

My Project

Generated by Doxygen 1.11.0

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 Studentas Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Studentas() [1/4]	8
4.1.2.2 Studentas() [2/4]	8
4.1.2.3 ~Studentas()	9
4.1.2.4 Studentas() [3/4]	9
4.1.2.5 Studentas() [4/4]	9
4.1.3 Member Function Documentation	9
4.1.3.1 addGrade()	9
4.1.3.2 calculateFinalGrades()	10
4.1.3.3 getEgzaminoRez()	10
4.1.3.4 getNamudarbuRez()	10
4.1.3.5 getPavarde()	10
4.1.3.6 getVardas()	10
4.1.3.7 operator=() [1/2]	10
4.1.3.8 operator=() [2/2]	11
4.1.3.9 readStudent()	11
4.1.3.10 setEgzaminoRez()	11
4.1.3.11 setPavarde()	11
4.1.3.12 setVardas()	11
4.1.4 Member Data Documentation	12
4.1.4.1 galutinisbalasmediana_	12
4.1.4.2 galutinisbalasvidurkis_	12
4.1.4.3 mediana_	12
4.1.4.4 namudarburezsuma_	12
4.1.4.5 vidurkis_	12
4.2 Vector< T, Allocator > Class Template Reference	12
4.2.1 Detailed Description	15
4.2.2 Member Typedef Documentation	15
4.2.2.1 allocator_type	15
4.2.2.2 const_iterator	15
4.2.2.3 const_pointer	16

4.2.2.4 const_reference	16
4.2.2.5 const_reverse_iterator	16
4.2.2.6 difference_type	16
4.2.2.7 iterator	16
4.2.2.8 pointer	16
4.2.2.9 reference	16
4.2.2.10 reverse_iterator	17
4.2.2.11 size_type	17
4.2.2.12 value_type	17
4.2.3 Constructor & Destructor Documentation	17
4.2.3.1 Vector() [1/5]	17
4.2.3.2 Vector() [2/5]	17
4.2.3.3 Vector() [3/5]	17
4.2.3.4 Vector() [4/5]	18
4.2.3.5 Vector() [5/5]	18
4.2.3.6 ~Vector()	18
4.2.4 Member Function Documentation	18
4.2.4.1 at() [1/2]	18
4.2.4.2 at() [2/2]	19
4.2.4.3 back() [1/2]	19
4.2.4.4 back() [2/2]	20
4.2.4.5 begin() [1/2]	20
4.2.4.6 begin() [2/2]	20
4.2.4.7 capacity()	20
4.2.4.8 cbegin()	21
4.2.4.9 cend()	21
4.2.4.10 clear()	21
4.2.4.11 crbegin()	21
4.2.4.12 crend()	21
4.2.4.13 data() [1/2]	22
4.2.4.14 data() [2/2]	22
4.2.4.15 emplace_back()	22
4.2.4.16 empty()	23
4.2.4.17 end() [1/2]	23
4.2.4.18 end() [2/2]	23
4.2.4.19 erase() [1/2]	23
4.2.4.20 erase() [2/2]	23
4.2.4.21 front()	24
4.2.4.22 insert() [1/4]	24
4.2.4.23 insert() [2/4]	24
4.2.4.24 insert() [3/4]	25
4.2.4.25 insert() [4/4]	25

4.2.4.26 max_size()	26
4.2.4.27 operator=() [1/2]	26
4.2.4.28 operator=() [2/2]	26
4.2.4.29 operator[]() [1/2]	26
4.2.4.30 operator[]() [2/2]	27
4.2.4.31 pop_back()	27
4.2.4.32 push_back() [1/2]	27
4.2.4.33 push_back() [2/2]	27
4.2.4.34 rbegin() [1/2]	28
4.2.4.35 rbegin() [2/2]	28
4.2.4.36 rend() [1/2]	28
4.2.4.37 rend() [2/2]	28
4.2.4.38 reserve()	28
4.2.4.39 resize() [1/2]	29
4.2.4.40 resize() [2/2]	29
4.2.4.41 shrink_to_fit()	29
4.2.4.42 size()	29
4.2.4.43 swap()	30
4.3 Zmogus Class Reference	30
4.3.1 Detailed Description	30
4.3.2 Constructor & Destructor Documentation	31
4.3.2.1 ~Zmogus()	31
4.3.3 Member Function Documentation	31
4.3.3.1 getEgzaminoRez()	31
4.3.3.2 getNamudarbuRez()	31
4.3.3.3 getPavarde()	31
4.3.3.4 getVardas()	31
5 File Documentation	33
5.1 src/ConsoleApplication1.cpp File Reference	33
5.1.1 Function Documentation	33
5.1.1.1 main()	33
5.2 src/MokiniuProcessing.cpp File Reference	33
5.2.1 Function Documentation	34
5.2.1.1 ContainsNumbers()	34
5.2.1.2 GeneruotiPavardes()	34
5.2.1.3 GeneruotiVardus()	34
5.2.1.4 PatikrintiTeigiamajiSkaiciu()	35
5.3 src/MokiniuProcessing.h File Reference	36
5.3.1 Function Documentation	36
5.3.1.1 ContainsNumbers()	36
5.3.1.2 GeneruotiPavardes()	37

5.3.1.3 GeneruotiVardus()	37
5.3.1.4 PatikrintiTeigiamajiSkaiciu()	37
5.4 MokiniuProcessing.h	38
5.5 src/Skaiciavimaidarbai.cpp File Reference	38
5.5.1 Function Documentation	39
5.5.1.1 compareByGalutinisMed()	39
5.5.1.2 compareByGalutinisVid()	39
5.5.1.3 GenerateRandomGrade()	39
5.5.1.4 GeneruotiFaila()	40
5.5.1.5 Mediana()	40
5.5.1.6 NeraFailo()	40
5.5.1.7 Netinkamalvestis()	40
5.6 src/Skaiciavimaidarbai.h File Reference	41
5.6.1 Function Documentation	41
5.6.1.1 compareByGalutinisMed()	41
5.6.1.2 compareByGalutinisVid()	41
5.6.1.3 GenerateRandomGrade()	42
5.6.1.4 GeneruotiFaila()	42
5.6.1.5 Mediana()	42
5.6.1.6 NeraFailo()	43
5.6.1.7 Netinkamalvestis()	43
5.7 Skaiciavimaidarbai.h	43
5.8 src/Studentas.cpp File Reference	44
5.8.1 Function Documentation	44
5.8.1.1 medianoslyginimas()	44
5.8.1.2 operator<<()	45
5.8.1.3 operator>>()	45
5.8.1.4 partitionStudents1()	45
5.8.1.5 partitionStudents2()	45
5.8.1.6 partitionStudents3()	46
5.8.1.7 pavardeslyginimas()	46
5.8.1.8 PrintStudents()	46
5.8.1.9 readAndProcessData()	47
5.8.1.10 sortStudents()	47
5.8.1.11 testConstructors()	47
5.8.1.12 vardolyginimas()	47
5.8.1.13 vidurkiolyginimas()	48
5.8.1.14 WriteNormalStudents()	48
5.8.1.15 WriteWeirdStudents()	48
5.9 src/Studentas.h File Reference	48
5.9.1 Function Documentation	49
5.9.1.1 medianoslyginimas()	49

5.9.1.2 partitionStudents1()	50
5.9.1.3 partitionStudents2()	50
5.9.1.4 partitionStudents3()	50
5.9.1.5 pavardeslyginimas()	51
5.9.1.6 PrintStudents()	51
5.9.1.7 readAndProcessData()	51
5.9.1.8 sortStudents()	52
5.9.1.9 testConstructors()	52
5.9.1.10 vardolyginimas()	52
5.9.1.11 vidurkiolyginimas()	53
5.9.1.12 WriteNormalStudents()	54
5.9.1.13 WriteWeirdStudents()	54
5.10 Studentas.h	54
5.11 src/Vektorius.cpp File Reference	55
5.11.1 Function Documentation	56
5.11.1.1 findMaxElement()	56
5.11.1.2 isSorted()	56
5.11.1.3 mergeSortedVectors()	56
5.11.1.4 removeDuplicates()	57
5.11.1.5 sumVector()	57
5.11.1.6 VectorExample()	57
5.11.1.7 VectorUzpildymas()	58
5.12 src/Vektorius.h File Reference	58
5.12.1 Function Documentation	58
5.12.1.1 findMaxElement()	58
5.12.1.2 isSorted()	59
5.12.1.3 mergeSortedVectors()	59
5.12.1.4 removeDuplicates()	60
5.12.1.5 sumVector()	60
5.12.1.6 VectorExample()	61
5.12.1.7 VectorUzpildymas()	61
5.13 Vektorius.h	61
Index	67

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Vector< T, Allocator >	12
Vector< double >	12
Zmogus	30
Studentas	7

Chapter 2

Class Index

2.1 Class List

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

Studentas	Class representing a student	7
Vector< T, Allocator >	A dynamically resizing array similar to std::vector	12
Zmogus	Abstract class representing a person	30

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/ ConsoleApplication1.cpp	33
src/ MokiniuProcessing.cpp	33
src/ MokiniuProcessing.h	36
src/ Skaiciavimaidarbai.cpp	38
src/ Skaiciavimaidarbai.h	41
src/ Studentas.cpp	44
src/ Studentas.h	48
src/ Vektorius.cpp	55
src/ Vektorius.h	58

Chapter 4

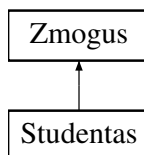
Class Documentation

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 implementation
constructor 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

<i>is</i>	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

<i>is</i>	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

<i>T</i>	The type of elements stored in the vector.
<i>Allocator</i>	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

<i>count</i>	The number of elements in the vector.
<i>value</i>	The value to initialize the elements with.
<i>alloc</i>	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

<i>other</i>	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

<i>other</i>	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

<i>init</i>	The initializer list to initialize the elements of the vector.
<i>alloc</i>	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

<i>pos</i>	The position of the element to access.
------------	--

Returns

Reference to the element at the specified position.

Exceptions

<i>std::out_of_range</i>	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

<i>pos</i>	The position of the element to access.
------------	--

Returns

Const reference to the element at the specified position.

Exceptions

<i>std::out_of_range</i>	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

<i>Args</i>	Types of arguments to forward to the constructor of the element.
-------------	--

Parameters

<i>args</i>	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

<i>first</i>	Iterator to the beginning of the range to be erased.
<i>last</i>	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

<i>pos</i>	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

<i>pos</i>	Iterator pointing to the position where the element should be inserted.
<i>value</i>	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

<i>Input↔ It</i>	Type of the input iterator.
----------------------	-----------------------------

Parameters

<i>pos</i>	Iterator pointing to the position where the elements should be inserted.
<i>first</i>	Iterator to the beginning of the range.
<i>last</i>	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

<i>pos</i>	Iterator pointing to the position where the elements should be inserted.
<i>count</i>	The number of elements to insert.
<i>value</i>	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

<i>pos</i>	Iterator pointing to the position where the element should be inserted.
<i>value</i>	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

<i>other</i>	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

<i>other</i>	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

<i>pos</i>	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

<i>pos</i>	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

<i>value</i>	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

<i>value</i>	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

<i>new_cap</i>	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

<i>count</i>	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

<i>count</i>	The new size of the vector.
<i>value</i>	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

<i>other</i>	The vector to swap with.
--------------	--------------------------

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

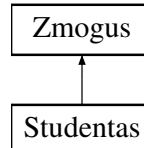
- src/[Vektorius.h](#)

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

<i>str</i>	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

<code>skaicius</code>	The number to check.
-----------------------	----------------------

Exceptions

<code>invalid_argument</code>	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

<i>str</i>	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

<code>skaicius</code>	The number to check.
-----------------------	----------------------

Exceptions

<code>invalid_argument</code>	If the number is not within the range 0 to 10.
-------------------------------	--

5.4 MokiniuProcessing.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <string>
00003 #include <vector>
00004 #include <stdexcept>
00005 #include <iostream>
00006 #include <locale>
00007 #include <numeric>
00008 #include <fstream>
00009 #include <sstream>
00010 #include <cctype>
00011 #include <algorithm>
00012 #include <chrono>
00013 #include <iomanip>
00014 #include <stdlib.h>
00015 #include "Studentas.h"
00016
00017 using namespace std;
00018
00027 string GeneruotiVardus();
00028
00037 string GeneruotiPavardes();
00038
00048 void PatikrintiTeigiamajiSkaiciu(double skaicius);
00049
00059 bool ContainsNumbers(const string& str);
```

5.5 src/Skaiciavimaidarbai.cpp File Reference

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

Functions

- void `Netinkamalvestis()`
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

<i>a</i>	The first student to compare.
<i>b</i>	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

<i>a</i>	The first student to compare.
<i>b</i>	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

<i>pavadinimas</i>	The name of the file to generate.
<i>studentuskaicius</i>	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

<i>vec</i>	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 Netinkamalvestis()

```
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 [Netinkamalvestis](#) ()
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

<i>a</i>	The first student to compare.
<i>b</i>	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

<i>a</i>	The first student to compare.
<i>b</i>	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

<i>pavadinimas</i>	The name of the file to generate.
<i>studentuskaicius</i>	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 Netinkamalvestis()

```
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.](#)

```
00001 #pragma once
00002 #include "MokiniuProcessing.h"
00003
00010 void NetinkamaIvestis();
00011
00018 void NeraFailo();
00019
00030 double Mediana(Vector<double>& vec);
00031
00039 double GenerateRandomGrade();
00040
00051 void GeneruotiFaila(const string& pavadinimas, int studentuskaicius);
00052
00063 bool compareByGalutinisVid(const Studentas& a, const Studentas& b);
00064
00075 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

<i>a</i>	The first student.
<i>b</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>normalus</i>	The Vector to store normal students.
<i>nenormalus</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>nenormalus</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>normalus</i>	The Vector to store normal students.
<i>nenormalus</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>studentai</i>	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

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

5.8.1.10 sortStudents()

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

Sorts students using the STL sort function.

Parameters

<i>studentai</i>	The Vector to store the read students.
<i>sortpasirinkimas</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>normalus</i>	The Vector of normal students.
-----------------	--

5.8.1.15 WriteWeirdStudents()

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

Writes weird students to a file.

Parameters

<i>nenormalus</i>	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 &namudarbais, 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

<i>a</i>	The first student.
<i>b</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>normalus</i>	The Vector to store normal students.
<i>nenormalus</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>nenormalus</i>	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

<i>studentai</i>	The Vector to store the read students.
<i>normalus</i>	The Vector to store normal students.
<i>nenormalus</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>studentai</i>	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

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

<i>studentai</i>	The Vector to store the read students.
<i>sortpasirinkimas</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>a</i>	The first student.
<i>b</i>	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

<i>normalus</i>	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

<i>nenormalus</i>	The Vector of weird students.
-------------------	---

5.10 Studentas.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <string>
00004 #include "Vektorius.h"
00005
00012 class Zmogus {
00013 public:
00014     virtual ~Zmogus() = default;
00015     // Getters
00021     virtual std::string getVardas() const = 0;
00027     virtual std::string getPavarde() const = 0;
00033     virtual double getEgzaminoRez() const = 0;
00039     virtual Vector<double>& getNamudarbuRez() = 0;
00040 };
00047 class Studentas : public Zmogus {
00048 private:
```

```

00049     std::string vardas_;
00050     std::string pavarde_;
00051     Vector<double> namudarburez_;
00052     double egzaminorez_;
00053
00054 public:
00055     double namudarburezsuma_;
00056     double vidurkis_;
00057     double galutinisbalasvidurkis_;
00058     double mediana_;
00059     double galutinisbalasmediana_;
00061     // Constructors
00062     Studentas();
00063     Studentas(std::istream& is);
00065     // Getters
00066     std::string getVardas() const { return vardas_; }
00067     std::string getPavarde() const { return pavarde_; }
00068     double getEgzaminoRez() const { return egzaminorez_; }
00069     Vector<double>& getNamudarbuRez() { return namudarburez_; }
00071     // Setters
00072     void setVardas(const std::string& vardas) { vardas_ = vardas; }
00073     void setPavarde(const std::string& pavarde) { pavarde_ = pavarde; }
00074     void setEgzaminoRez(double egzaminorez) { egzaminorez_ = egzaminorez; }
00075     void addGrade(double grade) { namudarburez_.push_back(grade); }
00077     // Other member functions
00078     std::istream& readStudent(std::istream&);
00079     void calculateFinalGrades();
00081     //Destructor
00082     ~Studentas();
00083     Studentas(const Studentas& other);
00084     Studentas& operator=(const Studentas& other);
00085     Studentas(Studentas& other) noexcept;
00086     Studentas& operator=(Studentas&& other) noexcept;
00088 };
00089 // Negalima kurti zmogus objekto -
00090 // Zmogus a;
00091
00092 // Comparison functions
00093 bool vardolyginimas(const Studentas& a, const Studentas& b);
00094 bool pavardeslyginimas(const Studentas& a, const Studentas& b);
00095 bool vidurkiolyginimas(const Studentas& a, const Studentas& b);
00096 bool medianoslyginimas(const Studentas& a, const Studentas& b);
00098 // Utility functions
00099 void PrintStudents(const Vector<Studentas>& studentai);
00100 void readAndProcessData(const std::string& filename, Vector<Studentas>& studentai, int& namudarbais,
    int studentuskaicius);
00101 void sortStudents(Vector<Studentas>& studentai, int sortpasirinkimas);
00102 void partitionStudents1(const Vector<Studentas>& studentai, Vector<Studentas>& normalus,
    Vector<Studentas>& nenormalus);
00103 void partitionStudents2(Vector<Studentas>& studentai, Vector<Studentas>& nenormalus);
00104 void partitionStudents3(Vector<Studentas>& studentai, Vector<Studentas>& normalus, Vector<Studentas>&
    nenormalus);
00105 void WriteWeirdStudents(Vector<Studentas>& nenormalus);
00106 void WriteNormalStudents(Vector<Studentas>& normalus);
00107 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

<code>vec</code>	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

<code>vec</code>	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

<i>vec1</i>	The first sorted vector.
<i>vec2</i>	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

<i>vec</i>	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

<i>vec</i>	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

<code>vec</code>	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

<code>vec</code>	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

<code>vec</code>	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

<code>vec</code>	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

<i>vec1</i>	The first sorted vector.
<i>vec2</i>	The second sorted vector.

Returns

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

Parameters

<i>vec1</i>	The first sorted vector.
<i>vec2</i>	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

<i>vec</i>	The vector from which duplicate elements are to be removed.
------------	---

Removes duplicate elements from the given vector.

Parameters

<i>vec</i>	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

<i>vec</i>	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.](#)

```
00001 #pragma once
00002
00003 #include <memory>
00004 #include <stdexcept>
00005 #include <initializer_list>
00006 #include <algorithm>
00007 #include <iterator>
00008
00015 template<typename T, typename Allocator = std::allocator<T>
00016 class Vector {
00017 public:
00018     // Member types
00019     using value_type = T;
00020     using allocator_type = Allocator;
00021     using size_type = typename std::allocator_traits<Allocator>::size_type;
00022     using difference_type = typename std::allocator_traits<Allocator>::difference_type;
00023     using reference = T&;
00024     using const_reference = const T&;
00025     using pointer = typename std::allocator_traits<Allocator>::pointer;
00026     using const_pointer = typename std::allocator_traits<Allocator>::const_pointer;
00027     using iterator = T*;
00028     using const_iterator = const T*;
00029     using reverse_iterator = std::reverse_iterator<iterator>;
00030     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00032     // Constructors
00036     Vector() : _data(nullptr), _size(0), _capacity(0), _alloc(Allocator()) {}
00037
00045     explicit Vector(size_type count, const T& value = T(), const Allocator& alloc = Allocator())
00046         : _size(count), _capacity(count), _alloc(alloc) {
00047         _data = _alloc.allocate(_capacity);
00048         std::uninitialized_fill_n(_data, _size, value);
00049     }
00050
00056     Vector(const Vector& other) : _size(other._size), _capacity(other._capacity), _alloc(other._alloc)
    {
```

```

00057         _data = _alloc.allocate(_capacity);
00058         std::uninitialized_copy_n(other._data, _size, _data);
00059     }
00060
00061     Vector(Vector&& other) noexcept
00062         : _data(other._data), _size(other._size), _capacity(other._capacity),
00063       _alloc(std::move(other._alloc)) {
00064         other._data = nullptr;
00065         other._size = 0;
00066         other._capacity = 0;
00067     }
00068
00069     Vector(std::initializer_list<T> init, const Allocator& alloc = Allocator())
00070         : _size(init.size()), _capacity(init.size()), _alloc(alloc) {
00071         _data = _alloc.allocate(_capacity);
00072         std::uninitialized_copy(init.begin(), init.end(), _data);
00073     }
00074
00075     ~Vector() {
00076         clear();
00077         _alloc.deallocate(_data, _capacity);
00078     }
00079
00080     // Assignment operators
00081     Vector& operator=(const Vector& other) {
00082         if (this != &other) {
00083             clear();
00084             _alloc.deallocate(_data, _capacity);
00085
00086             _size = other._size;
00087             _capacity = other._capacity;
00088             _alloc = other._alloc;
00089             _data = _alloc.allocate(_capacity);
00090             std::uninitialized_copy_n(other._data, _size, _data);
00091         }
00092         return *this;
00093     }
00094
00095     Vector& operator=(Vector&& other) noexcept {
00096         if (this != &other) {
00097             clear();
00098             _alloc.deallocate(_data, _capacity);
00099
00100             _data = other._data;
00101             _size = other._size;
00102             _capacity = other._capacity;
00103             _alloc = std::move(other._alloc);
00104
00105             other._data = nullptr;
00106             other._size = 0;
00107             other._capacity = 0;
00108         }
00109         return *this;
00110     }
00111
00112     // Element access
00113     reference at(size_type pos) {
00114         if (pos >= _size) {
00115             throw std::out_of_range("Vector::at: index out of range");
00116         }
00117         return _data[pos];
00118     }
00119
00120     const_reference at(size_type pos) const {
00121         if (pos >= _size) {
00122             throw std::out_of_range("Vector::at: index out of range");
00123         }
00124         return _data[pos];
00125     }
00126
00127     reference operator[](size_type pos) {
00128         return _data[pos];
00129     }
00130
00131     const_reference operator[](size_type pos) const {
00132         return _data[pos];
00133     }
00134
00135     const_reference front() const {
00136         return _data[0];
00137     }
00138
00139     reference back() {
00140         return _data[_size - 1];
00141     }
00142
00143     const_reference back() const {

```

```

00210         return _data[_size - 1];
00211     }
00212
00218     pointer data() noexcept {
00219         return _data;
00220     }
00221
00227     const_pointer data() const noexcept {
00228         return _data;
00229     }
00230
00231     // Iterators
00237     iterator begin() noexcept {
00238         return _data;
00239     }
00240
00246     const_iterator begin() const noexcept {
00247         return _data;
00248     }
00249
00255     const_iterator cbegin() const noexcept {
00256         return _data;
00257     }
00258
00264     iterator end() noexcept {
00265         return _data + _size;
00266     }
00267
00273     const_iterator end() const noexcept {
00274         return _data + _size;
00275     }
00276
00282     const_iterator cend() const noexcept {
00283         return _data + _size;
00284     }
00285
00291     reverse_iterator rbegin() noexcept {
00292         return reverse_iterator(end());
00293     }
00294
00300     const_reverse_iterator rbegin() const noexcept {
00301         return const_reverse_iterator(end());
00302     }
00303
00309     reverse_iterator rend() noexcept {
00310         return reverse_iterator(begin());
00311     }
00312
00318     const_reverse_iterator rend() const noexcept {
00319         return const_reverse_iterator(begin());
00320     }
00321
00327     const_reverse_iterator crbegin() const noexcept {
00328         return const_reverse_iterator(cend());
00329     }
00330
00336     const_reverse_iterator crend() const noexcept {
00337         return const_reverse_iterator(cbegin());
00338     }
00339
00340     // Capacity
00346     bool empty() const noexcept {
00347         return _size == 0;
00348     }
00349
00355     size_type size() const noexcept {
00356         return _size;
00357     }
00358
00364     size_type max_size() const noexcept {
00365         return std::allocator_traits<Allocator>::max_size(_alloc);
00366     }
00367
00373     void reserve(size_type new_cap) {
00374         if (new_cap > _capacity) {
00375             pointer new_data = _alloc.allocate(new_cap);
00376             std::uninitialized_move(_data, _data + _size, new_data);
00377             std::destroy(_data, _data + _size);
00378             _alloc.deallocate(_data, _capacity);
00379             _data = new_data;
00380             _capacity = new_cap;
00381         }
00382     }
00383
00389     size_type capacity() const noexcept {
00390         return _capacity;
00391     }

```

```

00392
00396 void shrink_to_fit() {
00397     if (_size < _capacity) {
00398         pointer new_data = _alloc.allocate(_size);
00399         std::uninitialized_move(_data, _data + _size, new_data);
00400         std::destroy(_data, _data + _size);
00401         _alloc.deallocate(_data, _capacity);
00402         _data = new_data;
00403         _capacity = _size;
00404     }
00405 }
00406
00407 // Modifiers
00411 void clear() noexcept {
00412     std::destroy(_data, _data + _size);
00413     _size = 0;
00414 }
00415
00421 void push_back(const T& value) {
00422     if (_size == _capacity) {
00423         reserve(_capacity == 0 ? 1 : _capacity * 2);
00424     }
00425     new(_data + _size) T(value);
00426     ++_size;
00427 }
00428
00434 void push_back(T&& value) {
00435     if (_size == _capacity) {
00436         reserve(_capacity == 0 ? 1 : _capacity * 2);
00437     }
00438     new(_data + _size) T(std::move(value));
00439     ++_size;
00440 }
00441
00445 void pop_back() {
00446     if (_size > 0) {
00447         --_size;
00448     }
00449 }
00450
00458 template <typename... Args>
00459 reference emplace_back(Args&&... args) {
00460     if (_size == _capacity) {
00461         reserve(_capacity == 0 ? 1 : _capacity * 2);
00462     }
00463     new(_data + _size) T(std::forward<Args>(args)...);
00464     ++_size;
00465     return _data[_size - 1];
00466 }
00467
00475 iterator insert(const_iterator pos, const T& value) {
00476     difference_type offset = pos - begin();
00477     if (_size == _capacity) {
00478         reserve(_capacity == 0 ? 1 : _capacity * 2);
00479     }
00480     iterator it = begin() + offset;
00481     if (it != end()) {
00482         new(it) T(value);
00483         std::rotate(it, end() - 1, end());
00484     }
00485     ++_size;
00486     return it;
00487 }
00488
00496 iterator insert(const_iterator pos, T&& value) {
00497     difference_type offset = pos - begin();
00498     if (_size == _capacity) {
00499         reserve(_capacity == 0 ? 1 : _capacity * 2);
00500     }
00501     iterator it = begin() + offset;
00502     if (it != end()) {
00503         new(it) T(std::move(value));
00504         std::rotate(it, end() - 1, end());
00505     }
00506     ++_size;
00507     return it;
00508 }
00509
00518 iterator insert(const_iterator pos, size_type count, const T& value) {
00519     difference_type offset = pos - begin();
00520     if (_size + count > _capacity) {
00521         reserve(_capacity + count);
00522     }
00523     iterator it = begin() + offset;
00524     if (it != end()) {
00525         std::uninitialized_move(it, end(), it + count);
00526         std::fill(it, it + count, value);

```



```

00527     }
00528     _size += count;
00529     return it;
00530 }
00531
00541 template <typename InputIt>
00542 iterator insert(const_iterator pos, InputIt first, InputIt last) {
00543     difference_type offset = pos - begin();
00544     size_type count = std::distance(first, last);
00545     if (_size + count > _capacity) {
00546         reserve(_capacity + count);
00547     }
00548     iterator it = begin() + offset;
00549     if (it != end()) {
00550         std::uninitialized_move(it, end(), it + count);
00551         std::copy(first, last, it);
00552     }
00553     _size += count;
00554     return it;
00555 }
00556
00563 iterator erase(const_iterator pos) {
00564     return erase(pos, pos + 1);
00565 }
00566
00574 iterator erase(const_iterator first, const_iterator last) {
00575     iterator it = const_cast<iterator>(first);
00576     iterator end_pos = const_cast<iterator>(last);
00577     std::move(end_pos, end(), it);
00578     size_type count = std::distance(first, last);
00579     std::destroy(end() - count, end());
00580     _size -= count;
00581     return it;
00582 }
00583
00592 void resize(size_type count) {
00593     if (count > _size) {
00594         if (count > _capacity) {
00595             reserve(count);
00596         }
00597         for (size_type i = _size; i < count; ++i) {
00598             new(_data + i) T();
00599         }
00600     }
00601     else if (count < _size) {
00602         std::destroy(_data + count, _data + _size);
00603     }
00604     _size = count;
00605 }
00606
00616 void resize(size_type count, const value_type& value) {
00617     if (count > _size) {
00618         if (count > _capacity) {
00619             reserve(count);
00620         }
00621         for (size_type i = _size; i < count; ++i) {
00622             new(_data + i) T(value);
00623         }
00624     }
00625     else if (count < _size) {
00626         std::destroy(_data + count, _data + _size);
00627     }
00628     _size = count;
00629 }
00630
00636 void swap(Vector& other) noexcept {
00637     swap(_data, other._data);
00638     std::swap(_size, other._size);
00639     std::swap(_capacity, other._capacity);
00640     std::swap(_alloc, other._alloc);
00641 }
00642 private:
00643     T* _data;
00644     size_type _size;
00645     size_type _capacity;
00646     Allocator _alloc;
00647 };
00648
00652 void VectorExample();
00653
00660 int sumVector(const Vector<int>& vec);
00661
00668 bool isSorted(const Vector<int>& vec);
00669
00675 void removeDuplicates(Vector<int>& vec);
00676
00684 Vector<int> mergeSortedVectors(const Vector<int>& vec1, const Vector<int>& vec2);

```

```
00685
00692 std::pair<int, int> findMaxElement(const Vector<int>& vec);
00693
00697 void VectorUzpildymas();
```

Index

- ~Studentas
 - Studentas, [9](#)
- ~Vector
 - Vector< T, Allocator >, [18](#)
- ~Zmogus
 - Zmogus, [31](#)
- addGrade
 - Studentas, [9](#)
- allocator_type
 - Vector< T, Allocator >, [15](#)
- at
 - Vector< T, Allocator >, [18](#), [19](#)
- back
 - Vector< T, Allocator >, [19](#)
- begin
 - Vector< T, Allocator >, [20](#)
- calculateFinalGrades
 - Studentas, [9](#)
- capacity
 - Vector< T, Allocator >, [20](#)
- cbegin
 - Vector< T, Allocator >, [20](#)
- cend
 - Vector< T, Allocator >, [21](#)
- clear
 - Vector< T, Allocator >, [21](#)
- compareByGalutinisMed
 - Skaiciavimaidarbai.cpp, [39](#)
 - Skaiciavimaidarbai.h, [41](#)
- compareByGalutinisVid
 - Skaiciavimaidarbai.cpp, [39](#)
 - Skaiciavimaidarbai.h, [41](#)
- ConsoleApplication1.cpp
 - main, [33](#)
- const_iterator
 - Vector< T, Allocator >, [15](#)
- const_pointer
 - Vector< T, Allocator >, [15](#)
- const_reference
 - Vector< T, Allocator >, [16](#)
- const_reverse_iterator
 - Vector< T, Allocator >, [16](#)
- ContainsNumbers
 - MokiniuProcessing.cpp, [34](#)
 - MokiniuProcessing.h, [36](#)
- crbegin
 - Vector< T, Allocator >, [21](#)
- crend
 - Vector< T, Allocator >, [21](#)
- data
 - Vector< T, Allocator >, [21](#), [22](#)
- difference_type
 - Vector< T, Allocator >, [16](#)
- emplace_back
 - Vector< T, Allocator >, [22](#)
- empty
 - Vector< T, Allocator >, [22](#)
- end
 - Vector< T, Allocator >, [23](#)
- erase
 - Vector< T, Allocator >, [23](#)
- findMaxElement
 - Vektorius.cpp, [56](#)
 - Vektorius.h, [58](#)
- front
 - Vector< T, Allocator >, [24](#)
- galutinisbalasmediana_
 - Studentas, [12](#)
- galutinisbalasvidurkis_
 - Studentas, [12](#)
- GenerateRandomGrade
 - Skaiciavimaidarbai.cpp, [39](#)
 - Skaiciavimaidarbai.h, [42](#)
- GeneruotiFaila
 - Skaiciavimaidarbai.cpp, [39](#)
 - Skaiciavimaidarbai.h, [42](#)
- GeneruotiPavardes
 - MokiniuProcessing.cpp, [34](#)
 - MokiniuProcessing.h, [37](#)
- GeneruotiVardus
 - MokiniuProcessing.cpp, [34](#)
 - MokiniuProcessing.h, [37](#)
- getEgzaminoRez
 - Studentas, [10](#)
 - Zmogus, [31](#)
- getNamudarbuRez
 - Studentas, [10](#)
 - Zmogus, [31](#)
- getPavarde
 - Studentas, [10](#)
 - Zmogus, [31](#)
- getVardas
 - Studentas, [10](#)

- Zmogus, 31
- insert
 - Vector< T, Allocator >, 24, 25
- isSorted
 - Vektorius.cpp, 56
 - Vektorius.h, 59
- iterator
 - Vector< T, Allocator >, 16
- main
 - ConsoleApplication1.cpp, 33
- max_size
 - Vector< T, Allocator >, 25
- Mediana
 - Skaiciavimaidarbai.cpp, 40
 - Skaiciavimaidarbai.h, 42
- mediana_
 - Studentas, 12
- medianoslyginimas
 - Studentas.cpp, 44
 - Studentas.h, 49
- mergeSortedVectors
 - Vektorius.cpp, 56
 - Vektorius.h, 59
- MokiniuProcessing.cpp
 - ContainsNumbers, 34
 - GeneruotiPavardes, 34
 - GeneruotiVardus, 34
 - PatikrintiTeigiamajiSkaiciu, 34
- MokiniuProcessing.h
 - ContainsNumbers, 36
 - GeneruotiPavardes, 37
 - GeneruotiVardus, 37
 - PatikrintiTeigiamajiSkaiciu, 37
- namudarburesuma_
 - Studentas, 12
- NeraFailo
 - Skaiciavimaidarbai.cpp, 40
 - Skaiciavimaidarbai.h, 43
- Netinkamalvestis
 - Skaiciavimaidarbai.cpp, 40
 - Skaiciavimaidarbai.h, 43
- operator<<
 - Studentas.cpp, 45
- operator>>
 - Studentas.cpp, 45
- operator=
 - Studentas, 10
 - Vector< T, Allocator >, 26
- operator[]
 - Vector< T, Allocator >, 26, 27
- partitionStudents1
 - Studentas.cpp, 45
 - Studentas.h, 50
- partitionStudents2
 - Studentas.cpp, 45
 - Studentas.h, 50
- partitionStudents3
 - Studentas.cpp, 46
 - Studentas.h, 50
- PatikrintiTeigiamajiSkaiciu
 - MokiniuProcessing.cpp, 34
 - MokiniuProcessing.h, 37
- pavardeslyginimas
 - Studentas.cpp, 46
 - Studentas.h, 51
- pointer
 - Vector< T, Allocator >, 16
- pop_back
 - Vector< T, Allocator >, 27
- PrintStudents
 - Studentas.cpp, 46
 - Studentas.h, 51
- push_back
 - Vector< T, Allocator >, 27
- rbegin
 - Vector< T, Allocator >, 28
- readAndProcessData
 - Studentas.cpp, 46
 - Studentas.h, 51
- readStudent
 - Studentas, 11
- reference
 - Vector< T, Allocator >, 16
- removeDuplicates
 - Vektorius.cpp, 57
 - Vektorius.h, 60
- rend
 - Vector< T, Allocator >, 28
- reserve
 - Vector< T, Allocator >, 28
- resize
 - Vector< T, Allocator >, 29
- reverse_iterator
 - Vector< T, Allocator >, 16
- setEgzaminoRez
 - Studentas, 11
- setPavarde
 - Studentas, 11
- setVardas
 - Studentas, 11
- shrink_to_fit
 - Vector< T, Allocator >, 29
- size
 - Vector< T, Allocator >, 29
- size_type
 - Vector< T, Allocator >, 17
- Skaiciavimaidarbai.cpp
 - compareByGalutinisMed, 39
 - compareByGalutinisVid, 39
 - GenerateRandomGrade, 39
 - GeneruotiFaila, 39

- Mediana, [40](#)
- NeraFailo, [40](#)
- Netinkamalvestis, [40](#)
- Skaiciavimaidarbai.h
 - compareByGalutinisMed, [41](#)
 - compareByGalutinisVid, [41](#)
 - GenerateRandomGrade, [42](#)
 - GeneruotiFaila, [42](#)
 - Mediana, [42](#)
 - NeraFailo, [43](#)
 - Netinkamalvestis, [43](#)
- sortStudents
 - Studentas.cpp, [47](#)
 - Studentas.h, [52](#)
- src/ConsoleApplication1.cpp, [33](#)
- src/MokiniuProcessing.cpp, [33](#)
- src/MokiniuProcessing.h, [36](#), [38](#)
- src/Skaiciavimaidarbai.cpp, [38](#)
- src/Skaiciavimaidarbai.h, [41](#), [43](#)
- src/Studentas.cpp, [44](#)
- src/Studentas.h, [48](#), [54](#)
- src/Vektorius.cpp, [55](#)
- src/Vektorius.h, [58](#), [61](#)
- Studentas, [7](#)
 - ~Studentas, [9](#)
 - addGrade, [9](#)
 - calculateFinalGrades, [9](#)
 - galutinisbalasmediana_, [12](#)
 - galutinisbalasvidurkis_, [12](#)
 - getEgzaminoRez, [10](#)
 - getNamudarduRez, [10](#)
 - getPavarde, [10](#)
 - getVardas, [10](#)
 - mediana_, [12](#)
 - namudarburezsuma_, [12](#)
 - operator=, [10](#)
 - readStudent, [11](#)
 - setEgzaminoRez, [11](#)
 - setPavarde, [11](#)
 - setVardas, [11](#)
 - Studentas, [8](#), [9](#)
 - vidurkis_, [12](#)
- Studentas.cpp
 - medianoslyginimas, [44](#)
 - operator<<, [45](#)
 - operator>>, [45](#)
 - partitionStudents1, [45](#)
 - partitionStudents2, [45](#)
 - partitionStudents3, [46](#)
 - pavardeslyginimas, [46](#)
 - PrintStudents, [46](#)
 - readAndProcessData, [46](#)
 - sortStudents, [47](#)
 - testConstructors, [47](#)
 - vardolyginimas, [47](#)
 - vidurkiolyginimas, [47](#)
 - WriteNormalStudents, [48](#)
 - WriteWeirdStudents, [48](#)
- Studentas.h
 - medianoslyginimas, [49](#)
 - partitionStudents1, [50](#)
 - partitionStudents2, [50](#)
 - partitionStudents3, [50](#)
 - pavardeslyginimas, [51](#)
 - PrintStudents, [51](#)
 - readAndProcessData, [51](#)
 - sortStudents, [52](#)
 - testConstructors, [52](#)
 - vardolyginimas, [52](#)
 - vidurkiolyginimas, [52](#)
 - WriteNormalStudents, [54](#)
 - WriteWeirdStudents, [54](#)
- sumVector
 - Vektorius.cpp, [57](#)
 - Vektorius.h, [60](#)
- swap
 - Vector< T, Allocator >, [29](#)
- testConstructors
 - Studentas.cpp, [47](#)
 - Studentas.h, [52](#)
- value_type
 - Vector< T, Allocator >, [17](#)
- vardolyginimas
 - Studentas.cpp, [47](#)
 - Studentas.h, [52](#)
- Vector
 - Vector< T, Allocator >, [17](#), [18](#)
- Vector< T, Allocator >, [12](#)
 - ~Vector, [18](#)
 - allocator_type, [15](#)
 - at, [18](#), [19](#)
 - back, [19](#)
 - begin, [20](#)
 - capacity, [20](#)
 - cbegin, [20](#)
 - cend, [21](#)
 - clear, [21](#)
 - const_iterator, [15](#)
 - const_pointer, [15](#)
 - const_reference, [16](#)
 - const_reverse_iterator, [16](#)
 - crbegin, [21](#)
 - crend, [21](#)
 - data, [21](#), [22](#)
 - difference_type, [16](#)
 - emplace_back, [22](#)
 - empty, [22](#)
 - end, [23](#)
 - erase, [23](#)
 - front, [24](#)
 - insert, [24](#), [25](#)
 - iterator, [16](#)
 - max_size, [25](#)
 - operator=, [26](#)
 - operator[], [26](#), [27](#)

- pointer, [16](#)
- pop_back, [27](#)
- push_back, [27](#)
- rbegin, [28](#)
- reference, [16](#)
- rend, [28](#)
- reserve, [28](#)
- resize, [29](#)
- reverse_iterator, [16](#)
- shrink_to_fit, [29](#)
- size, [29](#)
- size_type, [17](#)
- swap, [29](#)
- value_type, [17](#)
- Vector, [17](#), [18](#)
- VectorExample
 - Vektorius.cpp, [57](#)
 - Vektorius.h, [61](#)
- VectorUzpildymas
 - Vektorius.cpp, [57](#)
 - Vektorius.h, [61](#)
- Vektorius.cpp
 - findMaxElement, [56](#)
 - isSorted, [56](#)
 - mergeSortedVectors, [56](#)
 - removeDuplicates, [57](#)
 - sumVector, [57](#)
 - VectorExample, [57](#)
 - VectorUzpildymas, [57](#)
- Vektorius.h
 - findMaxElement, [58](#)
 - isSorted, [59](#)
 - mergeSortedVectors, [59](#)
 - removeDuplicates, [60](#)
 - sumVector, [60](#)
 - VectorExample, [61](#)
 - VectorUzpildymas, [61](#)
- vidurkiolyginimas
 - Studentas.cpp, [47](#)
 - Studentas.h, [52](#)
- vidurkis_
 - Studentas, [12](#)
- WriteNormalStudents
 - Studentas.cpp, [48](#)
 - Studentas.h, [54](#)
- WriteWeirdStudents
 - Studentas.cpp, [48](#)
 - Studentas.h, [54](#)
- Zmogus, [30](#)
 - ~Zmogus, [31](#)
 - getEgzaminoRez, [31](#)
 - getNamudarbuRez, [31](#)
 - getPavarde, [31](#)
 - getVardas, [31](#)