Balu-skaiciuokle

v3.0

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 mVector< T > Class Template Reference

`#include <mVector.h>`

Public Types

- using value_type = T
- using allocator_type = std::allocator<T>
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using reference = value_type&
- using const_reference = const value_type&
- using pointer = T*
- using const_pointer = const T*
- 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

- mVector ()
- ~mVector ()
- mVector (const mVector &other)
- mVector & operator= (const mVector &other)
- mVector (mVector &&other) noexcept
- mVector & operator= (mVector &&other) noexcept
- mVector (std::initializer_list< T > init)
- mVector (size_t size, const T &value)
- allocator_type get_allocator () const noexcept
- iterator begin ()
- const_iterator begin () const
- const_iterator cbegin () const
- iterator end ()
- const_iterator end () const

- const_iterator cend () const
- reverse_iterator rbegin ()
- const_reverse_iterator rbegin () const
- const_reverse_iterator crbegin () const
- reverse_iterator rend ()
- const_reverse_iterator rend () const
- const_reverse_iterator crend () const
- size_t size () const
- size_t capacity () const
- bool empty () const
- size_t max_size () const
- void resize (size_t new_size)
- void reserve (size_t new_capacity)
- void shrink_to_fit ()
- T & operator[ ] (size_t index)
- const T & operator[ ] (size_t index) const
- T & at (size_t index)
- const T & at (size_t index) const
- T & front ()
- const T & front () const
- T & back ()
- const T & back () const
- T * data () noexcept
- const T * data () const noexcept
- void pop_back ()
- void push_back (const T &value)
- void push_back (T &&value)
- template<typename T1 , typename T2 >
  void emplace_back (T1 &&arg1, T2 &&arg2)
- void insert (size_t index, const T &value)
- void erase (size_t index)
- void swap (mVector &other)
- void clear ()
- void assign (size_t count, const T &value)
- bool operator== (const mVector &other) const
- bool operator!= (const mVector &other) const

4.1.1 Member Typedef Documentation

4.1.1.1 allocator_type

```
template<class T >
using mVector< T >::allocator_type = std::allocator<T>
```

4.1.1.2 const_iterator

```
template<class T >
using mVector< T >::const_iterator = const T*
```

4.1.1.3 const_pointer

```
template<class T >
using mVector< T >::const_pointer = const T*
```

4.1.1.4 const_reference

```
template<class T >
using mVector< T >::const_reference = const value_type&
```

4.1.1.5 const_reverse_iterator

```
template<class T >
using mVector< T >::const_reverse_iterator = std::reverse_iterator<const_iterator>
```

4.1.1.6 difference_type

```
template<class T >
using mVector< T >::difference_type = std::ptrdiff_t
```

4.1.1.7 iterator

```
template<class T >
using mVector< T >::iterator = T*
```

4.1.1.8 pointer

```
template<class T >
using mVector< T >::pointer = T*
```

4.1.1.9 reference

```
template<class T >
using mVector< T >::reference = value_type&
```

4.1.1.10 reverse_iterator

```
template<class T >
using mVector< T >::reverse_iterator = std::reverse_iterator<iterator>
```

4.1.1.11 size_type

```
template<class T >
using mVector< T >::size_type = std::size_t
```

4.1.1.12 value_type

```
template<class T >
using mVector< T >::value_type = T
```

4.1.2 Constructor & Destructor Documentation

4.1.2.1 mVector() [1/5]

```
template<class T >
mVector< T >::mVector ( )  [inline]
```

4.1.2.2 ~mVector()

```
template<class T >
mVector< T >::~mVector ( )  [inline]
```

4.1.2.3 mVector() [2/5]

```
template<class T >
mVector< T >::mVector (
            const mVector< T > & other )  [inline]
```

4.1.2.4 mVector() [3/5]

```
template<class T >
mVector< T >::mVector (
            mVector< T > && other )  [inline], [noexcept]
```

4.1.2.5 mVector() [4/5]

```
template<class T >
mVector< T >::mVector (
            std::initializer_list< T > init )  [inline]
```

4.1.2.6 mVector() [5/5]

```
template<class T >
mVector< T >::mVector (
            size_t size,
            const T & value )  [inline]
```

4.1.3 Member Function Documentation

4.1.3.1 assign()

```
template<class T >
void mVector< T >::assign (
            size_t count,
            const T & value )  [inline]
```

4.1.3.2 at() [1/2]

```
template<class T >
T & mVector< T >::at (
            size_t index )  [inline]
```

4.1.3.3 at() [2/2]

```
template<class T >
const T & mVector< T >::at (
            size_t index ) const  [inline]
```

4.1.3.4 back() [1/2]

```
template<class T >
T & mVector< T >::back ( )  [inline]
```

4.1.3.5 back() [2/2]

```
template<class T >
const T & mVector< T >::back ( ) const  [inline]
```

4.1.3.6 begin() [1/2]

```
template<class T >
iterator mVector< T >::begin ( )  [inline]
```

4.1.3.7 begin() [2/2]

```
template<class T >
const_iterator mVector< T >::begin ( ) const  [inline]
```

4.1.3.8 capacity()

```
template<class T >
size_t mVector< T >::capacity ( ) const  [inline]
```

4.1.3.9 cbegin()

```
template<class T >
const_iterator mVector< T >::cbegin ( ) const  [inline]
```

4.1.3.10 cend()

```
template<class T >
const_iterator mVector< T >::cend ( ) const  [inline]
```

4.1.3.11 clear()

```
template<class T >
void mVector< T >::clear ( )  [inline]
```

4.1.3.12 crbegin()

```
template<class T >
const_reverse_iterator mVector< T >::crbegin ( ) const  [inline]
```

4.1.3.13 crend()

```
template<class T >
const_reverse_iterator mVector< T >::crend ( ) const  [inline]
```

4.1.3.14 data() [1/2]

```
template<class T >
const T * mVector< T >::data ( ) const  [inline], [noexcept]
```

4.1.3.15 data() [2/2]

```
template<class T >
T * mVector< T >::data ( )  [inline], [noexcept]
```

4.1.3.16 emplace_back()

```
template<class T >
template<typename T1 , typename T2 >
void mVector< T >::emplace_back (
            T1 && arg1,
            T2 && arg2 )  [inline]
```

4.1.3.17 empty()

```
template<class T >
bool mVector< T >::empty ( ) const  [inline]
```

4.1.3.18 end() [1/2]

```
template<class T >
iterator mVector< T >::end ( )  [inline]
```

4.1.3.19 end() [2/2]

```
template<class T >
const_iterator mVector< T >::end ( ) const  [inline]
```

4.1.3.20 erase()

```
template<class T >
void mVector< T >::erase (
            size_t index )  [inline]
```

4.1.3.21 front() [1/2]

```
template<class T >
T & mVector< T >::front ( )  [inline]
```

4.1.3.22 front() [2/2]

```
template<class T >
const T & mVector< T >::front ( ) const  [inline]
```

4.1.3.23 get_allocator()

```
template<class T >
allocator_type mVector< T >::get_allocator ( ) const  [inline], [noexcept]
```

4.1.3.24 insert()

```
template<class T >
void mVector< T >::insert (
            size_t index,
            const T & value )  [inline]
```

4.1.3.25 max_size()

```
template<class T >
size_t mVector< T >::max_size ( ) const  [inline]
```

4.1.3.26 operator"!=()

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

4.1.3.27 operator=() [1/2]

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

4.1.3.28 operator=() [2/2]

```
template<class T >
mVector & mVector< T >::operator= (
            mVector< T > && other )  [inline], [noexcept]
```

4.1.3.29 operator==()

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

4.1.3.30 operator[]() [1/2]

```
template<class T >
T & mVector< T >::operator[] (
            size_t index )  [inline]
```

4.1.3.31 operator[]() [2/2]

```
template<class T >
const T & mVector< T >::operator[] (
            size_t index ) const  [inline]
```

4.1.3.32 pop_back()

```
template<class T >
void mVector< T >::pop_back ( )  [inline]
```

4.1.3.33 push_back() [1/2]

```
template<class T >
void mVector< T >::push_back (
            const T & value )  [inline]
```

4.1.3.34 push_back() [2/2]

```
template<class T >
void mVector< T >::push_back (
            T && value )  [inline]
```

4.1.3.35 rbegin() [1/2]

```
template<class T >
reverse_iterator mVector< T >::rbegin ( )  [inline]
```

4.1.3.36 rbegin() [2/2]

```
template<class T >
const_reverse_iterator mVector< T >::rbegin ( ) const  [inline]
```

4.1.3.37 rend() [1/2]

```
template<class T >
reverse_iterator mVector< T >::rend ( )  [inline]
```

4.1.3.38 rend() [2/2]

```
template<class T >
const_reverse_iterator mVector< T >::rend ( ) const  [inline]
```

4.1.3.39 reserve()

```
template<class T >
void mVector< T >::reserve (
            size_t new_capacity )  [inline]
```

4.1.3.40 resize()

```
template<class T >
void mVector< T >::resize (
            size_t new_size )  [inline]
```

4.1.3.41 shrink_to_fit()

```
template<class T >
void mVector< T >::shrink_to_fit ( )  [inline]
```

4.1.3.42 size()

```
template<class T >
size_t mVector< T >::size ( ) const  [inline]
```

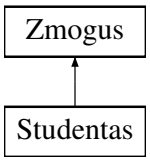
4.1.3.43 swap()

```
template<class T >
void mVector< T >::swap (
            mVector< T > & other )  [inline]
```

4.2 Studentas Class Reference

`#include <student.h>`

Inheritance diagram for Studentas:



Public Member Functions

- Studentas ()
- Studentas (const std::string &vardas_, const std::string &pavarde_, double egzaminas_, const mVector< int > &nd_)
- Studentas (const Studentas &other)
- Studentas & operator= (const Studentas &other)
- Studentas (Studentas &&other) noexcept
- Studentas & operator= (Studentas &&other) noexcept
- ~Studentas ()
- std::string getVardas () const
- std::string getPavarde () const
- const mVector< int > & getND () const
- double getEgzaminas ()
- void setVardas (const std::string &vardas)
- void setPavarde (const std::string &pavarde)
- void setEgzaminas (double egzaminas)
- void setNd (const mVector< int > &nd)
- void setUseMedian (bool useMedian)
- bool getUseMedian () const
- double calculateGalutinis () const

Public Member Functions inherited from Zmogus

- virtual ~Zmogus ()=default

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Studentas() [1/4]

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

4.2.1.2 Studentas() [2/4]

```
Studentas::Studentas (
            const std::string & vardas_,
            const std::string & pavarde_,
            double egzaminas_,
            const mVector< int > & nd_ )
```

4.2.1.3 Studentas() [3/4]

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

4.2.1.4 Studentas() [4/4]

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

4.2.1.5 ~Studentas()

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

4.2.2 Member Function Documentation

4.2.2.1 calculateGalutinis()

```
double Studentas::calculateGalutinis ( ) const
```

4.2.2.2 getEgzaminas()

```
double Studentas::getEgzaminas ( )  [inline], [virtual]
```

Implements Zmogus.

4.2.2.3 getND()

```
const mVector< int > & Studentas::getND ( ) const  [inline], [virtual]
```

Implements Zmogus.

4.2.2.4 getPavarde()

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

Implements Zmogus.

4.2.2.5 getUseMedian()

```
bool Studentas::getUseMedian ( ) const  [inline]
```

4.2.2.6 getVardas()

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

Implements Zmogus.

4.2.2.7 operator=() [1/2]

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

4.2.2.8 operator=() [2/2]

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

4.2.2.9 setEgzaminas()

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

4.2.2.10 setNd()

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

4.2.2.11 setPavarde()

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

4.2.2.12 setUseMedian()

```
void Studentas::setUseMedian (
            bool useMedian )  [inline]
```

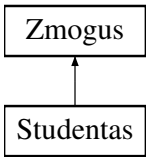
4.2.2.13 setVardas()

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

4.3 Zmogus Class Reference

```
#include <student.h>
```

Inheritance diagram for Zmogus:



Public Member Functions

- virtual ~Zmogus ()=default
- virtual std::string getVardas () const =0
- virtual std::string getPavarde () const =0
- virtual const mVector< int > & getND () const =0
- virtual double getEgzaminas ()=0

4.3.1 Constructor & Destructor Documentation

4.3.1.1 ~Zmogus()

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

4.3.2 Member Function Documentation

4.3.2.1 getEgzaminas()

```
virtual double Zmogus::getEgzaminas ( )  [pure virtual]
```

Implemented in Studentas.

4.3.2.2 getND()

```
virtual const mVector< int > & Zmogus::getND ( ) const  [pure virtual]
```

Implemented in Studentas.

4.3.2.3 getPavarde()

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

Implemented in Studentas.

4.3.2.4 getVardas()

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

Implemented in Studentas.

Chapter 5

File Documentation

5.1 main.cpp File Reference

```
#include "menu.h"
#include "student.h"
```

Functions

- void testConstructors ()
- int main ()

5.1.1 Function Documentation

5.1.1.1 main()

```
int main ( )
```

5.1.1.2 testConstructors()

```
void testConstructors ( )
```

5.2 menu.cpp File Reference

```
#include "menu.h"
#include "student.h"
```

Functions

- void Menu_execute ()

5.2.1 Function Documentation

5.2.1.1 Menu_execute()

```
void Menu_execute ( )
```

5.3 menu.h File Reference

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

Functions

- void Menu_execute ()

5.3.1 Function Documentation

5.3.1.1 Menu_execute()

```
void Menu_execute ( )
```

5.4 menu.h

Go to the documentation of this file.

```
00001 #ifndef MENU_H
00002 #define MENU_H
00003
00004 #include <string>
00005 #include "student.h"
00006
00007 void Menu_execute();
00008
00009 #endif // MAIN_LOGIC_H
00010 // Path: v0.3/main_logic.cpp
```

5.5 mVector.h File Reference

```
#include <cstddef>
#include <iterator>
#include <memory>
#include <stdexcept>
#include <algorithm>
#include <utility>
```

Classes

- class mVector< T >

5.6 mVector.h

Go to the documentation of this file.

```
00001 #include <cstddef>
00002 #include <iterator>
00003 #include <memory>
00004 #include <stdexcept>
00005 #include <algorithm>
00006 #include <utility>
00007
00008 template <class T>
00009 class mVector {
00010 private:
00011     T* data_;
00012     size_t size_;
00013     size_t capacity_;
00014
00015 public:
00016     // Member types
00017     using value_type = T;
00018     using allocator_type = std::allocator<T>;
00019     using size_type = std::size_t;
00020     using difference_type = std::ptrdiff_t;
00021     using reference = value_type&;
00022     using const_reference = const value_type&;
00023     using pointer = T*;
00024     using const_pointer = const T*;
00025     using iterator = T*;
00026     using const_iterator = const T*;
00027     using reverse_iterator = std::reverse_iterator<iterator>;
00028     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00029
00030     // Default constructor
00031     mVector() : data_(nullptr), size_(0), capacity_(0) {}
00032
00033     // Destructor
00034     ~mVector() {
00035         delete[] data_;
00036     }
00037
00038     // Copy constructor
00039     mVector(const mVector& other) : data_(nullptr), size_(0), capacity_(0) {
00040         reserve(other.size_);
00041         size_ = other.size_;
00042         std::copy(other.data_, other.data_ + other.size_, data_);
00043     }
00044
00045     // Copy assignment operator
00046     mVector& operator=(const mVector& other) {
00047         if (this != &other) {
00048             T* new_data = new T[other.capacity_];
00049             std::copy(other.data_, other.data_ + other.size_, new_data);
00050             delete[] data_;
00051             data_ = new_data;
00052             size_ = other.size_;
00053             capacity_ = other.capacity_;
00054         }
00055         return *this;
00056     }
00057
00058     // Move constructor
00059     mVector(mVector&& other) noexcept : data_(other.data_), size_(other.size_),
    capacity_(other.capacity_) {
00060         other.data_ = nullptr;
00061         other.size_ = 0;
00062         other.capacity_ = 0;
00063     }
00064
00065     // Move assignment operator
00066     mVector& operator=(mVector&& other) noexcept {
00067         if (this != &other) {
00068             delete[] data_;
00069             data_ = other.data_;
00070             size_ = other.size_;
00071             capacity_ = other.capacity_;
00072             other.data_ = nullptr;
00073             other.size_ = 0;
00074             other.capacity_ = 0;
00075         }
00076         return *this;
00077     }
00078
00079     // Constructor that takes an initializer list
00080     mVector(std::initializer_list<T> init) : size_(init.size()), capacity_(init.size()), data_(new
    T[init.size()]) {
```

```
00081         std::copy(init.begin(), init.end(), data_);
00082     }
00083
00084     // Constructor that takes a size and an initial value
00085     mVector(size_t size, const T& value) : size_(size), capacity_(size), data_(new T[size]) {
00086         std::fill(data_, data_ + size, value);
00087     }
00088
00089     // Return allocator
00090     allocator_type get_allocator() const noexcept {
00091         return allocator_type();
00092     }
00093
00094     // Iterators
00095     iterator begin() { return data_; }
00096     const_iterator begin() const { return data_; }
00097     const_iterator cbegin() const { return data_; }
00098     iterator end() { return data_ + size_; }
00099     const_iterator end() const { return data_ + size_; }
00100     const_iterator cend() const { return data_ + size_; }
00101     reverse_iterator rbegin() { return reverse_iterator(end()); }
00102     const_reverse_iterator rbegin() const { return const_reverse_iterator(end()); }
00103     const_reverse_iterator crbegin() const { return const_reverse_iterator(end()); }
00104     reverse_iterator rend() { return reverse_iterator(begin()); }
00105     const_reverse_iterator rend() const { return const_reverse_iterator(begin()); }
00106     const_reverse_iterator crend() const { return const_reverse_iterator(begin()); }
00107
00108     // Capacity
00109     size_t size() const { return size_; }
00110     size_t capacity() const { return capacity_; }
00111     bool empty() const { return size_ == 0; }
00112     size_t max_size() const { return std::allocator_traits<allocator_type>::max_size(get_allocator());
      }
00113
00114     // Resize the vector
00115     void resize(size_t new_size) {
00116         if (new_size > capacity_) {
00117             reserve(new_size);
00118         }
00119         if (new_size > size_) {
00120             std::fill(data_ + size_, data_ + new_size, T());
00121         }
00122         size_ = new_size;
00123     }
00124
00125     // Reserve capacity
00126     void reserve(size_t new_capacity) {
00127         if (new_capacity > capacity_) {
00128             T* new_data = new T[new_capacity];
00129             std::copy(data_, data_ + size_, new_data);
00130             delete[] data_;
00131             data_ = new_data;
00132             capacity_ = new_capacity;
00133         }
00134     }
00135
00136     void shrink_to_fit() {
00137         if (capacity_ > size_) {
00138             T* new_data = new T[size_];
00139             std::move(data_, data_ + size_, new_data);
00140             delete[] data_;
00141             data_ = new_data;
00142             capacity_ = size_;
00143         }
00144     }
00145
00146     // Element access
00147     T& operator[](size_t index) { return data_[index]; }
00148     const T& operator[](size_t index) const { return data_[index]; }
00149
00150     T& at(size_t index) {
00151         if (index >= size_) {
00152             throw std::out_of_range("mVector::at - index out of range");
00153         }
00154         return data_[index];
00155     }
00156
00157     const T& at(size_t index) const {
00158         if (index >= size_) {
00159             throw std::out_of_range("mVector::at - index out of range");
00160         }
00161         return data_[index];
00162     }
00163
00164     T& front() {
00165         if (size_ == 0) {
00166             throw std::out_of_range("mVector::front - vector is empty");
```

```
00167         }
00168         return data_[0];
00169     }
00170 
00171     const T& front() const {
00172         if (size_ == 0) {
00173             throw std::out_of_range("mVector::front - vector is empty");
00174         }
00175         return data_[0];
00176     }
00177 
00178     T& back() {
00179         if (size_ == 0) {
00180             throw std::out_of_range("mVector::back - vector is empty");
00181         }
00182         return data_[size_ - 1];
00183     }
00184 
00185     const T& back() const {
00186         if (size_ == 0) {
00187             throw std::out_of_range("mVector::back - vector is empty");
00188         }
00189         return data_[size_ - 1];
00190     }
00191 
00192     T* data() noexcept { return data_; }
00193     const T* data() const noexcept { return data_; }
00194 
00195     // Modifiers
00196     void pop_back() {
00197         if (size_ == 0) {
00198             throw std::out_of_range("mVector is empty");
00199         }
00200         --size_;
00201     }
00202 
00203     void push_back(const T& value) {
00204         if (size_ >= capacity_) {
00205             // Double the capacity
00206             capacity_ = (capacity_ == 0) ? 1 : capacity_ * 2;
00207 
00208             // Allocate new memory and copy old data
00209             T* new_data = new T[capacity_];
00210             std::copy(data_, data_ + size_, new_data);
00211 
00212             // Delete old data and update pointer
00213             delete[] data_;
00214             data_ = new_data;
00215         }
00216         // Add new value
00217         data_[size_++] = value;
00218     }
00219 
00220     void push_back(T&& value) {
00221         if (size_ == capacity_) {
00222             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00223         }
00224         data_[size_++] = std::move(value);
00225     }
00226 
00227     template <typename T1, typename T2>
00228     void emplace_back(T1&& arg1, T2&& arg2) {
00229         if (size_ == capacity_) {
00230             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00231         }
00232         data_[size_++] = T(std::forward<T1>(arg1), std::forward<T2>(arg2));
00233     }
00234 
00235     void insert(size_t index, const T& value) {
00236         if (index > size_) {
00237             throw std::out_of_range("mVector::insert - index out of range");
00238         }
00239         if (size_ == capacity_) {
00240             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00241         }
00242         std::move_backward(data_ + index, data_ + size_, data_ + size_ + 1);
00243         data_[index] = value;
00244         ++size_;
00245     }
00246 
00247     void erase(size_t index) {
00248         if (index >= size_) {
00249             throw std::out_of_range("mVector::erase - index out of range");
00250         }
00251         std::move(data_ + index + 1, data_ + size_, data_ + index);
00252         --size_;
00253     }
```

```
00254
00255     void swap(mVector& other) {
00256         std::swap(data_, other.data_);
00257         std::swap(size_, other.size_);
00258         std::swap(capacity_, other.capacity_);
00259     }
00260
00261     void clear() {
00262         delete[] data_;
00263         data_ = nullptr;
00264         size_ = 0;
00265         capacity_ = 0;
00266     }
00267
00268     void assign(size_t count, const T& value) {
00269         clear();
00270         reserve(count);
00271         for (size_t i = 0; i < count; ++i) {
00272             push_back(value);
00273         }
00274     }
00275
00276     // Comparison operators
00277     bool operator==(const mVector& other) const {
00278         if (size_ != other.size_) {
00279             return false;
00280         }
00281         for (size_t i = 0; i < size_; ++i) {
00282             if (data_[i] != other.data_[i]) {
00283                 return false;
00284             }
00285         }
00286         return true;
00287     }
00288
00289     bool operator!=(const mVector& other) const {
00290         return !(*this == other);
00291     }
00292 };
```

5.7 student.cpp File Reference

```
#include "student.h"
```

Functions

- std::ostream & operator<< (std::ostream &os, const Studentas &student)
- std::istream & operator>> (std::istream &is, Studentas &student)
- bool isNumber (const string &str)
- double useMediana (const mVector< int > &grades)
- void Ivedimas (mVector< Studentas > &stud, bool randomNames, bool randomGrades, bool studentCount)
- void Spausdinimas (const mVector< Studentas > &stud, bool Mediana)
- void sortStudents (mVector< Studentas > &students, const string &sortBy, bool Mediana)
- void Pasirinkimai (mVector< Studentas > &students)
- void Generacija (int Pas)
- void testVectors ()

Variables

- const int NUM_NAMES = 10
- mVector< string > vardai = { "Jonas", "Petras", "Algis", "Marius", "Gintaras", "Tomas", "Lukas", "Simas", "Gabrielius", "Olegas" }
- mVector< string > pavardes = { "Kelmutis", "Kelmutaite", "Dangavicius", "Pieliauskas", "Lukavicius", "Simonavicius", "Skaudavicius", "Juzenas", "Darbavicius", "Stankevicius" }

5.7.1 Function Documentation

5.7.1.1 Generacija()

```
void Generacija (
            int Pas )
```

5.7.1.2 isNumber()

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

5.7.1.3 Ivedimas()

```
void Ivedimas (
            mVector< Studentas > & stud,
            bool randomNames,
            bool randomGrades,
            bool studentCount )
```

5.7.1.4 operator<<()

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

5.7.1.5 operator>>()

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

5.7.1.6 Pasirinkimai()

```
void Pasirinkimai (
            mVector< Studentas > & students )
```

5.7.1.7 sortStudents()

```
void sortStudents (
            mVector< Studentas > & students,
            const string & sortBy,
            bool Mediana )
```

5.7.1.8 Spausdinimas()

```
void Spausdinimas (
            const mVector< Studentas > & stud,
            bool Mediana )
```

5.7.1.9 testVectors()

```
void testVectors ( )
```

5.7.1.10 useMediana()

```
double useMediana (
            const mVector< int > & grades )
```

5.7.2 Variable Documentation

5.7.2.1 NUM_NAMES

```
const int NUM_NAMES = 10
```

5.7.2.2 pavardes

```
mVector<string> pavardes = { "Kelmutis", "Kelmutaite", "Dangavicius", "Pieliauskas", "Lukavicius",
"Simonavicius", "Skaudavicius", "Juzenas", "Darbavicius", "Stankevicius" }
```

5.7.2.3 vardai

```
mVector<string> vardai = { "Jonas", "Petras", "Algis", "Marius", "Gintaras", "Tomas", "Lukas",
"Simas", "Gabrielius", "Olegas" }
```

5.8 student.h File Reference

```
#include <iostream>
#include <string>
#include <iomanip>
#include <algorithm>
#include <vector>
#include <numeric>
#include <ctime>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <cmath>
#include <chrono>
#include <random>
#include <iterator>
#include <functional>
#include <stdexcept>
#include "mVector.h"
```

Classes

- class Zmogus
- class Studentas

Functions

- bool compare (const Studentas &, const Studentas &)
- bool comparePagalPavarde (const Studentas &, const Studentas &)
- bool comparePagalEgza (const Studentas &, const Studentas &)
- void Ivedimas (mVector< Studentas > &stud, bool randomNames=false, bool randomGrades=false, bool studentCount=false)
- void Pasirinkimai (mVector< Studentas > &students)
- void Spausdinimas (const mVector< Studentas > &students, bool Mediana)
- bool isNumber (const string &str)
- double useMediana (const mVector< int > &grades)
- void Generacija (int Pas)
- void testVectors ()

5.8.1 Function Documentation

5.8.1.1 compare()

```
bool compare (
            const Studentas & ,
            const Studentas &  )
```

5.8.1.2 comparePagalEgza()

```
bool comparePagalEgza (
            const Studentas & ,
            const Studentas &  )
```

5.8.1.3 comparePagalPavarde()

```
bool comparePagalPavarde (
            const Studentas & ,
            const Studentas &  )
```

5.8.1.4 Generacija()

```
void Generacija (
            int Pas )
```

5.8.1.5 isNumber()

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

5.8.1.6 Ivedimas()

```
void Ivedimas (
            mVector< Studentas > & stud,
            bool randomNames = false,
            bool randomGrades = false,
            bool studentCount = false )
```

5.8.1.7 Pasirinkimai()

```
void Pasirinkimai (
            mVector< Studentas > & students )
```

5.8.1.8 Spausdinimas()

```
void Spausdinimas (
            const mVector< Studentas > & students,
            bool Mediana )
```

5.8.1.9 testVectors()

```
void testVectors ( )
```

5.8.1.10 useMediana()

```
double useMediana (
            const mVector< int > & grades )
```

5.9 student.h

Go to the documentation of this file.

```
00001 #ifndef STUDENT_H
00002 #define STUDENT_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <iomanip>
00007 #include <algorithm>
00008 #include <vector>
00009 #include <numeric>
00010 #include <ctime>
00011 #include <fstream>
00012 #include <sstream>
00013 #include <cstdlib>
00014 #include <cctype>
00015 #include <cmath>
00016 #include <chrono>
00017 #include <random>
00018 #include <iterator>
00019 #include <functional>
00020 #include <stdexcept>
00021 #include "mVector.h"
00022
00023
00024 using namespace std;
00025 using namespace std::chrono;
00026
```

```
00027 class Zmogus{
00028     public:
00029     virtual ~Zmogus() = default;
00030 
00031     //geteriai
00032     virtual std::string getVardas() const = 0;
00033     virtual std::string getPavarde() const = 0;
00034     virtual const mVector<int>& getND() const = 0;
00035     virtual double getEgzaminas() = 0;
00036 };
00037 
00038 class Studentas: public Zmogus{
00039 private:
00040     std::string vardas_;
00041     std::string pavarde_;
00042     double egzaminas_;
00043     mVector<int> nd_;
00044     bool useMedian;
00045 public:
00046         // Default constructor
00047         Studentas();
00048 
00049         // Constructor with parameters
00050         Studentas(const std::string& vardas_, const std::string& pavarde_, double egzaminas_, const
      mVector<int>& nd_);
00051 
00052         // Copy constructor
00053         Studentas(const Studentas& other);
00054 
00055         // Copy assignment operator
00056         Studentas& operator=(const Studentas& other);
00057 
00058         // Move constructor
00059         Studentas(Studentas&& other) noexcept;
00060 
00061         // Move assignment operator
00062         Studentas& operator=(Studentas&& other) noexcept;
00063 
00064         // Destructor
00065         ~Studentas();
00066 
00067         // Getters
00068         inline std::string getVardas() const { return vardas_; }
00069         inline std::string getPavarde() const { return pavarde_; }
00070         const mVector<int>& getND() const { return nd_; }
00071         double getEgzaminas() { return egzaminas_; }
00072 
00073         // Setters
00074         void setVardas(const std::string& vardas) { vardas_ = vardas; }
00075         void setPavarde(const std::string& pavarde) { pavarde_ = pavarde; }
00076         void setEgzaminas(double egzaminas) { egzaminas_ = egzaminas; }
00077         void setNd(const mVector<int>& nd) { nd_ = nd; }
00078 
00079         // Other functions
00080         void setUseMedian(bool useMedian) {this -> useMedian = useMedian; }
00081         bool getUseMedian() const { return useMedian; }
00082         // Calculate the final grade
00083         double calculateGalutinis() const; // Add the missing implementation
00084 };
00085 
00086 // Comparison functions for sorting
00087 bool compare(const Studentas&, const Studentas&);
00088 bool comparePagalPavarde(const Studentas&, const Studentas&);
00089 bool comparePagalEgza(const Studentas&, const Studentas&);
00090 
00091 // Input function
00092 void Ivedimas(mVector<Studentas>& stud, bool randomNames = false, bool randomGrades = false, bool
      studentCount = false);
00093 
00094 // Menu function
00095 void Pasirinkimai(mVector<Studentas>& students);
00096 
00097 // Output function
00098 void Spausdinimas(const mVector<Studentas>& students, bool Mediana);
00099 
00100 // Check if a string is a number
00101 bool isNumber(const string& str);
00102 
00103 // Calculate the final grade using median
00104 double useMediana(const mVector<int>& grades);
00105 
00106 // Generate random data
00107 void Generacija(int Pas);
00108 
00109 void testVectors();
00110 
00111 #endif // STUDENT_H
```

```
00112 // Path: v0.3/v0.3.cpp
```

Index