My Vector documentation

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Vector< T > Class Template Reference

A simple vector class.

`#include <myVector.h>`

Public Member Functions

- Vector ()
  Default constructor.
- Vector (std::initializer_list< T > list)
  Constructor with initializer list.
- ~Vector ()
  Destructor.
- Vector (const Vector &other)
  Copy constructor.
- Vector (Vector &&other) noexcept
  Move constructor.
- Vector & operator= (const Vector &other)
  Copy assignment operator.
- Vector & operator= (Vector &&other) noexcept
  Move assignment operator.
- void push_back (const T &value)
  Adds an element to the end of the vector.
- void pop_back ()
  Removes the last element of the vector.
- size_t size () const
  Returns the number of elements in the vector.
- size_t capacity () const
  Returns the capacity of the vector.
- void shrink_to_fit ()
  Reduces the capacity to fit the size.
- bool empty () const
  Checks if the vector is empty.

- T * begin ()

  Returns an iterator to the beginning.
- const T * **begin** () const
- T * end ()

  Returns an iterator to the end.
- const T * **end** () const
- void erase (size_t pos)

  Erases an element at a specific position.
- T * erase (T *pos)

  Erases an element at a specific position.
- T * erase (T *first, T *last)

  Erases elements in a range.
- void clear ()

  Clears all elements from the vector.
- T * insert (T *pos, const T &value)

  Inserts an element at a specific position.
- T * insert (T *pos, T &&value)

  Inserts an element at a specific position.
- void reserve (size_t new_capacity)

  Reserves memory for the vector.
- void resize (size_t new_size, const T &value=T())

  Resizes the vector.
- T & at (size_t index)

  Accesses an element at a specific index with bounds checking.
- const T & **at** (size_t index) const
- T & front ()

  Accesses the first element.
- const T & **front** () const
- T & back ()

  Accesses the last element.
- const T & **back** () const
- T * data2 ()

  Returns a pointer to the underlying array.
- const T * **data2** () const
- void assign (T *first, T *last)

  Assigns new content to the vector.
- void assign (size_t count, const T &value)

  Assigns new content to the vector.
- T & operator[ ] (size_t index)

  Accesses an element at a specific index.
- const T & **operator[ ]** (size_t index) const
- template<class... Args>
  void emplace_back (Args &&... args)

  Constructs and adds an element to the end of the vector.
- template<typename... Args>
  void emplace (size_t index, Args &&... args)

  Constructs and inserts an element at a specific position.
- void swap (Vector &other)

  Swaps the contents of this vector with another.

3.1.1 Detailed Description

template<typename T>
class Vector< T >

A simple vector class.

Template Parameters

T	Type of the elements stored in the vector.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Vector() [1/3]

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

Constructor with initializer list.

Parameters

list	Initializer list to initialize the vector.

3.1.2.2 Vector() [2/3]

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

Copy constructor.

Parameters

other	Vector to copy from.

3.1.2.3 Vector() [3/3]

```
template<typename T >
```

```
Vector< T >::Vector (
            Vector< T > && other )  [inline], [noexcept]
```

Move constructor.

Parameters

other	Vector to move from.

3.1.3 Member Function Documentation

3.1.3.1 assign() [1/2]

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

Assigns new content to the vector.

Parameters

count	Number of elements to assign.
value	Value to assign to the elements.

3.1.3.2 assign() [2/2]

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

Assigns new content to the vector.

Parameters

first	Iterator to the first element.
last	Iterator to the last element.

3.1.3.3 at()

```
template<typename T >
```

```
T& Vector< T >::at (
            size_t index )  [inline]
```

Accesses an element at a specific index with bounds checking.

Parameters

index	The index of the element.

Returns

Reference to the element at the specified index.

3.1.3.4 back()

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

Accesses the last element.

Returns

Reference to the last element.

3.1.3.5 begin()

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

Returns an iterator to the beginning.

Returns

Iterator to the beginning.

3.1.3.6 capacity()

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

Returns the capacity of the vector.

Returns

Capacity of the vector.

3.1.3.7 data2()

```
template<typename T >
T* Vector< T >::data2 ( )  [inline]
```

Returns a pointer to the underlying array.

Returns

Pointer to the underlying array.

3.1.3.8 emplace()

```
template<typename T >
template<typename...  Args>
void Vector< T >::emplace (
            size_t index,
            Args &&...  args )  [inline]
```

Constructs and inserts an element at a specific position.

Template Parameters

Args	Types of the arguments.

Parameters

index	The position to insert the element at.
args	Arguments to forward to the constructor.

3.1.3.9 emplace_back()

```
template<typename T >
template<class...  Args>
void Vector< T >::emplace_back (
            Args &&...  args )  [inline]
```

Constructs and adds an element to the end of the vector.

Template Parameters

Args	Types of the arguments.

Parameters

args	Arguments to forward to the constructor.

3.1.3.10 empty()

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

Checks if the vector is empty.

Returns

true if the vector is empty, false otherwise.

3.1.3.11 end()

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

Returns an iterator to the end.

Returns

Iterator to the end.

3.1.3.12 erase() [1/3]

```
template<typename T >
void Vector< T >::erase (
            size_t pos )  [inline]
```

Erases an element at a specific position.

Parameters

pos	Position of the element to erase.

3.1.3.13 erase() [2/3]

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

Erases elements in a range.

Parameters

first	Iterator to the first element to erase.
last	Iterator to the last element to erase.

Returns

Iterator to the next element.

3.1.3.14 erase() [3/3]

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

Erases an element at a specific position.

Parameters

pos	Iterator to the position of the element to erase.

Returns

Iterator to the next element.

3.1.3.15 front()

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

Accesses the first element.

Returns

Reference to the first element.

3.1.3.16 insert() [1/2]

```
template<typename T >
T* Vector< T >::insert (
            T * pos,
            const T & value )  [inline]
```

Inserts an element at a specific position.

Parameters

pos	Iterator to the position.
value	The value to insert.

Returns

Iterator to the inserted element.

3.1.3.17 insert() [2/2]

```
template<typename T >
T* Vector< T >::insert (
            T * pos,
            T && value )  [inline]
```

Inserts an element at a specific position.

Parameters

pos	Iterator to the position.
value	The value to insert.

Returns

Iterator to the inserted element.

3.1.3.18 operator=() [1/2]

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

Copy assignment operator.

Parameters

other	Vector to copy from.

Returns

Reference to this vector.

3.1.3.19 operator=() [2/2]

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

Move assignment operator.

Parameters

other	Vector to move from.

Returns

Reference to this vector.

3.1.3.20 operator[]()

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

Accesses an element at a specific index.

Parameters

index	The index of the element.

Returns

Reference to the element at the specified index.

3.1.3.21 push_back()

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

Adds an element to the end of the vector.

Parameters

value	The value to add.

3.1.3.22 reserve()

```
template<typename T >
void Vector< T >::reserve (
            size_t new_capacity )  [inline]
```

Reserves memory for the vector.

Parameters

new_capacity	The new capacity to reserve.

3.1.3.23 resize()

```
template<typename T >
void Vector< T >::resize (
            size_t new_size,
            const T & value = T() )  [inline]
```

Resizes the vector.

Parameters

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

3.1.3.24 size()

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

Returns the number of elements in the vector.

Returns

Number of elements in the vector.

3.1.3.25 swap()

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

Swaps the contents of this vector with another.

Parameters

other	The other vector to swap with.

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

- headers/myVector.h

Chapter 4

File Documentation

4.1 headers/myVector.h File Reference

Header file for the custom vector implementation.

```
#include <iostream>
#include <stdexcept>
#include <iterator>
#include <algorithm>
```

Include dependency graph for myVector.h:

Index