# My Project

Generated by Doxygen 1.8.13

# Contents

# 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:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

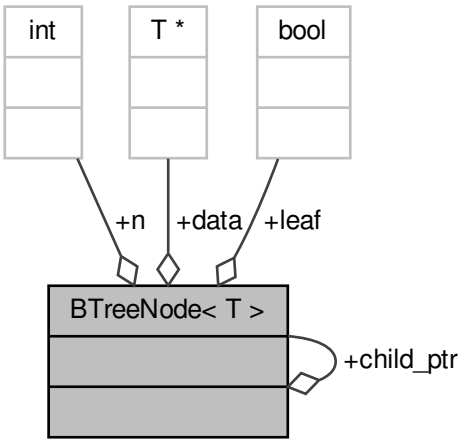
# Chapter 5

# Class Documentation

## 5.1 BTreeNode< T > Struct Template Reference

`#include <BTree.h>`

Collaboration diagram for BTreeNode< T >:




### Public Attributes

- T * data
- BTreeNode ** child_ptr
- bool leaf
- int n

### 5.1.1 Detailed Description

**template<typename T>**
**struct BTreeNode< T >**

Definition at line 11 of file BTree.h.

### 5.1.2 Member Data Documentation

#### 5.1.2.1 child_ptr

```
template<typename T >
BTreeNode** BTreeNode< T >::child_ptr
```

Definition at line 14 of file BTree.h.

#### 5.1.2.2 data

```
template<typename T >
T* BTreeNode< T >::data
```

Definition at line 13 of file BTree.h.

#### 5.1.2.3 leaf

```
template<typename T >
bool BTreeNode< T >::leaf
```

Definition at line 15 of file BTree.h.

#### 5.1.2.4 n

```
template<typename T >
int BTreeNode< T >::n
```

Definition at line 16 of file BTree.h.

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

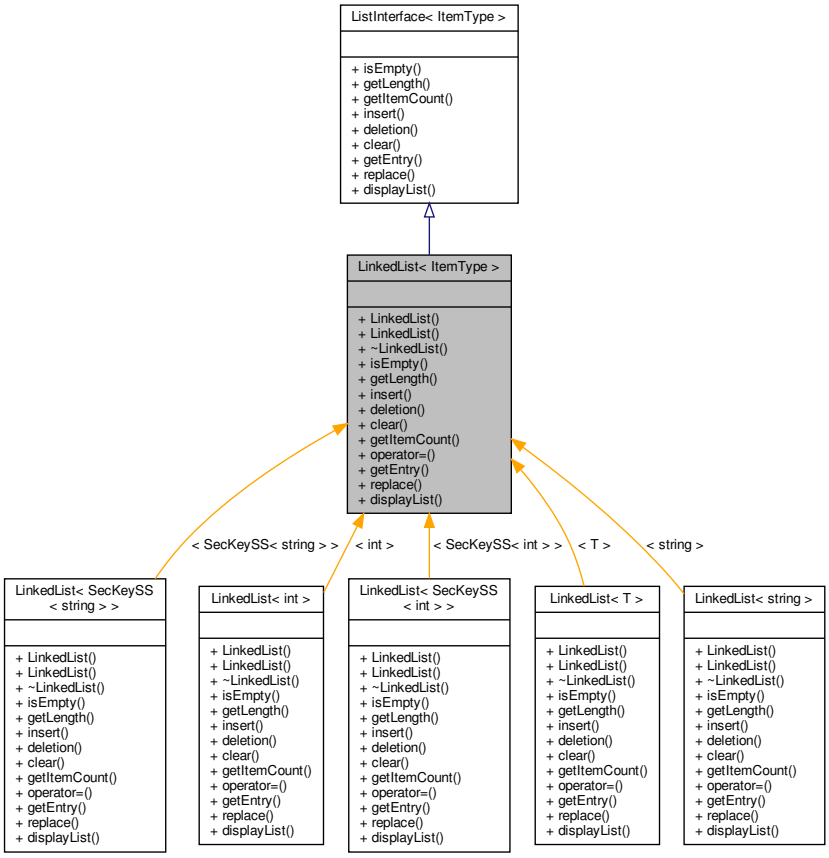
- BTree.h

## 5.2 LinkedList< ItemType > Class Template Reference

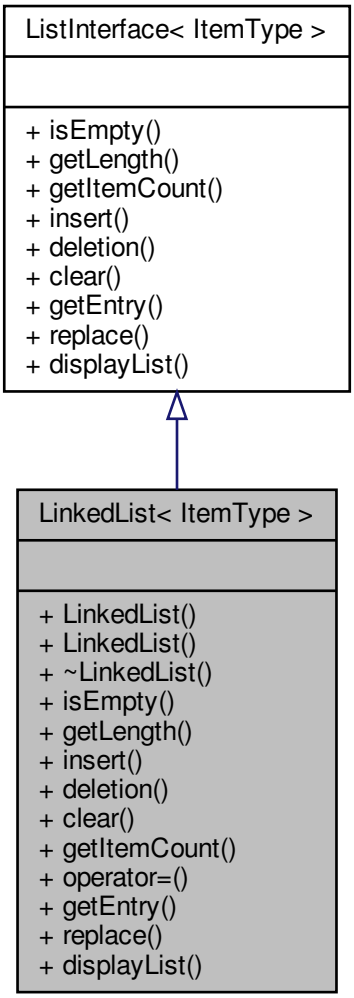
This is LinkedList class creating a list of linked nodes.

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

Inheritance diagram for LinkedList< ItemType >:



Collaboration diagram for LinkedList< ItemType >:




## Public Member Functions

- LinkedList ()

  *LinkedList default constructor.*
- LinkedList (const LinkedList< ItemType > &aList)

  *LinkedList constructor.*
- virtual ~LinkedList ()

  *LinkedList deconstructor.*
- bool isEmpty () const

  *Memebr function to check if a LinkedList is empty.*
- int getLength () const

  *Member function to get the length of the LinkedList.*
- bool insert (int newPosition, const ItemType &newEntry)

  *Memebr function to insert a new item into a Node of a LinkedList.*

- bool deletion (int position)

  *Member function for deletion of a Node.*
- void clear ()

  *Memebr Fucntion to clear a LinkedList.*
- int getItemCount () const

  *Member function to get the item count.*
- LinkedList< ItemType > & operator= (const LinkedList< ItemType > &rhs)

  *operator function =*
- ItemType getEntry (int position) const

  *Memebr function to get (return) an entry at a position.*
- void replace (int position, const ItemType &newEntry)

  *Member function to replace an item at a position.*
- ItemType displayList ()

  *Member function to display the list.*

### 5.2.1 Detailed Description

**template<class ItemType>**
**class LinkedList< ItemType >**

This is LinkedList class creating a list of linked nodes.

This class is to create a linked list of nodes. The nodes are of type template ItemType, item and a Node pointer of item type, next.

Definition at line 19 of file LinkedList.h.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 LinkedList() [1/2]

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

LinkedList default constructor.

Sets headptr to null and itemCount to 0.

Definition at line 18 of file LinkedList.cpp.

#### 5.2.2.2 LinkedList() [2/2]

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

LinkedList constructor.

A copy constructor with one argumet passed, aList.

**Parameters**

| | |
|---|---|
| *aLsit* | a refrence to a list |

Definition at line 28 of file LinkedList.cpp.

#### 5.2.2.3 ~LinkedList()

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

LinkedList deconstructor.

A deconstructor to clear a LinkedList

Definition at line 70 of file LinkedList.cpp.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 clear()

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

Memebr Fucntion to clear a LinkedList.

Removes 1 Node at a time while the LinkedList is not Empty

Implements ListInterface< ItemType >.

Definition at line 185 of file LinkedList.cpp.

#### 5.2.3.2 deletion()

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

Member function for deletion of a Node.

**Parameters**

| | |
|---|---|
| *position* | the position of te Node to be removed |

**Returns**

ableToRemove returns true if the Node is a valid Node.

**Precondition**

To be a valid Node to remove, psition >= 1 and position <= itemCount

Implements ListInterface< ItemType >.

Definition at line 151 of file LinkedList.cpp.

#### 5.2.3.3 displayList()

```
template<class ItemType >
ItemType LinkedList< ItemType >::displayList ( )  [virtual]
```

Member function to display the list.

Displays the list by returing one Node item at a time

**Returns**

nodePtr->getItem() an item at a node

Implements ListInterface< ItemType >.

Definition at line 274 of file LinkedList.cpp.

#### 5.2.3.4 getEntry()

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

Memebr function to get (return) an entry at a position.

**Exceptions**

| | |
|---|---|
| *PrecondViolatedExcep* | if position < 1 or position > getLength(). |

**Parameters**

| | |
|---|---|
| *position* | the position of a Node to return anItem |

**Returns**

nodePtr->getItem() an item at the position, position.

**Precondition**

position > 0 and position <= itemCount

Implements ListInterface< ItemType >.

Definition at line 198 of file LinkedList.cpp.

#### 5.2.3.5 getItemCount()

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

Member function to get the item count.

/return itemCount the count of items in the LinkedList

Implements ListInterface< ItemType >.

Definition at line 263 of file LinkedList.cpp.

#### 5.2.3.6 getLength()

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

Member function to get the length of the LinkedList.

**Returns**

itemCount the length (count of items) of the LinkedList

Implements ListInterface< ItemType >.

Definition at line 91 of file LinkedList.cpp.

#### 5.2.3.7 insert()

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

Memebr function to insert a new item into a Node of a LinkedList.

**Parameters**

| | |
|---|---|
| *newPosition* | a node position to insert a item into |
| *newEntry* | a reference to an item of itemType to be inserted into the Node. |

**Returns**

ableToInsert if newEntry can be inserted into the Node at newPosition

**Precondition**

newPosition >= 1
newPosition <= itemCount + 1

Implements ListInterface< ItemType >.

Definition at line 106 of file LinkedList.cpp.

#### 5.2.3.8 isEmpty()

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

Memebr function to check if a LinkedList is empty.

Checks and returns a boolean value if the list is true or not

**Returns**

itemCount == 0 returns 1 if the LinkedList is empty, 0 otherwise.

Implements ListInterface< ItemType >.

Definition at line 81 of file LinkedList.cpp.

#### 5.2.3.9 operator=()

```
template<class ItemType>
LinkedList< ItemType > & LinkedList< ItemType >::operator= (
             const LinkedList< ItemType > & rhs )
```

operator function =

**Parameters**

| | |
|---|---|
| *rhs* | referance to a LinkedList |

**Returns**

*this a pointer to the LinkedList

Definition at line 289 of file LinkedList.cpp.

#### 5.2.3.10 replace()

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

Member function to replace an item at a position.

**Exceptions**

| | |
|---|---|
| *PrecondViolatedExcep* | if position < 1 or position > getLength(). |

**Parameters**

| | |
|---|---|
| *position* | the position of the Node whos item will be replaced |
| *newEntry* | the new entery to replace the old entry of a Node |

Implements ListInterface< ItemType >.

Definition at line 219 of file LinkedList.cpp.

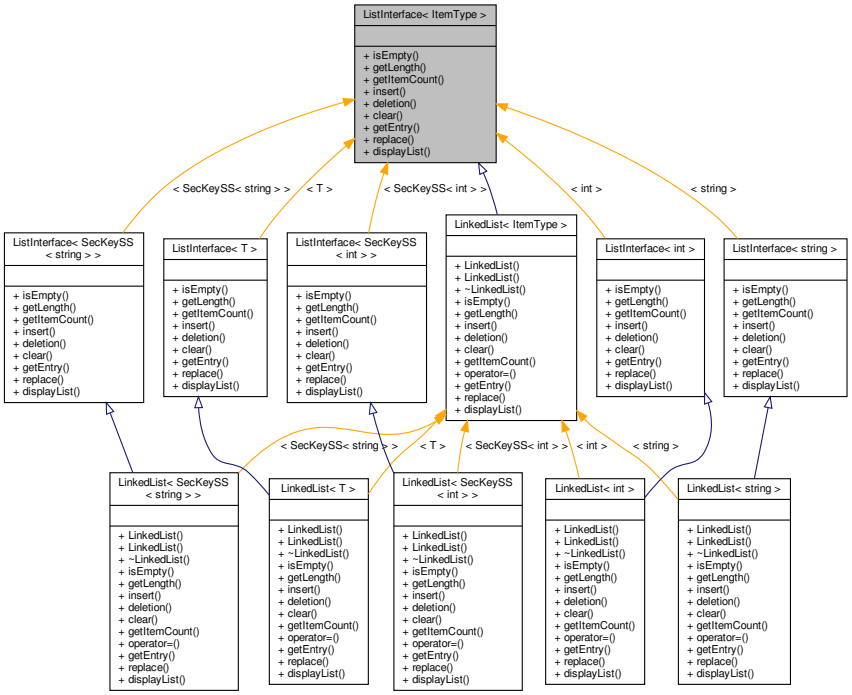
The documentation for this class was generated from the following files:

- LinkedList.h
- LinkedList.cpp

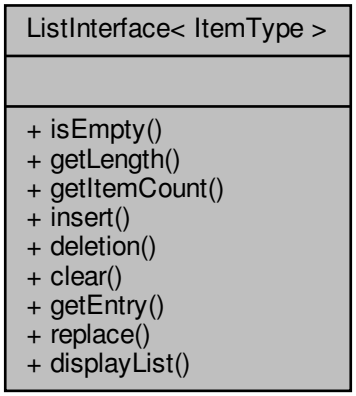
## 5.3 ListInterface< ItemType > Class Template Reference

```
#include <ListInterface.h>
```

Inheritance diagram for ListInterface< ItemType >:




Collaboration 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

### 5.3.1 Detailed Description

**template<class ItemType>**
**class ListInterface< ItemType >**

Definition at line 7 of file ListInterface.h.

### 5.3.2 Member Function Documentation

#### 5.3.2.1 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< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.2 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 <= 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 >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.3 displayList()

```
template<class ItemType>
virtual ItemType ListInterface< ItemType >::displayList ( )  [pure virtual]
```

Implemented in LinkedList< ItemType >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.4 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**

| | |
|---|---|
| *position* | The list position of the desired entry. |

**Returns**

The entry at the given position.

Implemented in LinkedList< ItemType >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.5 getItemCount()

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

Implemented in LinkedList< ItemType >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.6 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< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.7 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 <= 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 >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.8 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< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

#### 5.3.2.9 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**

| | |
|---|---|
| *position* | The list position of the entry to replace. |
| *newEntry* | The replacement entry. |

Implemented in LinkedList< ItemType >, LinkedList< SecKeySS< string > >, LinkedList< int >, LinkedList< SecKeySS< int > >, LinkedList< T >, and LinkedList< string >.

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

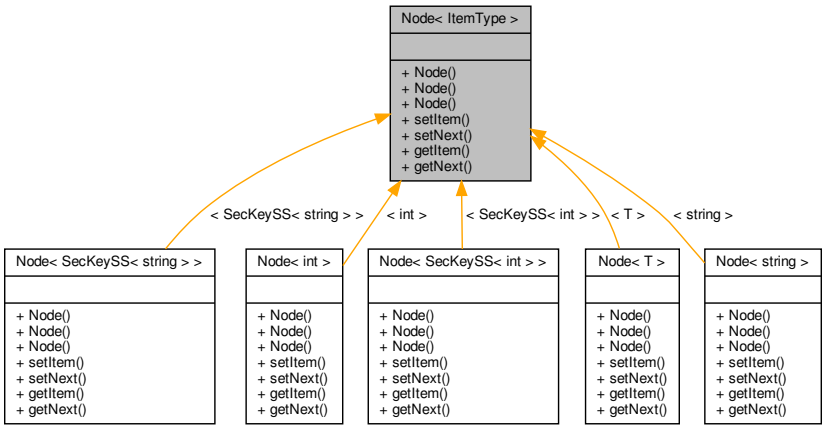
- ListInterface.h

## 5.4 Node< ItemType > Class Template Reference

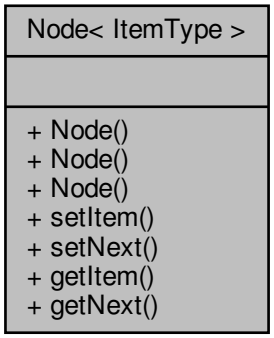
This is Node class for linked list.

`#include "Node.h"`

Inheritance diagram for Node< ItemType >:




Collaboration diagram for Node< ItemType >:



### 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.*

## 5.4.1 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.

## 5.4.2 Constructor & Destructor Documentation

### 5.4.2.1 Node() [1/3]

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

Node default constructor.

Default constructor assiging next as NULLPTR

Definition at line 8 of file Node.cpp.

### 5.4.2.2 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**

| | |
|---|---|
| *anItem* | a constant reference to an item of itemtype |

Definition at line 18 of file Node.cpp.

#### 5.4.2.3 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**

| | |
|---|---|
| *anItem* | a constant reference to an item of itemtype |
| *nextNodePtr* | a pointer to the next node |

Definition at line 30 of file Node.cpp.

### 5.4.3 Member Function Documentation

#### 5.4.3.1 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.

#### 5.4.3.2 getNext()

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

Memebr funtion to get the pointer to the next Node.

/return a pointer to the next node.

Definition at line 70 of file Node.cpp.

#### 5.4.3.3 setItem()

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

Member function taking one argument to set the memebr item.

**Parameters**

| | |
|---|---|
| *anItem* | to be reference to by item |

Definition at line 40 of file Node.cpp.

#### 5.4.3.4 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.

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

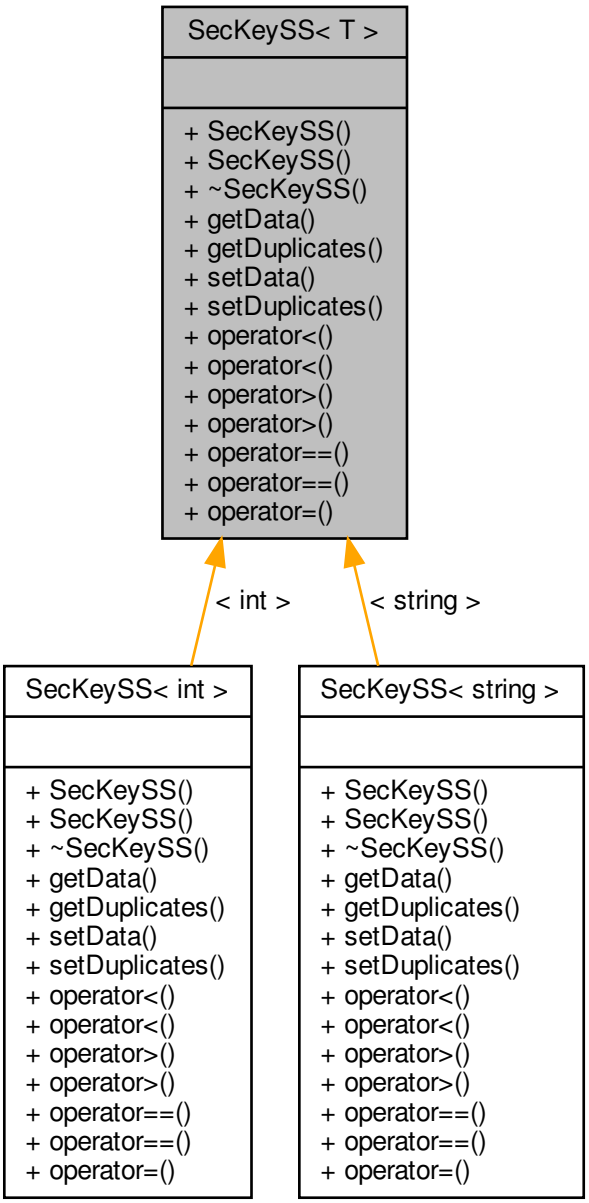
- Node.h
- Node.cpp

## 5.5 SecKeySS< T > Class Template Reference

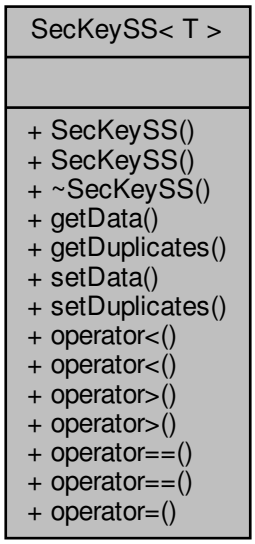
This is the class for Section Keys of the SS class.

```
#include "SecKeySS.h"
```

Inheritance diagram for SecKeySS< T >:



Collaboration diagram for SecKeySS< T >:




## Public Member Functions

- SecKeySS ()
- SecKeySS (SecKeySS< T > &s)
- ~SecKeySS ()
- T getData () const
- LinkedList< T > getDuplicates ()
- void setData (const T s)
- void setDuplicates (LinkedList< T > dup)
- bool operator< (const T &s) const
- bool operator< (const SecKeySS< T > &s) const
- bool operator> (const T &s) const
- bool operator> (const SecKeySS< T > &s) const
- bool operator== (const T &s) const
- bool operator== (const SecKeySS< T > &s) const
- void operator= (const SecKeySS< T > &s)

### 5.5.1 Detailed Description

**template<typename T>**
**class SecKeySS< T >**

This is the class for Section Keys of the SS class.

Definition at line 14 of file SecKeySS.h.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 SecKeySS() [1/2]

```
template<typename T>
SecKeySS< T >::SecKeySS ( )  [inline]
```

Default constructor

Definition at line 22 of file SecKeySS.h.

#### 5.5.2.2 SecKeySS() [2/2]

```
template<typename T>
SecKeySS< T >::SecKeySS (
            SecKeySS< T > & s )
```

Copy Constructor

Definition at line 109 of file SecKeySS.h.

#### 5.5.2.3 ~SecKeySS()

```
template<typename T >
SecKeySS< T >::~SecKeySS ( )
```

Deconstuctor

Definition at line 111 of file SecKeySS.h.

### 5.5.3 Member Function Documentation

#### 5.5.3.1 getData()

```
template<typename T>
T SecKeySS< T >::getData ( ) const  [inline]
```

Gets data

**Returns**

data the data to be returned

Definition at line 36 of file SecKeySS.h.

#### 5.5.3.2 getDuplicates()

```
template<typename T >
LinkedList< T > SecKeySS< T >::getDuplicates ( )
```

Gets duplicates

**Returns**

LinkedList of itemType

Definition at line 130 of file SecKeySS.h.

#### 5.5.3.3 operator<() [1/2]

```
template<typename T>
bool SecKeySS< T >::operator< (
            const T & s ) const  [inline]
```

Operator less than

**Parameters**

| | |
|---|---|
| *s* | a reference to a string to check if than |

**Returns**

true is data < s

Definition at line 61 of file SecKeySS.h.

#### 5.5.3.4 operator<() [2/2]

```
template<typename T>
bool SecKeySS< T >::operator< (
            const SecKeySS< T > & s ) const  [inline]
```

Operator less than to check Sec key

**Parameters**

| | |
|---|---|
| *s* | a string to check if than |

**Returns**

true is data < s.data

Definition at line 69 of file SecKeySS.h.

#### 5.5.3.5 operator=()

```
template<typename T>
void SecKeySS< T >::operator= (
            const SecKeySS< T > & s )
```

Operator equal for copy constructor

**Parameters**

| | |
|---|---|
| *s* | a reference to a SecKeySS |

Definition at line 125 of file SecKeySS.h.

#### 5.5.3.6 operator==() [1/2]

```
template<typename T>
bool SecKeySS< T >::operator== (
            const T & s ) const  [inline]
```

Operator is equal

**Parameters**

| | |
|---|---|
| *s* | a reference to a string |

**Returns**

true if data is equal to s

Definition at line 92 of file SecKeySS.h.

#### 5.5.3.7 operator==() [2/2]

```
template<typename T>
bool SecKeySS< T >::operator== (
            const SecKeySS< T > & s ) const  [inline]
```

Operator is equal

**Parameters**

| | |
|---|---|
| *s* | a reference to a secKeySS |

**Returns**

true if data is equal to s.data

Definition at line 100 of file SecKeySS.h.

#### 5.5.3.8 operator>() [1/2]

```
template<typename T>
bool SecKeySS< T >::operator> (
            const T & s ) const  [inline]
```

Operator geater than

**Parameters**

| | |
|---|---|
| *s* | a reference to a string to check if > than |

**Returns**

true is data > s

Definition at line 77 of file SecKeySS.h.

#### 5.5.3.9 operator>() [2/2]

```
template<typename T>
bool SecKeySS< T >::operator> (
            const SecKeySS< T > & s ) const  [inline]
```

Operator greater than to check a Sec key

**Parameters**

| | |
|---|---|
| *s* | a string to check if greater than |

**Returns**

true is data > s.data

Definition at line 85 of file SecKeySS.h.

#### 5.5.3.10 setData()

```
template<typename T>
void SecKeySS< T >::setData (
            const T s )  [inline]
```

Sets the data equal to argument 1

**Parameters**

| | |
|---|---|
| *s* | a string to set data to |

Definition at line 48 of file SecKeySS.h.

#### 5.5.3.11 setDuplicates()

```
template<typename T>
void SecKeySS< T >::setDuplicates (
            LinkedList< T > dup )
```

Sets duplicates

**Parameters**

| | |
|---|---|
| *LinkedList* | dup |

Definition at line 139 of file SecKeySS.h.

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

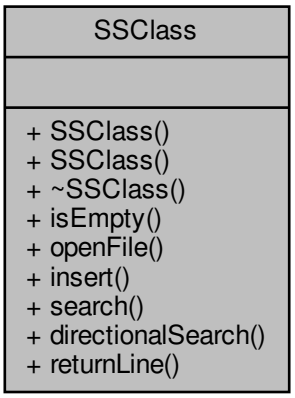
- SecKeySS.h

## 5.6 SSClass Class Reference

LinkedList integration for blocks, records, and fields.

```
#include "SSClass.h"
```

Collaboration diagram for SSClass:




### Public Member Functions

- SSClass ()

  *Default constructor.*
- SSClass (const SSClass &ss)

  *Constructor.*
- ∼SSClass ()

  *Deconstructor.*
- bool isEmpty ()

  *Check if numRecords is 0.*
- bool openFile (string input)

  *Opens external file.*
- void insert (string s)

  *inserts line by line into data*
- vector< int > search (string s, unsigned fieldNum)

  *Searches for record.*
- int directionalSearch (string state, char direction)

  *Searches directionly (N, S, W, E)*
- string returnLine (int rrn)

  *Fills secondary key vector.*

## 5.6.1 Detailed Description

LinkedList integration for blocks, records, and fields.

**Authors**

Jordan Bremer, Melvin Schmid, ..., ..., ...

Sequence Set class: – allows for insert and deletion of linked list – populates secondary keys – allows for searching of said linked list – ability to return city, state, county, lattitude, longitude, zip, and lower and upper indicies – ability to input a txt file and populate it's contents

Implementation and assumptions: – size defaults are listed towards the top of the program – array/vector elements are initialized to zero

Definition at line 65 of file SSClass.h.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 SSClass() [1/2]

```
SSClass::SSClass ( )
```

Default constructor.

Definition at line 39 of file SSClass.cpp.

#### 5.6.2.2 SSClass() [2/2]

```
SSClass::SSClass (
            const SSClass & ss )
```

Constructor.

Definition at line 43 of file SSClass.cpp.

#### 5.6.2.3 ~SSClass()

```
SSClass::~SSClass ( )
```

Deconstructor.

Definition at line 55 of file SSClass.cpp.

### 5.6.3 Member Function Documentation

#### 5.6.3.1 directionalSearch()

```
int SSClass::directionalSearch (
            string state,
            char direction )
```

Searches directionly (N, S, W, E)

**Parameters**

| | |
|---|---|
| *state* | the state to search "MN" for example |
| *direction* | (N, S, W, E) |

**Returns**

the line contating the soght after direction

Definition at line 179 of file SSClass.cpp.

#### 5.6.3.2 insert()

```
void SSClass::insert (
            string s )
```

inserts line by line into data

**Parameters**

| | |
|---|---|
| *s* | a string to insert |

Insertion of records into both the index file as well as the linkedlist of linkedlists /param s string to be inserted

Definition at line 70 of file SSClass.cpp.

#### 5.6.3.3 isEmpty()

```
bool SSClass::isEmpty ( )  [inline]
```

Check if numRecords is 0.

**Returns**

returns false if empty, otherwise returns true

Definition at line 207 of file SSClass.h.

#### 5.6.3.4 openFile()

```
bool SSClass::openFile (
            string input )
```

Opens external file.

**Parameters**

| | |
|---|---|
| *input* | string |

**Precondition**

data file

**Returns**

true if file location exists, otherwise returns false

Definition at line 10 of file SSClass.cpp.

#### 5.6.3.5 returnLine()

```
string SSClass::returnLine (
            int rrn )
```

Fills secondary key vector.

**Parameters**

| | |
|---|---|
| *rrn* | and integer refring to the line to get |

**Returns**

string containging the contents of the line

Definition at line 93 of file SSClass.cpp.

#### 5.6.3.6 search()

```
vector< int > SSClass::search (
            string s,
            unsigned fieldNum )
```

Searches for record.

**Parameters**

| | |
|---|---|
| *s* | strign to search for fieldNum the field in whitch to search |

**Returns**

vector of results

Definition at line 101 of file SSClass.cpp.

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

- SSClass.h
- SSClass.cpp

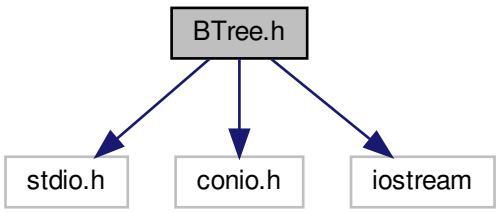
# Chapter 6

# File Documentation

## 6.1 BTree.h File Reference

```
#include <stdio.h>
#include <conio.h>
#include <iostream>
```

Include dependency graph for BTree.h:




### Classes

- struct BTreeNode< T >

### Functions

- template<typename T >
  BTreeNode * init ()
- template<typename T >
  void traverse (BTreeNode *p)
- template<typename T >
  void sort (int *p, int n)
- template<typename T >
  T split_child (BTreeNode *x, int i)
- template<typename T >
  void insert (T a)

### Variables

- struct BTreeNode * root = NULL
- struct BTreeNode * np = NULL
- struct BTreeNode * x = NULL

## 6.1.1 Function Documentation

#### 6.1.1.1 init()

```
template<typename T >
BTreeNode* init ( )
```

Definition at line 19 of file BTree.h.

#### 6.1.1.2 insert()

```
template<typename T >
void insert (
            T a )
```

Definition at line 122 of file BTree.h.

#### 6.1.1.3 sort()

```
template<typename T >
void sort (
            int * p,
            int n )
```

Definition at line 52 of file BTree.h.

#### 6.1.1.4 split_child()

```
template<typename T >
T split_child (
            BTreeNode * x,
            int i )
```

Definition at line 70 of file BTree.h.

#### 6.1.1.5 traverse()

```
template<typename T >
void traverse (
            BTreeNode * p )
```

Definition at line 33 of file BTree.h.

### 6.1.2 Variable Documentation

#### 6.1.2.1 np

```
struct BTreeNode * np = NULL
```

#### 6.1.2.2 root

```
struct BTreeNode* root = NULL
```

#### 6.1.2.3 x

```
struct BTreeNode * x = NULL
```

## 6.2 BTree.h

```
00001 #ifndef BTREE
00002 #define BTREE
00003
00004
00005 #include<stdio.h>
00006 #include<conio.h>
00007 #include<iostream>
00008 using namespace std;
00009
00010 template <typename T>
00011 struct BTreeNode
00012 {
00013     T *data;
00014     BTreeNode** child_ptr;
00015     bool leaf;
00016     int n;
00017 }*root = NULL, * np = NULL, * x = NULL;
00018 template <typename T>
00019 BTreeNode* init()
00020 {
00021     np = new BTreeNode;
00022     np->data = new T[5];
00023     np->child_ptr = new BTreeNode * [6];
00024     np->leaf = true;
00025     np->n = 0;
00026     for (int i = 0; i < 6; i++)
00027     {
```

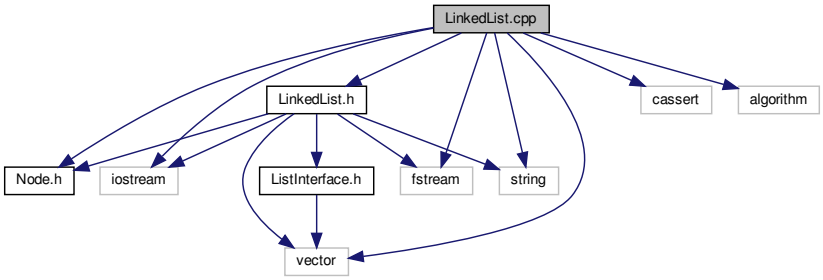
```
00028         np->child_ptr[i] = NULL;
00029     }
00030     return np;
00031 }
00032 template <typename T>
00033 void traverse(BTreeNode* p)
00034 {
00035     cout << endl;
00036     int i;
00037     for (i = 0; i < p->n; i++)
00038     {
00039         if (p->leaf == false)
00040         {
00041             traverse(p->child_ptr[i]);
00042         }
00043         cout << " " << p->data[i];
00044     }
00045     if (p->leaf == false)
00046     {
00047         traverse(p->child_ptr[i]);
00048     }
00049     cout << endl;
00050 }
00051 template <typename T>
00052 void sort(int* p, int n)
00053 {
00054     int i, j;
00055     T temp;
00056     for (i = 0; i < n; i++)
00057     {
00058         for (j = i; j <= n; j++)
00059         {
00060             if (p[i] > p[j])
00061             {
00062                 temp = p[i];
00063                 p[i] = p[j];
00064                 p[j] = temp;
00065             }
00066         }
00067     }
00068 }
00069 template <typename T>
00070 T split_child(BTreeNode* x, int i)
00071 {
00072     int j;
00073     T mid;
00074     BTreeNode* np1, * np3, * y;
00075     np3 = init();
00076     np3->leaf = true;
00077     if (i == -1)
00078     {
00079         mid = x->data[2];
00080         x->data[2] = 0;
00081         x->n--;
00082         np1 = init();
00083         np1->leaf = false;
00084         x->leaf = true;
00085         for (j = 3; j < 5; j++)
00086         {
00087             np3->data[j - 3] = x->data[j];
00088             np3->child_ptr[j - 3] = x->child_ptr[j];
00089             np3->n++;
00090             x->data[j] = 0;
00091             x->n--;
00092         }
00093         for (j = 0; j < 6; j++)
00094         {
00095             x->child_ptr[j] = NULL;
00096         }
00097         np1->data[0] = mid;
00098         np1->child_ptr[np1->n] = x;
00099         np1->child_ptr[np1->n + 1] = np3;
00100         np1->n++;
00101         root = np1;
00102     }
00103     else
00104     {
00105         y = x->child_ptr[i];
00106         mid = y->data[2];
00107         y->data[2] = 0;
00108         y->n--;
00109         for (j = 3; j < 5; j++)
00110         {
00111             np3->data[j - 3] = y->data[j];
00112             np3->n++;
00113             y->data[j] = 0;
00114             y->n--;
```

```
00115         }
00116         x->child_ptr[i + 1] = y;
00117         x->child_ptr[i + 1] = np3;
00118     }
00119     return mid;
00120 }
00121 template <typename T>
00122 void insert(T a)
00123 {
00124     int i;
00125     T temp;
00126     x = root;
00127     if (x == NULL)
00128     {
00129         root = init();
00130         x = root;
00131     }
00132     else
00133     {
00134         if (x->leaf == true && x->n == 5)
00135         {
00136             temp = split_child(x, -1);
00137             x = root;
00138             for (i = 0; i < (x->n); i++)
00139             {
00140                 if ((a > x->data[i]) && (a < x->data[i + 1]))
00141                 {
00142                     i++;
00143                     break;
00144                 }
00145                 else if (a < x->data[0])
00146                 {
00147                     break;
00148                 }
00149                 else
00150                 {
00151                     continue;
00152                 }
00153             }
00154             x = x->child_ptr[i];
00155         }
00156         else
00157         {
00158             while (x->leaf == false)
00159             {
00160                 for (i = 0; i < (x->n); i++)
00161                 {
00162                     if ((a > x->data[i]) && (a < x->data[i + 1]))
00163                     {
00164                         i++;
00165                         break;
00166                     }
00167                     else if (a < x->data[0])
00168                     {
00169                         break;
00170                     }
00171                     else
00172                     {
00173                         continue;
00174                     }
00175                 }
00176                 if ((x->child_ptr[i])->n == 5)
00177                 {
00178                     temp = split_child(x, i);
00179                     x->data[x->n] = temp;
00180                     x->n++;
00181                     continue;
00182                 }
00183                 else
00184                 {
00185                     x = x->child_ptr[i];
00186                 }
00187             }
00188         }
00189     }
00190     x->data[x->n] = a;
00191     sort(x->data, x->n);
00192     x->n++;
00193 }
00194 #endif
```

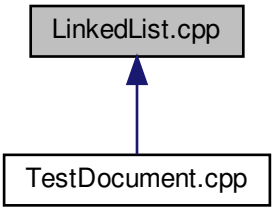
## 6.3 LinkedList.cpp File Reference

```
#include "LinkedList.h"
#include "Node.h"
#include <cassert>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
```

Include dependency graph for LinkedList.cpp:




This graph shows which files directly or indirectly include this file:




## 6.4 LinkedList.cpp

```
00001 #include "LinkedList.h"  // Header file
00002 #include "Node.h"
00003 //#include "PrecondViolatedExcep.h"
00004 #include <cassert>
00005 #include <fstream>
00006 #include <iostream>
00007 #include <string>
00008 #include <vector>
00009 #include <algorithm>
00010
00011 using namespace std;
00012
00014
```

```
00017 template<class ItemType>
00018 LinkedList<ItemType>::LinkedList() : headPtr(NULL), itemCount(0)
00019 {
00020 }  // end default constructor
00021
00023
00027 template<class ItemType>
00028 LinkedList<ItemType>::LinkedList(const
      LinkedList<ItemType>& aList) : itemCount(aList.itemCount)
00029 {
00030    Node<ItemType>* origChainPtr = aList.headPtr;  // Points to nodes in original chain
00031
00032    if (origChainPtr == NULL)
00033       headPtr = NULL;  // Original list is empty
00034    else
00035    {
00036       // Copy first node
00037       headPtr = new Node<ItemType>();
00038       headPtr->setItem(origChainPtr->getItem());
00039
00040       // Copy remaining nodes
00041       Node<ItemType>* newChainPtr = headPtr;      // Points to last node in new chain
00042       origChainPtr = origChainPtr->getNext();     // Advance original-chain pointer
00043       while (origChainPtr != NULL)
00044       {
00045          // Get next item from original chain
00046          ItemType nextItem = origChainPtr->getItem();
00047
00048          // Create a new node containing the next item
00049          Node<ItemType>* newNodePtr = new Node<ItemType>(nextItem);
00050
00051          // Link new node to end of new chain
00052          newChainPtr->setNext(newNodePtr);
00053
00054          // Advance pointer to new last node
00055          newChainPtr = newChainPtr->getNext();
00056
00057          // Advance original-chain pointer
00058          origChainPtr = origChainPtr->getNext();
00059       }  // end while
00060
00061       newChainPtr->setNext(NULL);              // Flag end of chain
00062    }  // end if
00063 }  // end copy constructor
00064
00066
00069 template<class ItemType>
00070 LinkedList<ItemType>::~LinkedList()
00071 {
00072    clear();
00073 }  // end destructor
00074
00076
00080 template<class ItemType>
00081 bool LinkedList<ItemType>::isEmpty() const
00082 {
00083    return itemCount == 0;
00084 }  // end isEmpty
00085
00087
00090 template<class ItemType>
00091 int LinkedList<ItemType>::getLength() const
00092 {
00093    return itemCount;
00094 }  // end getLength
00095
00097
00105 template<class ItemType>
00106 bool LinkedList<ItemType>::insert(int newPosition, const ItemType& newEntry)
00107 {
00108    bool ableToInsert = (newPosition >= 1) && (newPosition <= itemCount + 1);
00109    if (ableToInsert)
00110    {
00111       Node<ItemType>* newNodePtr = new Node<ItemType>(newEntry);
00112       if (newPosition == 1)
00113       {
00114          newNodePtr->setNext(headPtr);
00115          headPtr = newNodePtr;
00116       }
00117       else
00118       {
00119          Node<ItemType>* prevPtr = getNodeAt(newPosition - 1);
00120          newNodePtr->setNext(prevPtr->getNext());
00121          prevPtr->setNext(newNodePtr);
00122       }  // end if
00123       itemCount++;
00124    }  // end if
```

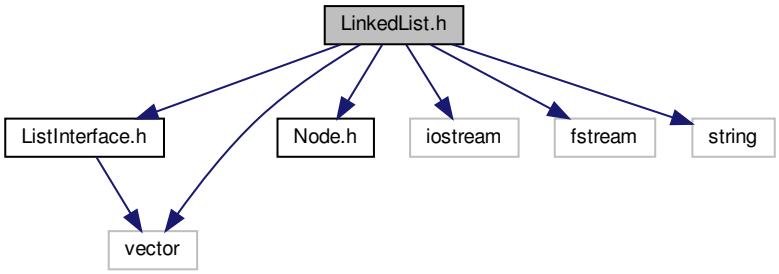
```
00125    return ableToInsert;
00126 }  // end inser
00127
00128
00129 /*
00130 template<class ItemType>
00131 void LinkedList<ItemType>::remove(int position)
00132 {
00133        bool ableToNull = (position >= 1) && (position <= itemCount);
00134        if (ableToNull)
00135        {
00136           Node<ItemType>* nodePtr = getNodeAt(position);
00137           nodePtr->setItem(NULL);
00138        }  // end if
00139
00140
00141 }  // end remove
00142 */
00143
00145
00150 template<class ItemType>
00151 bool LinkedList<ItemType>::deletion(int position)
00152 {
00153        bool ableToRemove = (position >= 1) && (position <= itemCount);
00154        if (ableToRemove)
00155        {
00156           Node<ItemType>* curPtr = NULL;
00157           if (position == 1)
00158           {
00159              curPtr = headPtr; // Save pointer to node
00160              headPtr = headPtr->getNext();
00161           }
00162           else
00163           {
00164              Node<ItemType>* prevPtr = getNodeAt(position - 1);
00165              curPtr = prevPtr->getNext();
00166              prevPtr->setNext(curPtr->getNext());
00167           }  // end if
00168
00169           curPtr->setNext(NULL);
00170           delete curPtr;
00171           curPtr = NULL;
00172           itemCount--;
00173
00174  // Decrease count of entries
00175    }  // end if
00176
00177    return ableToRemove;
00178 }  // end remove
00179
00181
00184 template<class ItemType>
00185 void LinkedList<ItemType>::clear()
00186 {
00187    while (!isEmpty())
00188       deletion(1);
00189 }  // end clear
00190
00192
00197 template<class ItemType>
00198 ItemType LinkedList<ItemType>::getEntry(int position) const//const
       throw(PrecondViolatedExcep)
00199 {
00200    bool ableToGet = (position > 0) && (position <= itemCount);
00201    if (ableToGet)
00202    {
00203       Node<ItemType>* nodePtr = getNodeAt(position);
00204       return nodePtr->getItem();
00205    }
00206    else
00207    {
00208        return ItemType();
00209        //throw(PrecondViolatedExcep(message));
00210    }  // end if
00211 }  // end getEntr
00212
00214
00218 template<class ItemType>
00219 void LinkedList<ItemType>::replace(int position, const ItemType& newEntry)//
       throw(PrecondViolatedExcep)
00220 {
00221    bool ableToSet = (position >= 1) && (position <= itemCount);
00222    if (ableToSet)
00223    {
00224       Node<ItemType>* nodePtr = getNodeAt(position);
00225       nodePtr->setItem(newEntry);
00226    }
```

```
00227    else
00228    {
00229       string message = "replace() called with an invalid position.";
00230       //throw(PrecondViolatedExcep(message));
00231    }  // end if
00232 }  // end replace
00233
00234
00236
00243 template<class ItemType>
00244 Node<ItemType>* LinkedList<ItemType>::getNodeAt(int position)
       const
00245 {
00246    // Debugging check of precondition
00247    assert( (position >= 1) && (position <= itemCount) );
00248
00249    // Count from the beginning of the chain
00250    Node<ItemType>* curPtr = headPtr;
00251    for (int skip = 1; skip < position; skip++)
00252       curPtr = curPtr->getNext();
00253
00254    return curPtr;
00255 }  // end getNodeAt
00256 //  End of implementation file.
00257
00259
00262 template<class ItemType>
00263 int LinkedList<ItemType>::getItemCount() const
00264 {
00265     return itemCount;
00266 }
00267
00269
00273 template<class ItemType>
00274 ItemType LinkedList<ItemType>::displayList()
00275 {
00276     for (int i = 0; i > itemCount; i++)
00277     {
00278         Node<ItemType>* nodePtr = getNodeAt(i);
00279         return nodePtr->getItem();
00280     }
00281 }
00282
00284
00288 template<class ItemType>
00289 LinkedList<ItemType>& LinkedList<ItemType>::operator =
     (const LinkedList<ItemType>& rhs)
00290 {
00291     LinkedList<ItemType> temp(rhs);
00292     swap(temp.headPtr, headPtr);
00293     return *this;
00294 }
```

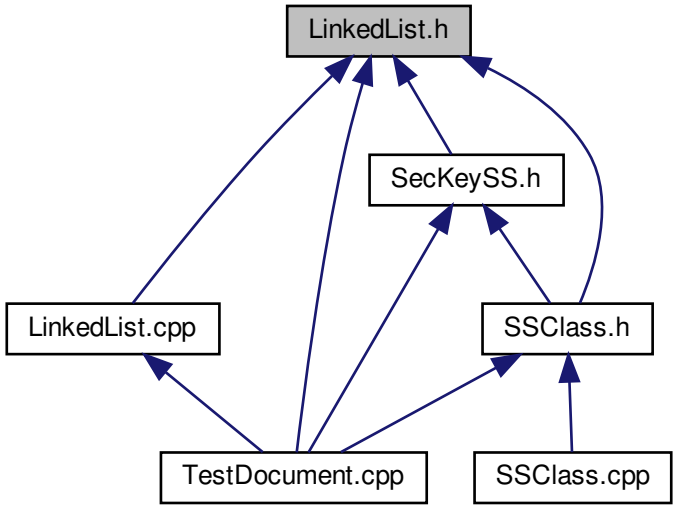
## 6.5 LinkedList.h File Reference

```
#include "ListInterface.h"
#include "Node.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
```

Include dependency graph for LinkedList.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class LinkedList< ItemType >

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

## 6.6 LinkedList.h

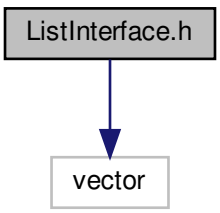
```
00001 /*************************************************************************/
00008 #ifndef LINKED_LIST_
00009 #define LINKED_LIST_
00010
```

```
00011 #include "ListInterface.h"
00012 #include "Node.h"
00013 #include <iostream>
00014 #include <fstream>
00015 #include <string>
00016 #include <vector>
00017
00018 template<class ItemType>
00019 class LinkedList : public ListInterface<ItemType>
00020 {
00021 private:
00022    Node<ItemType>* headPtr; // Pointer to first node in the chain;
00023    // (contains the first entry in the list)
00024    int itemCount; // Current count of list items
00025    // Locates a specified node in this linked list.
00026    // @pre  position is the number of the desired node;
00027    //       position >= 1 and position <= itemCount.
00028    // @post  The node is found and a pointer to it is returned.
00029    // @param position  The number of the node to locate.
00030    // @return  A pointer to the node at the given position.
00031    Node<ItemType>* getNodeAt(int position) const;
00032
00033 public:
00034    LinkedList();
00035    LinkedList(const LinkedList<ItemType>& aList);
00036    virtual ~LinkedList();
00037
00038
00039    bool isEmpty() const;
00040    int getLength() const;
00041    bool insert(int newPosition, const ItemType& newEntry);
00042    //void remove(int position);
00043    bool deletion(int position);
00044    void clear();
00045    int getItemCount() const;
00046    LinkedList<ItemType>& operator = (const
      LinkedList<ItemType>& rhs);
00047
00050    ItemType getEntry(int position) const;
00051
00054    void replace(int position, const ItemType &newEntry);
00055
00056    ItemType displayList();
00057
00058
00059
00060 }; // end LinkedList
00061
00062 //#include "LinkedList.cpp"
00063 #endif
```

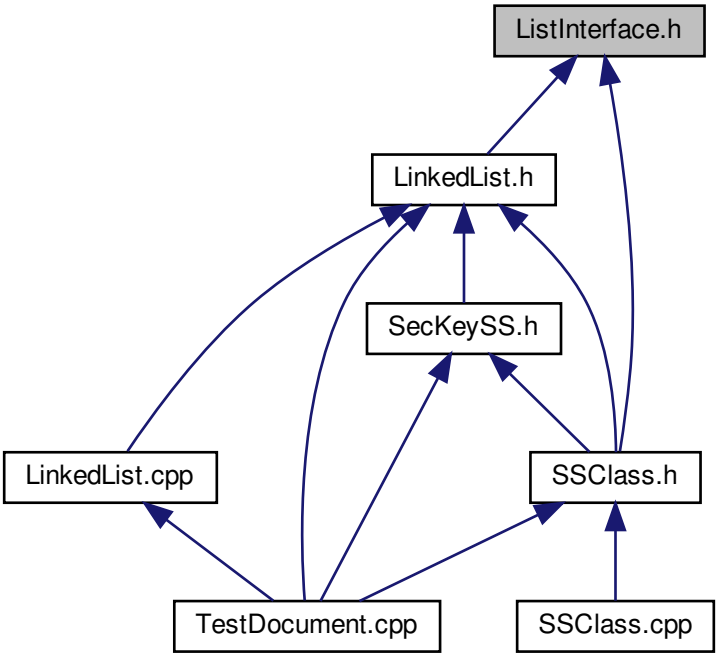
## 6.7 ListInterface.h File Reference

`#include <vector>`

Include dependency graph for ListInterface.h:



This graph shows which files directly or indirectly include this file:




### Classes

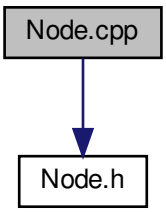
- class ListInterface< ItemType >

## 6.8 ListInterface.h

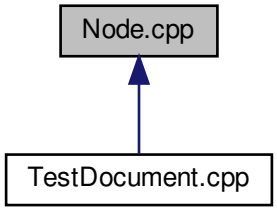
```
00001 #include <vector>
00002
00003 #ifndef _LIST_INTERFACE
00004 #define _LIST_INTERFACE
00005
00006 template<class ItemType>
00007 class ListInterface
00008 {
00009 public:
00012    virtual bool isEmpty() const = 0;
00013
00016    virtual int getLength() const = 0;
00017    virtual int getItemCount() const = 0;
00027    virtual bool insert(int newPosition, const ItemType& newEntry) = 0;
00028
00029    //virtual void remove(int position);
00037    //virtual void remove(int position) = 0;
00038    virtual bool deletion(int position) = 0;
00041    virtual void clear() = 0;
00042
00048    virtual ItemType getEntry(int position) const = 0;
00049
00055    virtual void replace(int position, const ItemType& newEntry) = 0;
00056
00057    virtual ItemType displayList() = 0;
00058 }; // end ListInterface
00059 #endif
```

## 6.9 Node.cpp File Reference

`#include "Node.h"`
Include dependency graph for Node.cpp:




This graph shows which files directly or indirectly include this file:



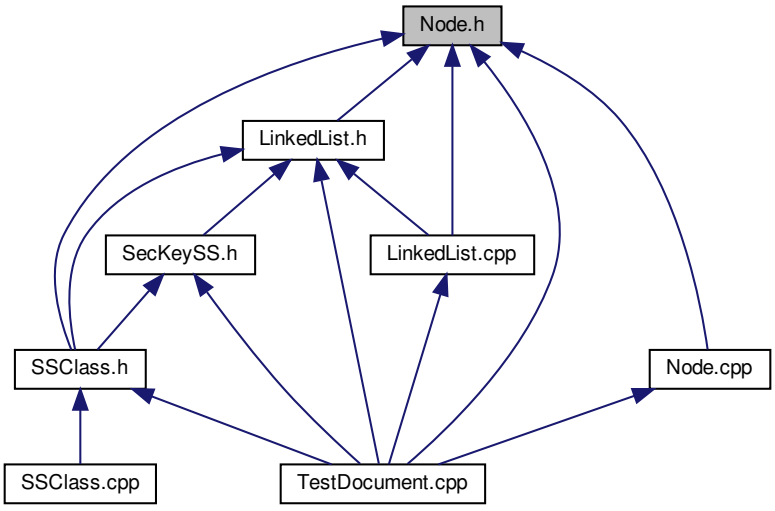

## 6.10 Node.cpp

```
00001 #include "Node.h"
00002
00004
00007 template<class ItemType>
00008 Node<ItemType>::Node() : next(nullptr)
00009 {
00010 } // end default constructor
00011
00013
00017 template<class ItemType>
00018 Node<ItemType>::Node(const ItemType& anItem) : item(anItem), next(nullptr)
00019 {
00020 } // end constructor
00021
00023
00029 template<class ItemType>
00030 Node<ItemType>::Node(const ItemType& anItem, Node<ItemType>* nextNodePtr)
      :
00031     item(anItem), next(nextNodePtr)
00032 {
00033 } // end constructor
00034
```

```
00036
00039 template<class ItemType>
00040 void Node<ItemType>::setItem(const ItemType& anItem)
00041 {
00042     item = anItem;
00043 } // end setItem
00044
00046
00049 template<class ItemType>
00050 void Node<ItemType>::setNext(Node<ItemType>* nextNodePtr)
00051 {
00052     next = nextNodePtr;
00053 } // end setNext
00054
00056
00059 template<class ItemType>
00060 ItemType Node<ItemType>::getItem() const
00061 {
00062     return item;
00063 } // end getItem
00064
00066
00069 template<class ItemType>
00070 Node<ItemType>* Node<ItemType>::getNext() const
00071 {
00072     return next;
00073 } // end getNext
```

## 6.11 Node.h File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class Node< ItemType >

  *This is Node class for linked list.*

## 6.12 Node.h

```
00001 /*****************************************************************************/
00008 #ifndef NODE_
00009 #define NODE_
00010
00011 template<class ItemType>
00012 class Node
00013 {
00014 private:
00015    ItemType        item; // A data item
00016    Node<ItemType>* next; // Pointer to next node
00017
00018 public:
00019    Node();
00020    Node(const ItemType& anItem);
00021    Node(const ItemType& anItem, Node<ItemType>* nextNodePtr);
00022    void setItem(const ItemType& anItem);
00023    void setNext(Node<ItemType>* nextNodePtr);
00024    ItemType getItem() const ;
00025    Node<ItemType>* getNext() const ;
00026 }; // end Node
00027
00028 //#include "Node.cpp"
00029 #endif
```

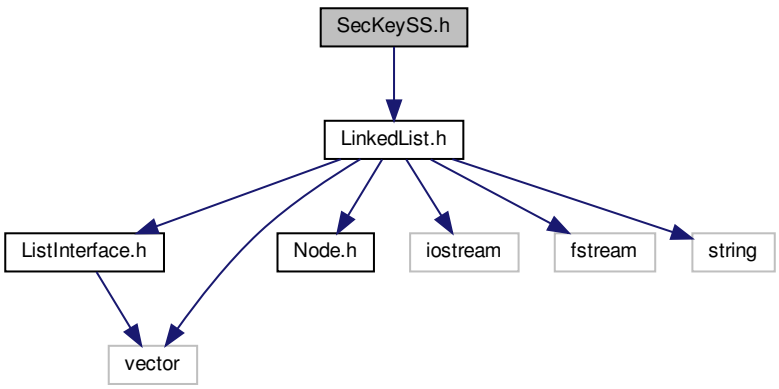
## 6.13 README.md File Reference

## 6.14 README.md

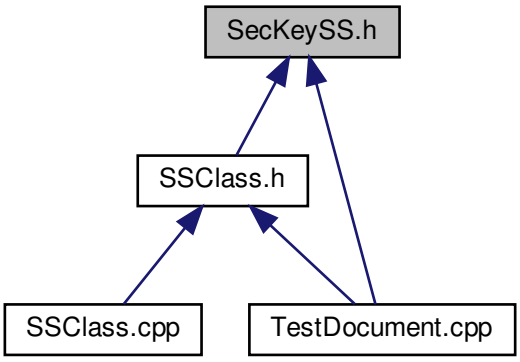
```
00001 # CSCI331Project
00002 Github for the CSCI 331 Sequence Set Class Group Programming Project
```

## 6.15 SecKeySS.h File Reference

`#include "LinkedList.h"`
Include dependency graph for SecKeySS.h:



This graph shows which files directly or indirectly include this file:




## Classes

- class SecKeySS< T >

  *This is the class for Section Keys of the SS class.*

## Functions

- template<typename T >
  bool operator< (const T s1, SecKeySS< T > &s2)
- template<typename T >
  bool operator> (const T s1, SecKeySS< T > s2)
- template<typename T >
  bool operator== (const T s1, SecKeySS< T > s2)

### 6.15.1 Function Documentation

#### 6.15.1.1 operator<()

```
template<typename T >
bool operator< (
            const T s1,
            SecKeySS< T > & s2 )
```

Definition at line 113 of file SecKeySS.h.

#### 6.15.1.2 operator==()

```
template<typename T >
bool operator== (
             const T s1,
             SecKeySS< T > s2 )
```

Definition at line 121 of file SecKeySS.h.

#### 6.15.1.3 operator>()

```
template<typename T >
bool operator> (
             const T s1,
             SecKeySS< T > s2 )
```

Definition at line 117 of file SecKeySS.h.

## 6.16 SecKeySS.h

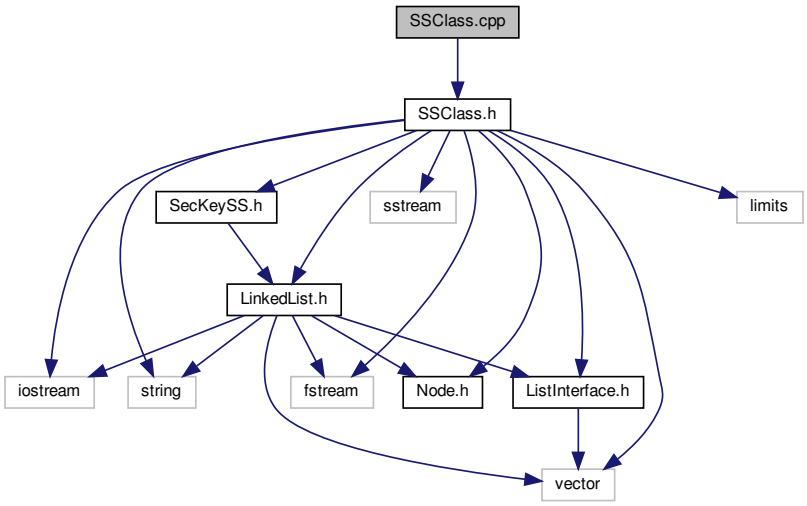
```
00001
00006 #ifndef SECKEYSS
00007 #define SECKEYSS
00008
00009 #include "LinkedList.h"
00010 //#include <string>
00011
00012 using namespace std;
00013 template <typename T>
00014 class SecKeySS {
00015 private:
00016     T data;
00017     LinkedList<T> duplicates;
00018     LinkedList<T> list;
00019 public:
00021     //template <typename T>
00022     SecKeySS() { duplicates = LinkedList<T>(); };
00023
00025 //  template <typename T>
00026     SecKeySS( SecKeySS<T>& s);
00027
00029 //  template <typename T>
00030     ~SecKeySS();
00031
00035 //  template <typename T>
00036     T getData() const { return data; };
00037
00041     //template <typename T>
00042     LinkedList<T> getDuplicates() ;
00043
00047 //  template <typename T>
00048     void setData(const T s) { data = s; };
00049
00053 //  template <typename T>
00054     void setDuplicates( LinkedList<T> dup);
00055
00060 //  template <typename T>
00061     bool operator <(const T &s)const { return data < s; };
00062
00063
00068 //  template <typename T>
00069     bool operator <(const SecKeySS<T>& s)const { return data < s.data; };
00070
00071
00076 //  template <typename T>
00077     bool operator >(const T &s)const { return data > s; };
00078
```

```
00079
00084 //  template <typename T>
00085     bool operator >(const SecKeySS<T> &s)const { return data > s.data; };
00086
00091 //  template <typename T>
00092     bool operator ==(const T &s)const { return data == s; };
00093
00094
00099 //  template <typename T>
00100     bool operator ==(const SecKeySS<T> &s)const { return data == s.data; };
00101
00105 //  template <typename T>
00106     void operator = (const SecKeySS<T> &s);
00107 };
00108 template <typename T>
00109 SecKeySS<T>::SecKeySS( SecKeySS<T>& s) { data = s.
      getData(); setDuplicates(s.getDuplicates()); }
00110 template <typename T>
00111 SecKeySS<T>::~SecKeySS() { duplicates.clear(); }
00112 template <typename T>
00113 bool operator <(const T s1, SecKeySS<T> &s2) {
00114     return s1 < s2.getData();
00115 }
00116 template <typename T>
00117 bool operator >(const T s1, SecKeySS<T> s2) {
00118     return s1 > s2.getData();
00119 }
00120 template <typename T>
00121 bool operator ==(const T s1, SecKeySS<T> s2) {
00122     return s1 == s2.getData();
00123 }
00124 template <typename T>
00125 void SecKeySS<T>::operator = (const SecKeySS<T> &s){
00126     data = s.data;
00127     duplicates = s.duplicates;
00128 }
00129 template <typename T>
00130 LinkedList<T> SecKeySS<T>::getDuplicates() {
00131     T temp;
00132     for (int i = 1; i < duplicates.getItemCount() + 1; i++) {
00133         temp = duplicates.getEntry(i);
00134         list.insert(i, temp);
00135     }
00136     return list;
00137 }
00138 template <typename T>
00139 void SecKeySS<T>::setDuplicates(LinkedList<T> list) {
00140     T temp;
00141     duplicates.clear();
00142     for (int i = 1; i < list.getItemCount() + 1; i++) {
00143         temp = list.getEntry(i);
00144         duplicates.insert(i, temp);
00145     }
00146 }
00147
00148 #endif
```

## 6.17 SSClass.cpp File Reference

```
#include "SSClass.h"
```

Include dependency graph for SSClass.cpp:




## 6.18 SSClass.cpp

```
00001
00002 #include "SSClass.h"
00003
00004
00005 /*-------------------------------------------------------------------------
00006     Opens file
00007     Preconditions:   File needs to be created
00008     Postconditions:  None
00009 */
00010 bool SSClass::openFile(string input) { //input is a file name
00011     indexFile.open(input);
00012     nextEmpty = -1;
00013     return (indexFile.is_open());
00014
00015 }
00016
00017 /*
00018 bool SSClass::createIndexFile() {
00019     indexFile.open("index.txt");
00020     return indexFile.is_open();
00021 }
00022 */
00023 /*-------------------------------------------------------------------------
00024     Creates block record file
00025     Preconditions:   None
00026     Postconditions:  None
00027 */
00028 /*
00029 bool SSClass::createBlockRecordFile() {
00030     blockRecordFile.open("blockRecord.txt");
00031     return blockRecord.is_open();
00032 }
00033 */
00034 /*-------------------------------------------------------------------------
00035     Default constructor
00036     Preconditions:   None
00037     Postconditions:  None
00038 */
00039 SSClass::SSClass() {
00040     numRecords = 0;
00041     openFile("us_postal_codes.txt");
00042 }
00043 SSClass::SSClass(const SSClass& ss) {
00044     numLinesIndex = ss.numLinesIndex;
00045     numRecords = ss.numRecords;
```

```
00046     nextEmpty = ss.nextEmpty;
00047     secKeyZip = ss.secKeyZip;
00048     secKeyPlace= ss.secKeyPlace;
00049     secKeyState = ss.secKeyState;
00050     secKeyCounty = ss.secKeyCounty;
00051     secKeyLat = ss.secKeyLat;
00052     secKeyLon = ss.secKeyLon;
00053     openFile("us_postal_codes.txt");
00054 }
00055 SSClass::~SSClass(){
00056     secKeyZip.clear();
00057     secKeyPlace.clear();
00058     secKeyState.clear();
00059     secKeyCounty.clear();
00060     secKeyLat.clear();
00061     secKeyLon.clear();
00062     indexFile.close();
00063     //blockRecord.close();
00064 }
00065
00070 void SSClass::insert(string s) {
00071     if (nextEmpty == -1) {
00072         goToLine(indexFile, numLinesIndex);
00073         indexFile << "\n" << s;
00074         insertZip(getZip(s), numLinesIndex);
00075         insertPlace(getPlace(s), numLinesIndex);
00076         insertState(getState(s), numLinesIndex);
00077         insertCounty(getCounty(s), numLinesIndex);
00078         insertLat(getLat(s), numLinesIndex);
00079         insertLon(getLon(s), numLinesIndex);
00080         numLinesIndex++;
00081         return;
00082     }
00083     goToLine(indexFile, nextEmpty);
00084     //replace(s, nextEmpty);
00085     insertZip(getZip(s), nextEmpty);
00086     insertPlace(getPlace(s), nextEmpty);
00087     insertState(getState(s), nextEmpty);
00088     insertCounty(getCounty(s), nextEmpty);
00089     insertLat(getLat(s), nextEmpty);
00090     insertLon(getLon(s), nextEmpty);
00091 }
00092
00093 string SSClass::returnLine(int rrn) {
00094     string returnVal;
00095     goToLine(indexFile, rrn);
00096     getline(indexFile, returnVal);
00097     return returnVal;
00098 }
00099
00100
00101 vector<int> SSClass::search(string s, unsigned fieldNum) {
00102     typedef SecKeySS<string> secCopy;
00103     int i;
00104     vector<int> results;
00105     switch (fieldNum) {
00106     case 1:
00107     {
00108         for (i = 1; (i < (secKeyZip.getItemCount() + 1)) && (secKeyZip.
      getEntry(i).getData() < stoi(s)); i++);
00109         if (secKeyZip.getEntry(i).getData() == stoi(s)) {
00110             LinkedList<int> toCopy = LinkedList<int>(secKeyZip.
      getEntry(i).getDuplicates());
00111             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00112                 results.push_back(toCopy.getEntry(j));
00113             }
00114         }
00115     }
00116     break;
00117     case 2:
00118     {
00119         for(i = 1; (i < (secKeyPlace.getItemCount() + 1)) && (secKeyPlace.
      getEntry(i).getData() < s); i++);
00120         if ((secKeyPlace.getEntry(i).getData()) == (s)) {
00121             LinkedList<string> toCopy = LinkedList<string>(secKeyPlace.
      getEntry(i).getDuplicates());
00122             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00123                 // stoi toCopy.getEntry returns string
00124                 results.push_back(stoi(toCopy.getEntry(j)));
00125             }
00126         }
00127     }
00128     break;
00129     case 3:
00130     {
00131         for (i = 1; (i < (secKeyState.getItemCount() + 1)) && (secKeyState.
      getEntry(i).getData() < s); i++);
```

```
00132         if ((secKeyState.getEntry(i).getData()) == (s)) {
00133             LinkedList<string> toCopy = LinkedList<string>(secKeyState.
      getEntry(i).getDuplicates());
00134             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00135                 // stoi toCopy.getEntry returns string
00136                 results.push_back(stoi(toCopy.getEntry(j)));
00137             }
00138         }
00139     }
00140     break;
00141     case 4:
00142     {
00143         for (i = 1; (i < (secKeyCounty.getItemCount() + 1)) && (secKeyCounty.
      getEntry(i).getData() < s); i++);
00144         if ((secKeyCounty.getEntry(i).getData()) == (s)) {
00145             LinkedList<string> toCopy = LinkedList<string>(secKeyCounty
      .getEntry(i).getDuplicates());
00146             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00147                 // stoi toCopy.getEntry returns string
00148                 results.push_back(stoi(toCopy.getEntry(j)));
00149             }
00150         }
00151     }
00152     break;
00153     case 5:
00154     {
00155         for (i = 1; (i < (secKeyLat.getItemCount() + 1)) && (secKeyLat.
      getEntry(i).getData() < stoi(s)); i++);
00156         if (secKeyLat.getEntry(i).getData() == static_cast<int>(stod(s))) {
00157             LinkedList<int> toCopy = LinkedList<int>(secKeyLat.
      getEntry(i).getDuplicates());
00158             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00159                 results.push_back(toCopy.getEntry(j));
00160             }
00161         }
00162     }
00163     break;
00164     case 6:
00165     {
00166         for (i = 1; (i < (secKeyLon.getItemCount() + 1)) && (secKeyLon.
      getEntry(i).getData() < stoi(s)); i++);
00167         if (secKeyLon.getEntry(i).getData() == static_cast<int>(stod(s))) {
00168             LinkedList<int> toCopy = LinkedList<int>(secKeyLon.
      getEntry(i).getDuplicates());
00169             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
00170                 results.push_back(toCopy.getEntry(j));
00171             }
00172         }
00173     }
00174     break;
00175     }
00176     return results;
00177 }
00178
00179 int SSClass::directionalSearch(string stateS, char direction) {
00180     direction = toupper(direction);
00181     int i = 1;
00182     int returnIndex = -1;
00183     double highOrLow;
00184     vector<int> state = search(stateS, 3);
00185     switch (direction) {
00186     case 'N':
00187     {
00188         returnIndex = state[0];
00189         highOrLow = stod(getLat(returnLine(state[0])));
00190         for (i; i < state.size(); i++) {
00191             if (highOrLow < stod(getLat(returnLine(state[i])))) {
00192                 highOrLow = stod(getLat(returnLine(state[i])));
00193                 returnIndex = i;
00194             }
00195
00196         }
00197     }
00198     break;
00199     case 'E':
00200     {
00201         returnIndex = state[0];
00202         highOrLow = stod(getLon(returnLine(state[0])));
00203         for (i; i < state.size(); i++) {
00204             if (highOrLow < stod(getLon(returnLine(state[i])))) {
00205                 highOrLow = stod(getLon(returnLine(state[i])));
00206                 returnIndex = i;
00207             }
00208         }
00209
00210     }
00211     break;
```

```
00212     case 'S':
00213     {
00214         returnIndex = state[0];
00215         highOrLow = stod(getLat(returnLine(state[0])));
00216         for (i; i < state.size(); i++) {
00217             if (highOrLow > stod(getLat(returnLine(state[i])))) {
00218                 highOrLow = stod(getLat(returnLine(state[i])));
00219                 returnIndex = i;
00220             }
00221         }
00222         break;
00223     }
00224     case 'W':
00225     {
00226         returnIndex = state[0];
00227         highOrLow = stod(getLon(returnLine(state[0])));
00228         for (i; i < state.size(); i++) {
00229             if (highOrLow > stod(getLon(returnLine(state[i])))) {
00230                 highOrLow = stod(getLon(returnLine(state[i])));
00231                 returnIndex = i;
00232             }
00233         }
00234
00235     }
00236     break;
00237     }
00238     return returnIndex;
00239
00240 }
00241
00242 //get value at index in getEntry(index)          insert is insert(index)
00243 void SSClass::insertZip(string st, int rrn) {                //no sec key matching -> create new one....
     match found -> insert at index 1
00244     int index;
00245     int s = stoi(st);
00246     SecKeySS<int> secCopy;
00247     LinkedList<int> copyDup;
00248         int i;
00249     for (i = 1; (i < (secKeyZip.getItemCount() + 1)) && (secKeyZip.
    getEntry(i).getData() < s); i++);
00250     if (secKeyZip.getEntry(i).getData() == s) {
00251         secCopy = secKeyZip.getEntry(i);
00252         copyDup = LinkedList<int>(secCopy.getDuplicates());
00253         copyDup.insert(1, rrn);
00254         secCopy.setDuplicates(copyDup);
00255         secKeyZip.replace(i, secCopy);
00256         return;
00257     }
00258     copyDup.insert(1, rrn);
00259     secCopy.setDuplicates(copyDup);
00260     secCopy.setData(s);
00261     secKeyZip.insert(i, secCopy);
00262
00263 }
00264
00265 void SSClass::insertPlace(string s, int rrn) {
00266     int index;
00267     SecKeySS<string> secCopy;
00268     LinkedList<string> copyDup;
00269         int i;
00270     for (i = 1; (i < (secKeyPlace.getItemCount() + 1)) && (secKeyPlace.
    getEntry(i).getData() < s); i++);
00271     if (secKeyPlace.getEntry(i).getData() == s) {
00272         secCopy = secKeyPlace.getEntry(i);
00273         copyDup = LinkedList<string>(secCopy.getDuplicates());
00274         copyDup.insert(1, to_string(rrn));
00275         secCopy.setDuplicates(copyDup);
00276         secKeyPlace.replace(i, secCopy);
00277         return;
00278     }
00279     copyDup.insert(1, to_string(rrn));
00280     secCopy.setDuplicates(copyDup);
00281     secCopy.setData(getPlace(s));
00282     secKeyPlace.insert(i, secCopy);
00283 }
00284
00285 void SSClass::insertState(string s, int rrn) {
00286     int index;
00287     SecKeySS<string> secCopy;
00288     LinkedList<string> copyDup;
00289         int i;
00290     for (i = 1; (i < (secKeyState.getItemCount() + 1)) && (secKeyState.
    getEntry(i).getData() < s); i++);
00291     if (secKeyState.getEntry(i).getData() == s) {
00292         secCopy = secKeyState.getEntry(i);
00293         copyDup = LinkedList<string>(secCopy.getDuplicates());
00294         copyDup.insert(1, to_string(rrn));
```

```
00295         secCopy.setDuplicates(copyDup);
00296         secKeyState.replace(i, secCopy);
00297         return;
00298     }
00299     copyDup.insert(1, to_string(rrn));
00300     secCopy.setDuplicates(copyDup);
00301     secCopy.setData(getState(s));
00302     secKeyState.insert(i, secCopy);
00303 }
00304
00305 void SSClass::insertCounty(string s, int rrn) {
00306     int index;
00307     SecKeySS<string> secCopy;
00308     LinkedList<string> copyDup;
00309         int i;
00310     for (i = 1; (i < (secKeyCounty.getItemCount() + 1)) && (secKeyCounty.
      getEntry(i).getData() < s); i++);
00311     if (secKeyCounty.getEntry(i).getData() == s) {
00312         secCopy = secKeyCounty.getEntry(i);
00313         copyDup = LinkedList<string>(secCopy.getDuplicates());
00314         copyDup.insert(1, to_string(rrn));
00315         secCopy.setDuplicates(copyDup);
00316         secKeyCounty.replace(i, secCopy);
00317         return;
00318     }
00319     copyDup.insert(1, to_string(rrn));
00320     secCopy.setDuplicates(copyDup);
00321     secCopy.setData(getCounty(s));
00322     secKeyCounty.insert(i, secCopy);
00323 }
00324
00325 void SSClass::insertLat(string st, int rrn) {
00326     int index;
00327     int s = static_cast<int>(stod(st));
00328     SecKeySS<int> secCopy;
00329     LinkedList<int> copyDup;
00330         int i;
00331     for (i = 1; (i < (secKeyLat.getItemCount() + 1)) && (secKeyLat.
      getEntry(i).getData() < s); i++);
00332     if (secKeyLat.getEntry(i).getData() == s) {
00333         secCopy = secKeyLat.getEntry(i);
00334         copyDup = LinkedList<int>(secCopy.getDuplicates());
00335         copyDup.insert(1, rrn);
00336         secCopy.setDuplicates(copyDup);
00337         secKeyLat.replace(i, secCopy);
00338         return;
00339     }
00340     copyDup.insert(1, rrn);
00341     secCopy.setDuplicates(copyDup);
00342     secCopy.setData(static_cast<int>(stod(st)));
00343     secKeyLat.insert(i, secCopy);
00344 }
00345
00346 void SSClass::insertLon(string st, int rrn) {
00347     int index;
00348     int s = static_cast<int>(stod(st));
00349     SecKeySS<int> secCopy;
00350     LinkedList<int> copyDup;
00351         int i;
00352     for (i = 1; (i < (secKeyLon.getItemCount() + 1)) && (secKeyLon.
      getEntry(i).getData() < s); i++);
00353     if (secKeyLon.getEntry(i).getData() == s) {
00354         secCopy = secKeyLon.getEntry(i);
00355         copyDup = LinkedList<int>(secCopy.getDuplicates());
00356         copyDup.insert(1, rrn);
00357         secCopy.setDuplicates(copyDup);
00358         secKeyLon.replace(i, secCopy);
00359         return;
00360     }
00361     copyDup.insert(1, rrn);
00362     secCopy.setDuplicates(copyDup);
00363     secCopy.setData(static_cast<int>(stod(st)));
00364     secKeyLon.insert(i, secCopy);
00365 }
00366
00367 void SSClass::goToLine(fstream& file, unsigned num) {
00368     goToData(file); //beginning of our data file
00369     for (int i = 0; i < num - 1; ++i) {
00370         file.ignore(1000, '\n'); //ignore one line
00371     }
00372     //return file;
00373 }
00374
00375 void SSClass::goToData(fstream& file) { //puts cursor at the beginning of the data portion of the txt file
00376     file.seekg(ios::beg);
00377     string in;
00378     getline(file, in);
```

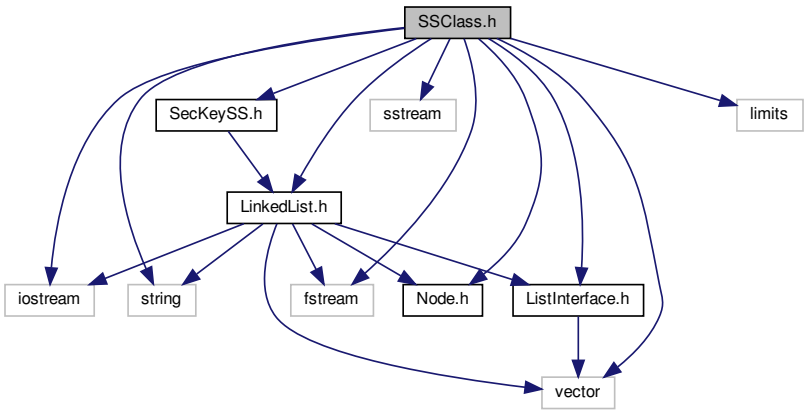
```
00379     while (in != "ENDOFHDR")
00380         getline(file, in);
00381 }
00382
00383 string SSClass::getZip(string s) { //use stoi(getzip(s)); to return int value
00384     string returnValue;
00385     for (int i = 0; i < ZIPSIZE; i++)
00386         returnValue[i] = s[ZIPOFFSET + i];
00387     return returnValue;
00388 }
00389
00390 string SSClass::getPlace(string s) {
00391     string returnvalue;
00392     for (int i = 0; i < PLACESIZE; i++)
00393         returnvalue[i] = s[PLACEOFFSET + i];
00394     return returnvalue;
00395 }
00396
00397 string SSClass::getState(string s) {
00398     string returnvalue;
00399     for (int i = 0; i < STATESIZE; i++)
00400         returnvalue[i] = s[STATEOFFSET + i];
00401     return returnvalue;
00402 }
00403
00404 string SSClass::getCounty(string s) {
00405     string returnvalue;
00406     for (int i = 0; i < COUNTYSIZE; i++)
00407         returnvalue[i] = s[COUNTYOFFSET + i];
00408     return returnvalue;
00409 }
00410
00411 string SSClass::getLat(string s) { //use stod(getlat(s)); to return double value
00412     string returnvalue;
00413     for (int i = 0; i < LATSIZE; i++)
00414         returnvalue[i] = s[LATOFFSET + i];
00415     return returnvalue;
00416 }
00417
00418 string SSClass::getLon(string s) { //use stod(getLon(s)); to return double value
00419     string returnValue;
00420     for (int i = 0; i < LONSIZE; i++)
00421         returnValue[i] = s[LONOFFSET + i];
00422     return returnValue;
00423 }
00424
00425 string SSClass::createUnusedLine(int next) { //pass in the integer value of the next empty line
00426     string unusedLine = to_string(next);
00427     int i;
00428     for (i = unusedLine.size(); i < CHARINLINE; i++) {
00429         unusedLine += " ";
00430     }
00431     return unusedLine;
00432 }
00433
00434 /*
00435 bool SSClass::replace(string s, int line) { // To be able to replace a line in a text file, you have to
       write everything to a new file, with the updated line, then delete the previous file
00436     goToLine(indexFile, line);              // and rename the temperary file
00437     string strReplace;
00438     getline(indexFile, strReplace);
00439     string strNew = s;
00440     ofstream fileout("temp_file.txt"); //Temporary file
00441     if (!fileout)
00442         return false;
00443
00444     string strTemp;
00445     indexFile.seekg(ios::beg);
00446     while (strTemp = indexFile.getline())
00447     {
00448         if (strTemp == strReplace) {
00449             strTemp = strNew;
00450         }
00451         fileout << "\n";
00452         for (int i = 0; i < ZIPSIZE; i++) { //use this for zip since there may be leading whitespace
00453             fileout << strTemp[i];
00454             strTemp[i] = ' ';
00455         }
00456         fileout << strTemp;
00457     }
00458     remove(indexFile);
00459     rename("temp_file.txt", "us_postal_codes.txt");
00460     close(fileout);
00461     openFile("us_postal_codes.txt");
00462     return true;
00463 }
00464 }
```

```
00465 */
00466
00467 void SSClass::populate() {
00468     goToData(indexFile);
00469     string line;
00470     while (!indexFile.eof()) {
00471         getline(indexFile, line);
00472         insert(line);
00473     }
00474 }
```

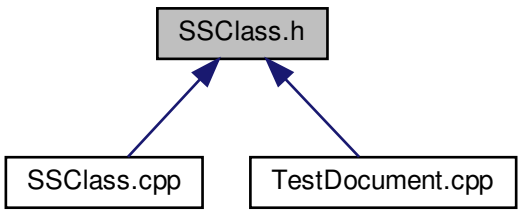
## 6.19 SSClass.h File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include "LinkedList.h"
#include "Node.h"
#include "SecKeySS.h"
#include "ListInterface.h"
#include <limits>
```

Include dependency graph for SSClass.h:




This graph shows which files directly or indirectly include this file:



## Classes

- class SSClass

  *LinkedList integration for blocks, records, and fields.*

## Variables

- const int NUMSECKEYS = 6

  *NUMSECKEYS The numebr of section keys.*
- const int ZIPSIZE = 6

  *ZIPSIZE The size of the zip code.*
- const int PLACESIZE = 31

  *PLACESIZE The size of the place (city)*
- const int STATESIZE = 2

  *STATESIZE The size of the sate letters.*
- const int COUNTYSIZE = 36

  *COUNTYSIZE The size of letters for the county.*
- const int LATSIZE = 9

  *LATSIZE The size of the Lattatude.*
- const int LONSIZE = 10

  *LONSIZE The size (including sign) of the longitude.*
- const int ZIPOFFSET = 0
- const int PLACEOFFSET = ZIPSIZE - 1
- const int STATEOFFSET = PLACEOFFSET + PLACESIZE
- const int COUNTYOFFSET = STATEOFFSET + STATESIZE
- const int LATOFFSET = COUNTYOFFSET + COUNTYSIZE
- const int LONOFFSET = LATOFFSET + LATSIZE
- const int CHARINLINE = LONOFFSET + LONSIZE

### 6.19.1 Variable Documentation

#### 6.19.1.1 CHARINLINE

```
const int CHARINLINE = LONOFFSET + LONSIZE
```

Definition at line 63 of file SSClass.h.

#### 6.19.1.2 COUNTYOFFSET

```
const int COUNTYOFFSET = STATEOFFSET + STATESIZE
```

Definition at line 60 of file SSClass.h.

#### 6.19.1.3 COUNTYSIZE

```
const int COUNTYSIZE = 36
```

COUNTYSIZE The size of letters for the county.

Definition at line 49 of file SSClass.h.

#### 6.19.1.4 LATOFFSET

```
const int LATOFFSET = COUNTYOFFSET + COUNTYSIZE
```

Definition at line 61 of file SSClass.h.

#### 6.19.1.5 LATSIZE

```
const int LATSIZE = 9
```

LATSIZE The size of the Lattatude.

Definition at line 52 of file SSClass.h.

#### 6.19.1.6 LONOFFSET

```
const int LONOFFSET = LATOFFSET + LATSIZE
```

Definition at line 62 of file SSClass.h.

#### 6.19.1.7 LONSIZE

```
const int LONSIZE = 10
```

LONSIZE The size (including sign) of the longitude.

Definition at line 55 of file SSClass.h.

#### 6.19.1.8 NUMSECKEYS

```
const int NUMSECKEYS = 6
```

NUMSECKEYS The numebr of section keys.

Definition at line 37 of file SSClass.h.

#### 6.19.1.9 PLACEOFFSET

```
const int PLACEOFFSET = ZIPSIZE - 1
```

Definition at line 58 of file SSClass.h.

#### 6.19.1.10 PLACESIZE

```
const int PLACESIZE = 31
```

PLACESIZE The size of the place (city)

Definition at line 43 of file SSClass.h.

#### 6.19.1.11 STATEOFFSET

```
const int STATEOFFSET = PLACEOFFSET + PLACESIZE
```

Definition at line 59 of file SSClass.h.

#### 6.19.1.12 STATESIZE

```
const int STATESIZE = 2
```

STATESIZE The size of the sate letters.

Definition at line 46 of file SSClass.h.

#### 6.19.1.13 ZIPOFFSET

```
const int ZIPOFFSET = 0
```

Definition at line 57 of file SSClass.h.

#### 6.19.1.14 ZIPSIZE

```
const int ZIPSIZE = 6
```

ZIPSIZE The size of the zip code.

Definition at line 40 of file SSClass.h.

## 6.20 SSClass.h

```
00001
00020 #ifndef SSCLASS_
00021 #define SSCLASS_
00022
00023 #include <iostream>
00024 #include <string>
00025 #include <vector>
00026 #include <fstream>
00027 #include <sstream>
00028 #include "LinkedList.h"
00029 #include "Node.h"
00030 #include "SecKeySS.h"
00031 #include "ListInterface.h"
00032 #include <limits>
00033
00034 using namespace std;
00035
00037 const int NUMSECKEYS = 6;
00038
00040 const int ZIPSIZE = 6;
00041
00043 const int PLACESIZE = 31;
00044
00046 const int STATESIZE = 2;
00047
00049 const int COUNTYSIZE = 36;
00050
00052 const int LATSIZE = 9;
00053
00055 const int LONSIZE = 10;
00056
00057 const int ZIPOFFSET = 0;
00058 const int PLACEOFFSET = ZIPSIZE - 1;
00059 const int STATEOFFSET = PLACEOFFSET + PLACESIZE;
00060 const int COUNTYOFFSET = STATEOFFSET + STATESIZE;
00061 const int LATOFFSET = COUNTYOFFSET + COUNTYSIZE;
00062 const int LONOFFSET = LATOFFSET + LATSIZE;
00063 const int CHARINLINE = LONOFFSET + LONSIZE;
00064
00065 class SSClass
00066 {
00067 private:
00068     unsigned numLinesIndex;
00069     unsigned numRecords;
00070     int nextEmpty;
00071     //int will be the zipcode location (RRN) The first LinkedList is a list of different sec key values
00072     LinkedList<SecKeySS<int>> secKeyZip;
00073     LinkedList<SecKeySS<string>> secKeyPlace;
00074     LinkedList<SecKeySS<string>> secKeyState;
00075     LinkedList<SecKeySS<string>> secKeyCounty;
00076     LinkedList<SecKeySS<int>> secKeyLat;
00077     LinkedList<SecKeySS<int>> secKeyLon;
00078     fstream indexFile;
```

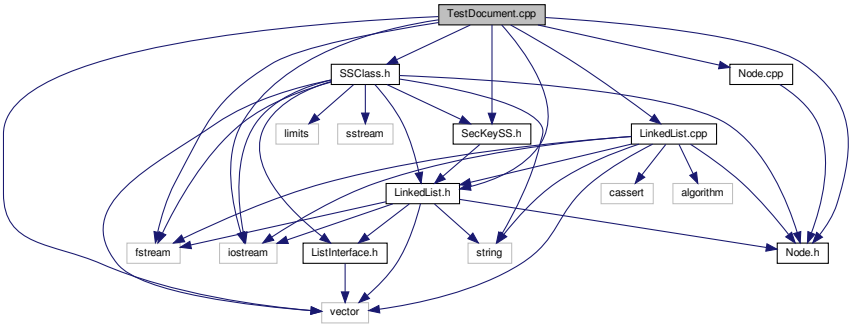
```
00079     //fstream blockRecordFile;
00080
00082
00086     void insertZip(string s, int rrn);
00087
00089
00093     void insertPlace(string s, int rrn);
00094
00096
00100     void insertState(string s, int rrn);
00101
00103
00107     void insertCounty(string s, int rrn);
00108
00110
00114     void insertLat(string s, int rrn);
00115
00117
00121     void insertLon(string s, int rrn);
00122
00123
00124     //get functions take the entire line for a record and return the specified data member
00126
00130     string getZip(string s);
00131
00133
00137     string getPlace(string s);
00138
00140
00144     string getState(string s);
00145
00147
00151     string getCounty(string s);
00152
00154
00158     string getLat(string s);
00159
00161
00165     string getLon(string s);
00166
00168
00172     void goToLine(fstream& file, unsigned num);
00173
00175
00179     void goToData(fstream& file);
00180     //bool replace(string s);
00181     //bool delete(int position);
00182
00184
00188     string createUnusedLine(int next); //creates the string needed when removing a record
00189
00191     void populate(); //populates data from text file
00192
00193 public:
00195     SSClass();
00196
00198     SSClass(const SSClass& ss);
00199
00201     ~SSClass();
00202
00204
00207     bool isEmpty() { return numRecords == 0; };
00208
00210
00215     bool openFile(string input);
00216     /*-------------------------------------------------------------------------
00217        Creates external file
00218        Preconditions:   data file
00219        Postconditions:  returns true if file location exists, otherwise returns false */
00220     //bool createIndexFile();
00221     /*-------------------------------------------------------------------------
00222        Creates external file
00223        Preconditions:   data file
00224        Postconditions:  returns true if file location exists, otherwise returns false */
00225     //bool createBlockRecordFile();
00226
00228
00231     void insert(string s);
00232
00234
00239     vector<int> search(string s, unsigned fieldNum);
00240
00242
00247     int directionalSearch(string state, char direction);
00248
00250
00254     string returnLine(int rrn);
```

```
00255 };
00256
00257
00258 #endif
```

## 6.21 TestDocument.cpp File Reference

```
#include <fstream>
#include <iostream>
#include "SSClass.h"
#include <vector>
#include "LinkedList.h"
#include "LinkedList.cpp"
#include "Node.h"
#include "Node.cpp"
#include "SecKeySS.h"
```

Include dependency graph for TestDocument.cpp:




### Functions

- void menu (uint8_t &)
- int main ()

### 6.21.1 Function Documentation

#### 6.21.1.1 main()

```
int main ( )
```

Definition at line 17 of file TestDocument.cpp.

#### 6.21.1.2 menu()

```
void menu (
            uint8_t & menuSelection )
```

Definition at line 57 of file TestDocument.cpp.

## 6.22 TestDocument.cpp

```
00001 #include <fstream>
00002 #include <iostream>
00003 #include "SSClass.h"
00004 #include <vector>
00005
00006 #include "LinkedList.h"
00007 #include "LinkedList.cpp"
00008 #include "Node.h"
00009 #include "Node.cpp"
00010 #include "SecKeySS.h"
00011
00012 using namespace std;
00013
00014 // Function prototype for menu
00015 void menu(uint8_t &);
00016
00017 int main()
00018 {
00019
00020     uint8_t menuSelection = 9;  // allocates the memory for menu selection, and innitile to 0 to display
      menu
00021
00022     cout << "\n\nWelcome to CSCI 331 SS Class Program: " << endl;   // welcom message
00023
00024     // do while loop to call menu after every selection, and display menu when selection is 0
00025     do
00026     {
00027         if(menuSelection == 9)
00028         {
00029             //Menu that will be displayed to the user.
00030             cout << endl << endl;
00031             cout << "*********************************************************" << endl;
00032             cout << "*                            Menu\t\t\t*" << endl;
00033             cout << "*\t0. \tMenu  \t\t\t\t\t*" << endl;
00034             cout << "*\t1. \tOpen file\t\t\t*" << endl;
00035             cout << "*\t2. \tInsert  \t\t\t\t*" << endl;
00036             cout << "*\t3. \tRemove \t\t\t\t**" << endl;
00037             cout << "*\t4. \tModify \t\t\t\t**" << endl;
00038             cout << "*\t5. \tDisplay recrod \t\t\t\t**" << endl;
00039             cout << "*\t6. \tDisplay feild in recrod\t\t\t\t*" << endl;
00040             cout << "*\t7. \tVerify \t\t\t\t*" << endl;
00041             cout << "*\t8. \tRun Test Sequence\t\t\t\t*" << endl;
00042             cout << "*\t9. \tSearch state \t\t\t\t*" << endl;
00043             cout << "*\t10. \tQuit\t\t\t\t*" << endl;
00044             cout << "*********************************************************" << endl;
00045             cout << endl << endl;
00046         }
00047
00048         menu(menuSelection);    // Calls menu function and passes menu selection
00049
00050     }while(menuSelection != 10 );   // if selection is 10 quit the program
00051
00052     cout << "Thank you for using SS Class program. Have a great day!!\n" << endl;   //good bye message
00053
00054     return 0;
00055 }
00056
00057 void menu(uint8_t &menuSelection)
00058 {
00059     SSClass sequence;
00060     string file_name;   // allocates memory for file name
00061     char direction;
00062     char state;
00063     string zipCode;
00064     vector rrnVector;
00065
00066     cout << "Menu Selection: ";     // message to user
00067     cin >> menuSelection;           // takes in user input for menuSelection
00068     cout << endl;
```

```
00069
00070     switch(menuSelection)
00071     {
00072         case 0:
00073                 break;
00074
00075         case 1: cout << "Enter a file name: ";
00076                 cin >> file_name; cout << endl;
00077                 //sequence.openFile(file_name); // TODO should param be a string or char* []?
00078                 break;
00079
00080         case 2: cout << "Insert: ";
00081                 //cin >> temp; cout << endl;
00082                 //list.insert();    // TODO add param(s)
00083                 break;
00084
00085         case 3: cout << "remove: ";
00086                 //cin >> temp; cout << endl;
00087                 //list.remove();    // TODO add param(s)
00088                 break;
00089
00090         case 4: // TODO call modify function
00091                 break;
00092
00093         case 5: // TODO call Display recrod function
00094                 break;
00095
00096         case 6: // TODO call Display feild in recrod function
00097             cout << "Display field in record\n";
00098             cout << "What is the zip code you would like to know the state and place name of?:"
00099                 cin << zipCode; cout << endl;
00100             rrnVector = sequence.search(zipCode, 1);
00101             for (int i = 0; i < rrnVector.size(); i++) {
00102                 cout << sequence.returnLine(rrnVector[i]);
00103             }
00104
00105                 break;
00106
00107         case 7: // TODO call Verify function
00108                 break;
00109
00110         case 8: // TODO call Run Test Sequence function
00111                 break;
00112
00113         case 9: // TODO call Search state function
00114                 cout << "enter direction N, E, S, or W: ";
00115                 cin >> direction; cout << endl;
00116                 cout << "enter state";
00117                 cin >> state; cout << endl;
00118
00119                 if (direction == 'N' || direction == 'S' || direction == 'E' || direction == 'W')
00120                 {
00121                     cout << sequence.returnLine(sequence.
      directionalSearch(state, direction));
00122                 }
00123
00124                 else
00125                 {
00126                     cout << direction << " is not a valid response. please try again. " << endl;
00127                 }
00128                 break;
00129
00130         case 10: // quit
00131                 break;
00132
00133         default: cout << "****Please make a valid menu selection. ****" << endl << endl;
00134                 break;
00135     }
00136 }
```

# Index