# 3.0 projektas

Generated by Doxygen 1.10.0

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

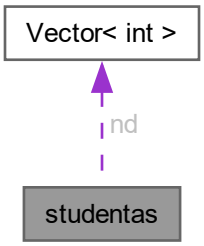
# Chapter 4

# Class Documentation

## 4.1 studentas Struct Reference

`#include <fun.h>`

Collaboration diagram for studentas:




### Public Attributes

- string vardas
- string pavarde
- Vector< int > nd
- int egz
- double balas
- double vid
- double mediana

### 4.1.1 Member Data Documentation

#### 4.1.1.1 balas

`double studentas::balas`

#### 4.1.1.2 egz

```
int studentas::egz
```

#### 4.1.1.3 mediana

```
double studentas::mediana
```

#### 4.1.1.4 nd

```
Vector<int> studentas::nd
```

#### 4.1.1.5 pavarde

```
string studentas::pavarde
```

#### 4.1.1.6 vardas

```
string studentas::vardas
```

#### 4.1.1.7 vid

```
double studentas::vid
```

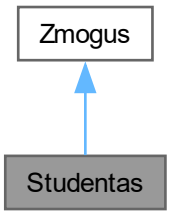
The documentation for this struct was generated from the following file:

- fun.h

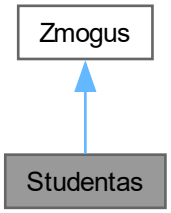
## 4.2 Studentas Class Reference

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

Inheritance diagram for Studentas:



Collaboration diagram for Studentas:




### Public Member Functions

- Studentas ()
- Studentas (const string &vardas, const string &pavarde, const Vector< int > &nd, int egz, double balas, double vid, double mediana)
- ~Studentas ()
- Studentas (const Studentas &other)
- Studentas & operator= (const Studentas &other)
- Studentas (Studentas &&other) noexcept
- Studentas & operator= (Studentas &&other) noexcept
- string getVardas () const override
- string getPavarde () const override
- Vector< int > & getNd ()
- int getEgz () const
- double getBalas () const
- double getVid () const
- double getMediana () const
- void setVardas (const string &vardas)
- void setPavarde (const string &pavarde)
- void setNd (const Vector< int > &nd)
- void setEgz (const int &egz)
- void setBalas (const double &balas)
- void setVid (const double &vid)
- void setMediana (const double &mediana)

## Public Member Functions inherited from Zmogus

- Zmogus ()=default
- Zmogus (const string &vardas, const string &pavarde)
- virtual ~Zmogus ()

### Friends

- istream & operator>> (istream &is, Studentas &student)
- ostream & operator<< (ostream &os, const Studentas &student)

**Additional Inherited Members**

## Protected Attributes inherited from Zmogus

- string vardas_
- string pavarde_

### 4.2.1 Constructor & Destructor Documentation

#### 4.2.1.1 Studentas() [1/4]

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

#### 4.2.1.2 Studentas() [2/4]

```
Studentas::Studentas (
            const string & vardas,
            const string & pavarde,
            const Vector< int > & nd,
            int egz,
            double balas,
            double vid,
            double mediana )  [inline]
```

#### 4.2.1.3 ~Studentas()

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

#### 4.2.1.4 Studentas() [3/4]

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

#### 4.2.1.5 Studentas() [4/4]

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

### 4.2.2 Member Function Documentation

#### 4.2.2.1 getBalas()

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

#### 4.2.2.2 getEgz()

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

#### 4.2.2.3 getMediana()

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

#### 4.2.2.4 getNd()

```
Vector< int > & Studentas::getNd ( )  [inline]
```

#### 4.2.2.5 getPavarde()

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

Implements Zmogus.

#### 4.2.2.6 getVardas()

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

Implements Zmogus.

#### 4.2.2.7 getVid()

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

#### 4.2.2.8 operator=() [1/2]

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

#### 4.2.2.9 operator=() [2/2]

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

#### 4.2.2.10 setBalas()

```
void Studentas::setBalas (
            const double & balas )  [inline]
```

#### 4.2.2.11 setEgz()

```
void Studentas::setEgz (
            const int & egz )  [inline]
```

#### 4.2.2.12 setMediana()

```
void Studentas::setMediana (
            const double & mediana )  [inline]
```

#### 4.2.2.13 setNd()

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

#### 4.2.2.14 setPavarde()

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

Reimplemented from Zmogus.

#### 4.2.2.15 setVardas()

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

Reimplemented from Zmogus.

#### 4.2.2.16 setVid()

```
void Studentas::setVid (
            const double & vid )  [inline]
```

### 4.2.3 Friends And Related Symbol Documentation

#### 4.2.3.1 operator<<

```
ostream & operator<< (
            ostream & os,
            const Studentas & student )  [friend]
```

#### 4.2.3.2 operator>>

```
istream & operator>> (
            istream & is,
            Studentas & student )  [friend]
```

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

- studentas.h

## 4.3 Vector< T > Class Template Reference

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

### Public Types

- typedef T value_type
  *MEMBER TYPES.*
- typedef size_t size_type
- typedef T & reference
- typedef const T & const_reference
- typedef T * iterator
- typedef const T * const_iterator

### Public Member Functions

- Vector ()
- Vector (size_type n, const T &t=T{})
  *fill*
- Vector (const Vector &v)
  *copy konstruktorius*
- template<class InputIterator >
  Vector (InputIterator first, InputIterator last)
  *range konstruktorius*
- Vector (Vector &&v)
  *move konstruktorius*
- Vector (const std::initializer_list< T > il)
  *initializer list konstruktorius*
- ~Vector ()
  *destruktorius*
- operator std::vector< T > () const
  *operator=*
- Vector & operator= (const Vector &other)
  *copy assignment operatorius*
- Vector & operator= (Vector &&other)
  *move assignment operatorius*
- template<class InputIterator >
  void assign (InputIterator first, InputIterator last)
  *assign*

- void assign (size_type n, const value_type &val)
- void assign (std::initializer_list< value_type > il)
- const_reference at (size_type n) const

  *element access*
- T & operator[ ] (size_type n)
- const T & operator[ ] (size_type n) const
- reference at (size_type n)
- reference front ()
- const_reference front () const
- reference back ()
- const_reference back () const
- value_type * data () noexcept
- const value_type * data () const noexcept
- iterator begin ()

  *iterators*
- const_iterator begin () const
- iterator end ()
- const_iterator end () const
- size_type size () const

  *capacity*
- size_type max_size () const
- void resize (size_type sz)
- void resize (size_type sz, const value_type &value)
- size_type capacity () const
- bool empty () const noexcept
- void reserve (size_type n)
- void shrink_to_fit ()
- void clear () noexcept

  *modifiers*
- iterator insert (const_iterator position, const value_type &val)
- iterator insert (iterator position, size_type n, const value_type &val)
- iterator erase (iterator position)
- iterator erase (iterator first, iterator last)
- void push_back (const value_type &t)
- void push_back (value_type &&val)
- void pop_back ()
- void swap (Vector &x)
- bool operator== (const Vector< T > &other) const

  *NON-MEMBER FUNCTIONS.*
- bool operator!= (const Vector< T > &other) const
- bool operator< (const Vector< T > &other) const
- bool operator<= (const Vector< T > &other) const
- bool operator> (const Vector< T > &other) const
- bool operator>= (const Vector< T > &other) const
- void swap (Vector< T > &x, Vector< T > &y)

### 4.3.1 Member Typedef Documentation

#### 4.3.1.1 const_iterator

```
template<typename T >
typedef const T* Vector< T >::const_iterator
```

#### 4.3.1.2 const_reference

```
template<typename T >
typedef const T& Vector< T >::const_reference
```

#### 4.3.1.3 iterator

```
template<typename T >
typedef T* Vector< T >::iterator
```

#### 4.3.1.4 reference

```
template<typename T >
typedef T& Vector< T >::reference
```

#### 4.3.1.5 size_type

```
template<typename T >
typedef size_t Vector< T >::size_type
```

#### 4.3.1.6 value_type

```
template<typename T >
typedef T Vector< T >::value_type
```

MEMBER TYPES.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Vector() [1/6]

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

MEMBER FUNCTIONS default konstruktorius

#### 4.3.2.2 Vector() [2/6]

```
template<typename T >
Vector< T >::Vector (
            size_type n,
            const T & t = T{} )  [inline], [explicit]
```

fill

#### 4.3.2.3 Vector() [3/6]

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

copy konstruktorius

#### 4.3.2.4 Vector() [4/6]

```
template<typename T >
template<class InputIterator >
Vector< T >::Vector (
            InputIterator first,
            InputIterator last )  [inline]
```

range konstruktorius

#### 4.3.2.5 Vector() [5/6]

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

move konstruktorius

#### 4.3.2.6 Vector() [6/6]

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

initializer list konstruktorius

#### 4.3.2.7 ~Vector()

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

destruktorius

### 4.3.3 Member Function Documentation

#### 4.3.3.1 assign() [1/3]

```
template<typename T >
template<class InputIterator >
void Vector< T >::assign (
            InputIterator first,
            InputIterator last )  [inline]
```

assign

#### 4.3.3.2 assign() [2/3]

```
template<typename T >
void Vector< T >::assign (
            size_type n,
            const value_type & val )  [inline]
```

#### 4.3.3.3 assign() [3/3]

```
template<typename T >
void Vector< T >::assign (
            std::initializer_list< value_type > il )  [inline]
```

#### 4.3.3.4 at() [1/2]

```
template<typename T >
reference Vector< T >::at (
            size_type n )  [inline]
```

#### 4.3.3.5 at() [2/2]

```
template<typename T >
const_reference Vector< T >::at (
            size_type n ) const  [inline]
```

element access

#### 4.3.3.6 back() [1/2]

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

#### 4.3.3.7 back() [2/2]

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

#### 4.3.3.8 begin() [1/2]

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

iterators

#### 4.3.3.9 begin() [2/2]

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

#### 4.3.3.10 capacity()

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

#### 4.3.3.11 clear()

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

modifiers

#### 4.3.3.12 data() [1/2]

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

#### 4.3.3.13 data() [2/2]

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

#### 4.3.3.14 empty()

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

#### 4.3.3.15 end() [1/2]

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

#### 4.3.3.16 end() [2/2]

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

#### 4.3.3.17 erase() [1/2]

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

#### 4.3.3.18 erase() [2/2]

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

#### 4.3.3.19 front() [1/2]

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

#### 4.3.3.20 front() [2/2]

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

#### 4.3.3.21 insert() [1/2]

```
template<typename T >
iterator Vector< T >::insert (
            const_iterator position,
            const value_type & val )  [inline]
```

#### 4.3.3.22 insert() [2/2]

```
template<typename T >
iterator Vector< T >::insert (
            iterator position,
            size_type n,
            const value_type & val )  [inline]
```

#### 4.3.3.23 max_size()

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

#### 4.3.3.24 operator std::vector< T >()

```
template<typename T >
Vector< T >::operator std::vector< T > ( ) const  [inline]
```

operator=

#### 4.3.3.25 operator"!=()

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

#### 4.3.3.26 operator<()

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

#### 4.3.3.27 operator<=()

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

#### 4.3.3.28 operator=() [1/2]

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

copy assignment operatorius

#### 4.3.3.29 operator=() [2/2]

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

move assignment operatorius

#### 4.3.3.30 operator==()

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

NON-MEMBER FUNCTIONS.

#### 4.3.3.31 operator>()

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

#### 4.3.3.32 operator>=()

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

#### 4.3.3.33 operator[]() [1/2]

```
template<typename T >
T & Vector< T >::operator[] (
            size_type n )  [inline]
```

#### 4.3.3.34 operator[]() [2/2]

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

#### 4.3.3.35 pop_back()

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

#### 4.3.3.36 push_back() [1/2]

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

#### 4.3.3.37 push_back() [2/2]

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

#### 4.3.3.38 reserve()

```
template<typename T >
void Vector< T >::reserve (
            size_type n )  [inline]
```

#### 4.3.3.39 resize() [1/2]

```
template<typename T >
void Vector< T >::resize (
            size_type sz )  [inline]
```

#### 4.3.3.40 resize() [2/2]

```
template<typename T >
void Vector< T >::resize (
            size_type sz,
            const value_type & value )  [inline]
```

#### 4.3.3.41 shrink_to_fit()

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

#### 4.3.3.42 size()

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

capacity

#### 4.3.3.43 swap() [1/2]

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

#### 4.3.3.44 swap() [2/2]

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

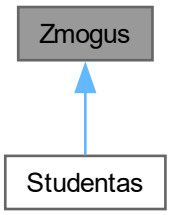
The documentation for this class was generated from the following file:

- vector.h

## 4.4 Zmogus Class Reference

`#include <studentas.h>`

Inheritance diagram for Zmogus:




### Public Member Functions

- Zmogus ()=default
- Zmogus (const string &vardas, const string &pavarde)
- virtual ~Zmogus ()
- virtual void setVardas (const string &vardas)
- virtual void setPavarde (const string &pavarde)
- virtual string getVardas () const =0
- virtual string getPavarde () const =0

### Protected Attributes

- string vardas_
- string pavarde_

### 4.4.1 Constructor & Destructor Documentation

#### 4.4.1.1 Zmogus() [1/2]

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

#### 4.4.1.2 Zmogus() [2/2]

```
Zmogus::Zmogus (
            const string & vardas,
            const string & pavarde )  [inline]
```

#### 4.4.1.3 ~Zmogus()

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

### 4.4.2 Member Function Documentation

#### 4.4.2.1 getPavarde()

```
virtual string Zmogus::getPavarde ( ) const  [pure virtual]
```

Implemented in Studentas.

#### 4.4.2.2 getVardas()

```
virtual string Zmogus::getVardas ( ) const  [pure virtual]
```

Implemented in Studentas.

#### 4.4.2.3 setPavarde()

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

Reimplemented in Studentas.

#### 4.4.2.4 setVardas()

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

Reimplemented in Studentas.

### 4.4.3 Member Data Documentation

#### 4.4.3.1 pavarde_

```
string Zmogus::pavarde_  [protected]
```

#### 4.4.3.2 vardas_

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

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

- studentas.h

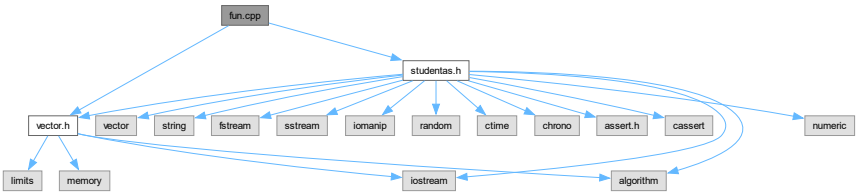
# Chapter 5

# File Documentation

## 5.1 fun.cpp File Reference

```
#include "studentas.h"
#include "vector.h"
```

Include dependency graph for fun.cpp:




### Functions

- bool isValidName (const string &name)
- bool pagalVarda (const Studentas &A, const Studentas &B)
- bool pagalPavarde (const Studentas &A, const Studentas &B)
- bool pagalVidurki (const Studentas &A, const Studentas &B)
- bool pagalMediana (const Studentas &A, const Studentas &B)
- void skaitymas (Vector< Studentas > &A, int n)
- void skaitymasTeksto (Vector< Studentas > &A)
- void skaiciavimas (Vector< Studentas > &A)
- void padalintiStudentus (Vector< Studentas > &A)
- void spausdinti (const Vector< Studentas > &A)
- void spausdintiTeksto (const Vector< Studentas > &A)
- void generavimasPazymiu (Vector< Studentas > &A, int n2)
- void generavimasPazymiuCase2 (Vector< Studentas > &A)
- void generavimasStudentu (Vector< Studentas > &A, int n)
- void generavimasFailo (int kiekis)
- void test_constructor ()
- void test_copy_constructor ()
- void test_move_constructor ()
- void test_copy_assignment ()
- void test_move_assignment ()
- void test_input_operator ()
- void test_output_operator ()

### 5.1.1 Function Documentation

#### 5.1.1.1 generavimasFailo()

```
void generavimasFailo (
            int kiekis )
```

#### 5.1.1.2 generavimasPazymiu()

```
void generavimasPazymiu (
            Vector< Studentas > & A,
            int n2 )
```

#### 5.1.1.3 generavimasPazymiuCase2()

```
void generavimasPazymiuCase2 (
            Vector< Studentas > & A )
```

#### 5.1.1.4 generavimasStudentu()

```
void generavimasStudentu (
            Vector< Studentas > & A,
            int n )
```

#### 5.1.1.5 isValidName()

```
bool isValidName (
            const string & name )
```

#### 5.1.1.6 padalintiStudentus()

```
void padalintiStudentus (
            Vector< Studentas > & A )
```

#### 5.1.1.7 pagalMediana()

```
bool pagalMediana (
            const Studentas & A,
            const Studentas & B )
```

#### 5.1.1.8 pagalPavarde()

```
bool pagalPavarde (
            const Studentas & A,
            const Studentas & B )
```

#### 5.1.1.9 pagalVarda()

```
bool pagalVarda (
            const Studentas & A,
            const Studentas & B )
```

#### 5.1.1.10 pagalVidurki()

```
bool pagalVidurki (
            const Studentas & A,
            const Studentas & B )
```

#### 5.1.1.11 skaiciavimas()

```
void skaiciavimas (
            Vector< Studentas > & A )
```

#### 5.1.1.12 skaitymas()

```
void skaitymas (
            Vector< Studentas > & A,
            int n )
```

#### 5.1.1.13 skaitymasTeksto()

```
void skaitymasTeksto (
            Vector< Studentas > & A )
```

#### 5.1.1.14 spausdinti()

```
void spausdinti (
            const Vector< Studentas > & A )
```

#### 5.1.1.15 spausdintiTeksto()

```
void spausdintiTeksto (
            const Vector< Studentas > & A )
```

#### 5.1.1.16 test_constructor()

```
void test_constructor ( )
```

#### 5.1.1.17 test_copy_assignment()

```
void test_copy_assignment ( )
```

#### 5.1.1.18 test_copy_constructor()

```
void test_copy_constructor ( )
```

#### 5.1.1.19 test_input_operator()

```
void test_input_operator ( )
```

#### 5.1.1.20 test_move_assignment()

```
void test_move_assignment ( )
```

#### 5.1.1.21 test_move_constructor()

```
void test_move_constructor ( )
```

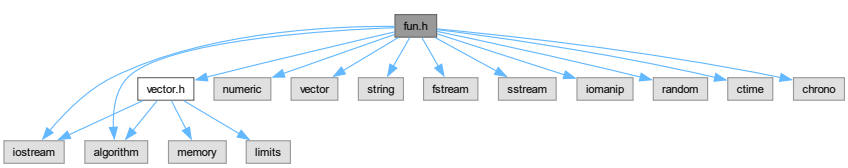
#### 5.1.1.22 test_output_operator()

```
void test_output_operator ( )
```

## 5.2 fun.h File Reference

```
#include <iostream>
#include <numeric>
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <random>
#include <ctime>
#include <chrono>
#include "vector.h"
```

Include dependency graph for fun.h:



### Classes

- struct studentas

### Functions

- void skaitymas (Vector< studentas > &A, int n)
- void skaitymasTeksto (Vector< studentas > &A)
- bool pagalVarda (const studentas &A, const studentas &B)
- bool pagalPavarde (const studentas &A, const studentas &B)
- bool pagalVidurki (const studentas &A, const studentas &B)
- bool pagalMediana (const studentas &A, const studentas &B)
- void spausdinti (const Vector< studentas > &A)
- void spausdintiTeksto (const Vector< studentas > &A)
- void skaiciavimas (Vector< studentas > &A)
- void padalintiStudentus (Vector< studentas > &A)
- void generavimasPazymiu (Vector< studentas > &A, int n)
- void generavimasPazymiuCase2 (Vector< studentas > &A)
- void generavimasStudentu (Vector< studentas > &A, int n)
- void generavimasFailo (Vector< studentas > &A, int n)
- bool isValidName (const string &name)

### Variables

- const int MAX_ND_SIZE = 500
- const int MAX_STUDENTS = 500

## 5.2.1 Function Documentation

#### 5.2.1.1 generavimasFailo()

```
void generavimasFailo (
            Vector< studentas > & A,
            int n )
```

#### 5.2.1.2 generavimasPazymiu()

```
void generavimasPazymiu (
            Vector< studentas > & A,
            int n )
```

#### 5.2.1.3 generavimasPazymiuCase2()

```
void generavimasPazymiuCase2 (
            Vector< studentas > & A )
```

#### 5.2.1.4 generavimasStudentu()

```
void generavimasStudentu (
            Vector< studentas > & A,
            int n )
```

#### 5.2.1.5 isValidName()

```
bool isValidName (
            const string & name )
```

#### 5.2.1.6 padalintiStudentus()

```
void padalintiStudentus (
            Vector< studentas > & A )
```

#### 5.2.1.7 pagalMediana()

```
bool pagalMediana (
            const studentas & A,
            const studentas & B )
```

#### 5.2.1.8 pagalPavarde()

```
bool pagalPavarde (
            const studentas & A,
            const studentas & B )
```

#### 5.2.1.9 pagalVarda()

```
bool pagalVarda (
            const studentas & A,
            const studentas & B )
```

#### 5.2.1.10 pagalVidurki()

```
bool pagalVidurki (
            const studentas & A,
            const studentas & B )
```

#### 5.2.1.11 skaiciavimas()

```
void skaiciavimas (
            Vector< studentas > & A )
```

#### 5.2.1.12 skaitymas()

```
void skaitymas (
            Vector< studentas > & A,
            int n )
```

#### 5.2.1.13 skaitymasTeksto()

```
void skaitymasTeksto (
            Vector< studentas > & A )
```

#### 5.2.1.14 spausdinti()

```
void spausdinti (
            const Vector< studentas > & A )
```

#### 5.2.1.15 spausdintiTeksto()

```
void spausdintiTeksto (
            const Vector< studentas > & A )
```

### 5.2.2 Variable Documentation

#### 5.2.2.1 MAX_ND_SIZE

```
const int MAX_ND_SIZE = 500
```

#### 5.2.2.2 MAX_STUDENTS

```
const int MAX_STUDENTS = 500
```

## 5.3 fun.h

Go to the documentation of this file.

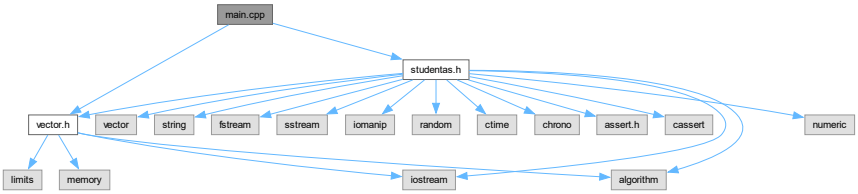
```
00001 #ifndef LABAS
00002 #define LABAS
00003 #include <iostream>
00004 #include <numeric>
00005 #include <vector>
00006 #include <string>
00007 #include <algorithm>
00008 #include <fstream>
00009 #include <sstream>
00010 #include <iomanip>
00011 #include <random>
00012 #include <ctime>
00013 #include <chrono>
00014 #include "vector.h"
00015 using namespace std;
00016
00017 const int MAX_ND_SIZE = 500;
00018 const int MAX_STUDENTS = 500;
```

```
00019
00020 struct studentas
00021 {
00022     string vardas;
00023     string pavarde;
00024      Vector <int> nd;
00025      int egz;
00026      double balas;
00027      double vid;
00028      double mediana;
00029 };
00030
00031 void skaitymas (Vector <studentas> & A, int n);
00032 void skaitymasTeksto (Vector <studentas> & A);
00033 bool pagalVarda(const studentas & A, const studentas &B);
00034 bool pagalPavarde(const studentas &A, const studentas &B);
00035 bool pagalVidurki(const studentas &A, const studentas &B);
00036 bool pagalMediana(const studentas &A, const studentas &B);
00037 void spausdinti (const Vector <studentas> & A);
00038 void spausdintiTeksto(const Vector <studentas> & A);
00039 void skaiciavimas (Vector <studentas> & A);
00040 void padalintiStudentus(Vector <studentas> &A);
00041 void generavimasPazymiu (Vector <studentas> & A, int n);
00042 void generavimasPazymiuCase2 (Vector <studentas> & A);
00043 void generavimasStudentu (Vector <studentas> & A, int n);
00044 void generavimasFailo (Vector <studentas> & A, int n);
00045 bool isValidName(const string &name);
00046 #endif
```

## 5.4 main.cpp File Reference

```
#include "studentas.h"
#include "vector.h"
```

Include dependency graph for main.cpp:




### Functions

- int main ()

### 5.4.1 Function Documentation

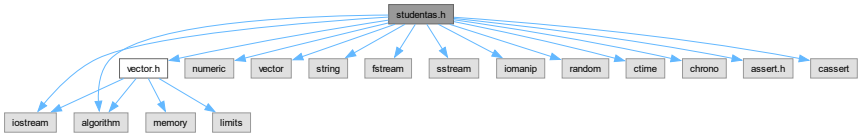
#### 5.4.1.1 main()

```
int main ( )
```

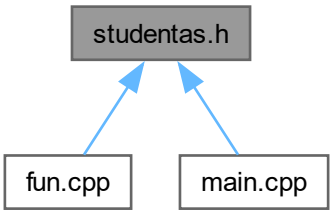
## 5.5 studentas.h File Reference

```
#include <iostream>
#include <numeric>
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <random>
#include <ctime>
#include <chrono>
#include <assert.h>
#include <cassert>
#include "vector.h"
```

Include dependency graph for studentas.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class Zmogus
- class Studentas

### Functions

- void skaitymas (Vector< Studentas > &, int)
- void skaitymasTeksto (Vector< Studentas > &)
- bool pagalVarda (const Studentas &, const Studentas &)

- bool pagalPavarde (const Studentas &, const Studentas &)
- bool pagalVidurki (const Studentas &, const Studentas &)
- bool pagalMediana (const Studentas &, const Studentas &)
- void spausdinti (const Vector< Studentas > &)
- void spausdintiTeksto (const Vector< Studentas > &)
- void skaiciavimas (Vector< Studentas > &)
- void padalintiStudentus (Vector< Studentas > &)
- void generavimasPazymiu (Vector< Studentas > &, int n)
- void generavimasPazymiuCase2 (Vector< Studentas > &)
- void generavimasStudentu (Vector< Studentas > &, int)
- void generavimasFailo (Vector< Studentas > &, int)
- bool isValidName (const string &)
- void test_constructor ()
- void test_copy_constructor ()
- void test_move_constructor ()
- void test_copy_assignment ()
- void test_move_assignment ()
- void test_input_operator ()
- void test_output_operator ()

**Variables**

- const int MAX_ND_SIZE = 500
- const int MAX_STUDENTS = 500

### 5.5.1 Function Documentation

#### 5.5.1.1 generavimasFailo()

```
void generavimasFailo (
            Vector< Studentas > & ,
            int  )
```

#### 5.5.1.2 generavimasPazymiu()

```
void generavimasPazymiu (
            Vector< Studentas > & A,
            int n )
```

#### 5.5.1.3 generavimasPazymiuCase2()

```
void generavimasPazymiuCase2 (
            Vector< Studentas > & A )
```

#### 5.5.1.4 generavimasStudentu()

```
void generavimasStudentu (
            Vector< Studentas > & A,
            int n )
```

#### 5.5.1.5 isValidName()

```
bool isValidName (
            const string & name )
```

#### 5.5.1.6 padalintiStudentus()

```
void padalintiStudentus (
            Vector< Studentas > & A )
```

#### 5.5.1.7 pagalMediana()

```
bool pagalMediana (
            const Studentas & A,
            const Studentas & B )
```

#### 5.5.1.8 pagalPavarde()

```
bool pagalPavarde (
            const Studentas & A,
            const Studentas & B )
```

#### 5.5.1.9 pagalVarda()

```
bool pagalVarda (
            const Studentas & A,
            const Studentas & B )
```

#### 5.5.1.10 pagalVidurki()

```
bool pagalVidurki (
            const Studentas & A,
            const Studentas & B )
```

#### 5.5.1.11 skaiciavimas()

```
void skaiciavimas (
            Vector< Studentas > & A )
```

#### 5.5.1.12 skaitymas()

```
void skaitymas (
            Vector< Studentas > & A,
            int n )
```

#### 5.5.1.13 skaitymasTeksto()

```
void skaitymasTeksto (
            Vector< Studentas > & A )
```

#### 5.5.1.14 spausdinti()

```
void spausdinti (
            const Vector< Studentas > & A )
```

#### 5.5.1.15 spausdintiTeksto()

```
void spausdintiTeksto (
            const Vector< Studentas > & A )
```

#### 5.5.1.16 test_constructor()

```
void test_constructor ( )
```

#### 5.5.1.17 test_copy_assignment()

```
void test_copy_assignment ( )
```

#### 5.5.1.18 test_copy_constructor()

```
void test_copy_constructor ( )
```

#### 5.5.1.19 test_input_operator()

```
void test_input_operator ( )
```

#### 5.5.1.20 test_move_assignment()

```
void test_move_assignment ( )
```

#### 5.5.1.21 test_move_constructor()

```
void test_move_constructor ( )
```

#### 5.5.1.22 test_output_operator()

```
void test_output_operator ( )
```

### 5.5.2 Variable Documentation

#### 5.5.2.1 MAX_ND_SIZE

```
const int MAX_ND_SIZE = 500
```

#### 5.5.2.2 MAX_STUDENTS

```
const int MAX_STUDENTS = 500
```

## 5.6 studentas.h

Go to the documentation of this file.

```
00001 #ifndef KA
00002 #define KA
00003
00004 #include <iostream>
00005 #include <numeric>
00006 #include <vector>
00007 #include <string>
00008 #include <algorithm>
00009 #include <fstream>
00010 #include <sstream>
00011 #include <iomanip>
00012 #include <random>
00013 #include <ctime>
00014 #include <chrono>
00015 #include <assert.h>
00016 #include <cassert>
00017 #include "vector.h"
00018 using namespace std;
00019
00020 const int MAX_ND_SIZE = 500;
00021 const int MAX_STUDENTS = 500;
00022
00023 class Zmogus {
00024     protected:
00025     string vardas_;
00026     string pavarde_;
00027     public:
00028     Zmogus() = default;
00029     Zmogus(const string& vardas, const string& pavarde) : vardas_(vardas), pavarde_(pavarde) {}
00030     virtual ~Zmogus() {}
00031
00032     virtual void setVardas(const string& vardas) { vardas_ = vardas; }
00033     virtual void setPavarde(const string& pavarde) { pavarde_ = pavarde; }
00034     virtual string getVardas() const = 0; // { return vardas_; }
00035     virtual string getPavarde() const = 0; // { return pavarde_; }
00036 };
00037
00038 class Studentas : public Zmogus {
00039 private:
00040     Vector <int> nd_;
00041     int egz_;
00042     double balas_;
00043     double vid_;
00044     double mediana_;
00045
00046 public:
00047     Studentas() : egz_(0), balas_(0), vid_(0), mediana_(0) {} // default konstruktorius
00048
00049     Studentas(const string& vardas, const string& pavarde, const Vector<int>& nd, int egz, double
      balas, double vid, double mediana) // konstruktorius
00050         : Zmogus(vardas, pavarde),  nd_(nd), egz_(egz), balas_(balas), vid_(vid), mediana_(mediana) {}
00051     ~Studentas() {nd_.clear();} // destruktorius
00052
00053     Studentas(const Studentas& other)
00054         : Zmogus(other.getVardas(), other.getPavarde()), nd_(other.nd_), egz_(other.egz_),
      balas_(other.balas_), vid_(other.vid_), mediana_(other.mediana_) {} // copy constructor
00055
00056     Studentas& operator=(const Studentas& other) { // copy assignment operatorius
00057         if (this != &other) {
00058             Zmogus::setVardas(other.getVardas());
```

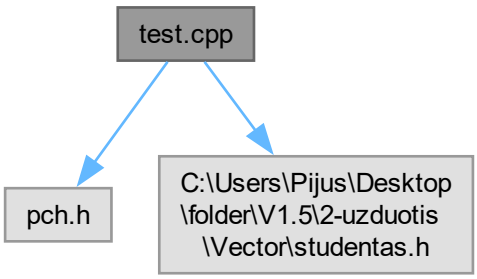
```
00059             Zmogus::setPavarde(other.getPavarde());
00060             nd_ = other.nd_;
00061             egz_ = other.egz_;
00062             balas_ = other.balas_;
00063             vid_ = other.vid_;
00064             mediana_ = other.mediana_;
00065         }
00066         return *this;
00067     }
00068
00069     Studentas(Studentas&& other) noexcept
00070       {Zmogus::setVardas(other.getVardas());
00071        Zmogus::setPavarde(other.getPavarde());
00072        nd_ = other.nd_;
00073        egz_ = other.egz_;
00074        balas_ = other.balas_;
00075        vid_ = other.vid_;
00076        mediana_ = mediana_;
00077
00078        other.setVardas("");
00079        other.setPavarde("");
00080        other.nd_.clear();
00081        other.egz_ = 0;
00082        other.balas_ = 0;
00083        other.vid_ = 0;
00084        other.mediana_ = 0;
00085
00086        //cout « "Move konstruktorius suveike" « endl;
00087     };
00088
00089     Studentas& operator=(Studentas&& other) noexcept { // move assignment operatorius
00090         if (this != &other) {
00091             Zmogus::setVardas(move(other.getVardas()));
00092             Zmogus::setPavarde(move(other.getPavarde()));
00093             nd_ = move(other.nd_);
00094             egz_ = other.egz_;
00095             balas_ = other.balas_;
00096             vid_ = other.vid_;
00097             mediana_ = other.mediana_;
00098             other.egz_ = 0;
00099             other.balas_ = 0;
00100             other.vid_ = 0;
00101             other.mediana_ = 0;
00102         }
00103         return *this;
00104     }
00105
00106     // Getter and setter functions
00107     inline string getVardas() const override{return vardas_;}
00108     inline string getPavarde() const override {return pavarde_;}
00109     inline Vector<int>& getNd() { return nd_; }
00110     inline int getEgz() const { return egz_; }
00111     inline double getBalas() const { return balas_; }
00112     inline double getVid() const { return vid_; }
00113     inline double getMediana() const { return mediana_; }
00114     inline void setVardas(const string& vardas) { vardas_ = vardas; }
00115     inline void setPavarde(const string& pavarde) { pavarde_ = pavarde; }
00116     inline void setNd(const Vector<int>& nd) { nd_ = nd; }
00117     inline void setEgz(const int& egz) { egz_ = egz; }
00118     inline void setBalas(const double& balas) { balas_ = balas; }
00119     inline void setVid(const double& vid) { vid_ = vid; }
00120     inline void setMediana(const double& mediana) { mediana_ = mediana; }
00121
00122      friend istream& operator»(istream& is, Studentas& student) { // input metodas
00123         cout « "Enter student's name and surname: ";
00124         is » student.vardas_ » student.pavarde_;
00125
00126         cout « "Enter student's exam score: ";
00127         is » student.egz_;
00128
00129         cout « "Enter student's homework scores (enter -1 to stop): ";
00130         int score;
00131         while (is » score && score != -1) {
00132             student.nd_.push_back(score);
00133         }
00134
00135
00136         return is;
00137     }
00138
00139     friend ostream& operator«(ostream& os, const Studentas& student) { // output metodas
00140         os « "Name: " « student.vardas_ « " " « student.pavarde_ « endl;
00141         os « "Exam score: " « student.egz_ « endl;
00142         os « "Homework scores: ";
00143         for (int score : student.nd_) {
00144             os « score « " ";
00145         }
```

```
00146         os « endl;
00147         os « "Final score: " « student.balas_ « endl;
00148         os « "Average: " « student.vid_ « endl;
00149         os « "Median: " « student.mediana_ « endl;
00150         return os;
00151     }
00152 };
00153
00154 void skaitymas(Vector<Studentas>&, int);
00155 void skaitymasTeksto(Vector<Studentas>&);
00156 bool pagalVarda(const Studentas&, const Studentas&);
00157 bool pagalPavarde(const Studentas&, const Studentas&);
00158 bool pagalVidurki(const Studentas&, const Studentas&);
00159 bool pagalMediana(const Studentas&, const Studentas&);
00160 void spausdinti(const Vector<Studentas>&);
00161 void spausdintiTeksto(const Vector<Studentas>&);
00162 void skaiciavimas(Vector<Studentas>&);
00163 void padalintiStudentus(Vector<Studentas>&);
00164 void generavimasPazymiu(Vector<Studentas>&, int n);
00165 void generavimasPazymiuCase2(Vector<Studentas>&);
00166 void generavimasStudentu(Vector<Studentas>&, int);
00167 void generavimasFailo(Vector<Studentas>&, int);
00168 bool isValidName(const string&);
00169 void test_constructor ();
00170 void test_copy_constructor ();
00171 void test_move_constructor ();
00172 void test_copy_assignment ();
00173 void test_move_assignment ();
00174 void test_input_operator ();
00175 void test_output_operator();
00176 #endif
```

## 5.7 test.cpp File Reference

```
#include "pch.h"
#include "C:\Users\Pijus\Desktop\folder\V1.5\2-uzduotis\Vector\studentas.h"
```

Include dependency graph for test.cpp:




### Functions

- TEST (StudentTest, ConstructorTest)
- TEST (StudentTest, CopyConstructor)
- TEST (StudentTest, MoveConstructor)
- TEST (StudentTest, CopyAssignment)
- TEST (StudentTest, MoveAssignment)

### 5.7.1 Function Documentation

#### 5.7.1.1 TEST() [1/5]

```
TEST (
            StudentTest ,
            ConstructorTest  )
```

#### 5.7.1.2 TEST() [2/5]

```
TEST (
            StudentTest ,
            CopyAssignment  )
```

#### 5.7.1.3 TEST() [3/5]

```
TEST (
            StudentTest ,
            CopyConstructor  )
```

#### 5.7.1.4 TEST() [4/5]

```
TEST (
            StudentTest ,
            MoveAssignment  )
```

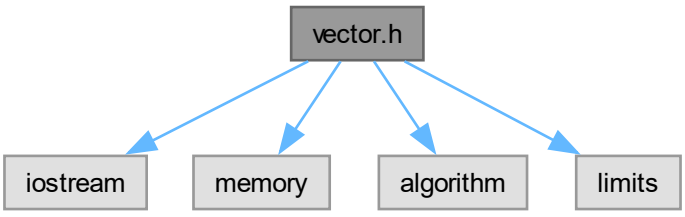
#### 5.7.1.5 TEST() [5/5]

```
TEST (
            StudentTest ,
            MoveConstructor  )
```

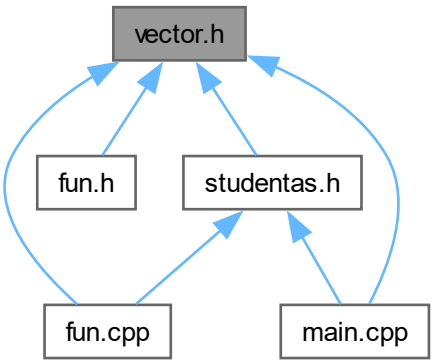
## 5.8 vector.h File Reference

```
#include <iostream>
#include <memory>
#include <algorithm>
#include <limits>
```

Include dependency graph for vector.h:



This graph shows which files directly or indirectly include this file:




### Classes

- class Vector< T >

## 5.9 vector.h

Go to the documentation of this file.

```
00001 #ifndef TA
00002 #define TA
00003
00004 #include <iostream>
00005 #include <memory>
00006 #include <algorithm>
00007 #include <limits>
00008
00009 template <typename T>
00010
00011 class Vector{
00013 public:
00014     typedef T value_type;
00015     typedef size_t size_type;
00016     typedef T& reference;
00017     typedef const T& const_reference;
00018     typedef T* iterator;
00019     typedef const T* const_iterator;
00020
00023     Vector() {create ();}
00025     explicit Vector (size_type n, const T& t = T{}) {create (n, t);}
00027     Vector (const Vector& v) {create(v.begin(), v.end());}
00028     template <class InputIterator>
00030     Vector (InputIterator first, InputIterator last) {create(first, last);}
00032     Vector (Vector && v) {create(); swap(v); v.uncreate();}
00034     Vector (const std::initializer_list <T> il) {create(il.begin(), il.end());}
00036     ~Vector(){uncreate();}
00038     operator std::vector <T> () const{
00039          return std::vector<T>(dat, avail);
00040     }
00042     Vector& operator = (const Vector& other) {
00043         if(this != &other){
00044             uncreate();
00045             create(other.begin(), other.end());
00046         }
00047         return *this;
00048     };
00050     Vector& operator = (Vector&& other){
00051         if (this != &other){
```

```
00052             create(other.begin(), other.end());
00053             uncreate();
00054         }
00055         return *this;
00056     };
00058     template <class InputIterator>
00059     void assign (InputIterator first, InputIterator last){
00060         uncreate();
00061         create(first, last);
00062     }
00063     void assign (size_type n, const value_type& val){
00064         uncreate();
00065         create(n, val);
00066     }
00067     void assign (std::initializer_list <value_type> il){
00068         uncreate ();
00069         create (il);
00070     }
00072     const_reference at (size_type n) const {
00073         if (n >= size() || n < 0){
00074             throw std::out_of_range("Index Out of Range");
00075             return dat[n];
00076         }
00077     }
00078     T& operator[] (size_type n) {return dat[n];}
00079     const T& operator [] (size_type n) const { return dat[n];}
00080     reference at (size_type n){
00081         if (n >= size() || n < 0){
00082             throw std::out_of_range("Index Out of Range");
00083             return dat[n];
00084         }
00085     }
00086     reference front(){
00087         return dat[0];
00088     };
00089     const_reference front() const{
00090         return dat[0];
00091     }
00092     reference back(){
00093         return dat[size() - 1];
00094     }
00095     const_reference back() const{
00096         return dat[size() - 1];
00097     }
00098     value_type* data() noexcept{
00099         return dat;
00100     }
00101     const value_type* data() const noexcept{
00102         return dat;
00103     }
00105     iterator begin() {return dat;}
00106     const_iterator begin() const {return dat;}
00107     iterator end() {return avail;}
00108     const_iterator end() const{return avail;}
00110     size_type size() const{return avail-dat;}
00111     size_type max_size() const {return std::numeric_limits<size_type>::max();}
00112     void resize(size_type sz){
00113         if (sz < size()) {
00114             iterator it = dat + sz;
00115             while (it != avail) {
00116                 alloc.destroy(it++);
00117             }
00118                 avail = dat + sz;
00119             }
00120         else if (sz > capacity()) {
00121             grow(sz);
00122             std::uninitialized_fill(avail, dat + sz, value_type());
00123             avail = dat + sz;
00124             }
00125         else if (sz > size()) {
00126             std::uninitialized_fill(avail, dat + sz, value_type());
00127             avail = dat + sz;
00128             }
00129         }
00130      void resize(size_type sz, const value_type& value) {
00131         if (sz > capacity()) {
00132             grow(sz);
00133         }
00134
00135         if (sz > size()) {
00136             insert(end(), sz - size(), value);
00137         } else if (sz < size()) {
00138             avail = dat + sz;
00139         }
00140     }
00141     size_type capacity() const {return limit-dat;}
00142     bool empty() const noexcept { return size() == 0;}
```

```
00143     void reserve (size_type n) {
00144         if (n > capacity()) {
00145             grow(n);
00146         }
00147     }
00148     void shrink_to_fit(){
00149         if (limit > avail)
00150         limit = avail;
00151     }
00153     void clear () noexcept {uncreate();}
00154     iterator insert (const_iterator position, const value_type& val){return insert(position, 1, val);}
00155     iterator insert(iterator position, size_type n, const value_type& val) {
00156         if (position < begin() || position > end()) {
00157             throw std::out_of_range("Index out of range");
00158         }
00159         if (avail + n > limit) {
00160             size_type index = position - begin();
00161             grow(n);
00162             position = begin() + index;
00163         }
00164         for (iterator it = end() + n - 1; it != position + n - 1; --it) {
00165             *it = std::move(*(it - n));
00166         }
00167         std::uninitialized_fill(position, position + n, val);
00168         avail += n;
00169 
00170         return position;
00171         }
00172     iterator erase(iterator position) {
00173         if (position < dat || position > avail) {
00174             throw std::out_of_range("Index out of range");
00175         }
00176         std::move(position + 1, avail, position);
00177         alloc.destroy(avail - 1);
00178         --avail;
00179 
00180         return position;
00181     }
00182 
00183     iterator erase(iterator first, iterator last) {
00184         iterator new_available = std::uninitialized_copy(last, avail, first);
00185 
00186         iterator it = avail;
00187         while (it != new_available) {
00188             alloc.destroy(--it);
00189         }
00190 
00191         avail= new_available;
00192         return last;
00193     }
00194      void push_back (const value_type& t) {
00195             if (avail==limit)
00196                 grow();
00197             unchecked_append(t);
00198         }
00199 
00200         void push_back (value_type&& val) {
00201             if (avail == limit)
00202                 grow();
00203             unchecked_append(val);
00204         }
00205 
00206         void pop_back() {
00207             if (avail != dat)
00208                 alloc.destroy(--avail);
00209         }
00210 
00211         void swap(Vector& x) {
00212             std::swap(dat, x.dat);
00213             std::swap(avail, x.avail);
00214             std::swap(limit, x.limit);
00215         }
00216 
00218         bool operator== (const Vector <T>& other) const{
00219             if (size() != other.size()){
00220                 return false;
00221             }
00222             return std::equal(begin(), end(), other.begin());
00223         }
00224         bool operator!= (const Vector <T>& other) const{
00225             return !(*this == other);
00226         }
00227         bool operator< (const Vector <T>& other) const{
00228             return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
00229         }
00230         bool operator<= (const Vector <T>& other) const{
00231             return !(other < *this);
```

```
00232         }
00233         bool operator> (const Vector <T>& other) const{
00234             return std::lexicographical_compare(other.begin(), other.end(), begin(), end());
00235         }
00236         bool operator>= (const Vector <T>& other) const{
00237             return !(other > *this);
00238         }
00239         void swap (Vector<T>& x, Vector<T>& y) {
00240         std::swap(x,y);
00241         }
00242 private:
00243     iterator dat;
00244     iterator avail;
00245     iterator limit;
00246     std::allocator<T> alloc;
00247 
00248      void create() {dat = avail = limit = nullptr;}
00249         void create (size_type n, const T& val) {
00250             dat = alloc.allocate(n);
00251             limit = avail = dat + n;
00252             std::uninitialized_fill(dat, limit, val);
00253         }
00254         void create(const_iterator i, const_iterator j) {
00255             dat = alloc.allocate(j - i);
00256             limit = avail = std::uninitialized_copy(i, j, dat);
00257         }
00258         void uncreate(){
00259             if (dat) {
00260                 iterator it = avail;
00261                 while (it != dat) {
00262                     alloc.destroy(--it);
00263                 }
00264             alloc.deallocate(dat, limit - dat);
00265             }
00266             dat = limit = avail = nullptr;
00267         }
00268     void grow(size_type new_capacity = 1) {
00269         size_type new_size = std::max(new_capacity, 2 * capacity());
00270         iterator new_data = alloc.allocate(new_size);
00271         iterator new_avail = std::uninitialized_copy(dat, avail, new_data);
00272         uncreate();
00273         dat = new_data;
00274         avail = new_avail;
00275         limit = dat + new_size;
00276     }
00277 
00278     void unchecked_append(const T& val) {
00279         alloc.construct(avail++, val);
00280     }
00281 };
00282 
00283 
00284 #endif
```

# Index