

Assingment 1

1.0

Generated by Doxygen 1.8.13

Contents

1	CSCI331Project	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	Class Documentation	7
4.1	LinkedList< ItemType > Class Template Reference	7
4.2	ListInterface< ItemType > Class Template Reference	7
4.2.1	Detailed Description	7
4.2.2	Member Function Documentation	8
4.2.2.1	clear()	8
4.2.2.2	deletion()	8
4.2.2.3	getEntry()	9
4.2.2.4	getLength()	9
4.2.2.5	insert()	10
4.2.2.6	isEmpty()	11
4.2.2.7	replace()	11
4.3	Node< ItemType > Class Template Reference	12
4.3.1	Detailed Description	12
4.3.2	Constructor & Destructor Documentation	12
4.3.2.1	Node() [1/3]	13
4.3.2.2	Node() [2/3]	13
4.3.2.3	Node() [3/3]	13
4.3.3	Member Function Documentation	14
4.3.3.1	getItem()	14
4.3.3.2	getNext()	14
4.3.3.3	setItem()	14
4.3.3.4	setNext()	15
4.4	SecKeySS Class Reference	15
4.4.1	Detailed Description	16
4.5	SSClass Class Reference	16
4.5.1	Detailed Description	16

[Index](#)

17

Chapter 1

CSCI331Project

Github for the CSCI 331 Sequence Set Class Group Programming Project

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

ListInterface< ItemType >	7
LinkedList< ItemType >	7
ListInterface< int >	7
LinkedList< int >	7
ListInterface< SecKeySS >	7
LinkedList< SecKeySS >	7
Node< ItemType >	12
Node< int >	12
Node< SecKeySS >	12
SecKeySS	15
SSClass	16

Chapter 3

Class Index

3.1 Class List

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

LinkedList< ItemType >	
This is LinkedList class creating a list of linked nodes	7
ListInterface< ItemType >	7
Node< ItemType >	
This is Node class for linked list	12
SecKeySS	15
SSClass	16

Chapter 4

Class Documentation

4.1 `LinkedList< ItemType >` Class Template Reference

This is `LinkedList` class creating a list of linked nodes.

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

Inheritance diagram for `LinkedList< ItemType >`:

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 int `getItemCount` () const =0
- virtual bool `insert` (int newPosition, const ItemType &newEntry)=0
- virtual bool `deletion` (int position)=0
- virtual void `clear` ()=0
- virtual ItemType `getEntry` (int position) const =0
- virtual void `replace` (int position, const ItemType &newEntry)=0
- virtual ItemType `displayList` ()=0

4.2.1 Detailed Description

```
template<class ItemType>  
class ListInterface< ItemType >
```

Definition at line 7 of file ListInterface.h.

4.2.2 Member Function Documentation

4.2.2.1 clear()

```
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 >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.2 deletion()

```
template<class ItemType>
virtual bool ListInterface< ItemType >::deletion (
    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 >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.3 getEntry()

```
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 >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.4 getLength()

```
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 >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.5 insert()

```
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 newEntry.
<i>newEntry</i>	The entry to insert into the list.

Returns

True if insertion is successful, or false if not.

Implemented in [LinkedList< ItemType >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.6 isEmpty()

```
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 >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

4.2.2.7 replace()

```
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

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

Implemented in [LinkedList< ItemType >](#), [LinkedList< int >](#), and [LinkedList< SecKeySS >](#).

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

- ListInterface.h

4.3 Node< ItemType > Class Template Reference

This is [Node](#) class for linked list.

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

Public Member Functions

- [Node](#) ()
Node default constructor.
- [Node](#) (const ItemType &anItem)
Node constructor.
- [Node](#) (const ItemType &anItem, [Node](#)< ItemType > *nextNodePtr)
Node constructor.
- void [setItem](#) (const ItemType &anItem)
Member function taking one argument to set the memebr item.
- void [setNext](#) ([Node](#)< ItemType > *nextNodePtr)
Member function taking one argument, a pointer to a Node.
- ItemType [getItem](#) () const
Member function returning an item.
- [Node](#)< ItemType > * [getNext](#) () const
Memebr funtion to get the pointer to the next Node.

4.3.1 Detailed Description

```
template<class ItemType>
class Node< ItemType >
```

This is [Node](#) class for linked list.

This class is to create a node that is used in linked list class. The [Node](#) will store a template ItemType, item and a [Node](#) pointer of item type, next.

Definition at line 12 of file Node.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Node() [1/3]

```
template<class ItemType >
Node< ItemType >::Node ( )
```

Node default constructor.

Default constructor assigning next as NULLPTR

Definition at line 8 of file Node.cpp.

```
8             : next (nullptr)
9 {
10 } // end default constructor
```

4.3.2.2 Node() [2/3]

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

Node constructor.

Taking one argument to assign to item and assigns next to null pointer.

Parameters

<i>anItem</i>	a constant reference to an item of itemtype
---------------	---

Definition at line 18 of file Node.cpp.

```
18             : item(anItem), next (nullptr)
19 {
20 } // end constructor
```

4.3.2.3 Node() [3/3]

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

Node constructor.

Taking two arguments. The first to assign to item and the other assigns next to argument.

Parameters

<i>anItem</i>	a constant reference to an item of itemType
<i>nextNodePtr</i>	a pointer to the next node

Definition at line 30 of file Node.cpp.

```

30                                     :
31     item(anItem), next(nextNodePtr)
32 {
33 } // end constructor

```

4.3.3 Member Function Documentation**4.3.3.1 getItem()**

```

template<class ItemType >
ItemType Node< ItemType >::getItem ( ) const

```

Member function returning an item.

/return the item of itemType

Definition at line 60 of file Node.cpp.

```

61 {
62     return item;
63 } // end getItem

```

4.3.3.2 getNext()

```

template<class ItemType >
Node< ItemType > * Node< ItemType >::getNext ( ) const

```

Member function to get the pointer to the next Node.

/return a pointer to the next node.

Definition at line 70 of file Node.cpp.

```

71 {
72     return next;
73 } // end getNext

```

4.3.3.3 setItem()

```

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

```

Member function taking one argument to set the member item.

Parameters

<i>anItem</i>	to be reference to by item
---------------	----------------------------

Definition at line 40 of file Node.cpp.

```
41 {
42     item = anItem;
43 } // end setItem
```

4.3.3.4 setNext()

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

Member function taking one argument, a pointer to a [Node](#).

/param nextNodePtr a point to a [Node](#), the next [Node](#) in a linked list

Definition at line 50 of file Node.cpp.

```
51 {
52     next = nextNodePtr;
53 } // end setNext
```

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

- Node.h
- Node.cpp

4.4 SecKeySS Class Reference

Public Member Functions

- **SecKeySS** (const [SecKeySS](#) &s)
- string **getData** () const
- [LinkedList](#)< int > **getDuplicates** () const
- void **setData** (const string s)
- void **setDuplicates** ([LinkedList](#)< int > dup)
- bool **operator**< (const string &s) const
- bool **operator**< (const [SecKeySS](#) &s) const
- bool **operator**> (const string &s) const
- bool **operator**> (const [SecKeySS](#) &s) const
- bool **operator**== (const string &s) const
- bool **operator**== (const [SecKeySS](#) &s) const
- void **operator**= (const [SecKeySS](#) &s)

4.4.1 Detailed Description

Definition at line 9 of file SecKeySS.h.

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

- SecKeySS.h

4.5 SSClass Class Reference

Public Member Functions

- **SSClass** (const [SSClass](#) &ss)
- bool **isEmpty** ()
- bool **openFile** (string input)
- void **insert** (string s)
- vector< int > **search** (string s, unsigned fieldNum)
- int **directionalSearch** (string state, char direction)
- string **returnLine** (int rrn)

4.5.1 Detailed Description

Definition at line 52 of file SSClass.h.

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

- SSClass.h

Index

- clear
 - ListInterface, [8](#)
- deletion
 - ListInterface, [8](#)
- getEntry
 - ListInterface, [8](#)
- getItem
 - Node, [14](#)
- getLength
 - ListInterface, [9](#)
- getNext
 - Node, [14](#)
- insert
 - ListInterface, [9](#)
- isEmpty
 - ListInterface, [11](#)
- LinkedList< ItemType >, [7](#)
- ListInterface
 - clear, [8](#)
 - deletion, [8](#)
 - getEntry, [8](#)
 - getLength, [9](#)
 - insert, [9](#)
 - isEmpty, [11](#)
 - replace, [11](#)
- ListInterface< ItemType >, [7](#)
- Node
 - getItem, [14](#)
 - getNext, [14](#)
 - Node, [12](#), [13](#)
 - setItem, [14](#)
 - setNext, [15](#)
- Node< ItemType >, [12](#)
- replace
 - ListInterface, [11](#)
- SSClass, [16](#)
- SecKeySS, [15](#)
- setItem
 - Node, [14](#)
- setNext
 - Node, [15](#)