

PA01

Generated by Doxygen 1.8.6

Mon Sep 19 2016 00:25:52



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	LinkedList< ItemType > Class Template Reference . . . . .	7
4.1.1	Constructor & Destructor Documentation . . . . .	7
4.1.1.1	LinkedList . . . . .	7
4.1.2	Member Function Documentation . . . . .	8
4.1.2.1	clear . . . . .	8
4.1.2.2	getEntry . . . . .	8
4.1.2.3	getLength . . . . .	8
4.1.2.4	insert . . . . .	8
4.1.2.5	isEmpty . . . . .	9
4.1.2.6	remove . . . . .	9
4.1.2.7	replace . . . . .	9
4.2	ListInterface< ItemType > Class Template Reference . . . . .	10
4.2.1	Member Function Documentation . . . . .	10
4.2.1.1	clear . . . . .	10
4.2.1.2	getEntry . . . . .	10
4.2.1.3	getLength . . . . .	11
4.2.1.4	insert . . . . .	11
4.2.1.5	isEmpty . . . . .	11
4.2.1.6	remove . . . . .	12
4.2.1.7	replace . . . . .	12
4.3	Node< ItemType > Class Template Reference . . . . .	12
4.3.1	Constructor & Destructor Documentation . . . . .	13

4.3.1.1	Node	13
4.3.1.2	Node	13
4.3.2	Member Function Documentation	13
4.3.2.1	getItem	13
4.3.2.2	setItem	13
4.3.2.3	setNext	14
4.4	PrecondViolatedExcept Class Reference	14
4.4.1	Constructor & Destructor Documentation	14
4.4.1.1	PrecondViolatedExcept	14
<b>5</b>	<b>File Documentation</b>	<b>15</b>
5.1	LinkedList.cpp File Reference	15
5.1.1	Detailed Description	15
5.2	LinkedList.h File Reference	15
5.2.1	Detailed Description	15
5.3	ListInterface.h File Reference	16
5.3.1	Detailed Description	16
5.4	Node.cpp File Reference	16
5.4.1	Detailed Description	16
5.5	Node.h File Reference	16
5.5.1	Detailed Description	17
5.6	PrecondViolatedExcept.cpp File Reference	17
5.6.1	Detailed Description	17
5.7	PrecondViolatedExcept.h File Reference	17
5.7.1	Detailed Description	17
<b>Index</b>		<b>19</b>

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

ListInterface< ItemType > . . . . .	10
LinkedList< ItemType > . . . . .	7
logic_error	
PrecondViolatedExcept . . . . .	14
Node< ItemType > . . . . .	12



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">LinkedList&lt; ItemType &gt;</a>	7
<a href="#">ListInterface&lt; ItemType &gt;</a>	10
<a href="#">Node&lt; ItemType &gt;</a>	12
<a href="#">PrecondViolatedExcept</a>	14





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">LinkedList.cpp</a>	Implementation file for the <a href="#">LinkedList</a> data type . . . . .	15
<a href="#">LinkedList.h</a>	Interface file for the <a href="#">LinkedList</a> data type . . . . .	15
<a href="#">ListInterface.h</a>	Interface file for the List ADT . . . . .	16
<a href="#">Node.cpp</a>	Implementation file for the <a href="#">Node</a> data type . . . . .	16
<a href="#">Node.h</a>	Interface file for the <a href="#">Node</a> data type . . . . .	16
<a href="#">PrecondViolatedExcept.cpp</a>	Implementation file for the <a href="#">PrecondViolatedExcept</a> data type . . . . .	17
<a href="#">PrecondViolatedExcept.h</a>	Interface file for the <a href="#">PrecondViolatedExcept</a> data type . . . . .	17

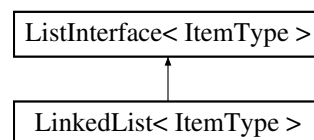


## Chapter 4

# Class Documentation

### 4.1 `LinkedList< ItemType >` Class Template Reference

Inheritance diagram for `LinkedList< ItemType >`:



#### Public Member Functions

- `LinkedList ()`  
*Constructs an empty linked list.*
- `LinkedList (const LinkedList< ItemType > &other)`  
*Copies the contents of another linked list.*
- `virtual ~LinkedList ()`  
*Destructs the linked list object.*
- `bool isEmpty () const`  
*Sees whether this list is empty.*
- `int getLength () const`
- `bool insert (int newPosition, const ItemType &newEntry)`
- `bool remove (int position)`
- `void clear ()`
- `ItemType getEntry (int position) const throw (PrecondViolatedExcept)`  
*Gets the entry found at the given position.*
- `void replace (int position, const ItemType &newEntry) throw (PrecondViolatedExcept)`  
*Replaces the entry at the given position with the specified new entry.*

#### 4.1.1 Constructor & Destructor Documentation

4.1.1.1 `template<class ItemType > LinkedList< ItemType >::LinkedList ( const LinkedList< ItemType > & other )`

Copies the contents of another linked list.

## Parameters

<i>other</i>	The linked list to copy.
--------------	--------------------------

## 4.1.2 Member Function Documentation

4.1.2.1 `template<class ItemType > void LinkedList< ItemType >::clear ( ) [virtual]`

Removes all entries from this list.

## Postcondition

List contains no entries and the count of items is 0.

Implements [ListInterface< ItemType >](#).

4.1.2.2 `template<class ItemType > ItemType LinkedList< ItemType >::getEntry ( int position ) const throw PrecondViolatedExcept) [virtual]`

Gets the entry found at the given position.

Throws [PrecondViolatedExcept](#) on precondition violation.

## Precondition

`getLength() != 0, 1 <= position <= getLength()`

## Parameters

<i>position</i>	The position of the entry to return.
-----------------	--------------------------------------

## Returns

The entry at the specified position.

Implements [ListInterface< ItemType >](#).

4.1.2.3 `template<class ItemType > int LinkedList< ItemType >::getLength ( ) const [virtual]`

Gets the current number of entries in this list.

## Returns

The integer number of entries currently in the list.

Implements [ListInterface< ItemType >](#).

4.1.2.4 `template<class ItemType > bool LinkedList< ItemType >::insert ( int newPosition, const ItemType & newEntry ) [virtual]`

Inserts an entry into this list at a given position.

## Precondition

None.

## Postcondition

If `1 <= position <= getLength() + 1` and the insertion is successful, `newEntry` is at the given position in the list, other entries are renumbered accordingly, and the returned value is true.

## Parameters

<i>newPosition</i>	The list position at which to insert newEntry.
<i>newEntry</i>	The entry to insert into the list.

## Returns

True if insertion is successful, or false if not.

Implements [ListInterface< ItemType >](#).

4.1.2.5 `template<class ItemType > bool LinkedList< ItemType >::isEmpty ( ) const [virtual]`

Sees whether this list is empty.

## Returns

True if the list is empty, false otherwise.

Implements [ListInterface< ItemType >](#).

4.1.2.6 `template<class ItemType > bool LinkedList< ItemType >::remove ( int position ) [virtual]`

Removes the entry at a given position from this list.

## Precondition

None.

## Postcondition

If  $1 \leq \text{position} \leq \text{getLength}()$  and the removal is successful, the entry at the given position in the list is removed, other items are renumbered accordingly, and the returned value is true.

## Parameters

<i>position</i>	The list position of the entry to remove.
-----------------	---

## Returns

True if removal is successful, or false if not.

Implements [ListInterface< ItemType >](#).

4.1.2.7 `template<class ItemType > void LinkedList< ItemType >::replace ( int position, const ItemType & newEntry ) throw PrecondViolatedExcept [virtual]`

Replaces the entry at the given position with the specified new entry.

Throws [PrecondViolatedExcept](#) on precondition violation.

## Precondition

$\text{getLength}() \neq 0, 1 \leq \text{position} \leq \text{getLength}()$

## Postcondition

The entry at the given position has been replaced.

## Parameters

<i>position</i>	The position of the entry to replace.
<i>newEntry</i>	The new value of the entry.

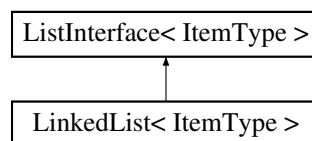
Implements [ListInterface< ItemType >](#).

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

- [LinkedList.h](#)
- [LinkedList.cpp](#)

## 4.2 ListInterface< ItemType > Class Template Reference

Inheritance diagram for ListInterface< ItemType >:



### Public Member Functions

- virtual bool [isEmpty](#) () const =0
- virtual int [getLength](#) () const =0
- virtual bool [insert](#) (int newPosition, const ItemType &newEntry)=0
- virtual bool [remove](#) (int position)=0
- virtual void [clear](#) ()=0
- virtual ItemType [getEntry](#) (int position) const =0
- virtual void [replace](#) (int position, const ItemType &newEntry)=0

### 4.2.1 Member Function Documentation

#### 4.2.1.1 `template<class ItemType > virtual void ListInterface< ItemType >::clear ( ) [pure virtual]`

Removes all entries from this list.

##### Postcondition

List contains no entries and the count of items is 0.

Implemented in [LinkedList< ItemType >](#).

#### 4.2.1.2 `template<class ItemType > virtual ItemType ListInterface< ItemType >::getEntry ( int position ) const [pure virtual]`

Gets the entry at the given position in this list.

##### Precondition

`1 <= position <= getLength()`.

##### Postcondition

The desired entry has been returned.

## Parameters

<i>position</i>	The list position of the desired entry.
-----------------	---

## Returns

The entry at the given position.

Implemented in [LinkedList< ItemType >](#).

**4.2.1.3** `template<class ItemType > virtual int ListInterface< ItemType >::getLength ( ) const` [pure virtual]

Gets the current number of entries in this list.

## Returns

The integer number of entries currently in the list.

Implemented in [LinkedList< ItemType >](#).

**4.2.1.4** `template<class ItemType > virtual bool ListInterface< ItemType >::insert ( int newPosition, const ItemType & newEntry )` [pure virtual]

Inserts an entry into this list at a given position.

## Precondition

None.

## Postcondition

If  $1 \leq \text{position} \leq \text{getLength}() + 1$  and the insertion is successful, *newEntry* is at the given position in the list, other entries are renumbered accordingly, and the returned value is true.

## Parameters

<i>newPosition</i>	The list position at which to insert <i>newEntry</i> .
<i>newEntry</i>	The entry to insert into the list.

## Returns

True if insertion is successful, or false if not.

Implemented in [LinkedList< ItemType >](#).

**4.2.1.5** `template<class ItemType > virtual bool ListInterface< ItemType >::isEmpty ( ) const` [pure virtual]

Sees whether this list is empty.

## Returns

True if the list is empty; otherwise returns false.

Implemented in [LinkedList< ItemType >](#).

**4.2.1.6** `template<class ItemType > virtual bool ListInterface< ItemType >::remove ( int position )` [pure virtual]

Removes the entry at a given position from this list.

#### Precondition

None.

#### Postcondition

If  $1 \leq \text{position} \leq \text{getLength}()$  and the removal is successful, the entry at the given position in the list is removed, other items are renumbered accordingly, and the returned value is true.

#### Parameters

<i>position</i>	The list position of the entry to remove.
-----------------	---

#### Returns

True if removal is successful, or false if not.

Implemented in [LinkedList< ItemType >](#).

**4.2.1.7** `template<class ItemType > virtual void ListInterface< ItemType >::replace ( int position, const ItemType & newEntry )` [pure virtual]

Replaces the entry at the given position in this list.

#### Precondition

$1 \leq \text{position} \leq \text{getLength}()$ .

#### Postcondition

The entry at the given position is newEntry.

#### Parameters

<i>position</i>	The list position of the entry to replace.
<i>newEntry</i>	The replacement entry.

Implemented in [LinkedList< ItemType >](#).

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

- [ListInterface.h](#)

## 4.3 Node< ItemType > Class Template Reference

### Public Member Functions

- [Node](#) ()  
*Construct an empty node.*
- [Node](#) (const ItemType &anItem)  
*Construct a node containing the given item.*



- **Node** (const ItemType &anItem, **Node**< ItemType > \*nextNodePtr)  
*Construct a node containing the given item and pointing to the given next node.*
- void **setItem** (const ItemType &anItem)  
*Set the contained item to the given item.*
- void **setNext** (**Node**< ItemType > \*nextNodePtr)  
*Set the next node to the given node.*
- ItemType **getItem** () const  
*Get the item contained by this node.*
- **Node**< ItemType > \* **getNext** () const

#### 4.3.1 Constructor & Destructor Documentation

##### 4.3.1.1 `template<class ItemType > Node< ItemType >::Node ( const ItemType & anItem )`

Construct a node containing the given item.

###### Postcondition

The node will contain the given item.

###### Parameters

<i>anItem</i>	The item to contain.
---------------	----------------------

##### 4.3.1.2 `template<class ItemType > Node< ItemType >::Node ( const ItemType & anItem, Node< ItemType > * nextNodePtr )`

Construct a node containing the given item and pointing to the given next node.

###### Postcondition

The node will contain the given item and will point to the given next node.

###### Parameters

<i>anItem</i>	The item to contain.
<i>nextNodePtr</i>	The pointer to the next node.

#### 4.3.2 Member Function Documentation

##### 4.3.2.1 `template<class ItemType > ItemType Node< ItemType >::getItem ( ) const`

Get the item contained by this node.

###### Returns

The item contained by this node.

##### 4.3.2.2 `template<class ItemType > void Node< ItemType >::setItem ( const ItemType & anItem )`

Set the contained item to the given item.

###### Postcondition

The item will have been set to the given item.

## Parameters

<i>anItem</i>	The new item to be contained.
---------------	-------------------------------

4.3.2.3 `template<class ItemType > void Node< ItemType >::setNext ( Node< ItemType > * nextNodePtr )`

Set the next node to the given node.

## Postcondition

This node will point to the given next node.

## Parameters

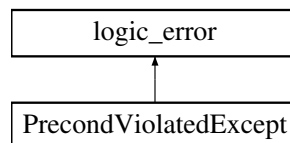
<i>nextNodePtr</i>	The pointer to the next node.
--------------------	-------------------------------

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

- [Node.h](#)
- [Node.cpp](#)

## 4.4 PrecondViolatedExcept Class Reference

Inheritance diagram for PrecondViolatedExcept:



### Public Member Functions

- [PrecondViolatedExcept](#) (const std::string &message="")  
Construct a [PrecondViolatedExcept](#) with the given message.

#### 4.4.1 Constructor & Destructor Documentation

4.4.1.1 `PrecondViolatedExcept::PrecondViolatedExcept ( const std::string & message = " " )`

Construct a [PrecondViolatedExcept](#) with the given message.

## Postcondition

The constructed object's message will match the one specified by the message parameter.

## Parameters

<i>message</i>	The message to be held.
----------------	-------------------------

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

- [PrecondViolatedExcept.h](#)
- [PrecondViolatedExcept.cpp](#)

## Chapter 5

# File Documentation

### 5.1 LinkedList.cpp File Reference

Implementation file for the [LinkedList](#) data type.

```
#include "LinkedList.h"
```

#### 5.1.1 Detailed Description

Implementation file for the [LinkedList](#) data type.

Author

Matthew Bauer

Implements the [LinkedList](#) data type Adapted from/inspired by "Data Abstractions & Problem Solving with C++" Seventh Edition by Frank M. Carrano and Timothy M. Henry.

### 5.2 LinkedList.h File Reference

Interface file for the [LinkedList](#) data type.

```
#include "ListInterface.h"  
#include "Node.h"  
#include "PrecondViolatedExcept.h"  
#include "LinkedList.cpp"
```

#### Classes

- class [LinkedList](#)< [ItemType](#) >

#### 5.2.1 Detailed Description

Interface file for the [LinkedList](#) data type.

**Author**

Matthew Bauer

Specifies the interface of the [LinkedList](#) data type Adapted from/inspired by "Data Abstractions & Problem Solving with C++" Seventh Edition by Frank M. Carrano and Timothy M. Henry.

## 5.3 ListInterface.h File Reference

Interface file for the List ADT.

**Classes**

- class [ListInterface](#)< [ItemType](#) >

### 5.3.1 Detailed Description

Interface file for the List ADT.

**Author**

Rory Pierce

Specifies the implementation contract of the List ADT

**Version**

0.10

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

## 5.4 Node.cpp File Reference

Implementation file for the [Node](#) data type.

```
#include "Node.h"
```

### 5.4.1 Detailed Description

Implementation file for the [Node](#) data type.

**Author**

Matthew Bauer

Implements the [Node](#) data type

## 5.5 Node.h File Reference

Interface file for the [Node](#) data type.

```
#include "Node.cpp"
```

## Classes

- class [Node< ItemType >](#)

### 5.5.1 Detailed Description

Interface file for the [Node](#) data type.

#### Author

Matthew Bauer

Specifies the interface of the [Node](#) data type Specification taken from "Data Abstractions a& Problem Solving with C++" Seventh Edition by Frank Carrano and Timothy M.; doxygen comments written by Matthew Bauer (student).

## 5.6 PrecondViolatedExcept.cpp File Reference

Implementation file for the [PrecondViolatedExcept](#) data type.

```
#include "PrecondViolatedExcept.h"
```

### 5.6.1 Detailed Description

Implementation file for the [PrecondViolatedExcept](#) data type.

#### Author

Matthew Bauer

Implements the [PrecondViolatedExcept](#) data type Code taken from "Data Abstractions & Problem Solving with C++" Seventh Edition by Frank Carrano and Timothy M.

## 5.7 PrecondViolatedExcept.h File Reference

Interface file for the [PrecondViolatedExcept](#) data type.

```
#include <stdexcept>
#include <string>
```

## Classes

- class [PrecondViolatedExcept](#)

### 5.7.1 Detailed Description

Interface file for the [PrecondViolatedExcept](#) data type.

**Author**

Matthew Bauer

Specifies the interface of the [PrecondViolatedExcept](#) data type Specification taken from "Data Abstractions & Problem Solving with C++" Seventh Edition by Frank Carrano and Timothy M.; doxygen comments written by Matthew Bauer (student).

# Index

clear  
    LinkedList, 8  
    ListInterface, 10

getEntry  
    LinkedList, 8  
    ListInterface, 10

getItem  
    Node, 13

getLength  
    LinkedList, 8  
    ListInterface, 11

insert  
    LinkedList, 8  
    ListInterface, 11

isEmpty  
    LinkedList, 9  
    ListInterface, 11

LinkedList  
    clear, 8  
    getEntry, 8  
    getLength, 8  
    insert, 8  
    isEmpty, 9  
    LinkedList, 7  
    LinkedList, 7  
    remove, 9  
    replace, 9

LinkedList< ItemType >, 7

LinkedList.cpp, 15

LinkedList.h, 15

ListInterface  
    clear, 10  
    getEntry, 10  
    getLength, 11  
    insert, 11  
    isEmpty, 11  
    remove, 11  
    replace, 12

ListInterface< ItemType >, 10

ListInterface.h, 16

Node  
    getItem, 13  
    Node, 13  
    setItem, 13  
    setNext, 14

Node< ItemType >, 12

Node.cpp, 16

Node.h, 16

PrecondViolatedExcept, 14  
    PrecondViolatedExcept, 14  
    PrecondViolatedExcept, 14  
PrecondViolatedExcept.cpp, 17  
PrecondViolatedExcept.h, 17

remove  
    LinkedList, 9  
    ListInterface, 11

replace  
    LinkedList, 9  
    ListInterface, 12

setItem  
    Node, 13

setNext  
    Node, 14