My Project

Generated by Doxygen 1.10.0

Chapter 1

README

//3 užduotis//

Sukūriau Vector klasę, kuri yra pilnavertė alternatyva std::vector konteineriui, t.y. visi fonkciunalumai (Member types, Member functions, Non-member functions). Naudojausi: https://en.cppreference.com/w/cpp/container/vector

1.1 Įsitikinti, ar Vector konteineris veikia (funkcionalumo prasme) lygiai taip, kaip std::vector

Pasirinkau 5 funkcijas ir stebėjau, ar gauti rezultatai su Vector konteineriu atitinka std::vector rezultatus. Naudojausi: https://en.cppreference.com/w/cpp/container/vector

MAZ_SIZE():

```
Vector<char> p;
   Vector<long> q;

   locale("C"));
   cout « uppercase
        « "p.max_size() = " « std::dec « p.max_size() « " = 0x"
        « std::hex « p.max_size() « '\n'
        « "q.max_size() = " « std::dec « q.max_size() « " = 0x"
        « std::hex « q.max_size() « '\n';
```

(TEISINGAS) OUTPUT:

```
p.max_size() = 9,223,372,036,854,775,807 = 0x7,FFF,FFF,FFF,FFF,FFF
q.max_size() = 1,152,921,504,606,846,975 = 0xFFF,FFF,FFF,FFF,FFF
```

EMPTY():

```
cout « boolalpha;
vector<int> numbers;
cout « "Initially, numbers.empty(): " « numbers.empty() « '\n';

numbers.push_back(42);
cout « "After adding elements, numbers.empty(): " « numbers.empty() « '\n';
```

(TEISINGAS) OUTPUT:

```
Initially, numbers.empty(): true
After adding elements, numbers.empty(): false
```

PUSH_BACK():

```
Vector<string> letters;

letters.push_back("abc");
string s{"def"};
letters.push_back(move(s));

std::cout « "std::vector letters holds: ";
for (auto&& e : letters)
    cout « quoted(e) « ' ';

cout « "\nMoved-from string s holds: " « quoted(s) « '\n';
```

(TEISINGAS) OUTPUT:

```
std::vector letters holds: "abc" "def"
Moved-from string s holds: ""
```

OPERATOR = :

```
Vector<int> foo (3,0);
Vector<int> bar (5,0);

bar = foo;
foo = Vector<int>();

cout « "Size of foo: " « int(foo.size()) « '\n';
cout « "Size of bar: " « int(bar.size()) « '\n';
```

(TEISINGAS) OUTPUT:

```
Size of foo: 0
Size of bar: 3
```

SWAP():

```
  Vector<int> alice{1, 2, 3};
Vector<int> bob{7, 8, 9, 10};

auto print = [](const int& n) { cout « ' ' « n; };

// Print state before swap
cout « "Alice:";
for_each(alice.begin(), alice.end(), print);
cout « "\n" "Bob  :";
for_each(bob.begin(), bob.end(), print);
cout « '\n';

cout « "-- SWAP\n";
swap(alice, bob);

// Print state after swap
cout « "Alice:";
for_each(alice.begin(), alice.end(), print);
cout « "\n" "Bob  :";
for_each(bob.begin(), bob.end(), print);
cout « '\n';
```

(TEISINGAS) OUTPUT:

```
Alice: 1 2 3
Bob  : 7 8 9 10
-- SWAP
Alice: 7 8 9 10
Bob  : 1 2 3
```

Visi rezultatai sutampa.

1.2 Efektyvumo/spartos analizė std::vector vs Vector

Skaičiuoju, kiek vidutiniškai laiko užtrunka užpildyti tuščius vektorius: 10 000, 100 000, 1 000 000, 10 000 000 ir 100 000 000 int elementų naudojant push_back() funkciją.

	10 000	100 000	1 000 000	10 000 000	100 000 000	iš viso
std::vector	0.0008438s	0.0059773s	0.0387429s	0.184515s	1.4759s	1,705979s
Vector	0.000569s	0.0061737s	0.0113419s	0.118442s	1.14663s	1,2831566s

Rezultatai:

Mažiausia laiko uztrunka su Vector.

1.3 Kiek kartų įvyksta konteinerių (Vector ir std::vector) atminties perskirstymai užpildant 100000000 elementų.

Perskirstymas įvyksta tada, kai yra patenkinama sąlyga: capacity() == size(), t.y. kai nelieka vietos capacity()`naujiems elementams.

Naudotas kodas:

```
auto start = std::chrono::steady_clock::now();
   unsigned int sz = 100000000;  // 100000, 1000000, 10000000, 100000000
   int std_vector = 0;
   std::vector<int> v1;
   for (int i = 1; i <= sz; ++i){
       v1.push_back(i);
       if (v1.capacity() == v1.size()){
           std_vector++;
       }
   }
   auto end = std::chrono::steady_clock::now();

   auto start_2 = std::chrono::steady_clock::now();
   int Vector_2 = 0;
   Vector<int> v2;
   for (int i = 1; i <= sz; ++i){
       v2.push_back(i);
        if (v2.capacity() == v2.size()){
           Vector_2++;
       }
   }
   auto end_2 = std::chrono::steady_clock::now();

   // Laukiam, kol procesorius laiko
   std::chrono::duration<double> elapsed_seconds = end - start;
   std::chrono::duration<double> elapsed_seconds_2 = end_2 - start_2;

   cout « "std::vector: " « elapsed_seconds.count() « "s\n";
   cout « "std::vector atmintis perskirstyta: " « std_vector « " kartus" « endl;
   cout « "Vector: " « elapsed_seconds_2.count() « "s\n";
   cout « "Vector atmintis perskirstyta: " « Vector_2 « " kartus" « endl;
```

Rezultatai:

```
std::vector atmintis perskirstyta: 27 kartus
Vector atmintis perskirstyta: 26 kartus
```

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 duomenys Struct Reference

Public Member Functions

- duomenys ()

Public Attributes

- string vard
- string pav
- int * nd
- int egzaminas
- int nd_kiekis
- double gal_vid
- double gal_bal
- double gal_med

5.1.1 Constructor & Destructor Documentation

5.1.1.1 duomenys()

```
duomenys::duomenys ( )  [inline]
```

5.1.2 Member Data Documentation

5.1.2.1 egzaminas

```
int duomenys::egzaminas
```

5.1.2.2 gal_bal

```
double duomenys::gal_bal
```

5.1.2.3 gal_med

```
double duomenys::gal_med
```

5.1.2.4 gal_vid

```
double duomenys::gal_vid
```

5.1.2.5 nd

```
int* duomenys::nd
```

5.1.2.6 nd_kiekis

```
int duomenys::nd_kiekis
```

5.1.2.7 pav

```
string duomenys::pav
```

5.1.2.8 vard

```
string duomenys::vard
```

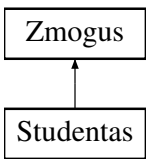
The documentation for this struct was generated from the following file:

- C:/Darbai/2_OP/2_OP/3_OP/masyvai.cpp

5.2 Studentas Class Reference

```
#include <studentas.h>
```

Inheritance diagram for Studentas:



Public Member Functions

- bool operator< (const Studentas &other) const
- Studentas ()
- Studentas (const std::string &vardas, const std::string &pav, int egzaminas, const std::vector< int > &nd, int nd_kiekis, double gal_vid, double gal_med)
- ~Studentas ()
- Studentas (const Studentas &other)
- Studentas & operator= (const Studentas &other)
- Studentas (Studentas &&other) noexcept
- Studentas & operator= (Studentas &&other) noexcept
- const std::string & getVardas () const override
- const std::string & getPavarde () const override
- const std::vector< int > & getNd () const
- int getEgzaminas () const
- double getGalutinisVid () const
- double getGalutinisBal () const
- double getGalutineMed () const
- int getNdKiekis () const
- void setVardas (const std::string &vardas) override
- void setPavarde (const std::string &pavarde) override
- void setNd (const std::vector< int > &nd)
- void setEgzaminas (int egzaminas)
- void setGalutinisVid (double gal_vid)
- void setGalutinisBal (double gal_bal)
- void setGalutineMed (double gal_med)
- void setNdKiekis (int nd_kiekis)
- void addnd (int nd)
- void skaiciuotiGalutiniBal ()

Private Attributes

- std::vector< int > nd_
- int egzaminas_
- int nd_kiekis_
- double gal_vid_
- double gal_bal_
- double gal_med_

Friends

- std::istream & operator>> (std::istream &in, Studentas &studentas)
- std::ostream & operator<< (std::ostream &out, const Studentas &studentas)

Additional Inherited Members

Protected Member Functions inherited from Zmogus

- Zmogus ()=default
- Zmogus (const std::string &vardas, const std::string &pav)
- virtual ~Zmogus ()

Protected Attributes inherited from Zmogus

- std::string vardas_
- std::string pav_

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Studentas() [1/4]

```
Studentas::Studentas ( )  [inline]
```

5.2.1.2 Studentas() [2/4]

```
Studentas::Studentas (
            const std::string & vardas,
            const std::string & pav,
            int egzaminas,
            const std::vector< int > & nd,
            int nd_kiekis,
            double gal_vid,
            double gal_med )  [inline]
```

5.2.1.3 ~Studentas()

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

5.2.1.4 Studentas() [3/4]

```
Studentas::Studentas (
            const Studentas & other )  [inline]
```

5.2.1.5 Studentas() [4/4]

```
Studentas::Studentas (
            Studentas && other )  [inline], [noexcept]
```

5.2.2 Member Function Documentation

5.2.2.1 addnd()

```
void Studentas::addnd (
            int nd )  [inline]
```

5.2.2.2 getEgzaminas()

```
int Studentas::getEgzaminas ( ) const  [inline]
```

5.2.2.3 getGalutineMed()

```
double Studentas::getGalutineMed ( ) const  [inline]
```

5.2.2.4 getGalutinisBal()

```
double Studentas::getGalutinisBal ( ) const  [inline]
```

5.2.2.5 getGalutinisVid()

```
double Studentas::getGalutinisVid ( ) const  [inline]
```

5.2.2.6 getNd()

```
const std::vector< int > & Studentas::getNd ( ) const  [inline]
```

5.2.2.7 getNdKiekis()

```
int Studentas::getNdKiekis ( ) const  [inline]
```

5.2.2.8 getPavarde()

```
const std::string & Studentas::getPavarde ( ) const  [inline], [override], [virtual]
```

Reimplemented from Zmogus.

5.2.2.9 getVardas()

```
const std::string & Studentas::getVardas ( ) const  [inline], [override], [virtual]
```

Reimplemented from Zmogus.

5.2.2.10 operator<()

```
bool Studentas::operator< (
            const Studentas & other ) const  [inline]
```

5.2.2.11 operator=() [1/2]

```
Studentas & Studentas::operator= (
            const Studentas & other )  [inline]
```

5.2.2.12 operator=() [2/2]

```
Studentas & Studentas::operator= (
            Studentas && other )  [inline], [noexcept]
```

5.2.2.13 setEgzaminas()

```
void Studentas::setEgzaminas (
            int egzaminas )  [inline]
```

5.2.2.14 setGalutineMed()

```
void Studentas::setGalutineMed (
            double gal_med )  [inline]
```

5.2.2.15 setGalutinisBal()

```
void Studentas::setGalutinisBal (
            double gal_bal )  [inline]
```

5.2.2.16 setGalutinisVid()

```
void Studentas::setGalutinisVid (
            double gal_vid )  [inline]
```

5.2.2.17 setNd()

```
void Studentas::setNd (
            const std::vector< int > & nd )  [inline]
```

5.2.2.18 setNdKiekis()

```
void Studentas::setNdKiekis (
            int nd_kiekis )  [inline]
```

5.2.2.19 setPavarde()

```
void Studentas::setPavarde (
            const std::string & pavarde )  [inline], [override], [virtual]
```

Reimplemented from Zmogus.

5.2.2.20 setVardas()

```
void Studentas::setVardas (
            const std::string & vardas )  [inline], [override], [virtual]
```

Reimplemented from Zmogus.

5.2.2.21 skaiciuotiGalutiniBal()

```
void Studentas::skaiciuotiGalutiniBal ( )  [inline]
```

5.2.3 Friends And Related Symbol Documentation

5.2.3.1 operator<<

```
std::ostream & operator<< (
            std::ostream & out,
            const Studentas & studentas )  [friend]
```

5.2.3.2 operator>>

```
std::istream & operator>> (
            std::istream & in,
            Studentas & studentas )  [friend]
```

5.2.4 Member Data Documentation

5.2.4.1 egzaminas_

```
int Studentas::egzaminas_  [private]
```

5.2.4.2 gal_bal_

```
double Studentas::gal_bal_  [private]
```

5.2.4.3 gal_med_

```
double Studentas::gal_med_  [private]
```

5.2.4.4 gal_vid_

```
double Studentas::gal_vid_  [private]
```

5.2.4.5 nd_

```
std::vector<int> Studentas::nd_  [private]
```

5.2.4.6 nd_kiekis_

```
int Studentas::nd_kiekis_  [private]
```

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

- C:/Darbai/2_OP/2_OP/3_OP/studentas.h

5.3 Vector< T, Allocator > Class Template Reference

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

Public Types

- using value_type = T
- using allocator_type = Allocator
- using size_type = typename std::allocator_traits<Allocator>::size_type
- using difference_type = typename std::allocator_traits<Allocator>::difference_type
- using reference = value_type&
- using const_reference = const value_type&
- using pointer = typename std::allocator_traits<Allocator>::pointer
- using const_pointer = typename std::allocator_traits<Allocator>::const_pointer
- 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 (size_type count, const T &value=T(), const Allocator &alloc=Allocator())
- Vector (std::initializer_list< T > ilist, const Allocator &alloc=Allocator())
- ~Vector ()
- Vector (const Vector &other)
- Vector (Vector &&other) noexcept
- Vector & operator= (const Vector &other)
- Vector & operator= (Vector &&other) noexcept
- void assign (size_type count, const T &value)
- void assign (std::initializer_list< T > ilist)
- template<typename InputIt >
  void assign (InputIt first, InputIt last)
- reference at (size_type pos)
- const_reference at (size_type pos) const
- reference operator[ ] (size_type pos)
- const_reference operator[ ] (size_type pos) const

- reference front ()
- const_reference front () const
- reference back ()
- const_reference back () const
- T * data () noexcept
- const T * data () const noexcept
- iterator begin () noexcept
- const_iterator begin () const noexcept
- const_iterator cbegin () const noexcept
- iterator end () noexcept
- const_iterator end () const noexcept
- const_iterator cend () const noexcept
- reverse_iterator rbegin () noexcept
- const_reverse_iterator rbegin () const noexcept
- const_reverse_iterator crbegin () const noexcept
- reverse_iterator rend () noexcept
- const_reverse_iterator rend () const noexcept
- const_reverse_iterator crend () const noexcept
- bool empty () const noexcept
- size_type size () const noexcept
- size_type max_size () const noexcept
- void reserve (size_type new_cap)
- size_type capacity () const noexcept
- void shrink_to_fit ()
- void clear () noexcept
- iterator insert (const_iterator pos, const T &value)
- iterator insert (const_iterator pos, size_type count, const T &value)
- template<typename InputIt >
  iterator insert (const_iterator pos, InputIt first, InputIt last)
- iterator erase (const_iterator pos)
- iterator erase (const_iterator first, const_iterator last)
- void push_back (const T &value)
- void push_back (T &&value)
- template<typename... Args>
  reference emplace_back (Args &&... args)
- void pop_back ()
- void resize (size_type count)
- void resize (size_type count, const T &value)
- void swap (Vector &other) noexcept

Private Attributes

- T * m_data
- std::size_t m_size
- std::size_t m_capacity
- Allocator m_alloc

Friends

- bool operator== (const Vector &lhs, const Vector &rhs)
- bool operator!= (const Vector &lhs, const Vector &rhs)
- bool operator< (const Vector &lhs, const Vector &rhs)
- bool operator<= (const Vector &lhs, const Vector &rhs)
- bool operator> (const Vector &lhs, const Vector &rhs)
- bool operator>= (const Vector &lhs, const Vector &rhs)

5.3.1 Member Typedef Documentation

5.3.1.1 allocator_type

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::allocator_type = Allocator
```

5.3.1.2 const_iterator

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::const_iterator = const T*
```

5.3.1.3 const_pointer

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::const_pointer = typename std::allocator_traits<Allocator>↩
::const_pointer
```

5.3.1.4 const_reference

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::const_reference = const value_type&
```

5.3.1.5 const_reverse_iterator

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

5.3.1.6 difference_type

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::difference_type = typename std::allocator_traits<Allocator>↩
::difference_type
```

5.3.1.7 iterator

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::iterator = T*
```

5.3.1.8 pointer

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::pointer = typename std::allocator_traits<Allocator>::pointer
```

5.3.1.9 reference

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::reference = value_type&
```

5.3.1.10 reverse_iterator

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

5.3.1.11 size_type

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::size_type = typename std::allocator_traits<Allocator>::size_↩
type
```

5.3.1.12 value_type

```
template<typename T , typename Allocator = std::allocator<T>>
using Vector< T, Allocator >::value_type = T
```

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Vector() [1/5]

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

5.3.2.2 Vector() [2/5]

```
template<typename T , typename Allocator = std::allocator<T>>
Vector< T, Allocator >::Vector (
            size_type count,
            const T & value = T(),
            const Allocator & alloc = Allocator() )  [inline], [explicit]
```

5.3.2.3 Vector() [3/5]

```
template<typename T , typename Allocator = std::allocator<T>>
Vector< T, Allocator >::Vector (
            std::initializer_list< T > ilist,
            const Allocator & alloc = Allocator() )  [inline], [explicit]
```

5.3.2.4 ~Vector()

```
template<typename T , typename Allocator = std::allocator<T>>
Vector< T, Allocator >::~Vector ( )  [inline]
```

5.3.2.5 Vector() [4/5]

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

5.3.2.6 Vector() [5/5]

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

5.3.3 Member Function Documentation

5.3.3.1 assign() [1/3]

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

5.3.3.2 assign() [2/3]

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

5.3.3.3 assign() [3/3]

```
template<typename T , typename Allocator = std::allocator<T>>
void Vector< T, Allocator >::assign (
            std::initializer_list< T > ilist )  [inline]
```

5.3.3.4 at() [1/2]

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

5.3.3.5 at() [2/2]

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

5.3.3.6 back() [1/2]

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

5.3.3.7 back() [2/2]

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

5.3.3.8 begin() [1/2]

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

5.3.3.9 begin() [2/2]

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

5.3.3.10 capacity()

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

5.3.3.11 cbegin()

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

5.3.3.12 cend()

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

5.3.3.13 clear()

```
template<typename T , typename Allocator = std::allocator<T>>
void Vector< T, Allocator >::clear ( )  [inline], [noexcept]
```

5.3.3.14 crbegin()

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

5.3.3.15 crend()

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

5.3.3.16 data() [1/2]

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

5.3.3.17 data() [2/2]

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

5.3.3.18 emplace_back()

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

5.3.3.19 empty()

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

5.3.3.20 end() [1/2]

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

5.3.3.21 end() [2/2]

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

5.3.3.22 erase() [1/2]

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

5.3.3.23 erase() [2/2]

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

5.3.3.24 front() [1/2]

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

5.3.3.25 front() [2/2]

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

5.3.3.26 insert() [1/3]

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

5.3.3.27 insert() [2/3]

```
template<typename T , typename Allocator = std::allocator<T>>
template<typename InputIt >
iterator Vector< T, Allocator >::insert (
            const_iterator pos,
            InputIt first,
            InputIt last )  [inline]
```

5.3.3.28 insert() [3/3]

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

5.3.3.29 max_size()

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

5.3.3.30 operator=() [1/2]

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

5.3.3.31 operator=() [2/2]

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

5.3.3.32 operator[]() [1/2]

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

5.3.3.33 operator[]() [2/2]

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

5.3.3.34 pop_back()

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

5.3.3.35 push_back() [1/2]

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

5.3.3.36 push_back() [2/2]

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

5.3.3.37 rbegin() [1/2]

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

5.3.3.38 rbegin() [2/2]

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

5.3.3.39 rend() [1/2]

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

5.3.3.40 rend() [2/2]

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

5.3.3.41 reserve()

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

5.3.3.42 resize() [1/2]

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

5.3.3.43 resize() [2/2]

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

5.3.3.44 shrink_to_fit()

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

5.3.3.45 size()

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

5.3.3.46 swap()

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

5.3.4 Friends And Related Symbol Documentation

5.3.4.1 operator"!=

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator!= (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.4.2 operator<

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator< (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.4.3 operator<=

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator<= (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.4.4 operator==

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator== (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.4.5 operator>

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator> (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.4.6 operator>=

```
template<typename T , typename Allocator = std::allocator<T>>
bool operator>= (
            const Vector< T, Allocator > & lhs,
            const Vector< T, Allocator > & rhs )  [friend]
```

5.3.5 Member Data Documentation

5.3.5.1 m_alloc

```
template<typename T , typename Allocator = std::allocator<T>>
Allocator Vector< T, Allocator >::m_alloc  [private]
```

5.3.5.2 m_capacity

```
template<typename T , typename Allocator = std::allocator<T>>
std::size_t Vector< T, Allocator >::m_capacity  [private]
```

5.3.5.3 m_data

```
template<typename T , typename Allocator = std::allocator<T>>
T* Vector< T, Allocator >::m_data  [private]
```

5.3.5.4 m_size

```
template<typename T , typename Allocator = std::allocator<T>>
std::size_t Vector< T, Allocator >::m_size  [private]
```

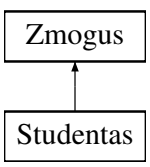
The documentation for this class was generated from the following file:

- C:/Darbai/2_OP/2_OP/3_OP/vector.h

5.4 Zmogus Class Reference

```
#include <studentas.h>
```

Inheritance diagram for Zmogus:



Public Member Functions

- virtual const std::string & getVardas () const
- virtual const std::string & getPavarde () const
- virtual void setVardas (const std::string &vardas)
- virtual void setPavarde (const std::string &pavarde)

Protected Member Functions

- Zmogus ()=default
- Zmogus (const std::string &vardas, const std::string &pav)
- virtual ~Zmogus ()

Protected Attributes

- std::string vardas_
- std::string pav_

5.4.1 Constructor & Destructor Documentation

5.4.1.1 Zmogus() [1/2]

```
Zmogus::Zmogus ( )  [protected], [default]
```

5.4.1.2 Zmogus() [2/2]

```
Zmogus::Zmogus (
            const std::string & vardas,
            const std::string & pav )  [inline], [protected]
```

5.4.1.3 ~Zmogus()

```
virtual Zmogus::~Zmogus ( )  [inline], [protected], [virtual]
```

5.4.2 Member Function Documentation

5.4.2.1 getPavarde()

```
virtual const std::string & Zmogus::getPavarde ( ) const  [inline], [virtual]
```

Reimplemented in Studentas.

5.4.2.2 getVardas()

```
virtual const std::string & Zmogus::getVardas ( ) const  [inline], [virtual]
```

Reimplemented in Studentas.

5.4.2.3 setPavarde()

```
virtual void Zmogus::setPavarde (
            const std::string & pavarde )  [inline], [virtual]
```

Reimplemented in Studentas.

5.4.2.4 setVardas()

```
virtual void Zmogus::setVardas (
            const std::string & vardas )  [inline], [virtual]
```

Reimplemented in Studentas.

5.4.3 Member Data Documentation

5.4.3.1 pav_

```
std::string Zmogus::pav_  [protected]
```

5.4.3.2 vardas_

```
std::string Zmogus::vardas_  [protected]
```

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

- C:/Darbai/2_OP/2_OP/3_OP/studentas.h

Chapter 6

File Documentation

6.1 C:/Darbai/2_OP/2_OP/3_OP/functions.cpp File Reference

```
#include "functions.h"
#include "vector.h"
```

Functions

- void func_input_hands ()
- void func_generate_numbers ()
- void func_generate_names ()
- void func_input_file ()
- void generate_new_file ()
- void use_existing_file ()
- void read_list ()
- void read_deque ()
- void use_existing_file_2 ()
- void read_list_2 ()
- void read_deque_2 ()
- void use_existing_file_3 ()
- void read_list_3 ()
- void read_deque_3 ()
- void func_generate ()
- void func_tests ()
- void func_input_output ()
- void func_vector ()
- void func_time ()

6.1.1 Function Documentation

6.1.1.1 func_generate()

```
void func_generate ( )
```

6.1.1.2 func_generate_names()

```
void func_generate_names ( )
```

6.1.1.3 func_generate_numbers()

```
void func_generate_numbers ( )
```

6.1.1.4 func_input_file()

```
void func_input_file ( )
```

6.1.1.5 func_input_hands()

```
void func_input_hands ( )
```

6.1.1.6 func_input_output()

```
void func_input_output ( )
```

6.1.1.7 func_tests()

```
void func_tests ( )
```

6.1.1.8 func_time()

```
void func_time ( )
```

6.1.1.9 func_vector()

```
void func_vector ( )
```

6.1.1.10 generate_new_file()

```
void generate_new_file ( )
```

6.1.1.11 read_deque()

```
void read_deque ( )
```

6.1.1.12 read_deque_2()

```
void read_deque_2 ( )
```

6.1.1.13 read_deque_3()

```
void read_deque_3 ( )
```

6.1.1.14 read_list()

```
void read_list ( )
```

6.1.1.15 read_list_2()

```
void read_list_2 ( )
```

6.1.1.16 read_list_3()

```
void read_list_3 ( )
```

6.1.1.17 use_existing_file()

```
void use_existing_file ( )
```

6.1.1.18 use_existing_file_2()

```
void use_existing_file_2 ( )
```

6.1.1.19 use_existing_file_3()

```
void use_existing_file_3 ( )
```

6.2 C:/Darbai/2_OP/2_OP/3_OP/functions.h File Reference

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <iomanip>
#include <ctime>
#include <fstream>
#include <istream>
#include <sstream>
#include <stdexcept>
#include <bits/stdc++.h>
#include <chrono>
#include <list>
#include <deque>
#include "studentas.h"
#include <cassert>
```

Functions

- void func_input_hands ()
- void func_generate_numbers ()
- void func_generate_names ()
- void func_input_file ()
- void func_generate ()
- void generate_new_file ()
- void use_existing_file ()
- void read_list ()
- void read_deque ()
- void use_existing_file_2 ()
- void read_list_2 ()
- void read_deque_2 ()
- void use_existing_file_3 ()
- void read_list_3 ()
- void read_deque_3 ()
- void func_tests ()
- void func_input_output ()
- void func_vector ()
- void func_time ()

6.2.1 Function Documentation

6.2.1.1 func_generate()

```
void func_generate ( )
```

6.2.1.2 func_generate_names()

```
void func_generate_names ( )
```

6.2.1.3 func_generate_numbers()

```
void func_generate_numbers ( )
```

6.2.1.4 func_input_file()

```
void func_input_file ( )
```

6.2.1.5 func_input_hands()

```
void func_input_hands ( )
```

6.2.1.6 func_input_output()

```
void func_input_output ( )
```

6.2.1.7 func_tests()

```
void func_tests ( )
```

6.2.1.8 func_time()

```
void func_time ( )
```

6.2.1.9 func_vector()

```
void func_vector ( )
```

6.2.1.10 generate_new_file()

```
void generate_new_file ( )
```

6.2.1.11 read_deque()

```
void read_deque ( )
```

6.2.1.12 read_deque_2()

```
void read_deque_2 ( )
```

6.2.1.13 read_deque_3()

```
void read_deque_3 ( )
```

6.2.1.14 read_list()

```
void read_list ( )
```

6.2.1.15 read_list_2()

```
void read_list_2 ( )
```

6.2.1.16 read_list_3()

```
void read_list_3 ( )
```

6.2.1.17 use_existing_file()

```
void use_existing_file ( )
```

6.2.1.18 use_existing_file_2()

```
void use_existing_file_2 ( )
```

6.2.1.19 use_existing_file_3()

```
void use_existing_file_3 ( )
```

6.3 functions.h

Go to the documentation of this file.

```
00001 #ifndef FUNCTIONS_H
00002 #define FUNCTIONS_H
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <algorithm>
00007 #include <numeric>
00008 #include <iomanip>
00009 #include <ctime>
00010 #include <fstream>
00011 #include <istream>
00012 #include <sstream>
00013 #include <numeric>
00014 #include <stdexcept>
00015 #include <bits/stdc++.h>
00016 #include <chrono>
00017 #include <list>
00018 #include <deque>
00019 #include "studentas.h"
00020 #include <cassert>
00021
00022 using namespace std;
00023
00024 void func_input_hands();
00025 void func_generate_numbers();
00026 void func_generate_names();
00027 void func_input_file();
00028 void func_generate();
00029 void generate_new_file();
00030 //3 STRATEGIJA
00031 void use_existing_file();
00032 void read_list();
00033 void read_deque();
00034 //2 STRATEGIJA
00035 void use_existing_file_2();
00036 void read_list_2();
00037 void read_deque_2();
00038 //3 STRATEGIJA
00039 void use_existing_file_3();
00040 void read_list_3();
00041 void read_deque_3();
00042
00043 void func_tests();
00044 void func_input_output();
00045
00046 void func_vector();
00047 void func_time();
00048 #endif /* FUNCTIONS_H */
```

6.4 C:/Darbai/2_OP/2_OP/3_OP/masyvai.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <random>
#include <ctime>
```

Classes

- struct duomenys

Functions

- void func_input_hands ()
- void func_generate_numbers ()
- void func_generate_names ()
- int main ()

Variables

- const int MAX_ND = 100

6.4.1 Function Documentation

6.4.1.1 func_generate_names()

```
void func_generate_names ( )
```

6.4.1.2 func_generate_numbers()

```
void func_generate_numbers ( )
```

6.4.1.3 func_input_hands()

```
void func_input_hands ( )
```

6.4.1.4 main()

```
int main ( )
```

6.4.2 Variable Documentation

6.4.2.1 MAX_ND

```
const int MAX_ND = 100
```

6.5 C:/Darbai/2_OP/2_OP/3_OP/README.md File Reference

6.6 C:/Darbai/2_OP/2_OP/3_OP/studentas.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
```

Classes

- class Zmogus
- class Studentas

Variables

- const int MAX_ND = 100

6.6.1 Variable Documentation

6.6.1.1 MAX_ND

```
const int MAX_ND = 100
```

6.7 studentas.h

Go to the documentation of this file.

```
00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
00003
00004 #include <vector>
00005 #include <string>
00006 #include <iostream>
00007
00008 const int MAX_ND = 100;
00009
00010 class Zmogus {
00011 protected:
00012     std::string vardas_, pav_; // Privatūs nariai, kurie saugo žmogaus vardą ir pavardę
00013
00014     Zmogus() = default;  // Standartinis konstruktorius, inicializuoja vardą ir pavardę
00015     // Konstruktorius, kuris inicializuoja vardą ir pavardę pagal pateiktus parametrus
00016     Zmogus(const std::string& vardas, const std::string& pav)
00017     : vardas_(vardas), pav_(pav) {}
00018     // Virtualus destruktorius, leidžiantis paveldėtoms klasėms tvarkyti atminties išlaisvinimą
00019     virtual ~Zmogus() {}
```

```
00020
00021 public:
00022     // Virtualus get metodai, gražinantys vardą arba pavarde
00023     virtual const std::string& getVardas() const { return vardas_; }
00024     virtual const std::string& getPavarde() const { return pav_; }
00025     // Virtualus set metodai, nustatantys varda arba pavarde
00026     virtual void setVardas(const std::string& vardas) { vardas_ = vardas; }
00027     virtual void setPavarde(const std::string& pavarde) { pav_ = pavarde; }
00028 };
00029
00030 class Studentas : public Zmogus {
00031 private:
00032     std::vector<int> nd_;
00033     int egzaminas_, nd_kiekis_;
00034     double gal_vid_, gal_bal_, gal_med_;
00035
00036 public:
00037     bool operator<(const Studentas& other) const {
00038         return gal_vid_ < other.gal_vid_;
00039     }
00040
00041     Studentas() : egzaminas_(0), nd_kiekis_(0), gal_vid_(0), gal_bal_(0), gal_med_(0) {}
00042
00043     Studentas(const std::string& vardas, const std::string& pav, int egzaminas, const
      std::vector<int>& nd, int nd_kiekis, double gal_vid, double gal_med)
00044         : Zmogus(vardas, pav), nd_(nd), egzaminas_(egzaminas), nd_kiekis_(nd_kiekis),
00045         gal_vid_(gal_vid), gal_med_(gal_med) {}
00046
00047     ~Studentas() {}
00048
00049     // COPY KONSTRUKTORIUS
00050     Studentas(const Studentas& other)
00051         : Zmogus(other.getVardas(), other.getPavarde()), nd_(other.nd_), egzaminas_(other.egzaminas_),
00052         nd_kiekis_(other.nd_kiekis_), gal_vid_(other.gal_vid_), gal_bal_(other.gal_bal_),
      gal_med_(other.gal_med_) {}
00053
00054     // COPY PRISKYRIMO OPERATORIUS
00055     Studentas& operator=(const Studentas& other)
00056     {
00057         if (this != &other) {
00058             setVardas(other.getVardas());
00059             setPavarde(other.getPavarde());
00060             nd_ = other.nd_;
00061             egzaminas_ = other.egzaminas_;
00062             nd_kiekis_ = other.nd_kiekis_;
00063             gal_vid_ = other.gal_vid_;
00064             gal_bal_ = other.gal_bal_;
00065             gal_med_ = other.gal_med_;
00066         }
00067         return *this;
00068     }
00069
00070     // MOVE KONSTRUKTORIUS
00071     Studentas(Studentas&& other) noexcept
00072         : Zmogus(std::move(other.vardas_), std::move(other.pav_)), nd_(std::move(other.nd_)),
00073         egzaminas_(std::move(other.egzaminas_)), nd_kiekis_(std::move(other.nd_kiekis_)),
00074         gal_vid_(std::move(other.gal_vid_)), gal_bal_(std::move(other.gal_bal_)),
      gal_med_(std::move(other.gal_med_))
00075     {
00076         other.vardas_ = "";
00077         other.pav_ = "";
00078         other.egzaminas_ = 0;
00079         other.nd_kiekis_ = 0;
00080         other.gal_vid_ = 0;
00081         other.gal_bal_ = 0;
00082         other.gal_med_ = 0;
00083     }
00084
00085     // MOVE PRISKYRIMO OPERATORIUS
00086     Studentas& operator=(Studentas&& other) noexcept
00087     {
00088         if (this != &other) {
00089             setVardas(std::move(other.getVardas()));
00090             setPavarde(std::move(other.getPavarde()));
00091             nd_ = std::move(other.nd_);
00092             egzaminas_ = std::move(other.egzaminas_);
00093             nd_kiekis_ = std::move(other.nd_kiekis_);
00094             gal_vid_ = std::move(other.gal_vid_);
00095             gal_bal_ = std::move(other.gal_bal_);
00096             gal_med_ = std::move(other.gal_med_);
00097
00098             other.vardas_ = "";
00099             other.pav_ = "";
00100             other.egzaminas_ = 0;
00101             other.nd_kiekis_ = 0;
00102             other.gal_vid_ = 0;
00103             other.gal_bal_ = 0;
```

```
00104             other.gal_med_ = 0;
00105         }
00106         return *this;
00107     }
00108
00109     const std::string& getVardas() const override { return Zmogus::getVardas(); }
00110     const std::string& getPavarde() const override { return Zmogus::getPavarde(); }
00111     const std::vector<int>& getNd() const { return nd_; }
00112     int getEgzaminas() const { return egzaminas_; }
00113     double getGalutinisVid() const { return gal_vid_; }
00114     double getGalutinisBal() const { return gal_bal_; }
00115     double getGalutineMed() const { return gal_med_; }
00116     int getNdKiekis() const { return nd_kiekis_; }
00117
00118     void setVardas(const std::string& vardas) override { Zmogus::setVardas(vardas); }
00119     void setPavarde(const std::string& pavarde) override { Zmogus::setPavarde(pavarde); }
00120     void setNd(const std::vector<int>& nd) { nd_ = nd; nd_kiekis_ = nd.size();
      skaiciuotiGalutiniBal(); }
00121     void setEgzaminas(int egzaminas) { egzaminas_ = egzaminas; }
00122     void setGalutinisVid(double gal_vid) { gal_vid_ = gal_vid; }
00123     void setGalutinisBal(double gal_bal) { gal_bal_ = gal_bal; }
00124     void setGalutineMed(double gal_med) { gal_med_ = gal_med; }
00125     void setNdKiekis(int nd_kiekis) { nd_kiekis_ = nd_kiekis; }
00126
00127     void addnd(int nd) { nd_.push_back(nd); nd_kiekis_++; skaiciuotiGalutiniBal(); }
00128
00129     void skaiciuotiGalutiniBal() {
00130         if (nd_.empty()) {
00131             gal_bal_ = egzaminas_;
00132             return;
00133         }
00134         double suma = 0;
00135         for (int pazymys : nd_) {
00136             suma += pazymys;
00137         }
00138         gal_bal_ = 0.4 * (suma / nd_.size()) + 0.6 * egzaminas_;
00139     }
00140
00141     // Input
00142     friend std::istream& operator»(std::istream& in, Studentas& studentas) {
00143         in » studentas.vardas_ » studentas.pav_;
00144
00145         int pazymys;
00146         studentas.nd_.clear(); // Išvalome namų darbų sąrašą
00147         while (in » pazymys && pazymys >= 0) {
00148             studentas.addnd(pazymys); // Pridedame naują namų darbo pažymį
00149         }
00150
00151         in » studentas.egzaminas_; // Skaitome egzamino rezultatą
00152
00153         return in;
00154     }
00155
00156     // Output
00157     friend std::ostream& operator«(std::ostream& out, const Studentas& studentas) {
00158         out « "Vardas: " « studentas.vardas_ « std::endl;
00159         out « "Pavarde: " « studentas.pav_ « std::endl;
00160         out « "Namu darbai: ";
00161         for (int pazymys : studentas.nd_) {
00162             out « pazymys « " ";
00163         }
00164         out « std::endl;
00165         out « "Egzamino rezultatas: " « studentas.egzaminas_ « std::endl;
00166         return out;
00167     }
00168 };
00169
00170 #endif /* STUDENTAS_H */
```

6.8 C:/Darbai/2_OP/2_OP/3_OP/test.cpp File Reference

```
#include <iostream>
#include "catch2/catch.hpp"
#include "vector.h"
```

Macros

- #define CATCH_CONFIG_MAIN

Functions

- TEST_CASE ("Vector operations", "[Vector]")

6.8.1 Macro Definition Documentation

6.8.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

6.8.2 Function Documentation

6.8.2.1 TEST_CASE()

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

6.9 C:/Darbai/2_OP/2_OP/3_OP/vector.h File Reference

```
#include <cstddef>
#include <iterator>
#include <memory>
#include <initializer_list>
#include <algorithm>
```

Classes

- class Vector< T, Allocator >

6.10 vector.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <cstddef> // std::size_t
00004 #include <iterator> // std::reverse_iterator
00005 #include <memory> // std::allocator
00006 #include <initializer_list> // std::initializer_list
00007 #include <algorithm> // std::copy, std::equal
00008
00009 template<typename T, typename Allocator = std::allocator<T»
00010 class Vector {
00011 private:
00012     T* m_data;
00013     std::size_t m_size;
00014     std::size_t m_capacity;
00015     Allocator m_alloc;
00016
00017 public:
00018     // MEMBER TYPES
00019     using value_type = T;
00020     using allocator_type = Allocator;
```

```
00021     using size_type = typename std::allocator_traits<Allocator>::size_type;
00022     using difference_type = typename std::allocator_traits<Allocator>::difference_type;
00023     using reference = value_type&;
00024     using const_reference = const value_type&;
00025     using pointer = typename std::allocator_traits<Allocator>::pointer;
00026     using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;
00027     using iterator = T*;
00028     using const_iterator = const T*;
00029     using reverse_iterator = std::reverse_iterator<iterator>;
00030     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00031
00032     //MEMBER FUNCTIONS
00033     // Constructors
00034     Vector() : m_data(nullptr), m_size(0), m_capacity(0), m_alloc() {}
00035     explicit Vector(size_type count, const T& value = T(), const Allocator& alloc = Allocator())
00036         : m_size(count), m_capacity(count), m_alloc(alloc) {
00037         m_data = m_alloc.allocate(m_capacity);
00038         for (std::size_t i = 0; i < m_size; ++i) {
00039             m_alloc.construct(&m_data[i], value);
00040         }
00041     }
00042     explicit Vector(std::initializer_list<T> ilist, const Allocator& alloc = Allocator())
00043         : m_size(ilist.size()), m_capacity(ilist.size()), m_alloc(alloc) {
00044         m_data = m_alloc.allocate(m_capacity);
00045         std::copy(ilist.begin(), ilist.end(), m_data);
00046     }
00047     ~Vector() {
00048         clear();
00049         m_alloc.deallocate(m_data, m_capacity);
00050     }
00051
00052     // Copy constructor
00053     Vector(const Vector& other) : m_size(other.m_size), m_capacity(other.m_capacity),
    m_alloc(other.m_alloc) {
00054         m_data = m_alloc.allocate(m_capacity);
00055         std::copy(other.begin(), other.end(), m_data);
00056     }
00057
00058     // Move constructor
00059     Vector(Vector&& other) noexcept : m_data(other.m_data), m_size(other.m_size),
    m_capacity(other.m_capacity), m_alloc(std::move(other.m_alloc)) {
00060         other.m_data = nullptr;
00061         other.m_size = other.m_capacity = 0;
00062     }
00063
00064     // Operator=
00065     Vector& operator=(const Vector& other) {
00066         if (this != &other) {
00067             clear();
00068             reserve(other.m_size);
00069             std::copy(other.begin(), other.end(), begin());
00070             m_size = other.m_size;
00071         }
00072         return *this;
00073     }
00074     Vector& operator=(Vector&& other) noexcept {
00075         if (this != &other) {
00076             clear();
00077             std::swap(m_data, other.m_data);
00078             std::swap(m_size, other.m_size);
00079             std::swap(m_capacity, other.m_capacity);
00080             std::swap(m_alloc, other.m_alloc);
00081         }
00082         return *this;
00083     }
00084     // assign
00085     void assign(size_type count, const T& value) {
00086         clear();
00087         reserve(count);
00088         for (size_type i = 0; i < count; ++i) {
00089             m_alloc.construct(&m_data[i], value);
00090         }
00091         m_size = count;
00092     }
00093     void assign(std::initializer_list<T> ilist) {
00094         assign(ilist.begin(), ilist.end());
00095     }
00096     template<typename InputIt>
00097     void assign(InputIt first, InputIt last) {
00098         clear();
00099         reserve(std::distance(first, last));
00100         std::copy(first, last, m_data);
00101         m_size = std::distance(first, last);
00102     }
00103
00104     // Non-member functions
00105     friend bool operator==(const Vector& lhs, const Vector& rhs) {
```

```
00106         return lhs.size() == rhs.size() && std::equal(lhs.begin(), lhs.end(), rhs.begin());
00107     }
00108     friend bool operator!=(const Vector& lhs, const Vector& rhs) {
00109         return !(lhs == rhs);
00110     }
00111     friend bool operator<(const Vector& lhs, const Vector& rhs) {
00112         return std::lexicographical_compare(lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
00113     }
00114     friend bool operator<=(const Vector& lhs, const Vector& rhs) {
00115         return !(rhs < lhs);
00116     }
00117     friend bool operator>(const Vector& lhs, const Vector& rhs) {
00118         return rhs < lhs;
00119     }
00120     friend bool operator>=(const Vector& lhs, const Vector& rhs) {
00121         return !(lhs < rhs);
00122     }
00123 
00124     // Element access
00125     reference at(size_type pos) {
00126         if (pos >= m_size) {
00127             throw std::out_of_range("Vector::at");
00128         }
00129         return m_data[pos];
00130     }
00131     const_reference at(size_type pos) const {
00132         if (pos >= m_size) {
00133             throw std::out_of_range("Vector::at");
00134         }
00135         return m_data[pos];
00136     }
00137     reference operator[](size_type pos) {
00138         return m_data[pos];
00139     }
00140     const_reference operator[](size_type pos) const {
00141         return m_data[pos];
00142     }
00143     reference front() {
00144         return m_data[0];
00145     }
00146     const_reference front() const {
00147         return m_data[0];
00148     }
00149     reference back() {
00150         return m_data[m_size - 1];
00151     }
00152     const_reference back() const {
00153         return m_data[m_size - 1];
00154     }
00155     T* data() noexcept {
00156         return m_data;
00157     }
00158     const T* data() const noexcept {
00159         return m_data;
00160     }
00161 
00162     // Iterators
00163     iterator begin() noexcept {
00164         return m_data;
00165     }
00166     const_iterator begin() const noexcept {
00167         return m_data;
00168     }
00169     const_iterator cbegin() const noexcept {
00170         return m_data;
00171     }
00172     iterator end() noexcept {
00173         return m_data + m_size;
00174     }
00175     const_iterator end() const noexcept {
00176         return m_data + m_size;
00177     }
00178     const_iterator cend() const noexcept {
00179         return m_data + m_size;
00180     }
00181     reverse_iterator rbegin() noexcept {
00182         return reverse_iterator(end());
00183     }
00184     const_reverse_iterator rbegin() const noexcept {
00185         return const_reverse_iterator(end());
00186     }
00187     const_reverse_iterator crbegin() const noexcept {
00188         return const_reverse_iterator(end());
00189     }
00190     reverse_iterator rend() noexcept {
00191         return reverse_iterator(begin());
00192     }
```

```
00193     const_reverse_iterator rend() const noexcept {
00194         return const_reverse_iterator(begin());
00195     }
00196     const_reverse_iterator crend() const noexcept {
00197         return const_reverse_iterator(begin());
00198     }
00199
00200     // Capacity
00201     bool empty() const noexcept {
00202         return m_size == 0;
00203     }
00204     size_type size() const noexcept {
00205         return m_size;
00206     }
00207     size_type max_size() const noexcept {
00208         return std::allocator_traits<Allocator>::max_size(m_alloc);
00209     }
00210     void reserve(size_type new_cap) {
00211         if (new_cap > m_capacity) {
00212             pointer new_data = m_alloc.allocate(new_cap);
00213             std::copy(begin(), end(), new_data);
00214             m_alloc.deallocate(m_data, m_capacity); // Atlaisvinama sena atmintis
00215             m_data = new_data;
00216             m_capacity = new_cap;
00217         }
00218     }
00219     size_type capacity() const noexcept {
00220         return m_capacity;
00221     }
00222     void shrink_to_fit() {
00223         if (m_size < m_capacity) {
00224             reserve(m_size);
00225         }
00226     }
00227
00228     // Modifiers
00229     void clear() noexcept {
00230         for (std::size_t i = 0; i < m_size; ++i) {
00231             m_alloc.destroy(&m_data[i]);
00232         }
00233         m_size = 0;
00234     }
00235     iterator insert(const_iterator pos, const T& value) {
00236         difference_type index = pos - begin();
00237         if (m_size >= m_capacity) {
00238             reserve(m_capacity == 0 ? 1 : m_capacity * 2);
00239         }
00240         std::move_backward(begin() + index, end(), end() + 1);
00241         m_alloc.construct(&m_data[index], value);
00242         ++m_size;
00243         return begin() + index;
00244     }
00245     iterator insert(const_iterator pos, size_type count, const T& value) {
00246         difference_type index = pos - begin();
00247         if (count == 0) {
00248             return begin() + index;
00249         }
00250         if (m_size + count > m_capacity) {
00251             reserve(m_capacity == 0 ? count : m_capacity + count);
00252         }
00253         std::move_backward(begin() + index, end(), end() + count);
00254         for (size_type i = 0; i < count; ++i) {
00255             m_alloc.construct(&m_data[index + i], value);
00256         }
00257         m_size += count;
00258         return begin() + index;
00259     }
00260     template<typename InputIt>
00261     iterator insert(const_iterator pos, InputIt first, InputIt last) {
00262         difference_type index = pos - begin();
00263         difference_type count = std::distance(first, last);
00264         if (count == 0) {
00265             return begin() + index;
00266         }
00267         if (m_size + count > m_capacity) {
00268             reserve(m_capacity == 0 ? count : m_capacity + count);
00269         }
00270         std::move_backward(begin() + index, end(), end() + count);
00271         std::copy(first, last, begin() + index);
00272         m_size += count;
00273         return begin() + index;
00274     }
00275     iterator erase(const_iterator pos) {
00276         difference_type index = pos - begin();
00277         m_alloc.destroy(&m_data[index]);
00278         std::move(begin() + index + 1, end(), begin() + index);
00279         --m_size;
```

```
00280         return begin() + index;
00281     }
00282     iterator erase(const_iterator first, const_iterator last) {
00283         difference_type index_first = first - begin();
00284         difference_type index_last = last - begin();
00285         for (difference_type i = index_first; i < index_last; ++i) {
00286             m_alloc.destroy(&m_data[i]);
00287         }
00288         std::move(begin() + index_last, end(), begin() + index_first);
00289         m_size -= (index_last - index_first);
00290         return begin() + index_first;
00291     }
00292     void push_back(const T& value) {
00293         if (m_size >= m_capacity) {
00294             reserve(m_capacity == 0 ? 1 : m_capacity * 2);
00295         }
00296         m_alloc.construct(&m_data[m_size], value);
00297         ++m_size;
00298     }
00299     void push_back(T&& value) {
00300         if (m_size >= m_capacity) {
00301             reserve(m_capacity == 0 ? 1 : m_capacity * 2);
00302         }
00303         m_alloc.construct(&m_data[m_size], std::move(value));
00304         ++m_size;
00305     }
00306     template<typename... Args>
00307     reference emplace_back(Args&&... args) {
00308         if (m_size >= m_capacity) {
00309             reserve(m_capacity == 0 ? 1 : m_capacity * 2);
00310         }
00311         m_alloc.construct(&m_data[m_size], std::forward<Args>(args)...);
00312         ++m_size;
00313         return back();
00314     }
00315     void pop_back() {
00316         if (m_size > 0) {
00317             m_alloc.destroy(&m_data[m_size - 1]);
00318             --m_size;
00319         }
00320     }
00321     void resize(size_type count) {
00322         resize(count, T());
00323     }
00324     void resize(size_type count, const T& value) {
00325         if (count < m_size) {
00326             for (size_type i = count; i < m_size; ++i) {
00327                 m_alloc.destroy(&m_data[i]);
00328             }
00329         } else if (count > m_size) {
00330             if (count > m_capacity) {
00331                 reserve(count);
00332             }
00333             for (size_type i = m_size; i < count; ++i) {
00334                 m_alloc.construct(&m_data[i], value);
00335             }
00336         }
00337         m_size = count;
00338     }
00339     void swap(Vector& other) noexcept {
00340         std::swap(m_data, other.m_data);
00341         std::swap(m_size, other.m_size);
00342         std::swap(m_capacity, other.m_capacity);
00343         std::swap(m_alloc, other.m_alloc);
00344     }
00345 };
```

6.11 C:/Darbai/2_OP/2_OP/3_OP/vektoriai.cpp File Reference

```
#include "functions.h"
#include "vector.h"
```

Functions

- int main ()

6.11.1 Function Documentation

6.11.1.1 main()

```
int main ( )
```

Index