

Reference Manual

Generated by Doxygen 1.8.6

Sat Mar 25 2017 08:55:24

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	Class Documentation	5
3.1	List< typename > Class Template Reference	5
3.1.1	Detailed Description	6
3.1.2	Constructor & Destructor Documentation	6
3.1.2.1	List	6
3.1.2.2	List	6
3.1.2.3	~List	6
3.1.3	Member Function Documentation	7
3.1.3.1	add	7
3.1.3.2	add	8
3.1.3.3	addAll	8
3.1.3.4	addAll	8
3.1.3.5	addFirst	8
3.1.3.6	addLast	9
3.1.3.7	clear	9
3.1.3.8	contains	9
3.1.3.9	empty	9
3.1.3.10	get	9
3.1.3.11	getFirst	10
3.1.3.12	getlthNode	10
3.1.3.13	getLast	10
3.1.3.14	indexOf	11
3.1.3.15	initialize	11

3.1.3.16	listIterator	11
3.1.3.17	listIterator	11
3.1.3.18	operator=	11
3.1.3.19	printInternal	12
3.1.3.20	remove	12
3.1.3.21	remove	12
3.1.3.22	removeFirst	12
3.1.3.23	removeFirstOccurrence	13
3.1.3.24	removeLast	13
3.1.3.25	removeLastOccurrence	13
3.1.3.26	set	13
3.1.3.27	size	14
3.1.3.28	toArray	14
3.1.4	Friends And Related Function Documentation	14
3.1.4.1	operator<<	14
3.2	ListIterator< T > Class Template Reference	15
3.2.1	Detailed Description	15
3.2.2	Constructor & Destructor Documentation	15
3.2.2.1	ListIterator	15
3.2.3	Member Function Documentation	16
3.2.3.1	hasNext	16
3.2.3.2	hasPrevious	16
3.2.3.3	next	16
3.2.3.4	previous	16
3.2.3.5	printInternal	16
3.2.3.6	set	16
3.3	Node< T > Class Template Reference	17
3.4	NoSuchObject Class Reference	17
Index		18

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- List< typename > 5
- ListIterator< T > 15
- logic_error
 - NoSuchObject 17
- Node< T > 17

Chapter 2

Class Index

2.1 Class List

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

List< typename >	5
ListIterator< T >	15
Node< T >	17
NoSuchObject	17

Chapter 3

Class Documentation

3.1 List< typename > 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` > * `getlthNode` (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` > &)

3.1.1 Detailed Description

template<typename>class `List`< typename >

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

Author

Mark Maloof

Version

1.1, 3/23/10

3.1.2 Constructor & Destructor Documentation

3.1.2.1 template<typename T > `List`< T >::List ()

Default constructor.

3.1.2.2 template<typename T > `List`< T >::List (const `List`< T > & *list*) throw `bad_alloc`

Copy constructor.

Exceptions

<code>bad_alloc</code>	if memory cannot be allocated.
------------------------	--------------------------------

3.1.2.3 template<typename T > `List`< T >::~List ()

Class destructor.

3.1.3 Member Function Documentation

3.1.3.1 `template<typename T> void List< T >::add (unsigned index, const T & object) throw bad_alloc, out_of_range)`

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

Parameters

<i>index</i>	the position at which to insert the object
<i>object</i>	the object to be inserted

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
<i>out_of_range</i>	if the index is out of range

3.1.3.2 `template<typename T > void List< T >::add (Node< T > * current, const T & object) throw bad_alloc)`
`[private]`

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

Parameters

<i>current</i>	the position at which to insert the object
<i>object</i>	the object to be inserted

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
------------------	-------------------------------

3.1.3.3 `template<typename T > void List< T >::addAll (const List< T > & list) throw bad_alloc)`

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

Parameters

<i>list</i>	containing the objects to be added
-------------	------------------------------------

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
------------------	-------------------------------

3.1.3.4 `template<typename T > void List< T >::addAll (unsigned index, const List< T > & list) throw bad_alloc, out_of_range)`

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

Parameters

<i>index</i>	the position at which to insert the object
<i>list</i>	containing the objects to be added

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
<i>out_of_range</i>	if the index is out of range

3.1.3.5 `template<typename T > void List< T >::addFirst (const T & object) throw bad_alloc)`

Adds the specified object to the front of the list.

Parameters

<i>object</i>	the object to be added to the front of the list
---------------	---

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
------------------	-------------------------------

3.1.3.6 `template<typename T> void List< T >::addLast (const T & object) throw bad_alloc)`

Adds the specified object to the end of the list.

Parameters

<i>object</i>	the object to be added to the back of the list
---------------	--

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated
------------------	-------------------------------

3.1.3.7 `template<typename T> void List< T >::clear ()`

Clears this list by removing all of its elements.

3.1.3.8 `template<typename T> bool List< T >::contains (const T & object) const`

Returns true if this list contains the specified object

Parameters

<i>object</i>	the specified object
---------------	----------------------

Returns

true if the list contains the object; false otherwise

3.1.3.9 `template<typename T> bool List< T >::empty () const`

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

Returns

true if empty; false otherwise

3.1.3.10 `template<typename T> T & List< T >::get (unsigned index) const throw out_of_range)`

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

Parameters

<i>index</i>	the position of the object
--------------	----------------------------

Returns

a reference to the object at the specified position

Exceptions

<i>out_of_range</i>	if the index is out of range
---------------------	------------------------------

3.1.3.11 `template<typename T> T & List< T >::getFirst () const throw NoSuchObject)`

Returns a reference to the first object in this list.

Returns

a reference to the first object

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.12 `template<typename T> Node< T > * List< T >::getlthNode (unsigned index) const throw out_of_range)`
[private]

Returns a pointer to the node at the specified position.

Parameters

<i>index</i>	the position of the object
--------------	----------------------------

Returns

a pointer to the node at the specified position

Exceptions

<i>out_of_range</i>	if the position is out of range
---------------------	---------------------------------

3.1.3.13 `template<typename T> T & List< T >::getLast () const throw NoSuchObject)`

Returns a reference to the last object in this list.

Returns

a reference to the last object

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.14 `template<typename T> int List< T >::indexOf (const T & object) const`

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

Parameters

<i>object</i>	the object to be found in the list
---------------	------------------------------------

Returns

the position of object in the list

3.1.3.15 `template<typename T> void List< T >::initialize () [private]`

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

3.1.3.16 `template<typename T> ListIterator< T > List< T >::listIterator ()`

Returns a bidirectional iterator for this list.

Returns

3.1.3.17 `template<typename T> ListIterator< T > List< T >::listIterator (unsigned index) throw out_of_range)`

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

Parameters

<i>index</i>	the starting position of the iterator
--------------	---------------------------------------

Returns

a bidirectional list iterator

Exceptions

<i>out_of_range</i>	if the index is out of range
---------------------	------------------------------

3.1.3.18 `template<typename T> const List< T > & List< T >::operator= (const List< T > & list) throw bad_alloc)`

Returns a deep copy of the specified list.

Parameters

<i>list</i>	the list to be copied.
-------------	------------------------

Returns

a copy of the list.

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated.
------------------	--------------------------------

3.1.3.19 `template<typename T> void List< T >::printInternal (ostream & out = cout)`

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

3.1.3.20 `template<typename T> T List< T >::remove (unsigned index) throw out_of_range)`

Removes and returns the object at the specified position.

Parameters

<i>index</i>	the position of the object
--------------	----------------------------

Returns

the object at the specified position

Exceptions

<i>out_of_range</i>	if the index is out of range
---------------------	------------------------------

3.1.3.21 `template<typename T> T List< T >::remove (Node< T > * current) [private]`

Removes and returns the object to which current points.

Parameters

<i>current</i>	the position of the object
----------------	----------------------------

Returns

the object at the specified position

3.1.3.22 `template<typename T> T List< T >::removeFirst () throw NoSuchObject)`

Removes and returns the first object in this list.

Returns

the object at the front of the list

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.23 `template<typename T> T List< T >::removeFirstOccurrence (const T & object) throw NoSuchObject)`

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

Parameters

<i>object</i>	the object in the list
---------------	------------------------

Returns

the first occurrence of object in this list

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.24 `template<typename T> T List< T >::removeLast () throw NoSuchObject)`

Removes and returns the last object in this list.

Returns

the last object in this list

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.25 `template<typename T> T List< T >::removeLastOccurrence (const T & object) throw NoSuchObject)`

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

Parameters

<i>object</i>	the object in the list
---------------	------------------------

Returns

the last occurrence of object in this list

Exceptions

<i>NoSuchObject</i>	if no such object exists in this list
---------------------	---------------------------------------

3.1.3.26 `template<typename T> T List< T >::set (unsigned index, const T & object) throw out_of_range)`

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

Parameters

<i>index</i>	the position of the object
<i>object</i>	the object

Returns

the replaced object

Exceptions

<i>out_of_range</i>	if the index is out of range
---------------------	------------------------------

3.1.3.27 `template<typename T> unsigned List< T >::size () const`

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

Returns

an unsigned integer indicating this list's size

3.1.3.28 `template<typename T> T * List< T >::toArray () const throw bad_alloc`

Returns an array containing the objects of this list.

Returns

an array containing the objects of this list

Exceptions

<i>bad_alloc</i>	if memory cannot be allocated.
------------------	--------------------------------

3.1.4 Friends And Related Function Documentation**3.1.4.1** `template<typename > 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

<i>out</i>	the output stream
<i>list</i>	the specified list

Returns

the modified output stream

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

- iterator.h
- list.h

3.2 ListIterator< T > Class Template Reference

```
#include <iterator.h>
```

Public Member Functions

- [ListIterator](#) ()
- bool [hasNext](#) () const
- bool [hasPrevious](#) () const
- T & [next](#) () throw (NoSuchElementException)
- T & [previous](#) () throw (NoSuchElementException)
- void [set](#) (const T &)
- void [printInternal](#) () const

Private Attributes

- [Node](#)< T > * **current**
- bool **atFront**
- bool **atEnd**

Friends

- class [List](#)< T >

3.2.1 Detailed Description

```
template<typename T>class ListIterator< T >
```

Implements a bidirectional iterator for [List](#).

Author

Mark Maloof

Version

1.0, 3/25/10

3.2.2 Constructor & Destructor Documentation

3.2.2.1 `template<typename T > ListIterator< T >::ListIterator ()`

Default constructor.

3.2.3 Member Function Documentation

3.2.3.1 `template<typename T> bool ListIterator< T >::hasNext () const`

Returns true if this iterator has a next object.

Returns

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

3.2.3.2 `template<typename T> bool ListIterator< T >::hasPrevious () const`

Returns true if this iterator has a previous object.

Returns

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

3.2.3.3 `template<typename T> T & ListIterator< T >::next () throw NoSuchElementException`

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
------------------------------	---

3.2.3.4 `template<typename T> T & ListIterator< T >::previous () throw NoSuchElementException`

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
------------------------------	---

3.2.3.5 `template<typename T> void ListIterator< T >::printInternal () const`

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

3.2.3.6 `template<typename T> void ListIterator< T >::set (const T & object)`

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

Parameters

<i>object</i>	the object to be set
---------------	----------------------

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

- iterator.h

3.3 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

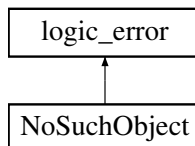
- T **object**
- Node< T > * **nextPtr**
- Node< T > * **prevPtr**

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

- node.h

3.4 NoSuchObject Class Reference

Inheritance diagram for NoSuchObject:



Public Member Functions

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

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

- nosuchobject.h

Index

- ~List
 - List, 6
- add
 - List, 7, 8
- addAll
 - List, 8
- addFirst
 - List, 8
- addLast
 - List, 9
- clear
 - List, 9
- contains
 - List, 9
- empty
 - List, 9
- get
 - List, 9
- getFirst
 - List, 10
- getIthNode
 - List, 10
- getLast
 - List, 10
- hasNext
 - ListIterator, 16
- hasPrevious
 - ListIterator, 16
- indexOf
 - List, 11
- initialize
 - List, 11
- List
 - ~List, 6
 - add, 7, 8
 - addAll, 8
 - addFirst, 8
 - addLast, 9
 - clear, 9
 - contains, 9
 - empty, 9
 - get, 9
 - getFirst, 10
 - getIthNode, 10
 - getLast, 10
 - indexOf, 11
 - initialize, 11
 - List, 6
 - listIterator, 11
 - operator<<, 14
 - operator=, 11
 - printInternal, 12
 - remove, 12
 - removeFirst, 12
 - removeFirstOccurrence, 13
 - removeLast, 13
 - removeLastOccurrence, 13
 - set, 13
 - size, 14
 - toArray, 14
- List< typename >, 5
- ListIterator
 - hasNext, 16
 - hasPrevious, 16
 - ListIterator, 15
 - ListIterator, 15
 - next, 16
 - previous, 16
 - printInternal, 16
 - set, 16
- listIterator
 - List, 11
- ListIterator< T >, 15
- next
 - ListIterator, 16
- NoSuchObject, 17
- Node< T >, 17
- operator<<
 - List, 14
- operator=
 - List, 11
- previous
 - ListIterator, 16

printInternal
List, [12](#)
ListIterator, [16](#)

remove
List, [12](#)
removeFirst
List, [12](#)
removeFirstOccurrence
List, [13](#)
removeLast
List, [13](#)
removeLastOccurrence
List, [13](#)

set
List, [13](#)
ListIterator, [16](#)

size
List, [14](#)

toArray
List, [14](#)