Reference Manual

Generated by Doxygen 1.6.3

Thu Mar 24 15:12:01 2011

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 addAll (const List< T > &) throw (bad_alloc)
- void addAll (unsigned, const List< 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)
- ListIterator< T > listIterator ()
- ListIterator< T > listIterator (unsigned) throw (out_of_range)
- 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
- T * toArray () const throw (bad_alloc)
- const List< T > & operator= (const List< T > &) throw (bad_alloc)
- void printInternal (ostream &=cout)

Private Member Functions

- void add (Node< T > *, const T &) throw (bad_alloc)
- void initialize ()
- Node< T > * getIthNode (unsigned) const throw (out_of_range)
- T remove (Node< T > *)

Private Attributes

- Node< T > * **frontPtr**
- Node< T > * **backPtr**
- unsigned **sz**

Friends

- ostream & operator<< (ostream &, const List< T > &)

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 (Node< T > * *current*, const T & *object*) throw (bad_alloc) [inline, private]

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

Parameters

current the position at which to insert the object

object the object to be inserted

Exceptions

bad_alloc if memory cannot be allocated

2.1.3.2 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

2.1.3.3 template<typename T > void List< T >::addAll (unsigned *index*, const List< T > & *list*) throw (bad_alloc, out_of_range) [inline]

Adds all of objects in the specified list to the specified position of this list.

Parameters

index the position at which to insert the object

list containing the objects to be added

Exceptions

bad_alloc if memory cannot be allocated

out_of_range if the index is out of range

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

Adds all of objects in the specified list to the end of this list.

Parameters

list containing the objects to be added

Exceptions

bad_alloc if memory cannot be allocated

2.1.3.5 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

2.1.3.6 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

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

Clears this list by removing all of its elements.

2.1.3.8 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

2.1.3.9 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

2.1.3.10 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

2.1.3.11 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

2.1.3.12 template<typename T > Node< T > * List< T >::getIthNode (unsigned *index*) const throw (out_of_range) [inline, private]

Returns a pointer to the node at the specified position.

Parameters

index the position of the object

Returns

a pointer to the node at the specified position

Exceptions

out_of_range if the position is out of range

2.1.3.13 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

2.1.3.14 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

2.1.3.15 template<typename T > void List< T >::initialize () [inline, private]

Sets the private data members of this list to their defaults values; that is, sets the points to null and size to zero.

2.1.3.16 template<typename T > ListIterator< T > List< T >::listIterator (unsigned *index*) throw (out_of_range) [inline]

Returns a bidirectional iterator for this list that starts at the specified position.

Parameters

index the starting position of the iterator

Returns

a bidirectional list iterator

Exceptions

out_of_range if the index is out of range

2.1.3.17 template<typename T > ListIterator< T > List< T >::listIterator () [inline]

Returns a bidirectional iterator for this list.

Returns

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.

2.1.3.20 template<typename T > T List< T >::remove (Node< T > * *current*) [inline, private]

Removes and returns the object to which current points.

Parameters

current the position of the object

Returns

the object at the specified position

2.1.3.21 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

2.1.3.22 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

2.1.3.23 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

2.1.3.24 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

2.1.3.25 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

2.1.3.26 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

2.1.3.27 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

2.1.3.28 template<typename T > T * List< T >::toArray () const throw (bad_alloc) [inline]

Returns an array containing the objects of this list.

Returns

an array containing the objects of this list

Exceptions

bad_alloc if memory cannot be allocated.

2.1.4 Friends And Related Function Documentation

2.1.4.1 template<typename T > ostream& operator<< (ostream & *out*, const List< T > & *list*) [friend]

Overloaded stream insertion operator that outputs the specified list to the output stream in the format [e1,e2,e3,...], where e1, e2, ... are the elements of the list.

Parameters

out the output stream

list the specified list

Returns

the modified output stream

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

- list.h

2.2 ListIterator< T > Class Template Reference

`#include <iterator.h>`

Public Member Functions

- ListIterator ()
- void **add** (const T &object)
- bool hasNext () const
- bool hasPrevious () const
- T & next () throw (NoSuchObject)
- T & previous () throw (NoSuchObject)
- void set (const T &)
- void printInternal () const

Private Attributes

- Node< T > * **current**
- bool **atFront**
- bool **atEnd**

Friends

- class List< T >

2.2.1 Detailed Description

template<typename T> class ListIterator< T >

Implements a bidirectional iterator for List.

Author

Mark Maloof

Version

1.0, 3/25/10

2.2.2 Constructor & Destructor Documentation

2.2.2.1 template<typename T > ListIterator< T >::ListIterator () [inline]

Default constructor.

2.2.3 Member Function Documentation

2.2.3.1 template<typename T > bool ListIterator< T >::hasNext () const [inline]

Returns true if this iterator has a next object.

Returns

true if this iterator has a next object; false otherwise.

2.2.3.2 template<typename T > bool ListIterator< T >::hasPrevious () const [inline]

Returns true if this iterator has a previous object.

Returns

true if this iterator has a previous object; false otherwise.

2.2.3.3 template<typename T > T & ListIterator< T >::next () throw (NoSuchObject) [inline]

Returns a reference to the next object in this iterator.

Returns

a reference to the next object

Exceptions

NoSuchObject if no such object exists in this iterator

2.2.3.4 template<typename T > T & ListIterator< T >::previous () throw (NoSuchObject) [inline]

Returns a reference to the previous object in this iterator.

Returns

a reference to the previous object

Exceptions

NoSuchObject if no such object exists in this iterator

2.2.3.5 template<typename T > void ListIterator< T >::printInternal () const [inline]

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

2.2.3.6 template<typename T > void ListIterator< T >::set (const T & *object*) [inline]

Sets the object at the position of this iterator to the specified object.

Parameters

object the object to be set

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

- iterator.h

2.3 Node< T > Class Template Reference

Public Member Functions

- **Node** (const T &)
- void **setObject** (const T &)
- T & **getObject** ()
- void **setNextPtr** (Node< T > ∗)
- Node< T > ∗ **getNextPtr** () const
- void **setPrevPtr** (Node< T > ∗)
- Node< T > ∗ **getPrevPtr** () const

Private Attributes

- T **object**
- Node< T > ∗ **nextPtr**
- Node< T > ∗ **prevPtr**

template<typename T> class Node< T >

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

- node.h

2.4 NoSuchObject Class Reference

`#include <nosuchobject.h>`

Public Member Functions

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

2.4.1 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