Objektinis 3

Generated by Doxygen 1.10.0

Chapter 1

Objektinio užduotis 3

Programa skaičiuojanti galutinį studento rezultatą pagal pateiktus namų darbų ir egzamino rezultatus.

1.1 Funkcionalumas

- Meniu kuriame galima pasirinkti, ką atsitiktinai generuoti.
- Duomenų skaitymas iš išankstinio failo.
- Duomenų įvedimas.
- Duomenų failo kūrimas.
- Galutinio balo skaičiavimas pagal vidurkį ir medianą.
- Galima skaityti duomenis iš tam tikru formatu pateiktų teksto failų.
- "Pažengusių" ir "Žlugusių" mokinių išvedimas atskiruose failuose.

1.2 Perdengti operatoriai

Jei naudojama "cin >> klasė" arba "cout << klasė>" atspausdinama arba įrašoma atitinkama klasės informacija visur pasitaikanti šioje užduotyje. Jei rašoma kažkas kitas, o ne klasė, jie veikia kaip įprastai. Pakeisti kur spausdinama galima arba keičiant freopen parametrus arba kode pritaikant ifstream ir ofstream, kas leistų naudoti skirtingus raktažodžius spausdinimui į skirtingas vietas.

1.3 Naudojimosi instrukcijos

- Įjungti programą.
- Sekti terminale matomus žingsnius.
- Jei prašome vesti failo pavadinimą, vesti be ".txt" pabaigoje.
- Gauti rezultatus.

1.4 Sistemos specifikacijos

- **CPU**: AMD Ryzen 5 5600H 3.30 GHz
- **RAM**: DDR4 16GB
- **HDD**: SSD 512GB

1.5 Greičio testai (5 testų vidurkis) ms

1.5.0.1 Vector

Failas	Skaitymo trukmė	Rūšiavimo trukmė	Skirstymo trukmė
stud1000	11	0	1
stud10000	87	16	1
stud100000	907	147	25
stud1000000	8296	1911	292
stud10000000	90697	25683	2911

1.5.0.2 List

Failas	Skaitymo trukmė	Rūšiavimo trukmė	Skirstymo trukmė
stud1000	14	0	1
stud10000	160	6	17
stud100000	1160	78	144
stud1000000	11753	1202	1570
stud10000000	120384	18116	40532

1.5.0.3 Deque

Failas	Skaitymo trukmė	Rūšiavimo trukmė	Skirstymo trukmė
stud1000	12	2	1
stud10000	97	29	6
stud100000	918	394	71
stud1000000	9375	5177	937
stud10000000	96751	69256	61005

1.6 Skirstymas pagal skirtingas strategijas (3 testų vidurkis) ms

1.6.0.1 Vector

Failas	1 strategija	2 strategija (originali)	3 strategija
stud1000	0	1	1
stud10000	3	1	6
stud100000	28	25	31
stud1000000	372	262	335
stud10000000	5580	2911	3398

1.6.0.2 List

Failas	1 strategija	2 strategija (originali)	3 strategija
stud1000	1	1	1
stud10000	20	17	10
stud100000	242	144	149
stud1000000	2628	1570	1806
stud10000000	60549	40532	25499

1.6.0.3 Deque

Failas	1 strategija	2 strategija (originali)	3 strategija
stud1000	1	1	0
stud10000	12	6	6
stud100000	159	71	70
stud1000000	1698	937	884
stud10000000	102817	61005	52611

1.7 Klasių ir struktūrų spartos palyginimas (naudojant vektorių)

1.7.0.1 Struktūra

Failas	Skaitymo trukmė	Rušiavimo trukmė	Skirtsymo trukmė
stud1000000	8996	1911	292
stud10000000	90697	25683	2911

1.7.0.2 Klasė

Failas	Skaitymo trukmė	Rušiavimo trukmė	Skirtsymo trukmė
stud1000000	8152	2805	125
stud10000000	86862	37644	1424

1.8 Optimizavimo "flag'ų" palyginimas (stud1000000)

	Skaitymo, rūšiavimo ir skirtsymo trukmė (ms)	.exe dysis
Struct -O1	10118	466 KB
Struct -O2	10166	466 KB
Struct -O3	10388	466 KB
Class -O1	12071	451 KB
Class -O2	11781	451 KB
Class -O3	11082	451 KB

1.9 Release'ų istorija

- V.pradinė: pirma prelimenati programa, kuri skaičiuoja ranka įvestus mokinio duomeis ir išveda galutinius rezultatus.
- v0.1: nereikia iš anksto nustatyti duomenų kiekio, padarytas atsitiktinės generacijos funkcionalumas. Programa padaryta naudojant vektorius ir, atskirai, naudojant masyvus.
- v0.2: programa gali duomenis priimti iš failo.
- v0.3: programa paskirstyta per kelis failus, pridėtas išimčių valdymas.
- v0.4: programoje galima generuoti naujus failus, duomenys atspausdinami į 2 atskirus failus, atliekama laiko analizė.
- v1.0: atliktas programos testavimas su skirtingais konteineriais, naudotos skirtingos mokinių skirstymo strategijos, padarytos jų efektyvumo strategijos.
- v1.1: programa perdaryta naudojant custom klases o ne struktūras.
- v1.2: pritaikyta rule of five, sukurti move ir copy operatoriai
- v1.5: klasė padalinta į dvi dalis, viena iš kurių abstrakti bazinė.
- v2.0: su doxygen sukurta dokumentacija ir padaryti unit testai.
- v3.0: sukurtas Vector konteineris, testai, setup failas.

1.10 Funkcijų paaiškinimas

- front(): grąžiną pirmo vektoriaus nario reikšmę.
- insert(): į pasirinktą vektoriaus vietą įterpia objektą.
- shrink_to_fit(): sumažina talpą, kad ji būtu tokia, koks ir dydis.
- pop_back(): pašalina paskutinį vektoriaus elementą, bet nesumažina jo talpos.
- begin(): grąžina iteratorių į pirmą vektoriaus narį.

1.11 Naujo vektoriaus konteinerio palyginimas

1.11.0.1 Atminties perskirstymai

Elementų sk.	std::vector	Vector
100000000	27	28

1.11.0.2 Trukmė užpildant vektorių (5 testų vidurkis)

Elementų sk.	std::vector (ms)	Vector (ms)
10000	0	0
100000	2	1
1000000	16	4

Elementų sk.	std::vector (ms)	Vector (ms)
10000000	170	41
100000000	1557	433

1.12 Kompiuterio paruošimas programai

Čia gidas Windows sistemoms.

- Atsisiųskite c++ kompiliatorių. Gidas čia: https://www.geeksforgeeks.org/installing-mingw-tools-fo
- Atsisiųskite make. Gidas čia: https://linuxhint.com/install-use-make-windows/.

1.13 Programos diegimas ir paleidimas

1. Atsisiųskite programos kodą iš repozitorijos.
2. Terminale pasiekite atsisiuntimo aplanką.
3. Terminale parašykite "make" (pirmą kartą, kai leidžiate programą).
4. Jei norite patikrinti testus, naudokite "make test".
5. Paleiskite programą terminale įvesdami .\prog.exe (Windows) arba .\prog (Linux)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

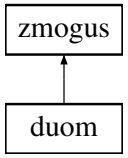
Chapter 5

Class Documentation

5.1 duom Class Reference

`#include <funkcijos.h>`

Inheritance diagram for duom:



Public Member Functions

- duom ()
- ~duom ()
- duom (istream &cin)
- duom (const duom &temp)
- duom (duom &&temp) noexcept
- duom & operator= (const duom &temp)
- duom & operator= (duom &&temp) noexcept
- string vard () const override
- string pav () const override
- double galvid () const
- double galmed () const
- void vardas (const string &va) override
- void pavarde (const string &pa) override
- void nd (int nd)
- void egz (int egz)
- void calc ()
- void vpskait ()
- void skaitduom ()
- void spausdinti ()
- void vardoGen ()
- void ndGen ()
- void egzGen ()

Public Member Functions inherited from zmogus

- zmogus ()
- ~zmogus ()
- zmogus (const zmogus &temp)
- zmogus (zmogus &&temp) noexcept
- zmogus & operator= (const zmogus &temp)
- zmogus & operator= (zmogus &&temp) noexcept

Friends

- istream & operator>> (istream &cin, duom &s)
- ostream & operator<< (ostream &cout, const duom &s)

Additional Inherited Members

Protected Attributes inherited from zmogus

- string vard_
- string pav_

5.1.1 Constructor & Destructor Documentation

5.1.1.1 duom() [1/4]

```
duom::duom ( )  [inline]
```

5.1.1.2 ~duom()

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

5.1.1.3 duom() [2/4]

```
duom::duom (
            istream & cin )
```

5.1.1.4 duom() [3/4]

```
duom::duom (
            const duom & temp )  [inline]
```

5.1.1.5 duom() [4/4]

```
duom::duom (
            duom && temp )  [inline], [noexcept]
```

5.1.2 Member Function Documentation

5.1.2.1 calc()

```
void duom::calc ( )
```

5.1.2.2 egz()

```
void duom::egz (
            int egz )  [inline]
```

5.1.2.3 egzGen()

```
void duom::egzGen ( )
```

5.1.2.4 galmed()

```
double duom::galmed ( ) const  [inline]
```

5.1.2.5 galvid()

```
double duom::galvid ( ) const  [inline]
```

5.1.2.6 nd()

```
void duom::nd (
            int nd )  [inline]
```

5.1.2.7 ndGen()

```
void duom::ndGen ( )
```

5.1.2.8 operator=() [1/2]

```
duom & duom::operator= (
            const duom & temp )  [inline]
```

5.1.2.9 operator=() [2/2]

```
duom & duom::operator= (
            duom && temp )  [inline], [noexcept]
```

5.1.2.10 pav()

```
string duom::pav ( ) const  [inline], [override], [virtual]
```

Implements zmogus.

5.1.2.11 pavarde()

```
void duom::pavarde (
            const string & pa )  [inline], [override], [virtual]
```

Implements zmogus.

5.1.2.12 skaitduom()

```
void duom::skaitduom ( )
```

5.1.2.13 spausdinti()

```
void duom::spausdinti ( )
```

5.1.2.14 vard()

```
string duom::vard ( ) const  [inline], [override], [virtual]
```

Implements zmogus.

5.1.2.15 vardas()

```
void duom::vardas (
            const string & va )  [inline], [override], [virtual]
```

Implements zmogus.

5.1.2.16 vardoGen()

```
void duom::vardoGen ( )
```

5.1.2.17 vpskait()

```
void duom::vpskait ( )
```

5.1.3 Friends And Related Symbol Documentation

5.1.3.1 operator<<

```
ostream & operator<< (
            ostream & cout,
            const duom & s )  [friend]
```

5.1.3.2 operator>>

```
istream & operator>> (
            istream & cin,
            duom & s )  [friend]
```

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

- funkcijos.h
- funkcijos.cpp

5.2 Vector< T > Class Template Reference

```
#include <Vector.h>
```

Public Types

- using iterator = T*
- using const_iterator = const T*
- using reverse_iterator = std::reverse_iterator<iterator>
- using const_reverse_iterator = std::reverse_iterator<const_iterator>

Public Member Functions

- Vector ()
- Vector (const Vector &other)
- Vector (Vector &&other) noexcept
- Vector (std::initializer_list< T > init_list)
- ~Vector ()
- Vector & operator= (const Vector &other)
- Vector & operator= (Vector &&other) noexcept
- Vector & operator= (std::initializer_list< T > init_list)
- T & operator[ ] (size_t index)
- const T & operator[ ] (size_t index) const
- void reserve (size_t new_capacity)
- void shrink_to_fit ()
- void resize (size_t count)
- void resize (size_t count, const T &x)
- void swap (Vector &other) noexcept
- void push_back (const T &x)

- void pop_back ()
- size_t getSize () const
- size_t max_size () const
- size_t capacity () const
- bool empty () const
- void clear () noexcept
- T & front ()
- const T & front () const
- T & back ()
- const T & back () const
- void assign (size_t count, const T &x)
- template<typename InputIt >
  void assign (InputIt first, InputIt last)
- T & at (size_t index)
- const T & at (size_t index) const
- T * data () noexcept
- const T * data () const noexcept
- std::allocator< T > get_allocator () const
- iterator begin () noexcept
- const_iterator begin () const noexcept
- iterator end () noexcept
- const_iterator end () const noexcept
- const_iterator cbegin () const noexcept
- const_iterator cend () const noexcept
- reverse_iterator rbegin () noexcept
- const_reverse_iterator rbegin () const noexcept
- reverse_iterator rend () noexcept
- const_reverse_iterator rend () const noexcept
- const_reverse_iterator crbegin () const noexcept
- const_reverse_iterator crend () const noexcept
- iterator insert (const_iterator pos, const T &x)
- iterator insert (const_iterator pos, T &&x)
- iterator insert (const_iterator pos, size_t count, const T &x)
- template<class InputIt >
  iterator insert (const_iterator pos, InputIt first, InputIt last)
- iterator insert (const_iterator pos, std::initializer_list< T > ilist)
- template<typename... Args>
  iterator emplace (const_iterator pos, Args &&... args)
- template<typename... Args>
  void emplace_back (Args &&... args)
- iterator erase (const_iterator pos)
- iterator erase (const_iterator first, const_iterator last)

5.2.1 Member Typedef Documentation

5.2.1.1 const_iterator

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

5.2.1.2 const_reverse_iterator

```
template<typename T >
using Vector< T >::const_reverse_iterator = std::reverse_iterator<const_iterator>
```

5.2.1.3 iterator

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

5.2.1.4 reverse_iterator

```
template<typename T >
using Vector< T >::reverse_iterator = std::reverse_iterator<iterator>
```

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Vector() [1/4]

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

5.2.2.2 Vector() [2/4]

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

5.2.2.3 Vector() [3/4]

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

5.2.2.4 Vector() [4/4]

```
template<typename T >
Vector< T >::Vector (
            std::initializer_list< T > init_list )
```

5.2.2.5 ~Vector()

```
template<typename T >
Vector< T >::~Vector ( )
```

5.2.3 Member Function Documentation

5.2.3.1 assign() [1/2]

```
template<typename T >
template<typename InputIt >
void Vector< T >::assign (
            InputIt first,
            InputIt last )
```

5.2.3.2 assign() [2/2]

```
template<typename T >
void Vector< T >::assign (
            size_t count,
            const T & x )
```

5.2.3.3 at() [1/2]

```
template<typename T >
T & Vector< T >::at (
            size_t index )
```

5.2.3.4 at() [2/2]

```
template<typename T >
const T & Vector< T >::at (
            size_t index ) const
```

5.2.3.5 back() [1/2]

```
template<typename T >
T & Vector< T >::back ( )
```

5.2.3.6 back() [2/2]

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

5.2.3.7 begin() [1/2]

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

5.2.3.8 begin() [2/2]

```
template<typename T >
iterator Vector< T >::begin ( )  [inline], [noexcept]
```

5.2.3.9 capacity()

```
template<typename T >
size_t Vector< T >::capacity ( ) const
```

5.2.3.10 cbegin()

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

5.2.3.11 cend()

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

5.2.3.12 clear()

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

5.2.3.13 crbegin()

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

5.2.3.14 crend()

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

5.2.3.15 data() [1/2]

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

5.2.3.16 data() [2/2]

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

5.2.3.17 emplace()

```
template<typename T >
template<typename... Args>
Vector< T >::iterator Vector< T >::emplace (
            const_iterator pos,
            Args &&... args )
```

5.2.3.18 emplace_back()

```
template<typename T >
template<typename... Args>
void Vector< T >::emplace_back (
            Args &&... args )
```

5.2.3.19 empty()

```
template<typename T >
bool Vector< T >::empty ( ) const
```

5.2.3.20 end() [1/2]

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

5.2.3.21 end() [2/2]

```
template<typename T >
iterator Vector< T >::end ( )  [inline], [noexcept]
```

5.2.3.22 erase() [1/2]

```
template<typename T >
Vector< T >::iterator Vector< T >::erase (
            const_iterator first,
            const_iterator last )
```

5.2.3.23 erase() [2/2]

```
template<typename T >
Vector< T >::iterator Vector< T >::erase (
            const_iterator pos )
```

5.2.3.24 front() [1/2]

```
template<typename T >
T & Vector< T >::front ( )
```

5.2.3.25 front() [2/2]

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

5.2.3.26 get_allocator()

```
template<typename T >
std::allocator< T > Vector< T >::get_allocator ( ) const
```

5.2.3.27 getSize()

```
template<typename T >
size_t Vector< T >::getSize ( ) const
```

5.2.3.28 insert() [1/5]

```
template<typename T >
Vector< T >::iterator Vector< T >::insert (
            const_iterator pos,
            const T & x )
```

5.2.3.29 insert() [2/5]

```
template<typename T >
template<class InputIt >
Vector< T >::iterator Vector< T >::insert (
            const_iterator pos,
            InputIt first,
            InputIt last )
```

5.2.3.30 insert() [3/5]

```
template<typename T >
Vector< T >::iterator Vector< T >::insert (
            const_iterator pos,
            size_t count,
            const T & x )
```

5.2.3.31 insert() [4/5]

```
template<typename T >
Vector< T >::iterator Vector< T >::insert (
            const_iterator pos,
            std::initializer_list< T > ilist )
```

5.2.3.32 insert() [5/5]

```
template<typename T >
Vector< T >::iterator Vector< T >::insert (
            const_iterator pos,
            T && x )
```

5.2.3.33 max_size()

```
template<typename T >
size_t Vector< T >::max_size ( ) const
```

5.2.3.34 operator=() [1/3]

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

5.2.3.35 operator=() [2/3]

```
template<typename T >
Vector< T > & Vector< T >::operator= (
            std::initializer_list< T > init_list )
```

5.2.3.36 operator=() [3/3]

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

5.2.3.37 operator[]() [1/2]

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

5.2.3.38 operator[]() [2/2]

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

5.2.3.39 pop_back()

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

5.2.3.40 push_back()

```
template<typename T >
void Vector< T >::push_back (
            const T & x )
```

5.2.3.41 rbegin() [1/2]

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

5.2.3.42 rbegin() [2/2]

```
template<typename T >
reverse_iterator Vector< T >::rbegin ( )  [inline], [noexcept]
```

5.2.3.43 rend() [1/2]

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

5.2.3.44 rend() [2/2]

```
template<typename T >
reverse_iterator Vector< T >::rend ( )  [inline], [noexcept]
```

5.2.3.45 reserve()

```
template<typename T >
void Vector< T >::reserve (
            size_t new_capacity )
```

5.2.3.46 resize() [1/2]

```
template<typename T >
void Vector< T >::resize (
            size_t count )
```

5.2.3.47 resize() [2/2]

```
template<typename T >
void Vector< T >::resize (
            size_t count,
            const T & x )
```

5.2.3.48 shrink_to_fit()

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

5.2.3.49 swap()

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

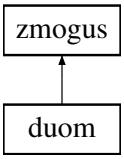
The documentation for this class was generated from the following file:

- Vector.h

5.3 zmogus Class Reference

```
#include <funkcijos.h>
```

Inheritance diagram for zmogus:



Public Member Functions

- virtual void vardas (const string &va)=0
- virtual void pavarde (const string &pa)=0
- virtual string vard () const =0
- virtual string pav () const =0
- zmogus ()
- ~zmogus ()
- zmogus (const zmogus &temp)
- zmogus (zmogus &&temp) noexcept
- zmogus & operator= (const zmogus &temp)
- zmogus & operator= (zmogus &&temp) noexcept

Protected Attributes

- string vard_
- string pav_

5.3.1 Constructor & Destructor Documentation

5.3.1.1 zmogus() [1/3]

```
zmogus::zmogus ( )  [inline]
```

5.3.1.2 ~zmogus()

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

5.3.1.3 zmogus() [2/3]

```
zmogus::zmogus (
            const zmogus & temp )  [inline]
```

5.3.1.4 zmogus() [3/3]

```
zmogus::zmogus (
            zmogus && temp )  [inline], [noexcept]
```

5.3.2 Member Function Documentation

5.3.2.1 operator=() [1/2]

```
zmogus & zmogus::operator= (
            const zmogus & temp )  [inline]
```

5.3.2.2 operator=() [2/2]

```
zmogus & zmogus::operator= (
            zmogus && temp )  [inline], [noexcept]
```

5.3.2.3 pav()

```
virtual string zmogus::pav ( ) const  [pure virtual]
```

Implemented in duom.

5.3.2.4 pavarde()

```
virtual void zmogus::pavarde (
            const string & pa )  [pure virtual]
```

Implemented in duom.

5.3.2.5 vard()

```
virtual string zmogus::vard ( ) const  [pure virtual]
```

Implemented in duom.

5.3.2.6 vardas()

```
virtual void zmogus::vardas (
            const string & va )  [pure virtual]
```

Implemented in duom.

5.3.3 Member Data Documentation

5.3.3.1 pav_

```
string zmogus::pav_  [protected]
```

5.3.3.2 vard_

```
string zmogus::vard_  [protected]
```

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

- funkcijos.h

Chapter 6

File Documentation

6.1 funkcijos.cpp File Reference

```
#include "funkcijos.h"
```

Functions

- bool sort1 (const duom &a, const duom &b)
- bool sort2 (const duom &a, const duom &b)
- bool sort3 (const duom &a, const duom &b)
- bool sort4 (const duom &a, const duom &b)
- bool sort1u (const duom &a, const duom &b)
- bool sort2u (const duom &a, const duom &b)
- bool sort3u (const duom &a, const duom &b)
- bool sort4u (const duom &a, const duom &b)
- istream & operator>> (istream &cin, duom &s)
- ostream & operator<< (ostream &cout, duom &s)
- template<typename sk = int, typename talpa >
  void rusiuoti (sk &x, sk &t, talpa &mok)
- template<typename sk = int>
  void rusiuotilist (sk &x, sk &t, list< duom > &mok)
- bool pagalVid (const duom &x, const double d)
- bool pagalMed (const duom &x, const double d)
- template<typename talpa , typename sk >
  void strategija3 (talpa &x, talpa &y, sk t)
- template<typename talpa , typename sk >
  void strategija2 (talpa &x, talpa &y, sk t)
- template<typename talpa , typename sk >
  void strategija1 (talpa &x, talpa &y, sk t, talpa &z)
- template<typename sk , typename talpa >
  void skaitymas (sk &moksk, talpa &mok)
- template<typename talpa , typename sk = int>
  void isfailo (talpa &mok, sk &s)
- void kurtifaila ()
- template<typename sk , typename talpa >
  void rankinis (sk &x, sk &moksk, talpa &mok)
- void input ()
- void testas ()

6.1.1 Function Documentation

6.1.1.1 input()

```
void input ( )
```

6.1.1.2 isfailo()

```
template<typename talpa , typename sk = int>
void isfailo (
            talpa & mok,
            sk & s )
```

6.1.1.3 kurtifaila()

```
void kurtifaila ( )
```

6.1.1.4 operator<<()

```
ostream & operator<< (
            ostream & cout,
            duom & s )
```

6.1.1.5 operator>>()

```
istream & operator>> (
            istream & cin,
            duom & s )
```

6.1.1.6 pagalMed()

```
bool pagalMed (
            const duom & x,
            const double d )
```

6.1.1.7 pagalVid()

```
bool pagalVid (
            const duom & x,
            const double d )
```

6.1.1.8 rankinis()

```
template<typename sk , typename talpa >
void rankinis (
            sk & x,
            sk & moksk,
            talpa & mok )
```

6.1.1.9 rusiuoti()

```
template<typename sk = int, typename talpa >
void rusiuoti (
             sk & x,
             sk & t,
             talpa & mok )
```

6.1.1.10 rusiuotilist()

```
template<typename sk = int>
void rusiuotilist (
             sk & x,
             sk & t,
             list< duom > & mok )
```

6.1.1.11 skaitymas()

```
template<typename sk , typename talpa >
void skaitymas (
             sk & moksk,
             talpa & mok )
```

6.1.1.12 sort1()

```
bool sort1 (
             const duom & a,
             const duom & b )
```

6.1.1.13 sort1u()

```
bool sort1u (
             const duom & a,
             const duom & b )
```

6.1.1.14 sort2()

```
bool sort2 (
             const duom & a,
             const duom & b )
```

6.1.1.15 sort2u()

```
bool sort2u (
             const duom & a,
             const duom & b )
```

6.1.1.16 sort3()

```
bool sort3 (
            const duom & a,
            const duom & b )
```

6.1.1.17 sort3u()

```
bool sort3u (
            const duom & a,
            const duom & b )
```

6.1.1.18 sort4()

```
bool sort4 (
            const duom & a,
            const duom & b )
```

6.1.1.19 sort4u()

```
bool sort4u (
            const duom & a,
            const duom & b )
```

6.1.1.20 strategija1()

```
template<typename talpa , typename sk >
void strategija1 (
            talpa & x,
            talpa & y,
            sk t,
            talpa & z )
```

6.1.1.21 strategija2()

```
template<typename talpa , typename sk >
void strategija2 (
            talpa & x,
            talpa & y,
            sk t )
```

6.1.1.22 strategija3()

```
template<typename talpa , typename sk >
void strategija3 (
            talpa & x,
            talpa & y,
            sk t )
```

6.1.1.23 testas()

```
void testas ( )
```

6.2 funkcijos.h File Reference

```
#include <bits/stdc++.h>
#include "Vector.h"
```

Classes

- class zmogus
- class duom

Functions

- bool sort1 (const duom &, const duom &)
- bool sort2 (const duom &, const duom &)
- bool sort3 (const duom &, const duom &)
- bool sort4 (const duom &, const duom &)
- bool sort1u (const duom &, const duom &)
- bool sort2u (const duom &, const duom &)
- bool sort3u (const duom &, const duom &)
- bool sort4u (const duom &, const duom &)
- bool pagalVid (const duom &x, const double d)
- bool pagalMed (const duom &x, const double d)
- template<typename sk = int, typename talpa >
  void rusiuoti (sk &, sk &, talpa &)
- template<typename talpa , typename sk >
  void strategija3 (talpa &, talpa &, sk)
- template<typename talpa , typename sk >
  void strategija2 (talpa &, talpa &, sk)
- template<typename talpa , typename sk >
  void strategija1 (talpa &, talpa &, sk, talpa &)
- template<typename sk , typename talpa >
  void skaitymas (sk &, talpa &)
- template<typename talpa , typename sk = int>
  void isfailo (talpa &, sk &)
- void kurtifaila ()
- template<typename sk , typename talpa >
  double rankinis (sk &, talpa &, sk &)
- void input ()
- void testas ()

6.2.1 Function Documentation

6.2.1.1 input()

```
void input ( )
```

6.2.1.2 isfailo()

```
template<typename talpa , typename sk = int>
void isfailo (
             talpa & mok,
             sk & s )
```

6.2.1.3 kurtifaila()

```
void kurtifaila ( )
```

6.2.1.4 pagalMed()

```
bool pagalMed (
             const duom & x,
             const double d )
```

6.2.1.5 pagalVid()

```
bool pagalVid (
             const duom & x,
             const double d )
```

6.2.1.6 rankinis()

```
template<typename sk , typename talpa >
double rankinis (
             sk & ,
             talpa & ,
             sk &  )
```

6.2.1.7 rusiuoti()

```
template<typename sk = int, typename talpa >
void rusiuoti (
             sk & x,
             sk & t,
             talpa & mok )
```

6.2.1.8 skaitymas()

```
template<typename sk , typename talpa >
void skaitymas (
             sk & moksk,
             talpa & mok )
```

6.2.1.9 sort1()

```
bool sort1 (
            const duom & a,
            const duom & b )
```

6.2.1.10 sort1u()

```
bool sort1u (
            const duom & a,
            const duom & b )
```

6.2.1.11 sort2()

```
bool sort2 (
            const duom & a,
            const duom & b )
```

6.2.1.12 sort2u()

```
bool sort2u (
            const duom & a,
            const duom & b )
```

6.2.1.13 sort3()

```
bool sort3 (
            const duom & a,
            const duom & b )
```

6.2.1.14 sort3u()

```
bool sort3u (
            const duom & a,
            const duom & b )
```

6.2.1.15 sort4()

```
bool sort4 (
            const duom & a,
            const duom & b )
```

6.2.1.16 sort4u()

```
bool sort4u (
            const duom & a,
            const duom & b )
```

6.2.1.17 strategija1()

```
template<typename talpa , typename sk >
void strategija1 (
             talpa & x,
             talpa & y,
             sk t,
             talpa & z )
```

6.2.1.18 strategija2()

```
template<typename talpa , typename sk >
void strategija2 (
             talpa & x,
             talpa & y,
             sk t )
```

6.2.1.19 strategija3()

```
template<typename talpa , typename sk >
void strategija3 (
             talpa & x,
             talpa & y,
             sk t )
```

6.2.1.20 testas()

```
void testas ( )
```

6.3 funkcijos.h

Go to the documentation of this file.

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003
00004 #include <bits/stdc++.h>
00005 #include "Vector.h"
00006
00007 using namespace std;
00008 using namespace std::chrono;
00009 using std::setw;
00010 using std::left;
00011
00012 class zmogus{
00013     protected:
00014         string vard_;
00015         string pav_;
00016     public:
00017         virtual void vardas(const string &va) = 0;
00018         virtual void pavarde(const string &pa) = 0;
00019
00020         virtual string vard() const = 0;
00021         virtual string pav() const = 0;
00022
00023         zmogus() {}
00024         ~zmogus() {}
00025
00026         // copy c
00027         zmogus(const zmogus &temp)
```

```
00028             : vard_(temp.vard_), pav_(temp.pav_) {}
00029
00030         // move c
00031         zmogus(zmogus &&temp) noexcept
00032             : vard_(move(temp.vard_)), pav_(move(temp.pav_)) {}
00033
00034         // copy a
00035         zmogus& operator=(const zmogus &temp) {
00036             if(this!=&temp){
00037                 vard_=temp.vard_;
00038                 pav_=temp.pav_;
00039             }
00040             return *this;
00041         }
00042
00043         // move a
00044         zmogus& operator=(zmogus &&temp) noexcept {
00045             if(this!=&temp){
00046                 vard_=move(temp.vard_);
00047                 pav_=move(temp.pav_);
00048             }
00049             return *this;
00050         }
00051 };
00052
00053 class duom : public zmogus{
00054     private:
00055         Vector<int> ndrez_;
00056         int egzrez_;
00057         double galvid_, galmed_;
00058     public:
00059         duom() : galvid_(0), galmed_(0) {}
00060         ~duom() {}
00061         duom(istream &cin);
00062
00063         // copy c
00064         duom(const duom &temp)
00065             : zmogus(temp), ndrez_(temp.ndrez_), egzrez_(temp.egzrez_), galvid_(temp.galvid_),
      galmed_(temp.galmed_) {}
00066
00067         // move c
00068         duom(duom &&temp) noexcept
00069             : zmogus(move(temp)), ndrez_(move(temp.ndrez_)), egzrez_(temp.egzrez_),
      galvid_(temp.galvid_), galmed_(temp.galmed_) {
00070                 temp.egzrez_={};
00071                 temp.galvid_={};
00072                 temp.galmed_={};
00073                 temp.ndrez_.clear();
00074             }
00075
00076         // copy a
00077         duom& operator=(const duom &temp) {
00078             if(this!=&temp){
00079                 zmogus::operator=(temp);
00080                 ndrez_=temp.ndrez_;
00081                 egzrez_=temp.egzrez_;
00082                 galvid_=temp.galvid_;
00083                 galmed_=temp.galmed_;
00084             }
00085             return *this;
00086         }
00087
00088         // move a
00089         duom& operator=(duom &&temp) noexcept {
00090             if(this!=&temp){
00091                 zmogus::operator=(move(temp));
00092                 ndrez_=move(temp.ndrez_);
00093                 egzrez_=move(temp.egzrez_);
00094                 temp.egzrez_={};
00095                 galvid_=move(temp.galvid_);
00096                 temp.galvid_={};
00097                 galmed_=move(temp.galmed_);
00098                 temp.galmed_={};
00099                 temp.ndrez_.clear();
00100             }
00101             return *this;
00102         }
00103
00104         inline string vard() const override { return this->vard_; }
00105         inline string pav() const override { return this->pav_; }
00106         inline double galvid() const { return galvid_; }
00107         inline double galmed() const { return galmed_; }
00108
00109         void vardas(const string &va) override { this->vard_=va; }
00110         void pavarde(const string &pa) override { this->pav_=pa; }
00111         void nd(int nd) { ndrez_.push_back(nd); }
00112         void egz(int egz) { egzrez_=egz; }
```

```
00113         void calc();
00114
00115         void vpskait();
00116         void skaitduom();
00117         void spausdinti();
00118         void vardoGen();
00119         void ndGen();
00120         void egzGen();
00121
00122         friend istream& operator»(istream &cin, duom &s);
00123         friend ostream& operator«(ostream &cout, const duom &s);
00124
00125 };
00126
00127 bool sort1(const duom &, const duom &);
00128 bool sort2(const duom &, const duom &);
00129 bool sort3(const duom &, const duom &);
00130 bool sort4(const duom &, const duom &);
00131 bool sort1u(const duom &, const duom &);
00132 bool sort2u(const duom &, const duom &);
00133 bool sort3u(const duom &, const duom &);
00134 bool sort4u(const duom &, const duom &);
00135 bool pagalVid(const duom &x, const double d);
00136 bool pagalMed(const duom &x, const double d);
00137
00138 template <typename sk=int, typename talpa>
00139 void rusiuoti(sk &, sk &, talpa &);
00140
00141 template <typename talpa, typename sk>
00142 void strategija3(talpa &, talpa &, sk);
00143
00144 template <typename talpa, typename sk>
00145 void strategija2(talpa &, talpa &, sk);
00146
00147 template <typename talpa, typename sk>
00148 void strategija1(talpa &, talpa &, sk, talpa &);
00149
00150 template <typename sk, typename talpa>
00151 void skaitymas(sk &, talpa &);
00152
00153 template <typename talpa, typename sk=int>
00154 void isfailo(talpa &, sk &);
00155
00156 void kurtifaila();
00157
00158 template <typename sk, typename talpa>
00159 double rankinis(sk &, talpa &, sk &);
00160
00161 void input();
00162
00163 void testas();
00164
00165 #endif
```

6.4 Main.cpp File Reference

```
#include "funkcijos.h"
```

Functions

- int main ()

6.4.1 Function Documentation

6.4.1.1 main()

```
int main ( )
```

6.5 README.md File Reference

6.6 tests.cpp File Reference

```
#include "funkcijos.h"
#include "catch2/catch.hpp"
```

Macros

- #define CATCH_CONFIG_MAIN

Functions

- TEST_CASE ("Input Operation")
- TEST_CASE ("Copy Constructor")
- TEST_CASE ("Move Constructor")
- TEST_CASE ("Copy Assignment")
- TEST_CASE ("Move Assignment")
- TEST_CASE ("Vector Constructor")
- TEST_CASE ("Vector Resize")
- TEST_CASE ("Reserve/shrink_to_fit")
- TEST_CASE ("Push_back")
- TEST_CASE ("Pop_back")
- TEST_CASE ("front/back/at")
- TEST_CASE ("iterators")
- TEST_CASE ("insert")
- TEST_CASE ("erase")
- TEST_CASE ("swap")
- TEST_CASE ("clear")
- TEST_CASE ("emplace")

6.6.1 Macro Definition Documentation

6.6.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

6.6.2 Function Documentation

6.6.2.1 TEST_CASE() [1/17]

```
TEST_CASE (
            "clear"  )
```

6.6.2.2 TEST_CASE() [2/17]

```
TEST_CASE (
            "Copy Assignment"  )
```

6.6.2.3 TEST_CASE() [3/17]

```
TEST_CASE (
            "Copy Constructor"  )
```

6.6.2.4 TEST_CASE() [4/17]

```
TEST_CASE (
            "emplace"  )
```

6.6.2.5 TEST_CASE() [5/17]

```
TEST_CASE (
            "erase"  )
```

6.6.2.6 TEST_CASE() [6/17]

```
TEST_CASE (
            "front/back/at"  )
```

6.6.2.7 TEST_CASE() [7/17]

```
TEST_CASE (
            "Input Operation"  )
```

6.6.2.8 TEST_CASE() [8/17]

```
TEST_CASE (
            "insert"  )
```

6.6.2.9 TEST_CASE() [9/17]

```
TEST_CASE (
            "iterators"  )
```

6.6.2.10 TEST_CASE() [10/17]

```
TEST_CASE (
            "Move Assignment"  )
```

6.6.2.11 TEST_CASE() [11/17]

```
TEST_CASE (
            "Move Constructor"  )
```

6.6.2.12 TEST_CASE() [12/17]

```
TEST_CASE (
            "Pop_back"  )
```

6.6.2.13 TEST_CASE() [13/17]

```
TEST_CASE (
            "Push_back"  )
```

6.6.2.14 TEST_CASE() [14/17]

```
TEST_CASE (
            "Reserve/shrink_to_fit"  )
```

6.6.2.15 TEST_CASE() [15/17]

```
TEST_CASE (
            "swap"  )
```

6.6.2.16 TEST_CASE() [16/17]

```
TEST_CASE (
            "Vector Constructor"  )
```

6.6.2.17 TEST_CASE() [17/17]

```
TEST_CASE (
            "Vector Resize"  )
```

6.7 Vector.h File Reference

```
#include <iostream>
#include <stdexcept>
#include <initializer_list>
#include <algorithm>
#include <memory>
#include <iterator>
#include <limits>
```

Classes

- class Vector< T >

6.8 Vector.h

Go to the documentation of this file.

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <iostream>
00005 #include <stdexcept>
00006 #include <initializer_list>
00007 #include <algorithm>
00008 #include <memory>
00009 #include <iterator>
00010 #include <limits>
00011
00012 template <typename T>
00013 class Vector {
00014 private:
00015     T* data_;
00016     size_t size_;
00017     size_t capacity_;
00018     std::allocator<T> allocator_;
00019
00020 public:
00021     Vector() : data_(nullptr), size_(0), capacity_(0) {}
00022     Vector(const Vector& other);
00023     Vector(Vector&& other) noexcept;
00024     Vector(std::initializer_list<T> init_list);
00025     ~Vector();
00026
00027     Vector& operator=(const Vector& other);
00028     Vector& operator=(Vector&& other) noexcept;
00029     Vector& operator=(std::initializer_list<T> init_list);
00030
00031     T& operator[](size_t index);
00032     const T& operator[](size_t index) const;
00033
00034     void reserve(size_t new_capacity);
00035     void shrink_to_fit();
00036     void resize(size_t count);
00037     void resize(size_t count, const T& x);
00038     void swap(Vector& other) noexcept;
00039     void push_back(const T& x);
00040     void pop_back();
00041     size_t getSize() const;
00042     size_t max_size() const;
00043     size_t capacity() const;
00044     bool empty() const;
00045     void clear() noexcept;
00046     T& front();
00047     const T& front() const;
00048     T& back();
00049     const T& back() const;
00050     void assign(size_t count, const T& x);
00051     template <typename InputIt>
00052     void assign(InputIt first, InputIt last);
00053     // template <typename InputIt>
00054     // void append_range(InputIt first, InputIt last);
00055
00056     T& at(size_t index);
00057     const T& at(size_t index) const;
00058     T* data() noexcept;
00059     const T* data() const noexcept;
00060     std::allocator<T> get_allocator() const;
00061
00062     using iterator = T*;
00063     using const_iterator = const T*;
00064     using reverse_iterator = std::reverse_iterator<iterator>;
00065     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00066
00067     iterator begin() noexcept { return data_; }
00068     const_iterator begin() const noexcept { return data_; }
00069     iterator end() noexcept { return data_ + size_; }
00070     const_iterator end() const noexcept { return data_ + size_; }
00071     const_iterator cbegin() const noexcept { return data_; }
00072     const_iterator cend() const noexcept { return data_ + size_; }
00073     reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
```

```
00074     const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
00075     reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
00076     const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
00077     const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(cend()); }
00078     const_reverse_iterator crend() const noexcept { return const_reverse_iterator(cbegin()); }
00079
00080     iterator insert(const_iterator pos, const T& x);
00081     iterator insert(const_iterator pos, T&& x);
00082     iterator insert(const_iterator pos, size_t count, const T& x);
00083     template <class InputIt>
00084     iterator insert(const_iterator pos, InputIt first, InputIt last);
00085     iterator insert(const_iterator pos, std::initializer_list<T> ilist);
00086
00087     template <typename... Args>
00088     iterator emplace(const_iterator pos, Args&&... args);
00089
00090     template <typename... Args>
00091     void emplace_back(Args&&... args);
00092
00093     iterator erase(const_iterator pos);
00094     iterator erase(const_iterator first, const_iterator last);
00095
00096 };
00097
00098 template <typename T>
00099 Vector<T>::Vector(const Vector& other) : data_(nullptr), size_(0), capacity_(0) {
00100     if (other.size_ > 0) {
00101         size_ = other.size_;
00102         capacity_ = other.capacity_;
00103         data_ = allocator_.allocate(capacity_);
00104         for (size_t i = 0; i < size_; ++i) {
00105             allocator_.construct(&data_[i], other.data_[i]);
00106         }
00107     }
00108 }
00109
00110 template <typename T>
00111 Vector<T>::Vector(Vector&& other) noexcept : data_(nullptr), size_(0), capacity_(0) {
00112     swap(other);
00113 }
00114
00115 template <typename T>
00116 Vector<T>::Vector(std::initializer_list<T> init_list) : data_(nullptr), size_(0), capacity_(0) {
00117     size_ = init_list.size();
00118     capacity_ = init_list.size();
00119     data_ = allocator_.allocate(capacity_);
00120     size_t i = 0;
00121     for (const auto& elem : init_list) {
00122         allocator_.construct(&data_[i++], elem);
00123     }
00124 }
00125
00126 template <typename T>
00127 Vector<T>::~Vector() {
00128     clear();
00129     allocator_.deallocate(data_, capacity_);
00130 }
00131
00132 template <typename T>
00133 Vector<T>& Vector<T>::operator=(const Vector& other) {
00134     if (this != &other) {
00135         Vector temp(other);
00136         swap(temp);
00137     }
00138     return *this;
00139 }
00140
00141 template <typename T>
00142 Vector<T>& Vector<T>::operator=(Vector&& other) noexcept {
00143     if (this != &other) {
00144         clear();
00145         allocator_.deallocate(data_, capacity_);
00146         data_ = other.data_;
00147         size_ = other.size_;
00148         capacity_ = other.capacity_;
00149         other.data_ = nullptr;
00150         other.size_ = 0;
00151         other.capacity_ = 0;
00152     }
00153     return *this;
00154 }
00155
00156 template <typename T>
00157 Vector<T>& Vector<T>::operator=(std::initializer_list<T> init_list) {
00158     Vector temp(init_list);
00159     swap(temp);
00160     return *this;
```

```
00161 }
00162
00163 template <typename T>
00164 T& Vector<T>::operator[](size_t index) {
00165     if (index >= size_) {
00166         throw std::out_of_range("Index out of range");
00167     }
00168     return data_[index];
00169 }
00170
00171 template <typename T>
00172 const T& Vector<T>::operator[](size_t index) const {
00173     if (index >= size_) {
00174         throw std::out_of_range("Index out of range");
00175     }
00176     return data_[index];
00177 }
00178
00179 template <typename T>
00180 void Vector<T>::reserve(size_t new_capacity) {
00181     if (new_capacity > capacity_) {
00182         T* new_data = allocator_.allocate(new_capacity);
00183         for (size_t i = 0; i < size_; ++i) {
00184             allocator_.construct(&new_data[i], std::move(data_[i]));
00185             allocator_.destroy(&data_[i]);
00186         }
00187         allocator_.deallocate(data_, capacity_);
00188         data_ = new_data;
00189         capacity_ = new_capacity;
00190     }
00191 }
00192
00193 template <typename T>
00194 void Vector<T>::shrink_to_fit() {
00195     if (capacity_ > size_) {
00196         T* new_data = allocator_.allocate(size_);
00197         for (size_t i = 0; i < size_; ++i) {
00198             allocator_.construct(&new_data[i], std::move(data_[i]));
00199             allocator_.destroy(&data_[i]);
00200         }
00201         allocator_.deallocate(data_, capacity_);
00202         data_ = new_data;
00203         capacity_ = size_;
00204     }
00205 }
00206
00207 template <typename T>
00208 void Vector<T>::resize(size_t count) {
00209     if (count > capacity_) {
00210         reserve(count);
00211     }
00212     if (count > size_) {
00213         for (size_t i = size_; i < count; ++i) {
00214             allocator_.construct(&data_[i], T());
00215         }
00216     } else {
00217         for (size_t i = count; i < size_; ++i) {
00218             allocator_.destroy(&data_[i]);
00219         }
00220     }
00221     size_ = count;
00222 }
00223
00224 template <typename T>
00225 void Vector<T>::resize(size_t count, const T& x) {
00226     if (count > capacity_) {
00227         reserve(count);
00228     }
00229     if (count > size_) {
00230         for (size_t i = size_; i < count; ++i) {
00231             allocator_.construct(&data_[i], x);
00232         }
00233     } else {
00234         for (size_t i = count; i < size_; ++i) {
00235             allocator_.destroy(&data_[i]);
00236         }
00237     }
00238     size_ = count;
00239 }
00240
00241 template <typename T>
00242 void Vector<T>::push_back(const T& x) {
00243     if (size_ == capacity_) {
00244         reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00245     }
00246     allocator_.construct(&data_[size_], x);
00247     ++size_;
```

```
00248 }
00249
00250 template <typename T>
00251 void Vector<T>::pop_back() {
00252     if (size_ > 0) {
00253         allocator_.destroy(&data_[size_ - 1]);
00254         --size_;
00255     } else {
00256         throw std::out_of_range("Vector is empty");
00257     }
00258 }
00259
00260 template <typename T>
00261 size_t Vector<T>::getSize() const {
00262     return size_;
00263 }
00264
00265 template <typename T>
00266 size_t Vector<T>::capacity() const {
00267     return capacity_;
00268 }
00269
00270 template <typename T>
00271 bool Vector<T>::empty() const {
00272     return size_ == 0;
00273 }
00274
00275 template <typename T>
00276 void Vector<T>::clear() noexcept {
00277     for (size_t i = 0; i < size_; ++i) {
00278         allocator_.destroy(&data_[i]);
00279     }
00280     size_ = 0;
00281 }
00282
00283 template <typename T>
00284 T& Vector<T>::front() {
00285     if (size_ == 0) {
00286         throw std::out_of_range("Vector is empty");
00287     }
00288     return data_[0];
00289 }
00290
00291 template <typename T>
00292 const T& Vector<T>::front() const {
00293     if (size_ == 0) {
00294         throw std::out_of_range("Vector is empty");
00295     }
00296     return data_[0];
00297 }
00298
00299 template <typename T>
00300 T& Vector<T>::back() {
00301     if (size_ == 0) {
00302         throw std::out_of_range("Vector is empty");
00303     }
00304     return data_[size_ - 1];
00305 }
00306
00307 template <typename T>
00308 const T& Vector<T>::back() const {
00309     if (size_ == 0) {
00310         throw std::out_of_range("Vector is empty");
00311     }
00312     return data_[size_ - 1];
00313 }
00314
00315 template <typename T>
00316 void Vector<T>::swap(Vector& other) noexcept {
00317     std::swap(data_, other.data_);
00318     std::swap(size_, other.size_);
00319     std::swap(capacity_, other.capacity_);
00320 }
00321
00322 template <typename T>
00323 void Vector<T>::assign(size_t count, const T& x) {
00324     clear();
00325     if (capacity_ < count) {
00326         allocator_.deallocate(data_, capacity_);
00327         capacity_ = count;
00328         data_ = allocator_.allocate(capacity_);
00329     }
00330     for (size_t i = 0; i < count; ++i) {
00331         allocator_.construct(&data_[i], x);
00332     }
00333     size_ = count;
00334 }
```

```
00335
00336 template <typename T>
00337 template <typename InputIt>
00338 void Vector<T>::assign(InputIt first, InputIt last) {
00339     clear();
00340     size_t count = std::distance(first, last);
00341     if (capacity_ < count) {
00342         allocator_.deallocate(data_, capacity_);
00343         capacity_ = count;
00344         data_ = allocator_.allocate(capacity_);
00345     }
00346     size_t i = 0;
00347     for (auto it = first; it != last; ++it) {
00348         allocator_.construct(&data_[i++], *it);
00349     }
00350     size_ = count;
00351 }
00352
00353 template <typename T>
00354 T& Vector<T>::at(size_t index) {
00355     if (index >= size_) {
00356         throw std::out_of_range("Index out of range");
00357     }
00358     return data_[index];
00359 }
00360
00361 template <typename T>
00362 const T& Vector<T>::at(size_t index) const {
00363     if (index >= size_) {
00364         throw std::out_of_range("Index out of range");
00365     }
00366     return data_[index];
00367 }
00368
00369 template <typename T>
00370 T* Vector<T>::data() noexcept {
00371     return data_;
00372 }
00373
00374 template <typename T>
00375 const T* Vector<T>::data() const noexcept {
00376     return data_;
00377 }
00378
00379 template <typename T>
00380 std::allocator<T> Vector<T>::get_allocator() const {
00381     return allocator_;
00382 }
00383
00384 template <typename T>
00385 size_t Vector<T>::max_size() const {
00386     return std::numeric_limits<size_t>::max() / sizeof(T);
00387 }
00388
00389 template <typename T>
00390 typename Vector<T>::iterator Vector<T>::insert(const_iterator pos, const T& x) {
00391     size_t index = pos - cbegin();
00392     if (size_ == capacity_) {
00393         reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00394     }
00395     for (size_t i = size_; i > index; --i) {
00396         allocator_.construct(&data_[i], std::move(data_[i - 1]));
00397         allocator_.destroy(&data_[i - 1]);
00398     }
00399     allocator_.construct(&data_[index], x);
00400     ++size_;
00401     return data_ + index;
00402 }
00403
00404 template <typename T>
00405 typename Vector<T>::iterator Vector<T>::insert(const_iterator pos, T&& x) {
00406     size_t index = pos - cbegin();
00407     if (size_ == capacity_) {
00408         reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00409     }
00410     for (size_t i = size_; i > index; --i) {
00411         allocator_.construct(&data_[i], std::move(data_[i - 1]));
00412         allocator_.destroy(&data_[i - 1]);
00413     }
00414     allocator_.construct(&data_[index], std::move(x));
00415     ++size_;
00416     return data_ + index;
00417 }
00418
00419 template <typename T>
00420 typename Vector<T>::iterator Vector<T>::insert(const_iterator pos, size_t count, const T& x) {
00421     size_t index = pos - cbegin();
```

```
00422     if (size_ + count > capacity_) {
00423         reserve(size_ + count);
00424     }
00425     for (size_t i = size_; i > index; --i) {
00426         allocator_.construct(&data_[i + count - 1], std::move(data_[i - 1]));
00427         allocator_.destroy(&data_[i - 1]);
00428     }
00429     for (size_t i = 0; i < count; ++i) {
00430         allocator_.construct(&data_[index + i], x);
00431     }
00432     size_ += count;
00433     return data_ + index;
00434 }
00435
00436 template <typename T>
00437 template <class InputIt>
00438 typename Vector<T>::iterator Vector<T>::insert(const_iterator pos, InputIt first, InputIt last) {
00439     size_t index = pos - cbegin();
00440     size_t count = std::distance(first, last);
00441     if (size_ + count > capacity_) {
00442         reserve(size_ + count);
00443     }
00444     for (size_t i = size_; i > index; --i) {
00445         allocator_.construct(&data_[i + count - 1], std::move(data_[i - 1]));
00446         allocator_.destroy(&data_[i - 1]);
00447     }
00448     size_t i = 0;
00449     for (auto it = first; it != last; ++it, ++i) {
00450         allocator_.construct(&data_[index + i], *it);
00451     }
00452     size_ += count;
00453     return data_ + index;
00454 }
00455
00456 template <typename T>
00457 typename Vector<T>::iterator Vector<T>::insert(const_iterator pos, std::initializer_list<T> ilist) {
00458     return insert(pos, ilist.begin(), ilist.end());
00459 }
00460
00461 template <typename T>
00462 template <typename... Args>
00463 typename Vector<T>::iterator Vector<T>::emplace(const_iterator pos, Args&&... args) {
00464     size_t index = pos - cbegin();
00465     if (size_ == capacity_) {
00466         reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00467     }
00468     if (index < size_) {
00469         for (size_t i = size_; i > index; --i) {
00470             allocator_.construct(&data_[i], std::move(data_[i - 1]));
00471             allocator_.destroy(&data_[i - 1]);
00472         }
00473     }
00474     allocator_.construct(&data_[index], std::forward<Args>(args)...);
00475     ++size_;
00476     return data_ + index;
00477 }
00478
00479 template <typename T>
00480 template <typename... Args>
00481 void Vector<T>::emplace_back(Args&&... args) {
00482     if (size_ == capacity_) {
00483         reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00484     }
00485     allocator_.construct(&data_[size_], std::forward<Args>(args)...);
00486     ++size_;
00487 }
00488
00489 template <typename T>
00490 typename Vector<T>::iterator Vector<T>::erase(const_iterator pos) {
00491     if (pos < cbegin() || pos >= cend()) {
00492         throw std::out_of_range("Iterator out of range");
00493     }
00494
00495     size_t index = pos - cbegin();
00496     allocator_.destroy(&data_[index]);
00497     for (size_t i = index; i < size_ - 1; ++i) {
00498         allocator_.construct(&data_[i], std::move(data_[i + 1]));
00499         allocator_.destroy(&data_[i + 1]);
00500     }
00501     --size_;
00502     return data_ + index;
00503 }
00504
00505 template <typename T>
00506 typename Vector<T>::iterator Vector<T>::erase(const_iterator first, const_iterator last) {
00507     if (first < cbegin() || last > cend() || first >= last) {
00508         throw std::out_of_range("Iterator range out of range");
```

```
00509     }
00510
00511     size_t first_index = first - cbegin();
00512     size_t last_index = last - cbegin();
00513     size_t count = last_index - first_index;
00514
00515     for (size_t i = first_index; i < last_index; ++i) {
00516         allocator_.destroy(&data_[i]);
00517     }
00518
00519     for (size_t i = last_index; i < size_; ++i) {
00520         allocator_.construct(&data_[i - count], std::move(data_[i]));
00521         allocator_.destroy(&data_[i]);
00522     }
00523
00524     size_ -= count;
00525     return data_ + first_index;
00526 }
00527
00528 #endif
```

Index