Programming Project #3

Overview: Lists are common programming data structures for storing collections of items with a simple first-to-last linear relationship. Programmers often use arrays for such collections because the indexing mechanism is so clean and efficient. But building and maintaining sorted lists with arrays requires data to be moved as items are inserted and deleted. A list implemented as a linked list trades efficient indexing for efficient insertions and deletions.

Objective: Demonstrate your ability to implement a list ADT with a linked list mechanism and use the ADT to work with sorted lists.

Download Required Project Files: (from Moodle assignment)

- See the document "readme.pdf" in the zip file for explanations of zip file contents and some additional help.

Input: A standard ASCII text file containing an unknown amount of integers separated by whitespace. Positive integers are to be added to the list while negative integers indicate the number's positive form should be removed from the list, if it exists.

Output: Simple display of a counts summary and the final content of the list.

Linked List Implementation:

- Add necessary code to implement all existing functions in the file linkedlist.c. DO NOT add any additional functions and DO NOT make changes to any function headers. Also, DO NOT make any changes to the indicated sections of code.
- List function implementations must make use of assert() function calls to check validity of parameters and list status for pre-conditions stated in the comments.
- DO NOT use comparison operators directly with the data stored in the list or function parameters. When a comparison is needed, you must use the compareItem() function provided by the DataItem interface.
- Any debugging output must be commented-out for assignment submission.

Main Program Modification: Below the existing main program code provided in the zip file, add the necessary code to create sorted lists as follows.

- Reopen the input data file and process all data in the file to build each sorted list, closing the data file before printing list results.
- Reuse the variable `mylist1` to create a sorted list in ascending order with no duplicates allowed. Then print the count summary and final contents of the list as done in the previous section of code, using a for-loop as required in the coding requirements below.
- Reuse the variable `mylist2` to create a sorted list in descending order with duplicates allowed. Then print the count summary and final contents of the list as done in the previous step.
- Destroy both lists before program terminates.
- You may put additional debugging code in your sorts to show what is happening with the lists but comment-out that code to submit the assignment.

Coding Requirements:

- Code your program in standard C – project must have no compiler warnings at level 4.
- Main program must make use of the input data file and the command line argument that provides the name of the data file.
- Your newly added code in main() that builds the sorted lists **MUST** do so using the list functions findListItemAscend() and findListItemDescend() which enable items to be inserted into their properly sorted positions. Do **NOT** build unsorted lists and then sort them in some fashion by processing the list a second time.
- When main() prints the final list results for the sorted lists, use for-loops with list functions instead of the while-loops used in the previous sections of code.
- DO NOT MAKE CHANGES to the code files or portions of code files where indicated by comments!! I will test your submission with my original code files.

Program Testing:

- Start with the existing main program provided in the assignment's zip file – it should work and produce the exact results also provided in the zip file. Once working, keep this code active to recheck linked list operation as testing progresses and changes are made to complete the linked list implementation!
- Add new code to the main program to create the ascending list and compare results to those provided in the zip file. Once working, copy and paste with necessary changes to produce the required descending list. Compare results to those provided.
- My sample main program does not exercise every possible list function and situation but my grading program will. You should figure out what is not being exercised and add necessary list actions to main to perform additional testing.

Grading:

- In order to receive a **non-zero grade**, your program must execute, produce correct output, and terminate normally.
- Each time your submission does not meet the above requirement, it will be returned for corrective action and your maximum achievable grade will be reduced by 10%. Late penalties may also apply.
- This program will be 150% of the first program.

Submission: Upload a zip file of C code to Moodle.
Zip must contain all project code files, both .c and .h files

Due Date: Oct 27th (upload to Moodle before midnight)