Reference Manual

Generated by Doxygen 1.5.5

Wed Aug 4 15:22:15 2010

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 List< T > Class Template Reference

`#include <list.h>`

Public Member Functions

- List ()
- List (const List< T > &) throw (bad_alloc)
- ~List ()
- void add (unsigned, const T &) throw (bad_alloc, out_of_range)
- void addFirst (const T &) throw (bad_alloc)
- void addLast (const T &) throw (bad_alloc)
- void clear ()
- bool contains (const T &) const
- bool empty () const
- int indexOf (const T &) const
- T & get (unsigned) const throw (out_of_range)
- T & getFirst () const throw (NoSuchObject)
- T & getLast () const throw (NoSuchObject)
- T remove (unsigned) throw (out_of_range)
- T removeFirst () throw (NoSuchObject)
- T removeFirstOccurrence (const T &) throw (NoSuchObject)
- T removeLast () throw (NoSuchObject)
- T removeLastOccurrence (const T &) throw (NoSuchObject)
- T set (unsigned, const T &) throw (out_of_range)
- unsigned size () const
- const List< T > & operator= (const List< T > &) throw (bad_alloc)
- void printInternal (ostream &=cout)

2.1.1 Detailed Description

template<typename T> class List< T >

Implementation of a List ADT using a doubly-linked list.

Author:

Mark Maloof

Version:

1.1, 3/23/10

2.1.2 Constructor & Destructor Documentation

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

Default constructor.

2.1.2.2 template<typename T> List< T >::List (const List< T > & *list*) throw (bad_alloc) [inline]

Copy constructor.

Exceptions:

bad_alloc if memory cannot be allocated.

2.1.2.3 template<typename T> List< T >::~List () [inline]

Class destructor.

2.1.3 Member Function Documentation

2.1.3.1 template<typename T> void List< T >::add (unsigned *index*, const T & *object*) throw (bad_alloc, out_of_range) [inline]

Adds the specified object to this list at the specified position.

Parameters:

index the position at which to insert the object

object the object to be inserted

Exceptions:

bad_alloc if memory cannot be allocated

out_of_range if the index is out of range

2.1.3.2 template<typename T> void List< T >::addFirst (const T & *object*) throw (bad_alloc) [inline]

Adds the specified object to the front of the list.

Parameters:

object the object to be added to the front of the list

Exceptions:

bad_alloc if memory cannot be allocated

2.1.3.3 template<typename T> void List< T >::addLast (const T & *object*) throw (bad_alloc) [inline]

Adds the specified object to the end of the list.

Parameters:

object the object to be added to the back of the list

Exceptions:

bad_alloc if memory cannot be allocated

2.1.3.4 template<typename T> void List< T >::clear () [inline]

Clears this list by removing all of its elements.

2.1.3.5 template<typename T> bool List< T >::contains (const T & *object*) const [inline]

Returns true if this list contains the specified object

Parameters:

object the specified object

Returns:

true if the list contains the object; false otherwise

2.1.3.6 template<typename T> bool List< T >::empty () const [inline]

Returns true if this list is empty; returns false otherwise.

Returns:

true if empty; false otherwise

2.1.3.7 template<typename T> int List< T >::indexOf (const T & *object*) const [inline]

Returns the position of the specified object in this list. Returns -1 if the object is not in this list.

Parameters:

object the object to be found in the list

Returns:

the position of object in the list

2.1.3.8 template<typename T> T & List< T >::get (unsigned *index*) const throw (out_of_range) [inline]

Returns a reference to the object at the specified position in this list.

Parameters:

index the position of the object

Returns:

a reference to the object at the specified position

Exceptions:

out_of_range if the index is out of range

2.1.3.9 template<typename T> T & List< T >::getFirst () const throw (NoSuchObject) [inline]

Returns a reference to the first object in this list.

Returns:

a reference to the first object

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.10 template<typename T> T & List< T >::getLast () const throw (NoSuchObject) [inline]

Returns a reference to the last object in this list.

Returns:

a reference to the last object

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.11 template<typename T> T List< T >::remove (unsigned *index*) throw (out_of_range) [inline]

Removes and returns the object at the specified position.

Parameters:

index the position of the object

Returns:

the object at the specified position

Exceptions:

out_of_range if the index is out of range

2.1.3.12 template<typename T> T List< T >::removeFirst () throw (NoSuchObject) [inline]

Removes and returns the first object in this list.

Returns:

the object at the front of the list

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.13 template<typename T> T List< T >::removeFirstOccurrence (const T & *object*) throw (NoSuchObject) [inline]

Removes and returns the first occurrence of the object in this list.

Parameters:

object the object in the list

Returns:

the first occurence of object in this list

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.14 template<typename T> T List< T >::removeLast () throw (NoSuchObject) [inline]

Removes and returns the last object in this list.

Returns:

the last object in this list

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.15 template<typename T> T List< T >::removeLastOccurrence (const T & *object*) throw (NoSuchObject) [inline]

Removes and returns the last occurrence of the object in this list.

Parameters:

object the object in the list

Returns:

the last occurence of object in this list

Exceptions:

NoSuchObject if no such object exists in this list

2.1.3.16 template<typename T> T List< T >::set (unsigned *index*, const T & *object*) throw (out_of_range) [inline]

Sets the object at the specified position to the specified object and returns the replaced object.

Parameters:

index the position of the object

object the object

Returns:

the replaced object

Exceptions:

out_of_range if the index is out of range

2.1.3.17 template<typename T> unsigned List< T >::size () const [inline]

Returns the size (i.e., number of objects) of this list.

Returns:

an unsigned integer indicating this list's size

2.1.3.18 template<typename T> const List< T > & List< T >::operator= (const List< T > & *list*) throw (bad_alloc) [inline]

Returns a deep copy of the specified list.

Parameters:

list the list to be copied.

Returns:

a copy of the list.

Exceptions:

bad_alloc if memory cannot be allocated.

2.1.3.19 template<typename T> void List< T >::printInternal (ostream & *out* = cout) [inline]

A utility method that prints the internal state of this list.

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

- list.h

2.2 NoSuchObject Class Reference

`#include <nosuchobject.h>`

Public Member Functions

- **NoSuchObject** (const string &what)

2.2.1 Detailed Description

Implementation of the NoSuchObject exception class.

Author:

Mark Maloof

Version:

1.0, 3/23/10

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

- nosuchobject.h

Index