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

Public Member Functions

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

4.4.1 Detailed Description

Definition at line 9 of file SecKeySS.h.

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

- SecKeySS.h

4.5 SSClass Class Reference

Public Member Functions

- SSClass ()
  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 directly (N, S, W, E)
- string returnLine (int rrn)
  Fills secondary key vector.

4.5.1 Detailed Description

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 directly (N, S, W, E)

Parameters

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

Definition at line 342 of file SSClass.h.

```
342                                                         {
343     direction = toupper(direction);
344     int i = 1;
345     int returnIndex = -1;
346     double highOrLow;
347     vector<int> state = search(stateS, 3);
348     switch (direction) {
349     case 'N':
350     {
351         returnIndex = state[0];
352         highOrLow = stod(getLat(returnLine(state[0])));
353         for (i; i < state.size(); i++) {
354             if (highOrLow < stod(getLat(returnLine(state[i])))) {
355                 highOrLow = stod(getLat(returnLine(state[i])));
356                 returnIndex = i;
357             }
358
359         }
360     }
361     break;
362     case 'E':
363     {
364         returnIndex = state[0];
365         highOrLow = stod(getLon(returnLine(state[0])));
366         for (i; i < state.size(); i++) {
367             if (highOrLow < stod(getLon(returnLine(state[i])))) {
368                 highOrLow = stod(getLon(returnLine(state[i])));
369                 returnIndex = i;
370             }
371         }
372
373     }
374     break;
375     case 'S':
376     {
377         returnIndex = state[0];
378         highOrLow = stod(getLat(returnLine(state[0])));
379         for (i; i < state.size(); i++) {
380             if (highOrLow > stod(getLat(returnLine(state[i])))) {
381                 highOrLow = stod(getLat(returnLine(state[i])));
382                 returnIndex = i;
383             }
384         }
385         break;
386     }
387     case 'W':
388     {
389         returnIndex = state[0];
390         highOrLow = stod(getLon(returnLine(state[0])));
391         for (i; i < state.size(); i++) {
392             if (highOrLow > stod(getLon(returnLine(state[i])))) {
393                 highOrLow = stod(getLon(returnLine(state[i])));
394                 returnIndex = i;
395             }
396         }
397
398     }
399     break;
400     }
401     return returnIndex;
402
403 }
```

4.5.2.2 insert()

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

inserts line by line into data

Parameters

s	a string to insert

Definition at line 233 of file SSClass.h.

```
233                             {
234     if (nextEmpty == -1) {
235         goToLine(indexFile, numLinesIndex);
236         indexFile << "\n" << s;
237         insertZip(getZip(s), numLinesIndex);
238         insertPlace(getPlace(s), numLinesIndex);
239         insertState(getState(s), numLinesIndex);
240         insertCounty(getCounty(s), numLinesIndex);
241         insertLat(getLat(s), numLinesIndex);
242         insertLon(getLon(s), numLinesIndex);
243         numLinesIndex++;
244         return;
245     }
246     goToLine(indexFile, nextEmpty);
247     //replace(s, nextEmpty);
248     insertZip(getZip(s), nextEmpty);
249     insertPlace(getPlace(s), nextEmpty);
250     insertState(getState(s), nextEmpty);
251     insertCounty(getCounty(s), nextEmpty);
252     insertLat(getLat(s), nextEmpty);
253     insertLon(getLon(s), nextEmpty);
254 }
```

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 114 of file SSClass.h.

```
114 { 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 168 of file SSClass.h.

```
168                                  { //input is a file name
169     indexFile.open(input);
170     nextEmpty = -1;
171     return (indexFile.is_open());
172
173 }
```

4.5.2.5 returnLine()

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

Fills secondary key vector.

Parameters

rrn	and integer refring to the line to get

Definition at line 260 of file SSClass.h.

```
260                                  {
261     string returnVal;
262     goToLine(indexFile, rrn);
263     getline(indexFile, returnVal);
264     return returnVal;
265 }
```

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 268 of file SSClass.h.

```
268                                                     {
269     SecKeySS secCopy;
270     int i;
271     vector<int> results;
272     switch (fieldNum) {
273     case 1:
274     {
275         for (i = 1; (i < (secKeyZip.getItemCount() + 1)) && (secKeyZip.
    getEntry(i).getData() < s); i++);
276         if (stoi(secKeyZip.getEntry(i).getData()) == stoi(s)) {
277             LinkedList<int> toCopy = LinkedList<int>(secKeyZip.
    getEntry(i).getDuplicates());
278             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
279                 results.push_back(toCopy.getEntry(j));
280             }
281         }
282     }
283     break;
284     case 2:
285     {
286         for(i = 1; (i < (secKeyPlace.getItemCount() + 1)) && (secKeyPlace.
    getEntry(i).getData() < s); i++);
287         if ((secKeyPlace.getEntry(i).getData()) == (s)) {
288             LinkedList<int> toCopy = LinkedList<int>(secKeyPlace.
    getEntry(i).getDuplicates());
289             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
290                 results.push_back(toCopy.getEntry(j));
291             }
292         }
293     }
294     break;
295     case 3:
296     {
297         for (i = 1; (i < (secKeyState.getItemCount() + 1)) && (secKeyState.
    getEntry(i).getData() < s); i++);
298         if ((secKeyState.getEntry(i).getData()) == (s)) {
299             LinkedList<int> toCopy = LinkedList<int>(secKeyState.
    getEntry(i).getDuplicates());
300             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
301                 results.push_back(toCopy.getEntry(j));
302             }
303         }
304     }
305     break;
306     case 4:
307     {
308         for (i = 1; (i < (secKeyCounty.getItemCount() + 1)) && (secKeyCounty.
    getEntry(i).getData() < s); i++);
309         if ((secKeyCounty.getEntry(i).getData()) == (s)) {
310             LinkedList<int> toCopy = LinkedList<int>(secKeyCounty.
    getEntry(i).getDuplicates());
311             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
312                 results.push_back(toCopy.getEntry(j));
313             }
314         }
315     }
316     break;
317     case 5:
318     {
319         for (i = 1; (i < (secKeyLat.getItemCount() + 1)) && (secKeyLat.
    getEntry(i).getData() < s); i++);
320         if (stoi(secKeyLat.getEntry(i).getData()) == static_cast<int>(stod(s))) {
321             LinkedList<int> toCopy = LinkedList<int>(secKeyLat.
    getEntry(i).getDuplicates());
322             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
323                 results.push_back(toCopy.getEntry(j));
324             }
325         }
326     }
327     break;
328     case 6:
329     {
330         for (i = 1; (i < (secKeyLon.getItemCount() + 1)) && (secKeyLon.
    getEntry(i).getData() < s); i++);
331         if (stoi(secKeyLon.getEntry(i).getData()) == static_cast<int>(stod(s))) {
332             LinkedList<int> toCopy = LinkedList<int>(secKeyLon.
    getEntry(i).getDuplicates());
333             for (int j = 1; j < (toCopy.getItemCount() + 1); j++) {
```

```
334                 results.push_back(toCopy.getEntry(j));
335             }
336         }
337     }
338     break;
339     }
340     return results;
341 }
```

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

- SSClass.h

Index