My Project

Generated by Doxygen 1.11.0

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

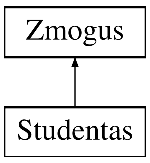
Class Documentation

4.1 Studentas Class Reference

Class representing a student.

```
#include <Studentas.h>
```

Inheritance diagram for Studentas:



Public Member Functions

- Studentas ()

    Default constructor for Studentas class.
- Studentas (std::istream &is)

    Constructor that reads student data from an input stream.
- std::string getVardas () const
- std::string getPavarde () const
- double getEgzaminoRez () const
- Vector< double > & getNamudarbuRez ()
- void setVardas (const std::string &vardas)
- void setPavarde (const std::string &pavarde)
- void setEgzaminoRez (double egzaminorez)
- void addGrade (double grade)
- std::istream & readStudent (std::istream &)

    Reads student data from an input stream.
- void calculateFinalGrades ()

    Calculates the final grades for the student.
- ∼Studentas ()

    ClassDestructor for Studentas class.
- Studentas (const Studentas &other)

Copy constructor for Studentas class.

- Studentas & operator= (const Studentas &other)

  Copy assignment operator implementation.
- Studentas (Studentas &&other) noexcept

  Move constructor implementation.
- Studentas & operator= (Studentas &&other) noexcept

  Move assignment operator implementationconstructor implementation.

Public Member Functions inherited from Zmogus

- virtual ∼Zmogus ()=default

Public Attributes

- double namudarburezsuma_
- double vidurkis_
- double galutinisbalasvidurkis_
- double mediana_
- double galutinisbalasmediana_

4.1.1 Detailed Description

Class representing a student.

This class stores information about a student, including their name, grades for homework and exams, and calculated final grades.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Studentas() `[1/4]`

```
Studentas::Studentas ()
```

Default constructor for Studentas class.

Default constructor.

Initializes a new Studentas object with default values.

4.1.2.2 Studentas() `[2/4]`

```
Studentas::Studentas (
            std::istream & is)
```

Constructor that reads student data from an input stream.

Constructor to initialize a student with input stream.

Parameters

is	The input stream to read from.

4.1.2.3 ∼Studentas()

```
Studentas::∼Studentas ()
```

ClassDestructor for Studentas class.

Destructor.

Destructor for the Studentas class.

4.1.2.4 Studentas() `[3/4]`

```
Studentas::Studentas (
            const Studentas & other)
```

Copy constructor for Studentas class.

Copy constructor.

Constructor for the Studentas class.

4.1.2.5 Studentas() `[4/4]`

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

Move constructor implementation.

Move constructor.

Move constructor implementation.

4.1.3 Member Function Documentation

4.1.3.1 addGrade()

```
void Studentas::addGrade (
            double grade)  [inline]
```

Add a grade to the homework grades of the student.

4.1.3.2 calculateFinalGrades()

```
void Studentas::calculateFinalGrades ()
```

Calculates the final grades for the student.

Calculate final grades for the student.

This function calculates both the final average grade and the final median grade based on the student's homework and exam results.

4.1.3.3 getEgzaminoRez()

```
double Studentas::getEgzaminoRez () const  [inline], [virtual]
```

Get the exam grade of the student.

Implements Zmogus.

4.1.3.4 getNamudarbuRez()

```
Vector< double > & Studentas::getNamudarbuRez ()  [inline], [virtual]
```

Get the grades of homework for the student.

Implements Zmogus.

4.1.3.5 getPavarde()

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

Get the last name of the student.

Implements Zmogus.

4.1.3.6 getVardas()

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

Get the first name of the student.

Implements Zmogus.

4.1.3.7 operator=() [1/2]

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

Copy assignment operator implementation.

Copy assignment operator.

Copy assignment operator implementation.

4.1.3.8 operator=() [2/2]

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

Move assignment operator implementationconstructor implementation.

Move assignment operator.

Move assignment operator implementation.

4.1.3.9 readStudent()

```
std::istream & Studentas::readStudent (
             std::istream & is)
```

Reads student data from an input stream.

Read student data from an input stream.

Parameters

is	The input stream to read from.

Returns

istream& The input stream after reading the student data.

4.1.3.10 setEgzaminoRez()

```
void Studentas::setEgzaminoRez (
             double egzaminorez)  [inline]
```

Set the exam grade of the student.

4.1.3.11 setPavarde()

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

Set the last name of the student.

4.1.3.12 setVardas()

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

Set the first name of the student.

4.1.4 Member Data Documentation

4.1.4.1 galutinisbalasmediana_

```
double Studentas::galutinisbalasmediana_
```

Final grade calculated using median method.

4.1.4.2 galutinisbalasvidurkis_

```
double Studentas::galutinisbalasvidurkis_
```

Final grade calculated using average method.

4.1.4.3 mediana_

```
double Studentas::mediana_
```

Median final grade.

4.1.4.4 namudarburezsuma_

```
double Studentas::namudarburezsuma_
```

Sum of homework grades.

4.1.4.5 vidurkis_

```
double Studentas::vidurkis_
```

Average final grade.

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

- src/Studentas.h
- src/Studentas.cpp

4.2 Vector< T, Allocator > Class Template Reference

A dynamically resizing array similar to std::vector.

```
#include <Vektorius.h>
```

Public Types

- using value_type = T
- using allocator_type = Allocator
- using size_type = typename std::allocator_traits< Allocator >::size_type
- using difference_type = typename std::allocator_traits< Allocator >::difference_type
- using reference = T&
- using const_reference = const T&
- using pointer = typename std::allocator_traits< Allocator >::pointer
- using const_pointer = typename std::allocator_traits< Allocator >::const_pointer
- using iterator = T∗
- using const_iterator = const T∗
- using reverse_iterator = std::reverse_iterator< iterator >
- using const_reverse_iterator = std::reverse_iterator< const_iterator >

Public Member Functions

- Vector ()

  Constructs an empty vector.
- Vector (size_type count, const T &value=T(), const Allocator &alloc=Allocator())

  Constructs the vector with a specified number of elements, each initialized with a given value.
- Vector (const Vector &other)

  Constructs the vector by copying elements from another vector.
- Vector (Vector &&other) noexcept

  Constructs the vector by moving elements from another vector.
- Vector (std::initializer_list< T > init, const Allocator &alloc=Allocator())

  Constructs the vector with the elements from the initializer list.
- ∼Vector ()

  Destroys the vector and deallocates its memory.
- Vector & operator= (const Vector &other)

  Assigns the contents of another vector to this vector.
- Vector & operator= (Vector &&other) noexcept

  Assigns the contents of another vector to this vector by moving.
- reference at (size_type pos)

  Accesses the element at the specified position with bounds checking.
- const_reference at (size_type pos) const

  Accesses the element at the specified position with bounds checking.
- reference operator[ ] (size_type pos)

  Accesses the element at the specified position without bounds checking.
- const_reference operator[ ] (size_type pos) const

  Accesses the element at the specified position without bounds checking.
- const_reference front () const

  Accesses the first element in the vector.
- reference back ()

  Accesses the last element in the vector.
- const_reference back () const

  Accesses the last element in the vector.
- pointer data () noexcept

  Returns a pointer to the underlying array serving as element storage.
- const_pointer data () const noexcept

  Returns a const pointer to the underlying array serving as element storage.

- iterator begin () noexcept

    Returns an iterator to the beginning of the vector.
- const_iterator begin () const noexcept

    Returns a const iterator to the beginning of the vector.
- const_iterator cbegin () const noexcept

    Returns a const iterator to the beginning of the vector.
- iterator end () noexcept

    Returns an iterator to the end of the vector.
- const_iterator end () const noexcept

    Returns a const iterator to the end of the vector.
- const_iterator cend () const noexcept

    Returns a const iterator to the end of the vector.
- reverse_iterator rbegin () noexcept

    Returns a reverse iterator to the beginning of the reversed vector.
- const_reverse_iterator rbegin () const noexcept

    Returns a const reverse iterator to the beginning of the reversed vector.
- reverse_iterator rend () noexcept

    Returns a reverse iterator to the end of the reversed vector.
- const_reverse_iterator rend () const noexcept

    Returns a const reverse iterator to the end of the reversed vector.
- const_reverse_iterator crbegin () const noexcept

    Returns a const reverse iterator to the beginning of the reversed vector.
- const_reverse_iterator crend () const noexcept

    Returns a const reverse iterator to the end of the reversed vector.
- bool empty () const noexcept

    Checks if the vector is empty.
- size_type size () const noexcept

    Returns the number of elements in the vector.
- size_type max_size () const noexcept

    Returns the maximum number of elements the vector can hold.
- void reserve (size_type new_cap)

    Increases the capacity of the vector to a value greater than or equal to new_cap.
- size_type capacity () const noexcept

    Returns the current capacity of the vector.
- void shrink_to_fit ()

    Reduces the capacity of the vector to fit its size.
- void clear () noexcept

    Clears the contents of the vector.
- void push_back (const T &value)

    Adds an element to the end of the vector.
- void push_back (T &&value)

    Moves an element to the end of the vector.
- void pop_back ()

    Removes the last element from the vector.
- template<typename... Args>
  reference emplace_back (Args &&... args)

    Constructs an element in-place at the end of the vector.
- iterator insert (const_iterator pos, const T &value)

    Inserts an element into the vector at the specified position.
- iterator insert (const_iterator pos, T &&value)

    Inserts an element into the vector at the specified position by moving.

- iterator insert (const_iterator pos, size_type count, const T &value)

    Inserts multiple elements into the vector at the specified position.
- template< typename InputIt >
  iterator insert (const_iterator pos, InputIt first, InputIt last)

    Inserts elements from a range into the vector at the specified position.
- iterator erase (const_iterator pos)

    Erases an element from the vector at the specified position.
- iterator erase (const_iterator first, const_iterator last)

    Erases elements in the range [first, last) from the vector.
- void resize (size_type count)

    Resizes the vector to contain the specified number of elements.
- void resize (size_type count, const value_type &value)

    Resizes the vector to contain the specified number of elements.
- void swap (Vector &other) noexcept

    Swaps the contents of this vector with another vector.

4.2.1 Detailed Description

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

A dynamically resizing array similar to std::vector.

Template Parameters

T	The type of elements stored in the vector.
Allocator	The type of allocator used to manage memory allocation.

4.2.2 Member Typedef Documentation

4.2.2.1 allocator_type

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

The type of allocator used for memory allocation.

4.2.2.2 const_iterator

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

Const iterator for traversing the vector.

4.2.2.3 const_pointer

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

Const pointer to an element.

4.2.2.4 const_reference

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

Const reference to an element.

4.2.2.5 const_reverse_iterator

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

Const reverse iterator for traversing the vector in reverse.

4.2.2.6 difference_type

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

A signed integer type used to represent differences between iterators.

4.2.2.7 iterator

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

Iterator for traversing the vector.

4.2.2.8 pointer

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

Pointer to an element.

4.2.2.9 reference

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

Reference to an element.

4.2.2.10 reverse_iterator

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

Reverse iterator for traversing the vector in reverse.

4.2.2.11 size_type

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

An unsigned integer type used to represent sizes.

4.2.2.12 value_type

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

The type of elements stored in the vector.

4.2.3 Constructor & Destructor Documentation

4.2.3.1 Vector() [1/5]

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

Constructs an empty vector.

4.2.3.2 Vector() [2/5]

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

Constructs the vector with a specified number of elements, each initialized with a given value.

Parameters

count	The number of elements in the vector.
value	The value to initialize the elements with.
alloc	The allocator used for memory allocation.

4.2.3.3 Vector() [3/5]

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

Constructs the vector by copying elements from another vector.

Parameters

other	The vector to copy.

4.2.3.4 Vector() [4/5]

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

Constructs the vector by moving elements from another vector.

Parameters

other	The vector to move from.

4.2.3.5 Vector() [5/5]

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

Constructs the vector with the elements from the initializer list.

Parameters

init	The initializer list to initialize the elements of the vector.
alloc	The allocator used for memory allocation.

4.2.3.6 ∼Vector()

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

Destroys the vector and deallocates its memory.

4.2.4 Member Function Documentation

4.2.4.1 at() [1/2]

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

Accesses the element at the specified position with bounds checking.

Parameters

pos	The position of the element to access.

Returns

Reference to the element at the specified position.

Exceptions

std::out_of_range	if pos is out of range.

4.2.4.2 at() [2/2]

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

Accesses the element at the specified position with bounds checking.

Parameters

pos	The position of the element to access.

Returns

Const reference to the element at the specified position.

Exceptions

std::out_of_range	if pos is out of range.

4.2.4.3 back() [1/2]

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

Accesses the last element in the vector.

Returns

Reference to the last element.

4.2.4.4 back() [2/2]

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

Accesses the last element in the vector.

Returns

Const reference to the last element.

4.2.4.5 begin() [1/2]

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

Returns a const iterator to the beginning of the vector.

Returns

Const iterator to the beginning of the vector.

4.2.4.6 begin() [2/2]

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

Returns an iterator to the beginning of the vector.

Returns

Iterator to the beginning of the vector.

4.2.4.7 capacity()

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

Returns the current capacity of the vector.

Returns

The current capacity of the vector.

4.2.4.8 cbegin()

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

Returns a const iterator to the beginning of the vector.

Returns

Const iterator to the beginning of the vector.

4.2.4.9 cend()

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

Returns a const iterator to the end of the vector.

Returns

Const iterator to the end of the vector.

4.2.4.10 clear()

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

Clears the contents of the vector.

4.2.4.11 crbegin()

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

Returns a const reverse iterator to the beginning of the reversed vector.

Returns

Const reverse iterator to the beginning of the reversed vector.

4.2.4.12 crend()

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

Returns a const reverse iterator to the end of the reversed vector.

Returns

Const reverse iterator to the end of the reversed vector.

4.2.4.13 data() [1/2]

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

Returns a const pointer to the underlying array serving as element storage.

Returns

Const pointer to the underlying element storage.

4.2.4.14 data() [2/2]

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

Returns a pointer to the underlying array serving as element storage.

Returns

Pointer to the underlying element storage.

4.2.4.15 emplace_back()

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

Constructs an element in-place at the end of the vector.

Template Parameters

Args	Types of arguments to forward to the constructor of the element.

Parameters

args	Arguments to forward to the constructor of the element.

Returns

Reference to the newly constructed element.

4.2.4.16 empty()

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

Checks if the vector is empty.

Returns

True if the vector is empty, false otherwise.

4.2.4.17 end() [1/2]

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

Returns a const iterator to the end of the vector.

Returns

Const iterator to the end of the vector.

4.2.4.18 end() [2/2]

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

Returns an iterator to the end of the vector.

Returns

Iterator to the end of the vector.

4.2.4.19 erase() [1/2]

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

Erases elements in the range [first, last) from the vector.

Parameters

first	Iterator to the beginning of the range to be erased.
last	Iterator to the end of the range to be erased.

Returns

Iterator following the last removed element.

4.2.4.20 erase() [2/2]

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

Erases an element from the vector at the specified position.

Parameters

pos	Iterator pointing to the element to be erased.

Returns

Iterator following the last removed element.

4.2.4.21 front()

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

Accesses the first element in the vector.

Returns

Reference to the first element.

4.2.4.22 insert() [1/4]

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

Inserts an element into the vector at the specified position.

Parameters

pos	Iterator pointing to the position where the element should be inserted.
value	The value to be inserted.

Returns

Iterator pointing to the inserted element.

4.2.4.23 insert() [2/4]

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

Inserts elements from a range into the vector at the specified position.

Template Parameters

Input↩ It	Type of the input iterator.

Parameters

pos	Iterator pointing to the position where the elements should be inserted.
first	Iterator to the beginning of the range.
last	Iterator to the end of the range.

Returns

Iterator pointing to the first inserted element.

4.2.4.24 insert() [3/4]

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

Inserts multiple elements into the vector at the specified position.

Parameters

pos	Iterator pointing to the position where the elements should be inserted.
count	The number of elements to insert.
value	The value to initialize the inserted elements with.

Returns

Iterator pointing to the first inserted element.

4.2.4.25 insert() [4/4]

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

Inserts an element into the vector at the specified position by moving.

Parameters

pos	Iterator pointing to the position where the element should be inserted.
value	The value to be moved into the vector.

Returns

Iterator pointing to the inserted element.

4.2.4.26 max_size()

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

Returns the maximum number of elements the vector can hold.

Returns

The maximum number of elements the vector can hold.

4.2.4.27 operator=() [1/2]

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

Assigns the contents of another vector to this vector.

Parameters

other	The vector to copy.

Returns

A reference to this vector after the assignment.

4.2.4.28 operator=() [2/2]

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

Assigns the contents of another vector to this vector by moving.

Parameters

other	The vector to move from.

Returns

A reference to this vector after the assignment.

4.2.4.29 operator[]() [1/2]

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

Accesses the element at the specified position without bounds checking.

Parameters

pos	The position of the element to access.

Returns

Reference to the element at the specified position.

4.2.4.30 operator[]() [2/2]

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

Accesses the element at the specified position without bounds checking.

Parameters

pos	The position of the element to access.

Returns

Const reference to the element at the specified position.

4.2.4.31 pop_back()

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

Removes the last element from the vector.

4.2.4.32 push_back() [1/2]

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

Adds an element to the end of the vector.

Parameters

value	The value to be added to the vector.

4.2.4.33 push_back() [2/2]

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

Moves an element to the end of the vector.

Parameters

value	The value to be moved to the vector.

4.2.4.34 rbegin() [1/2]

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

Returns a const reverse iterator to the beginning of the reversed vector.

Returns

Const reverse iterator to the beginning of the reversed vector.

4.2.4.35 rbegin() [2/2]

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

Returns a reverse iterator to the beginning of the reversed vector.

Returns

Reverse iterator to the beginning of the reversed vector.

4.2.4.36 rend() [1/2]

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

Returns a const reverse iterator to the end of the reversed vector.

Returns

Const reverse iterator to the end of the reversed vector.

4.2.4.37 rend() [2/2]

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

Returns a reverse iterator to the end of the reversed vector.

Returns

Reverse iterator to the end of the reversed vector.

4.2.4.38 reserve()

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

Increases the capacity of the vector to a value greater than or equal to new_cap.

Parameters

new_cap	The new capacity of the vector.

4.2.4.39 resize() [1/2]

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

Resizes the vector to contain the specified number of elements.

If the current size is greater than the count, elements are destroyed. If the current size is less than the count, additional elements are default-constructed.

Parameters

count	The new size of the vector.

4.2.4.40 resize() [2/2]

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

Resizes the vector to contain the specified number of elements.

If the current size is greater than the count, elements are destroyed. If the current size is less than the count, additional elements are value-initialized with value.

Parameters

count	The new size of the vector.
value	The value to initialize the new elements with.

4.2.4.41 shrink_to_fit()

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

Reduces the capacity of the vector to fit its size.

4.2.4.42 size()

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

Returns the number of elements in the vector.

Returns

The number of elements in the vector.

4.2.4.43 swap()

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

Swaps the contents of this vector with another vector.

Parameters

other	The vector to swap with.

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

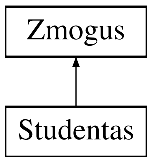
- src/Vektorius.h

4.3 Zmogus Class Reference

Abstract class representing a person.

`#include <Studentas.h>`

Inheritance diagram for Zmogus:



Public Member Functions

- virtual ~Zmogus ()=default
- virtual std::string getVardas () const =0

  Get the first name of the person.
- virtual std::string getPavarde () const =0

  Get the last name of the person.
- virtual double getEgzaminoRez () const =0

  Get the exam grade of the person.
- virtual Vector< double > & getNamudarbuRez ()=0

  Get the grades of homework for the person.

4.3.1 Detailed Description

Abstract class representing a person.

This class defines the basic interface for a person, providing methods to retrieve information such as name and exam grades.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ∼Zmogus()

```
virtual Zmogus::∼Zmogus ()  [virtual], [default]
```

4.3.3 Member Function Documentation

4.3.3.1 getEgzaminoRez()

```
virtual double Zmogus::getEgzaminoRez () const  [pure virtual]
```

Get the exam grade of the person.

Returns

The exam grade of the person.

Implemented in Studentas.

4.3.3.2 getNamudarbuRez()

```
virtual Vector< double > & Zmogus::getNamudarbuRez ()  [pure virtual]
```

Get the grades of homework for the person.

Returns

A reference to the Vector containing homework grades of the person.

Implemented in Studentas.

4.3.3.3 getPavarde()

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

Get the last name of the person.

Returns

The last name of the person.

Implemented in Studentas.

4.3.3.4 getVardas()

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

Get the first name of the person.

Returns

The first name of the person.

Implemented in Studentas.

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

- src/Studentas.h

Chapter 5

File Documentation

5.1 src/ConsoleApplication1.cpp File Reference

```
#include "MokiniuProcessing.h"
#include "Skaiciavimaidarbai.h"
#include "Vektorius.h"
```

Functions

- int main ()

    Main function that serves as the entry point of the program.

5.1.1 Function Documentation

5.1.1.1 main()

```
int main ()
```

Main function that serves as the entry point of the program.

The main function initializes the necessary components, handles user input, and directs the program flow based on the user's choices. It provides options for entering student data manually, generating random data, reading from files, and testing different strategies.

Returns

int Returns 0 on successful execution.

5.2 src/MokiniuProcessing.cpp File Reference

```
#include "MokiniuProcessing.h"
```

Functions

- string GeneruotiVardus ()

    Generates a random Lithuanian first name from a predefined list.
- string GeneruotiPavardes ()

    Generates a random Lithuanian last name from a predefined list.
- void PatikrintiTeigiamajiSkaiciu (double skaicius)

    Checks if a given number is within a valid range (0 to 10).
- bool ContainsNumbers (const string &str)

    Checks if a string contains any numeric digits.

5.2.1 Function Documentation

5.2.1.1 ContainsNumbers()

```
bool ContainsNumbers (
            const string & str)
```

Checks if a string contains any numeric digits.

This function returns true if the provided string contains at least one numeric digit, and false otherwise.

Parameters

str	The string to check.

Returns

bool True if the string contains numeric digits, false otherwise.

5.2.1.2 GeneruotiPavardes()

```
string GeneruotiPavardes ()
```

Generates a random Lithuanian last name from a predefined list.

This function selects a random last name from a Vector of common Lithuanian surnames and returns it as a string.

Returns

string A randomly selected Lithuanian last name.

5.2.1.3 GeneruotiVardus()

```
string GeneruotiVardus ()
```

Generates a random Lithuanian first name from a predefined list.

This function selects a random first name from a Vector of common Lithuanian names and returns it as a string.

Returns

string A randomly selected Lithuanian first name.

5.2.1.4 PatikrintiTeigiamajiSkaiciu()

```
void PatikrintiTeigiamajiSkaiciu (
             double skaicius)
```

Checks if a given number is within a valid range (0 to 10).

This function throws an invalid_argument exception if the provided number is not within the range of 0 to 10.

Parameters

skaicius	The number to check.

Exceptions

invalid_argument	If the number is not within the range 0 to 10.

5.3 src/MokiniuProcessing.h File Reference

```
#include <string>
#include <vector>
#include <stdexcept>
#include <iostream>
#include <locale>
#include <numeric>
#include <fstream>
#include <sstream>
#include <cctype>
#include <algorithm>
#include <chrono>
#include <iomanip>
#include <stdlib.h>
#include "Studentas.h"
```

Functions

- string GeneruotiVardus ()

  Generates a random Lithuanian first name from a predefined list.
- string GeneruotiPavardes ()

  Generates a random Lithuanian last name from a predefined list.
- void PatikrintiTeigiamajiSkaiciu (double skaicius)

  Checks if a given number is within a valid range (0 to 10).
- bool ContainsNumbers (const string &str)

  Checks if a string contains any numeric digits.

5.3.1 Function Documentation

5.3.1.1 ContainsNumbers()

```
bool ContainsNumbers (
            const string & str)
```

Checks if a string contains any numeric digits.

This function returns true if the provided string contains at least one numeric digit, and false otherwise.

Parameters

str	The string to check.

Returns

bool True if the string contains numeric digits, false otherwise.

5.3.1.2 GeneruotiPavardes()

```
string GeneruotiPavardes ()
```

Generates a random Lithuanian last name from a predefined list.

This function selects a random last name from a vector of common Lithuanian surnames and returns it as a string.

Returns

string A randomly selected Lithuanian last name.

This function selects a random last name from a Vector of common Lithuanian surnames and returns it as a string.

Returns

string A randomly selected Lithuanian last name.

5.3.1.3 GeneruotiVardus()

```
string GeneruotiVardus ()
```

Generates a random Lithuanian first name from a predefined list.

This function selects a random first name from a vector of common Lithuanian names and returns it as a string.

Returns

string A randomly selected Lithuanian first name.

This function selects a random first name from a Vector of common Lithuanian names and returns it as a string.

Returns

string A randomly selected Lithuanian first name.

5.3.1.4 PatikrintiTeigiamajiSkaiciu()

```
void PatikrintiTeigiamajiSkaiciu (
             double skaicius)
```

Checks if a given number is within a valid range (0 to 10).

This function throws an invalid_argument exception if the provided number is not within the range of 0 to 10.

Parameters

skaicius	The number to check.

Exceptions

invalid_argument	If the number is not within the range 0 to 10.

5.4 MokiniuProcessing.h

Go to the documentation of this file.

```
#pragma once
#include <string>
#include <vector>
#include <stdexcept>
#include <iostream>
#include <locale>
#include <numeric>
#include <fstream>
#include <sstream>
#include <cctype>
#include <algorithm>
#include <chrono>
#include <iomanip>
#include <stdlib.h>
#include "Studentas.h"

using namespace std;

string GeneruotiVardus();

string GeneruotiPavardes();

void PatikrintiTeigiamajiSkaiciu(double skaicius);

bool ContainsNumbers(const string& str);
```

5.5 src/Skaiciavimaidarbai.cpp File Reference

```
#include "Skaiciavimaidarbai.h"
#include "MokiniuProcessing.h"
#include "Studentas.h"
```

Functions

- void NetinkamaIvestis ()

  Prints a message indicating invalid input and terminates the program.
- void NeraFailo ()

  Prints a message indicating that a file was not found and terminates the program.
- double Mediana (Vector< double > &vec)

  Calculates the median of a Vector of doubles.
- double GenerateRandomGrade ()

  Generates a random grade between 0 and 10.
- void GeneruotiFaila (const string &pavadinimas, int studentuskaicius)

  Generates a file with random student data.
- bool compareByGalutinisVid (const Studentas &a, const Studentas &b)

  Comparator function to compare students by their final average grade.
- bool compareByGalutinisMed (const Studentas &a, const Studentas &b)

  Comparator function to compare students by their final median grade.

5.5.1 Function Documentation

5.5.1.1 compareByGalutinisMed()

```
bool compareByGalutinisMed (
            const Studentas & a,
            const Studentas & b)
```

Comparator function to compare students by their final median grade.

This function compares two students based on their final median grade and returns true if the first student's grade is less than the second student's grade.

Parameters

a	The first student to compare.
b	The second student to compare.

Returns

bool True if the first student's final median grade is less than the second student's grade.

5.5.1.2 compareByGalutinisVid()

```
bool compareByGalutinisVid (
            const Studentas & a,
            const Studentas & b)
```

Comparator function to compare students by their final average grade.

This function compares two students based on their final average grade and returns true if the first student's grade is less than the second student's grade.

Parameters

a	The first student to compare.
b	The second student to compare.

Returns

bool True if the first student's final average grade is less than the second student's grade.

5.5.1.3 GenerateRandomGrade()

```
double GenerateRandomGrade ()
```

Generates a random grade between 0 and 10.

This function returns a random grade in the range of 0 to 10.

Returns

double A randomly generated grade between 0 and 10.

5.5.1.4 GeneruotiFaila()

```
void GeneruotiFaila (
            const string & pavadinimas,
            int studentuskaicius)
```

Generates a file with random student data.

This function creates a file with the specified name and populates it with a specified number of students. Each student will have a generated name, surname, and a set of random grades.

Parameters

pavadinimas	The name of the file to generate.
studentuskaicius	The number of students to generate data for.

5.5.1.5 Mediana()

```
double Mediana (
            Vector< double > & vec)
```

Calculates the median of a Vector of doubles.

Calculates the median of a vector of doubles.

This function sorts the provided Vector and calculates the median value. If the Vector size is even, it returns the average of the two middle elements. If the Vector size is odd, it returns the middle element.

Parameters

vec	The Vector of doubles to calculate the median for.

Returns

double The median value of the Vector.

5.5.1.6 NeraFailo()

```
void NeraFailo ()
```

Prints a message indicating that a file was not found and terminates the program.

This function outputs a message to the console indicating that the specified file was not found and then terminates the program.

5.5.1.7 NetinkamaIvestis()

```
void NetinkamaIvestis ()
```

Prints a message indicating invalid input and terminates the program.

This function outputs a message to the console indicating that the input was invalid and then terminates the program.

5.6 src/Skaiciavimaidarbai.h File Reference

```
#include "MokiniuProcessing.h"
```

Functions

- void NetinkamaIvestis ()

  Prints a message indicating invalid input and terminates the program.
- void NeraFailo ()

  Prints a message indicating that a file was not found and terminates the program.
- double Mediana (Vector< double > &vec)

  Calculates the median of a vector of doubles.
- double GenerateRandomGrade ()

  Generates a random grade between 0 and 10.
- void GeneruotiFaila (const string &pavadinimas, int studentuskaicius)

  Generates a file with random student data.
- bool compareByGalutinisVid (const Studentas &a, const Studentas &b)

  Comparator function to compare students by their final average grade.
- bool compareByGalutinisMed (const Studentas &a, const Studentas &b)

  Comparator function to compare students by their final median grade.

5.6.1 Function Documentation

5.6.1.1 compareByGalutinisMed()

```
bool compareByGalutinisMed (
            const Studentas & a,
            const Studentas & b)
```

Comparator function to compare students by their final median grade.

This function compares two students based on their final median grade and returns true if the first student's grade is less than the second student's grade.

Parameters

a	The first student to compare.
b	The second student to compare.

Returns

bool True if the first student's final median grade is less than the second student's grade.

5.6.1.2 compareByGalutinisVid()

```
bool compareByGalutinisVid (
            const Studentas & a,
            const Studentas & b)
```

Comparator function to compare students by their final average grade.

This function compares two students based on their final average grade and returns true if the first student's grade is less than the second student's grade.

Parameters

a	The first student to compare.
b	The second student to compare.

Returns

bool True if the first student's final average grade is less than the second student's grade.

5.6.1.3 GenerateRandomGrade()

```
double GenerateRandomGrade ()
```

Generates a random grade between 0 and 10.

This function returns a random grade in the range of 0 to 10.

Returns

double A randomly generated grade between 0 and 10.

5.6.1.4 GeneruotiFaila()

```
void GeneruotiFaila (
             const string & pavadinimas,
             int studentuskaicius)
```

Generates a file with random student data.

This function creates a file with the specified name and populates it with a specified number of students. Each student will have a generated name, surname, and a set of random grades.

Parameters

pavadinimas	The name of the file to generate.
studentuskaicius	The number of students to generate data for.

5.6.1.5 Mediana()

```
double Mediana (
             Vector< double > & vec)
```

Calculates the median of a vector of doubles.

This function sorts the provided vector and calculates the median value. If the vector size is even, it returns the average of the two middle elements. If the vector size is odd, it returns the middle element.

Parameters

vec	The vector of doubles to calculate the median for.

Returns

double The median value of the vector.

Calculates the median of a vector of doubles.

This function sorts the provided Vector and calculates the median value. If the Vector size is even, it returns the average of the two middle elements. If the Vector size is odd, it returns the middle element.

Parameters

vec	The Vector of doubles to calculate the median for.

Returns

double The median value of the Vector.

5.6.1.6 NeraFailo()

```
void NeraFailo ()
```

Prints a message indicating that a file was not found and terminates the program.

This function outputs a message to the console indicating that the specified file was not found and then terminates the program.

5.6.1.7 NetinkamaIvestis()

```
void NetinkamaIvestis ()
```

Prints a message indicating invalid input and terminates the program.

This function outputs a message to the console indicating that the input was invalid and then terminates the program.

5.7 Skaiciavimaidarbai.h

Go to the documentation of this file.

```
#pragma once
#include "MokiniuProcessing.h"

void NetinkamaIvestis();

void NeraFailo();

double Mediana(Vector<double>& vec);

double GenerateRandomGrade();

void GeneruotiFaila(const string& pavadinimas, int studentuskaicius);

bool compareByGalutinisVid(const Studentas& a, const Studentas& b);

bool compareByGalutinisMed(const Studentas& a, const Studentas& b);
```

5.8 src/Studentas.cpp File Reference

```
#include "Studentas.h"
#include "Skaiciavimaidarbai.h"
#include <fstream>
#include <chrono>
#include <algorithm>
#include "Vektorius.h"
```

Functions

- bool vardolyginimas (const Studentas &a, const Studentas &b)

  Compares two students by their first name.
- bool pavardeslyginimas (const Studentas &a, const Studentas &b)

  Compares two students by their last name.
- bool vidurkiolyginimas (const Studentas &a, const Studentas &b)

  Compares two students by their final average grade.
- bool medianoslyginimas (const Studentas &a, const Studentas &b)

  Compares two students by their final median grade.
- void PrintStudents (const Vector< Studentas > &studentai)

  Prints a list of students to the console.
- void WriteNormalStudents (Vector< Studentas > &normalus)

  Writes normal students to a file.
- void WriteWeirdStudents (Vector< Studentas > &nenormalus)

  Writes weird students to a file.
- void readAndProcessData (const std::string &filename, Vector< Studentas > &studentai, int &namudarbai, int studentuskaicius)

  Reads and processes student data from a file.
- void sortStudents (Vector< Studentas > &studentai, int sortpasirinkimas)

  Sorts students using the STL sort function.
- void partitionStudents1 (const Vector< Studentas > &studentai, Vector< Studentas > &normalus, Vector< Studentas > &nenormalus)

  Partitions students into normal and not normal students using method 1.
- void partitionStudents2 (Vector< Studentas > &studentai, Vector< Studentas > &nenormalus)

  Partitions students into normal and not normal students using method 2.
- void partitionStudents3 (Vector< Studentas > &studentai, Vector< Studentas > &normalus, Vector< Studentas > &nenormalus)

  Partitions students into normal and not normal students using method 3.
- std::ostream & operator<< (std::ostream &os, Studentas &studentas)

  Output operator implementation.
- std::istream & operator>> (std::istream &is, Studentas &studentas)

  Input operator implementation.
- void testConstructors ()

  Test constructors.

5.8.1 Function Documentation

5.8.1.1 medianoslyginimas()

```
bool medianoslyginimas (
          const Studentas & a,
          const Studentas & b)
```

Compares two students by their final median grade.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's final median grade is less than the second student's final median grade.

5.8.1.2 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            Studentas & studentas)
```

Output operator implementation.

Output operator implementation.

5.8.1.3 operator>>()

```
std::istream & operator>> (
            std::istream & is,
            Studentas & studentas)
```

Input operator implementation.

Input operator implementation.

5.8.1.4 partitionStudents1()

```
void partitionStudents1 (
            const Vector< Studentas > & studentai,
            Vector< Studentas > & normalus,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 1.

Parameters

studentai	The Vector to store the read students.
normalus	The Vector to store normal students.
nenormalus	The Vector to store not normal students.

5.8.1.5 partitionStudents2()

```
void partitionStudents2 (
            Vector< Studentas > & studentai,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 2.

Parameters

studentai	The Vector to store the read students.
nenormalus	The Vector to store not normal students.

5.8.1.6 partitionStudents3()

```
void partitionStudents3 (
            Vector< Studentas > & studentai,
            Vector< Studentas > & normalus,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 3.

Parameters

studentai	The Vector to store the read students.
normalus	The Vector to store normal students.
nenormalus	The Vector to store not normal students.

5.8.1.7 pavardeslyginimas()

```
bool pavardeslyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their last name.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's last name is less than the second student's last name.

5.8.1.8 PrintStudents()

```
void PrintStudents (
            const Vector< Studentas > & studentai)
```

Prints a list of students to the console.

Parameters

studentai	The Vector of students to print.

5.8.1.9 readAndProcessData()

```
void readAndProcessData (
            const std::string & filename,
            Vector< Studentas > & studentai,
            int & namudarbai,
            int studentuskaicius)
```

Reads and processes student data from a file.

Parameters

filename	The name of the file to read from.
studentai	The Vector to store the read students.
namudarbai	The number of homework grades.
studentuskaicius	The number of students.

5.8.1.10 sortStudents()

```
void sortStudents (
            Vector< Studentas > & studentai,
            int sortpasirinkimas)
```

Sorts students using the STL sort function.

Parameters

studentai	The Vector to store the read students.
sortpasirinkimas	The sorting option to use.

5.8.1.11 testConstructors()

```
void testConstructors ()
```

Test constructors.

Test constructors.

5.8.1.12 vardolyginimas()

```
bool vardolyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their first name.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's name is less than the second student's name.

5.8.1.13 vidurkiolyginimas()

```
bool vidurkiolyginimas (
             const Studentas & a,
             const Studentas & b)
```

Compares two students by their final average grade.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's final average grade is less than the second student's final average grade.

5.8.1.14 WriteNormalStudents()

```
void WriteNormalStudents (
             Vector< Studentas > & normalus)
```

Writes normal students to a file.

Parameters

normalus	The Vector of normal students.

5.8.1.15 WriteWeirdStudents()

```
void WriteWeirdStudents (
             Vector< Studentas > & nenormalus)
```

Writes weird students to a file.

Parameters

nenormalus	The Vector of weird students.

5.9 src/Studentas.h File Reference

```
#include <string>
#include "Vektorius.h"
```

Classes

- class Zmogus

    Abstract class representing a person.
- class Studentas

    Class representing a student.

Functions

- bool vardolyginimas (const Studentas &a, const Studentas &b)

    Compares two students by their first name.
- bool pavardeslyginimas (const Studentas &a, const Studentas &b)

    Compares two students by their last name.
- bool vidurkiolyginimas (const Studentas &a, const Studentas &b)

    Compares two students by their final average grade.
- bool medianoslyginimas (const Studentas &a, const Studentas &b)

    Compares two students by their final median grade.
- void PrintStudents (const Vector< Studentas > &studentai)

    Prints a list of students to the console.
- void readAndProcessData (const std::string &filename, Vector< Studentas > &studentai, int &namudarbai, int studentuskaicius)

    Reads and processes student data from a file.
- void sortStudents (Vector< Studentas > &studentai, int sortpasirinkimas)

    Sorts students using the STL sort function.
- void partitionStudents1 (const Vector< Studentas > &studentai, Vector< Studentas > &normalus, Vector< Studentas > &nenormalus)

    Partitions students into normal and not normal students using method 1.
- void partitionStudents2 (Vector< Studentas > &studentai, Vector< Studentas > &nenormalus)

    Partitions students into normal and not normal students using method 2.
- void partitionStudents3 (Vector< Studentas > &studentai, Vector< Studentas > &normalus, Vector< Studentas > &nenormalus)

    Partitions students into normal and not normal students using method 3.
- void WriteWeirdStudents (Vector< Studentas > &nenormalus)

    Writes weird students to a file.
- void WriteNormalStudents (Vector< Studentas > &normalus)

    Writes normal students to a file.
- void testConstructors ()

    Test constructors.

5.9.1 Function Documentation

5.9.1.1 medianoslyginimas()

```
bool medianoslyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their final median grade.

Compare students by median final grade.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's final median grade is less than the second student's final median grade.

5.9.1.2 partitionStudents1()

```
void partitionStudents1 (
            const Vector< Studentas > & studentai,
            Vector< Studentas > & normalus,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 1.

Partition students into normal and weird categories (method 1).

Parameters

studentai	The Vector to store the read students.
normalus	The Vector to store normal students.
nenormalus	The Vector to store not normal students.

5.9.1.3 partitionStudents2()

```
void partitionStudents2 (
            Vector< Studentas > & studentai,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 2.

Partition students into normal and weird categories (method 2).

Parameters

studentai	The Vector to store the read students.
nenormalus	The Vector to store not normal students.

5.9.1.4 partitionStudents3()

```
void partitionStudents3 (
            Vector< Studentas > & studentai,
            Vector< Studentas > & normalus,
            Vector< Studentas > & nenormalus)
```

Partitions students into normal and not normal students using method 3.

Partition students into normal and weird categories (method 3).

Parameters

studentai	The Vector to store the read students.
normalus	The Vector to store normal students.
nenormalus	The Vector to store not normal students.

5.9.1.5 pavardeslyginimas()

```
bool pavardeslyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their last name.

Compare students by last name.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's last name is less than the second student's last name.

5.9.1.6 PrintStudents()

```
void PrintStudents (
            const Vector< Studentas > & studentai)
```

Prints a list of students to the console.

Print a list of students.

Parameters

studentai	The Vector of students to print.

5.9.1.7 readAndProcessData()

```
void readAndProcessData (
            const std::string & filename,
            Vector< Studentas > & studentai,
            int & namudarbai,
            int studentuskaicius)
```

Reads and processes student data from a file.

Read and process student data from a file.

Parameters

filename	The name of the file to read from.
studentai	The Vector to store the read students.
namudarbai	The number of homework grades.
studentuskaicius	The number of students.

5.9.1.8 sortStudents()

```
void sortStudents (
            Vector< Studentas > & studentai,
            int sortpasirinkimas)
```

Sorts students using the STL sort function.

Sort students based on specified criteria.

Parameters

studentai	The Vector to store the read students.
sortpasirinkimas	The sorting option to use.

5.9.1.9 testConstructors()

```
void testConstructors ()
```

Test constructors.

Test constructors of the Studentas class.

Test constructors.

5.9.1.10 vardolyginimas()

```
bool vardolyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their first name.

Compare students by first name.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's name is less than the second student's name.

5.9.1.11 vidurkiolyginimas()

```
bool vidurkiolyginimas (
            const Studentas & a,
            const Studentas & b)
```

Compares two students by their final average grade.

Compare students by average final grade.

Parameters

a	The first student.
b	The second student.

Returns

bool True if the first student's final average grade is less than the second student's final average grade.

5.9.1.12 WriteNormalStudents()

```
void WriteNormalStudents (
            Vector< Studentas > & normalus)
```

Writes normal students to a file.

Write normal students to a file.

Parameters

normalus	The Vector of normal students.

5.9.1.13 WriteWeirdStudents()

```
void WriteWeirdStudents (
            Vector< Studentas > & nenormalus)
```

Writes weird students to a file.

Write weird students to a file.

Parameters

nenormalus	The Vector of weird students.

5.10 Studentas.h

Go to the documentation of this file.

```
#pragma once

#include <string>
#include "Vektorius.h"

class Zmogus {
public:
    virtual ~Zmogus() = default;
    // Getters
    virtual std::string getVardas() const = 0;
    virtual std::string getPavarde() const = 0;
    virtual double getEgzaminoRez() const = 0;
    virtual Vector<double>& getNamudarbuRez() = 0;
};
class Studentas : public Zmogus {
private:
    std::string vardas_;
    std::string pavarde_;
    Vector<double> namudarburez_;
    double egzaminorez_;

public:
    double namudarburezsuma_;
    double vidurkis_;
    double galutinisbalasvidurkis_;
    double mediana_;
    double galutinisbalasmediana_;
    // Constructors
    Studentas();
    Studentas(std::istream& is);
    // Getters
    std::string getVardas() const { return vardas_; }
    std::string getPavarde() const { return pavarde_; }
    double getEgzaminoRez() const { return egzaminorez_; }
    Vector<double>& getNamudarbuRez() { return namudarburez_; }
    // Setters
    void setVardas(const std::string& vardas) { vardas_ = vardas; }
    void setPavarde(const std::string& pavarde) { pavarde_ = pavarde; }
    void setEgzaminoRez(double egzaminorez) { egzaminorez_ = egzaminorez; }
    void addGrade(double grade) { namudarburez_.push_back(grade); }
    // Other member functions
    std::istream& readStudent(std::istream&);
    void calculateFinalGrades();
    //Destructor
    ~Studentas();
    Studentas(const Studentas& other);
    Studentas& operator=(const Studentas& other);
    Studentas(Studentas&& other) noexcept;
    Studentas& operator=(Studentas&& other) noexcept;
};
// Negalima kurti zmogus objekto -
// Zmogus a;

// Comparison functions
bool vardolyginimas(const Studentas& a, const Studentas& b);
bool pavardeslyginimas(const Studentas& a, const Studentas& b);
bool vidurkiolyginimas(const Studentas& a, const Studentas& b);
bool medianoslyginimas(const Studentas& a, const Studentas& b);
// Utility functions
void PrintStudents(const Vector<Studentas>& studentai);
void readAndProcessData(const std::string& filename, Vector<Studentas>& studentai, int& namudarbai,
int studentuskaicius);
void sortStudents(Vector<Studentas>& studentai, int sortpasirinkimas);
void partitionStudents1(const Vector<Studentas>& studentai, Vector<Studentas>& normalus,
Vector<Studentas>& nenormalus);
void partitionStudents2(Vector<Studentas>& studentai, Vector<Studentas>& nenormalus);
void partitionStudents3(Vector<Studentas>& studentai, Vector<Studentas>& normalus, Vector<Studentas>&
nenormalus);
void WriteWeirdStudents(Vector<Studentas>& nenormalus);
void WriteNormalStudents(Vector<Studentas>& normalus);
void testConstructors();
```

5.11 src/Vektorius.cpp File Reference

```
#include "Vektorius.h"
#include <iostream>
#include <utility>
#include <vector>
#include <chrono>
```

Functions

- void VectorExample ()

  An example function demonstrating the usage of the Vector class.
- int sumVector (const Vector< int > &vec)

  Calculates the sum of elements in a vector.
- bool isSorted (const Vector< int > &vec)

Checks if a vector is sorted in ascending order.

- void removeDuplicates (Vector< int > &vec)

    Removes duplicate elements from a vector.

- Vector< int > mergeSortedVectors (const Vector< int > &vec1, const Vector< int > &vec2)

    Merges two sorted vectors into a single sorted vector.

- std::pair< int, int > findMaxElement (const Vector< int > &vec)

    Finds the index and value of the maximum element in a vector.

- void VectorUzpildymas ()

    Compares the time taken to fill vectors of varying sizes using std::vector and Vector classes.

5.11.1 Function Documentation

5.11.1.1 findMaxElement()

```
std::pair< int, int > findMaxElement (
            const Vector< int > & vec)
```

Finds the index and value of the maximum element in a vector.

Finds the maximum element and its index in the given vector.

Parameters

vec	The vector in which to find the maximum element.

Returns

A pair containing the index and value of the maximum element. If the vector is empty, returns {-1, -1} to indicate an empty vector.

5.11.1.2 isSorted()

```
bool isSorted (
            const Vector< int > & vec)
```

Checks if a vector is sorted in ascending order.

Checks if the given vector is sorted in ascending order.

Parameters

vec	The vector to be checked.

Returns

True if the vector is sorted, false otherwise.

5.11.1.3 mergeSortedVectors()

```
Vector< int > mergeSortedVectors (
            const Vector< int > & vec1,
            const Vector< int > & vec2)
```

Merges two sorted vectors into a single sorted vector.

Parameters

vec1	The first sorted vector.
vec2	The second sorted vector.

Returns

A new vector containing elements of both input vectors in sorted order.

5.11.1.4 removeDuplicates()

```
void removeDuplicates (
            Vector< int > & vec)
```

Removes duplicate elements from a vector.

Removes duplicate elements from the given vector.

Parameters

vec	The vector from which duplicates are to be removed.

5.11.1.5 sumVector()

```
int sumVector (
            const Vector< int > & vec)
```

Calculates the sum of elements in a vector.

Sums the elements of the given vector.

Parameters

vec	The vector whose elements are to be summed.

Returns

The sum of elements in the vector.

5.11.1.6 VectorExample()

```
void VectorExample ()
```

An example function demonstrating the usage of the Vector class.

Example function demonstrating the usage of the Vector class.

5.11.1.7 VectorUzpildymas()

```
void VectorUzpildymas ()
```

Compares the time taken to fill vectors of varying sizes using std::vector and Vector classes.

Example function demonstrating the usage of filling the vector with data.

5.12 src/Vektorius.h File Reference

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

Classes

- class Vector< T, Allocator >

  A dynamically resizing array similar to std::vector.

Functions

- void VectorExample ()

  Example function demonstrating the usage of the Vector class.
- int sumVector (const Vector< int > &vec)

  Sums the elements of the given vector.
- bool isSorted (const Vector< int > &vec)

  Checks if the given vector is sorted in ascending order.
- void removeDuplicates (Vector< int > &vec)

  Removes duplicate elements from the given vector.
- Vector< int > mergeSortedVectors (const Vector< int > &vec1, const Vector< int > &vec2)

  Merges two sorted vectors into a single sorted vector.
- std::pair< int, int > findMaxElement (const Vector< int > &vec)

  Finds the maximum element and its index in the given vector.
- void VectorUzpildymas ()

  Example function demonstrating the usage of filling the vector with data.

5.12.1 Function Documentation

5.12.1.1 findMaxElement()

```
std::pair< int, int > findMaxElement (
            const Vector< int > & vec)
```

Finds the maximum element and its index in the given vector.

Parameters

vec	The vector in which to search for the maximum element.

Returns

A pair containing the maximum element and its index.

Finds the maximum element and its index in the given vector.

Parameters

vec	The vector in which to find the maximum element.

Returns

A pair containing the index and value of the maximum element. If the vector is empty, returns {-1, -1} to indicate an empty vector.

5.12.1.2 isSorted()

```
bool isSorted (
            const Vector< int > & vec)
```

Checks if the given vector is sorted in ascending order.

Parameters

vec	The vector to be checked.

Returns

True if the vector is sorted, false otherwise.

Checks if the given vector is sorted in ascending order.

Parameters

vec	The vector to be checked.

Returns

True if the vector is sorted, false otherwise.

5.12.1.3 mergeSortedVectors()

```
Vector< int > mergeSortedVectors (
            const Vector< int > & vec1,
            const Vector< int > & vec2)
```

Merges two sorted vectors into a single sorted vector.

Parameters

vec1	The first sorted vector.
vec2	The second sorted vector.

Returns

A new vector containing all elements from both input vectors, sorted in ascending order.

Parameters

vec1	The first sorted vector.
vec2	The second sorted vector.

Returns

A new vector containing elements of both input vectors in sorted order.

5.12.1.4 removeDuplicates()

```
void removeDuplicates (
             Vector< int > & vec)
```

Removes duplicate elements from the given vector.

Parameters

vec	The vector from which duplicate elements are to be removed.

Removes duplicate elements from the given vector.

Parameters

vec	The vector from which duplicates are to be removed.

5.12.1.5 sumVector()

```
int sumVector (
             const Vector< int > & vec)
```

Sums the elements of the given vector.

Parameters

vec	The vector whose elements are to be summed.

Returns

The sum of the elements.

Sums the elements of the given vector.

Parameters

vec	The vector whose elements are to be summed.

Returns

The sum of elements in the vector.

5.12.1.6 VectorExample()

```
void VectorExample ()
```

Example function demonstrating the usage of the Vector class.

Example function demonstrating the usage of the Vector class.

5.12.1.7 VectorUzpildymas()

```
void VectorUzpildymas ()
```

Example function demonstrating the usage of filling the vector with data.

Example function demonstrating the usage of filling the vector with data.

5.13 Vektorius.h

Go to the documentation of this file.

```
#pragma once

#include <memory>
#include <stdexcept>
#include <initializer_list>
#include <algorithm>
#include <iterator>

template<typename T, typename Allocator = std::allocator<T>
class Vector {
public:
    // Member types
    using value_type = T;
    using allocator_type = Allocator;
    using size_type = typename std::allocator_traits<Allocator>::size_type;
    using difference_type = typename std::allocator_traits<Allocator>::difference_type;
    using reference = T&;
    using const_reference = const T&;
    using pointer = typename std::allocator_traits<Allocator>::pointer;
    using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;
    using iterator = T*;
    using const_iterator = const T*;
    using reverse_iterator = std::reverse_iterator<iterator>;
    using const_reverse_iterator = std::reverse_iterator<const_iterator>;
    // Constructors
    Vector() : _data(nullptr), _size(0), _capacity(0), _alloc(Allocator()) {}

    explicit Vector(size_type count, const T& value = T(), const Allocator& alloc = Allocator())
        : _size(count), _capacity(count), _alloc(alloc) {
        _data = _alloc.allocate(_capacity);
        std::uninitialized_fill_n(_data, _size, value);
    }

    Vector(const Vector& other) : _size(other._size), _capacity(other._capacity), _alloc(other._alloc)
{
        _data = _alloc.allocate(_capacity);
        std::uninitialized_copy_n(other._data, _size, _data);
    }

    Vector(Vector&& other) noexcept
        : _data(other._data), _size(other._size), _capacity(other._capacity), _alloc(std::move(other._alloc)) {
        other._data = nullptr;
        other._size = 0;
        other._capacity = 0;
    }

    Vector(std::initializer_list<T> init, const Allocator& alloc = Allocator())
        : _size(init.size()), _capacity(init.size()), _alloc(alloc) {
        _data = _alloc.allocate(_capacity);
        std::uninitialized_copy(init.begin(), init.end(), _data);
    }

    ~Vector() {
        clear();
        _alloc.deallocate(_data, _capacity);
    }

    // Assignment operators
    Vector& operator=(const Vector& other) {
        if (this != &other) {
            clear();
            _alloc.deallocate(_data, _capacity);

            _size = other._size;
            _capacity = other._capacity;
            _alloc = other._alloc;
            _data = _alloc.allocate(_capacity);
            std::uninitialized_copy_n(other._data, _size, _data);
        }
        return *this;
    }

    Vector& operator=(Vector&& other) noexcept {
        if (this != &other) {
            clear();
            _alloc.deallocate(_data, _capacity);

            _data = other._data;
            _size = other._size;
            _capacity = other._capacity;
            _alloc = std::move(other._alloc);

            other._data = nullptr;
            other._size = 0;
            other._capacity = 0;
        }
        return *this;
    }

    // Element access
    reference at(size_type pos) {
        if (pos >= _size) {
            throw std::out_of_range("Vector::at: index out of range");
        }
        return _data[pos];
    }

    const_reference at(size_type pos) const {
        if (pos >= _size) {
            throw std::out_of_range("Vector::at: index out of range");
        }
        return _data[pos];
    }

    reference operator[](size_type pos) {
        return _data[pos];
    }

    const_reference operator[](size_type pos) const {
        return _data[pos];
    }

    const_reference front() const {
        return _data[0];
    }

    reference back() {
        return _data[_size - 1];
    }

    const_reference back() const {
        return _data[_size - 1];
    }

    pointer data() noexcept {
        return _data;
    }

    const_pointer data() const noexcept {
        return _data;
    }

    // Iterators
    iterator begin() noexcept {
        return _data;
    }

    const_iterator begin() const noexcept {
        return _data;
    }

    const_iterator cbegin() const noexcept {
        return _data;
    }

    iterator end() noexcept {
        return _data + _size;
    }

    const_iterator end() const noexcept {
        return _data + _size;
    }

    const_iterator cend() const noexcept {
        return _data + _size;
    }

    reverse_iterator rbegin() noexcept {
        return reverse_iterator(end());
    }

    const_reverse_iterator rbegin() const noexcept {
        return const_reverse_iterator(end());
    }

    reverse_iterator rend() noexcept {
        return reverse_iterator(begin());
    }

    const_reverse_iterator rend() const noexcept {
        return const_reverse_iterator(begin());
    }

    const_reverse_iterator crbegin() const noexcept {
        return const_reverse_iterator(cend());
    }

    const_reverse_iterator crend() const noexcept {
        return const_reverse_iterator(cbegin());
    }

    // Capacity
    bool empty() const noexcept {
        return _size == 0;
    }

    size_type size() const noexcept {
        return _size;
    }

    size_type max_size() const noexcept {
        return std::allocator_traits<Allocator>::max_size(_alloc);
    }

    void reserve(size_type new_cap) {
        if (new_cap > _capacity) {
            pointer new_data = _alloc.allocate(new_cap);
            std::uninitialized_move(_data, _data + _size, new_data);
            std::destroy(_data, _data + _size);
            _alloc.deallocate(_data, _capacity);
            _data = new_data;
            _capacity = new_cap;
        }
    }

    size_type capacity() const noexcept {
        return _capacity;
    }
```

```cpp

    void shrink_to_fit() {
        if (_size < _capacity) {
            pointer new_data = _alloc.allocate(_size);
            std::uninitialized_move(_data, _data + _size, new_data);
            std::destroy(_data, _data + _size);
            _alloc.deallocate(_data, _capacity);
            _data = new_data;
            _capacity = _size;
        }
    }

    // Modifiers
    void clear() noexcept {
        std::destroy(_data, _data + _size);
        _size = 0;
    }

    void push_back(const T& value) {
        if (_size == _capacity) {
            reserve(_capacity == 0 ? 1 : _capacity * 2);
        }
        new(_data + _size) T(value);
        ++_size;
    }

    void push_back(T&& value) {
        if (_size == _capacity) {
            reserve(_capacity == 0 ? 1 : _capacity * 2);
        }
        new(_data + _size) T(std::move(value));
        ++_size;
    }

    void pop_back() {
        if (_size > 0) {
            --_size;
        }
    }

    template <typename... Args>
    reference emplace_back(Args&&... args) {
        if (_size == _capacity) {
            reserve(_capacity == 0 ? 1 : _capacity * 2);
        }
        new(_data + _size) T(std::forward<Args>(args)...);
        ++_size;
        return _data[_size - 1];
    }

    iterator insert(const_iterator pos, const T& value) {
        difference_type offset = pos - begin();
        if (_size == _capacity) {
            reserve(_capacity == 0 ? 1 : _capacity * 2);
        }
        iterator it = begin() + offset;
        if (it != end()) {
            new(it) T(value);
            std::rotate(it, end() - 1, end());
        }
        ++_size;
        return it;
    }

    iterator insert(const_iterator pos, T&& value) {
        difference_type offset = pos - begin();
        if (_size == _capacity) {
            reserve(_capacity == 0 ? 1 : _capacity * 2);
        }
        iterator it = begin() + offset;
        if (it != end()) {
            new(it) T(std::move(value));
            std::rotate(it, end() - 1, end());
        }
        ++_size;
        return it;
    }

    iterator insert(const_iterator pos, size_type count, const T& value) {
        difference_type offset = pos - begin();
        if (_size + count > _capacity) {
            reserve(_capacity + count);
        }
        iterator it = begin() + offset;
        if (it != end()) {
            std::uninitialized_move(it, end(), it + count);
            std::fill(it, it + count, value);
        }
        _size += count;
        return it;
    }

    template <typename InputIt>
    iterator insert(const_iterator pos, InputIt first, InputIt last) {
        difference_type offset = pos - begin();
        size_type count = std::distance(first, last);
        if (_size + count > _capacity) {
            reserve(_capacity + count);
        }
        iterator it = begin() + offset;
        if (it != end()) {
            std::uninitialized_move(it, end(), it + count);
            std::copy(first, last, it);
        }
        _size += count;
        return it;
    }

    iterator erase(const_iterator pos) {
        return erase(pos, pos + 1);
    }

    iterator erase(const_iterator first, const_iterator last) {
        iterator it = const_cast<iterator>(first);
        iterator end_pos = const_cast<iterator>(last);
        std::move(end_pos, end(), it);
        size_type count = std::distance(first, last);
        std::destroy(end() - count, end());
        _size -= count;
        return it;
    }

    void resize(size_type count) {
        if (count > _size) {
            if (count > _capacity) {
                reserve(count);
            }
            for (size_type i = _size; i < count; ++i) {
                new(_data + i) T();
            }
        }
        else if (count < _size) {
            std::destroy(_data + count, _data + _size);
        }
        _size = count;
    }

    void resize(size_type count, const value_type& value) {
        if (count > _size) {
            if (count > _capacity) {
                reserve(count);
            }
            for (size_type i = _size; i < count; ++i) {
                new(_data + i) T(value);
            }
        }
        else if (count < _size) {
            std::destroy(_data + count, _data + _size);
        }
        _size = count;
    }

    void swap(Vector& other) noexcept {
        swap(_data, other._data);
        std::swap(_size, other._size);
        std::swap(_capacity, other._capacity);
        std::swap(_alloc, other._alloc);
    }
private:
     T* _data;
     size_type _size;
     size_type _capacity;
     Allocator _alloc;
};

void VectorExample();

int sumVector(const Vector<int>& vec);

bool isSorted(const Vector<int>& vec);

void removeDuplicates(Vector<int>& vec);

Vector<int> mergeSortedVectors(const Vector<int>& vec1, const Vector<int>& vec2);
```

```

std::pair<int, int> findMaxElement(const Vector<int>& vec);

void VectorUzpildymas();
```

Index