Reference Manual

Generated by Doxygen 1.8.6

Thu Apr 20 2017 18:06:11

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Vector< T > Class Template Reference

`#include <vector.h>`

Public Member Functions

- Vector ()
- Vector (const unsigned) throw (bad_alloc)
- Vector (const unsigned, const T &) throw (bad_alloc)
- Vector (const Vector< T > &) throw (bad_alloc)
- ~Vector ()
- void assign (const unsigned, const T &) throw (out_of_range)
- T & at (const unsigned) const throw (out_of_range)
- unsigned capacity () const
- void clear ()
- bool empty () const
- void insert (const unsigned, const T &) throw (bad_alloc, out_of_range)
- T & operator[] (const unsigned) const throw (out_of_range)
- const Vector< T > & operator= (const Vector< T > &) throw (bad_alloc)
- void push_back (const T &) throw (bad_alloc)
- void resize (const unsigned, const T &=T()) throw (bad_alloc)
- unsigned size () const
- void sort ()
- void remove (const unsigned) throw (out_of_range)

Private Member Functions

- void increaseCapacity () throw (bad_alloc)

Private Attributes

- T * contents
- unsigned sz
- unsigned cap

2.1.1 Detailed Description

template<typename T>class Vector< T >

Implementation of a resizable Vector ADT using a dynamically allocated C-style array.

Author

Mark Maloof (maloof@cs.georgetown.edu)

Version

1.3, 5/7/13

2.1.2 Constructor & Destructor Documentation

2.1.2.1 template<typename T > Vector< T >::Vector ( )

Default constructor.

2.1.2.2 template<typename T > Vector< T >::Vector (const unsigned *cap*) throw bad_alloc)

Constructor for initializing an empty vector with a fixed capacity.

Parameters

cap	the capacity of the vector

Exceptions

bad_alloc	if memory cannot be allocated

2.1.2.3 template<typename T > Vector< T >::Vector (const unsigned *sz*, const T & *object*) throw bad_alloc)

Constructor for initializing a vector to an initial size with its components initialized to a specified object. If there is no such object, then the default is used.

Parameters

sz	the size of the vector
object	the initial object for the components

Exceptions

bad_alloc	if memory cannot be allocated

2.1.2.4 template<typename T > Vector< T >::Vector (const Vector< T > & *vector*) throw bad_alloc)

Copy constructor. Make a deep copy of the specified vector.

Parameters

vector	the vector to be copied

Exceptions

bad_alloc	if memory cannot be allocated

2.1.2.5 template<typename T > Vector< T >::~Vector ( )

Destructor.

2.1.3 Member Function Documentation

2.1.3.1 template<typename T > void Vector< T >::assign (const unsigned *i*, const T & *object*) throw out_of_range)

Assigns the object to the specified position in the vector.

Parameters

i	the position to be assigned.
object	the object to be stored in the vector.

Exceptions

out_of_range	if index parameter is out of bounds.

2.1.3.2 template<typename T > T & Vector< T >::at (const unsigned *i*) const throw out_of_range)

Returns a reference to the object stored at a given position in the vector.

Parameters

i	the object's location.

Returns

a reference to the object.

Exceptions

out_of_range	if index parameter is out of bounds.

2.1.3.3 template<typename T > unsigned Vector< T >::capacity ( ) const

Returns the capacity of the vector, which is the number of elements that the vector can store before increasing the capacity.

Returns

an unsigned integer indicating the vector's capacity.

2.1.3.4 template<typename T > void Vector< T >::clear ( )

Removes the elements of the vector.

2.1.3.5 template<typename T > bool Vector< T >::empty ( ) const

Returns true if the vector is empty; returns false otherwise.

Returns

true if empty; false otherwise.

2.1.3.6 template<typename T > void Vector< T >::increaseCapacity ( ) throw bad_alloc) [private]

Increases the capacity of the vector by doubling its current capacity.

Exceptions

bad_alloc	if memory cannot be allocated.

2.1.3.7 template<typename T > void Vector< T >::insert (const unsigned *i,* const T & *object*) throw bad_alloc, out_of_range)

Inserts the object at the given position. Increases capacity if necessary.

Parameters

i	the position of insertion.
object	the object to be inserted.

Exceptions

bad_alloc	if memory cannot be allocated.
out_of_range	if index parameter is out of bounds.

2.1.3.8 template<typename T > const Vector< T > & Vector< T >::operator= (const Vector< T > & *vector*) throw bad_alloc)

Returns a deep copy of the vector passed in as the parameter.

Parameters

vector	the vector to be copied.

Returns

a copy of the vector.

Exceptions

bad_alloc	if memory cannot be allocated.

2.1.3.9 template<typename T > T & Vector< T >::operator[] (const unsigned *i*) const throw out_of_range)

Returns a reference to the object stored at a given position in the vector.

Parameters

i	the object's location.

Returns

a reference to the object.

Exceptions

out_of_range	if index parameter is out of bounds.

2.1.3.10 template<typename T > void Vector< T >::push_back (const T & *object*) throw bad_alloc)

Adds the object to the end of the vector. Increases capacity if necessary.

Parameters

object	the object to be added to the end of the vector.

Exceptions

bad_alloc	if memory cannot be allocated.

2.1.3.11 template<typename T > void Vector< T >::remove (const unsigned *i*) throw out_of_range)

Removes the object stored in the given position.

Parameters

i	the position of removal.

Exceptions

out_of_range	if index parameter is out of bounds.

2.1.3.12 template<typename T > void Vector< T >::resize (const unsigned *newSize,* const T & *object* = `T()`) throw bad_alloc)

Resizes the vector to its new size. After allocating new memory and copy the contents of old memory, stores the object in any unassigned locations.

Parameters

newSize	the new size of the vector
object	the object for any new, unassigned locations

Exceptions

bad_alloc	if memory cannot be allocated

2.1.3.13 template<typename T > unsigned Vector< T >::size ( ) const

Returns the size (i.e., the number of elements) of the vector.

Returns

an unsigned integer indicating the vector's size.

2.1.3.14 template<typename T > void Vector< T >::sort ( )

Sorts the elements of this vector in ascending order.

2.1.4 Member Data Documentation

2.1.4.1 template<typename T > unsigned Vector< T >::cap [private]

the capacity of this vector

2.1.4.2 template<typename T > T* Vector< T >::contents [private]

the contents of this vector

2.1.4.3 template<typename T > unsigned Vector< T >::sz [private]

the size of this vector

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

- vector.h

Index