Preprogramming Approach

5/3/23:

1. Review assignment instructions carefully
    a. Implement the three self-organizing list heuristics: count, move-to-front, and transpose
    b. Keep track of compares to be able to track the cost of each heuristic
    c. Use SelfOrderedListADT and the provided linked list files to implement the self-ordered lists (can use inheritance or composition)
    d. Can make changes to any files EXCEPT SelfOrderedListADT and test.txt
    e. This program will run two tests:
        i. Char Types: uses the add() function to build an initial list from A to H. Create a second function to input the character list containing I. For each heuristic, display the order of the final list with the number of compares.
        ii. Second test uses the test.txt file using the data type string. Pull the test.txt into the program using the find() function to add words to the list, then print out the total number of words, compares, and the first 10 words in the list along with frequency. (THE EXAMPLE IS NOT THE EXACT EXPECTED OUTPUT)
    f. Submit plenty of screenshots
    g. Implement the SelfOrderedListADT methods:
        i. find() - finds a value in list and increments frequency variable, if value not found, then add() is called to put value at the end of the list. Both cases calls reorder() to order according to heuristic being used. find() increments the number of compares made, but add() does not.
        ii. add() - appends the value to the end of the list without doing any compares or adjusting frequencies.
        iii. getCompares() - returns the total number of compares done by find when searching for values in a list
        iv. size() - returns the size of the list
        v. printList() prints the list following this syntax: value-## where value is the value of the node (char or string) and ## is the freq of that value.
        vi. printList(n) - separate method to allow for only 10 nodes to be printed
        vii. reorder() - can be called anywhere, but this is a method that reorders list based on heuristic being used (typically called by find())
2. Review self-organizing lists in textbook:
    a. Organizes records by expected frequency of access.
    b. Assumes that most frequently used record will be stored first.
    c. 80/20 rule says 80% of access methods hit 20% of records stored.
    d. Self Org Lists (SOLs) modify the order of records based on the actual pattern of record access
    e. The most obvious approach is the Count heuristic, similar to LFU buffer method
    f. Move to front heuristic is similar to LRU buffer method, and is easy with linked lists

g. Transpose heuristic swaps the record with the record preceding it with each access.

5/5/23:

1. Download code files and review.
    a. Preliminary thoughts - since there is no limits/requirements to how the lists are implemented, it might be best to use a different implementation for each list heuristic (Ex. Count inherits SelfOrderedListADT, Move-to-Front inherits SelfOrderedListADT, and Transpose inherits SelfOrderedListADT)
    b. I'm going to create a main.cpp, but I'll only use comments with it to keep track of what is needed, I will create the code for this file last.

5/7/23:

1. Started working on the count heuristic header file.
    a. Constructor sets the compares to zero, destructor implemented.
    b. Completed the getCompares method
        i. Did this so I could verify that the constructor does set this variable
        ii. Added a list object to data members for other methods to use
    c. Wrote the add() method and added code to fill the list
    d. I finished both of the printlist methods.
        i. Had to comment out printlist(n), was showing an error that said that printlist(n) did not take an argument, maybe this is because I haven't finished the mainDriver's call to the test.txt strings
        ii. Had some errors with the list object, made the following modifications to other files; 5/8/2023 added more methods
            1. LList.h: wrote the implementation of the print method using objects from link class.
            2. Put in get/set methods in LList to get the value of timesaccessed from the link class, added a get method to the list.h file, added the get/sets to link.h
            3. Removed the assertion line from the remove method (planning to handle when curr->next is null within the find() method, and want to see if I can do that without this line)
            4. Added methods to get the tail and front items in the list.
        iii. Same thing as above for print(n)
    e. getSize method done, started the find method.

5/8/23:

1. Finished the find method
    a. May have to come back to this, I did not see any errors with it - 5/11/2023 - was not able to use add() method, values would be inserted but reorder was not working correctly

        i. Used a bool flag called isFound to determine whether the value needed to be added or not
        ii. if/else statements conditioned on the bool flag, reorder commented out for now.
    b. Completed the reorder function
        i. Compared neighboring elements to push highest accessed to front of list.
        ii. Uncommented reorder from find, so far so good
2. Since the only real difference appears to be the reorder function for these methods, I will just copy/paste the files to new headers titled mtf.h and transpose.h and change names where appropriate.
3. Started mtf.h heuristic header
    a. Made changes to constructor, destructor.
    b. Changed reorder function to push elements to front of list that were most recently accessed
4. Added logic for mtf adds/finds to mainDriver.
5. Started transpose.h
    a. Just as above, modified the constructor and destructor from the count.h file to fit
    b. Reorder was a bit more challenging with transpose, ultimately had to create two separate local data members to keep track of the swaps occurring.

5/9/23:

1. Adding the required logic to mainDriver to perform the string reads from the test file.
    a. Having issues with creating string objects in mainDriver.
        i. Char objects from earlier are also erroring out
    b. I spent a few hours debugging (even reverted the changes), still seeing the errors.
        i. Going to try creating a new project in VS2017 and just move the c++ files into the new solution
        ii. The project was still not able to compile, deleting the files from that project and creating blank ones + copying the raw code from the old files
        iii. The project now compiles, will try adding the string methods again tomorrow or Thursday. I will be wary of how much I add with each step.

5/10/23:

1. Added code for string objects and nothing else to see if project build would succeed
    a. Appears to work at this point
2. Added logic to take each word from file, and add to all 3 lists with each pass
    a. Appears to still be working
3. Added printlist(10) function calls
    a. The lists are printing to the console, but the reorder method must not be working whatsoever, because the move to front heuristic has entries at the front with 0 and 1 accesses.

    b. After debugging for a while, changing ‘this->add(it)’ to ‘list.insert(it)’ within the find method seems to have fixed mtf, will copy for the other headers and adjust as needed.
4. Program appears to work as intended, will work on conciseness and readability tomorrow.

5/11/23:

1. Added comments to code that detail what each method is doing
    a. Reviewed for opportunities to remove/edit portions for cleaner implementations, started getting errors again so I decided it was better left alone with the deadline fast approaching (I am unable to work on this tomorrow, 5/12)
2. Added pausespace() method to mainDriver to improve readability.
3. I noticed the number of compares on the strings are incredibly large, but I don’t see any issues with the code so I will submit as is.