Vector

Generated by Doxygen 1.11.0

Chapter 1

README

Objektinio programavimo užduotis

V3.0

1.0.1 Užpildymo greitis

	std::vector	Vector
10 000	0 ms	0 ms
100 000	0 ms	0 ms
1 000 000	3 ms	3 ms
10 000 000	36 ms	28 ms
100 000 000	391 ms	340 ms

1.0.2 Atminties perskirstymų skaičius

	std::vector	Vector
10 000	23	10
100 000	29	14
1 000 000	34	17
10 000 000	40	20
100 000 000	46	24

1.0.3 Projektas su std::vector VS Vector

	std::vector	Vector
100 000	1224 ms	879 ms
1 000 000	12768 ms	9558 ms
10 000 000	100522 ms	98734 ms

Išvados Kaip matome, Vector klase veikia greičiau, nei std::vector, ir atlieka mažiau perskirstymų
V1.5 Pridėta bazinė klasė "zmogus":

```
class zmogus {
protected:
    string var = " ", pav = " ";
public:
    virtual void setvar(const string& vardas) = 0;
    virtual string getvar() const = 0;
    virtual void setpav(const string& pavarde) = 0;
```

```
    virtual string getpav() const = 0;

    zmogus() = default;
    virtual ~zmogus() {};

    zmogus(const zmogus& laikStud);  // Copy constructor
    zmogus(zmogus&& laikStud) noexcept;  // Move constructor
    zmogus& operator=(const zmogus& laikStud);  // Copy assignment operator
    zmogus& operator=(zmogus&& laikStud) noexcept;  // Move assignment operator
};
```

Kadangi klasė yra abstrakti, naujas objektas tipo "zmogus", nėra leistinas:
Šiai versijai taip pat pritaikyta penkių metodų taisyklė, funkcijos su zmogaus tipo kintamaisiais perkeltos į klasę "zmogus".

V1.2 RULE OF FIVE

Rule of five:

1. Destructor (Destruktorius) - išvalo atmintį, kad nebūtų duomenų nutekėjimo

```
mok::~mok() {
    nd.clear();
}
```

1. Copy Constructor (Kopijabvimo konstruktorius) - leidžia saugiai kopijuoti visą objektą
   ```
   mok::mok(const mok& laikStud) {
       var = laikStud.var;
       pav = laikStud.pav;
       eg = laikStud.eg;
       gal_vid = laikStud.gal_vid;
       gal_med = laikStud.gal_med;
       nd = laikStud.nd;
   }
   ```
2. Copy Assignment Operator (Kopijavimo priskyrimo operatorius) - leidžia jau sukurtam objektui priskirti reikšmes iš kito objekto
   ```
   mok& mok::operator=(const mok& laikStud) {
       if (this != &laikStud) {
           var = laikStud.var;
           pav = laikStud.pav;
           eg = laikStud.eg;
           gal_vid = laikStud.gal_vid;
           gal_med = laikStud.gal_med;
           nd = laikStud.nd;
       }
       return *this;
   }
   ```
3. Move Constructor (Perkėlimo konstruktorius) - leidžia perkelti duomenis iš vieno objekto į kitą
   ```
   mok::mok(mok&& laikStud) noexcept
   :   var(move(laikStud.var)),
       pav(move(laikStud.pav)),
       eg(laikStud.eg),
       gal_vid(laikStud.gal_vid),
       gal_med(laikStud.gal_med),
       nd(move(laikStud.nd))
   {
       laikStud.eg = 0;
       laikStud.gal_vid = 0.0;
       laikStud.gal_med = 0.0;
   }
   ```
4. Move Assignment Operator (Perkįlimo priskyrimo operatorius) - panašus į perkėlimo konstruktorių, bet naudojamas kai duomenys turi būti perkelti į jau egzistuojantį objektą
   ```
   mok& mok::operator=(mok&& laikStud) noexcept {
       if (this != &laikStud) {
           var = move(laikStud.var);
           pav = move(laikStud.pav);
           eg = laikStud.eg;
           gal_vid = laikStud.gal_vid;
           gal_med = laikStud.gal_med;
           nd = move(laikStud.nd);
           laikStud.eg = 0;
           laikStud.gal_vid = 0.0;
           laikStud.gal_med = 0.0;
       }
       return *this;
   }
   ```

Įvesties ir išvesties persidengimo metodai:
Jei rašome pvz.: cout << studentas; - bus išvedami visi objekto "studentas" kintamieji

```
ostream& operator«(ostream& output, const mok& stud) {
```

```
    output « stud.getvar() « " " « stud.getpav() « " " « stud.geteg() « " ";
    vector<int> pazymiai = stud.getnd();
    for (int pazymys : pazymiai) {
        output « pazymys « " ";
    }
    return output;
}
```

Jei rašome pvz.: cin >> studentas; - vartotojas turės švesti visus kintamuosius, kurie turi būti objekte "studentas"

```
istream& operator»(istream& input, mok& stud) {
    string vardas, pavarde;
    int pazymys, egzaminas;
    vector<int> namuD;
    input » vardas » pavarde » egzaminas;
    stud.setvar(vardas);
    stud.setpav(pavarde);
    stud.seteg(egzaminas);
    stud.getnd().clear();
    while (input » pazymys) {
        namuD.push_back(pazymys);
    }
    stud.setnd(namuD);
    return input;
}
```

V1.1 TESTAVIMAS

CLASS VS STRUCT

struct:

class:

Kaip matome, programa su struktūra veikia greičiau.

OPTIMIZAVIMAS

KAIP VEIKIA PROGRAMA

Ši programa skaičiuoja studentų galutinį tam tikro kurso balą, naudodama studento pažymius ir egzaminų rezultatus. Programa realizuota su trimis skirtingais konteineriais: vector, deque, list; ir su trejomis skirtingomis strategijomis, todėl galima pasirinkti, kaip norima dirbti. Galima duomenis įvedinėti ranka, galima ir liepti programai juos skaityti iš failo. Štai taip veikia paleista programa:

1. Paleidus programą išvedamas klausimas, kaip norima, kad būtų nuskaityti duomenys - ar iš failo, ar įvesti ranka;
2. Jei pasirenkame duomenis įrašyti ranka, išvedamas pasirinkimo meniu: ar įvedinėjame ranka, ar leidžiame programai sugeneruoti pažymius automatiškai, ar norime kad tiek studentų vardai, tiek jų pažymiai būtų generuojami auromatiškai;
3. Jei pasirinkame, kad duomenys būtų skaitomi iš failo svarbu, kad failas būtų įkeltas į tą patį aplanką kartu su programa;
4. Vedant vis naujus duomenis, programa paklaus jai rūpimų klausimų, kurie būtini galutinio balo skaičiavimui;
5. Jei buvo pasirinkta duomenis skaityti iš failo, reiks pasirinkti, ir kur juos išvesti - ar į ekraną, ar į naujai sugeneruotą failą.

PASIRUOŠIMAS

Prieš naudojant programą, reikia atsisiųsti Visual Studio: https://visualstudio.microsoft.↩com/downloads/ paspaudę "download" būsite nunaviguoti į puslapį, kuriame bus paaiškinta, kaip dirbti su Visual Studio.

Taip pat reikia atsisiųsti CMake, kuris jums paruoš .exe failus, kad galėtumėte iš karto naudoti programą: https://cmake.org/download/ . Atsisiuntę nueikite į komandinę eilute, nunaviguokite į aplanką, kuriame yra šis projektas (naudokite komandą cd ir įrašykište kelią). Tada įveskite

```
cmake -G "Visual Studio 17 2022" -A x64 .
```

Ir:

```
cmake --build . --config Release
```

V1.0 TESTAVIMAS

Testavimo sistemos parametrai:

PIRMA TESTAVIMO DALIS

Rezultatu apibendrinimas ir palyginimas:

ANTRA TESTAVIMO DALIS

1 STRATEGIJA:

2 STRATEGIJA

3 STRATEGIJA

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

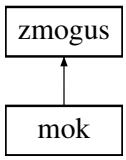
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 mok Class Reference

Isvestine klase, aprasanti studenta.
`#include <studentas.h>`
Inheritance diagram for mok:



Public Member Functions

- void setvar (const string &vardas)

  < *Galutinis balas, apskaiciuotas naudojant mediana*
- string getvar () const

  Grazina varda.
- void setpav (const string &pavarde)

  Nustato pavarde.
- string getpav () const

  Grazina pavarde.
- void seteg (int egzaminas)

  Nustato egzamino bala.
- int geteg () const

  Grazina egzamino bala.
- void setgal_vid (double Gal_vid)

  Nustato galutini bala, apskaiciuota naudojant vidurki.
- double getgal_vid () const

  Grazina galutini bala, apskaiciuota naudojant vidurki.
- void setgal_med (double Gal_med)

  Nustato galutini bala, apskaiciuota naudojant mediana.
- double getgal_med () const

  Grazina galutini bala, apskaiciuota naudojant mediana.
- void setnd (const Vector< int > &ND)

  Nustato namu darbu pazymius.
- Vector< int > getnd () const

  Grazina namu darbu pazymius.
- void **isvalymas** ()

Isvalo objekto duomenis.

- **mok** ()=default

  Numatytasis konstruktorius.
- ∼**mok** ()

  Destruktorius.
- mok (const mok &laikStud)

  Kopijavimo konstruktorius.
- mok & operator= (const mok &laikStud)

  Kopijavimo priskyrimo operatorius.
- mok (mok &&laikStud) noexcept

  Perkelimo konstruktorius.
- mok & operator= (mok &&laikStud) noexcept

  Perkelimo priskyrimo operatorius.

Public Member Functions inherited from zmogus

- **zmogus** ()=default

  Numatytasis konstruktorius.
- virtual ∼**zmogus** ()

  Destruktorius.
- zmogus (const zmogus &laikStud)

  Kopijavimo konstruktorius.
- zmogus (zmogus &&laikStud) noexcept

  Perkelimo konstruktorius.
- zmogus & operator= (const zmogus &laikStud)

  Kopijavimo priskyrimo operatorius.
- zmogus & operator= (zmogus &&laikStud) noexcept

  Perkelimo priskyrimo operatorius.

Friends

- ostream & operator<< (ostream &output, const mok &stud)

  Operatorius srauto isvedimui.
- istream & operator>> (istream &input, mok &stud)

  Operatorius srauto ivedimui.

Additional Inherited Members

Protected Attributes inherited from zmogus

- string **var** = " "

  Zmogaus vardas.
- string **pav** = " "

  Zmogaus pavarde.

5.1.1 Detailed Description

Isvestine klase, aprasanti studenta.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 mok() [1/2]

```
mok::mok (
            const mok & laikStud)
```

Kopijavimo konstruktorius.

Parameters

laikStud	Objektas, is kurio kopijuojama.

5.1.2.2 mok() [2/2]

```
mok::mok (
            mok && laikStud)  [noexcept]
```

Perkelimo konstruktorius.

Parameters

laikStud	Objektas, is kurio perkeliama.

5.1.3 Member Function Documentation

5.1.3.1 geteg()

```
int mok::geteg () const
```

Grazina egzamino bala.

Returns

Egzamino balas.

5.1.3.2 getgal_med()

```
double mok::getgal_med () const
```

Grazina galutini bala, apskaiciuota naudojant mediana.

Returns

Galutinis balas, apskaiciuotas maudojant mediana.

5.1.3.3 getgal_vid()

```
double mok::getgal_vid () const
```

Grazina galutini bala, apskaiciuota naudojant vidurki.

Returns

Galutinis balas, apskaiciuotas maudojant vidurki.

5.1.3.4 getnd()

```
Vector< int > mok::getnd () const
```

Grazina namu darbu pazymius.

Returns

Namu darbu pazymiai.

5.1.3.5 getpav()

```
string mok::getpav () const  [virtual]
```

Grazina pavarde.

Returns

Pavarde.

Implements zmogus.

5.1.3.6 getvar()

```
string mok::getvar () const  [virtual]
```

Grazina varda.

Returns

Vardas.

Implements zmogus.

5.1.3.7 operator=() [1/2]

```
mok & mok::operator= (
            const mok & laikStud)
```

Kopijavimo priskyrimo operatorius.

Parameters

laikStud	Objektas, is kurio kopijuojama.

Returns

Priskirtas objektas.

5.1.3.8 operator=() [2/2]

```
mok & mok::operator= (
            mok && laikStud)  [noexcept]
```

Perkelimo priskyrimo operatorius.

Parameters

laikStud	Objektas, is kurio perkeliama.

Returns

Priskirtas objektas.

5.1.3.9 seteg()

```
void mok::seteg (
            int egzaminas)
```

Nustato egzamino bala.

Parameters

egzaminas	Egzamino balas.

5.1.3.10 setgal_med()

```
void mok::setgal_med (
            double Gal_med)
```

Nustato galutini bala, apskaiciuota naudojant mediana.

Parameters

Gal_vid	Galutinis balas, apskaiciuotas naudojant mediana.

5.1.3.11 setgal_vid()

```
void mok::setgal_vid (
            double Gal_vid)
```

Nustato galutini bala, apskaiciuota naudojant vidurki.

Parameters

Gal_vid	Galutinis balas, apskaiciuotas naudojant vidurki.

5.1.3.12 setnd()

```
void mok::setnd (
            const Vector< int > & ND)
```

Nustato namu darbu pazymius.

Parameters

ND	Namu darbu pazymiai.

5.1.3.13 setpav()

```
void mok::setpav (
            const string & pavarde)  [virtual]
```

Nustato pavarde.

Parameters

pavarde	Pavarde.

Implements zmogus.

5.1.3.14 setvar()

```
void mok::setvar (
            const string & vardas)  [virtual]
```

< Galutinis balas, apskaiciuotas naudojant mediana
Nustato varda.

Parameters

vardas	Vardas.

Implements zmogus.

5.1.4 Friends And Related Symbol Documentation

5.1.4.1 operator<<

```
ostream & operator<< (
            ostream & output,
            const mok & stud)  [friend]
```

Operatorius srauto isvedimui.

Parameters

output	Srautas.
stud	Studentas.

Returns

Srauto objektas.

5.1.4.2 operator>>

```
istream & operator>> (
            istream & input,
            mok & stud)  [friend]
```

Operatorius srauto ivedimui.

Parameters

input	Srautas.
stud	Studentas.

Returns

Srauto objektas.

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

- studentas.h
- studentas.cpp

5.2 Vector< T > Class Template Reference

Vector klase, sukurta remiantis std::vector.

`#include <Vector.hpp>`

Public Types

- using value_type = T
- using iterator = T*
- using const_iterator = const T*
- using reference = T&
- using const_reference = const T&
- using size_type = int

Public Member Functions

- **Vector** ()

  Konstruktorius.
- **Vector** (initializer_list< T > init_list)

  Konstruktorius, kuris leidzia is karto inicializuoti sarasa param init_list Inicializavimo sarasas.
- ~**Vector** ()

  Destruktorius.
- Vector (const Vector< T > &copy)

  Kopijavimo konstruktorius.
- Vector & operator= (const Vector< T > &copy)

  Kopijavimo priskyrimo operatorius.
- Vector (Vector< T > &&kitas) noexcept

  Perkelimo konstruktorius.
- Vector & operator= (Vector< T > &&kitas) noexcept

  Perkelimo priskyrimo operatorius.
- iterator Begin ()

Grazina iteratoriu i pirmaji elementa.

- iterator End ()

  Grazina iteratoriu i elementa po paskutiniojo.
- const_iterator begin () const

  Grazina konstantini iteratoriu i pirmaji elementa.
- const_iterator end () const

  Grazina konstantini iteratoriu i elementa po paskutiniojo.
- int Size () const

  Grazina elementu skaiciu vektoriuje.
- int Capacity () const

  Grazina vektoriaus talpa.
- bool IsEmpty () const

  Patikrina, ar vektorius tuscias.
- int MaxSize () const

  Grazina didziausia elementa vektoriuje.
- void Resize (int newSize)

  Keicia vektoriaus dydi.
- void Reserve (int newCapacity)

  Keicia vektoriaus talpa.
- void **ShrinkToFit** ()

  Sumazina vektoriaus talpa iki jo dydzio.
- T & operator[ ] (int index)

  Grazina elementa pagal indeksa (su modifikavimo galimybe)
- const T & operator[ ] (int index) const

  Grazina elementa pagal indeksa (be modifikavimo galimybes)
- T & Front ()

  Grazina pirmaji elementa.
- T & Back ()

  Grazina paskutini elementa.
- void Assign (int n, const T &value)

  Pakeicia vektoriaus elementus naujais.
- void PushBack (const T &object)

  Prideda nauja elementa i vektoriaus gala.
- void **PopBack** ()

  Pasalina paskutini elementa.
- void Insert (int index, const T &value)

  Iterpia nauja elementa nurodytoje pozicijoje.
- template<typename InputIt >
  void Insert (iterator pos, InputIt first, InputIt last)

  Iterpia elementus nurodytoje pozicijoje.
- void Erase (int index)

  Pasalina elementa nurodytoje pozicijoje.
- void Erase (iterator first, iterator last)

  Pasalina elementus nurodytame intervale.
- void **Clear** ()

  Isvalo visus vektoriaus elementus.
- void Swap (Vector< T > &first, Vector< T > &second)

  Sukeicia vietomis du vektorius.

Friends

- bool operator== (const Vector< T > &first, const Vector< T > &second)

  Lygina du vektorius.
- bool operator!= (const Vector< T > &first, const Vector< T > &second)

  Lygina du vektorius.

5.2.1 Detailed Description

template<typename T>
class Vector< T >

Vector klase, sukurta remiantis std::vector.

5.2.2 Member Typedef Documentation

5.2.2.1 const_iterator

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

Konstantinio iteratoriaus tipas

5.2.2.2 const_reference

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

Konstantines nuorodos tipas

5.2.2.3 iterator

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

Iteratoriaus tipas

5.2.2.4 reference

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

Nuorodos tipas

5.2.2.5 size_type

```
template<typename T >
using Vector< T >::size_type = int
```

Dydzio tipas

5.2.2.6 value_type

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

Elementu tipas

5.2.3 Constructor & Destructor Documentation

5.2.3.1 Vector() [1/2]

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

Kopijavimo konstruktorius.

Parameters

copy	Kitas vektorius, kuri reikia kopijuoti

5.2.3.2 Vector() [2/2]

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

Perkelimo konstruktorius.

Parameters

kitas	Kitas vektorius, kuri reikia perkelti

5.2.4 Member Function Documentation

5.2.4.1 Assign()

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

Pakeicia vektoriaus elementus naujais.

Parameters

n	Naujas elementu skaicius
value	Nauja reiksme

5.2.4.2 Back()

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

Grazina paskutini elementa.

Returns

Nuoroda i paskutini elementa

5.2.4.3 Begin()

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

Grazina iteratoriu i pirmaji elementa.

Returns

Iteratorius i pirmaji elementa

5.2.4.4 begin()

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

Grazina konstantini iteratoriu i pirmaji elementa.

Returns

Konstantinis iteratorius i pirmaji elementa

5.2.4.5 Capacity()

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

Grazina vektoriaus talpa.

Returns

Vektoriaus talpa

5.2.4.6 End()

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

Grazina iteratoriu i elementa po paskutiniojo.

Returns

Iteratorius i elementa po paskutiniojo

5.2.4.7 end()

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

Grazina konstantini iteratoriu i elementa po paskutiniojo.

Returns

Konstantinis iteratorius i elementa po paskutiniojo

5.2.4.8 Erase() [1/2]

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

Pasalina elementa nurodytoje pozicijoje.

Parameters

index	Elemento indeksas

Exceptions

out_of_range	Jei indeksas netinkamas

5.2.4.9 Erase() [2/2]

```
template<typename T >
void Vector< T >::Erase (
            iterator first,
            iterator last)  [inline]
```

Pasalina elementus nurodytame intervale.

Parameters

first	Pirmasis iteratorius
last	Paskutinis iteratorius

Exceptions

out_of_range	Jei intervalas netinkamas

5.2.4.10 Front()

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

Grazina pirmaji elementa.

Returns

Nuoroda i pirmaji elementa

5.2.4.11 Insert() [1/2]

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

Iterpia nauja elementa nurodytoje pozicijoje.

Parameters

index	Iterpimo pozicija
value	Naujasis elementas

Exceptions

out_of_range	Jei indeksas netinkamas

5.2.4.12 Insert() [2/2]

```
template<typename T >
template<typename InputIt >
void Vector< T >::Insert (
            iterator pos,
            InputIt first,
            InputIt last)  [inline]
```

Iterpia elementus nurodytoje pozicijoje.

Parameters

pos	Pozicija, kurioje iterpti elementai
first	Pirmas elementas
last	Paskutinis elementas

Exceptions

out_of_range	Jei indeksas netinkamas

5.2.4.13 IsEmpty()

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

Patikrina, ar vektorius tuscias.

Returns

true, jei tuscias, false, jei ne

5.2.4.14 MaxSize()

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

Grazina didziausia elementa vektoriuje.

Returns

Maksimalus dydis vektoriuje

5.2.4.15 operator=() [1/2]

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

Kopijavimo priskyrimo operatorius.

Parameters

copy	Kitas vektorius, kuri reikia kopijuoti

Returns

Nuoroda i si vektoriu

5.2.4.16 operator=() [2/2]

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

Perkelimo priskyrimo operatorius.

Parameters

kitas	Kitas vektorius, kuri reikia perkelti

Returns

Nuoroda i si vektoriu

5.2.4.17 operator[]() [1/2]

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

Grazina elementa pagal indeksa (su modifikavimo galimybe)

Parameters

index	Elemento indeksas

Returns

Nuoroda i elementa

Exceptions

out_of_range	Jei indeksas netinkamas

5.2.4.18 operator[]() [2/2]

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

Grazina elementa pagal indeksa (be modifikavimo galimybes)

Parameters

index	Elemento indeksas

Returns

Konstanta nuoroda i elementa

Exceptions

out_of_range	Jei indeksas netinkamas

5.2.4.19 PushBack()

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

Prideda nauja elementa i vektoriaus gala.

Parameters

object	Naujasis elementas

5.2.4.20 Reserve()

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

Keicia vektoriaus talpa.

Parameters

newCapacity	Nauja vektoriaus talpa

5.2.4.21 Resize()

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

Keicia vektoriaus dydi.

Parameters

newSize	Naujas vektoriaus dydis

5.2.4.22 Size()

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

Grazina elementu skaiciu vektoriuje.

Returns

Elementu skaicius

5.2.4.23 Swap()

```
template<typename T >
void Vector< T >::Swap (
            Vector< T > & first,
            Vector< T > & second)  [inline]
```

Sukeicia vietomis du vektorius.

Parameters

first	Pirmasis vektorius
second	Antrasis vektorius

5.2.5 Friends And Related Symbol Documentation

5.2.5.1 operator"!=

```
template<typename T >
bool operator!= (
            const Vector< T > & first,
            const Vector< T > & second)  [friend]
```

Lygina du vektorius.

Parameters

first	Pirmasis vektorius
second	Antrasis vektorius

Returns

true, jei vektoriai yra nelygus, false kitu atveju

5.2.5.2 operator==

```
template<typename T >
bool operator== (
            const Vector< T > & first,
            const Vector< T > & second)  [friend]
```

Lygina du vektorius.
Vektoriai lygus tada, kada ju dydis ir elementai vienodi.

Parameters

first	Pirmasis vektorius
second	Antrasis vektorius

Returns

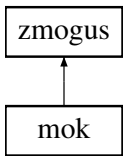
true, jei vektoriai yra lygus, false kitu atveju

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

- Vector.hpp

5.3 zmogus Class Reference

Abstrakti klase, sukurta aprasyti zmogu.
`#include <studentas.h>`
Inheritance diagram for zmogus:



Public Member Functions

- virtual void setvar (const string &vardas)=0
  Nustato varda.
- virtual string getvar () const =0
  Grazina varda.
- virtual void setpav (const string &pavarde)=0
  Nustato pavarde.
- virtual string getpav () const =0
  Grazina pavarde.
- **zmogus** ()=default
  Numatytasis konstruktorius.
- virtual ~**zmogus** ()
  Destruktorius.
- zmogus (const zmogus &laikStud)
  Kopijavimo konstruktorius.
- zmogus (zmogus &&laikStud) noexcept
  Perkelimo konstruktorius.
- zmogus & operator= (const zmogus &laikStud)
  Kopijavimo priskyrimo operatorius.
- zmogus & operator= (zmogus &&laikStud) noexcept
  Perkelimo priskyrimo operatorius.

Protected Attributes

- string **var** = " "

  Zmogaus vardas.
- string **pav** = " "

  Zmogaus pavarde.

5.3.1 Detailed Description

Abstrakti klase, sukurta aprasyti zmogu.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 zmogus() [1/2]

```
zmogus::zmogus (
            const zmogus & laikStud)
```

Kopijavimo konstruktorius.

Parameters

laikStud	Objektas, is kurio kopijuojama.

5.3.2.2 zmogus() [2/2]

```
zmogus::zmogus (
            zmogus && laikStud)  [noexcept]
```

Perkelimo konstruktorius.

Parameters

laikStud	Objektas, is kurio perkeliama.

5.3.3 Member Function Documentation

5.3.3.1 getpav()

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

Grazina pavarde.

Returns

Pavarde.

Implemented in mok.

5.3.3.2 getvar()

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

Grazina varda.

Returns

Vardas.

Implemented in mok.

5.3.3.3 operator=() [1/2]

```
zmogus & zmogus::operator= (
            const zmogus & laikStud)
```

Kopijavimo priskyrimo operatorius.

Parameters

laikStud	Objektas, is kurio kopijuojama.

Returns

Priskirtas objektas.

5.3.3.4 operator=() [2/2]

```
zmogus & zmogus::operator= (
            zmogus && laikStud)  [noexcept]
```

Perkelimo priskyrimo operatorius.

Parameters

laikStud	Objektas, is kurio perkeliama.

Returns

Priskirtas objektas.

5.3.3.5 setpav()

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

Nustato pavarde.

Parameters

pavarde	Pavarde.

Implemented in mok.

5.3.3.6 setvar()

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

Nustato varda.

Parameters

vardas	Vardas.

Implemented in mok.
The documentation for this class was generated from the following files:

- studentas.h
- studentas.cpp

Chapter 6

File Documentation

6.1 funkcijos.h File Reference

siame faile aprasytos kode naudojamos funkcijos

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <numeric>
#include <ctime>
#include <fstream>
#include <sstream>
#include <string>
#include <chrono>
#include <cstdlib>
#include <cassert>
#include "Vector.hpp"
```

Functions

- void ivedimas (Vector< mok > &stud)

  Ivedimo funkcija.
- void calculateResults (Vector< mok > &stud)

  Rezultatu skaiciavimo funkcija.
- char rikiavimoklausimas ()

  Rikiavimo klausimas.
- void isvedimas (Vector< mok > &stud, ostream &os, char a)

  Isvedimo funkcija.
- void meniu (int &antrasPasirinkimas)

  Pirmas meniu.
- void meniuAntras (int &ketvirtasPasirinkimas)

  Antras meniu.
- bool pagalVarda (const mok &a, const mok &b)

  Rikiavimo funkcija pagal varda.
- bool pagalPavarde (const mok &a, const mok &b)

  Rikiavimo funkcija pagal pavarde.
- bool pagalMediana (const mok &a, const mok &b)

  Rikiavimo funkcija pagal mediana.
- bool pagalVidurki (const mok &a, const mok &b)

  Rikiavimo funkcija pagal vidurki.
- milliseconds trukmesSkaiciavimas (high_resolution_clock::time_point pradzia, high_resolution_clock::time_point pabaiga)

Trukmes skaiciavimo funkcija.

- void failuGeneravimas (int studentuKiekis, const string &failoPavadinimas)

  Failu generavimo funkcija.
- void konteineriai (int studentuKiekis, Vector< mok > &studentai, char a, Vector< mok > &vargsiukai)

  Skirstymo i konteinerius funkcija.
- void isvalymas (Vector< mok > &vektorius)

  Vektoriaus isvalymo funkcija.
- void failuNuskaitymas (Vector< mok > &studentai, string &failoPavadinimas)

  Failu nuskaitymo funkcija.
- int pirmasP (int &pirmasPasirinkimas)

  Klausimas naudotojui.
- int treciasP (int &treciasPasirinkimas)

  Klausimas naudotojui.
- void rikiavimas (const int ketvirtaspasirinkimas, Vector< mok > &studentai)

  Rikiavimo funkcija.
- void testavimoRezultatai (bool result, const string &testas)

  Testavimo rezultatu funkcija.

6.1.1 Detailed Description

siame faile aprasytos kode naudojamos funkcijos

6.1.2 Function Documentation

6.1.2.1 calculateResults()

```
void calculateResults (
            Vector< mok > & stud)
```

Rezultatu skaiciavimo funkcija.

Parameters

stud	Studentu vektorius.

6.1.2.2 failuGeneravimas()

```
void failuGeneravimas (
            int studentuKiekis,
            const string & failoPavadinimas)
```

Failu generavimo funkcija.

Parameters

studentuKiekis	Studentu Kiekis.
failoPavadinimas	Failo pavadinimas.

6.1.2.3 failuNuskaitymas()

```
void failuNuskaitymas (
            Vector< mok > & studentai,
            string & failoPavadinimas)
```

Failu nuskaitymo funkcija.

Parameters

studentai	Studentu vektorius.
failoPavadinimas	Failo pavadinimas.

6.1.2.4 isvalymas()

```
void isvalymas (
            Vector< mok > & vektorius)
```

Vektoriaus isvalymo funkcija.

Parameters

vektorius	Vektorius, kuri reikia isvalyti.

6.1.2.5 isvedimas()

```
void isvedimas (
            Vector< mok > & stud,
            ostream & os,
            char a)
```

Isvedimo funkcija.
Si funkcija naudojama ir isvedimui i faila, ir i ekrana.

Parameters

stud	Studentu vektorius.
os	Ivesties srautas.
a	Rikiavimo tipa nurodantis simbolis.

6.1.2.6 ivedimas()

```
void ivedimas (
            Vector< mok > & stud)
```

Ivedimo funkcija.
Si funkcija naudojama tik pasirinkus ivedima ranka.

Parameters

stud	Studentu vektorius.

6.1.2.7 konteineriai()

```
void konteineriai (
            int studentuKiekis,
            Vector< mok > & studentai,
            char a,
            Vector< mok > & vargsiukai)
```

Skirstymo i konteinerius funkcija.

Parameters

studentuKiekis	Studentu kiekis.
studentai	Studentu vektorius.

a	Rikiavimo tipa nurodantis simbolis.
vargsiukai	Vargsiuku vektorius.

6.1.2.8 meniu()

```
void meniu (
            int & antrasPasirinkimas)
```

Pirmas meniu.
Sis meniu atsiranda tada, kada naudotojas pasirenka duomenis ivedineti ranka.

Parameters

antrasPasirinkimas	Naudotojo pasirinkimas.

6.1.2.9 meniuAntras()

```
void meniuAntras (
            int & ketvirtasPasirinkimas)
```

Antras meniu.
Naudotojas renkasi, pagal ka isrikiuotus duomenis nori matyti.

Parameters

ketvirtasPasirinkimas	Naudotojo pasirinkimas.

6.1.2.10 pagalMediana()

```
bool pagalMediana (
            const mok & a,
            const mok & b)
```

Rikiavimo funkcija pagal mediana.

Parameters

a	Pirmas mok objektas.
b	Antras mok objektas.

Returns

Tiesa, jei pirma mediana yra mazesne uz antra.

6.1.2.11 pagalPavarde()

```
bool pagalPavarde (
            const mok & a,
            const mok & b)
```

Rikiavimo funkcija pagal pavarde.

Parameters

a	Pirmas mok objektas.
b	Antras mok objektas.

Returns

Tiesa, jei pirma pavarde yra zemiau uz antra.

6.1.2.12 pagalVarda()

```
bool pagalVarda (
            const mok & a,
            const mok & b)
```

Rikiavimo funkcija pagal varda.

Parameters

a	Pirmas mok objektas.
b	Antras mok objektas.

Returns

Tiesa, jei pirmas vardas yra zemiau uz antra.

6.1.2.13 pagalVidurki()

```
bool pagalVidurki (
            const mok & a,
            const mok & b)
```

Rikiavimo funkcija pagal vidurki.

Parameters

a	Pirmas mok objektas.
b	Antras mok objektas.

Returns

Tiesa, jei pirmas vidurkis yra mazesnis uz antra.

6.1.2.14 pirmasP()

```
int pirmasP (
            int & pirmasPasirinkimas)
```

Klausimas naudotojui.
Funkcija, kuri klausia naudotojo ar jis nori duomenis ivesti ranka, ar nuskaityti is failo.

Parameters

pirmasPasirinkimas	Naudotojo pasirinkimas.

Returns

Naudotojo pasirinkimas.

6.1.2.15 rikiavimas()

```
void rikiavimas (
            const int ketvirtaspasirinkimas,
            Vector< mok > & studentai)
```

Rikiavimo funkcija.

Parameters

ketvirtaspasirinkimas	Ketvirtas pasirinkimas.
studentai	Studentu vektorius.

6.1.2.16 rikiavimoklausimas()

```
char rikiavimoklausimas ()
```

Rikiavimo klausimas.

Returns

Simbolis, nurodantis pagal ka bus rikiuojami duomenys.

6.1.2.17 testavimoRezultatai()

```
void testavimoRezultatai (
            bool result,
            const string & testas)
```

Testavimo rezultatu funkcija.

Parameters

result	Testo rezultatas.
testas	Testo pavadinimas.

6.1.2.18 treciasP()

```
int treciasP (
            int & treciasPasirinkimas)
```

Klausimas naudotojui.
Funkcija, kuri klausia naudotojo ar jis nori, kad duomenys butu isvesti ekrane, ar i faila.

Parameters

treciasPasirinkimas	Naudotojo pasirinkimas.

Returns

Naudotojo pasirinkimas.

6.1.2.19 trukmesSkaiciavimas()

```
milliseconds trukmesSkaiciavimas (
            high_resolution_clock::time_point pradzia,
            high_resolution_clock::time_point pabaiga)
```

Trukmes skaiciavimo funkcija.

Parameters

pradzia	Pradzios laikas.
pabaiga	Pabaigos laikas.

Returns

Trukmes intervalas milisekundemis.

6.2 funkcijos.h

Go to the documentation of this file.

```
00001
00006 #ifndef FUNKCIJOS_H
00007 #define FUNKCIJOS_H
00008
00009 #include <iostream>
00010 #include <iomanip>
00011 #include <algorithm>
00012 #include <numeric>
00013 #include <ctime>
00014 #include <fstream>
00015 #include <sstream>
00016 #include <string>
00017 #include <chrono>
00018 #include <cstdlib>
00019 #include <cassert>
00020 #include "Vector.hpp"
00021
00022 using namespace std;
00023 using namespace std::chrono;
00024
00030 void ivedimas(Vector<mok>& stud);
00031
00036 void calculateResults(Vector<mok>& stud);
00037
00042 char rikiavimoklausimas();
00043
00051 void isvedimas(Vector<mok>& stud, ostream& os, char a);
00052
00058 void meniu(int& antrasPasirinkimas);
00059
00065 void meniuAntras(int& ketvirtasPasirinkimas);
00066
00073 bool pagalVarda(const mok& a, const mok& b);
00074
00081 bool pagalPavarde(const mok& a, const mok& b);
00082
00089 bool pagalMediana(const mok& a, const mok& b);
00090
00097 bool pagalVidurki(const mok& a, const mok& b);
00098
00105 milliseconds trukmesSkaiciavimas(high_resolution_clock::time_point pradzia,
      high_resolution_clock::time_point pabaiga);
00106
00112 void failuGeneravimas(int studentuKiekis, const string& failoPavadinimas);
00113
00121 void konteineriai(int studentuKiekis, Vector<mok>& studentai, char a, Vector<mok>& vargsiukai);
00122
00127 void isvalymas(Vector<mok>& vektorius);
00128
00134 void failuNuskaitymas(Vector<mok>& studentai, string& failoPavadinimas);
00135
00142 int pirmasP(int& pirmasPasirinkimas);
00143
00150 int treciasP(int& treciasPasirinkimas);
00151
00157 void rikiavimas(const int ketvirtaspasirinkimas, Vector<mok>& studentai);
00158
00164 void testavimoRezultatai(bool result, const string& testas);
00165
00166 #endif
```

6.3 foo.h

```
00001
```

6.4 foo.h

```
00001
```

6.5 studentas.h File Reference

siame faile aprasytos studento ir zmogaus klases

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

```
#include "Vector.hpp"
```

Classes

- class zmogus

  Abstrakti klase, sukurta aprasyti zmogu.
- class mok

  Isvestine klase, aprasanti studenta.

6.5.1 Detailed Description

siame faile aprasytos studento ir zmogaus klases

6.6 studentas.h

Go to the documentation of this file.

```
00001
00006 #ifndef STUDENTAS_H
00007 #define STUDENTAS_H
00008
00009 #include <iostream>
00010 #include <string>
00011 #include "Vector.hpp"
00012
00013 using namespace std;
00014
00019 class zmogus {
00020 protected:
00021     string var = " ";
00022     string pav = " ";
00023 public:
00024
00029     virtual void setvar(const string& vardas) = 0;
00030
00035     virtual string getvar() const = 0;
00036
00041     virtual void setpav(const string& pavarde) = 0;
00042
00047     virtual string getpav() const = 0;
00048
00049
00053     zmogus() = default;
00054
00058     virtual ~zmogus() {};
00059
00060
00065     zmogus(const zmogus& laikStud);  // Copy constructor
00066
00071     zmogus(zmogus&& laikStud) noexcept;  // Move constructor
00072
00078     zmogus& operator=(const zmogus& laikStud);  // Copy assignment operator
00079
00085     zmogus& operator=(zmogus&& laikStud) noexcept;  // Move assignment operator
00086 };
00087
00092 class mok : public zmogus {
00093 private:
00094     int eg = 0;
00095     Vector<int> nd = { 0 };
00096     double gal_vid = 0.0;
00097     double gal_med = 0.0;
00098 public:
00099
00104     void setvar(const string& vardas);
00105
00110     string getvar() const;
00111
00116     void setpav(const string& pavarde);
00117
00122     string getpav() const;
00123
00128     void seteg(int egzaminas);
00129
00134     int geteg()const;
00135
00140     void setgal_vid(double Gal_vid);
```

```
00141
00146     double getgal_vid() const;
00147
00152     void setgal_med(double Gal_med);
00153
00158     double getgal_med() const;
00159
00164     void setnd(const Vector<int>& ND);
00165
00170     Vector<int> getnd() const;
00171
00175     void isvalymas();
00176     // RULE OF FIVE:
00177
00181     mok() = default;
00182
00186     ~mok();
00187
00192     mok(const mok& laikStud);
00193
00199     mok& operator=(const mok& laikStud);
00200
00205     mok(mok&& laikStud) noexcept;
00206
00212     mok& operator=(mok&& laikStud) noexcept;
00213
00220     friend ostream& operator« (ostream& output, const mok& stud);
00221
00228     friend istream& operator»(istream& input, mok& stud);
00229
00230 };
00231
00232 #endif
```

6.7 Vector.hpp File Reference

siame faile aprasyta naujo vektoriaus klase

```
#include <stdexcept>
#include <algorithm>
#include <limits>
```

Classes

- class Vector< T >

  Vector klase, sukurta remiantis std::vector.

Variables

- const int **DEFAULT_VECTOR_SIZE** = 10

6.7.1 Detailed Description

siame faile aprasyta naujo vektoriaus klase

6.8 Vector.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef VECTOR_H
00007 #define VECTOR_H
00008
00009 #include <stdexcept>
00010 #include <algorithm>
00011 #include <limits>
00012
00013
00014 using namespace std;
00015
00021 const int DEFAULT_VECTOR_SIZE = 10;
00022
00023 template <typename T>
```

```
00024 class Vector {
00025 private:
00026     int size;
00027     int capacity;
00028     T* elements;
00030 public:
00031     // MEMBER TYPES
00032     using value_type = T;
00033     using iterator = T*;
00034     using const_iterator = const T*;
00035     using reference = T&;
00036     using const_reference = const T&;
00037     using size_type = int;
00039
      //*****************************************************************************************************************************
00043     Vector() {                                      // Vector<int> v; - konstruktorius
00044         size = 0;
00045         capacity = DEFAULT_VECTOR_SIZE;
00046         elements = new T[DEFAULT_VECTOR_SIZE];
00047     }
00052     Vector(initializer_list<T> init_list) { // Intializer list konstruktorius
00053         size = init_list.size();
00054         capacity = size > DEFAULT_VECTOR_SIZE ? size : DEFAULT_VECTOR_SIZE;
00055         elements = new T[capacity];
00056         copy(init_list.begin(), init_list.end(), elements);
00057     }
00058
00062     ~Vector() {                                     // Destruktorius
00063         delete[] elements;
00064
00065     }
00066
00067
      //*****************************************************************************************************************************
00068
00069     //RULE OF FIVE
00070
00075     Vector(const Vector<T>& copy) {             // Vector<int> v(w); - kopijavimo kosntruktorius
00076         size = copy.size;
00077         capacity = copy.capacity;
00078         elements = new T[capacity];
00079         for (int i = 0; i < copy.size; i++) {
00080             elements[i] = copy.elements[i];
00081         }
00082     }
00083
00089     Vector& operator = (const Vector<T>& copy) {  // v = w; - kopijavimo priskyrimo operatorius
00090         if (this != &copy) {
00091             if (copy.size > capacity) {
00092                 delete[] elements;
00093                 capacity = copy.size * 2;
00094                 elements = new T[capacity];
00095             }
00096             for (int i = 0; i < copy.size; i++) {
00097                 elements[i] = copy.elements[i];
00098             }
00099             size = copy.size;
00100         }
00101         return *this;
00102     }
00103
00108     Vector(Vector<T>&& kitas) noexcept               // -perkelimo konstruktosius
00109     :   size(kitas.size),
00110         capacity(kitas.capacity),
00111         elements(kitas.elements)
00112     {
00113             kitas.size = 0;
00114             kitas.capacity = 0;
00115             kitas.elements = nullptr;
00116     }
00117
00123     Vector& operator=(Vector<T>&& kitas) noexcept { // perkelimo priskyrimo operatorius
00124         if (this != &kitas) {
00125             delete[] elements;
00126             size = kitas.size;
00127             capacity = kitas.capacity;
00128             elements = kitas.elements;
00129             kitas.size = 0;
00130             kitas.capacity = 0;
00131             kitas.elements = nullptr;
00132         }
00133         return *this;
00134     }
00135
00136
      //*****************************************************************************************************************************
00137
```

```
00138     // ITERTAORS
00139
00144     iterator Begin() {                          // v.Begin()
00145         return elements;
00146     }
00147
00152     iterator End() {                             //v.End()
00153         return elements + size;
00154     }
00155
00160     const_iterator begin() const {                  // const version of v.begin()
00161         return elements;
00162     }
00163
00168     const_iterator end() const {                  // const version of v.end()
00169         return elements + size;
00170     }
00171
      //*****************************************************************************************************************************
00172
00173
00174     // CAPACITY
00175
00180     int Size() const {                          // v.Size()
00181         return size;
00182     }
00183
00188     int Capacity() const {                      // v.Capacity()
00189         return capacity;
00190     }
00191
00196     bool IsEmpty() const {                       // v.IsEmpty()
00197         return size == 0;
00198     }
00199
00204     int MaxSize() const {
00205         return numeric_limits<int>::max();
00206     }
00207
00212     void Resize(int newSize) {
00213         if (newSize > capacity) {
00214             Reserve(newSize);
00215         }
00216         if (newSize > size) {
00217             for (int i = size; i < newSize; ++i) {
00218                 elements[i] = T();
00219             }
00220         }
00221         size = newSize;
00222     }
00223
00228     void Reserve(int newCapacity) {
00229         if (newCapacity > capacity) {
00230             T* newElements = new T[newCapacity];
00231             for (int i = 0; i < size; ++i) {
00232                 newElements[i] = move(elements[i]);
00233             }
00234             delete[] elements;
00235             elements = newElements;
00236             capacity = newCapacity;
00237         }
00238     }
00239
00243     void ShrinkToFit() {
00244         if (capacity > size) {
00245             T* newElements = new T[size];
00246             for (int i = 0; i < size; ++i) {
00247                 newElements[i] = move(elements[i]);
00248             }
00249             delete[] elements;
00250             elements = newElements;
00251             capacity = size;
00252         }
00253     }
00254
00255
      //*****************************************************************************************************************************
00256
00257     // ELEMENT ACCESS
00258
00265     T& operator [](int index) {                  // v[i] = x
00266         if (index < 0 || index >= size) {
00267             throw out_of_range("Netinkamas indeksas");
00268         }
00269         return elements[index];
00270     }
00271
```

```
00278     const T& operator [](int index) const {      // x = v[i]
00279         if (index < 0 || index >= size) {
00280             throw out_of_range("Netinkamas indeksas");
00281         }
00282         return elements[index];
00283     }
00284
00289     T& Front() {
00290         return elements[0];
00291     }
00292
00297     T& Back() {
00298         return elements[size - 1];
00299     }
00300
00301
      //*****************************************************************************************************************************
00302
00303     // MODIFIERS
00304
00310     void Assign(int n, const T& value) {      // v.Assign(10, 0)
00311         Resize(n);
00312         for (int i = 0; i < size; i++) {
00313             elements[i] = value;
00314         }
00315     }
00316
00321     void PushBack(const T& object) {            // v.PushBack('a')
00322         if (size == capacity) {
00323             Reserve(capacity * 2);
00324         }
00325         elements[size] = object;                // eina i paskutine pozicija ir ideda nauja elementa
00326         size++;                                 //pridejus nauja elementa gale, padidinam ir size
00327     }
00328
00332     void PopBack() {                        // v.PopBack()
00333         if (size > 0) {
00334             size--;
00335         }
00336     }
00337
00344     void Insert(int index, const T& value) {      // v.Insert(0, 'a')
00345         if (index < 0 || index > size) {
00346             throw out_of_range("Netinkamas indeksas");
00347         }
00348         if (size == capacity) {
00349             Resize(capacity * 2);
00350         }
00351         for (int i = size; i > index; i--) {
00352             elements[i] = elements[i - 1];
00353         }
00354         elements[index] = value;
00355         size++;
00356     }
00357
00365     template <typename InputIt>
00366     void Insert(iterator pos, InputIt first, InputIt last) {
00367         int index = pos - Begin();
00368         int numElements = last - first;
00369
00370         if (index < 0 || index > size) {
00371             throw out_of_range("Netinkamas indeksas");
00372         }
00373
00374         if (size + numElements > capacity) {
00375             Reserve((size + numElements) * 2);
00376         }
00377
00378         for (int i = size - 1; i >= index; --i) {
00379             elements[i + numElements] = elements[i];
00380         }
00381
00382         for (int i = 0; i < numElements; ++i) {
00383             elements[index + i] = *(first + i);
00384         }
00385
00386         size += numElements;
00387     }
00388
00394     void Erase(int index) {                       // v.Erase(1)
00395         if (index < 0 || index >= size) {
00396             throw out_of_range("Netinkamas indeksas");
00397         }
00398         for (int i = index; i < size - 1; i++) {
00399             elements[i] = elements[i + 1];
00400         }
00401         size--;
```

```
00402     }
00403
00410     void Erase(iterator first, iterator last) {
00411         if (first < Begin() || last > End() || first > last) {
00412             throw out_of_range("Netinkamas intervalas");
00413         }
00414
00415         int numElements = last - first;
00416         for (iterator it = first; it != End() - numElements; ++it) {
00417             *it = *(it + numElements);
00418         }
00419         size -= numElements;
00420     }
00421
00425     void Clear() {                              // v.Clear()
00426         size = 0;
00427     }
00428
00434     void Swap(Vector<T>& first, Vector<T>& second) {      // Swap(v,w)
00435         swap(first.size, second.size);
00436         swap(first.capacity, second.capacity);
00437         swap(first.elements, second.elements);
00438     }
00439
00440
    //*****************************************************************************************************************************
00441
00442     // NON_MEMBER FUNCTION OVERLOADS
00443
00451     friend bool operator == (const Vector<T>& first, const Vector<T>& second) {
00452         if (first.Size() != second.Size()) {
00453             return false;
00454         }
00455         else {
00456             for (int i = 0; i < first.Size(); i++) {
00457                 if (first.elements[i] != second.elements[i]) {
00458                     return false;
00459                 }
00460             }
00461         }
00462         return true;
00463     }
00464
00471     friend bool operator != (const Vector<T>& first, const Vector<T>& second) {
00472         return !(first == second);
00473      }
00474
00475 };
00476
00477 #endif
```