Chapter 1

Class Index

1.1 Class List

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

Vector<T> ................................................................. 3
Chapter 2

Class Documentation

2.1 Vector<T> Class Template Reference

#include <vector.h>

Public Member Functions

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

2.1.1 Detailed Description

template<typename T> class Vector<T>

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

Author:
  Mark Maloof

Version:
  1.0 3/1/05
2.1.2 Constructor & Destructor Documentation

2.1.2.1 template<typename T> Vector<T>::Vector () [inline]

Default constructor.

2.1.2.2 template<typename T> Vector<T>::Vector (const unsigned, const T & v = T()) throw (bad_alloc) [inline]

Constructor for initializing a vector to a fixed size and containing a given value.

Exceptions:

- bad_alloc if memory cannot be allocated.

2.1.2.3 template<typename T> Vector<T>::Vector (const Vector<T>& v) throw (bad_alloc) [inline]

Copy constructor.

Exceptions:

- bad_alloc if memory cannot be allocated.

2.1.2.4 template<typename T> Vector<T>::~Vector () [inline]

Class destructor.

2.1.3 Member Function Documentation

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

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

Returns:

- true if empty; false otherwise.

2.1.3.2 template<typename T> unsigned Vector<T>::size () const [inline]

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

Returns:

- an unsigned integer indicating the vector’s size.
2.1 Vector<uint> Class Template Reference

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

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() [inline]

Removes the elements of the vector.

2.1.3.5 template<typename T> void Vector<T>::resize(const unsigned, const T & v = T()) throw (bad_alloc) [inline]

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

Parameters:

newSize the new size of the vector.

v the value for any new, unassigned elements.

Exceptions:

bad_alloc if memory cannot be allocated.

2.1.3.6 template<typename T> T & Vector<T>::at(const unsigned) const throw (VectorEmpty, out_of_range) [inline]

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:

VectorEmpty if vector is empty.

out_of_range if index parameter is out of bounds.
2.1.3.7  template<typename T> void Vector<T>::assign (const unsigned, const T & object) throw (VectorEmpty, out_of_range) [inline]

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:
  
VectorEmpty    if vector is empty.
  
out_of_range    if index parameter is out of bounds.

2.1.3.8  template<typename T> void Vector<T>::push_back (const T & object) throw (bad_alloc) [inline]

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.

2.1.3.9  template<typename T> void Vector<T>::insert (const unsigned, const T & object) throw (bad_alloc, out_of_range) [inline]

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.

2.1.3.10  template<typename T> void Vector<T>::remove (const unsigned) throw (VectorEmpty, out_of_range) [inline]

Removes the object stored in the given position.

Parameters:
  
i    the position of removal.
Exceptions:

- `VectorEmpty` if vector is empty.
- `out_of_range` if index parameter is out of bounds.

2.1.3.11 template<typename T> T & Vector<T>::operator[](const unsigned) const throw (VectorEmpty, out_of_range) [inline]

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:

- `VectorEmpty` if vector is empty.
- `out_of_range` if index parameter is out of bounds.

2.1.3.12 template<typename T> const Vector<T> & Vector<T>::operator=(const Vector<T> & v) throw (bad_alloc) [inline]

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.

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

- `vector.h`
Index

~Vector  
Vector, 4

assign  
Vector, 5

at  
Vector, 5

capacity  
Vector, 4

clear  
Vector, 5

empty  
Vector, 4

insert  
Vector, 6

operator=  
Vector, 7

push_back  
Vector, 6

remove  
Vector, 6

resize  
Vector, 5

size  
Vector, 4

Vector, 3  
~Vector, 4
assign, 5
at, 5
capacity, 4
clear, 5
empty, 4
insert, 6
operator=, 7
push_back, 6
remove, 6
resize, 5
size, 4
Vector, 4