

My Project

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	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	BTreeNode< T > Struct Template Reference	9
5.1.1	Member Data Documentation	9
5.1.1.1	child_ptr	10
5.1.1.2	data	10
5.1.1.3	leaf	10
5.1.1.4	n	10
5.2	LinkedList< ItemType > Class Template Reference	10
5.2.1	Detailed Description	13
5.2.2	Constructor & Destructor Documentation	13
5.2.2.1	LinkedList() [1/2]	13
5.2.2.2	LinkedList() [2/2]	13
5.2.2.3	~LinkedList()	14
5.2.3	Member Function Documentation	14

5.2.3.1	<code>clear()</code>	14
5.2.3.2	<code>deletion()</code>	14
5.2.3.3	<code>displayList()</code>	15
5.2.3.4	<code>getEntry()</code>	15
5.2.3.5	<code>getItemCount()</code>	16
5.2.3.6	<code>getLength()</code>	16
5.2.3.7	<code>insert()</code>	16
5.2.3.8	<code>isEmpty()</code>	17
5.2.3.9	<code>operator=()</code>	17
5.2.3.10	<code>replace()</code>	17
5.3	<code>ListInterface< ItemType > Class Template Reference</code>	18
5.3.1	Member Function Documentation	19
5.3.1.1	<code>clear()</code>	19
5.3.1.2	<code>deletion()</code>	20
5.3.1.3	<code>displayList()</code>	20
5.3.1.4	<code>getEntry()</code>	20
5.3.1.5	<code>getItemCount()</code>	21
5.3.1.6	<code>getLength()</code>	21
5.3.1.7	<code>insert()</code>	21
5.3.1.8	<code>isEmpty()</code>	22
5.3.1.9	<code>replace()</code>	22
5.4	<code>Node< ItemType > Class Template Reference</code>	23
5.4.1	Detailed Description	24
5.4.2	Constructor & Destructor Documentation	24
5.4.2.1	<code>Node()</code> [1/3]	24
5.4.2.2	<code>Node()</code> [2/3]	24
5.4.2.3	<code>Node()</code> [3/3]	25
5.4.3	Member Function Documentation	25
5.4.3.1	<code>getItem()</code>	25
5.4.3.2	<code>getNext()</code>	25

5.4.3.3	setItem()	25
5.4.3.4	setNext()	26
5.5	SecKeySS< T > Class Template Reference	26
5.5.1	Detailed Description	28
5.5.2	Constructor & Destructor Documentation	29
5.5.2.1	SecKeySS() [1/2]	29
5.5.2.2	SecKeySS() [2/2]	29
5.5.2.3	~SecKeySS()	29
5.5.3	Member Function Documentation	29
5.5.3.1	getData()	29
5.5.3.2	getDuplicates()	30
5.5.3.3	operator<() [1/2]	30
5.5.3.4	operator<() [2/2]	30
5.5.3.5	operator=()	31
5.5.3.6	operator==([1/2]	31
5.5.3.7	operator==([2/2]	31
5.5.3.8	operator>() [1/2]	32
5.5.3.9	operator>() [2/2]	32
5.5.3.10	setData()	32
5.5.3.11	setDuplicates()	33
5.6	SSClass Class Reference	33
5.6.1	Detailed Description	34
5.6.2	Constructor & Destructor Documentation	34
5.6.2.1	SSClass() [1/2]	34
5.6.2.2	SSClass() [2/2]	35
5.6.2.3	~SSClass()	35
5.6.3	Member Function Documentation	35
5.6.3.1	directionalSearch()	35
5.6.3.2	insert()	35
5.6.3.3	isEmpty()	36
5.6.3.4	openFile()	36
5.6.3.5	returnLine()	36
5.6.3.6	search()	37

6 File Documentation	39
6.1 BTree.h File Reference	39
6.1.1 Function Documentation	40
6.1.1.1 init()	40
6.1.1.2 insert()	40
6.1.1.3 sort()	40
6.1.1.4 split_child()	40
6.1.1.5 traverse()	40
6.1.2 Variable Documentation	41
6.1.2.1 np	41
6.1.2.2 root	41
6.1.2.3 x	41
6.2 LinkedList.cpp File Reference	41
6.3 LinkedList.h File Reference	42
6.4 ListInterface.h File Reference	43
6.5 Node.cpp File Reference	44
6.6 Node.h File Reference	45
6.7 README.md File Reference	46
6.8 SecKeySS.h File Reference	46
6.8.1 Function Documentation	47
6.8.1.1 operator<()	47
6.8.1.2 operator==()	47
6.8.1.3 operator>()	47
6.9 SSClass.cpp File Reference	48
6.10 SSClass.h File Reference	48
6.10.1 Variable Documentation	50
6.10.1.1 CHARINLINE	50
6.10.1.2 COUNTYOFFSET	50
6.10.1.3 COUNTYSIZE	50
6.10.1.4 LATOFFSET	50

6.10.1.5	LATSIZE	51
6.10.1.6	LONOFFSET	51
6.10.1.7	LONSIZE	51
6.10.1.8	NUMSECKEYS	51
6.10.1.9	PLACEOFFSET	51
6.10.1.10	PLACESIZE	51
6.10.1.11	STATEOFFSET	51
6.10.1.12	STATESIZE	52
6.10.1.13	ZIPOFFSET	52
6.10.1.14	ZIPSIZE	52
6.11	TestDocument.cpp File Reference	52
6.11.1	Function Documentation	53
6.11.1.1	main()	53
6.11.1.2	menu()	53
Index		55

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:

BTreeNode< T >	9
ListInterface< ItemType >	18
LinkedList< ItemType >	10
ListInterface< int >	18
LinkedList< int >	10
ListInterface< SecKeySS< int > >	18
LinkedList< SecKeySS< int > >	10
ListInterface< SecKeySS< string > >	18
LinkedList< SecKeySS< string > >	10
ListInterface< string >	18
LinkedList< string >	10
ListInterface< T >	18
LinkedList< T >	10
Node< ItemType >	23
Node< int >	23
Node< SecKeySS< int > >	23
Node< SecKeySS< string > >	23
Node< string >	23
Node< T >	23
SecKeySS< T >	26
SecKeySS< int >	26
SecKeySS< string >	26
SSClass	33

Chapter 3

Class Index

3.1 Class List

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

BTreeNode< T >	9
LinkedList< ItemType >	
This is LinkedList class creating a list of linked nodes	10
ListInterface< ItemType >	18
Node< ItemType >	
This is Node class for linked list	23
SecKeySS< T >	
This is the class for Section Keys of the SS class	26
SSClass	
LinkedList integration for blocks, records, and fields	33

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

BTree.h	39
LinkedList.cpp	41
LinkedList.h	42
ListInterface.h	43
Node.cpp	44
Node.h	45
SecKeySS.h	46
SSClass.cpp	48
SSClass.h	48
TestDocument.cpp	52

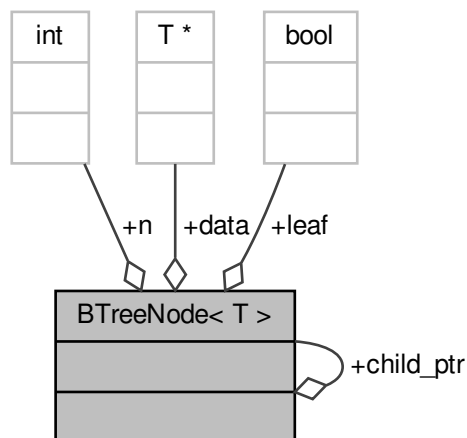
Chapter 5

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 Member Data Documentation

5.1.1.1 child_ptr

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

5.1.1.2 data

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

5.1.1.3 leaf

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

5.1.1.4 n

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

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

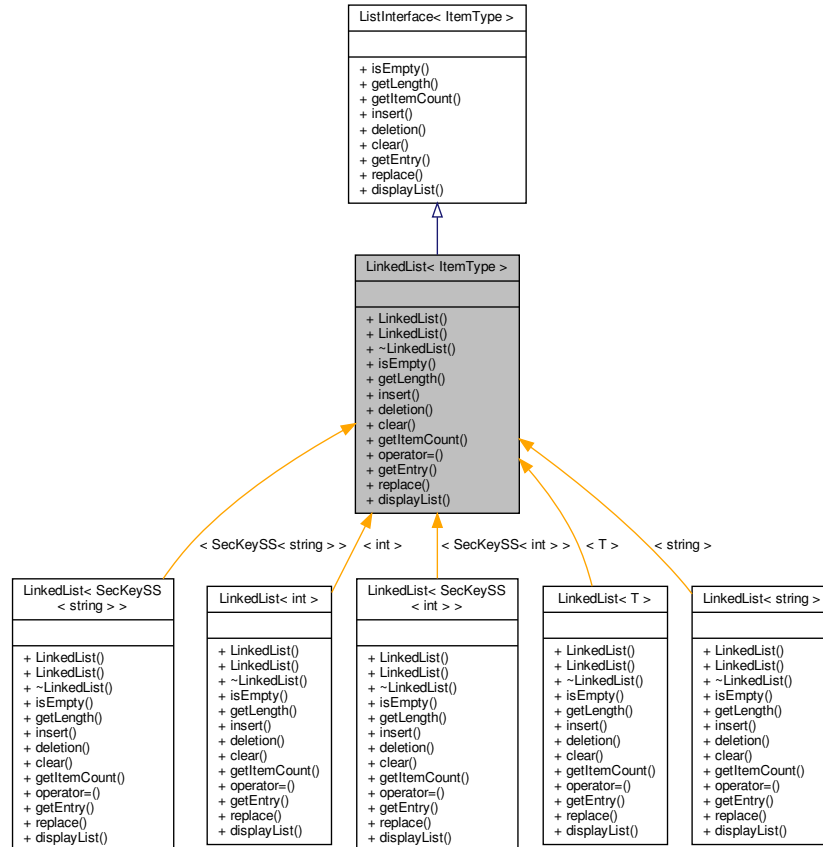
- [BTree.h](#)

5.2 LinkedList< ItemType > Class Template Reference

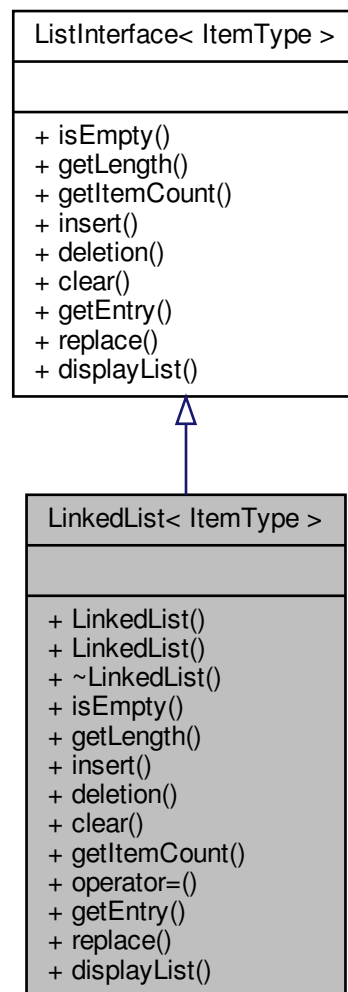
This is [LinkedList](#) class creating a list of linked nodes.

```
#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 destructor.
- bool [isEmpty \(\)](#) const
Member 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)
Member 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` ()
Member Function 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
Member 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.

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.

5.2.2.2 `LinkedList()` [2/2]

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

`LinkedList` constructor.

A copy constructor with one argument passed, `aList`.

Parameters

<i>aList</i>	a reference to a list
--------------	-----------------------

5.2.2.3 ~LinkedList()

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

[LinkedList](#) destructor.

A destructor to clear a [LinkedList](#)

5.2.3 Member Function Documentation**5.2.3.1 clear()**

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

Member Function to clear a [LinkedList](#).

Removes 1 [Node](#) at a time while the [LinkedList](#) is not Empty

Implements [ListInterface< ItemType >](#).

5.2.3.2 deletion()

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

Member function for deletion of a [Node](#).

Parameters

<i>position</i>	the position of the Node to be removed
-----------------	--

Returns

`ableToRemove` returns true if the [Node](#) is a valid [Node](#).

Precondition

To be a valid `Node` to remove, position ≥ 1 and position \leq itemCount

Implements `ListInterface< ItemType >`.

5.2.3.3 displayList()

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

Member function to display the list.

Displays the list by returning one `Node` item at a time

Returns

nodePtr->getItem() an item at a node

Implements `ListInterface< ItemType >`.

5.2.3.4 getEntry()

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

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

Exceptions

<i>PrecondViolatedExcep</i>	if position < 1 or position $>$ <code>getLength()</code> .
-----------------------------	--

Parameters

<i>position</i>	the position of a <code>Node</code> to return an item
-----------------	---

Returns

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

Precondition

position > 0 and position \leq itemCount

Implements `ListInterface< ItemType >`.

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 >](#).

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 >](#).

5.2.3.7 insert()

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

Member function to insert a new item into a [Node](#) of a [LinkedList](#).

Parameters

<i>newPosition</i>	a node position to insert a item into
<i>newEntry</i>	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 >](#).

5.2.3.8 `isEmpty()`

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

Member 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 >](#).

5.2.3.9 `operator=()`

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

operator function =

Parameters

<i>rhs</i>	reference to a LinkedList
------------	---

Returns

*this a pointer to the [LinkedList](#)

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

<i>PrecondViolatedExcep</i>	if position < 1 or position > getLength() .
-----------------------------	---

Parameters

<i>position</i>	the position of the Node whos item will be replaced
<i>newEntry</i>	the new entry to replace the old entry of a Node

Implements [ListInterface< ItemType >](#).

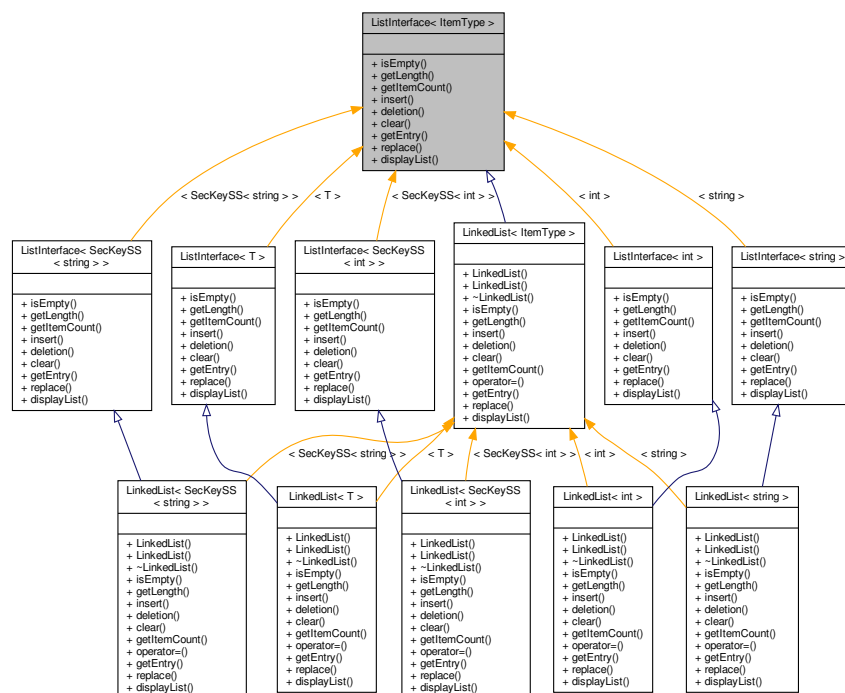
The documentation for this class was generated from the following files:

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

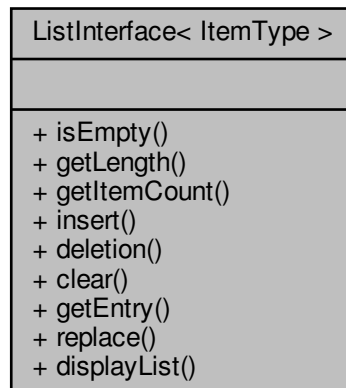
5.3 ListInterface< ItemType > Class Template Reference

```
#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 Member Function Documentation

5.3.1.1 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 >](#).

5.3.1.2 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< SecKeySS< string > >](#), [LinkedList< int >](#), [LinkedList< SecKeySS< int > >](#), [LinkedList< T >](#), and [LinkedList< string >](#).

5.3.1.3 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 >](#).

5.3.1.4 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

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

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

5.3.1.5 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 >](#).

5.3.1.6 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 >](#).

5.3.1.7 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 $1 \leq \text{position} \leq \text{getLength}() + 1$ 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< SecKeySS< string > >](#), [LinkedList< int >](#), [LinkedList< SecKeySS< int > >](#), [LinkedList< T >](#), and [LinkedList< string >](#).

5.3.1.8 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 >](#).

5.3.1.9 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

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

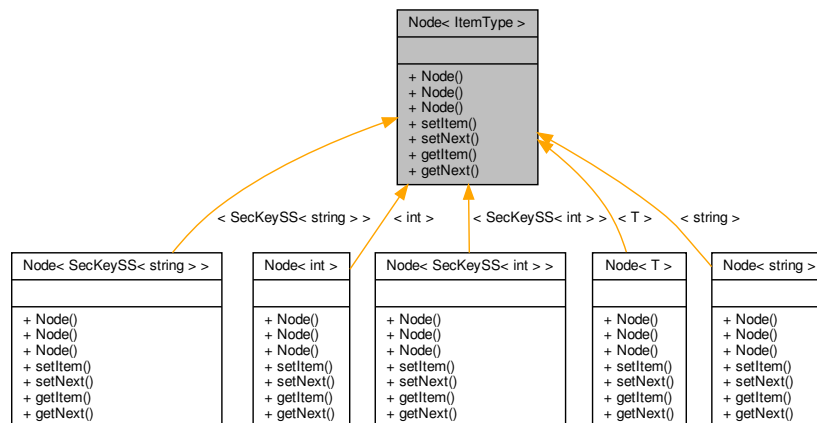
- [ListInterface.h](#)

5.4 Node< ItemType > Class Template Reference

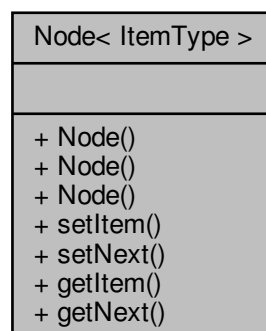
This is [Node](#) class for linked list.

```
#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.

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

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

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

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

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

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

5.4.3.2 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.

5.4.3.3 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
---------------	----------------------------

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

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

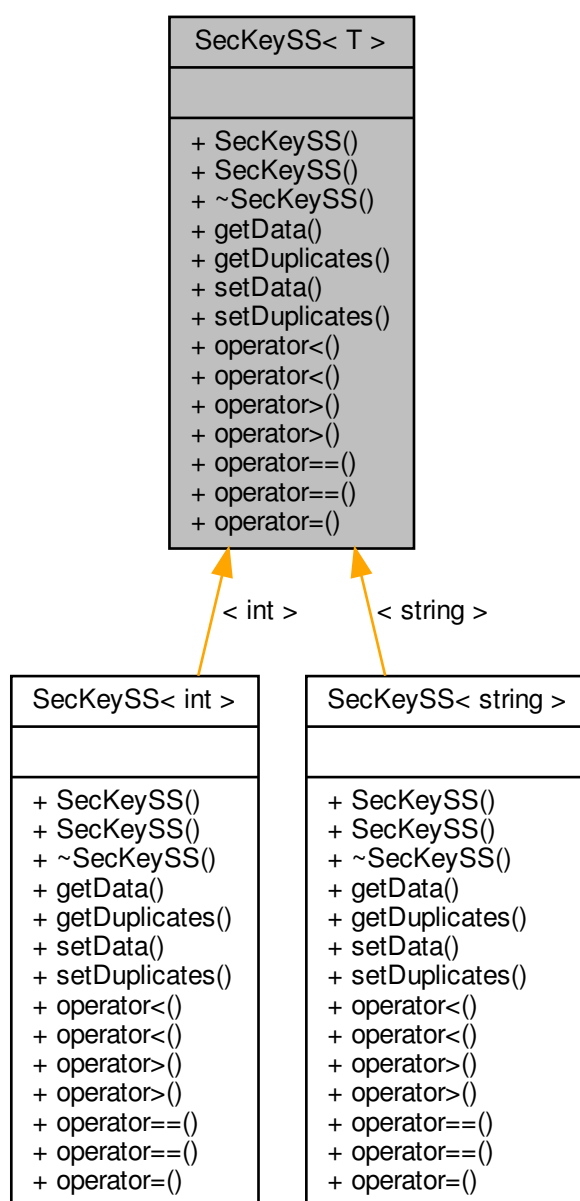
- [Node.h](#)
- [Node.cpp](#)

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.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 SecKeySS() [1/2]

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

Default constructor

5.5.2.2 SecKeySS() [2/2]

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

Copy Constructor

5.5.2.3 ~SecKeySS()

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

Deconstructor

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

5.5.3.2 getDuplicates()

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

Gets duplicates

Returns

LinkedList of itemType

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

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

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
---	---------------------------

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

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

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

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

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
---	-------------------------

5.5.3.11 setDuplicates()

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

Sets duplicates

Parameters

LinkedList	dup
----------------------------	-----

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

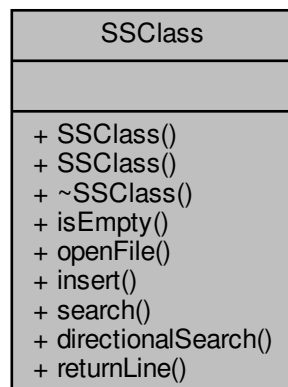
- [SecKeySS.h](#)

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](#) ()
Destructor.
- 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 rn)
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, latitude, 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

5.6.2 Constructor & Destructor Documentation

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

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

Default constructor.

5.6.2.2 SSClass() [2/2]

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

Constructor.

5.6.2.3 ~SSClass()

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

Deconstructor.

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

<i>state</i>	the state to search "MN" for example
<i>direction</i>	(N, S, W, E)

Returns

the line contating the soght after direction

5.6.3.2 insert()

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

inserts line by line into data

Parameters

<i>s</i>	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

5.6.3.3 isEmpty()

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

Check if numRecords is 0.

Returns

returns false if empty, otherwise returns true

5.6.3.4 openFile()

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

Opens external file.

Parameters

<i>input</i>	string
--------------	--------

Precondition

data file

Returns

true if file location exists, otherwise returns false

5.6.3.5 returnLine()

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

Fills secondary key vector.

Parameters

<i>rrn</i>	and integer refring to the line to get
------------	--

Returns

string containging the contents of the line

5.6.3.6 search()

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

Searches for record.

Parameters

<i>s</i>	strign to search for fieldNum the field in whitch to search
----------	---

Returns

vector of results

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

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

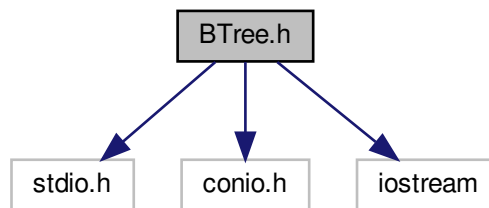
Chapter 6

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 ( )
```

6.1.1.2 `insert()`

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

6.1.1.3 `sort()`

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

6.1.1.4 `split_child()`

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

6.1.1.5 `traverse()`

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


6.1.2 Variable Documentation

6.1.2.1 np

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

6.1.2.2 root

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

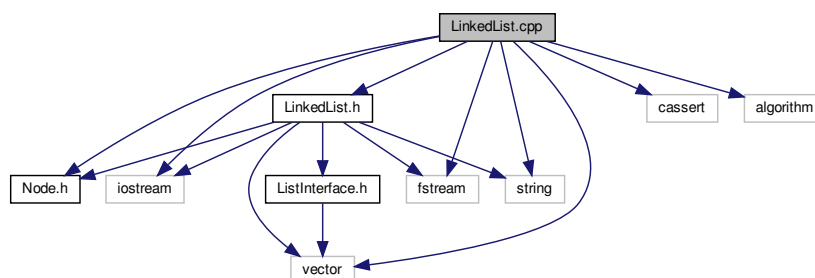
6.1.2.3 x

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

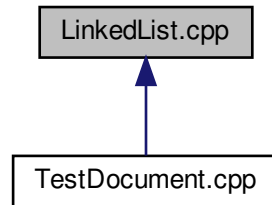
6.2 LinkedList.cpp File Reference

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

Include dependency graph for LinkedList.cpp:



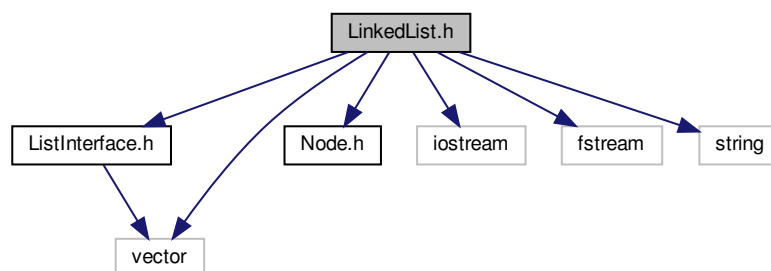
This graph shows which files directly or indirectly include this file:



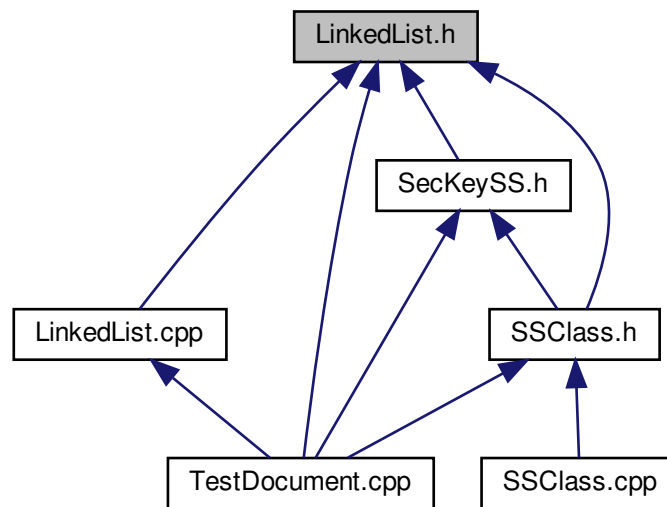
6.3 LinkedList.h File Reference

```
#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

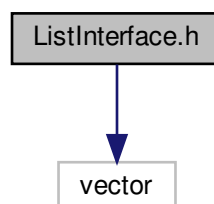
- class `LinkedList< ItemType >`

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

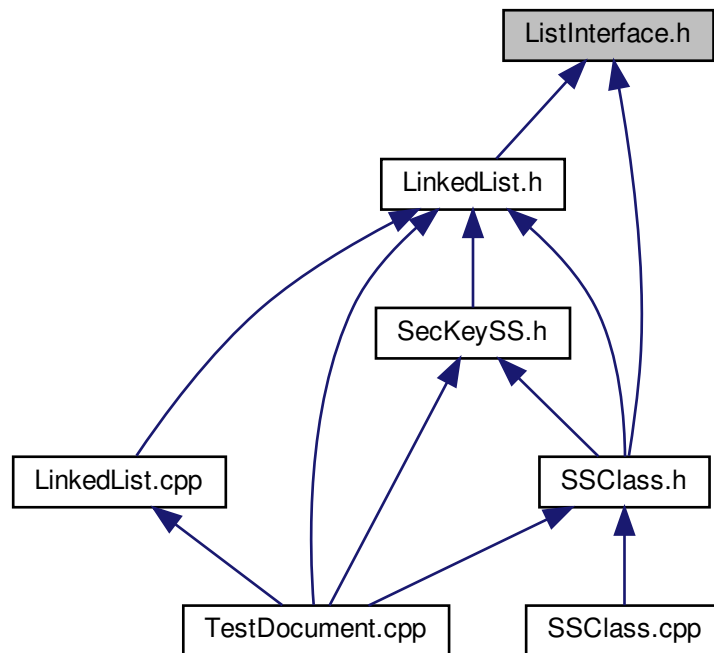
6.4 ListInterface.h File Reference

```
#include <vector>
```

Include dependency graph for `ListInterface.h`:



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



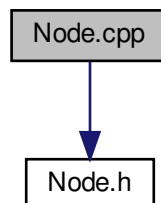
Classes

- class [ListInterface< ItemType >](#)

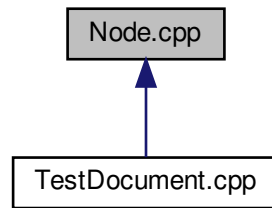
6.5 Node.cpp File Reference

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

Include dependency graph for `Node.cpp`:

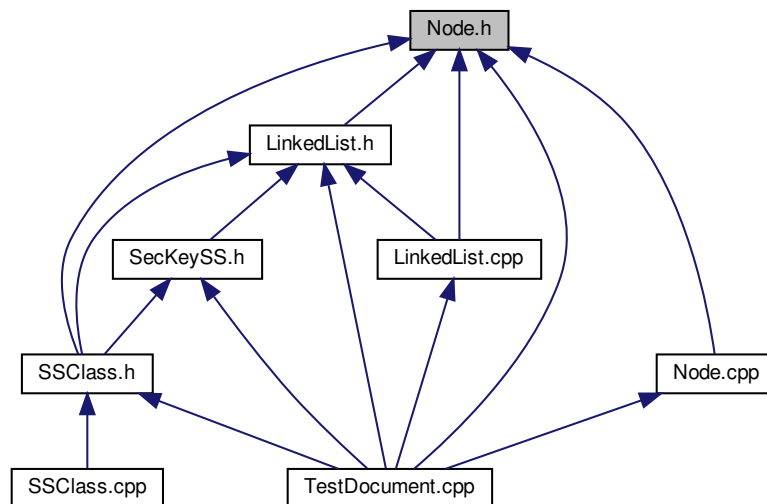


This graph shows which files directly or indirectly include this file:



6.6 Node.h File Reference

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



Classes

- class `Node< ItemType >`

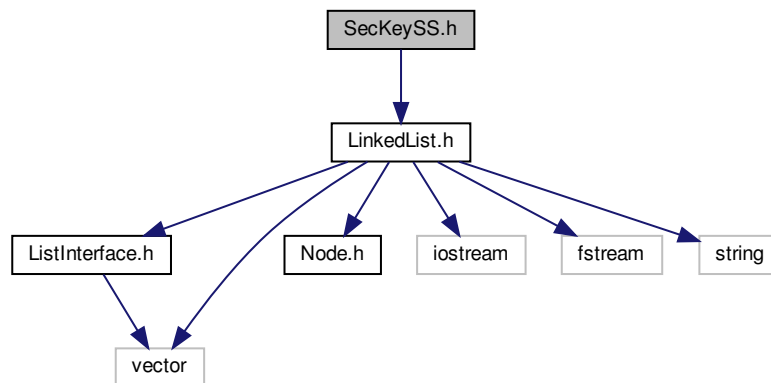
This is `Node` class for linked list.

6.7 README.md File Reference

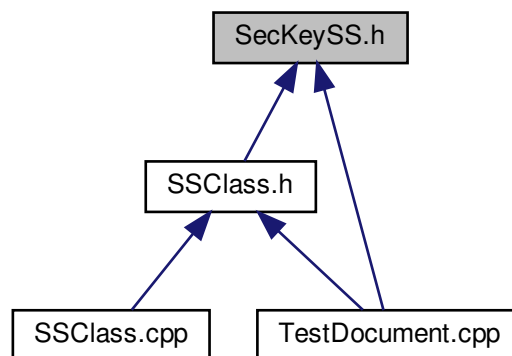
6.8 SecKeySS.h File Reference

```
#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)`

6.8.1 Function Documentation

6.8.1.1 operator<()

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

6.8.1.2 operator==()

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

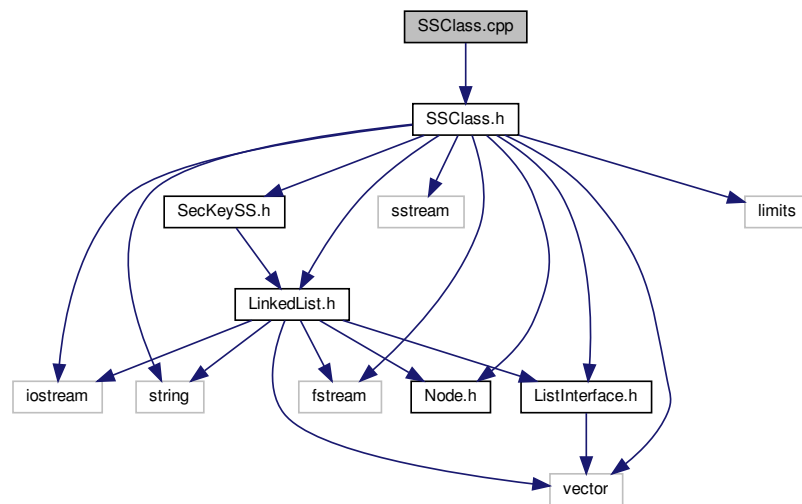
6.8.1.3 operator>()

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

6.9 SSClass.cpp File Reference

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

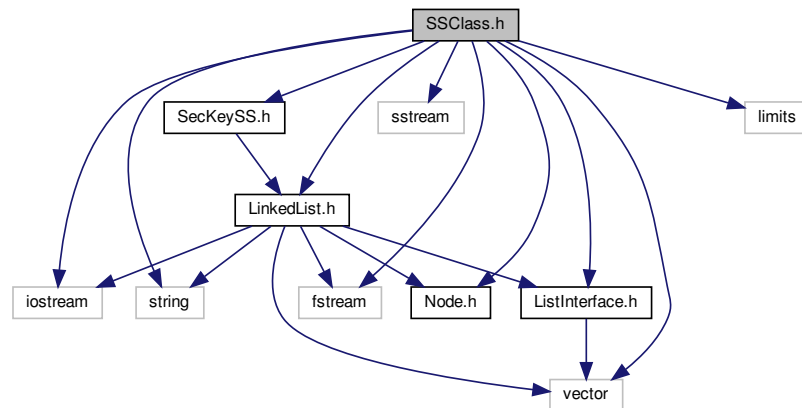
Include dependency graph for SSClass.cpp:



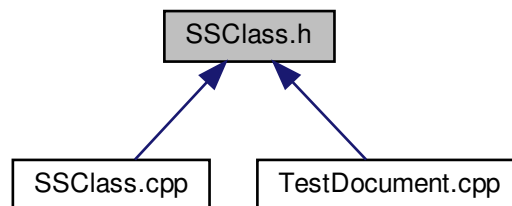
6.10 SSClass.h File Reference

```
#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`

6.10.1 Variable Documentation

6.10.1.1 CHARINLINE

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

6.10.1.2 COUNTYOFFSET

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

6.10.1.3 COUNTYSIZE

```
const int COUNTYSIZE = 36
```

COUNTYSIZE The size of letters for the county.

6.10.1.4 LATOFFSET

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

6.10.1.5 LATSIZE

```
const int LATSIZE = 9
```

LATSIZE The size of the Lattatude.

6.10.1.6 LONOFFSET

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

6.10.1.7 LONSIZE

```
const int LONSIZE = 10
```

LONSIZE The size (including sign) of the longitude.

6.10.1.8 NUMSECKEYS

```
const int NUMSECKEYS = 6
```

NUMSECKEYS The numebr of section keys.

6.10.1.9 PLACEOFFSET

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

6.10.1.10 PLACESIZE

```
const int PLACESIZE = 31
```

PLACESIZE The size of the place (city)

6.10.1.11 STATEOFFSET

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

6.10.1.12 STATESIZE

```
const int STATESIZE = 2
```

STATESIZE The size of the sate letters.

6.10.1.13 ZIPOFFSET

```
const int ZIPOFFSET = 0
```

6.10.1.14 ZIPSIZE

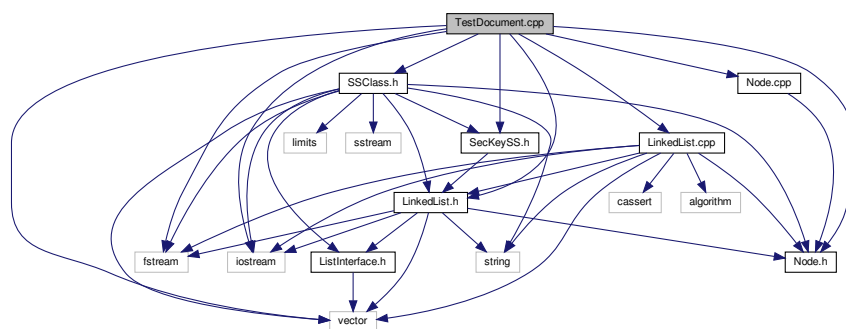
```
const int ZIPSIZE = 6
```

ZIPSIZE The size of the zip code.

6.11 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](#) ()

6.11.1 Function Documentation

6.11.1.1 main()

```
int main ( )
```

6.11.1.2 menu()

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


Index

- ~LinkedList
 - LinkedList, 14
- ~SSClass
 - SSClass, 35
- ~SecKeySS
 - SecKeySS, 29
- BTree.h, 39
 - init, 40
 - insert, 40
 - np, 41
 - root, 41
 - sort, 40
 - split_child, 40
 - traverse, 40
 - x, 41
- BTreeNode
 - child_ptr, 9
 - data, 10
 - leaf, 10
 - n, 10
- BTreeNode< T >, 9
- CHARINLINE
 - SSClass.h, 50
- COUNTYOFFSET
 - SSClass.h, 50
- COUNTYSIZE
 - SSClass.h, 50
- child_ptr
 - BTreeNode, 9
- clear
 - LinkedList, 14
 - ListInterface, 19
- data
 - BTreeNode, 10
- deletion
 - LinkedList, 14
 - ListInterface, 19
- directionalSearch
 - SSClass, 35
- displayList
 - LinkedList, 15
 - ListInterface, 20
- getData
 - SecKeySS, 29
- getDuplicates
 - SecKeySS, 29

- getEntry
 - LinkedList, 15
 - ListInterface, 20
- getItem
 - Node, 25
- getItemCount
 - LinkedList, 15
 - ListInterface, 21
- getLength
 - LinkedList, 16
 - ListInterface, 21
- getNext
 - Node, 25
- init
 - BTree.h, 40
- insert
 - BTree.h, 40
 - LinkedList, 16
 - ListInterface, 21
 - SSClass, 35
- isEmpty
 - LinkedList, 17
 - ListInterface, 22
 - SSClass, 36
- LATOFFSET
 - SSClass.h, 50
- LATSIZE
 - SSClass.h, 50
- LONOFFSET
 - SSClass.h, 51
- LONSIZE
 - SSClass.h, 51
- leaf
 - BTreeNode, 10
- LinkedList
 - ~LinkedList, 14
 - clear, 14
 - deletion, 14
 - displayList, 15
 - getEntry, 15
 - getItemCount, 15
 - getLength, 16
 - insert, 16
 - isEmpty, 17
 - LinkedList, 13
 - operator=, 17
 - replace, 17
- LinkedList< ItemType >, 10

- LinkedList.cpp, 41
- LinkedList.h, 42
- ListInterface
 - clear, 19
 - deletion, 19
 - displayList, 20
 - getEntry, 20
 - getItemCount, 21
 - getLength, 21
 - insert, 21
 - isEmpty, 22
 - replace, 22
- ListInterface< ItemType >, 18
- ListInterface.h, 43
- main
 - TestDocument.cpp, 53
- menu
 - TestDocument.cpp, 53
- n
 - BTreeNode, 10
- NUMSECKEYS
 - SSClass.h, 51
- Node
 - getItem, 25
 - getNext, 25
 - Node, 24, 25
 - setItem, 25
 - setNext, 26
- Node< ItemType >, 23
- Node.cpp, 44
- Node.h, 45
- np
 - BTree.h, 41
- openFile
 - SSClass, 36
- operator<
 - SecKeySS.h, 47
 - SecKeySS, 30
- operator>
 - SecKeySS.h, 47
 - SecKeySS, 31, 32
- operator=
 - LinkedList, 17
 - SecKeySS, 30
- operator==
 - SecKeySS.h, 47
 - SecKeySS, 31
- PLACEOFFSET
 - SSClass.h, 51
- PLACESIZE
 - SSClass.h, 51
- README.md, 46
- replace
 - LinkedList, 17
- ListInterface, 22
- returnLine
 - SSClass, 36
- root
 - BTree.h, 41
- SSClass, 33
 - ~SSClass, 35
 - directionalSearch, 35
 - insert, 35
 - isEmpty, 36
 - openFile, 36
 - returnLine, 36
 - SSClass, 34
 - search, 37
- SSClass.cpp, 48
- SSClass.h, 48
 - CHARINLINE, 50
 - COUNTYOFFSET, 50
 - COUNTYSIZE, 50
 - LATOFFSET, 50
 - LATSIZE, 50
 - LONOFFSET, 51
 - LONSIZE, 51
 - NUMSECKEYS, 51
 - PLACEOFFSET, 51
 - PLACESIZE, 51
 - STATEOFFSET, 51
 - STATESIZE, 51
 - ZIPOFFSET, 52
 - ZIPSIZE, 52
- STATEOFFSET
 - SSClass.h, 51
- STATESIZE
 - SSClass.h, 51
- search
 - SSClass, 37
- SecKeySS< T >, 26
- SecKeySS.h, 46
 - operator<, 47
 - operator>, 47
 - operator==, 47
- SecKeySS
 - ~SecKeySS, 29
 - getData, 29
 - getDuplicates, 29
 - operator<, 30
 - operator>, 31, 32
 - operator=, 30
 - operator==, 31
 - SecKeySS, 29
 - setData, 32
 - setDuplicates, 33
- setData
 - SecKeySS, 32
- setDuplicates
 - SecKeySS, 33
- setItem
 - Node, 25

- setNext
 - Node, [26](#)
- sort
 - BTree.h, [40](#)
- split_child
 - BTree.h, [40](#)
- TestDocument.cpp, [52](#)
 - main, [53](#)
 - menu, [53](#)
- traverse
 - BTree.h, [40](#)
- x
 - BTree.h, [41](#)
- ZIPOFFSET
 - SSClass.h, [52](#)
- ZIPSIZE
 - SSClass.h, [52](#)