Pazymiu Skaiciuokle

Sugeneruota Doxygen 1.13.2

skyrius 1

Naujo funkcialumo aprašymas (v1.5)

1.1 Klasės struktūra

Klasė	Tipas	Aprašymas
`Human`	Abstrakti	Bendra klasė visiems žmonėms, aprašo vardą, pavardę, turi virtualų metodą `print()`
`Student`	Išvestinė	Paveldi iš `Human`, aprašo studentų pažymius, egzaminą ir visą reikiamą logiką

1.2 Testų sistema

Visi testai iš **v1.2** versijos buvo **pakartotinai patikrinti**:

Testas	Būsena	Paaiškinimas
Rule of Five testai	Veikia	Visi kopijavimo/perkėlimo metodai veikia kaip tikėtasi
Operatoriai >> ir <<	Veikia	Įvedimas/išvedimas per srautus veikia korektiškai
`print()` per `Human*`	Veikia	Virtuali funkcija `print()` veikia per polimorfizmą
Bandymas kurti `Human` objektą	Užblokuota	Kompiliatorius blokuoja bandymą kurti `Human` objektą

1.3 Programos naudojimosi instrukcija

Šis projektas naudoja CMake kompiliavimui ir organizuoja kodą pagal standartinę struktūrą su atskirais include ir src katalogais.

1.4 Reikalavimai

Prieš pradedant, įsitikinkite, kad turite įdiegtus šiuos įrankius:

- C++ kompiliatorius (pvz., GCC arba MSVC)
- CMake (bent jau 3.25 versija)
- Git (jei norite klonuoti iš saugyklos)

1.5 Projekto paruošimas ir paleidimas

1.5.1 Projekto klonavimas (jei reikia)

Jei projektas dar nėra jūsų kompiuteryje, galite jį nusiklonuoti naudodami komandą:

```
git clone <projekto_git_nuoroda>
cd <projekto_katalogas>
```

1.5.2 Projekto kompiliavimas su CMake

Sukurkite naują katalogą, skirtą generuojamiems failams:

```
mkdir build
```

```
cd build
```

Generuokite projektą su CMake:

```
cmake ..
```

Paleiskite kompiliaciją:

```
cmake --build .
```

1.5.3 Programos paleidimas

Po sėkmingo kompiliavimo galite paleisti programą:

```
./run
```

arba Windows sistemoje tiesiog dukart spustelėkite `run.bat` failą.

1.6 Projekto struktūra

- `include/` – antraštiniai failai (.h, .hpp), kuriuose aprašomos klasės ir funkcijų prototipai.
- `src/` – pagrindinis programos kodas (.cpp).
- `CMakeLists.txt` – CMake konfigūracijos failas.
- `README.md` – ši naudojimosi instrukcija.
- `run.bat` – Windows skriptas greitam programos paleidimui.

skyrius 2

Hierarchijos Indeksas

2.1 Klasių hierarchija

Šis paveldėjimo sąrašas yra beveik surikiuotas abėcėlės tvarka:

skyrius 3

Klasės Indeksas

3.1 Klasės

Klasės, struktūros, sąjungos ir sąsajos su trumpais aprašymais:

skyrius 4

Failo Indeksas

4.1 Failai

Visų failų sąrašas su trumpais aprašymais:

skyrius 5

Klasės Dokumentacija

5.1 Human Klasė

`#include <student.h>`
Paveldimumo diagrama Human:

5.2 Student Klasė

`#include <student.h>`
Paveldimumo diagrama Student:
Bendradarbiavimo diagrama Student:

Vieši Metodai

- Student (const string &name="Vardenis", const string &surname="Pavardenis", int exam=0)
- ~Student ()
- Student (const Student &other)
- Student (Student &&other)
- Student & operator= (const Student &other)
- Student & operator= (Student &&other)
- const vector< int > & getMarks () const
- int getExam () const
- double getAverage () const
- double getMedian () const
- void addMark (int mark)
- void setExam (int newExam)
- void print () const
- void calculateAverage ()
- void calculateMedian ()
- void readLine (const string &line)

Vieši Metodai inherited from Human

- Human ()=default
- Human (const string &name="Vardenis", const string &surname="Pavardenis")
- virtual ~Human ()
- string getName () const
- string getSurname () const
- void setName (const string &newName)
- void setSurname (const string &newSurname)

Privatūs Atributai

- vector< int > marks_
- int exam_
- double average_
- double median_

Draugai

- ostream & operator<< (ostream &out, const Student &student)
- istream & operator>> (istream &in, Student &student)

Additional Inherited Members

Apsaugoti Atributai inherited from Human

- string name_
- string surname_

5.2.1 Smulkus aprašymas

Class that holds student data and inherits from Human

5.2.2 Konstruktoriaus ir Destruktoriaus Dokumentacija

5.2.2.1 Student() [1/3]

```
Student::Student (
            const string & name = "Vardenis",
            const string & surname = "Pavardenis",
            int exam = 0)  [inline]
```

Funkcijos kvietimo grafas: Here is the caller graph for this function:

5.2.2.2 ~Student()

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

5.2.2.3 Student() [2/3]

```
Student::Student (
            const Student & other)
```

Funkcijos kvietimo grafas:

5.2.2.4 Student() [3/3]

```
Student::Student (
            Student && other)
```

Funkcijos kvietimo grafas:

5.2.3 Metodų Dokumentacija

5.2.3.1 addMark()

```
void Student::addMark (
            int mark)  [inline]
```

Here is the caller graph for this function:

5.2.3.2 calculateAverage()

```
void Student::calculateAverage ()
```

Function that calculates the average marks of students. Here is the caller graph for this function:

5.2.3.3 calculateMedian()

```
void Student::calculateMedian ()
```

Function that calculates the median marks of students. Here is the caller graph for this function:

5.2.3.4 getAverage()

```
double Student::getAverage () const  [inline]
```

Here is the caller graph for this function:

5.2.3.5 getExam()

```
int Student::getExam () const  [inline]
```

5.2.3.6 getMarks()

```
const vector< int > & Student::getMarks () const  [inline]
```

5.2.3.7 getMedian()

```
double Student::getMedian () const  [inline]
```

Here is the caller graph for this function:

5.2.3.8 operator=() [1/2]

```
Student & Student::operator= (
            const Student & other)
```

Funkcijos kvietimo grafas:

5.2.3.9 operator=() [2/2]

```
Student & Student::operator= (
            Student && other)
```

Funkcijos kvietimo grafas:

5.2.3.10 print()

```
void Student::print () const  [virtual]
```

Function that prints the student data to the console.
Realizuoja Human.

5.2.3.11 readLine()

```
void Student::readLine (
            const string & line)
```

Function that reads a line from a file and assigns it to a student.

5.2.3.12 setExam()

```
void Student::setExam (
            int newExam)  [inline]
```

Here is the caller graph for this function:

5.2.4 Draugiškų Ir Susijusių Funkcijų Dokumentacija

5.2.4.1 operator<<

```
ostream & operator<< (
            ostream & out,
            const Student & student)  [friend]
```

5.2.4.2 operator>>

```
istream & operator>> (
            istream & in,
            Student & student)  [friend]
```

5.2.5 Atributų Dokumentacija

5.2.5.1 average_

```
double Student::average_  [private]
```

5.2.5.2 exam_

```
int Student::exam_  [private]
```

5.2.5.3 marks_

```
vector<int> Student::marks_  [private]
```

5.2.5.4 median_

```
double Student::median_  [private]
```

Dokumentacija šiai klasei sugeneruota iš šių failų:

- include/student.h
- src/student.cpp

5.3 Timer Klasė

```
#include <headers.h>
```

Vieši Metodai

- Timer ()
- void reset ()
- double elapsed () const

Privatūs Tipai

- using hrClock = std::chrono::high_resolution_clock
- using durationDouble = std::chrono::duration<double>

Privatūs Atributai

- std::chrono::time_point< hrClock > start

5.3.1 Smulkus aprašymas

Class that measures time.

5.3.2 Tipo Aprašymo Dokumentacija

5.3.2.1 durationDouble

```
using Timer::durationDouble = std::chrono::duration<double>  [private]
```

5.3.2.2 hrClock

```
using Timer::hrClock = std::chrono::high_resolution_clock  [private]
```

5.3.3 Konstruktoriaus ir Destruktoriaus Dokumentacija

5.3.3.1 Timer()

```
Timer::Timer ()  [inline]
```

5.3.4 Metodų Dokumentacija

5.3.4.1 elapsed()

```
double Timer::elapsed () const  [inline]
```

Here is the caller graph for this function:

5.3.4.2 reset()

```
void Timer::reset ()  [inline]
```

5.3.5 Atributų Dokumentacija

5.3.5.1 start

```
std::chrono::time_point<hrClock> Timer::start  [private]
```

Dokumentacija šiai klasei sugeneruota iš šio failo:

- include/headers.h

skyrius 6

Failo Dokumentacija

6.1 include/global.h Failo Nuoroda

```
#include <vector>
#include <list>
#include <deque>
#include <string>
#include <limits>
#include <iomanip>
#include <algorithm>
#include <random>
#include <iostream>
#include <fstream>
#include <sstream>
#include <chrono>
```

Įtraukimo priklausomybių diagrama global.h: Šis grafas rodo, kuris failas tiesiogiai ar netiesiogiai įtraukia šį failą:

6.2 global.h

Eiti į šio failo dokumentaciją.

```
00001 #pragma once
00002
00003 #include <vector>
00004 #include <list>
00005 #include <deque>
00006 #include <string>
00007 #include <limits>
00008 #include <iomanip>
00009 #include <algorithm>
00010 #include <random>
00011 #include <iostream>
00012 #include <fstream>
00013 #include <sstream>
00014 #include <chrono>
00015
00016 using std::cout;
00017 using std::cin;
00018 using std::endl;
00019 using std::vector;
00020 using std::list;
00021 using std::deque;
00022 using std::string;
00023 using std::ostream;
00024 using std::ifstream;
00025 using std::ofstream;
00026 using std::istream;
00027 using std::left;
00028 using std::setw;
00029 using std::fixed;
00030 using std::setprecision;
00031 using std::sort;
00032 using std::move;
00033 using std::partition;
```

6.3 include/headers.h Failo Nuoroda

```
#include <exception>
#include <stdexcept>
#include <system_error>
#include <future>
#include <type_traits>
#include <variant>
#include <string_view>
```

Įtraukimo priklausomybių diagrama headers.h: Šis grafas rodo, kuris failas tiesiogiai ar netiesiogiai įtraukia šį failą:

Klasės

- class Timer

Funkcijos

- void TestStudentClass ()
- int NumberCheck (int min, int max)
- int Menu ()
- void ProgramEnd ()
- void ProcessException ()

Kintamieji

- const int maxStud = 10000000
- double globalTime
- const vector< string > names = {"Jonas", "Petras", "Antanas", "Kazys", "Marius", "Lukas", "Tadas", "Dainius", "Arvydas", "Vytautas", "Mindaugas", "Rokas", "Dovydas", "Paulius", "Tomas", "Andrius", "Giedrius", "Saulius", "Algirdas", "Simas", "Egidijus", "Justas", "Laurynas", "Martynas", "Edvinas", "Kestutis", "Julius", "Raimondas", "Deividas", "Arnoldas"}
- const vector< string > surnames = {"Jonaitis", "Petraitis", "Antanaitis", "Kazlauskas", "Marciulionis", "Baltrusaitis", "Grigonis", "Kairys", "Landsbergis", "Zemaitis", "Mikalauskas", "Butkus", "Vaiciulis", "Bagdonas", "Salkauskas", "Daukantas", "Jankauskas", "Tamulevicius", "Skvernelis", "Navickas", "Kupcinskas", "Simkus", "Masiulis", "Zukauskas", "Cepaitis", "Vaitkus", "Urbsys", "Brazys", "Petrusaitis", "Daugela"}

6.3.1 Funkcijos Dokumentacija

6.3.1.1 Menu()

```
int Menu ()
```

Function that displays the menu and returns the selected action. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.3.1.2 NumberCheck()

```
int NumberCheck (
            int min,
            int max)
```

Function that checks if the input is a number and if it is within the specified range. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.3.1.3 ProcessException()

```
void ProcessException ()
```

Function that processes exceptions. Here is the caller graph for this function:

6.3.1.4 ProgramEnd()

```
void ProgramEnd ()
```

Function that ends the program. Here is the caller graph for this function:

6.3.1.5 TestStudentClass()

```
void TestStudentClass ()
```

Functions that are used in the main function.
Function that test the Student class and its methods. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.3.2 Kintamojo Dokumentacija

6.3.2.1 globalTime

```
double globalTime  [extern]
```

6.3.2.2 maxStud

```
const int maxStud = 10000000
```

6.3.2.3 names

```
const vector<string> names = {"Jonas", "Petras", "Antanas", "Kazys", "Marius", "Lukas", "Tadas",
"Dainius", "Arvydas", "Vytautas", "Mindaugas", "Rokas", "Dovydas", "Paulius", "Tomas", "Andrius",
"Giedrius", "Saulius", "Algirdas", "Simas", "Egidijus", "Justas", "Laurynas", "Martynas",
"Edvinas", "Kestutis", "Julius", "Raimondas", "Deividas", "Arnoldas"}
```

Global variables that hold names and surnames.

6.3.2.4 surnames

```
const vector<string> surnames = {"Jonaitis", "Petraitis", "Antanaitis", "Kazlauskas", "Marciulionis",
"Baltrusaitis", "Grigonis", "Kairys", "Landsbergis", "Zemaitis", "Mikalauskas", "Butkus",
"Vaiciulis", "Bagdonas", "Salkauskas", "Daukantas", "Jankauskas", "Tamulevicius", "Skvernelis",
"Navickas", "Kupcinskas", "Simkus", "Masiulis", "Zukauskas", "Cepaitis", "Vaitkus", "Urbsys",
"Brazys", "Petrusaitis", "Daugela"}
```

6.4 headers.h

Eiti į šio failo dokumentaciją.

```
00001 #pragma once
00002
00003 #include <exception>
00004 #include <stdexcept>
00005 #include <system_error>
00006 #include <future>
00007 #include <type_traits>
00008 #include <variant>
00009 #include <string_view>
00010
00011 using std::bad_alloc;
00012 using std::cerr;
00013 using std::exception;
00014 using std::future_error;
00015 using std::ios_base;
00016 using std::istringstream;
00017 using std::string_view;
00018 using std::system_error;
00019
00020 // Global variables.
00021 const int maxStud = 10000000;
00022 extern double globalTime;
00023
00025 class Timer
00026 {
00027 private:
00028     using hrClock = std::chrono::high_resolution_clock;
```

```
00029     using durationDouble = std::chrono::duration<double>;
00030     std::chrono::time_point<hrClock> start;
00031
00032 public:
00033     Timer() : start{hrClock::now()} {}
00034     void reset()
00035     {
00036         start = hrClock::now();
00037     }
00038     double elapsed() const
00039     {
00040         return durationDouble(hrClock::now() - start).count();
00041     }
00042 };
00043
00045 void TestStudentClass();
00046 int NumberCheck(int min, int max);
00047 int Menu();
00048 void ProgramEnd();
00049 void ProcessException();
00050
00052 const vector<string> names = {"Jonas", "Petras", "Antanas", "Kazys", "Marius", "Lukas", "Tadas",
      "Dainius", "Arvydas", "Vytautas", "Mindaugas", "Rokas", "Dovydas", "Paulius", "Tomas", "Andrius",
      "Giedrius", "Saulius", "Algirdas", "Simas", "Egidijus", "Justas", "Laurynas", "Martynas", "Edvinas",
      "Kestutis", "Julius", "Raimondas", "Deividas", "Arnoldas"};
00053 const vector<string> surnames = {"Jonaitis", "Petraitis", "Antanaitis", "Kazlauskas", "Marciulionis",
      "Baltrusaitis", "Grigonis", "Kairys", "Landsbergis", "Zemaitis", "Mikalauskas", "Butkus", "Vaiciulis",
      "Bagdonas", "Salkauskas", "Daukantas", "Jankauskas", "Tamulevicius", "Skvernelis", "Navickas",
      "Kupcinskas", "Simkus", "Masiulis", "Zukauskas", "Cepaitis", "Vaitkus", "Urbsys", "Brazys",
      "Petrusaitis", "Daugela"};
```

6.5 include/student.h Failo Nuoroda

Šis grafas rodo, kuris failas tiesiogiai ar netiesiogiai įtraukia šį failą:

Klasės

- class Human
- class Student

6.6 student.h

Eiti į šio failo dokumentaciją.

```
00001 #pragma once
00002
00004 class Human
00005 {
00006 protected:
00007     string name_;
00008     string surname_;
00009
00010 public:
00011     // Constructors and destructor
00012     Human() = default;
00013     Human(const string &name = "Vardenis", const string &surname = "Pavardenis")
00014         : name_(name), surname_(surname) {}
00015
00016     virtual ~Human()
00017     {
00018         cout « "[~] Base destructor called for: " « name_ « " " « surname_ « endl;
00019     }
00020
00021     // Getters and setters
00022     string getName() const { return name_; }
00023     string getSurname() const { return surname_; }
00024     void setName(const string &newName) { name_ = newName; }
00025     void setSurname(const string &newSurname) { surname_ = newSurname; }
00026
00027     virtual void print() const = 0;
00028 };
00029
00031 class Student : public Human
00032 {
00033 private:
00034     vector<int> marks_;
00035     int exam_;
00036     double average_, median_;
00037
```

```
00038 public:
00039     // Constructors and destructor
00040     Student(const string &name = "Vardenis", const string &surname = "Pavardenis", int exam = 0)
00041         : Human(name, surname), exam_(exam), average_(0.0), median_(0.0) { marks_.reserve(20); }
00042
00043     ~Student()
00044     {
00045         cout « "[~] Destructor called for: " « name_ « " " « surname_ « endl;
00046     }
00047
00048     // Copy constructor and move constructor
00049     Student(const Student &other);
00050     Student(Student &&other);
00051
00052     // Copy assignment operator and move assignment operator
00053     Student &operator=(const Student &other);
00054     Student &operator=(Student &&other);
00055
00056     // Overloaded operators for input and output
00057     friend ostream &operator«(ostream &out, const Student &student);
00058     friend istream &operator»(istream &in, Student &student);
00059
00060     // Getters and setters
00061     const vector<int> &getMarks() const { return marks_; }
00062     int getExam() const { return exam_; }
00063     double getAverage() const { return average_; }
00064     double getMedian() const { return median_; }
00065
00066     void addMark(int mark) { marks_.push_back(move(mark)); }
00067     void setExam(int newExam) { exam_ = newExam; }
00068
00069     // Functions
00070     void print() const;
00071     void calculateAverage();
00072     void calculateMedian();
00073     void readLine(const string &line);
00074 };
```

6.7 include/templates.h Failo Nuoroda

Šis grafas rodo, kuris failas tiesiogiai ar netiesiogiai įtraukia šį failą:

Funkcijos

- template<typename Container>
  void ReadFromFile (Container &group, int action)
- template<typename Container>
  void Action (Container &group, int action)
- template<typename Container>
  double Sort (Container &group, int &markAction)
- template<typename Container>
  void Output (Container &group, ostream &out, int markAction)
- template<typename Container>
  void SeparateStudents (Container &group, Container &failed)
- template<typename Container>
  void OutputSeparated (Container &group, Container &failed)
- template<typename Container>
  void GenerateFile (Container &group)

6.7.1 Funkcijos Dokumentacija

6.7.1.1 Action()

```
template<typename Container>
void Action (
            Container & group,
            int action)
```

Function that asks the user to input data manually or generates it randomly. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.2 GenerateFile()

```
template<typename Container>
void GenerateFile (
            Container & group)
```

Function that generates data and writes it to a file. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.3 Output()

```
template<typename Container>
void Output (
            Container & group,
            ostream & out,
            int markAction)
```

Function that outputs the results to the console or a file. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.4 OutputSeparated()

```
template<typename Container>
void OutputSeparated (
            Container & group,
            Container & failed)
```

Function that outputs the sorted students to two files. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.5 ReadFromFile()

```
template<typename Container>
void ReadFromFile (
            Container & group,
            int action)
```

Function that reads data from a file. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.6 SeparateStudents()

```
template<typename Container>
void SeparateStudents (
            Container & group,
            Container & failed)
```

Function that sorts students into two groups - those who passed and those who failed. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.7.1.7 Sort()

```
template<typename Container>
double Sort (
            Container & group,
            int & markAction)
```

Function that sorts the students by name, surname or final mark. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.8 templates.h

Eiti į šio failo dokumentaciją.

```
00001 #pragma once
00002
00004 template <typename Container>
00005 void ReadFromFile(Container &group, int action)
```

```
00006 {
00007     string readName;
00008     bool fileLoaded = false;
00009     while (!fileLoaded)
00010     {
00011         cout « "Iveskite failo pavadinima, is kurio bus skaitomi duomenys: " « endl;
00012         cin » readName;
00013         try
00014         {
00015             ifstream input(readName, std::ios::binary);
00016             if (!input)
00017                 throw std::ios_base::failure("Failas nerastas arba negali buti atidarytas.");
00018             else
00019             {
00020                 fileLoaded = true;
00021                 Timer inputTime;
00022                 string line;
00023                 getline(input, line);
00024                 while (getline(input, line))
00025                 {
00026                     group.emplace_back();
00027                     group.back().readLine(line);
00028                     group.back().calculateAverage();
00029                     group.back().calculateMedian();
00030                 }
00031                 input.close();
00032                 cout « " * Duomenu skaitymas uztruko: " « inputTime.elapsed() « " sekundziu. " « endl;
00033                 globalTime += inputTime.elapsed();
00034             }
00035         }
00036         catch (...)
00037         {
00038             ProcessException();
00039             cin.clear();
00040             cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00041         }
00042     }
00043     if (action != 6)
00044     {
00045         string writeName = "rezultatas.txt";
00046         ofstream output(writeName);
00047         int markAction;
00048         Sort(group, markAction);
00049         Output(group, output, markAction);
00050         output.close();
00051         cout « "Duomenys nukopijuoti i faila: " « writeName « endl;
00052     }
00053 }
00054
00056 template <typename Container>
00057 void Action(Container &group, int action)
00058 {
00059     cout « "Iveskite studentu skaiciu (iveskite 0, jei skaicius yra nezinomas): " « endl;
00060     int amountStud = NumberCheck(0, maxStud);
00061     bool amountStudKnown = (amountStud != 0);
00062     if (!amountStudKnown)
00063         amountStud = maxStud;
00064
00065     for (int i = 0; i < amountStud; i++)
00066     {
00067         Student temp;
00068         if (action == 2)
00069         {
00070             string name, surname;
00071             cout « "Iveskite studento varda: ";
00072             cin » name;
00073             temp.setName(name);
00074             cout « "Iveskite studento pavarde: ";
00075             cin » surname;
00076             temp.setSurname(surname);
00077         }
00078         else if (action == 3)
00079         {
00080             temp.setName(names[rand() % names.size()]);
00081             temp.setSurname(surnames[rand() % surnames.size()]);
00082         }
00083         if (action == 2 || action == 3)
00084         {
00085             int amountMarks = rand() % 100 + 1;
00086             for (int j = 0; j < amountMarks; ++j)
00087                 temp.addMark(rand() % 10 + 1);
00088             temp.setExam(rand() % 10 + 1);
00089             temp.calculateAverage();
00090             temp.calculateMedian();
00091         }
00092         else if (action == 1)
00093             cin » temp;
```

```
00094         group.push_back(move(temp));
00095         if (!amountStudKnown)
00096         {
00097             cout « "1 - ivesti dar vieno studento duomenis; 0 - baigti ivedima. " « endl;
00098             if (NumberCheck(0, 1) == 0)
00099                 break;
00100         }
00101     }
00102 }
00103
00105 template <typename Container>
00106 double Sort(Container &group, int &markAction)
00107 {
00108     cout « "Pasirinkite rezultatu isvedimo metoda: " « endl;
00109     cout « "1 - gauti vidurkius; 2 - gauti medianas. " « endl;
00110     markAction = NumberCheck(1, 2);
00111     cout « "Pairinkite rezultatu rusiavimo metoda: " « endl;
00112     cout « "1 - rusiuoti pagal varda (A-Z); 2 - rusiuoti pagal pavarde (A-Z); 3 - rusiuoti pagal
      galutini pazymi." « endl;
00113     int sortAction = NumberCheck(1, 3);
00114
00115     Timer sortTime;
00116     auto compare = [&](const Student &a, const Student &b)
00117     {
00118         if (sortAction == 1)
00119             return a.getName() < b.getName();
00120         if (sortAction == 2)
00121             return a.getSurname() < b.getSurname();
00122         if (sortAction == 3)
00123             return (markAction == 1) ? (a.getAverage() < b.getAverage()) : (a.getMedian() <
      b.getMedian());
00124         return false;
00125     };
00126     if constexpr (std::is_same_v<Container, vector<Student» || std::is_same_v<Container,
      deque<Student»)
00127     {
00128         sort(group.begin(), group.end(), compare);
00129     }
00130     else if constexpr (std::is_same_v<Container, list<Student»)
00131     {
00132         group.sort(compare);
00133     }
00134     return sortTime.elapsed();
00135 }
00136
00138 template <typename Container>
00139 void Output(Container &group, ostream &out, int markAction)
00140 {
00141     Timer outputTime;
00142     out « left « setw(20) « "Pavarde" « setw(20) « "Vardas";
00143     if (markAction == 1)
00144         out « setw(20) « "Galutinis (Vid.)" « endl;
00145     else if (markAction == 2)
00146         out « setw(20) « "Galutinis (Med.)" « endl;
00147     out « "-------------------------------------------------------------" « endl;
00148     for (auto &final : group)
00149     {
00150         out « left « setw(20) « final.getSurname() « setw(20) « final.getName();
00151         if (markAction == 1)
00152             out « setw(20) « fixed « setprecision(2) « final.getAverage() « endl;
00153         else if (markAction == 2)
00154             out « setw(20) « fixed « setprecision(2) « final.getMedian() « endl;
00155     }
00156     globalTime += outputTime.elapsed();
00157     cout « " * Rezultatu isvedimas uztruko: " « outputTime.elapsed() « " sekundziu. " « endl;
00158 }
00159
00161 template <typename Container>
00162 void SeparateStudents(Container &group, Container &failed)
00163 {
00164     Timer separationTime;
00165     auto it = std::partition(group.begin(), group.end(), [](const Student &final)
00166                              { return final.getAverage() >= 5; });
00167     failed.insert(failed.end(), std::make_move_iterator(it), std::make_move_iterator(group.end()));
00168     group.erase(it, group.end());
00169     globalTime += separationTime.elapsed();
00170     cout « " * Studentu skirstymas i 2 kategorijas uztruko: " « separationTime.elapsed() « "
      sekundziu. " « endl;
00171 }
00172
00174 template <typename Container>
00175 void OutputSeparated(Container &group, Container &failed)
00176 {
00177     int markAction;
00178     double sortTime1 = Sort(group, markAction);
00179     ofstream passedOut("kietiakai.txt");
00180     Timer passedTime;
```

```
00181     Output(group, passedOut, markAction);
00182     double outTime1 = passedTime.elapsed();
00183     passedOut.close();
00184     cout « "Kietiakai surasyti i faila: kietiakai.txt." « endl;
00185     double sortTime2 = Sort(failed, markAction);
00186     cout « " * Studentu rusiavimas uztruko: " « sortTime1 + sortTime2 « " sekundziu. " « endl;
00187     globalTime += sortTime1 + sortTime2;
00188     ofstream failedOut("vargsiukai.txt");
00189     Timer failedTime;
00190     Output(failed, failedOut, markAction);
00191     double outTime2 = failedTime.elapsed();
00192     failedOut.close();
00193     cout « "Vargsiukai surasyti i faila: vargsiukai.txt." « endl;
00194     cout « " * Rezultatu isvedimas i 2 failus uztruko: " « outTime1 + outTime2 « " sekundziu. " «
      endl;
00195 }
00196
00198 template <typename Container>
00199 void GenerateFile(Container &group)
00200 {
00201     cout « "Iveskite failo pavadinima, i kuri bus irasyti duomenys: " « endl;
00202     string fout;
00203     cin » fout;
00204     cout « "Iveskite studentu skaiciu, kuriu informacija norite sugeneruoti: " « endl;
00205     int amountStud = NumberCheck(1, maxStud);
00206     Timer generateTime;
00207     int amountMarks = rand() % 11 + 10;
00208     for (int i = 1; i <= amountStud; i++)
00209     {
00210         Student temp;
00211         temp.setName("VardasNr" + std::to_string(i));
00212         temp.setSurname("PavardeNr" + std::to_string(i));
00213         for (int j = 0; j < amountMarks; j++)
00214             temp.addMark((rand() % 10 + 1));
00215         temp.setExam(rand() % 10 + 1);
00216         group.push_back(move(temp));
00217     }
00218     ofstream out(fout);
00219     out « left « setw(20) « "Vardas" « setw(20) « "Pavarde";
00220     for (int i = 1; i <= amountMarks; i++)
00221         out « left « setw(10) « ("ND" + std::to_string(i));
00222     out « setw(10) « "Egz." « endl;
00223     for (auto &final : group)
00224     {
00225         out « left « setw(20) « final.getName() « setw(20) « final.getSurname();
00226         for (auto mark : final.getMarks())
00227             out « left « setw(10) « mark;
00228         out « setw(10) « final.getExam() « endl;
00229     }
00230     out.close();
00231     cout « "Duomenys buvo sekmingai sukurti faile: " « fout « endl;
00232     cout « " * Duomenu generavimas uztruko: " « generateTime.elapsed() « " sekundziu. " « endl;
00233     globalTime += generateTime.elapsed();
00234 }
```

6.9 README.md Failo Nuoroda

6.10 src/functions.cpp Failo Nuoroda

```
#include "../include/global.h"
#include "../include/headers.h"
#include "../include/student.h"
#include "../include/templates.h"
```

Įtraukimo priklausomybių diagrama functions.cpp:

Funkcijos

- void TestStudentClass ()
- int NumberCheck (int min, int max)
- int Menu ()
- void ProgramEnd ()
- void ProcessException ()

Kintamieji

- double globalTime = 0

6.10.1 Funkcijos Dokumentacija

6.10.1.1 Menu()

```
int Menu ()
```

Function that displays the menu and returns the selected action. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.10.1.2 NumberCheck()

```
int NumberCheck (
            int min,
            int max)
```

Function that checks if the input is a number and if it is within the specified range. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.10.1.3 ProcessException()

```
void ProcessException ()
```

Function that processes exceptions. Here is the caller graph for this function:

6.10.1.4 ProgramEnd()

```
void ProgramEnd ()
```

Function that ends the program. Here is the caller graph for this function:

6.10.1.5 TestStudentClass()

```
void TestStudentClass ()
```

Function that test the Student class and its methods. Funkcijos kvietimo grafas: Here is the caller graph for this function:

6.10.2 Kintamojo Dokumentacija

6.10.2.1 globalTime

```
double globalTime = 0
```

6.11 src/main.cpp Failo Nuoroda

```
#include "../include/global.h"
#include "../include/headers.h"
#include "../include/student.h"
#include "../include/templates.h"
```

Įtraukimo priklausomybių diagrama main.cpp:

Funkcijos

- int main ()

6.11.1 Funkcijos Dokumentacija

6.11.1.1 main()

```
int main ()
```

Funkcijos kvietimo grafas:

6.12 src/student.cpp Failo Nuoroda

```
#include "../include/global.h"
#include "../include/headers.h"
#include "../include/student.h"
#include "../include/templates.h"
```

Įtraukimo priklausomybių diagrama student.cpp:

Funkcijos

- ostream & operator<< (ostream &out, const Student &student)
- istream & operator>> (istream &in, Student &student)

6.12.1 Funkcijos Dokumentacija

6.12.1.1 operator<<()

```
ostream & operator<< (
            ostream & out,
            const Student & student)
```

6.12.1.2 operator>>()

```
istream & operator>> (
            istream & in,
            Student & student)
```

Rodyklė