Hayley Yukihiro

CSIS215 – D01

December 18, 2020

**Programming Assignment 4: Self-Organizing Lists**

**Approach**

To start off the approach document, I will outline the direction of the approach to the Self-Organizing Lists programming assignment. I reviewed the self-organizing list section of the textbook to refresh my understanding to outline my approach. I will be implementing the ADT files for self-ordering lists into three different header files that will hold the functional implementation of count, move-to-the-front, and transpose:

Outline Approach

* Create Count.h to implement SelfOrderedListADT.h
  + Constructor
  + Destructor
  + find()
  + add()
  + getCompares()
  + size()
  + printlist()
  + reorder()
* Create MoveToTheFront.h to implement SelfOrderedListADT.h
  + Constructor
  + Destructor
  + find()
  + add()
  + getCompare()
  + size()
  + printlist()
  + reorder()
* Create Transpose.h to implement SelfOrderedListADT.h
  + Constructor
  + Destructor
  + find()
  + add()
  + getCompare()
  + size()
  + printlist()
  + reorder()
* Create testmain.cpp
  + Write chartest
  + Write stringtest

The preliminary steps were completed to begin the assignment by compiling all the required .cpp and .h files into a folder that is named per instructions.

**Count.h**

To begin Count.h, I created a new header file and inherited the BufferBlockADT.h. I included llist.h to utilize some of its functionality. I reviewed the count description from section 9.2 in the textbook to help brainstorm the implementation. In the private section, I added an object to be used for function calls from llist.h , a int variable to represent the variable of number of comparisons, and a counter holder variable.

**Constructor:**

* Initialized the number of comparisons to 0
* Counter to 0

**Destructor:**

* Default destructor

**reorder(): Reorder items in the list based on the number of times accessed**

* Not part of the ADT but required by the instructions to have reorder implementation to order the list based on item usage
* Created two variables that hold the access times of the item being moved and the previous item
* In a while loop that continues till the move item is less than the previous item, the items will be reordered by removal and insertion in the list utilizing llist.h functions

**find(): Looks for item**

* Created a boolean instance variable that will be used to return true or false if the value is either found or not found in the list
* Called moveToStart from llist.h to bring the current position element to the beginning of the list to begin the search
* Created a for loop to look for the value that is being searched in the list. For every for loop, the number of comparisons is incremented.
* If the current element of llist.h element matches the key being searched, the Boolean instance is changed to true and ends the loop. If the current element does not match the key, then the loop will continue down the list.
* Created an if-else statement to do different processes if the item is already in the list or new to begin the reorder process
* For a duplicate item, the access times will be updated and then the list is reordered to return true
* For a new item, call add(). Then return true.
* I needed to find a way to set and get the access times. I thought about making another function within the Count class but I decided it be easier to make a set and get function for the times from the LList class. To do that, I added the function to LList.h that calls a function that I made in link.h that gets and sets the access numbers for a list item. In total, I added two functions to LList and Link header files.

**add(): Add the new item to the list**

* Current position is moved to the end of the list
* New item is inserted
* Access times for the item is set to zero

**getCompares(): Returns the number of accumulated compares**

* Returns the variable comp that holds the number of total comparisons

**size(): returns the size of the list**

* Returns the length of the list by calling the length function from LList.h

**printlist(): prints list (1 that prints entire list ; 1 that prints n nodes)**

* Both types of print lists follow the same implementation other than one prints the entire length of the list and one only prints a certain specified amount of nodes
* The LList class did not have an implementation so I added a print list implementation that requires an int for the function call.
* I copied the example print output layout for the print list implementation. For the entire list print, I used the length of the list for the function call of print() from LList. For the node print, I used the node for the function call of print().

**MoveToFront.h**

To begin MoveToFront.h, I created a new header file and inherited the SelfOrderedListADT.h. After creating Count.h, I made the reorder function that seems to be the only function that changes the most amongst the three header files. In the private section, I added an object to be used for function calls from llist.h , int variable to represent the variable of number of comparisons, and counter holder variable.

**Constructor:**

* Initialized the number of comparisons to 0
* Counter to 0

**Destructor:**

* Default destructor

**reorder(): Reorder items in the list by moving items to the front when they are accessed**

* Not part of the ADT but required by the instructions to have reorder implementation to order the list based on item usage
* I changed the code to moving an item to the front of the list instead of reordering the list based on the number of times accessed
* I created three variables that hold the item being moved, next, and front of the list.
* After the item is added to the move variable, the list goes to the next element and gets its access times.
* The lists goes back to the item about to be removed and removes it to be held in a temp variable. Then the item next on the list is updated on accesses.
* The current position of the list moves to the front where the temp variable inserts the removed item . Then goes to the next position on the list and updates the front item variable of the list.

**find(): Looks for item**

* Same code as Count.h

**add(): Add the new item to the list**

* Same code as Count.h

**getCompares(): Returns the number of accumulated compares**

* Same code as Count.h

**size(): returns the size of the list**

* Same code as Count.h

**printlist(): prints list (1 that prints entire list ; 1 that prints n nodes)**

* Same code as Count.h

**Transpose.h**

To begin Transpoe.h, I created a new header file and inherited the SelfOrderedListADT.h. After creating Transpose.h, I made the reorder function that seems to be the only function that changes the most amongst the three header files. In the private section, I added an object to be used for function calls from llist.h , int variable to represent the variable of number of comparisons, and counter holder variable.

**Constructor:**

* Initialized the number of comparisons to 0
* Counter to 0

**Destructor:**

* Default destructor

**reorder(): Reorder items in the list by swapping current item with previous item**

* Not part of the ADT but required by the instructions to have reorder implementation to order the list based on item usage
* I changed the code to swapping current item with previous item
* I created three variables: the holder of the first item of the list, the last item of the list, and the counter
* If the item matches the first item, it is returned
* If the current item matches the last item, it is switched with the previous item

**find(): Looks for item**

* Same code as Count.h

**add(): Add the new item to the list**

* Same code as Count.h

**getCompares(): Returns the number of accumulated compares**

* Same code as Count.h

**size(): returns the size of the list**

* Same code as Count.h

**printlist(): prints list (1 that prints entire list ; 1 that prints n nodes)**

* Same code as Count.h

**Test.cpp**

After completing all three heuristic header files. I began to create the test.cpp:

* I originally had printlist() create the string outputs for each test but I decided to just have printlist() print the list and not the entire test output. I wrote the strings of the test output in the test.cpp
* Instead of adding each item individually, I decided to create two arrays that hold the original list and the test list. Then I looped through the arrays to add each item to the lists.
* I created two test functions that outputted the required output of the character test and string read in test.
* I spent a while debugging because I was unable to get the total number of access related to each swap for each test. I realized I needed to change print() to get the access times and not the counter. I still had one less than the acceptable output example and I’m not quite sure why.
* I tried to figure out why I was not put to the front of the list for the move-to-front test when I realized that I should not call the add() function for an unknown item but call movetostart() function to go to the top of the list and then insert() function to put I at the top.