2.0

Generated by Doxygen 1.10.0

Chapter 1

Specs

```
CPU: AMD Ryzen 5 5600U with Radeon Graphics 2.30 GHz
RAM: 16.0 GB (15.3 GB usable)
SSD: WDC PC SN530 512Gb
GPU: Integrated with CPU
```

1.1 Structures vs Classes

Data from 1.1

Task	1000 students	10000 students	100000 students	1000000 students	10000000 students
Time taken to read data	0.006987000s	0.064826000s	0.6326890s	6.262007000s	64.8594s
Time taken to sort data	0.000997000s	0.042884000s	0.2950890s	3.173142000s	30.564123s
Time taken to divide students	0.000997000s	0.002991000s	0.0354300s	0.257002000s	2.654561s

Data with "Rule of Five"

Task	1000 students	10000 students	100000 students	1000000 students	10000000 students
Time taken to read data	0.006981000s	0.063828000s	0.628348000s	6.200408000s	64.8594s
Time taken to sort data	0.000952000s	0.048868000s	0.245346400s	2.10981s	22.098095s
Time taken to divide students	0.000000000s	0.002992000s	0.026930000s	0.34083900s	3.61350350s

1.2 Abstract Class "Zmogus"

An abstract class in C++ is a class that contains at least one pure virtual function. A pure virtual function is a virtual function for which we provide only the declaration in the base class, without providing any implementation. Abstract classes are designed to be used as base classes, and they cannot be instantiated directly. Instead, they are intended to serve as interfaces that define a common set of methods that derived classes must implement.

```
class Zmogus {
public:
    virtual void setVardas(std::string vardas) = 0;
    virtual std::string getVardas() const = 0;
    virtual void setPavarde(std::string pavarde) = 0;
    virtual std::string getPavarde() const = 0;
    virtual ~Zmogus() = default;

};
```

1.2.0.1 The key characteristics of an abstract class are:

- Contains Pure Virtual Functions: An abstract class contains at least one pure virtual function, which is declared with the virtual keyword and assigned the value 0 as its implementation.
- Cannot be Instantiated: Since abstract classes have at least one pure virtual function without an implementation, objects of abstract classes cannot be created directly. Attempting to create an instance of an abstract class will result in a compilation error.
- Used as Base Classes: Abstract classes are meant to be used as base classes. Derived classes inherit from abstract classes and provide concrete implementations for all the pure virtual functions defined in the abstract base class.

1.3 Rule of Five and Overloaded Methods

1.3.1 Rule of Five

In C++, the Rule of Five refers to a set of guidelines concerning resource management for classes that manage dynamic memory allocation or external resources. The Rule of Five consists of five special member functions that need to be defined or explicitly disabled if one of them is used:

1.3.1.1 Destructor

Responsible for releasing resources acquired by the object.

```
Studentas::~Studentas() {
nd_rezultatai.clear();
vardas.clear();
pavarde.clear();
egzaminas = 0;


}
```

1.3.1.2 Copy Constructor

Creates a new object as a copy of an existing object.

```
Studentas::Studentas(const Studentas &copy)
: vardas(copy.vardas), pavarde(copy.pavarde), nd_rezultatai(copy.nd_rezultatai),egzaminas(copy.egzaminas) {}
```

1.3.1.3 Copy Assignment Operator

Assigns the state of one object to another existing object.

```
Studentas& Studentas::operator=(const Studentas& copy)
{
    if(this !=&copy)
    {
        vardas = copy.vardas;
        pavarde = copy.pavarde;
        nd_rezultatai = copy.nd_rezultatai;
        egzaminas = copy.egzaminas;
    }
    return *this;
}
```

1.3.1.4 Move Constructor

Transfers resources from a temporary object to a new object.

```
    Studentas& Studentas::operator=(Studentas&& copy) noexcept {
        if (this!= &copy) {
            // Swap the members of the current object with the members of the other object
            std::swap(vardas, copy.vardas);
            std::swap(pavarde, copy.pavarde);
            std::swap(nd_rezultatai, copy.nd_rezultatai);
            std::swap(egzaminas, copy.egzaminas);
        }
        return *this;
    }
```

1.3.1.5 Move Assignment Operator

Transfers resources from one object to another existing object.

```
    Studentas& Studentas::operator=(Studentas&& copy) noexcept {
        if (this!= &copy) {
            // Swap the members of the current object with the members of the other object
            std::swap(vardas, copy.vardas);
            std::swap(pavarde, copy.pavarde);
            std::swap(nd_rezultatai, copy.nd_rezultatai);
            std::swap(egzaminas, copy.egzaminas);
        }
        return *this;
    }
```

1.3.2 Overloaded Methods

The `Studentas` class overloads the input and output operators (`operator<<` and `operator>>`) to provide serialization and deserialization capabilities. These overloaded methods allow objects of the `Studentas` class to be written to an output stream (e.g., `std::cout` or a file) and read from an input stream (e.g., `std::cin` or a file).

1.3.2.1 Output Operator (`operator<<`)

The output operator `operator<<` is overloaded to serialize a `Studentas` object to an output stream. It prints the `vardas`, `pavarde`, `egzaminas`, and `nd_rezultatai` member variables to the output stream.

```
std::ostream& operator<<(std::ostream& output, const Studentas &student) {
    output << student.vardas << " " << student.pavarde << " " << student.egzaminas << " ";
    for (int pazymys : student.nd_rezultatai) {
        output << std::to_string(pazymys) << " "; // Pries printinant pakeist int'a i string'a
    }
    return output;
}
```

1.3.2.2 Input Operator (`operator>>`)

The input operator `operator>>` is overloaded to deserialize a `Studentas` object from an input stream. It reads `vardas`, `pavarde`, `egzaminas`, and `nd_rezultatai` from the input stream and constructs a `Studentas` object accordingly.

```
std::istream& operator>>(std::istream& input, Studentas &student) {
    input >> student.vardas >> student.pavarde;
    input >> student.egzaminas;
    student.nd_rezultatai.clear();
    int pazymys;
    while (input >> pazymys) {
        student.nd_rezultatai.push_back(pazymys);
    }
    return input;
}
```

1.4 Running a Makefile for C/C++ Projects

This guide will walk you through the process of running a Makefile for compiling and executing C/C++ programs on both macOS and Windows. If this tutorial does not work for you, try these solutions `Makefile`.

1.4.1 Prerequisites

1. **Make Installation:**
   - macOS: Make is usually pre-installed. You can verify by opening a terminal and typing `make -v`.
   - Windows: Install Make using a package manager like `Chocolatey`. Run `choco install make` in PowerShell or Command Prompt.

- **C/C++ Compiler:**
  - Ensure you have a C/C++ compiler installed. On macOS, Clang is typically pre-installed. On Windows, you can use MinGW or Cygwin.
- **Text Editor or IDE:**
  - Use a text editor or IDE to write your C/C++ code and Makefile. Popular choices include Visual Studio Code, Sublime Text, Atom, etc.

1.4.1.1 1. Write Your Code

- Create your C/C++ code in one or more `.cpp` or `.c` files.

1.4.1.2 2. Write Makefile

- Create a file named `Makefile` (without extension) in the same directory as your source code.
- Open `Makefile` in a text editor and define build rules.

1.4.1.3 3. Open Terminal/Command Prompt

- macOS: Open Terminal.
- Windows: Open Command Prompt or PowerShell.

1.4.1.4 4. Navigate to Project Directory

- Use `cd` command to navigate to the directory containing your code and Makefile.

1.4.1.5 5. Run Make

- Type `make` and press Enter. This executes the default target (`all`) in the Makefile.

1.4.1.6 6. Run Your Program

- After successful build, an executable file (e.g., `run` on macOS or `run.exe` on Windows) will be generated in the same directory.
- Run the program by typing `./run` on macOS or `run.exe` on Windows, and press Enter.

Congratulations! You've successfully compiled and executed your C/C++ program using a Makefile. If you encounter any errors during compilation, check your Makefile and source code for issues.

1.4.1.7 6. How To Run Code With Flags

- Type `make optimize`
- Type `./run_o1 ./run_o2 ./run_o3`

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

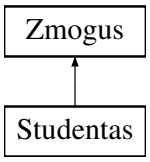
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Studentas Class Reference

Inheritance diagram for Studentas:



Public Member Functions

- **Studentas** (const std::string &vardas, const std::string &pavarde)
- **Studentas** (const Studentas &copy)
- Studentas & **operator=** (const Studentas &copy)
- Studentas & **operator=** (Studentas &&copy) noexcept
- void setVardas (std::string vardas)
- std::string getVardas () const
- void setPavarde (std::string pavarde)
- std::string getPavarde () const
- void **setNamuDarbai** (const std::vector< int > &nd)
- std::vector< int > **getNamuDarbai** () const
- void **addNamuDarbai** (int pazymys)
- void **setEgzaminas** (int egzaminas)
- int **getEgzaminas** () const
- double **calcVidurkis** () const
- double **calcMediana** () const
- double **calcGalutinis** (bool useVidurkis) const
- void **randomND** ()
- void **randomStudentai** ()

Friends

- std::ostream & **operator**<< (std::ostream &output, const Studentas &student)
- std::istream & **operator**>> (std::istream &input, Studentas &student)

5.1.1 Member Function Documentation

5.1.1.1 getPavarde()

```
std::string Studentas::getPavarde ( ) const  [virtual]
```

Implements Zmogus.

5.1.1.2 getVardas()

```
std::string Studentas::getVardas ( ) const  [virtual]
```

Implements Zmogus.

5.1.1.3 setPavarde()

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

Implements Zmogus.

5.1.1.4 setVardas()

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

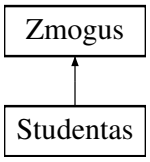
Implements Zmogus.

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

- studentas.h
- studentas.cpp

5.2 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- virtual void **setVardas** (std::string vardas)=0
- virtual std::string **getVardas** () const =0
- virtual void **setPavarde** (std::string pavarde)=0
- virtual std::string **getPavarde** () const =0

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

- studentas.h

Chapter 6

File Documentation

6.1 containers.h

```
00001 #ifndef CONTAINERS_H
00002 #define CONTAINERS_H
00003
00004 #include <vector>
00005
00006 enum class ContainerType { None, Vector};
00007 enum class Action { None, Generate, Sort };
00008 ContainerType getContainerChoice();
00009 Action getActionChoice();
00010
00011 void performAction(ContainerType containerChoice, Action actionChoice, const std::vector<int>& sizes);
00012 void runApp();
00013
00014 #endif
```

6.2 funkcijos.h

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003
00004 #include "studentas.h"
00005 #include <vector>
00006 #include <string>
00007
00008 void spausdintiGalutiniusBalus(const std::vector<Studentas>& studentai, const std::string&
      isvedimoFailoVardas = "", int rusiavimoTipas = 1);
00009 void manualInput(std::vector<Studentas>& studentai);
00010 void generateGradesOnly(std::vector<Studentas>& studentai);
00011 void readFileDataFromFile(std::vector<Studentas>& studentai, const std::string& failoVardas);
00012 void generateStudentFiles(const std::vector<int>& sizes);
00013 void rusiuotiStudentus(const std::vector<int>& sizes);
00014
00015 #endif
```

6.3 funkcijosVector.h

```
00001 #ifndef FUNKCIJOSVECTOR_H
00002 #define FUNKCIJOSVECTOR_H
00003
00004 #include "studentas.h"
00005 #include <vector>
00006
00007 void readDataVector(std::vector<Studentas>& studentai, const std::string& failoVardas);
00008 void generateStudentFilesVector(int size);
00009 void rusiuotStudentusVector(const std::string& failoVardas);
00010 void rusiuotStudentusVector2(const std::string& failoVardas);
00011 void rusiuotStudentusVector3(const std::string &failoVardas);
00012
00013 #endif // FUNKCIJOSVECTOR_H
```

6.4 studentas.h

```
00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
```

```
00003
00004 #include <string>
00005 #include <vector>
00006
00007 class Zmogus {
00008 public:
00009     virtual void setVardas(std::string vardas) = 0;
00010     virtual std::string getVardas() const = 0;
00011     virtual void setPavarde(std::string pavarde) = 0;
00012     virtual std::string getPavarde() const = 0;
00013     virtual ~Zmogus() = default;
00014
00015 };
00016
00017 class Studentas : public Zmogus {
00018 private:
00019     std::string vardas;
00020     std::string pavarde;
00021     std::vector<int> nd_rezultatai;
00022     int egzaminas;
00023 public:
00024     // Constructor
00025     Studentas();
00026     // Constructor with parameters
00027     Studentas(const std::string &vardas, const std::string &pavarde);
00028     //Destructor
00029     ~Studentas();
00030     // Copying constructor
00031     Studentas(const Studentas &copy);
00032
00033     // Copy assignment
00034     Studentas& operator=(const Studentas& copy);
00035
00036     // Move assignment operator
00037     Studentas& operator=(Studentas&& copy) noexcept;
00038
00039     void setVardas(std::string vardas);
00040     std::string getVardas() const;
00041
00042     void setPavarde(std::string pavarde);
00043     std::string getPavarde() const;
00044
00045     void setNamuDarbai(const std::vector<int> &nd);
00046     std::vector<int> getNamuDarbai() const;
00047
00048     void addNamuDarbai(int pazymys);
00049
00050     void setEgzaminas(int egzaminas);
00051     int getEgzaminas() const;
00052
00053     double calcVidurkis() const;
00054     double calcMediana() const;
00055     double calcGalutinis(bool useVidurkis) const;
00056     void randomND();
00057     void randomStudentai();
00058
00059     friend std::ostream &operator«(std::ostream &output, const Studentas &student);
00060     friend std::istream &operator»(std::istream &input, Studentas &student);
00061 };
00062
00063 #endif // STUDENTAS_H
```

6.5 testRules.h

```
00001 #ifndef TESTRULES_H
00002 #define TESTRULES_H
00003
00004 #include "studentas.h"
00005 #include <iostream>
00006 #include <fstream>
00007
00008 void testRuleOfFive();
00009 void testSerializationDeserialization();
00010
00011 #endif // TESTRULES_H
```

Index