PA01 - Linked List

Generated by Doxygen 1.8.6

Thu Sep 15 2016 14:54:50

Contents

1 Hierarchical Index

1.1 Class Hierarchy

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

2 Class Index

2.1 Class List

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

3 File Index

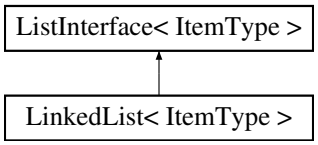
3.1 File List

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

4 Class Documentation

4.1 LinkedList< ItemType > Class Template Reference

Inheritance diagram for LinkedList< ItemType >:



Public Member Functions

- LinkedList ()

  Constructor for class LinkedList.
- virtual ~LinkedList ()

  Destructor for class LinkedList.
- bool isEmpty () const

  Function checks if LinkedList is empty.
- int getLength () const

  Function gets the itemCount of the Linked List.
- bool insert (int newPosition, const ItemType &newEntry)

  Function inserts a Node in a LinkedList.
- bool remove (int position)

  Function removes a Node from a LinkedList.
- void clear ()

  Function clears the LinkedList.
- ItemType getEntry (int position) const throw (PrecondViolatedExcept)

  Function finds a node at a given position and return the item stored in that node.
- ItemType replace (int position, const ItemType &newEntry) throw (PrecondViolatedExcept)

  Function finds a node at a given position and replaces that Node's data with another entry.

Private Member Functions

- Node< ItemType > * getNodeAt (int position) const

  Function finds a node at a given position and return the pointer to that node.

Private Attributes

- Node< ItemType > * **headPtr**
- int **itemCount**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 template<class ItemType > LinkedList< ItemType >::LinkedList ( )

Constructor for class LinkedList.

Able to construct a LinkedList object with Node<ItemType>* headPtr and int itemCount

Precondition

Node<ItemType>* headPtr is intialized to NULL
int itemCount is intialized to 0

Postcondition

headPtr and itemCount are not changed

Note

: None

4.1.1.2 template<class ItemType > LinkedList< ItemType >::~LinkedList ( ) [virtual]

Destructor for class LinkedList.

Able to destruct a LinkedList object by using clear()

Note

: None

4.1.2 Member Function Documentation

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

Function clears the LinkedList.

Function clears the LinkedList using the isEmpty function and the remove function

Precondition

None

Postcondition

None

Algorithm

Function uses a while loop that removes each node until isEmtpy() returns true

Parameters

in	*None*	
out	*None*	

Returns

None

Note

: None

Implements ListInterface< ItemType >.

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

Function finds a node at a given position and return the item stored in that node.

Function points to the head pointer and searches through the list until the specified position is reached and then returns the item of the Node in that position

Precondition

Node<ItemType>* nodePtr is intialized to Node at specified position
int position is intialized some specified value

Postcondition

nodePtr and position are not changed
curPtr gets the node at position

Algorithm

nodePtr is equal to the Node at position and the item in that node is returned

Parameters

in	*position*	holds the specified position in the LinkedList of the Node
out	*None*	

Returns

nodePtr->getItem()

Note

: None

Implements ListInterface< ItemType >.

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

Function gets the itemCount of the Linked List.

Function returns the length of the LinkedList by returning itemCount

Precondition

itemCount contains some value

Postcondition

itemCount is not changed

None

Parameters

in	*None*	
out	*None*	

Returns

itemCount

Note

: None

Implements ListInterface< ItemType >.

4.1.2.4 template<class ItemType > Node< ItemType > * LinkedList< ItemType >::getNodeAt (int *position*) const [private]

Function finds a node at a given position and return the pointer to that node.

Function points to the head pointer and searches through the list until the specified position is reached and then returns the node in that position

Precondition

Node<ItemType>* curPtr is intialized to headPtr
int current is intialized to 1

Postcondition

headPtr and position are not changed
curPtr gets the node at position
current gets position - 1

Algorithm

curPtr is calculated through a for loop that goes until current is equal to position - 1. At this point curPtr gets the Node that is stored in its next

Parameters

in	*position*	holds the specified position in the LinkedList of the Node
out	*None*	

Returns

curPtr

Note

: None

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

Function inserts a Node in a LinkedList.

Function inserts a Node in a LinkedList using a specified position and entry that the Node will contain

Precondition

int newPosition is initialized to some specified value
const ItemType& newEntry is initialized to some specified value
Node<ItemType>* newNodePtr is initialized to the value contained in newEntry
Node<ItemType>* prevPtr is initalized to the Node at position - 1

Postcondition

newPosition and newEntry are not changed
itemCount is incremented by one
newNodePtr's next value is either what headPtr used to be or what prevPtr used to be depending on the position
if the position is not equal to one, prevPtr's next value is now newNodePtr, otherwise headPtr is now equal to newNodePtr

Algorithm

newNodePtr is created containing the value newEntry. It is then inserted into the LinkedList. If the insertion position is 1, then newNodePtr is inserted before headPtr, otherwise it is inserted after prevPtr

Parameters

in	*newPosition*	holds the specified new position in the LinkedList of the Node
in	*newEntry*	holds the new data that will be constained in newNodePtr
out	*newEntry*	no changes made to newEntry, but its value is passed by reference

Returns

ableToInsert checks if newPosition is within the valid range

Note

: None

Implements ListInterface< ItemType >.

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

Function checks if LinkedList is empty.

Function returns 0 if LinkedList is not empty and 1 if LinkedKist is empty

Precondition

itemCount contains some value depending on if there are any Nodes in the LinkedList

Postcondition

itemCount is not changed

Algorithm

itemCount == 0 will return a 1 if this is true or a 0 if it is false

Parameters

in	*None*	
out	*None*	

Returns

itemCount

Note

: None

Implements ListInterface< ItemType >.

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

Function removes a Node from a LinkedList.

Function removes a Node from a LinkedList using a specified position

Precondition

int position is initialized to some specified value
Node<ItemType>* curPtr is initialized to NULL
Node<ItemType>* prevPtr is initalized to the Node at position - 1

Postcondition

newPosition is not changed
itemCount is decremented by one
if the position is not equal to one, prevPtr's next value is now curPtr's next Node, otherwise headPtr is now equal to its next Node

Algorithm

curPtr is set to NULL. If the position of the Node to be removed is 1 then curPtr takes the value of headPtr, and headPtr takes the value of its next node. Otherwise, curPtr takes the value of prevPtr's next, and prevPtr's next takes the value of curPtr's next. curPtr is then reset at the end of the algorithm, deleted, and pointing to NULL

Parameters

in	*position*	holds the specified position in the LinkedList of the Node
out	*None*	

Returns

ableToInsert checks if newPosition is within the valid range

Note

: None

Implements ListInterface< ItemType >.

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

Function finds a node at a given position and replaces that Node's data with another entry.

Function points to the head pointer and searches through the list until the specified position is reached and then replaces the data of the Node in that position with newEntry

Precondition

const ItemType& newEntry is initaialized to some specified value
int position is intialized to some specified value
nodePtr is initialized to the Node at position
replacedItem is initalized to nodePtr's item

Postcondition

none

Algorithm

Function uses remove() and insert() to replace the Node at position

Parameters

in	*position*	holds the specified position in the LinkedList of the Node
in	*newEntry*	holds the data that will be stored in the new Node
out	*None*	

Returns

replacedItem

Note

: None

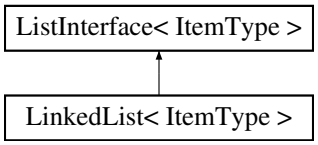
Implements ListInterface< ItemType >.

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

position	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 <= 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

newPosition	The list position at which to insert newEntry.
newEntry	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 <= position <= 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

position	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 <= position <= getLength().

Postcondition

The entry at the given position is newEntry.

Parameters

position	The list position of the entry to replace.
newEntry	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** (const ItemType &anItem)
- **Node** (const ItemType &anItem, Node< ItemType > *nextNodePtr)
- void **setItem** (const ItemType &anItem)
- void **setNext** (Node< ItemType > *nextNodePtr)
- ItemType **getItem** () const
- Node< ItemType > * **getNext** () const

Private Attributes

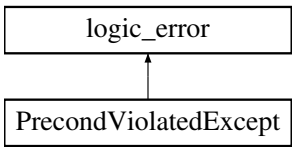
- ItemType **item**
- Node< ItemType > * **next**

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="")

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

- PrecondViolatedExcept.h
- PrecondViolatedExcept.cpp

5 File Documentation

5.1 LinkedList.cpp File Reference

Implementation file for LinkedList class.

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

5.1.1 Detailed Description

Implementation file for LinkedList class.

Author

Alex Kastanek

Implements all member methods of the LinkedList class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of LinkedList class

Note

Requires LinkedList.h
Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

5.2 LinkedList.h File Reference

Definition file for LinkedList class.

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

Classes

- class LinkedList< ItemType >

5.2.1 Detailed Description

Definition file for LinkedList class.

Author

Alex Kastanek

Specifies all member methods of the LinkedList class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of LinkedList class

Note

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

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 Node class.

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

5.4.1 Detailed Description

Implementation file for Node class.

Author

Alex Kastanek

Implements all member methods of the Node class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of Node class

Note

Requires Node.h
Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

5.5 Node.h File Reference

Definition file for Node class.

```
#include <iostream>
```

Classes

- class Node< ItemType >

5.5.1 Detailed Description

Definition file for Node class.

Author

Alex Kastanek

Specifies all member methods of the Node class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of Node class

Note

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

5.6 PA01.cpp File Reference

Driver program for LinkedList class.

```
#include "ListInterface.h"
#include "LinkedList.h"
#include "Node.h"
#include "PrecondViolatedExcept.h"
#include <iostream>
#include <string>
```

Functions

- int **main** ()

5.6.1 Detailed Description

Driver program for LinkedList class.

Author

Alex Kastanek

Allows LinkedList, Node, and PrecondViolatedExcept classes to compile together and interact

Version

1.00 C.S Student (14 September 2016) Iniitial development of LinkedList class

Note

Requires LinkedList.h, LinkedList.cpp, Node.h, Node.cpp, PrecondViolatedExcept.h, PrecondViolated-Except.cpp

5.7 PrecondViolatedExcept.cpp File Reference

Implementation file for PrecondViolatedExcept class.

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

5.7.1 Detailed Description

Implementation file for PrecondViolatedExcept class.

Author

Alex Kastanek

Implements all member methods of the PrecondViolatedExcept class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of PrecondViolatedExcept class

Note

Requires PrecondViolatedExcept.h
Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

5.8 PrecondViolatedExcept.h File Reference

Definition file for PrecondViolatedExcept class.

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

Classes

- class PrecondViolatedExcept

5.8.1 Detailed Description

Definition file for PrecondViolatedExcept class.

Author

Alex Kastanek

Specifies all member methods of the PrecondViolatedExcept class

Version

1.00 C.S. Student (14 September 2016) Initial development and testing of PrecondViolatedExcept class

Note

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 0217 Pearson Education, Hoboken, New Jersey.

Index