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:

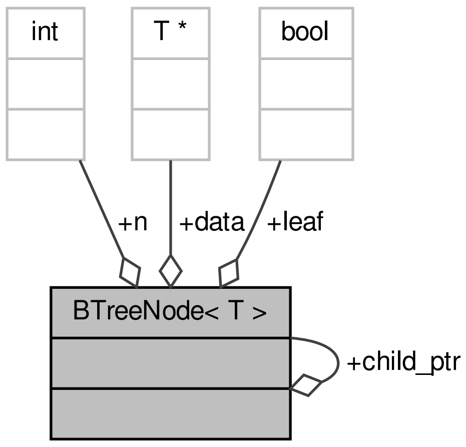
Chapter 5

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 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:

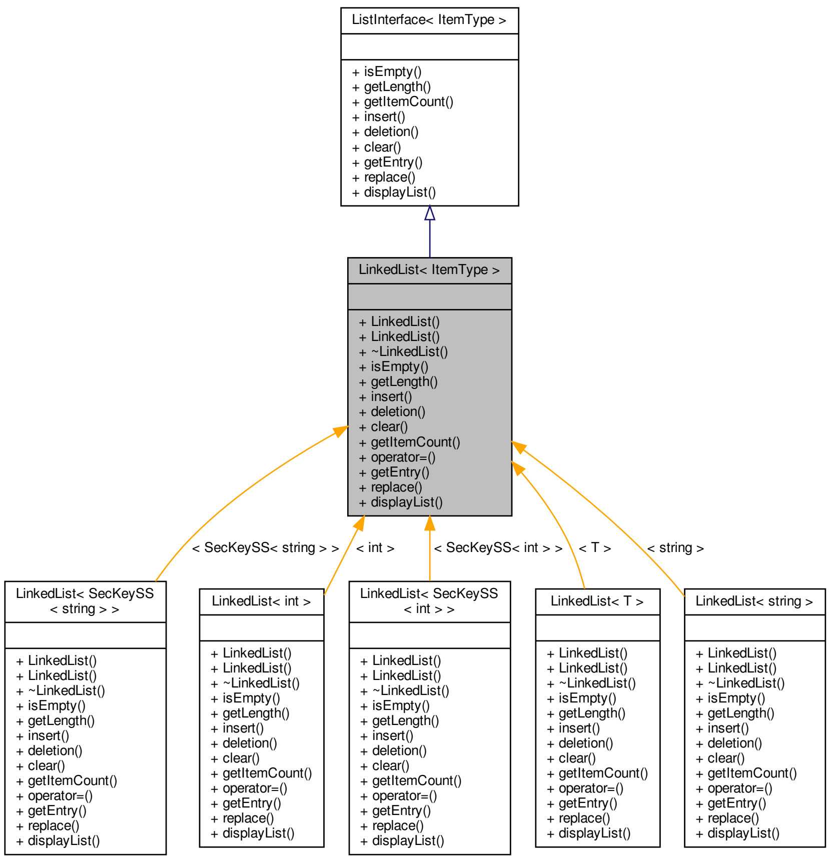
- BTree.h

5.2 LinkedList< ItemType > Class Template Reference

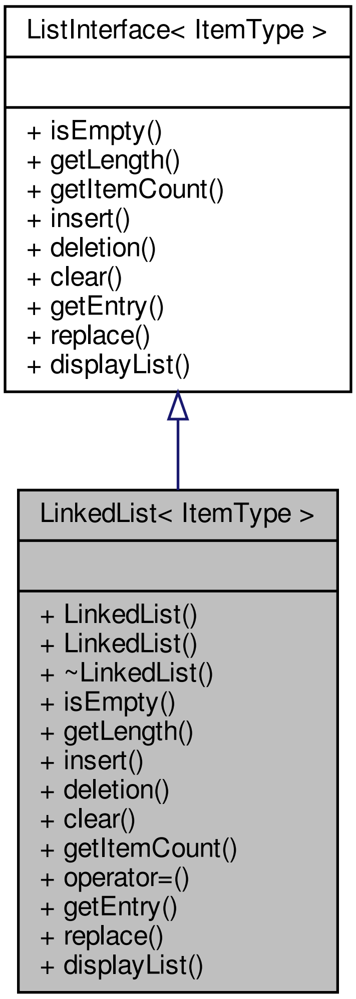
This is LinkedList class creating a list of linked nodes.

```
#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 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.

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 argumet passed, aList.

Parameters

aLsit	a refrence to a list

5.2.2.3 ∼LinkedList()

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

LinkedList deconstructor.

A deconstructor to clear a LinkedList

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 >.

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 >.

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 >.

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 >.

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]
```

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 >.

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 >.

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

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 >.

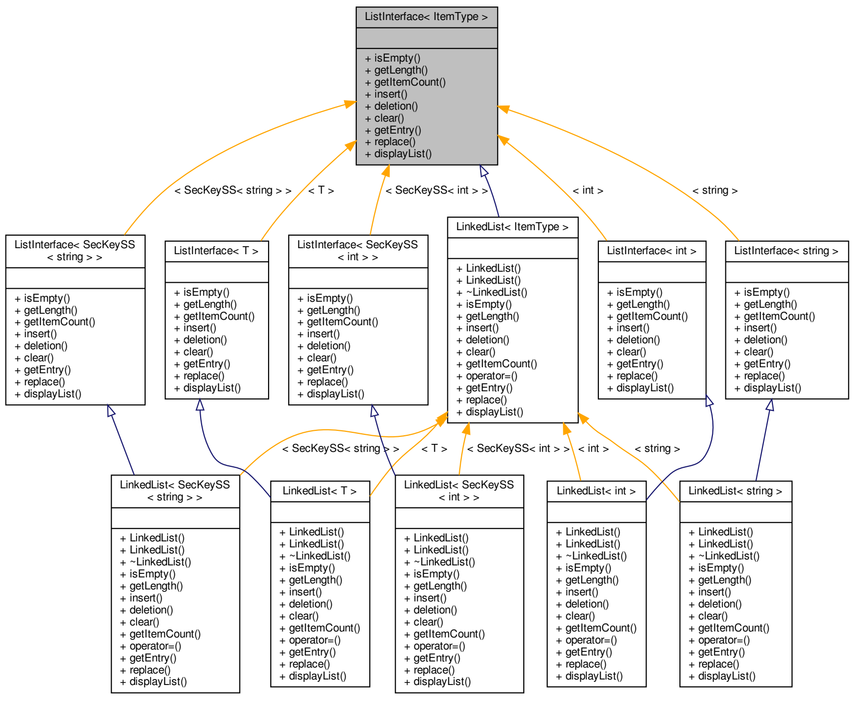
The documentation for this class was generated from the following files:

- LinkedList.h
- LinkedList.cpp

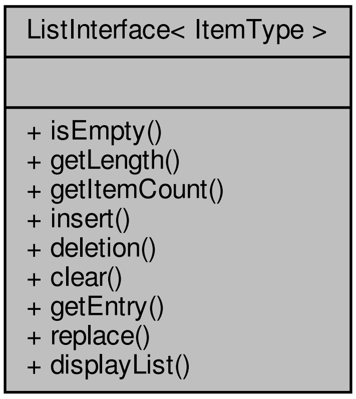
5.3 ListInterface< ItemType > Class Template Reference

```
#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 <= 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 >.

5.3.1.3 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 <= 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 >.

5.3.1.5 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 <= 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 >.

5.3.1.8 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

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:

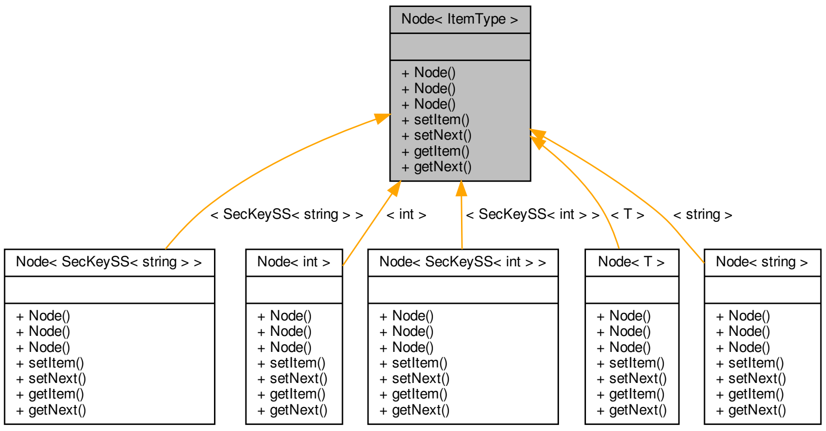
- ListInterface.h

5.4 Node< ItemType > Class Template Reference

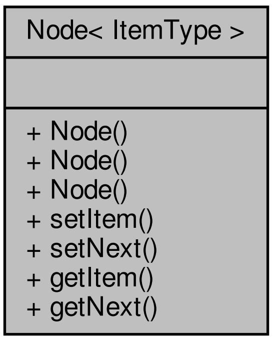
This is Node class for linked list.

```
#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.

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

anItem	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

anItem	a constant reference to an item of itemtype
nextNodePtr	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
```

Memebr funtion 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 memebr item.

Parameters

anItem	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:

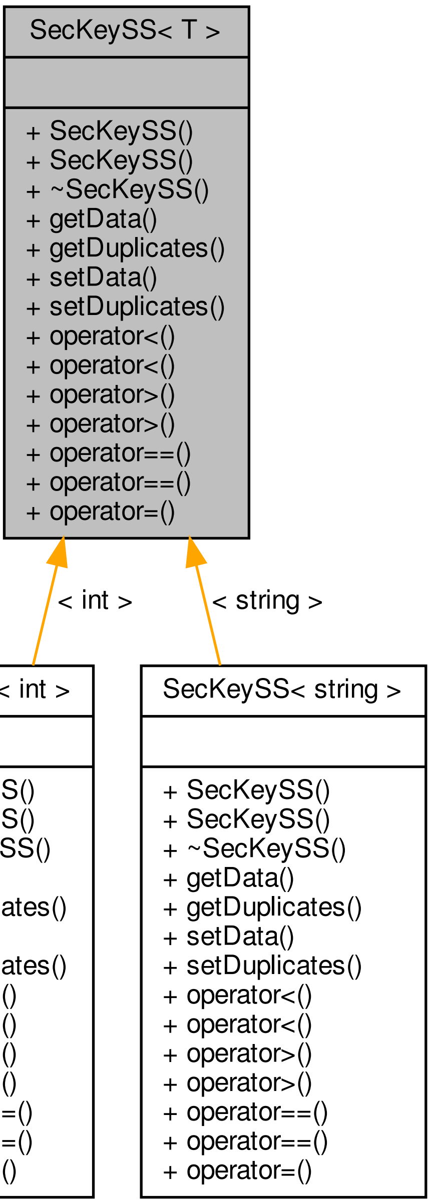
- Node.h
- Node.cpp

5.5 SecKeySS< T > Class Template Reference

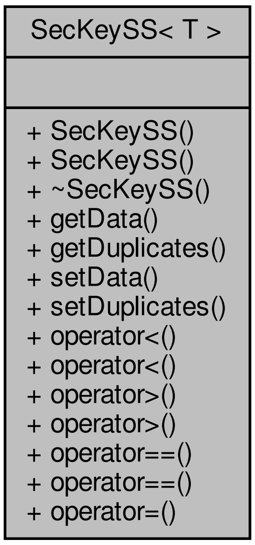
This is the class for Section Keys of the SS class.

```
#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 ( )
```

Deconstuctor

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:

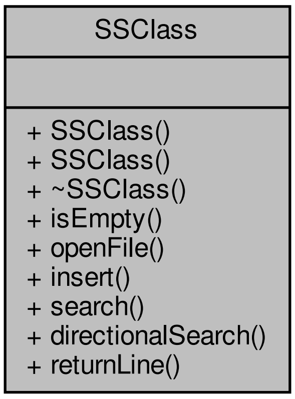
- SecKeySS.h

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 ()
  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

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

state	the state to search "MN" for example
direction	(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

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

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

input	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

rrn	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

s	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

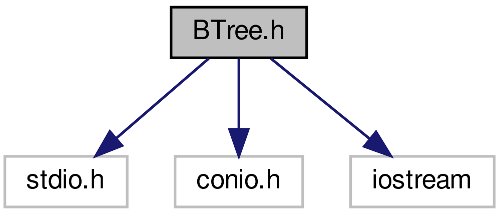
Chapter 6

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
```

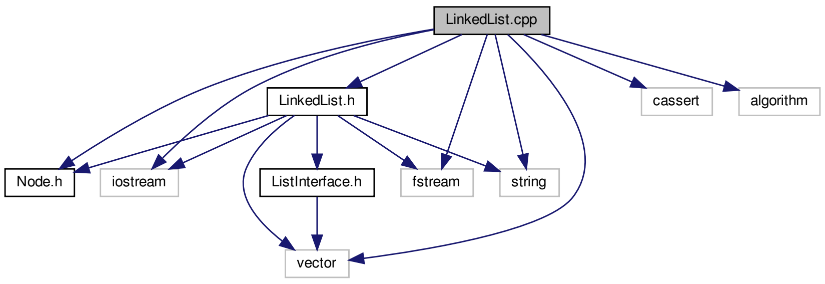
6.1.2.3 x

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

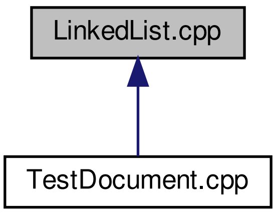
6.2 LinkedList.cpp File Reference

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

Include dependency graph for LinkedList.cpp:



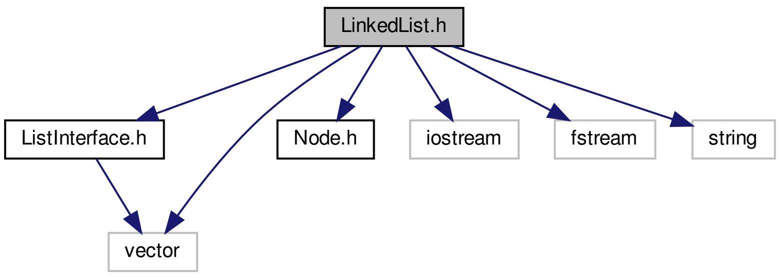
This graph shows which files directly or indirectly include this file:



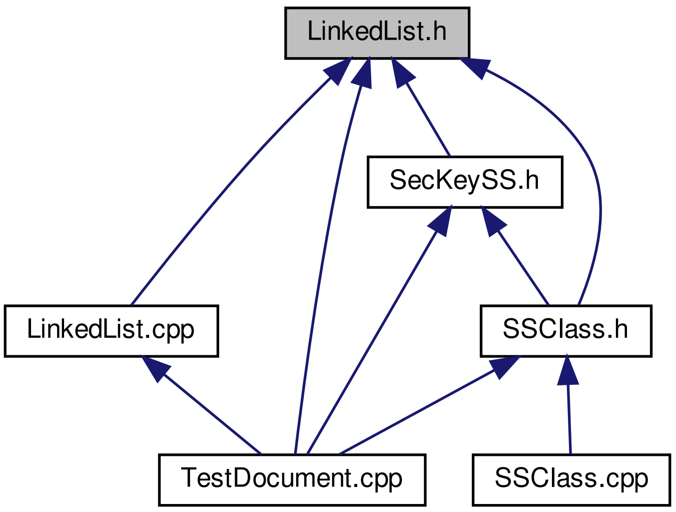
6.3 LinkedList.h File Reference

```
#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

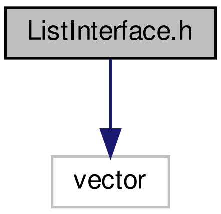
- class LinkedList< ItemType >

  This is LinkedList class creating a list of linked nodes.

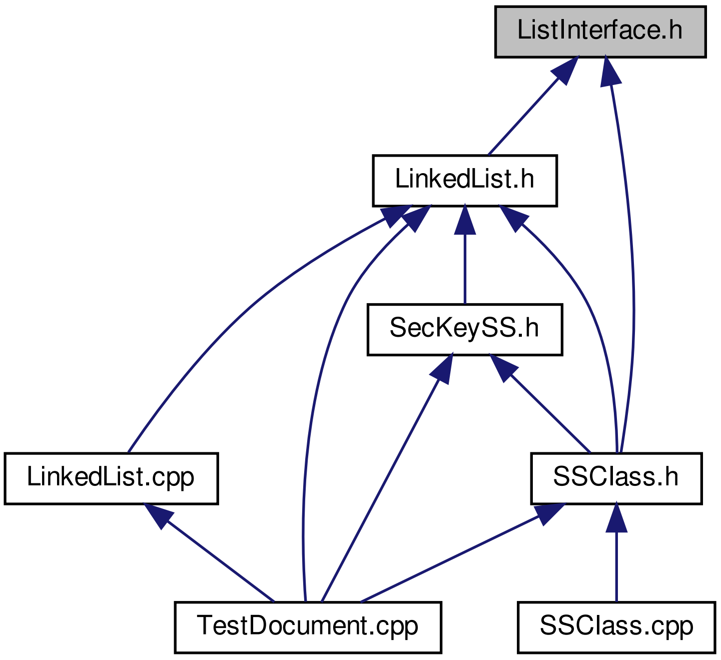
6.4 ListInterface.h File Reference

`#include <vector>`

Include dependency graph for ListInterface.h:



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



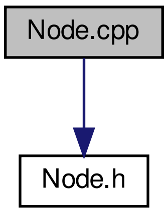
Classes

- class ListInterface< ItemType >

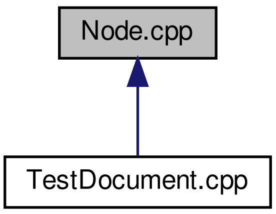
6.5 Node.cpp File Reference

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

Include dependency graph for Node.cpp:

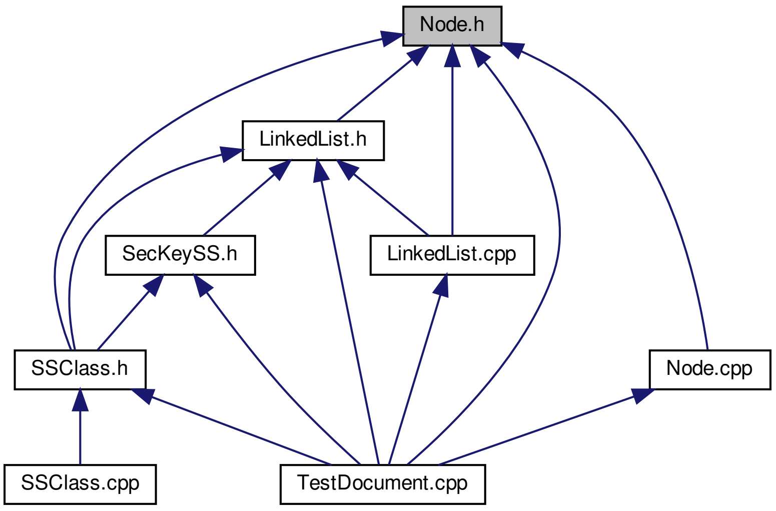


This graph shows which files directly or indirectly include this file:



6.6 Node.h File Reference

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



Classes

- class Node< ItemType >

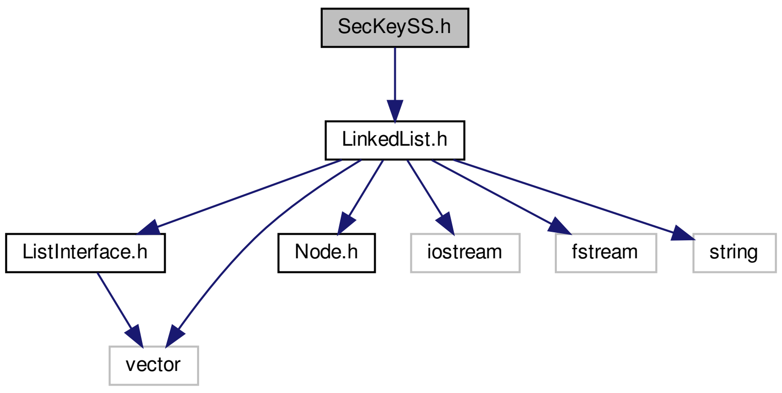
  This is Node class for linked list.

6.7 README.md File Reference

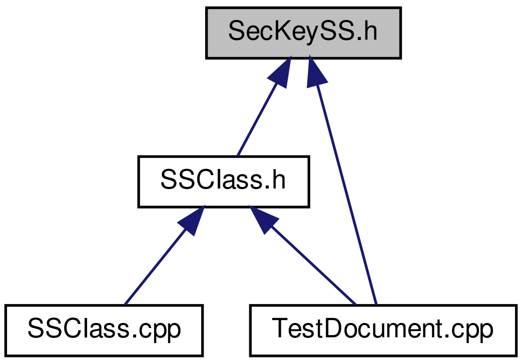
6.8 SecKeySS.h File Reference

`#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 )
```

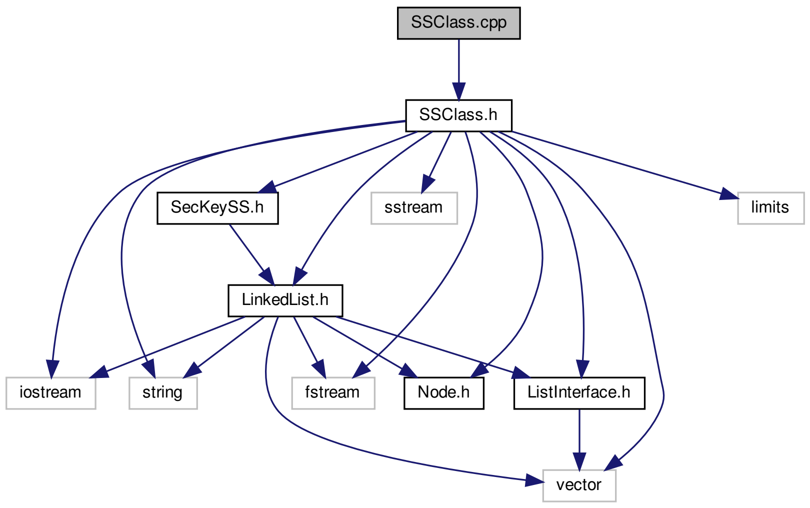
6.8.1.3 operator>()

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

6.9 SSClass.cpp File Reference

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

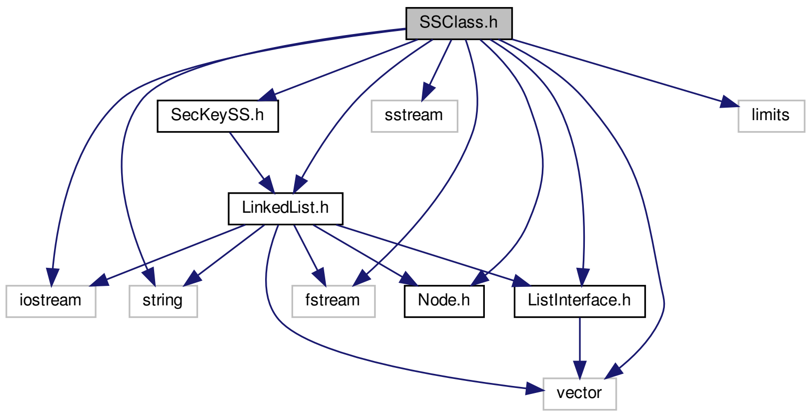
Include dependency graph for SSClass.cpp:



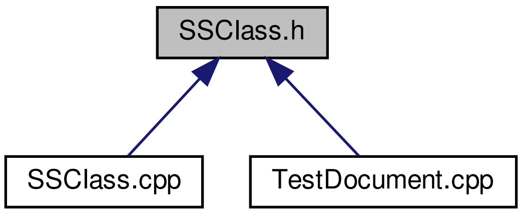
6.10 SSClass.h File Reference

```
#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

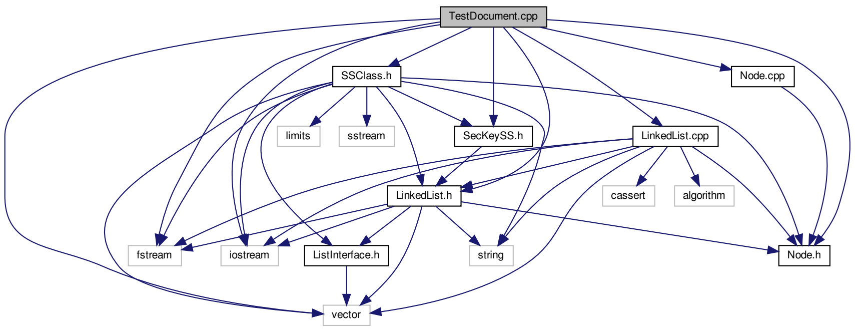
```
const int ZIPSIZE = 6
```

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