double > &namudarbas)
- 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)

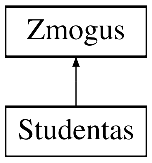
- void **erase** (int index)
- void **push_back** (T data)
- void **pop_back** ()
- void **resize** (int new_size)
- void **swap** (vectorClass< T > &other)

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

```
#ifndef HEADER_H // redefinition apsauga
#define HEADER_H

#include "MainIncludes.h"
#include "studentas.h"

void input1(Studentas& duom);
void input2(Studentas& duom);
void input3(Studentas& duom, int n);
void OutputBy(const vector<Studentas>& student);
void manualmode();
void readingmode(const string& fileName);
void filegeneration();
void SplitVector(const vector<Studentas>& student);
void SplitVector2(vector<Studentas>& student);
void SplitVector3(vector<Studentas>& student);
int NumberVerification(const string& prompt, int minValue, int maxValue);
int NumberVerification(const string& prompt, int minValue);
int YesNoVerification(const string& prompt);

#endif // HEADER_H
```

6.2 MainIncludes.h

```
#ifndef INCLUDES_H // redefinition apsauga
#define INCLUDES_H

// Standard Libraries
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>
#include <iomanip>
#include <string>
#include <algorithm>
#include <chrono>
#include <cstdlib>
#include <ctime>
#include <random>

using namespace std;
using namespace std::chrono;

// Custom Libraries
#include "RandInt.hpp"

#endif // INCLUDES_H
```

6.3 RandInt.hpp

```
#pragma once

class RandInt {
  public:
    RandInt(int low, int high) : mt{rd()}, dist{low, high} { }
    int operator()() { return dist(mt); } // generuok int'ą
  private:
    std::random_device rd;
    std::mt19937 mt;
    std::uniform_int_distribution<int> dist;
};
```

6.4 Studentas.h

```
#ifndef STUDENTAS_H
#define STUDENTAS_H

#include <algorithm>
#include <string>
#include <vector>
#include <iostream>
#include "Zmogus.h"

class Studentas : public Zmogus{
private:
    double egzaminas_;
    std::vector<double> namudarbas_;

public:
    // Default konstruktoriai
    Studentas() : Zmogus(), egzaminas_(0), namudarbas_({}) {}
    Studentas(const std::string& vardas, const std::string& pavarde, double egzaminas, const
std::vector<double>& namudarbas) : Zmogus(vardas, pavarde), egzaminas_(egzaminas),
namudarbas_(namudarbas) {}

    // Rule of Five
    ~Studentas();
    Studentas(const Studentas& other); // Copy constructor
    Studentas(Studentas&& other) noexcept; // Move constructor
    Studentas& operator=(const Studentas& other); // Copy assignment
    Studentas& operator=(Studentas&& other) noexcept; // Move assignment

    // Outputeriai
    std::string Vardas() const { return Vardas_; }
    std::string Pavarde() const { return Pavarde_; }
    double Egzaminas() const { return egzaminas_; }
    std::vector<double> Namudarbas() const { return namudarbas_; }

    // Set'eriai
    void setVardas(const std::string& vardas);
    void setPavarde(const std::string& pavarde);
    void setEgzaminas(double egzaminas);
    void setNamudarbas(const std::vector<double>& namudarbas);
    std::istream& readStudent(std::istream&);

    // Calculations
    friend double GalutinisVid(const Studentas& duom);
    friend double GalutinisMed(const Studentas& duom);

    // Static functions for calculations
    static double Vidurkis(const std::vector<double>& namudarbas);
    static double Mediana(const std::vector<double>& namudarbas);

    // Virtualios funkcijos deklaracija, kad klase Zmogus butu abstrakti
    virtual void doSomething();
};
#endif
```

6.5 vectorClass.h

```
#ifndef VECTORCLASS_H
#define VECTORCLASS_H

#include <iostream>
#include <stdexcept>
#include <limits>

```

```
template <typename T>
class vectorClass {
    T* arr;
    int capacity;
    int current;

public:
    // Destructor and Constructor
    vectorClass();
    ~vectorClass();

    // Member functions
    vectorClass(const vectorClass& other); // Copy constructor
    vectorClass& operator=(const vectorClass& other); // Copy assignment
    vectorClass(vectorClass&& other) noexcept; // Move constructor
    vectorClass& operator=(vectorClass&& other) noexcept; // Move assignment
    void assign(int n, const T& val);
    /* Truksta assing_range, get_allocator */

    // Element access
    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;
    /* Truksta data */

    // Iterators
    typedef T* iterator;
    typedef const T* const_iterator;
    typedef std::reverse_iterator<iterator> reverse_iterator;
    typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
    iterator begin() noexcept { return arr; }
    const_iterator begin() const noexcept { return arr; }
    iterator end() noexcept { return arr + current; }
    const_iterator end() const noexcept { return arr + current; }
    reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
    const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
    reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
    const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }

    // Capacity
    bool empty() const;
    int size() const;
    int max_size() const;
    void reserve(int new_capacity);
    int get_capacity() const;
    void shrink_to_fit();

    // Modifiers
    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);
    /* Truksta insert_range, append_range, emplace, emplace_back */

    /* Isviso truksta: assing_range, get_allocator, data, insert_range, append_range, emplace,
emplace_back */
    /* Realizuota 25 is 32 funciju, kas yra 78% */
};
#endif // VECTORCLASS_H
```

6.6 Zmogus.h

```
#ifndef ZMOGUS_H
#define ZMOGUS_H

class Zmogus {
protected:
    std::string Vardas_;
    std::string Pavarde_;

public:
    // Default konstruktorius
    Zmogus() : Vardas_(""), Pavarde_("") {}
    Zmogus(const std::string& vardas, const std::string& pavarde) : Vardas_(vardas), Pavarde_(pavarde)
{}
```

```cpp

    // Virtualus destruktorius
    virtual ~Zmogus() {}

    // Virtualus outputeriai
    virtual std::string Vardas() const { return Vardas_; }
    virtual std::string Pavarde() const { return Pavarde_; }

    // Virtualus set'eriai
    virtual void setVardas(const std::string& vardas) { Vardas_ = vardas; }
    virtual void setPavarde(const std::string& pavarde) { Pavarde_ = pavarde; }

    // Virtuali funkcija be realizacijos tam, kad klase Zmogus butu abstrakti
    virtual void doSomething() = 0;
};

#endif
```

Index