Chapter 1

v3.0

1.1 Diegimo instrukcija

- Atsisiųskite projekto kodą iš GitHub naudodami `git clone` komandą su projekto URL:
  `git clone https://github.com/nojusta/OOP_Lab3`

- Pereikite į projekto katalogą naudodami cd komandą. Pavyzdžiui:

```
cd *projekto vieta kompiuteryje*
```

- Sukurkite Makefile su reikiamomis taisyklėmis. Jūsų Makefile turėtų atrodyti maždaug taip (Unix OS atveju):

```
# Kompiliatorius
CXX = g++

# Kompiliatoriaus parametrai
CXXFLAGS = -std=c++20 -O3 -mmacosx-version-min=14.3

# Vykdomo failo pavadinimas
TARGET = v3

# Source failai
SRCS = main.cpp functionality.cpp input.cpp calculations.cpp student.cpp

# Object failai
OBJS = $(SRCS:.cpp=.o)

# Google Test biblioteka
GTEST = /usr/local/lib/libgtest.a /usr/local/lib/libgtest_main.a

# Bibliotekos
LIBS = $(GTEST)

# Testuojami failai
TEST_SRCS = myVector_test.cpp student_test.cpp

# Testuojami objekto failai
TEST_OBJS = $(TEST_SRCS:.cpp=.o)

# Testuojamo failo pavadinimas
```

```
TEST_TARGET = myVector_test student_test

# Taisyklė programa susieti
$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJS)

# Taisyklė testams susieti
myVector_test: student.o myVector_test.o
    $(CXX) $(CXXFLAGS) -o myVector_test student.o myVector_test.o $(LIBS)

student_test: student.o student_test.o
    $(CXX) $(CXXFLAGS) -o student_test student.o student_test.o $(LIBS)

# Taisyklė kompiliuoti failus
.cpp.o:
    $(CXX) $(CXXFLAGS) -c $<  -o $@

# Taisyklė išvalyti tarpinius failus
clean:
    $(RM) $(OBJS) $(TEST_OBJS)

# Taisyklė išvalyti viską
distclean: clean
    $(RM) $(TARGET) myVector_test student_test
```

- Sukompiliuokite programą naudodami make komandą:
  `make`

  Tada gausite tokį rezultatą:

- Paleiskite programą naudodami šią komandą:
  `./v3`

1.1.1 Valymo instrukcija

- Jei norite išvalyti sukompiliuotus failus, galite naudoti šias make komandas:
  `make clean`
  `make distclean`

1.2 Naudojimosi instrukcija

- Paleiskite programą naudodami šią komandą:
  `./v3`
- Programa pateiks meniu su įvairiomis funkcijomis. Pasirinkite funkciją įvedę atitinkamą numerį ir spauskite `Enter`.

1.2.1 Programos funkcijos:

- Nuskaito duomenis iš naudotojo arba failo ir patikrina ar jie yra teisingi (naudojant išimčių valdymą).
- Duoda naudotojui galimybė pasirinkti du galutinio balo skaičiavimo būdus - skaičiuojant vidurkį ar medianą.

- Leidžia pasirinkti 5 skirtingus būdus įvesti, nuskaityti ar sugeneruoti duomenis.
- Dinamiškai paskiria atmintį pagal įvesta / nuskaitytą duomenų kiekį.
- Atidaro testavimo failus ir apskaičiuoja laiką, kurį praleidžia apdorojant duomenis iš failų.
- Visi pranešimai išvedami lietuvių kalbą.
- Projektas išskaidytas į kelis failus (.h ir .cpp).
- Generuoja penkis atsitiktinius studentų sąrašų failus, sudarytus isˇ: 1 000, 10 000, 100 000, 1 000 000, 10 000 000 įrasˇų
- Atlieka tyrimus / testavimus su sugeneruotais failais.
- Surūšiuoja studentus ir išveda į du naujus failus.
- Yra 3 skirtingi konteinerio tipo pasirinkimai testavimui - vector, deque, list.
- Yra 3 skirtingos strategijos duomenų skirstymui.
- Naudojama klasė, saugojant studentų duomenis.
- Galima testuoti visus "Rule of five" konstruktorius ir I/O operatorius.
- Yra abstrakti klasė Person. Student klasė yra Person klasės išvestinė klasė.
- DoxyGen sugeneruota HTML/TEX formato dokumentacija.
- Atlikti unit testai su Google Test.
- Realizuota pilnavertė alternatyvą std::vector konteineriui - MyVector. MyVector turi visus Member tipus Member funkcijas, Non-member funkcijas reikalingas std::vector funkcionalumui.

Norėdami baigti darbą su programa, pasirinkite atitinkamą skaičių.

1.3 Efektyvumo/spartos analizė naudojant std::vector ir MyVector konteinerius.

Visi testavimai buvo atlliekami naudojant O3 optimizavimo flag'ą.

Matuojame kiek laiko užtrunka užpildyti konteinerius su 10000, 100000, 1000000, 10000000 ir 100000000 int elementų naudojant push_back() funkciją:

1.3.1 Išvados

Visais atvejais MyVector konteineris yra greitesnis nei std::vector konteineris. Tai rodo, kad MyVector konteineris yra efektyvesnis nei std::vector konteineris, kai naudojama push_back() funkcija.

Atminities paskirstymai yra panašūs, tačiau MyVector konteineris yra šiek tiek efektyvesnis nei std::vector konteineris.

1.4 Konteinerių testavimas naudojant duomenų failą su 100 000, 1 000 000 ir 10 000 000 studentų įrašų

Visi testavimai buvo atlliekami su konteinerių testavimo 1 strategija, naudojant O3 optimizavimo flag'ą.

1.4.1 Naudojant std::vector tipo konteinerį:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
100 000	0.251s	0.027s	0.011s	0.290s
1 000 000	2.112s	0.348s	0.235s	2.696s
10 000 000	21.528s	5.458s	4.983s	31.970s

rezultatai

1.4.2 Naudojant MyVector konteinerį:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
100 000	0.249s	0.011s	0.004s	0.265s
1 000 000	2.120s	0.004s	0.001s	2.126s
10 000 000	21.223s	1.021s	1.021s	22.985s

rezultatai

1.4.3 Išvados

Visais atvejais MyVector konteineris yra greitesnis nei std::vector konteineris. Skirtumas ypatingai pasimato su didesniais duomenų kiekiais.

1.5 Unit testai naudojant Google Test.

Šis projektas naudoja Google Test unit testavimui. Norėdami paleisti testus, sekite šiuos žingsnius:

1.5.1 MyVector konteinerio testavimas

1. Sukurkite testavimo failą naudodami `make myVector_test`.
2. Paleiskite testus su `./myVector_test`.

Išvestis turėtų atrodyti maždaug taip:

1.5.2 Studento klasės testavimas

1. Sukurkite testavimo failą naudodami `make student_test`.
2. Paleiskite testus su `./student_test`.

Išvestis turėtų atrodyti maždaug taip:

1.6 MyVector funkcijos

1.6.1 push_back

`push_back` funkcija prideda elementą į vektoriaus pabaigą. Jei vektorius yra pilnas (t.y., `current == capacity`), tai išplečia vektoriaus talpą dvigubai (`reserve(capacity == 0 ? 1 : capacity * 2)`), ir tada prideda elementą.

```
void push_back(const T& value) {
  if (current == capacity) {
    reserve(capacity == 0 ? 1 : capacity * 2);
  }
  std::allocator_traits<Allocator>::construct(allocator, arr + current, value);
  ++current;
}
```

1.6.2 pop_back

`pop_back` funkcija pašalina paskutinį elementą iš vektoriaus. Jei vektorius yra tuščias, išmeta `std::out_of_range` išimtį.

```
void pop_back() {
  if (current > 0) {
    --current;
    std::allocator_traits<Allocator>::destroy(allocator, arr + current);
  } else {
    throw std::out_of_range("Cannot pop_back from an empty MyVector");
  }
}
```

1.6.3 clear

`clear` funkcija pašalina visus elementus iš vektoriaus, palikdama jį tuščią.

```
void clear() noexcept {
        destroy_elements();
    }
```

1.6.4 empty

`empty` funkcija patikrina ar vektorius yra tuščias. Grąžina `true`, jei vektorius yra tuščias (`current == 0`), ir `false` priešingu atveju.

```
bool empty() const noexcept {
        return current == 0;
    }
```

1.6.5 swap

`swap` funkcija sukeičia du vektorius. Apkeičia elementus, talpą, dabartinį dydį ir skiriamąją atmintį su kitu vektoriumi.

```
void swap(MyVector& other) noexcept {
        std::swap(arr, other.arr);
        std::swap(current, other.current);
        std::swap(capacity, other.capacity);
        std::swap(allocator, other.allocator);
    }
```

1.7 Klasės naudojami "Rule of five" ir I/O operatoriai.

"Rule of five" yra C++ programavimo kalbos konceptas, kuris apima penkis pagrindinius komponentus, reikalingus objektų valdymui: destruktorius, kopijavimo konstruktorius, kopijavimo priskyrimo operatorius, perkeliamasis konstruktorius ir perkeliamasis priskyrimo operatorius. Šiame projekte "Rule of five" yra taikomas `Student` klasei.

1. **Destruktorius**: Šis komponentas naudojamas išvalyti `Student` objektą, kai jis nebereikalingas.
2. **Kopijavimo konstruktorius**: Šis komponentas leidžia sukurti naują `Student` objektą, kuris yra identiškas esamam `Student` objektui.
3. **Kopijavimo priskyrimo operatorius**: Šis operatorius leidžia priskirti vieno `Student` objekto vertę kitam `Student` objektui.
4. **Perkeliamasis konstruktorius**: Šis komponentas leidžia "perkelti" `Student` objektą, o ne kopijuoti jį. Tai yra efektyvesnis būdas sukurti naują `Student` objektą, kai turime laikiną `Student` objektą, kurio mums nebereikia.
5. **Perkeliamasis priskyrimo operatorius**: Šis operatorius leidžia "perkelti" vieno `Student` objekto vertę į kitą `Student` objektą, o ne kopijuoti ją.

Be to, šiame projekte yra naudojami įvesties (>>) ir išvesties (<<) operatoriai.

Įvesties operatorius (>>): Šis operatorius naudojamas nuskaitant duomenis iš įvesties srauto (pvz., `std::cin` ar `std::istringstream`) į `Student` objektą.**

****Išvesties operatorius (<<)**: Šis operatorius naudojamas rašant `Student` objektą į išvesties srautą (pvz., `std::cout` ar `std::ostringstream`).

Šie operatoriai leidžia lengvai ir efektyviai manipuliuoti `Student` objektais, nuskaitant duomenis iš įvesties srautų ir rašant juos į išvesties srautus.

Visiems konstruktoriams / operatoriams yra atlikti testai, siekiantys patikrinti ar visi jie veikia. Testus galima atlikti pasirinkus tai per meniu.

1.8 Programos sparta naudojant skirtingus kompiliatoriaus optimizavimo lygius

1.8.1 Testavimo sistemos parametrai:

- Saugykla: 256 GB, Integruota NVMe SSD
- Atmintis: 8 GB RAM
- Procesorius: Apple M1

1.8.2 Be optimizavimo testavimas

	Greitis(1mln.)	**Greitis(10mln.)**	**Failo dydis**
Struct	12.111s	130.058s	411kb
Klasė	11.358s	118.430s	404kb

Peržiūrėti
Struktūros testavimo rezultatai
Struktūros failo dydis
Klasės testavimo rezultatai
Klasės failo dydis

1.8.3 O1 optimizavimo lygio testavimas

	Greitis(1mln.)	**Greitis(10mln.)**	**Failo dydis**
Struct	2.790s	30.391s	162kb
Klasė	11.489s	117.949s	147kb

- Su struktūra greitesni testavimo rezultatai, naudojant šį optimizavimo raktą.
- Struktūros ir klasių failų dydžiai mažesni, naudojant šį optimizavimo raktą.

Peržiūrėti
Struktūros testavimo rezultatai
Struktūros failo dydis
Klasės testavimo rezultatai
Klasės failo dydis

1.8.4 O2 optimizavimo lygio testavimas

	Greitis(1mln.)	Greitis(10mln.)	Failo dydis
Struct	2.662s	30.459s	162kb
Klasė	2.767s	29.540s	147kb

- Su klase greitesni testavimo rezultatai, naudojant šį optimizavimo raktą.
- Struktūros ir klasių failų dydžiai išliko tokie patys, kaip su praeitu optimizavimo raktu.

Peržiūrėti
Struktūros testavimo rezultatai
Struktūros failo dydis
Klasės testavimo rezultatai
Klasės failo dydis

1.8.5 O3 optimizavimo lygio testavimas

	Greitis(1mln.)	Greitis(10mln.)	Failo dydis
Struct	2.734s	28.984s	161kb
Klasė	2.738s	29.735s	162kb

- Testavimo greičio rezultatai panašūs su praeitu optimizavimo raktu.
- Failų dydžiai minimaliai pasikeitė.

Peržiūrėti
Struktūros testavimo rezultatai
Struktūros failo dydis
Klasės testavimo rezultatai
Klasės failo dydis

1.9 Konteinerių testavimas

Peržiūrėti

1.9.1 Testavimo sistemos parametrai:

- Saugykla: 256 GB, Integruota NVMe SSD
- Atmintis: 8 GB RAM
- Procesorius: Apple M1

1.10 1 strategija

1.10.1 Naudojant Vector tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.023s	0.004s	0.005s	0.033s
10 000	0.113s	0.018s	0.023s	0.155s
100 000	0.763s	0.188s	0.237s	1.189s
1 000 000	7.448s	2.105s	2.673s	12.227s
10 000 000	74.810s	24.911s	31.783s	131.505s

peržiūrėti

1.10.2 Naudojant Deque tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.023s	0.004s	0.005s	0.034s
10 000	0.093s	0.019s	0.023s	0.136s
100 000	0.766s	0.193s	0.239s	1.198s
1 000 000	7.312s	2.160s	2.638s	12.111s
10 000 000	73.857s	25.580s	30.620s	130.058s

peržiūrėti

1.10.3 Naudojant List tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.023s	0.003s	0.004s	0.031s
10 000	0.112s	0.018s	0.023s	0.154s
100 000	0.761s	0.247s	0.296s	1.305s
1 000 000	7.352s	3.322s	3.818s	14.493s
10 000 000	73.862s	42.406s	47.849s	164.118s

peržiūrėti

1.11 2 strategija

1.11.1 Naudojant Vector tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.024s	0.004s	0.035s	0.064s
10 000	0.112s	0.018s	1.879s	2.010s
100 000	0.762s	0.184s	184.637s	184.637s

peržiūrėti

- Rezultatų su 1 000 000 ir 10 000 000 nėra, nes per ilgai trunka skaičiavimai (>10min).

1.11.2 Naudojant Deque tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.021s	0.004s	0.006s	0.032s
10 000	0.114s	0.019s	0.025s	0.158s
100 000	0.765s	0.192s	0.252s	1.210s
1 000 000	7.337s	2.128s	2.772s	12.238s

peržiūrėti

- Rezultatų su 10 000 000 nėra, nes per ilgai trunka skaičiavimai (>10min).

1.11.3 Naudojant List tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.023s	0.003s	0.001s	0.028s
10 000	0.113s	0.018s	0.005s	0.137s
100 000	0.761s	0.239s	0.061s	1.062s
1 000 000	7.433s	3.324s	0.668s	11.344s
10 000 000	72.891s	42.406s	8.586s	123.646s

peržiūrėti

1.12 3 strategija

1.12.1 Naudojant Vector tipo konteinerius:

Failo dydis	Skaitymo laikas	Rūšiavimo laikas	Skirstymo laikas	Veikimo laikas
1 000	0.023s	0.004s	0.001s	0.028s
10 000	0.112s	0.019s	0.005s	0.137s
100 000	0.758s	0.191s	0.051s	1.001s
1 000 000	7.303s	2.263s	0.622s	10.189s
10 000 000	73.373s	24.204s	8.597s	106.176s

peržiūrėti

1.13 Išvados

Remiantis atliktų testų rezultatais, galime padaryti keletą išvadų:

1. **Vector tipo konteineriai**: Vector tipo konteineriai parodė geriausius rezultatus su mažesniais failais. Tačiau, kai studentų kiekis padidėjo iki 1 000 000 ir 10 000 000, Vector tipo konteinerių veikimo laikas žymiai padidėjo, naudojant antrąją strategiją, kuri yra neefektyvi su šiuo konteineriu tipu. Pirmoji ir trečioji strategija parodė greičiausius rezultatus.

2. **Deque tipo konteineriai**: Deque tipo konteineriai parodė panašius rezultatus kaip ir Vector tipo konteineriai. Tačiau, jie buvo šiek tiek greitesni su didesniais failų dydžiais. Kaip ir su vektoriais, antroji strategija buvo neefektyvi.

3. **List tipo konteineriai**: List tipo konteineriai parodė geriausius rezultatus su didesniais failų dydžiais. Jie buvo ypač efektyvūs naudojant antrąją strategiją.

Bendra išvada yra, kad konteinerio tipo ir strategijos pasirinkimas gali turėti didelę įtaką programos veikimo laikui, ypač dirbant su didelėmis duomenų apimtimis.

1.14 Senesnių versijų aprašymai

- v.pradinė: Pradinė versija. Nuskaito vartotojo įvestį, patikrina ją, leidžia vartotojui pasirinkti tarp dviejų galutinio balo skaičiavimo būdų (vidurkis ar mediana), ir išveda duomenis ekrane.
- v0.1: Prideda galimybę pasirinkti iš 4 skirtingų būdų įvesti arba generuoti duomenis / baigti programą. Dinamiškai paskiria atmintį pagal įvestų duomenų kiekį. Išveda duomenis ekrane.
- v0.2: Prideda galimybę skaityti duomenis iš failo. Leidžia vartotojui pasirinkti iš 5 skirtingų būdų įvesti, skaityti arba generuoti duomenis. Dinamiškai skiria atmintį pagal įvestą / nuskaitytą duomenų kiekį. Atidaro testavimo failus ir apskaičiuoja laiką, kurį praleidžia apdorojant duomenis iš failų. Išveda duomenis pasirinktinai ekrane arba faile.
- v0.3: Prideda išimčių valdymą duomenų nuskaitymui. Visi pranešimai išvedami lietuvių kalba. Projektas išskaidytas į kelis failus (.h ir .cpp).
- v0.4: Prideda galimybę generuoti penkis atsitiktinius studentų sąrašų failus, sudarytus iš: 1 000, 10 000, 100 000, 1 000 000, 10 000 000 įrašų. Atlieka tyrimus / testavimus su sugeneruotais failais. Surūšiuoja studentus ir išveda į du naujus failus.
- v1.0_pradinė: Prideda 3 skirtingus konteinerio tipo pasirinkimus testavimui - vector, deque, list.
- v1.0: Yra 3 skirtingos konteinerių testavimo strategijos ir galimybė sukompiliuoti programą, naudojant Makefile.
- v1.1: Naudojamos klasės su destruktoriais ir konstruktoriais, vietoj struktūrų.
- v1.2: Prideda galimybė testuoti visus "Rule of five" konstruktorius ir I/O operatorius.
- v1.5: Abstrakti klasė Person. Student klasė yra Person klasės išvestinė klasė.
- v2.0: Google test unit testai, DoxyGen TeX/html dokumentacija.

Chapter 2

Vardų Srities Indeksas

2.1 Varų Srities Sąrašas

Sąrašas visų dokumentuotų vardų sričių su trumpais aprašymais:

Chapter 3

Hierarchijos Indeksas

3.1 Klasių hierarchija

Šis paveldėjimo sąrašas yra beveik surikiuotas abėcėlės tvarka:

Chapter 4

Klasės Indeksas

4.1 Klasės

Klasės, struktūros, sąjungos ir sąsajos su trumpais aprašymais:

Chapter 5

Failo Indeksas

5.1 Failai

Visų dokumentuotų failų sąrašas su trumpais aprašymais:

Chapter 6

Vardų Srities Dokumentacija

6.1 std Vardų Srities Nuoroda

Apkeitimo funkcija vektoriams.

Funkcijos

- template<typename T , typename Allocator >
  void **swap** (MyVector< T, Allocator > &lhs, MyVector< T, Allocator > &rhs) noexcept

6.1.1 Smulkus aprašymas

Apkeitimo funkcija vektoriams.

Template Parameters

T	Elementų tipas.
Allocator	Alokatoriaus tipas.

Parametrai

lhs	Kairysis vektorius.
rhs	Dešinysis vektorius.

Chapter 7

Klasės Dokumentacija

7.1 MyVector< T, Allocator > Klasė Šablonas

MyVector konteineris, kurios funkcionalumas yra panašus į std::vector.
`#include <myVector.h>`

Vieši Tipai

- using **value_type** = T
  Elementų tipas.
- using **allocator_type** = Allocator
  Skiriamosios atminties valdiklio tipas.
- using **size_type** = std::size_t
  Dydis, naudojamas vektoriuje.
- using **difference_type** = std::ptrdiff_t
  Skirtumas tarp dviejų iteratorių.
- using **reference** = value_type &
  Nuoroda į elementą.
- using **const_reference** = const value_type &
  Nuoroda į konstantų elementą.
- using **pointer** = typename std::allocator_traits<Allocator>::pointer
  Nuoroda į elementą atmintyje.
- using **const_pointer** = typename std::allocator_traits<Allocator>::const_pointer
  Nuoroda į konstantų elementą atmintyje.
- using **iterator** = pointer
  Iteratorius per elementus.
- using **const_iterator** = const_pointer
  Iteratorius per konstantus elementus.
- using **reverse_iterator** = std::reverse_iterator<iterator>

Atvirkštinis iteratorius per elementus.

- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>

  Atvirkštinis iteratorius per konstantus elementus.

Vieši Metodai

- **MyVector** ()

  Konstruktorius be parametrų, sukuriantis tuščią vektorių.
- ∼**MyVector** ()

  Destruktorius, atlaisvinantis vektoriaus resursus.
- MyVector (const MyVector &other)

  Kopijavimo konstruktorius.
- MyVector & operator= (const MyVector &other)

  Priskyrimo operatorius kopijavimui.
- MyVector (MyVector &&other) noexcept

  Perkėlimo konstruktorius.
- MyVector & operator= (MyVector &&other) noexcept

  Priskyrimo operatorius perkėlimui.
- reference at (size_type pos)

  Grąžina elementą nurodytoje pozicijoje su ribų tikrinimu.
- const_reference at (size_type pos) const

  Grąžina elementą nurodytoje pozicijoje su ribų tikrinimu (konstanta versija).
- reference operator[] (size_type pos)

  Grąžina elementą nurodytoje pozicijoje.
- const_reference operator[] (size_type pos) const

  Grąžina elementą nurodytoje pozicijoje (konstanta versija).
- allocator_type get_allocator () const noexcept

  Grąžina vektoriaus paskirstytoją.
- reference front ()

  Grąžina pirmąjį vektoriaus elementą.
- const_reference front () const

  Grąžina pirmąjį vektoriaus elementą (konstanta versija).
- reference back ()

  Grąžina paskutinį vektoriaus elementą.
- const_reference back () const

  Grąžina paskutinį vektoriaus elementą (konstanta versija).
- T ∗ data () noexcept

  Grąžina žaliąjį vektoriaus masyvą.
- const T ∗ data () const noexcept

  Grąžina žaliąjį vektoriaus masyvą (konstanta versija).

- iterator begin () noexcept

  Grąžina iteratorių į pirmąjį vektoriaus elementą.
- const_iterator begin () const noexcept

  Grąžina iteratorių į pirmąjį vektoriaus elementą (konstanta versija).
- const_iterator cbegin () const noexcept

  Grąžina konstanta iteratorių į pirmąjį vektoriaus elementą.
- iterator end () noexcept

  Grąžina iteratorių į paskutinį vektoriaus elementą.
- const_iterator end () const noexcept

  Grąžina iteratorių į paskutinį vektoriaus elementą (konstanta versija).
- const_iterator cend () const noexcept

  Grąžina konstanta iteratorių į paskutinį vektoriaus elementą.
- reverse_iterator rbegin () noexcept

  Grąžina atvirkštinį iteratorių į pirmąjį vektoriaus elementą.
- const_reverse_iterator rbegin () const noexcept

  Grąžina atvirkštinį iteratorių į pirmąjį vektoriaus elementą (konstanta versija).
- const_reverse_iterator crbegin () const noexcept

  Grąžina konstanta atvirkštinį iteratorių į pirmąjį vektoriaus elementą.
- reverse_iterator rend () noexcept

  Grąžina atvirkštinį iteratorių į paskutinį vektoriaus elementą.
- const_reverse_iterator rend () const noexcept

  Grąžina atvirkštinį iteratorių į paskutinį vektoriaus elementą (konstanta versija).
- const_reverse_iterator crend () const noexcept

  Grąžina konstanta atvirkštinį iteratorių į paskutinį vektoriaus elementą.
- bool empty () const noexcept

  Patikrina, ar vektorius yra tuščias.
- size_type size () const noexcept

  Grąžina elementų skaičių vektoriuje.
- size_type max_size () const noexcept

  Grąžina maksimalų galimą vektoriaus dydį.
- void reserve (size_type new_cap)

  Rezervuoja nurodytą vietos kiekį vektoriui.
- size_type getCapacity () const noexcept

  Grąžina vektoriaus talpą.
- void **shrink_to_fit** ()

  Sumažina vektoriaus talpą iki dabartinio elemento skaičiaus.
- void **clear** () noexcept

  Išvalo vektoriaus turinį.

- void push_back (const T &value)

  Prideda elementą į vektoriaus pabaigą.
- void pop_back ()

  Pašalina paskutinį vektoriaus elementą.
- void resize (size_type count, T value=T())

  Pakeičia vektoriaus dydį.
- void swap (MyVector &other) noexcept

  Pakeičia šio vektoriaus turinį su kitu vektoriumi.
- iterator insert (const_iterator pos, const T &value)

  Įterpia elementą nurodytoje pozicijoje.
- template<typename... Args>
  iterator emplace (const_iterator pos, Args &&...args)

  Įterpia elementą nurodytoje pozicijoje su argumentais.
- iterator erase (const_iterator pos)

  Pašalina elementą nurodytoje pozicijoje.
- template<typename InputIt >
  void assign (InputIt first, InputIt last)

  Pakeičia vektoriaus turinį intervalu.
- void assign (size_type count, const T &value)

  Pakeičia vektoriaus turinį nurodytu elementų skaičiumi.
- template<typename... Args>
  void emplace_back (Args &&...args)

  Prideda elementą į vektoriaus pabaigą su argumentais.
- template<typename InputIt >
  void append_range (InputIt first, InputIt last)

  Prideda elementų intervalą į vektoriaus pabaigą.

7.1.1 Smulkus aprašymas

template<typename T, typename Allocator = std::allocator<T>>
class MyVector< T, Allocator >

MyVector konteineris, kurios funkcionalumas yra panašus į std::vector.

Template Parameters

T	Elementų tipas
Allocator	Skiriamosios atminties valdiklio tipas

7.1.2 Konstruktoriaus ir Destruktoriaus Dokumentacija

MyVector() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
MyVector< T, Allocator >::MyVector (
            const MyVector< T, Allocator > & other )  [inline]
```

Kopijavimo konstruktorius.

Parametrai

other	Kitas vektorius, iš kurio bus kopijuojami duomenys.

MyVector() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
MyVector< T, Allocator >::MyVector (
            MyVector< T, Allocator > && other )  [inline], [noexcept]
```

Perkėlimo konstruktorius.

Parametrai

other	Kitas vektorius, kurio duomenys bus perkelti.

7.1.3 Metodų Dokumentacija

append_range()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename InputIt >
void MyVector< T, Allocator >::append_range (
            InputIt first,
            InputIt last )  [inline]
```

Prideda elementų intervalą į vektoriaus pabaigą.

Parametrai

first	Pradžios iteratorius.
last	Pabaigos iteratorius.

assign() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename InputIt >
```

```
void MyVector< T, Allocator >::assign (
              InputIt first,
              InputIt last )  [inline]
```

Pakeičia vektoriaus turinį intervalu.

Parametrai

first	Pradžios iteratorius.
last	Pabaigos iteratorius.

assign() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::assign (
              size_type count,
              const T & value )  [inline]
```

Pakeičia vektoriaus turinį nurodytu elementų skaičiumi.

Parametrai

count	Elementų skaičius.
value	Elementų reikšmė.

at() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference MyVector< T, Allocator >::at (
              size_type pos )  [inline]
```

Grąžina elementą nurodytoje pozicijoje su ribų tikrinimu.

Parametrai

pos	Elemento pozicija.

Gražina

Nuoroda į elementą.

Išimtys

std::out_of_range	Jei pozicija yra už vektoriaus ribų.

at() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference MyVector< T, Allocator >::at (
            size_type pos ) const  [inline]
```

Grąžina elementą nurodytoje pozicijoje su ribų tikrinimu (konstanta versija).

Parametrai

pos	Elemento pozicija.

Gražina

Nuoroda į elementą.

Išimtys

std::out_of_range	Jei pozicija yra už vektoriaus ribų.

back() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference MyVector< T, Allocator >::back ( )  [inline]
```

Grąžina paskutinį vektoriaus elementą.

Gražina

Nuoroda į paskutinį elementą.

back() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference MyVector< T, Allocator >::back ( ) const  [inline]
```

Grąžina paskutinį vektoriaus elementą (konstanta versija).

Gražina

Nuoroda į paskutinį elementą.

begin() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator MyVector< T, Allocator >::begin ( ) const  [inline], [noexcept]
```

Grąžina iteratorių į pirmąjį vektoriaus elementą (konstanta versija).

Gražina

Iteratorius į pirmąjį elementą.

begin() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
iterator MyVector< T, Allocator >::begin ( )  [inline], [noexcept]
```

Grąžina iteratorių į pirmąjį vektoriaus elementą.

Gražina

Iteratorius į pirmąjį elementą.

cbegin()

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator MyVector< T, Allocator >::cbegin ( ) const  [inline], [noexcept]
```

Grąžina konstanta iteratorių į pirmąjį vektoriaus elementą.

Gražina

Konstanta iteratorius į pirmąjį elementą.

cend()

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator MyVector< T, Allocator >::cend ( ) const  [inline], [noexcept]
```

Grąžina konstanta iteratorių į paskutinį vektoriaus elementą.

Gražina

Konstanta iteratorius į paskutinį elementą.

crbegin()

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator MyVector< T, Allocator >::crbegin ( ) const  [inline], [noexcept]
```

Grąžina konstanta atvirkštinį iteratorių į pirmąjį vektoriaus elementą.

Gražina

Konstanta atvirkštinis iteratorius į pirmąjį elementą.

crend()

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator MyVector< T, Allocator >::crend ( ) const  [inline], [noexcept]
```

Grąžina konstanta atvirkštinį iteratorių į paskutinį vektoriaus elementą.

Gražina

Konstanta atvirkštinis iteratorius į paskutinį elementą.

data() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const T * MyVector< T, Allocator >::data ( ) const  [inline], [noexcept]
```

Grąžina žaliąjį vektoriaus masyvą (konstanta versija).

Gražina

Rodyklė į masyvą.

data() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
T * MyVector< T, Allocator >::data ( )  [inline], [noexcept]
```

Grąžina žaliąjį vektoriaus masyvą.

Gražina

Rodyklė į masyvą.

emplace()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename...  Args>
iterator MyVector< T, Allocator >::emplace (
              const_iterator pos,
              Args &&...  args )  [inline]
```

Įterpia elementą nurodytoje pozicijoje su argumentais.

Parametrai

pos	Pozicija, kurioje bus įterptas elementas.
args	Argumentai, naudojami elemento konstravimui.

Gražina

Iteratorius į įterptą elementą.

emplace_back()

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename...  Args>
void MyVector< T, Allocator >::emplace_back (
              Args &&...  args )  [inline]
```

Prideda elementą į vektoriaus pabaigą su argumentais.

Parametrai

args	Argumentai, naudojami elemento konstravimui.

empty()

```
template<typename T , typename Allocator = std::allocator<T>>
bool MyVector< T, Allocator >::empty ( ) const  [inline], [noexcept]
```

Patikrina, ar vektorius yra tuščias.

Gražina

True, jei vektorius yra tuščias, priešingu atveju False.

end() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_iterator MyVector< T, Allocator >::end ( ) const  [inline], [noexcept]
```

Grąžina iteratorių į paskutinį vektoriaus elementą (konstanta versija).

Gražina

Iteratorius į paskutinį elementą.

end() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
iterator MyVector< T, Allocator >::end ( )  [inline], [noexcept]
```

Grąžina iteratorių į paskutinį vektoriaus elementą.

Gražina

Iteratorius į paskutinį elementą.

erase()

```
template<typename T , typename Allocator = std::allocator<T>>
iterator MyVector< T, Allocator >::erase (
              const_iterator pos )  [inline]
```

Pašalina elementą nurodytoje pozicijoje.

Parametrai

pos	Pozicija, iš kurios bus pašalintas elementas.

Gražina

Iteratorius į poziciją po pašalinto elemento.

front() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference MyVector< T, Allocator >::front ( )  [inline]
```

Grąžina pirmąjį vektoriaus elementą.

Gražina

Nuoroda į pirmąjį elementą.

front() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference MyVector< T, Allocator >::front ( ) const  [inline]
```

Grąžina pirmąjį vektoriaus elementą (konstanta versija).

Gražina

Nuoroda į pirmąjį elementą.

get_allocator()

```
template<typename T , typename Allocator = std::allocator<T>>
allocator_type MyVector< T, Allocator >::get_allocator ( ) const  [inline], [noexcept]
```

Grąžina vektoriaus paskirstytoją.

Gražina

Paskirstytojo objektas.

getCapacity()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type MyVector< T, Allocator >::getCapacity ( ) const  [inline], [noexcept]
```

Grąžina vektoriaus talpą.

Gražina

Talpa.

insert()

```
template<typename T , typename Allocator = std::allocator<T>>
iterator MyVector< T, Allocator >::insert (
            const_iterator pos,
            const T & value )  [inline]
```

Įterpia elementą nurodytoje pozicijoje.

Parametrai

pos	Pozicija, kurioje bus įterptas elementas.
value	Įterpiamas elementas.

Gražina

Iteratorius į įterptą elementą.

max_size()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type MyVector< T, Allocator >::max_size ( ) const  [inline], [noexcept]
```

Grąžina maksimalų galimą vektoriaus dydį.

Gražina

Maksimalus elementų skaičius.

operator=() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
MyVector & MyVector< T, Allocator >::operator= (
            const MyVector< T, Allocator > & other )  [inline]
```

Priskyrimo operatorius kopijavimui.

Parametrai

other	Kitas vektorius, iš kurio bus kopijuojami duomenys.

Gražina

Grąžinamas pats objektas po priskyrimo.

operator=() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
MyVector & MyVector< T, Allocator >::operator= (
            MyVector< T, Allocator > && other )  [inline], [noexcept]
```

Priskyrimo operatorius perkėlimui.

Parametrai

other	Kitas vektorius, kurio duomenys bus perkelti.

Gražina

Grąžinamas pats objektas po priskyrimo.

operator[]() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reference MyVector< T, Allocator >::operator[] (
            size_type pos )  [inline]
```

Grąžina elementą nurodytoje pozicijoje.

Parametrai

pos	Elemento pozicija.

Gražina

Nuoroda į elementą.

operator[]() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reference MyVector< T, Allocator >::operator[] (
            size_type pos ) const  [inline]
```

Grąžina elementą nurodytoje pozicijoje (konstanta versija).

Parametrai

pos	Elemento pozicija.

Gražina

Nuoroda į elementą.

pop_back()

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::pop_back ( )  [inline]
```

Pašalina paskutinį vektoriaus elementą.

Išimtys

std::out_of_range	Jei vektorius yra tuščias.

push_back()

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::push_back (
            const T & value )  [inline]
```

Prideda elementą į vektoriaus pabaigą.

Parametrai

value	Elemento reikšmė.

rbegin() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator MyVector< T, Allocator >::rbegin ( ) const  [inline], [noexcept]
```

Grąžina atvirkštinį iteratorių į pirmąjį vektoriaus elementą (konstanta versija).

Gražina

Atvirkštinis iteratorius į pirmąjį elementą.

rbegin() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reverse_iterator MyVector< T, Allocator >::rbegin ( )  [inline], [noexcept]
```

Grąžina atvirkštinį iteratorių į pirmąjį vektoriaus elementą.

Gražina

Atvirkštinis iteratorius į pirmąjį elementą.

rend() [1/2]

```
template<typename T , typename Allocator = std::allocator<T>>
const_reverse_iterator MyVector< T, Allocator >::rend ( ) const  [inline], [noexcept]
```

Grąžina atvirkštinį iteratorių į paskutinį vektoriaus elementą (konstanta versija).

Gražina

Atvirkštinis iteratorius į paskutinį elementą.

rend() [2/2]

```
template<typename T , typename Allocator = std::allocator<T>>
reverse_iterator MyVector< T, Allocator >::rend ( )  [inline], [noexcept]
```

Grąžina atvirkštinį iteratorių į paskutinį vektoriaus elementą.

Gražina

Atvirkštinis iteratorius į paskutinį elementą.

reserve()

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::reserve (
              size_type new_cap )  [inline]
```

Rezervuoja nurodytą vietos kiekį vektoriui.

Parametrai

new_cap	Nauja talpa.

resize()

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::resize (
              size_type count,
              T value = T() )  [inline]
```

Pakeičia vektoriaus dydį.

Parametrai

count	Naujas dydis.
value	Nauji elementai bus užpildyti šia reikšme.

size()

```
template<typename T , typename Allocator = std::allocator<T>>
size_type MyVector< T, Allocator >::size ( ) const  [inline], [noexcept]
```

Grąžina elementų skaičių vektoriuje.

Gražina

Elementų skaičius.

swap()

```
template<typename T , typename Allocator = std::allocator<T>>
void MyVector< T, Allocator >::swap (
              MyVector< T, Allocator > & other )  [inline], [noexcept]
```

Pakeičia šio vektoriaus turinį su kitu vektoriumi.

Parametrai

other	Kitas vektorius.

Dokumentacija šiai klasei sugeneruota iš šio failo:

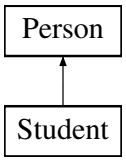
- myVector.h

7.2 Person Klasė

Klasė, aprašanti žmogų.

`#include <person.h>`

Paveldimumo diagrama Person:



Vieši Metodai

- virtual ~**Person** ()=default

  Virtualus destruktorius.
- virtual std::string getFirstName () const =0

  Virtualus metodas, grąžinantis asmens vardą.
- virtual std::string getLastName () const =0

  Virtualus metodas, grąžinantis asmens pavardę.
- virtual std::string getName () const =0

  Virtualus metodas, grąžinantis pilną asmens vardą ir pavardę.

7.2.1 Smulkus aprašymas

Klasė, aprašanti žmogų.

Ši klasė yra abstrakti bazinė klasė, nuo kurios paveldės Student klasė.

7.2.2 Metodų Dokumentacija

getFirstName()

```
virtual std::string Person::getFirstName ( ) const  [pure virtual]
```

Virtualus metodas, grąžinantis asmens vardą.

Gražina

Asmens vardas.

Realizuota Student.

getLastName()

```
virtual std::string Person::getLastName ( ) const  [pure virtual]
```

Virtualus metodas, grąžinantis asmens pavardę.

Gražina

Asmens pavardė.

Realizuota Student.

getName()

```
virtual std::string Person::getName ( ) const  [pure virtual]
```

Virtualus metodas, grąžinantis pilną asmens vardą ir pavardę.

Gražina

Pilnas asmens vardas ir pavardė.

Realizuota Student.

Dokumentacija šiai klasei sugeneruota iš šio failo:

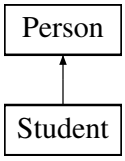
- person.h

7.3 Student Klasė

Klasė, aprašanti studentą.

`#include <student.h>`

Paveldimumo diagrama Student:



Vieši Metodai

- **Student** ()

  Konstruktorius be parametrų.
- Student (const std::string &firstName, const std::string &lastName, int examResults, const std::vector< int > &homeworkResults)

  Konstruktorius su parametrais.
- Student (const Student &other)

  Kopijavimo konstruktorius.
- Student (Student &&other) noexcept

  Perkėlimo konstruktorius.
- Student & operator= (const Student &other)

  Kopijavimo priskyrimo operatorius.
- Student & operator= (Student &&other) noexcept

  Perkėlimo priskyrimo operatorius.
- ∼**Student** ()

  Destruktorius.
- std::string getFirstName () const

  Grąžina studento vardą.
- std::string getLastName () const

  Grąžina studento pavardę.
- std::string getName () const

  Grąžina studento vardą ir pavardę.
- const std::vector< int > & getHomeworkResults () const

  Grąžina studento namų darbų rezultatus.
- int getExamResults () const

  Grąžina studento egzamino rezultatą.
- int getExamGrade () const

  Grąžina studento egzamino pažymį.

- void **removeLastHomeworkGrade** ()

  Pašalina paskutinį namų darbo pažymį.
- void setFirstName (std::string firstName)

  Nustato studento vardą.
- void setLastName (std::string lastName)

  Nustato studento pavardę.
- void addHomeworkResult (int result)

  Prideda namų darbo rezultatą.
- void **clearHomeworkResults** ()

  Išvalo namų darbų rezultatus.
- void setExamResults (int examResults)

  Nustato egzamino rezultatą.
- void setHomeworkResults (std::vector< int > results)

  Nustato namų darbų rezultatus.
- double calculateMedian () const

  Skaičiuoja medianą iš namų darbų rezultatų.
- double calculateAverage () const

  Skaičiuoja vidurkį iš namų darbų rezultatų.
- double calculateFinalGrade (bool median) const

  Skaičiuoja galutinį pažymį.

Vieši Metodai inherited from Person

- virtual ∼**Person** ()=default

  Virtualus destruktorius.

Draugai

- std::istream & operator>> (std::istream &is, Student &s)

  Įvesties operatorius.
- std::ostream & operator<< (std::ostream &os, const Student &s)

  Išvesties operatorius.

7.3.1 Smulkus aprašymas

Klasė, aprašanti studentą.

Ši klasė paveldi Person klasę ir prideda papildomus studento atributus.

7.3.2 Konstruktoriaus ir Destruktoriaus Dokumentacija

Student() [1/3]

```
Student::Student (
            const std::string & firstName,
            const std::string & lastName,
            int examResults,
            const std::vector< int > & homeworkResults )
```

Konstruktorius su parametrais.

Parametrai

firstName	Studento vardas.
lastName	Studento pavardė.
examResults	Studento gzamino rezultatas.
homeworkResults	Studento namų darbų rezultatai.

Student() [2/3]

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

Kopijavimo konstruktorius.

Parametrai

other	Kita Student klasės objekto instancija.

Student() [3/3]

```
Student::Student (
            Student && other )  [noexcept]
```

Perkėlimo konstruktorius.

Parametrai

other	Kita Student klasės objekto instancija.

7.3.3 Metodų Dokumentacija

addHomeworkResult()

```
void Student::addHomeworkResult (
```

```
            int result )  [inline]
```
Prideda namų darbo rezultatą.

Parametrai

result	Namų darbo rezultatas.

calculateAverage()

```
double Student::calculateAverage ( ) const
```
Skaičiuoja vidurkį iš namų darbų rezultatų.

Gražina

Vidurkis.

calculateFinalGrade()

```
double Student::calculateFinalGrade (
            bool median ) const
```
Skaičiuoja galutinį pažymį.

Parametrai

median	Jei true, naudoja medianą, jei false, naudoja vidurkį.

Gražina

Galutinis pažymys.

calculateMedian()

```
double Student::calculateMedian ( ) const
```
Skaičiuoja medianą iš namų darbų rezultatų.

Gražina

Mediana.

getExamGrade()

```
int Student::getExamGrade ( ) const  [inline]
```
Grąžina studento egzamino pažymį.

Gražina

Egzamino pažymys.

getExamResults()

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

Grąžina studento egzamino rezultatą.

Gražina

Egzamino rezultatas.

getFirstName()

```
std::string Student::getFirstName ( ) const  [inline], [virtual]
```

Grąžina studento vardą.

Gražina

Studento vardas.

Realizuoja Person.

getHomeworkResults()

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

Grąžina studento namų darbų rezultatus.

Gražina

Namų darbų rezultatai.

getLastName()

```
std::string Student::getLastName ( ) const  [inline], [virtual]
```

Grąžina studento pavardę.

Gražina

Studento pavardė.

Realizuoja Person.

getName()

```
std::string Student::getName ( ) const  [inline], [virtual]
```

Grąžina studento vardą ir pavardę.

Gražina

Studento vardas ir pavardė.

Realizuoja Person.

operator=() [1/2]

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

Kopijavimo priskyrimo operatorius.

Parametrai

other	Kita Student klasės objekto instancija.

Gražina

Grąžina ∗this.

operator=() [2/2]

```
Student & Student::operator= (
            Student && other )  [noexcept]
```

Perkėlimo priskyrimo operatorius.

Parametrai

other	Kita Student klasės objekto instancija.

Gražina

Grąžina ∗this.

setExamResults()

```
void Student::setExamResults (
            int examResults )  [inline]
```

Nustato egzamino rezultatą.

Parametrai

examResults	Egzamino rezultatas.

setFirstName()

```
void Student::setFirstName (
            std::string firstName )  [inline]
```

Nustato studento vardą.

Parametrai

firstName	Studento vardas.

setHomeworkResults()

```
void Student::setHomeworkResults (
              std::vector< int > results )  [inline]
```

Nustato namų darbų rezultatus.

Parametrai

results	Namų darbų rezultatai.

setLastName()

```
void Student::setLastName (
              std::string lastName )  [inline]
```

Nustato studento pavardę.

Parametrai

lastName	Studento pavardė.

7.3.4 Draugiškų Ir Susijusių Funkcijų Dokumentacija

operator<<

```
std::ostream & operator<< (
              std::ostream & os,
              const Student & s )  [friend]
```

Išvesties operatorius.

Parametrai

os	Išvesties srautas.
s	Student klasės objekto instancija.

Gražina

Grąžina išvesties srautą.

operator>>

```
std::istream & operator>> (
              std::istream & is,
              Student & s )  [friend]
```

Įvesties operatorius.

Parametrai

is	Įvesties srautas.
s	Student klasės objekto instancija.

Gražina

Grąžina įvesties srautą.

Dokumentacija šiai klasei sugeneruota iš šių failų:

- student.h
- student.cpp

Chapter 8

Failo Dokumentacija

8.1 calculations.h

```
00001 #ifndef CALCULATIONS_H
00002 #define CALCULATIONS_H
00003
00004 #include <vector>
00005 #include "student.h"
00006 #include "input.h"
00007
00008 bool compareByFirstName(const Student& a, const Student& b);
00009
00010 bool compareByLastName(const Student& a, const Student& b);
00011
00012 bool compareByGrade(const Student& a, const Student& b);
00013
00014 double calculateAverage(const std::vector<int>& homeworkResults);
00015
00016 double calculateFinalGrade(const Student& data, bool Median);
00017
00018 double calculateMedian(std::vector<int> homeworkResults);
00019
00020 template <typename Container>
00021 void sortStudents(Container& students, int criteria);
00022
00023 template <>
00024 void sortStudents<std::list<Student>>(std::list<Student>& students, int criteria);
00025
00026 #endif // CALCULATIONS_H
```

8.2 functionality.h

```
00001 #ifndef FUNCTIONALITY_H
00002 #define FUNCTIONALITY_H
00003
00004 #include <string>
00005 #include <cstdlib>
00006 #include <vector>
00007 #include "student.h"
00008 #include "calculations.h"
00009
00010 int getContainerTypeFromUser();
00011 int generateGrade();
00012 std::string generateName();
```

```
00013 std::string generateLastName();
00014 std::string isString(const std::string& prompt);
00015 int isGrade(const std::string& prompt);
00016 void generateFile(int n);
00017 void outputToTerminal(const std::vector<Student>& studentsLow, const
      std::vector<Student>& studentsHigh, bool Median);
00018 template <typename Container>
00019 void outputToFile(const Container& students, size_t m, bool Median, const std::string&
      filename);
00020 template <typename T>
00021 size_t getCapacity(const std::vector<T>& container);
00022 template <typename T>
00023 size_t getCapacity(const MyVector<T>& container);
00024 template <typename Container>
00025 void testContainer(unsigned int sz, const std::string& containerName);
00026
00027 #endif // FUNCTIONALITY_H
```

8.3 input.h

```
00001 #ifndef INPUT_H
00002 #define INPUT_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <fstream>
00007 #include "student.h"
00008 #include "calculations.h"
00009 #include "functionality.h"
00010
00011 bool getMedianPreference();
00012
00013 template <typename Container>
00014 void processStudents(Container& students, bool Median,
      std::chrono::high_resolution_clock::time_point startTotal);
00015
00016 int Menu();
00017
00018 std::string getFilenameFromUser();
00019
00020 template <typename Container>
00021 void readData(std::ifstream& fin, Container& students);
00022
00023 template <typename Container>
00024 void openFiles(const std::vector<std::string>& filenames, Container& students, bool
      Median, int strategy);
00025
00026 void input(Student& data, bool& Median);
00027
00028 std::string studentData(const Student& s);
00029
00030 #endif // INPUT_H
```

8.4 myVector.h

```
00001 #include <cstddef>
00002 #include <iterator>
00003 #include <memory>
00004 #include <stdexcept>
00005 #include <algorithm>
00006 #include <utility>
00007
00014 template <typename T, typename Allocator = std::allocator<T>
```

```
00015 class MyVector
00016 {
00017 public:
00018     // Member types
00022     using value_type = T;
00023 
00027     using allocator_type = Allocator;
00028 
00032     using size_type = std::size_t;
00033 
00037     using difference_type = std::ptrdiff_t;
00038 
00042     using reference = value_type &;
00043 
00047     using const_reference = const value_type &;
00048 
00052     using pointer = typename std::allocator_traits<Allocator>::pointer;
00053 
00057     using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;
00058 
00062     using iterator = pointer;
00063 
00067     using const_iterator = const_pointer;
00068 
00072     using reverse_iterator = std::reverse_iterator<iterator>;
00073 
00077     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00078 
00079 private:
00080     allocator_type allocator;
00081     pointer arr;
00082     size_type capacity;
00083     size_type current;
00084 
00091     void destroy_elements()
00092     {
00093         for (size_type i = 0; i < current; ++i)
00094         {
00095             std::allocator_traits<Allocator>::destroy(allocator, arr + i);
00096         }
00097         current = 0;
00098     }
00099 
00100 public:
00104     MyVector() : arr(nullptr), capacity(0), current(0) {}
00105 
00109     ~MyVector()
00110     {
00111         destroy_elements();
00112         if (arr)
00113         {
00114             allocator.deallocate(arr, capacity);
00115         }
00116     }
00117 
00123     MyVector(const MyVector &other) : allocator(other.allocator), arr(nullptr),
      capacity(0), current(0)
00124     {
00125         if (other.current > 0)
00126         {
00127             arr = allocator.allocate(other.capacity);
00128             try
00129             {
00130                 for (current = 0; current < other.current; ++current)
00131                 {
00132                     std::allocator_traits<Allocator>::construct(allocator, arr +
      current, other.arr[current]);
00133                 }
```

```
00134                 capacity = other.capacity;
00135             }
00136             catch (...)
00137             {
00138                 destroy_elements();
00139                 allocator.deallocate(arr, other.capacity);
00140                 throw;
00141             }
00142         }
00143     }
00144 
00151     MyVector &operator=(const MyVector &other)
00152     {
00153         if (this != &other)
00154         {
00155             MyVector temp(other);
00156             swap(temp);
00157         }
00158         return *this;
00159     }
00160 
00166     MyVector(MyVector &&other) noexcept : allocator(std::move(other.allocator)),
      arr(other.arr), capacity(other.capacity), current(other.current)
00167     {
00168         other.arr = nullptr;
00169         other.capacity = 0;
00170         other.current = 0;
00171     }
00172 
00179     MyVector &operator=(MyVector &&other) noexcept
00180     {
00181         if (this != &other)
00182         {
00183             destroy_elements();
00184             if (arr)
00185             {
00186                 allocator.deallocate(arr, capacity);
00187             }
00188 
00189             allocator = std::move(other.allocator);
00190             arr = other.arr;
00191             capacity = other.capacity;
00192             current = other.current;
00193 
00194             other.arr = nullptr;
00195             other.capacity = 0;
00196             other.current = 0;
00197         }
00198         return *this;
00199     }
00200 
00208     reference at(size_type pos)
00209     {
00210         if (pos >= current)
00211         {
00212             throw std::out_of_range("MyVector::at");
00213         }
00214         return arr[pos];
00215     }
00216 
00224     const_reference at(size_type pos) const
00225     {
00226         if (pos >= current)
00227         {
00228             throw std::out_of_range("MyVector::at");
00229         }
00230         return arr[pos];
00231     }
```

```
00232
00239     reference operator[](size_type pos)
00240     {
00241         return arr[pos];
00242     }
00243
00250     const_reference operator[](size_type pos) const
00251     {
00252         return arr[pos];
00253     }
00254
00260     allocator_type get_allocator() const noexcept
00261     {
00262         return allocator_type();
00263     }
00264
00270     reference front()
00271     {
00272         return arr[0];
00273     }
00274
00280     const_reference front() const
00281     {
00282         return arr[0];
00283     }
00284
00290     reference back()
00291     {
00292         return arr[current - 1];
00293     }
00294
00300     const_reference back() const
00301     {
00302         return arr[current - 1];
00303     }
00304
00310     T *data() noexcept
00311     {
00312         return arr;
00313     }
00314
00320     const T *data() const noexcept
00321     {
00322         return arr;
00323     }
00324
00330     iterator begin() noexcept
00331     {
00332         return arr;
00333     }
00334
00340     const_iterator begin() const noexcept
00341     {
00342         return arr;
00343     }
00344
00350     const_iterator cbegin() const noexcept
00351     {
00352         return arr;
00353     }
00354
00360     iterator end() noexcept
00361     {
00362         return arr + current;
00363     }
00364
00370     const_iterator end() const noexcept
00371     {
```

```
00372         return arr + current;
00373     }
00374
00380     const_iterator cend() const noexcept
00381     {
00382         return arr + current;
00383     }
00384
00390     reverse_iterator rbegin() noexcept
00391     {
00392         return reverse_iterator(end());
00393     }
00394
00400     const_reverse_iterator rbegin() const noexcept
00401     {
00402         return const_reverse_iterator(end());
00403     }
00404
00410     const_reverse_iterator crbegin() const noexcept
00411     {
00412         return const_reverse_iterator(end());
00413     }
00414
00420     reverse_iterator rend() noexcept
00421     {
00422         return reverse_iterator(begin());
00423     }
00424
00430     const_reverse_iterator rend() const noexcept
00431     {
00432         return const_reverse_iterator(begin());
00433     }
00434
00440     const_reverse_iterator crend() const noexcept
00441     {
00442         return const_reverse_iterator(begin());
00443     }
00444
00450     bool empty() const noexcept
00451     {
00452         return current == 0;
00453     }
00454
00460     size_type size() const noexcept
00461     {
00462         return current;
00463     }
00464
00470     size_type max_size() const noexcept
00471     {
00472         return std::numeric_limits<size_t>::max() / sizeof(T);
00473     }
00474
00480     void reserve(size_type new_cap)
00481     {
00482         if (new_cap > capacity)
00483         {
00484             pointer new_arr = allocator.allocate(new_cap);
00485             try
00486             {
00487                 for (size_type i = 0; i < current; ++i)
00488                 {
00489                     std::allocator_traits<Allocator>::construct(allocator, new_arr +
      i, std::move_if_noexcept(arr[i]));
00490                 }
00491             }
00492             catch (...)
00493             {
```

```
00494                 for (size_type i = 0; i < current; ++i)
00495                 {
00496                     std::allocator_traits<Allocator>::destroy(allocator, new_arr + i);
00497                 }
00498                 allocator.deallocate(new_arr, new_cap);
00499                 throw;
00500             }
00501             destroy_elements();
00502             allocator.deallocate(arr, capacity);
00503             arr = new_arr;
00504             capacity = new_cap;
00505         }
00506     }
00507 
00513     size_type getCapacity() const noexcept
00514     {
00515         return capacity;
00516     }
00517 
00521     void shrink_to_fit()
00522     {
00523         if (capacity > current)
00524         {
00525             pointer new_arr = allocator.allocate(current);
00526             try
00527             {
00528                 for (size_type i = 0; i < current; ++i)
00529                 {
00530                     std::allocator_traits<Allocator>::construct(allocator, new_arr +
      i, std::move_if_noexcept(arr[i]));
00531                 }
00532             }
00533             catch (...)
00534             {
00535                 for (size_type i = 0; i < current; ++i)
00536                 {
00537                     std::allocator_traits<Allocator>::destroy(allocator, new_arr + i);
00538                 }
00539                 allocator.deallocate(new_arr, current);
00540                 throw;
00541             }
00542             destroy_elements();
00543             allocator.deallocate(arr, capacity);
00544             arr = new_arr;
00545             capacity = current;
00546         }
00547     }
00548 
00552     void clear() noexcept
00553     {
00554         destroy_elements();
00555     }
00556 
00562     void push_back(const T &value)
00563     {
00564         if (current == capacity)
00565         {
00566             reserve(capacity == 0 ? 1 : capacity * 2);
00567         }
00568         std::allocator_traits<Allocator>::construct(allocator, arr + current, value);
00569         ++current;
00570     }
00571 
00577     void pop_back()
00578     {
00579         if (current > 0)
00580         {
00581             --current;
```

```
00582             std::allocator_traits<Allocator>::destroy(allocator, arr + current);
00583         }
00584         else
00585         {
00586             throw std::out_of_range("Cannot pop_back from an empty MyVector");
00587         }
00588     }
00589
00596     void resize(size_type count, T value = T())
00597     {
00598         if (count > current)
00599         {
00600             if (count > capacity)
00601             {
00602                 reserve(count);
00603             }
00604             for (size_type i = current; i < count; ++i)
00605             {
00606                 std::allocator_traits<Allocator>::construct(allocator, arr + i,
      value);
00607             }
00608         }
00609         else
00610         {
00611             for (size_type i = count; i < current; ++i)
00612             {
00613                 std::allocator_traits<Allocator>::destroy(allocator, arr + i);
00614             }
00615         }
00616         current = count;
00617     }
00618
00624     void swap(MyVector &other) noexcept
00625     {
00626         std::swap(arr, other.arr);
00627         std::swap(capacity, other.capacity);
00628         std::swap(current, other.current);
00629         std::swap(allocator, other.allocator);
00630     }
00631
00639     iterator insert(const_iterator pos, const T &value)
00640     {
00641         size_type index = std::distance(cbegin(), pos);
00642         if (current == capacity)
00643         {
00644             reserve(capacity == 0 ? 1 : capacity * 2);
00645         }
00646         if (index < current)
00647         {
00648             std::move_backward(arr + index, arr + current, arr + current + 1);
00649         }
00650         std::allocator_traits<Allocator>::construct(allocator, arr + index, value);
00651         ++current;
00652         return arr + index;
00653     }
00654
00662     template <typename... Args>
00663     iterator emplace(const_iterator pos, Args &&...args)
00664     {
00665         size_type index = std::distance(cbegin(), pos);
00666         if (current == capacity)
00667         {
00668             reserve(capacity == 0 ? 1 : capacity * 2);
00669         }
00670         std::move_backward(arr + index, arr + current, arr + current + 1);
00671         arr[index] = T(std::forward<Args>(args)...);
00672         ++current;
00673         return arr + index;
```

```
00674     }
00675
00682     iterator erase(const_iterator pos)
00683     {
00684         size_type index = std::distance(cbegin(), pos);
00685         std::allocator_traits<Allocator>::destroy(allocator, arr + index);
00686         std::move(arr + index + 1, arr + current, arr + index);
00687         --current;
00688         return arr + index;
00689     }
00690
00697     template <typename InputIt>
00698     void assign(InputIt first, InputIt last)
00699     {
00700         size_type count = std::distance(first, last);
00701         if (count > capacity)
00702         {
00703             clear();
00704             allocator.deallocate(arr, capacity);
00705             arr = allocator.allocate(count);
00706             capacity = count;
00707         }
00708         for (current = 0; current < count; ++current, ++first)
00709         {
00710             std::allocator_traits<Allocator>::construct(allocator, arr + current,
      *first);
00711         }
00712     }
00713
00720     void assign(size_type count, const T &value)
00721     {
00722         if (count > capacity)
00723         {
00724             clear();
00725             allocator.deallocate(arr, capacity);
00726             arr = allocator.allocate(count);
00727             capacity = count;
00728         }
00729         for (current = 0; current < count; ++current)
00730         {
00731             std::allocator_traits<Allocator>::construct(allocator, arr + current,
      value);
00732         }
00733     }
00734
00740     template <typename... Args>
00741     void emplace_back(Args &&...args)
00742     {
00743         if (current == capacity)
00744         {
00745             reserve(capacity == 0 ? 1 : capacity * 2);
00746         }
00747         std::allocator_traits<Allocator>::construct(allocator, arr + current,
      std::forward<Args>(args)...);
00748         ++current;
00749     }
00750
00757     template <typename InputIt>
00758     void append_range(InputIt first, InputIt last)
00759     {
00760         size_type count = std::distance(first, last);
00761         if (current + count > capacity)
00762         {
00763             reserve(current + count);
00764         }
00765         for (; first != last; ++first, ++current)
00766         {
00767             std::allocator_traits<Allocator>::construct(allocator, arr + current,
```

```
      *first);
00768         }
00769     }
00770 };
00771
00782 template <typename T, typename Allocator>
00783 bool operator==(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00784 {
00785     return lhs.size() == rhs.size() && std::equal(lhs.begin(), lhs.end(),
      rhs.begin());
00786 }
00787
00798 template <typename T, typename Allocator>
00799 bool operator!=(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00800 {
00801     return !(lhs == rhs);
00802 }
00803
00814 template <typename T, typename Allocator>
00815 bool operator<(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00816 {
00817     return std::lexicographical_compare(lhs.begin(), lhs.end(), rhs.begin(),
      rhs.end());
00818 }
00819
00830 template <typename T, typename Allocator>
00831 bool operator<=(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00832 {
00833     return !(rhs < lhs);
00834 }
00835
00846 template <typename T, typename Allocator>
00847 bool operator>(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00848 {
00849     return rhs < lhs;
00850 }
00851
00862 template <typename T, typename Allocator>
00863 bool operator>=(const MyVector<T, Allocator> &lhs, const MyVector<T, Allocator> &rhs)
00864 {
00865     return !(lhs < rhs);
00866 }
00867
00876 namespace std
00877 {
00878     template <typename T, typename Allocator>
00879     void swap(MyVector<T, Allocator> &lhs, MyVector<T, Allocator> &rhs) noexcept
00880     {
00881         lhs.swap(rhs);
00882     }
00883 }
00884
00894 template <typename T, typename Allocator, typename Pred>
00895 void erase(MyVector<T, Allocator> &vec, Pred pred)
00896 {
00897     vec.erase(std::remove_if(vec.begin(), vec.end(), pred), vec.end());
00898 }
00899
00909 template <typename T, typename Allocator, typename Pred>
00910 void erase_if(MyVector<T, Allocator> &vec, Pred pred)
00911 {
00912     vec.erase(std::remove_if(vec.begin(), vec.end(), pred), vec.end());
00913 }
```

8.5 person.h

```
00001 #ifndef PERSON_H
00002 #define PERSON_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <iostream>
00007 #include <algorithm>
00008 #include <numeric>
00009 #include <deque>
00010 #include <list>
00011 #include <cmath>
00012 #include <istream>
00013 #include <ostream>
00014 #include <iomanip>
00015 #include <sstream>
00016 #include "myVector.h"
00017
00024 class Person {
00025 public:
00029     virtual ~Person() = default;
00030
00035     virtual std::string getFirstName() const = 0;
00036
00041     virtual std::string getLastName() const = 0;
00042
00047     virtual std::string getName() const = 0;
00048 };
00049
00050 #endif // PERSON_H
```

8.6 student.h

```
00001 #ifndef STUDENT_H
00002 #define STUDENT_H
00003
00004 #include "person.h"
00005
00012 class Student : public Person {
00013 private:
00014     std::string firstName, lastName;
00015     std::vector<int> homeworkResults;
00016     int examResults;
00017
00018 public:
00022     Student();
00023
00031     Student(const std::string &firstName, const std::string &lastName, int
      examResults, const std::vector<int> &homeworkResults);
00032
00037     Student(const Student &other);
00038
00043     Student(Student &&other) noexcept;
00044
00050     Student &operator=(const Student &other);
00051
00057     Student &operator=(Student &&other) noexcept;
00058
00065     friend std::istream &operator»(std::istream &is, Student &s);
00066
00073     friend std::ostream &operator«(std::ostream &os, const Student &s);
00074
00078     ~Student() {
00079         homeworkResults.clear();
```

```
00080     }
00081
00082     // Get'eriai
00087     inline std::string getFirstName() const { return firstName; }
00088
00093     inline std::string getLastName() const { return lastName; }
00094
00099     std::string getName() const { return getFirstName() + " " + getLastName(); }
00100
00105     const std::vector<int> &getHomeworkResults() const { return homeworkResults; }
00106
00111     int getExamResults() const { return examResults; }
00112
00117     int getExamGrade() const { return homeworkResults.back(); }
00118
00122     void removeLastHomeworkGrade() { if (!homeworkResults.empty()) {
      homeworkResults.pop_back(); } }
00123
00124     // Set'eriai
00129     void setFirstName(std::string firstName) { this->firstName = std::move(firstName);
      }
00130
00135     void setLastName(std::string lastName) { this->lastName = std::move(lastName); }
00136
00141     void addHomeworkResult(int result) { homeworkResults.push_back(result); }
00142
00146     void clearHomeworkResults() { homeworkResults.clear(); }
00147
00152     void setExamResults(int examResults) { this->examResults = examResults; }
00153
00158     void setHomeworkResults(std::vector<int> results) { homeworkResults =
      std::move(results); }
00159
00160     // Funkcijos
00165     double calculateMedian() const;
00166
00171     double calculateAverage() const;
00172
00178     double calculateFinalGrade(bool median) const;
00179 };
00180
00181 #endif // STUDENT_H
```