Pazymiu Skaiciuokle

Sugeneruota Doxygen 1.13.2

skyrius 1

> Programos diegimo instrukcija <

Ši programa pateikiama kaip diegiamasis failas `setup.exe`, kurį galite paleisti ir sekti paprastais žingsniais, kad sėkmingai įdiegtumėte ją savo kompiuteryje.

1.1 Diegimo žingsniai

1. Atsisiųskite ir **paleiskite failą `setup.exe`** (du kartus spustelėkite).
2. Vykdykite ekrane rodomas instrukcijas, kol diegimas bus baigtas.
3. Po sėkmingo diegimo:
   - **Darbalaukyje** (Desktop) atsiras programos **sutrumpinimas** (shortcut),
   - Taip pat programą rasite per **Pradžios meniu** (Start Menu),
   - Programa bus įdiegta į katalogą:
     `C:\Program Files\Vilniaus Universitetas\Julius Vilkanec\PazymiuSkaiciuokle.exe`

1.2 Programos paleidimas

Programą galite paleisti dviem būdais:

- Spustelėkite **sutrumpinimą (shortcut)** darbalaukyje arba per **Pradžios meniu** (Start Menu).
- Arba atverkite programos katalogą rankiniu būdu:

`C:\Program Files\Vilniaus Universitetas\Julius Vilkanec\PazymiuSkaiciuokle.exe`

1.3 Tolimesni žingsniai

Kai programa bus atidaryta, vadovaukitės jos pateikiamomis instrukcijomis ekrane. Viskas paruošta darbui!
Šis projektas naudoja CMake kompiliavimui ir yra suskirstytas pagal aiškią struktūrą su `include/` ir `src/` katalogais.

1.4 > Programos paleidimas naudojant CmakeLists <

Tai yra ∗∗**alternatyvus būdas** paleisti programos exe failą.∗∗

1.5 Reikalavimai

Įsitikinkite, kad turite įdiegtus šiuos įrankius savo sistemoje:

- **C++ kompiliatorius** (pvz., GCC arba MSVC)
- **CMake** (bent 3.25 versija)

  Pastaba: Git nebūtinas, jei jau turite projekto failus.

1.6 Projekto paruošimas ir paleidimas

1.6.1 1. Projekto failų paruošimas

Jeigu dar neturite projekto aplanke, galite jį atsisiųsti arba nusiklonuoti iš saugyklos:

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

1.6.2 2. Paleidimas naudojant run.bat (Windows)

Norėdami automatiškai sukompiliuoti ir paleisti programą, tiesiog dukart spustelėkite failą:

```
run.bat
```

Šis failas:

- automatiškai sukuria `build/` katalogą (jeigu jo dar nėra),
- sugeneruoja ir sukompiliuoja projektą naudojant CMake,
- paleidžia sukurtą `.exe` failą (`Pazymiu_Skaiciuokle.exe`).

1.6.3 Alternatyvus rankinis paleidimas (jei reikia)

Jei norite viską daryti per komandų eilutę:

```
cmake -B build -S .
cmake --build build
build\Pazymiu_Skaiciuokle.exe
```

1.7 Testavimas

Catch2 testų failas (`tests.cpp`) yra projekte, tačiau testai **nėra** įtraukiami į automatinį kompiliavimą.
Jeigu reikia paleisti testus:

1. Atidarykite projektą Visual Studio Code aplinkoje.
2. Suraskite ir paleiskite `tests.cpp` failą rankiniu būdu su integruotu Catch2 palaikymu.

1.8 Projekto struktūra

- `include/` – antraštiniai failai (.h), kuriuose aprašomos klasės ir funkcijų prototipai.
- `src/` – pagrindinis programos kodas (.cpp).
- `CMakeLists.txt` – CMake konfigūracijos failas, skirtas kompiliavimui.
- `run.bat` – skriptas Windows sistemai, kuris automatizuoja kompiliavimą ir paleidimą.
- `README.md` – ši naudojimosi instrukcija.

1.8.1 Dokumentacija

- `Dokumentacija.pdf` – projekto dokumentacija.
- `Doxyfile` – Doxygen konfigūracijos failas, skirtas automatiškai generuoti projekto dokumentaciją.
- `html/` – Doxygen generuota HTML dokumentacija.
- `latex/` – Doxygen generuota LaTeX dokumentacija.

1.8.2 Catch2 testavimas

- `catch2/` – Catch2 testavimo sistema.

1.8.3 Diegimo failai

- `setup.exe` – paleidžiamasis diegimo failas, kuris pradeda programos įdiegimo procesą (Windows).
- `setup.msi` – pagrindinis diegimo paketas, kuriame yra visa reikalinga informacija ir failai, skirti programos įdiegimui.

1.9 —

1.10 **>Testavimo rezultatai (v3.0)<**

Testavimo sąlygos:

- CPU: i7-12700H; 2.70 GHz
- RAM: SODIMM; 16,0 GB
- SSD: Micron_2450; 954 GB
- Optimizavimas: `-O3`
- **Testavimo metodas:** Kiekvienas testas buvo atliekamas kelis kartus (ne mažiau nei tris) ir pateikiami laikų vidurkiai, gauti iš kelių bandymų. Laikai matuojami nuo programos pradžios iki pabaigos, apimant visus ir `std::vector` ir `Vector` veiksmus (skaitymas, skirstymas į grupes, rūšiavimas ir t. t.).

1.11 1. `std::vector` ir `Vector` klasių spartos palyginimas su skirtingais studentų įrašų kiekiais

1.11.1 1.1. Naudojant `std::vector`

Failas	Įrašų kiekis	Laikas (s)
st100000.txt	100 000	0.805
st1000000.txt	1 000 000	5.809
st10000000.txt	10 000 000	80.505

1.11.2 1.2. Naudojant `Vector`

Failas	Įrašų kiekis	Laikas (s)
st100000.txt	100 000	0.725
st1000000.txt	1 000 000	5.850
st10000000.txt	10 000 000	82.105

1.11.3 1.3. Išvada

Testavimo rezultatai parodė, kad `std::vector` ir `Vector` klasės pasižymi labai panašiu veikimo greičiu. Nors `std::vector` šiek tiek lenkia apdorojant labai didelius duomenų kiekius, `Vector` klasė taip pat demonstruoja stabilų ir pakankamai greitą veikimą.

1.12 2. `std::vector` ir `Vector` klasių spartos palyginimas naudojant `push_back()` funkciją

Lentelėje pateikiami rezultatai testų naudojant `push_back()` funkciją, su skirtingais `int` elementų kiekiais, taip pat nurodytas atminties realokacijų skaičius, kai vektorius pasiekia maksimalų dydį (100 000 000 elementų).

Elementų skaičius	`std::vector` laikas (s)	`Vector` laikas (s)
10 000	0,002	0,001
100 000	0,004	0,005
1 000 000	0,010	0,020
10 000 000	0,021	0,017
100 000 000	0,170	0,230
Realloc skaičius	25	22

1.12.1 2.1. Išvada

Nors `Vector` klasė kai kuriais atvejais demonstruoja panašų ar net geresnį greitį nei `std::vector`, bendras našumas vis dėlto rodo, kad `std::vector` yra stabilesnis ir geriau pritaikytas didelio kiekio duomenų tvarkymui. Tai yra natūralu, atsižvelgiant į tai, kad `std::vector` yra plačiai optimizuotas ir testuotas sprendimas standartinėje C++ bibliotekoje.

2.2. Naudotas testavimo kodas (spausti čia kad peržiūrėti)

```
unsigned int sz = 10000000;
Timer stdVecTime;
int rellacations = 0;
size_t prevCapacity = 0;
std::vector<int> v1;
for (int i = 1; i <= sz; ++i) {
v1.push_back(i);
if (v1.capacity() != prevCapacity) {
++rellacations;
prevCapacity = v1.capacity();
} }
cout « "std::vector v1 uzpildymo laikas: " « stdVecTime.elapsed() « " sekundziu." « endl;
cout « "std::vector v1 relokaciju skaicius: " « rellacations « endl;
Timer myVecTime;
rellacations = 0;
prevCapacity = 0;
Vector<int> v2;
for (int i = 1; i <= sz; ++i) {
v2.push_back(i);
if (v2.capacity() != prevCapacity) {
++rellacations;
prevCapacity = v2.capacity();
} }
cout « "Vector v2 uzpildymo laikas: " « myVecTime.elapsed() « " sekundziu." « endl;
cout « "Vector v2 relokaciju skaicius: " « rellacations « endl;
system("pause");
</details>
```

1.13 **>Vector klasės funkcijų aprašymas (v3.0)<**

1.14 1.1. Funkcijos pavadinimas

`push_back(const T&)`

1.15 1.2. Funkcijos paskirtis

Ši funkcija prideda naują elementą į vektoriaus pabaigą.

1.16 1.3. Veikimo principas

Jeigu konteinerio talpa yra pakankama, naujas elementas pridedamas į data[size], o size padidinamas vienetu. Jei talpa viršyta, iškviečiamas `reserve()`, talpa padidinama, sukuriamas naujas masyvas, esami elementai perkeliami, ir tada naujas elementas pridedamas.

1.17 1.4. Testavimo pavyzdys

```
Vector<std::string> letters;
std::vector<std::string> std_letters;
letters.push_back("abc");
std_letters.push_back("abc");
REQUIRE(letters.size() == std_letters.size());
for (size_t i = 0; i < letters.size(); ++i) {
REQUIRE(letters[i] == std_letters[i]);
}
```

1.18 2.1. Funkcijos pavadinimas

`pop_back()`

1.19 2.2. Funkcijos paskirtis

Pašalina paskutinį elementą iš vektoriaus.

1.20 2.3. Veikimo principas

Funkcija sumažina `size_` reikšmę vienetu, efektyviai pašalindama paskutinį elementą. Jei T yra klasė, jos destruktorius turėtų būti iškviestas.

1.21 2.4. Testavimo pavyzdys

```
Vector<std::string> pop_back_test{"one", "two", "three"};
std::vector<std::string> std_pop_back{"one", "two", "three"};
pop_back_test.pop_back();
std_pop_back.pop_back();
REQUIRE(pop_back_test.size() == std_pop_back.size());
for (size_t i = 0; i < pop_back_test.size(); ++i) {
REQUIRE(pop_back_test[i] == std_pop_back[i]);
}
```

1.22 3.1. Funkcijos pavadinimas

`resize(int newSize)`

1.23 3.2. Funkcijos paskirtis

Keičia vektoriaus dydį į nurodytą `newSize`.

1.24 3.3. Veikimo principas

Jeigu `newSize` yra didesnis nei esamas dydis, vektorius praplečiamas ir nauji elementai inicijuojami T() reikšme. Jei `newSize` mažesnis – pertekliniai elementai pašalinami.

1.25 3.4. Testavimo pavyzdys

```
Vector<int> v;
REQUIRE(v.size() == 0);
REQUIRE(v.capacity() > 0);
v.resize(5);
REQUIRE(v.size() == 5);
REQUIRE(v[4] == 0);
```

1.26 4.1. Funkcijos pavadinimas

`operator[](int index)`

1.27 4.2. Funkcijos paskirtis

Leidžia prieiti prie elemento pagal jo indeksą.

1.28 4.3. Veikimo principas

Tikrina, ar indeksas yra galiojantis (0 index < size), kitu atveju meta išimtį. Grąžina nuorodą į atitinkamą `data_` masyvo elementą.

1.29 4.4. Testavimo pavyzdys

```
Vector<int> numbers{2, 4, 6, 8};
std::vector<int> std_numbers{2, 4, 6, 8};
REQUIRE(numbers[1] == 4);
numbers[0] = 5;
std_numbers[0] = 5;
REQUIRE(numbers[0] == 5);
REQUIRE(std_numbers[0] == 5);
```

1.30 5.1. Funkcijos pavadinimas

`insert(int index, const T& value)`

1.31 5.2. Funkcijos paskirtis

Įterpia elementą į pasirinktą poziciją.

1.32 5.3. Veikimo principas

Jeigu reikia – plečia talpą `reserve`, po to perkelia visus elementus į dešinę nuo `index`, įterpia naują reikšmę ir padidina `size_`.

1.33 5.4. Testavimo pavyzdys

```
Vector<int> ins_vec{1, 3, 4};
std::vector<int> ins_std_vec{1, 3, 4};
ins_vec.insert(1, 2);
ins_std_vec.insert(ins_std_vec.begin() + 1, 2);
REQUIRE(ins_vec.size() == ins_std_vec.size());
for (size_t i = 0; i < ins_vec.size(); ++i) {
REQUIRE(ins_vec[i] == ins_std_vec[i]);
}
```

1.34 6.1. Funkcijos pavadinimas

`clear()`

1.35 6.2. Funkcijos paskirtis

Pašalina visus vektoriaus elementus.

1.36 6.3. Veikimo principas

Nustato `size_ = 0`.

1.37 6.4. Testavimo pavyzdys

```
std::vector<int> clearVec{1, 2, 3};
REQUIRE(clearVec.size() == 3);
clearVec.clear();
REQUIRE(clearVec.size() == 0);
REQUIRE(clearVec.empty() == true);
```

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 (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]
```

Constructors and destructor 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)
```

Copy constructor and move constructor
Copy constructor Funkcijos kvietimo grafas:

5.2.2.4 Student() [3/3]

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

Move constructor 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]
```

Here is the caller graph for this function:

5.2.3.6 getMarks()

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

Getters and setters

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)
```

Copy assignment operator and move assignment operator
Copy assignment operator Funkcijos kvietimo grafas:

5.2.3.9 operator=() [2/2]

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

Move assignment operator Funkcijos kvietimo grafas:

5.2.3.10 print()

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

Functions
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 Here is the caller graph for this function:

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]
```

Overloaded operators for input and output
Overloaded output operator for Student class

5.2.4.2 operator>>

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

Overloaded input operator for Student class

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

5.4 Vector< T > Klasė Šablonas

```
#include <vector.h>
```

Vieši Tipai

- using value_type = T
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using reference = value_type &
- using const_reference = const value_type &
- using pointer = value_type *
- using const_pointer = const value_type *
- using iterator = pointer
- using const_iterator = const_pointer

Vieši Metodai

- Vector ()
- Vector (const Vector &other)
- Vector (Vector &&other)
- Vector (int elements, T value=T())
- Vector (const std::initializer_list< T > &list)
- ~Vector ()
- const T & front () const
- T * begin ()
- const T * begin () const
- void push_front (const T &value)
- void pop_front ()
- const T & back () const
- T * end ()
- const T * end () const
- void push_back (const T &value)
- void push_back (T &&value)
- void pop_back ()
- int size () const noexcept
- void resize (int newSize)
- int capacity () const noexcept
- void reserve (int newCapacity)
- void shrink_to_fit ()
- bool empty () const noexcept
- void clear ()
- void swap (Vector &other)
- T * data () const
- T & operator[ ] (int index)
- const T & operator[ ] (int index) const
- T & at (int index)
- void insert (int index, const T &value)
- void erase (int index)
- Vector & operator= (const Vector &other)
- Vector & operator= (Vector &&other) noexcept
- bool operator== (const Vector &other) const
- bool operator!= (const Vector &other) const
- bool operator< (const Vector &other) const
- bool operator<= (const Vector &other) const
- bool operator> (const Vector &other) const
- bool operator>= (const Vector &other) const

Privatūs Atributai

- size_t size_
- size_t capacity_
- T * data_

Draugai

- ostream & operator<< (ostream &out, const Vector &other)
- istream & operator>> (istream &in, Vector &other)

5.4.1 Tipo Aprašymo Dokumentacija

5.4.1.1 const_iterator

```
template<typename T>
using Vector< T >::const_iterator = const_pointer
```

5.4.1.2 const_pointer

```
template<typename T>
using Vector< T >::const_pointer = const value_type *
```

5.4.1.3 const_reference

```
template<typename T>
using Vector< T >::const_reference = const value_type &
```

5.4.1.4 difference_type

```
template<typename T>
using Vector< T >::difference_type = std::ptrdiff_t
```

5.4.1.5 iterator

```
template<typename T>
using Vector< T >::iterator = pointer
```

5.4.1.6 pointer

```
template<typename T>
using Vector< T >::pointer = value_type *
```

5.4.1.7 reference

```
template<typename T>
using Vector< T >::reference = value_type &
```

5.4.1.8 size_type

```
template<typename T>
using Vector< T >::size_type = std::size_t
```

5.4.1.9 value_type

```
template<typename T>
using Vector< T >::value_type = T
```

5.4.2 Konstruktoriaus ir Destruktoriaus Dokumentacija

5.4.2.1 Vector() [1/5]

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

Default constructor Here is the caller graph for this function:

5.4.2.2 Vector() [2/5]

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

Copy constructor Funkcijos kvietimo grafas:

5.4.2.3 Vector() [3/5]

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

Move constructor Funkcijos kvietimo grafas:

5.4.2.4 Vector() [4/5]

```
template<typename T>
Vector< T >::Vector (
            int elements,
            T value = T())  [inline]
```

Parameterized constructor

5.4.2.5 Vector() [5/5]

```
template<typename T>
Vector< T >::Vector (
            const std::initializer_list< T > & list)  [inline]
```

Initializer list constructor Funkcijos kvietimo grafas:

5.4.2.6 ~Vector()

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

Destructor

5.4.3 Metodų Dokumentacija

5.4.3.1 at()

```
template<typename T>
T & Vector< T >::at (
            int index)  [inline]
```

Here is the caller graph for this function:

5.4.3.2 back()

```
template<typename T>
const T & Vector< T >::back () const  [inline]
```

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

5.4.3.3 begin() [1/2]

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

Here is the caller graph for this function:

5.4.3.4 begin() [2/2]

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

5.4.3.5 capacity()

```
template<typename T>
int Vector< T >::capacity () const  [inline], [noexcept]
```

Here is the caller graph for this function:

5.4.3.6 clear()

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

Here is the caller graph for this function:

5.4.3.7 data()

```
template<typename T>
T * Vector< T >::data () const  [inline]
```

Here is the caller graph for this function:

5.4.3.8 empty()

```
template<typename T>
bool Vector< T >::empty () const  [inline], [noexcept]
```

Here is the caller graph for this function:

5.4.3.9 end() [1/2]

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

Here is the caller graph for this function:

5.4.3.10 end() [2/2]

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

5.4.3.11 erase()

```
template<typename T>
void Vector< T >::erase (
            int index)  [inline]
```

5.4.3.12 front()

```
template<typename T>
const T & Vector< T >::front () const  [inline]
```

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

5.4.3.13 insert()

```
template<typename T>
void Vector< T >::insert (
            int index,
            const T & value)  [inline]
```

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

5.4.3.14 operator"!=()

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

Funkcijos kvietimo grafas:

5.4.3.15 operator<()

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

Funkcijos kvietimo grafas:

5.4.3.16 operator<=()

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

Funkcijos kvietimo grafas:

5.4.3.17 operator=() [1/2]

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

Funkcijos kvietimo grafas:

5.4.3.18 operator=() [2/2]

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

Funkcijos kvietimo grafas:

5.4.3.19 operator==()

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

Funkcijos kvietimo grafas:

5.4.3.20 operator>()

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

Funkcijos kvietimo grafas:

5.4.3.21 operator>=()

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

Funkcijos kvietimo grafas:

5.4.3.22 operator[]() [1/2]

```
template<typename T>
T & Vector< T >::operator[] (
            int index)  [inline]
```

5.4.3.23 operator[]() [2/2]

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

5.4.3.24 pop_back()

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

Here is the caller graph for this function:

5.4.3.25 pop_front()

```
template<typename T>
void Vector< T >::pop_front ()  [inline]
```

Here is the caller graph for this function:

5.4.3.26 push_back() [1/2]

```
template<typename T>
void Vector< T >::push_back (
            const T & value)  [inline]
```

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

5.4.3.27 push_back() [2/2]

```
template<typename T>
void Vector< T >::push_back (
            T && value)  [inline]
```

Funkcijos kvietimo grafas:

5.4.3.28 push_front()

```
template<typename T>
void Vector< T >::push_front (
            const T & value)  [inline]
```

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

5.4.3.29 reserve()

```
template<typename T>
void Vector< T >::reserve (
            int newCapacity)  [inline]
```

Here is the caller graph for this function:

5.4.3.30 resize()

```
template<typename T>
void Vector< T >::resize (
            int newSize)  [inline]
```

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

5.4.3.31 shrink_to_fit()

```
template<typename T>
void Vector< T >::shrink_to_fit ()  [inline]
```

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

5.4.3.32 size()

```
template<typename T>
int Vector< T >::size () const  [inline], [noexcept]
```

Here is the caller graph for this function:

5.4.3.33 swap()

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

Funkcijos kvietimo grafas:

5.4.4 Draugiškų Ir Susijusių Funkcijų Dokumentacija

5.4.4.1 operator<<

```
template<typename T>
ostream & operator<< (
            ostream & out,
            const Vector< T > & other)  [friend]
```

5.4.4.2 operator>>

```
template<typename T>
istream & operator>> (
            istream & in,
            Vector< T > & other)  [friend]
```

5.4.5 Atributų Dokumentacija

5.4.5.1 capacity_

```
template<typename T>
size_t Vector< T >::capacity_  [private]
```

5.4.5.2 data_

```
template<typename T>
T* Vector< T >::data_  [private]
```

5.4.5.3 size_

```
template<typename T>
size_t Vector< T >::size_  [private]
```

Dokumentacija šiai klasei sugeneruota iš šio failo:

- include/vector.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

- 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)
```

Functions that are used in the main function.
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:

6.3.2 Kintamojo Dokumentacija

6.3.2.1 globalTime

```
double globalTime  [extern]
```

6.3.2.2 maxStud

```
const int maxStud = 10000000
```

Global variables.

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::stringstream;
00018 using std::system_error;
00019
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 int NumberCheck(int min, int max);
00046 int Menu();
00047 void ProgramEnd();
00048 void ProcessException();
00049
00051 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"};
00052 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:
00012     Human(const string &name = "Vardenis", const string &surname = "Pavardenis")
00013         : name_(name), surname_(surname) {}
00014
00015     virtual ~Human()
00016     {
00017         //cout « "[~] Base destructor called for: " « name_ « " " « surname_ « endl;
00018         name_.clear();
00019         surname_.clear();
00020     }
00021
00023     string getName() const { return name_; }
00024     string getSurname() const { return surname_; }
00025     void setName(const string &newName) { name_ = newName; }
00026     void setSurname(const string &newSurname) { surname_ = newSurname; }
00027
00028     virtual void print() const = 0;
00029 };
00030
00032 class Student : public Human
00033 {
00034 private:
00035     vector<int> marks_;
00036     int exam_;
00037     double average_, median_;
00038
00039 public:
00041     Student(const string &name = "Vardenis", const string &surname = "Pavardenis", int exam = 0)
00042         : Human(name, surname), exam_(exam), average_(0.0), median_(0.0) { marks_.reserve(20); }
00043
00044     ~Student()
00045     {
00046         //cout « "[~] Destructor called for: " « name_ « " " « surname_ « endl;
00047         marks_.clear();
00048     }
00049
00051     Student(const Student &other);
```

```
00052     Student(Student &&other);
00053
00055     Student &operator=(const Student &other);
00056     Student &operator=(Student &&other);
00057
00059     friend ostream &operator«(ostream &out, const Student &student);
00060     friend istream &operator»(istream &in, Student &student);
00061
00063     const vector<int> &getMarks() const { return marks_; }
00064     int getExam() const { return exam_; }
00065     double getAverage() const { return average_; }
00066     double getMedian() const { return median_; }
00067
00068     void addMark(int mark) { marks_.push_back(move(mark)); }
00069     void setExam(int newExam) { exam_ = newExam; }
00070
00072     void print() const;
00073     void calculateAverage();
00074     void calculateMedian();
00075     void readLine(const string &line);
00076 };
```

6.7 include/templates.h Failo Nuoroda

`#include "../include/vector.h"`
Įtraukimo priklausomybių diagrama templates.h: 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 #include "../include/vector.h"
00003
00005 template <typename Container>
00006 void ReadFromFile(Container &group, int action)
00007 {
00008     string readName;
00009     bool fileLoaded = false;
00010     while (!fileLoaded)
00011     {
00012         cout « "Iveskite failo pavadinima, is kurio bus skaitomi duomenys: " « endl;
00013         cin » readName;
00014         try
00015         {
00016             ifstream input(readName, std::ios::binary);
00017             if (!input)
```

```
00018                 throw std::ios_base::failure("Failas nerastas arba negali buti atidarytas.");
00019             else
00020             {
00021                 fileLoaded = true;
00022                 Timer inputTime;
00023                 string line;
00024                 getline(input, line);
00025                 while (getline(input, line))
00026                 {
00027                     Student temp;
00028                     temp.readLine(line);
00029                     temp.calculateAverage();
00030                     temp.calculateMedian();
00031                     group.push_back(move(temp));
00032                 }
00033                 input.close();
00034                 cout « " * Duomenu skaitymas uztruko: " « inputTime.elapsed() « " sekundziu. " « endl;
00035                 globalTime += inputTime.elapsed();
00036             }
00037         }
00038         catch (...)
00039         {
00040             ProcessException();
00041             cin.clear();
00042             cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00043         }
00044     }
00045     if (action != 6)
00046     {
00047         string writeName = "rezultatas.txt";
00048         ofstream output(writeName);
00049         int markAction;
00050         Sort(group, markAction);
00051         Output(group, output, markAction);
00052         output.close();
00053         cout « "Duomenys nukopijuoti i faila: " « writeName « endl;
00054     }
00055 }
00056
00058 template <typename Container>
00059 void Action(Container &group, int action)
00060 {
00061     cout « "Iveskite studentu skaiciu (iveskite 0, jei skaicius yra nezinomas): " « endl;
00062     int amountStud = NumberCheck(0, maxStud);
00063     bool amountStudKnown = (amountStud != 0);
00064     if (!amountStudKnown)
00065         amountStud = maxStud;
00066
00067     for (int i = 0; i < amountStud; i++)
00068     {
00069         Student temp;
00070         if (action == 2)
00071         {
00072             string name, surname;
00073             cout « "Iveskite studento varda: ";
00074             cin » name;
00075             temp.setName(name);
00076             cout « "Iveskite studento pavarde: ";
00077             cin » surname;
00078             temp.setSurname(surname);
00079         }
00080         else if (action == 3)
00081         {
00082             temp.setName(names[rand() % names.size()]);
00083             temp.setSurname(surnames[rand() % surnames.size()]);
00084         }
00085         if (action == 2 || action == 3)
00086         {
00087             int amountMarks = rand() % 100 + 1;
00088             for (int j = 0; j < amountMarks; ++j)
00089                 temp.addMark(rand() % 10 + 1);
00090             temp.setExam(rand() % 10 + 1);
00091             temp.calculateAverage();
00092             temp.calculateMedian();
00093         }
00094         else if (action == 1)
00095             cin » temp;
00096         group.push_back(move(temp));
00097         if (!amountStudKnown)
00098         {
00099             cout « "1 - ivesti dar vieno studento duomenis; 0 - baigti ivedima. " « endl;
00100             if (NumberCheck(0, 1) == 0)
00101                 break;
00102         }
00103     }
00104 }
00105
```

```
00107 template <typename Container>
00108 double Sort(Container &group, int &markAction)
00109 {
00110     cout « "Pasirinkite rezultatu isvedimo metoda: " « endl;
00111     cout « "1 - gauti vidurkius; 2 - gauti medianas. " « endl;
00112     markAction = NumberCheck(1, 2);
00113     cout « "Pairinkite rezultatu rusiavimo metoda: " « endl;
00114     cout « "1 - rusiuoti pagal varda (A-Z); 2 - rusiuoti pagal pavarde (A-Z); 3 - rusiuoti pagal
      galutini pazymi." « endl;
00115     int sortAction = NumberCheck(1, 3);
00116
00117     Timer sortTime;
00118     auto compare = [&](const Student &a, const Student &b)
00119     {
00120         if (sortAction == 1)
00121             return a.getName() < b.getName();
00122         if (sortAction == 2)
00123             return a.getSurname() < b.getSurname();
00124         if (sortAction == 3)
00125             return (markAction == 1) ? (a.getAverage() < b.getAverage()) : (a.getMedian() <
      b.getMedian());
00126         return false;
00127     };
00128     if constexpr (std::is_same_v<Container, std::vector<Student» || std::is_same_v<Container,
      deque<Student» || std::is_same_v<Container, Vector<Student»)
00129     {
00130         sort(group.begin(), group.end(), compare);
00131     }
00132     else if constexpr (std::is_same_v<Container, list<Student»)
00133     {
00134         group.sort(compare);
00135     }
00136     return sortTime.elapsed();
00137 }
00138
00140 template <typename Container>
00141 void Output(Container &group, ostream &out, int markAction)
00142 {
00143     Timer outputTime;
00144     out « left « setw(20) « "Pavarde" « setw(20) « "Vardas";
00145     if (markAction == 1)
00146         out « setw(20) « "Galutinis (Vid.)" « endl;
00147     else if (markAction == 2)
00148         out « setw(20) « "Galutinis (Med.)" « endl;
00149     out « "-------------------------------------------------------------" « endl;
00150     for (auto &final : group)
00151     {
00152         out « left « setw(20) « final.getSurname() « setw(20) « final.getName();
00153         if (markAction == 1)
00154             out « setw(20) « fixed « setprecision(2) « final.getAverage() « endl;
00155         else if (markAction == 2)
00156             out « setw(20) « fixed « setprecision(2) « final.getMedian() « endl;
00157     }
00158     globalTime += outputTime.elapsed();
00159     cout « " * Rezultatu isvedimas uztruko: " « outputTime.elapsed() « " sekundziu. " « endl;
00160 }
00161
00163 template <typename Container>
00164 void SeparateStudents(Container &group, Container &failed)
00165 {
00166     Timer separationTime;
00167     Container passed;
00168     for (const auto &student : group)
00169     {
00170         if (student.getAverage() < 5)
00171             failed.push_back(student);
00172         else
00173             passed.push_back(student);
00174     }
00175     group.clear();
00176     for (auto& s : passed)
00177         group.push_back(std::move(s));
00178
00179     cout « " * Studentu skirstymas i 2 kategorijas uztruko: " « separationTime.elapsed() « "
      sekundziu. " « endl;
00180     globalTime += separationTime.elapsed();
00181 }
00182
00184 template <typename Container>
00185 void OutputSeparated(Container &group, Container &failed)
00186 {
00187     int markAction;
00188     double sortTime1 = Sort(group, markAction);
00189     ofstream passedOut("kietiakai.txt");
00190     Timer passedTime;
00191     Output(group, passedOut, markAction);
00192     double outTime1 = passedTime.elapsed();
```

```
00193     passedOut.close();
00194     cout « "Kietiakai surasyti i faila: kietiakai.txt." « endl;
00195     double sortTime2 = Sort(failed, markAction);
00196     cout « " * Studentu rusiavimas uztruko: " « sortTime1 + sortTime2 « " sekundziu. " « endl;
00197     globalTime += sortTime1 + sortTime2;
00198     ofstream failedOut("vargsiukai.txt");
00199     Timer failedTime;
00200     Output(failed, failedOut, markAction);
00201     double outTime2 = failedTime.elapsed();
00202     failedOut.close();
00203     cout « "Vargsiukai surasyti i faila: vargsiukai.txt." « endl;
00204     cout « " * Rezultatu isvedimas i 2 failus uztruko: " « outTime1 + outTime2 « " sekundziu. " «
      endl;
00205 }
00206
00208 template <typename Container>
00209 void GenerateFile(Container &group)
00210 {
00211     cout « "Iveskite failo pavadinima, i kuri bus irasyti duomenys: " « endl;
00212     string fout;
00213     cin » fout;
00214     cout « "Iveskite studentu skaiciu, kuriu informacija norite sugeneruoti: " « endl;
00215     int amountStud = NumberCheck(1, maxStud);
00216     Timer generateTime;
00217     int amountMarks = rand() % 11 + 10;
00218     for (int i = 1; i <= amountStud; i++)
00219     {
00220         Student temp;
00221         temp.setName("VardasNr" + std::to_string(i));
00222         temp.setSurname("PavardeNr" + std::to_string(i));
00223         for (int j = 0; j < amountMarks; j++)
00224             temp.addMark((rand() % 10 + 1));
00225         temp.setExam(rand() % 10 + 1);
00226         group.push_back(move(temp));
00227     }
00228     ofstream out(fout);
00229     out « left « setw(20) « "Vardas" « setw(20) « "Pavarde";
00230     for (int i = 1; i <= amountMarks; i++)
00231         out « left « setw(10) « ("ND" + std::to_string(i));
00232     out « setw(10) « "Egz." « endl;
00233     for (auto &final : group)
00234     {
00235         out « left « setw(20) « final.getName() « setw(20) « final.getSurname();
00236         for (auto mark : final.getMarks())
00237             out « left « setw(10) « mark;
00238         out « setw(10) « final.getExam() « endl;
00239     }
00240     out.close();
00241     cout « "Duomenys buvo sekmingai sukurti faile: " « fout « endl;
00242     cout « " * Duomenu generavimas uztruko: " « generateTime.elapsed() « " sekundziu. " « endl;
00243     globalTime += generateTime.elapsed();
00244 }
```

6.9 include/vector.h Failo Nuoroda

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

Klasės

- class Vector< T >

6.10 vector.h

Eiti į šio failo dokumentaciją.

```
00001 #pragma once
00002
00003 // ** Class that implements a dynamic array (vector) */
00004 template <typename T>
00005 class Vector
00006 {
00007 private:
00008     size_t size_;           // Current number of elements in the vector
00009     size_t capacity_;       // Current capacity of the vector
00010     T *data_;            // Pointer to to dynamically allocated data
00011
00012 public:
00013     // =============
00014     // Member types
00015     // =============
```

```
00016     using value_type = T;                        // Type of the elements in the vector
00017     using size_type = std::size_t;               // Size type for the vector
00018     using difference_type = std::ptrdiff_t;      // Difference type for the vector
00019     using reference = value_type &;              // Reference type for the vector
00020     using const_reference = const value_type &; // Const reference type for the vector
00021     using pointer = value_type *;                // Pointer type for the vector
00022     using const_pointer = const value_type *;    // Const pointer type for the vector
00023     using iterator = pointer;                    // Iterator type for the vector
00024     using const_iterator = const_pointer;        // Const iterator type for the vector
00025
00026     // ============================
00027     // Constructors and Destructor
00028     // ============================
00030     Vector() : size_(0), capacity_(5), data_(new T[capacity_]) {};
00032     Vector(const Vector &other) : size_(other.size()), capacity_(other.capacity()), data_(new
      T[capacity_])
00033     {
00034         for (int i = 0; i < other.size(); ++i)
00035         {
00036             data_[i] = other.data_[i];
00037         }
00038     }
00040     Vector(Vector &&other) : size_(other.size()), capacity_(other.capacity()), data_(other.data_)
00041     {
00042         other.data_ = nullptr;
00043         other.size_ = 0;
00044         other.capacity_ = 0;
00045     }
00047     Vector(int elements, T value = T()) : size_(elements), capacity_(elements + 5), data_(new
      T[capacity_])
00048     {
00049         for (int i = 0; i < size_; ++i)
00050         {
00051             data_[i] = value;
00052         }
00053     }
00055     Vector(const std::initializer_list<T> &list) : size_(0), capacity_(list.size() + 5), data_(new
      T[capacity_])
00056     {
00057         for (const T &i : list)
00058         {
00059             push_back(i);
00060         }
00061     }
00063     ~Vector()
00064     {
00065         delete[] data_;
00066     }
00067
00068     // ========================
00069     // First element functions
00070     // ========================
00071     /* Returns reference to the first element */
00072     const T &front() const
00073     {
00074         if (empty())
00075         {
00076             throw std::out_of_range("Vectorius yra tuscias, negalima pasiekti elemento.");
00077         }
00078         return data_[0];
00079     }
00080     /* Returns iterator to the first element */
00081     T* begin()
00082     {
00083         return data_;
00084     }
00085     /* Returns const iterator to the first element */
00086     const T* begin() const
00087     {
00088         return data_;
00089     }
00090     /* Adds an element to the beginning of the vector */
00091     void push_front(const T &value)
00092     {
00093         if (size_ < capacity_)
00094         {
00095             for (int i = size_; i > 0; --i)
00096             {
00097                 data_[i] = data_[i - 1];
00098             }
00099             data_[0] = value;
00100             ++size_;
00101         }
00102         else
00103         {
00104             int newCapacity = (capacity_ == 0) ? 5 : capacity_ * 2;
00105             reserve(newCapacity);
```

```
00106             for (int i = size_; i > 0; --i)
00107             {
00108                 data_[i] = data_[i - 1];
00109             }
00110             data_[0] = value;
00111             ++size_;
00112         }
00113     }
00114     /* Removes the first element from the vector */
00115     void pop_front()
00116     {
00117         if (size_ == 0)
00118         {
00119             throw std::out_of_range("Vectorius yra tuscias, negalima istrinti elemento.");
00120         }
00121         for (int i = 0; i < size_ - 1; ++i)
00122         {
00123             data_[i] = data_[i + 1];
00124         }
00125         --size_;
00126     }
00127
00128     // ======================
00129     // Last element functions
00130     // ======================
00131     /* Returns reference to the last element */
00132     const T &back() const
00133     {
00134         if (empty())
00135         {
00136             throw std::out_of_range("Vectorius yra tuscias, negalima pasiekti elemento.");
00137         }
00138         return data_[size_ - 1];
00139     }
00140     /* Returns iterator to the last element */
00141     T* end()
00142     {
00143         return data_ + size_;
00144     }
00145     /* Returns const iterator to the last element */
00146     const T* end() const
00147     {
00148         return data_ + size_;
00149     }
00150     /* Adds an element to the end of the vector */
00151     void push_back(const T& value)
00152     {
00153         if (size_ == capacity_)
00154         {
00155             reserve(capacity_ * 2);
00156         }
00157         new(&data_[size_]) T(value);
00158         ++size_;
00159     }
00160     void push_back(T&& value)
00161     {
00162         if (size_ == capacity_)
00163         {
00164             reserve(capacity_ * 2);
00165         }
00166         new(&data_[size_]) T(std::move(value));
00167         ++size_;
00168     }
00169     /* Removes the last element from the vector */
00170     void pop_back()
00171     {
00172         if (size_ == 0)
00173         {
00174             throw std::out_of_range("Vectorius yra tuscias, negalima istrinti elemento.");
00175         }
00176         --size_;
00177     }
00178
00179     // ================
00180     // Other functions
00181     // ================
00182     /* Returns the size of the vector */
00183     int size() const noexcept
00184     {
00185         return size_;
00186     }
00187     /* Resizes the vector to the specified size */
00188     void resize(int newSize)
00189     {
00190         if (newSize < 0)
00191         {
00192             throw std::invalid_argument("Neteisingas dydis. Dydis negali buti neigiamas.");
```

```
00193         }
00194         if (newSize > capacity_)
00195         {
00196             reserve(newSize + 5);
00197         }
00198         if (newSize > size_)
00199         {
00200             for (int i = size_; i < newSize; ++i)
00201             {
00202                 data_[i] = T();
00203             }
00204         }
00205         size_ = newSize;
00206     }
00207     /* Returns the capacity of the vector */
00208     int capacity() const noexcept
00209     {
00210         return capacity_;
00211     }
00212     /* Reserves space for the specified number of elements */
00213     void reserve(int newCapacity)
00214     {
00215         if (newCapacity <= capacity_)
00216             return;
00217         T *newData = new T[newCapacity];
00218         for (int i = 0; i < size_; ++i)
00219         {
00220             newData[i] = move(data_[i]);
00221         }
00222         delete[] data_;
00223         data_ = newData;
00224         capacity_ = newCapacity;
00225     }
00226     /* Shrinks the capacity of the vector to fit its size */
00227     void shrink_to_fit()
00228     {
00229         if (size_ == capacity_)
00230             return;
00231         if (size_ == 0)
00232         {
00233             delete[] data_;
00234             data_ = nullptr;
00235             capacity_ = 0;
00236         }
00237         else if (capacity_ > size_)
00238         {
00239             reserve(size_);
00240         }
00241     }
00242     /* Checks if the vector is empty */
00243     bool empty() const noexcept
00244     {
00245         return size_ == 0;
00246     }
00247     /* Clears the vector */
00248     void clear()
00249     {
00250         size_ = 0;
00251     }
00252     /* Swaps the contents of this vector with another */
00253     void swap(Vector &other)
00254     {
00255         std::swap(size, other.size);
00256         std::swap(capacity, other.capacity);
00257         std::swap(data_, other.data);
00258     }
00259
00260     // =========================
00261     // Element access functions
00262     // =========================
00263     /* Accesses data */
00264     T *data() const
00265     {
00266         return data_;
00267     }
00268     /* Accesses an element by index without bounds checking */
00269     T &operator[] (int index)
00270     {
00271         return data_[index];
00272     }
00273     const T &operator[] (int index) const
00274     {
00275         return data_[index];
00276     }
00277     /* Accesses an element by index with bounds checking */
00278     T &at(int index)
00279     {
```

```
00280         if (index < 0 || index >= size_)
00281         {
00282             throw std::out_of_range("Indeksas uz ribu. Negalima pasiekti elemento.");
00283         }
00284         return data_[index];
00285     }
00286     /* Inserts an element at the specified index */
00287     void insert(int index, const T &value)
00288     {
00289         if (index < 0 || index > size_)
00290         {
00291             throw std::out_of_range("Indeksas uz ribu. Negalima iterpti elemento.");
00292         }
00293         if (size_ != capacity_)
00294         {
00295             for (int i = size_ - 1; i >= index; --i)
00296             {
00297                 data_[i + 1] = data_[i];
00298             }
00299             data_[index] = value;
00300             ++size_;
00301         }
00302         else
00303         {
00304             int newCapacity = (capacity_ == 0) ? 5 : capacity_ * 2;
00305             reserve(newCapacity);
00306             insert(index, value);
00307         }
00308     }
00309     /* Removes an element at the specified index */
00310     void erase(int index)
00311     {
00312         if (index < 0 || index >= size_)
00313         {
00314             throw std::out_of_range("Indeksas uz ribu. Negalima istrinti elemento.");
00315         }
00316         for (int i = index; i < size_ - 1; ++i)
00317         {
00318             data_[i] = data_[i + 1];
00319         }
00320         --size_;
00321     }
00322
00323     // ===========
00324     // Operators
00325     // ===========
00326     /* Copy assignment operator */
00327     Vector &operator=(const Vector &other)
00328     {
00329         if (this != &other)
00330         {
00331             if (other.size_ > size_)
00332             {
00333                 delete[] data_;
00334                 capacity_ = other.size_ + 5;
00335                 data_ = new T[capacity_];
00336             }
00337             for (int i = 0; i < other.size(); ++i)
00338             {
00339                 data_[i] = other.data_[i];
00340             }
00341             size_ = other.size_;
00342         }
00343         return *this;
00344     }
00345     /* Move assignment operator */
00346     Vector &operator=(Vector &&other) noexcept
00347     {
00348         if (this != &other)
00349         {
00350             delete[] data_;
00351             data_ = other.data_;
00352             size_ = other.size_;
00353             capacity_ = other.capacity_;
00354             other.data_ = nullptr;
00355             other.size_ = 0;
00356             other.capacity_ = 0;
00357         }
00358         return *this;
00359     }
00360     /* Compares this vector with another for equality */
00361     bool operator==(const Vector &other) const
00362     {
00363         if (size() != other.size())
00364             return false;
00365         for (size_t i = 0; i < size(); ++i)
00366         {
```

```
00367             if (data_[i] != other.data_[i])
00368                 return false;
00369         }
00370         return true;
00371     }
00372     /* Compares this vector with another for inequality */
00373     bool operator!=(const Vector &other) const
00374     {
00375         return !(*this == other);
00376     }
00377     /* Compares this vector with another for less than */
00378     bool operator<(const Vector &other) const
00379     {
00380         int minSize = std::min(size_, other.size_);
00381         for (int i = 0; i < minSize; ++i)
00382         {
00383             if (data_[i] < other.data_[i])
00384                 return true;
00385             else if (data_[i] > other.data_[i])
00386                 return false;
00387         }
00388         return size_ < other.size_;
00389     }
00390     /* Compares this vector with another for less than or equal to */
00391     bool operator<=(const Vector &other) const
00392     {
00393         return (*this < other) || (*this == other);
00394     }
00395     /* Compares this vector with another for greater than */
00396     bool operator>(const Vector &other) const
00397     {
00398         return other < *this;
00399     }
00400     /* Compares this vector with another for greater than or equal to */
00401     bool operator>=(const Vector &other) const
00402     {
00403         return (other < *this) || (*this == other);
00404     }
00405
00406     // ============================
00407     // Overloaded input and output
00408     // ============================
00409     /* Overloads the stream output operator for printing */
00410     friend ostream &operator<<(ostream &out, const Vector &other)
00411     {
00412         for (int i = 0; i < other.size_; ++i)
00413         {
00414             out << other.data_[i] << " ";
00415         }
00416         out << " || ";
00417         for (int i = other.size(); i < other.capacity_; ++i)
00418         {
00419             out << other.data_[i] << " ";
00420         }
00421         out << endl;
00422         return out;
00423     }
00424     /* Overloads the stream input operator for reading */
00425     friend istream &operator>>(istream &in, Vector &other)
00426     {
00427     int size;
00428     in >> size;
00429     other.Resize(size);
00430     for (int i = 0; i < size; ++i)
00431     {
00432         in >> other.data[i];
00433     }
00434     return in;
00435     }
00436 };
```

6.11 README.md Failo Nuoroda

6.12 src/functions.cpp Failo Nuoroda

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

Įtraukimo priklausomybių diagrama functions.cpp:

Funkcijos

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

Kintamieji

- double globalTime = 0

6.12.1 Funkcijos Dokumentacija

6.12.1.1 Menu()

```
int Menu ()
```

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

6.12.1.2 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:

6.12.1.3 ProcessException()

```
void ProcessException ()
```

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

6.12.1.4 ProgramEnd()

```
void ProgramEnd ()
```

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

6.12.2 Kintamojo Dokumentacija

6.12.2.1 globalTime

```
double globalTime = 0
```

6.13 src/main.cpp Failo Nuoroda

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

Įtraukimo priklausomybių diagrama main.cpp:

Funkcijos

- int main ()

6.13.1 Funkcijos Dokumentacija

6.13.1.1 main()

```
int main ()
```

Funkcijos kvietimo grafas:

6.14 src/student.cpp Failo Nuoroda

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

Įtraukimo priklausomybių diagrama student.cpp:

Funkcijos

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

6.14.1 Funkcijos Dokumentacija

6.14.1.1 operator<<()

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

Overloaded output operator for Student class

6.14.1.2 operator>>()

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

Overloaded input operator for Student class

6.15 src/tests.cpp Failo Nuoroda

```
#include "../catch2/catch.hpp"
#include "../include/global.h"
#include "../include/headers.h"
#include "../include/student.h"
#include "../include/templates.h"
#include "../include/vector.h"
```

Įtraukimo priklausomybių diagrama tests.cpp:

Apibrėžimai

- #define CATCH_CONFIG_MAIN

Funkcijos

- TEST_CASE ("Constructors", "[Vector]")
- TEST_CASE ("front() and back()", "[Vector]")
- TEST_CASE ("push_back() and push_front(), pop_back() and pop_front()", "[Vector]")
- TEST_CASE ("clear() and empty(), size()", "[Vector]")
- TEST_CASE ("reserve() and capacity()", "[Vector]")

- TEST_CASE ("resize() and shrink_to_fit()", "[Vector]")
- TEST_CASE ("operator [ ]", "[Vector]")
- TEST_CASE ("insert() and erase()", "[Vector]")
- TEST_CASE ("at() test", "[Vector]")
- TEST_CASE ("Copy and move assignment operator", "[Vector]")
- TEST_CASE ("Comparison operators", "[Vector]")
- TEST_CASE ("Student klases Rule of five testai")
- TEST_CASE ("Student konstruktorius su parametrais")
- TEST_CASE ("Student ivesties/isvesties operatoriai")
- TEST_CASE ("Abstrakti klase Human naudojama su Student rodykle")

6.15.1 Apibrėžimų Dokumentacija

6.15.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

6.15.2 Funkcijos Dokumentacija

6.15.2.1 TEST_CASE() [1/15]

```
TEST_CASE (
            "Abstrakti klase Human naudojama su Student rodykle" )
```

Funkcijos kvietimo grafas:

6.15.2.2 TEST_CASE() [2/15]

```
TEST_CASE (
            "at() test" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.3 TEST_CASE() [3/15]

```
TEST_CASE (
            " clear) and empty(,
            size()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.4 TEST_CASE() [4/15]

```
TEST_CASE (
            "Comparison operators" ,
            "" [Vector])
```

6.15.2.5 TEST_CASE() [5/15]

```
TEST_CASE (
            "Constructors" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.6 TEST_CASE() [6/15]

```
TEST_CASE (
            "Copy and move assignment operator" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.7 TEST_CASE() [7/15]

```
TEST_CASE (
            "front() and back()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.8 TEST_CASE() [8/15]

```
TEST_CASE (
            "insert() and erase()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.9 TEST_CASE() [9/15]

```
TEST_CASE (
            "operator " [],
            "" [Vector])
```

6.15.2.10 TEST_CASE() [10/15]

```
TEST_CASE (
            " push_back) and push_front(,
            pop_back() and pop_front()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.11 TEST_CASE() [11/15]

```
TEST_CASE (
            "reserve() and capacity()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.12 TEST_CASE() [12/15]

```
TEST_CASE (
            "resize() and shrink_to_fit()" ,
            "" [Vector])
```

Funkcijos kvietimo grafas:

6.15.2.13 TEST_CASE() [13/15]

```
TEST_CASE (
            "Student ivesties/isvesties operatoriai" )
```

Funkcijos kvietimo grafas:

6.15.2.14 TEST_CASE() [14/15]

```
TEST_CASE (
            "Student klases Rule of five testai" )
```

Funkcijos kvietimo grafas:

6.15.2.15 TEST_CASE() [15/15]

```
TEST_CASE (
            "Student konstruktorius su parametrais" )
```

Funkcijos kvietimo grafas:

Rodyklė