Project 1

1.0

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

Class Documentation

4.1 LinkedList< ItemType > Class Template Reference

This is LinkedList class creating a list of linked nodes.

`#include "LinkedList.h"`

Inheritance diagram for LinkedList< ItemType >:

4.2 ListInterface< ItemType > Class Template Reference

Inheritance diagram for ListInterface< ItemType >:

Public Member Functions

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

4.2.1 Detailed Description

template<class ItemType>
class ListInterface< ItemType >

Definition at line 7 of file ListInterface.h.

4.2.2 Member Function Documentation

4.2.2.1 clear()

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

Removes all entries from this list.

Postcondition

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

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

4.2.2.2 deletion()

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

Removes the entry at a given position from this list.

Precondition

None.

Postcondition

If 1 <= 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< int >, and LinkedList< SecKeySS >.

4.2.2.3 getEntry()

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

Gets the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The desired entry has been returned.

Parameters

position	The list position of the desired entry.

Returns

The entry at the given position.

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

4.2.2.4 getLength()

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

Gets the current number of entries in this list.

Returns

The integer number of entries currently in the list.

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

4.2.2.5 insert()

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

Inserts an entry into this list at a given position.

Precondition

None.

Postcondition

If 1 <= 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< int >, and LinkedList< SecKeySS >.

4.2.2.6 isEmpty()

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

Sees whether this list is empty.

Returns

True if the list is empty; otherwise returns false.

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

4.2.2.7 replace()

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

Replaces the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The entry at the given position is newEntry.

Parameters

position	The list position of the entry to replace.
newEntry	The replacement entry.

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

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

- ListInterface.h

4.3 Node< ItemType > Class Template Reference

This is Node class for linked list.

`#include "Node.h"`

Public Member Functions

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

4.3.1 Detailed Description

template<class ItemType>
class Node< ItemType >

This is Node class for linked list.

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

Definition at line 12 of file Node.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Node() [1/3]

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

Node default constructor.

Default constructor assiging next as NULLPTR

Definition at line 8 of file Node.cpp.

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

4.3.2.2 Node() [2/3]

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

Node constructor.

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

Parameters

anItem	a constant reference to an item of itemtype

Definition at line 18 of file Node.cpp.

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

4.3.2.3 Node() [3/3]

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

Node constructor.

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

Parameters

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

Definition at line 30 of file Node.cpp.

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

4.3.3 Member Function Documentation

4.3.3.1 getItem()

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

Member function returning an item.

/return the item of itemType

Definition at line 60 of file Node.cpp.

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

4.3.3.2 getNext()

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

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.

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

4.3.3.3 setItem()

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

Member function taking one argument to set the memebr item.

Parameters

anItem	to be reference to by item

Definition at line 40 of file Node.cpp.

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

4.3.3.4 setNext()

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

Member function taking one argument, a pointer to a Node.

/param nextNodePtr a point to a Node, the next Node in a linked list

Definition at line 50 of file Node.cpp.

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

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

- Node.h
- Node.cpp

4.4 SecKeySS Class Reference

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

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

Public Member Functions

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

4.4.1 Detailed Description

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

Definition at line 14 of file SecKeySS.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SecKeySS() [1/2]

```
SecKeySS::SecKeySS ( )  [inline]
```

Default constructor

Definition at line 20 of file SecKeySS.h.

```
20 { duplicates = LinkedList<int>(); };
```

4.4.2.2 SecKeySS() [2/2]

```
SecKeySS::SecKeySS (
            const SecKeySS & s )
```

Constructor

Definition at line 94 of file SecKeySS.h.

```
94 { data = s.getData(); setDuplicates(s.getDuplicates()); }
```

4.4.2.3 ~SecKeySS()

```
SecKeySS::~SecKeySS ( )
```

Deconstuctor

Definition at line 95 of file SecKeySS.h.

```
95 { duplicates.clear(); }
```

4.4.3 Member Function Documentation

4.4.3.1 getData()

```
string SecKeySS::getData ( ) const  [inline]
```

Gets data

Returns

data the data to be returned

Definition at line 31 of file SecKeySS.h.

```
31 { return data; };
```

4.4.3.2 getDuplicates()

```
LinkedList< int > SecKeySS::getDuplicates ( ) const
```

Gets duplicates

Returns

LinkedList of itemType

Definition at line 110 of file SecKeySS.h.

```
110                                             {
111     LinkedList<int> list;
112     int temp;
113     for (int i = 1; i < duplicates.getItemCount() + 1; i++) {
114         temp = duplicates.getEntry(i);
115         list.insert(i, temp);
116     }
117     return list;
118 }
```

4.4.3.3 operator<() [1/2]

```
bool SecKeySS::operator< (
            const string & 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 52 of file SecKeySS.h.

```
52 { return data < s; };
```

4.4.3.4 operator<() [2/2]

```
bool SecKeySS::operator< (
            const SecKeySS & 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 59 of file SecKeySS.h.

```
59 { return data < s.data; };
```

4.4.3.5 operator=()

```
void SecKeySS::operator= (
            const SecKeySS & s )
```

Operator equal for copy constructor

Parameters

s	a reference to a SecKeySS

Definition at line 106 of file SecKeySS.h.

```
106                                         {
107     data = s.data;
108     duplicates = s.duplicates;
109 }
```

4.4.3.6 operator==() [1/2]

```
bool SecKeySS::operator== (
            const string & s ) const  [inline]
```

Operator is equal

Parameters

s	a reference to a string

Returns

true if data is equal to s

Definition at line 79 of file SecKeySS.h.

```
79 { return data == s; };
```

4.4.3.7 operator==() [2/2]

```
bool SecKeySS::operator== (
            const SecKeySS & 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 86 of file SecKeySS.h.

```
86 { return data == s.data; };
```

4.4.3.8 operator>() [1/2]

```
bool SecKeySS::operator> (
            const string & 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 66 of file SecKeySS.h.

```
66 { return data > s; };
```

4.4.3.9 operator>() [2/2]

```
bool SecKeySS::operator> (
            const SecKeySS & 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 73 of file SecKeySS.h.

```
73 { return data > s.data; };
```

4.4.3.10 setData()

```
void SecKeySS::setData (
            const string s )  [inline]
```

Sets the data equal to argument 1

Parameters

s	a string to set data to

Definition at line 41 of file SecKeySS.h.

```
41 { data = s; };
```

4.4.3.11 setDuplicates()

```
void SecKeySS::setDuplicates (
             LinkedList< int > dup )
```

Sets duplicates

Parameters

LinkedList	dup

Definition at line 119 of file SecKeySS.h.

```
119                                                 {
120     int temp;
121     duplicates.clear();
122     for (int i = 1; i < list.getItemCount() + 1; i++) {
123         temp = list.getEntry(i);
124         duplicates.insert(i, temp);
125     }
126 }
```

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

- SecKeySS.h

4.5 SSClass Class Reference

LinkedList integration for blocks, records, and fields.

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

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.

4.5.1 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.

4.5.2 Member Function Documentation

4.5.2.1 directionalSearch()

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

Searches directionly (N, S, W, E)

Parameters

state	the state to search
direction	(N, S, W, E)

Returns

the line contating the soght after direction

Definition at line 431 of file SSClass.h.

```
431                                                                 {
432     direction = toupper(direction);
433     int i = 1;
434     int returnIndex = -1;
435     double highOrLow;
436     vector<int> state = search(stateS, 3);
437     switch (direction) {
438     case 'N':
439     {
440         returnIndex = state[0];
441         highOrLow = stod(getLat(returnLine(state[0])));
442         for (i; i < state.size(); i++) {
443             if (highOrLow < stod(getLat(returnLine(state[i])))) {
444                 highOrLow = stod(getLat(returnLine(state[i])));
445                 returnIndex = i;
446             }
447
448         }
449     }
450     break;
451     case 'E':
452     {
453         returnIndex = state[0];
454         highOrLow = stod(getLon(returnLine(state[0])));
455         for (i; i < state.size(); i++) {
456             if (highOrLow < stod(getLon(returnLine(state[i])))) {
457                 highOrLow = stod(getLon(returnLine(state[i])));
458                 returnIndex = i;
459             }
460         }
461
462     }
463     break;
464     case 'S':
465     {
466         returnIndex = state[0];
467         highOrLow = stod(getLat(returnLine(state[0])));
468         for (i; i < state.size(); i++) {
469             if (highOrLow > stod(getLat(returnLine(state[i])))) {
470                 highOrLow = stod(getLat(returnLine(state[i])));
471                 returnIndex = i;
472             }
473         }
474         break;
475     }
476     case 'W':
477     {
478         returnIndex = state[0];
479         highOrLow = stod(getLon(returnLine(state[0])));
480         for (i; i < state.size(); i++) {
481             if (highOrLow > stod(getLon(returnLine(state[i])))) {
482                 highOrLow = stod(getLon(returnLine(state[i])));
483                 returnIndex = i;
484             }
485         }
486
487     }
488     break;
489     }
490     return returnIndex;
491
492 }
```

4.5.2.2 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 325 of file SSClass.h.

```
325                             {
326     if (nextEmpty == -1) {
327         goToLine(indexFile, numLinesIndex);
328         indexFile << "\n" << s;
329         insertZip(getZip(s), numLinesIndex);
330         insertPlace(getPlace(s), numLinesIndex);
331         insertState(getState(s), numLinesIndex);
332         insertCounty(getCounty(s), numLinesIndex);
333         insertLat(getLat(s), numLinesIndex);
334         insertLon(getLon(s), numLinesIndex);
335         numLinesIndex++;
336         return;
337     }
338     goToLine(indexFile, nextEmpty);
339     //replace(s, nextEmpty);
340     insertZip(getZip(s), nextEmpty);
341     insertPlace(getPlace(s), nextEmpty);
342     insertState(getState(s), nextEmpty);
343     insertCounty(getCounty(s), nextEmpty);
344     insertLat(getLat(s), nextEmpty);
345     insertLon(getLon(s), nextEmpty);
346 }
```

4.5.2.3 isEmpty()

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

Check if numRecords is 0.

Returns

returns false if empty, otherwise returns true

Definition at line 206 of file SSClass.h.

```
206 { return numRecords == 0; };
```

4.5.2.4 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 261 of file SSClass.h.

```
261                                  { //input is a file name
262     indexFile.open(input);
263     nextEmpty = -1;
264     return (indexFile.is_open());
265
266 }
```

4.5.2.5 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 348 of file SSClass.h.

```
348                                  {
349     string returnVal;
350     goToLine(indexFile, rrn);
351     getline(indexFile, returnVal);
352     return returnVal;
353 }
```

4.5.2.6 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 356 of file SSClass.h.

```
356                                                     {
357     SecKeySS secCopy;
358     int i;
359     vector<int> results;
360     switch (fieldNum) {
361     case 1:
362     {
363         for (i = 1; (i < (secKeyZip.getItemCount() + 1)) && (secKeyZip.
    getEntry(i).getData() < s); i++);
364         if (stoi(secKeyZip.getEntry(i).getData()) == stoi(s)) {
365             LinkedList<int> toCopy = LinkedList<int>(secKeyZip.
    getEntry(i).getDuplicates());
366             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
367                 results.push_back(toCopy.getEntry(j));
368             }
369         }
370     }
371     break;
372     case 2:
373     {
374         for(i = 1; (i < (secKeyPlace.getItemCount() + 1)) && (secKeyPlace.
    getEntry(i).getData() < s); i++);
375         if ((secKeyPlace.getEntry(i).getData()) == (s)) {
376             LinkedList<int> toCopy = LinkedList<int>(secKeyPlace.
    getEntry(i).getDuplicates());
377             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
378                 results.push_back(toCopy.getEntry(j));
379             }
380         }
381     }
382     break;
383     case 3:
384     {
385         for (i = 1; (i < (secKeyState.getItemCount() + 1)) && (secKeyState.
    getEntry(i).getData() < s); i++);
386         if ((secKeyState.getEntry(i).getData()) == (s)) {
387             LinkedList<int> toCopy = LinkedList<int>(secKeyState.
    getEntry(i).getDuplicates());
388             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
389                 results.push_back(toCopy.getEntry(j));
390             }
391         }
392     }
393     break;
394     case 4:
395     {
396         for (i = 1; (i < (secKeyCounty.getItemCount() + 1)) && (secKeyCounty.
    getEntry(i).getData() < s); i++);
397         if ((secKeyCounty.getEntry(i).getData()) == (s)) {
398             LinkedList<int> toCopy = LinkedList<int>(secKeyCounty.
    getEntry(i).getDuplicates());
399             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
400                 results.push_back(toCopy.getEntry(j));
401             }
402         }
403     }
404     break;
405     case 5:
406     {
407         for (i = 1; (i < (secKeyLat.getItemCount() + 1)) && (secKeyLat.
    getEntry(i).getData() < s); i++);
408         if (stoi(secKeyLat.getEntry(i).getData()) == static_cast<int>(stod(s))) {
409             LinkedList<int> toCopy = LinkedList<int>(secKeyLat.
    getEntry(i).getDuplicates());
410             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
411                 results.push_back(toCopy.getEntry(j));
412             }
413         }
414     }
```

```
415     break;
416     case 6:
417     {
418         for (i = 1; (i < (secKeyLon.getItemCount() + 1)) && (secKeyLon.
    getEntry(i).getData() < s); i++);
419         if (stoi(secKeyLon.getEntry(i).getData()) == static_cast<int>(stod(s))) {
420             LinkedList<int> toCopy = LinkedList<int>(secKeyLon.
    getEntry(i).getDuplicates());
421             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
422                 results.push_back(toCopy.getEntry(j));
423             }
424         }
425     }
426     break;
427     }
428     return results;
429 }
```

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

- SSClass.h

Index