Doubly Linked List

Generated by Doxygen 1.8.20

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

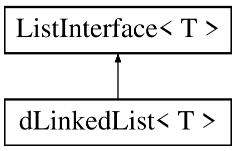
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 dLinkedList< T > Class Template Reference

Inheritance diagram for dLinkedList< T >:



Public Member Functions

- dLinkedList ()
- dLinkedList (const dLinkedList< T > &alist)
- virtual ~dLinkedList ()
- int getCurrentSize () const
- bool isEmpty () const
- bool add (const T &newEntry)
- bool remove (const T &anEntry)
- void clear ()
- int getFrequencyOf (const T &anEntry) const
- bool contains (const T &anEntry) const
- bool insertAt (const T &newEntry, const int index)
- bool removeAt (const int index)
- bool replace (const int index, const T &newEntry)
- int getIndex (const T &anEntry) const
- T getItem (const int index) const
- void printList () const
- void printReverseList () const
- bool addFront (const T &newEntry)
- bool addBack (const T &newEntry)
- dNode< T > * getHead () const
- dNode< T > * getTail () const

3.1.1 Constructor & Destructor Documentation

3.1.1.1 dLinkedList() [1/2]

```
template<class T >
dLinkedList< T >::dLinkedList ( )
```

Default contructor: creates a list with zero items

Returns

sets data fields appropriately

3.1.1.2 dLinkedList() [2/2]

```
template<class T >
dLinkedList< T >::dLinkedList (
            const dLinkedList< T > & alist )
```

Copy constructor: performs a deep copy of alist. If alist is empty, then the new list is also empty.

Returns

sets data fields appropriately.

3.1.1.3 ∼dLinkedList()

```
template<class T >
virtual dLinkedList< T >::∼dLinkedList ( )  [virtual]
```

Destructor: deallocates memory and sets all pointers to nullptr.

3.1.2 Member Function Documentation

3.1.2.1 add()

```
template<class T >
bool dLinkedList< T >::add (
            const T & newEntry )  [virtual]
```

Adds a new entry to the list.

Postcondition

If successful, newEntry is stored in the list and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if addition was successful, or false if not.

Implements ListInterface< T >.

3.1.2.2 addBack()

```
template<class T >
bool dLinkedList< T >::addBack (
             const T & newEntry )
```

Adds a new entry to the "back" of the list (as the last entry).

Postcondition

If successful, newEntry is stored as the last item of the list and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if addition was successful, or false if not.

3.1.2.3 addFront()

```
template<class T >
bool dLinkedList< T >::addFront (
             const T & newEntry )
```

3.1.2.4 Methods that are specific to dLinkedList

Adds a new entry to the "front" of the list (as the first entry).

Postcondition

If successful, newEntry is stored as the first item of the list and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if addition was successful, or false if not.

3.1.2.5 clear()

```
template<class T >
void dLinkedList< T >::clear ( ) [virtual]
```

Removes all entries from this list.

Postcondition

list contains no items, and the count of items is 0.

Implements ListInterface< T >.

3.1.2.6 contains()

```
template<class T >
bool dLinkedList< T >::contains (
             const T & anEntry ) const [virtual]
```

Tests whether this list contains a given entry.

Parameters

anEntry	The entry to locate.

Returns

True if list contains anEntry, or false otherwise.

Implements ListInterface< T >.

3.1.2.7 getCurrentSize()

```
template<class T >
int dLinkedList< T >::getCurrentSize ( ) const [virtual]
```

3.1.2.8 Methods that are specified in listInterface.hpp.

Gets the current number of entries in this list.

Returns

The integer number of entries currently in the list.

Implements ListInterface< T >.

3.1.2.9 getFrequencyOf()

```
template<class T >
int dLinkedList< T >::getFrequencyOf (
             const T & anEntry ) const [virtual]
```

Counts the number of times a given entry appears in list.

Parameters

anEntry	The entry to be counted.

Returns

The number of times anEntry appears in the list.

Implements ListInterface< T >.

3.1.2.10 getHead()

```
template<class T >
dNode<T>* dLinkedList< T >::getHead ( ) const
```

Gets the head pointer of the linked list list.

Returns

The head pointer of the list

3.1.2.11 getIndex()

```
template<class T >
int dLinkedList< T >::getIndex (
              const T & anEntry ) const  [virtual]
```

Returns the index of the first occurance of anEntry.

Parameters

anEntry	The object to find.

Returns

A valid index if the object is found, and -1 if it is not.

Implements ListInterface< T >.

3.1.2.12 getItem()

```
template<class T >
T dLinkedList< T >::getItem (
              const int index ) const  [virtual]
```

Returns the item at the position specified by index where index starts at 0.

Precondition

index must be between 0 and itemCount - 1.

Parameters

index	The index of the item to be returned.

Returns

A valid item if the index is between 0 and itemCount - 1. If somehow index $<$ 1 or index $>$ itemCount, print out the following error message to standard out (stdout): "Invalid index to getItem()" and return a blank default object.

Implements ListInterface< T >.

3.1.2.13 getTail()

```
template<class T >
dNode<T>* dLinkedList< T >::getTail ( ) const
```

Gets the tail pointer of the linked list list.

Returns

The tail pointer of the list

3.1.2.14 insertAt()

```
template<class T >
bool dLinkedList< T >::insertAt (
            const T & newEntry,
            const int index )  [virtual]
```

Inserts a new entry at the position specified by index, where index = 0 refers to the first entry in the list. If index + 1 $>$ itemCount, then the new entry is added to the back of the list.

Postcondition

If successful, newEntry is stored as the specified position of the list (or at the back of the list) and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.
index	The index of the list to insert the item at.

Returns

True if addition was successful, or false if not.

Implements ListInterface< T >.

3.1.2.15 isEmpty()

```
template<class T >
bool dLinkedList< T >::isEmpty ( ) const  [virtual]
```

Sees whether this list is empty.

Returns

True if the list is empty, or false if not.

Implements ListInterface< T >.

3.1.2.16 printList()

```
template<class T >
void dLinkedList< T >::printList ( ) const  [virtual]
```

Prints the content of the list to the screen.
Implements ListInterface< T >.

3.1.2.17 printReverseList()

```
template<class T >
void dLinkedList< T >::printReverseList ( ) const [virtual]
```

Prints the content of the list in reverse order to the screen.
Implements ListInterface< T >.

3.1.2.18 remove()

```
template<class T >
bool dLinkedList< T >::remove (
                const T & anEntry ) [virtual]
```

Removes one occurrence of a given entry from this list, if possible.

Postcondition

If successful, anEntry has been removed from the list and the count of items in the list has decreased by 1.

Parameters

anEntry	The entry to be removed.

Returns

True if removal was successful, or false if not.

Implements ListInterface< T >.

3.1.2.19 removeAt()

```
template<class T >
bool dLinkedList< T >::removeAt (
                const int index ) [virtual]
```

Removes the entry at the position specified by index, where index = 0 refers to the first entry in the list. If index + 1 > itemCount, then this method does nothing.

Postcondition

If successful, the item at the specified position is removed, unless the specified index does not exist in the list.

Parameters

index	The index of the list to remove the item at.

Returns

True if removal was successful, or false if not.

Implements ListInterface< T >.

3.1.2.20 replace()

```
template<class T >
```

```
bool dLinkedList< T >::replace (
            const int index,
            const T & newEntry )  [virtual]
```

Replaces the entry at the specified index with newEntry. Does nothing if the specified index does not exists, where index starts at 0.

Parameters

index	The index of the list at which to replace the item.
newEntry	The object to replace at the specified index.

Returns

True if replacement was successful, or false if not.

Implements ListInterface< T >.

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

- dLinkedList.hpp

3.2 dNode< T > Class Template Reference

Public Member Functions

- dNode ()
- dNode (const T &anItem)
- dNode (const T &anItem, dNode< T > ∗nextNodePtr, dNode< T > ∗prevNodePtr)
- void setItem (const T &anItem)
- void setNext (dNode< T > ∗nextNodePtr)
- void setPrev (dNode< T > ∗prevNodePtr)
- T getItem () const
- dNode< T > ∗ getNext () const
- dNode< T > ∗ getPrev () const
- void printNode () const

3.2.1 Constructor & Destructor Documentation

3.2.1.1 dNode() [1/3]

```
template<class T >
dNode< T >::dNode ( )
```

Default contructor: creates an empty node.

Returns

sets data fields appropriately.

3.2.1.2 dNode() [2/3]

```
template<class T >
dNode< T >::dNode (
            const T & anItem )
```

Parameterized contructor: creates a node with anItem as item.

Parameters

anItem	The object to be put in the "item" field of the node

Returns

sets data fields appropriately.

3.2.1.3 dNode() [3/3]

```
template<class T >
dNode< T >::dNode (
            const T & anItem,
            dNode< T > * nextNodePtr,
            dNode< T > * prevNodePtr )
```

Parameterized contructor: creates a node with anItem as item and initialized next and prev pointers.

Parameters

anItem	The object to be put in the "item" field of the node.
nextNodePtr	The pointer to the next node.
prevNodePtr	The pointer to the prev node.

Returns

sets data fields appropriately.

3.2.2 Member Function Documentation

3.2.2.1 getItem()

```
template<class T >
T dNode< T >::getItem ( ) const
```

Gets the item of the node.

Returns

gets the item field of the node.

3.2.2.2 getNext()

```
template<class T >
dNode<T>* dNode< T >::getNext ( ) const
```

Gets the item of the node.

Returns

gets the item field of the node.

3.2.2.3 getPrev()

```
template<class T >
dNode<T>* dNode< T >::getPrev ( ) const
```

Gets the item of the node.

Returns

gets the item field of the node.

3.2.2.4 printNode()

```
template<class T >
void dNode< T >::printNode ( ) const
```

Prints out node information.

Returns

displays all the fields of the node.

3.2.2.5 setItem()

```
template<class T >
void dNode< T >::setItem (
             const T & anItem )
```

Sets the item field of the node to anItem.

Parameters

anItem	The object to be set.

Returns

sets the item field appropriately.

3.2.2.6 setNext()

```
template<class T >
void dNode< T >::setNext (
             dNode< T > * nextNodePtr )
```

Sets the next field of the node to nextNodePtr.

Parameters

nextNodePtr	The pointer to be set.

Returns

sets the next field appropriately.

3.2.2.7 setPrev()

```
template<class T >
void dNode< T >::setPrev (
             dNode< T > * prevNodePtr )
```

Sets the prev field of the node to prevNodePtr.

Parameters

prevNodePtr	The pointer to be set.

Returns

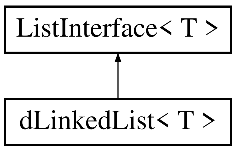
sets the prev field appropriately.

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

- dNode.hpp

3.3 ListInterface< T > Class Template Reference

Inheritance diagram for ListInterface< T >:



Public Member Functions

- virtual int getCurrentSize () const =0
- virtual bool isEmpty () const =0
- virtual bool add (const T &newEntry)=0
- virtual bool remove (const T &anEntry)=0
- virtual void clear ()=0
- virtual int getFrequencyOf (const T &anEntry) const =0
- virtual bool contains (const T &anEntry) const =0
- virtual bool insertAt (const T &newEntry, const int index)=0
- virtual bool removeAt (const int index)=0
- virtual bool replace (const int index, const T &newEntry)=0
- virtual int getIndex (const T &anEntry) const =0
- virtual T getItem (const int index) const =0
- virtual void printList () const =0
- virtual void printReverseList () const =0

3.3.1 Member Function Documentation

3.3.1.1 add()

```
template<class T >
virtual bool ListInterface< T >::add (
                const T & newEntry )  [pure virtual]
```

Adds a new entry to the list.

Postcondition

If successful, newEntry is stored in the list and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if addition was successful, or false if not.

Implemented in dLinkedList< T >.

3.3.1.2 clear()

```
template<class T >
virtual void ListInterface< T >::clear ( )  [pure virtual]
```

Removes all entries from this list.

Postcondition

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

Implemented in dLinkedList< T >.

3.3.1.3 contains()

```
template<class T >
virtual bool ListInterface< T >::contains (
             const T & anEntry ) const  [pure virtual]
```

Tests whether this list contains a given entry.

Parameters

anEntry	The entry to locate.

Returns

True if list contains anEntry, or false otherwise.

Implemented in dLinkedList< T >.

3.3.1.4 getCurrentSize()

```
template<class T >
virtual int ListInterface< T >::getCurrentSize ( ) const  [pure virtual]
```

Gets the current number of entries in this list.

Returns

The integer number of entries currently in the list.

Implemented in dLinkedList< T >.

3.3.1.5 getFrequencyOf()

```
template<class T >
virtual int ListInterface< T >::getFrequencyOf (
             const T & anEntry ) const  [pure virtual]
```

Counts the number of times a given entry appears in list.

Parameters

anEntry	The entry to be counted.

Returns

The number of times anEntry appears in the list.

Implemented in dLinkedList< T >.

3.3.1.6 getIndex()

```
template<class T >
virtual int ListInterface< T >::getIndex (
              const T & anEntry ) const  [pure virtual]
```

Returns the index of the first occurance of anEntry.

Parameters

anEntry	The object to find.

Returns

A valid index if the object is found, and -1 if it is not.

Implemented in dLinkedList< T >.

3.3.1.7 getItem()

```
template<class T >
virtual T ListInterface< T >::getItem (
              const int index ) const  [pure virtual]
```

Returns the item at the position specified by index where index starts at 0.

Precondition

index must be between 0 and itemCount - 1.

Parameters

index	The index of the item to be returned.

Returns

A valid item if the index is between 0 and itemCount - 1. If somehow index < 1 or index > itemCount, print out the following error message to standard out (stdout): "Invalid index to getItem()" and return a blank default object.

Implemented in dLinkedList< T >.

3.3.1.8 insertAt()

```
template<class T >
virtual bool ListInterface< T >::insertAt (
              const T & newEntry,
              const int index )  [pure virtual]
```

Inserts a new entry at the position specified by index, where index = 0 refers to the first entry in the list. If index + 1 > itemCount, then the new entry is added to the back of the list.

Postcondition

If successful, newEntry is stored as the specified position of the list (or at the back of the list) and the count of items in the list has increased by 1.

Parameters

newEntry	The object to be added as a new entry.
index	The index of the list to insert the item at.

Returns

True if addition was successful, or false if not.

Implemented in dLinkedList< T >.

3.3.1.9 isEmpty()

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

Sees whether this list is empty.

Returns

True if the list is empty, or false if not.

Implemented in dLinkedList< T >.

3.3.1.10 printList()

```
template<class T >
virtual void ListInterface< T >::printList ( ) const  [pure virtual]
```

Prints the content of the list to the screen.
Implemented in dLinkedList< T >.

3.3.1.11 printReverseList()

```
template<class T >
virtual void ListInterface< T >::printReverseList ( ) const  [pure virtual]
```

Prints the content of the list in reverse order to the screen.
Implemented in dLinkedList< T >.

3.3.1.12 remove()

```
template<class T >
virtual bool ListInterface< T >::remove (
             const T & anEntry )  [pure virtual]
```

Removes one occurrence of a given entry from this list, if possible.

Postcondition

If successful, anEntry has been removed from the list and the count of items in the list has decreased by 1.

Parameters

anEntry	The entry to be removed.

Returns

True if removal was successful, or false if not.

Implemented in dLinkedList< T >.

3.3.1.13 removeAt()

```
template<class T >
virtual bool ListInterface< T >::removeAt (
             const int index )  [pure virtual]
```

Removes the entry at the position specified by index, where index = 0 refers to the first entry in the list. If index + 1 > itemCount, then this method does nothing.

Postcondition

If successful, the item at the specified position is removed, unless the specified index does not exist in the list.

Parameters

index	The index of the list to remove the item at.

Returns

True if removal was successful, or false if not.

Implemented in dLinkedList< T >.

3.3.1.14 replace()

```
template<class T >
virtual bool ListInterface< T >::replace (
             const int index,
             const T & newEntry )  [pure virtual]
```

Replaces the entry at the specified index with newEntry. Does nothing if the specified index does not exists, where index starts at 0.

Parameters

index	The index of the list at which to replace the item.
newEntry	The object to replace at the specified index.

Returns

True if replacement was successful, or false if not.

Implemented in dLinkedList< T >.
The documentation for this class was generated from the following file:

- ListInterface.hpp

Index