Sanjana Rathore

UID: 006216683

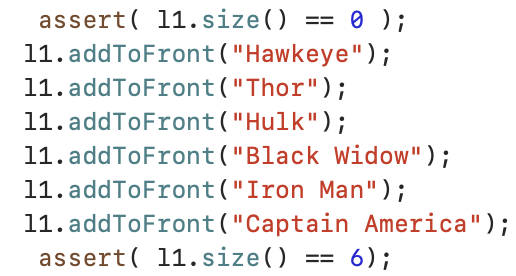
Project 3 Report

Notable Obstacles and Coding Process:

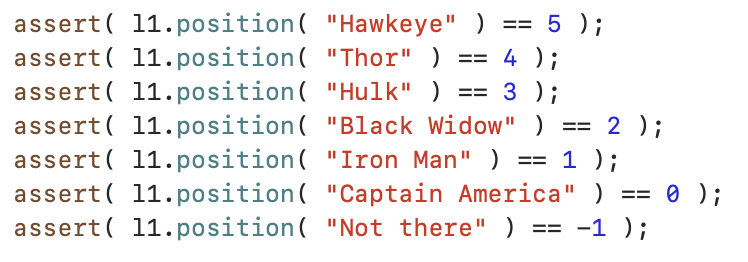
I went about the coding process for Project 3 by doing one method and a time and then testing it. In the past, I have coded very similar methods but for arrays so I was able to apply the same logic but make it work for a different data structure which in this case was a linked list. The coding process for the first five methods was relatively smooth. For one of the methods, I forgot to set the pointer of the node to point to the next node so my code was stuck in an infinite loop, but I quickly noticed the problem and was able to fix it. However, when I was coding the removeAllBiggerThan method, I ran into a notable obstacle. Instead of removing ALL words bigger than the data parameter string, it would only remove one. At first, I thought this was because the node was somehow being set to nullptr (potentially with the deleteItem method) and the while loop was terminating before it iterated through the whole list. However, upon understanding the implementation of the deleteItem method, I realized that wasn’t the case. Then, I realized that because I was deleting nodes from the linked list, the code might not be traversing through every node of the list and skipping over some. Some of the nodes were getting shifted forward due to the deletion. I tried to fix the issue by only setting the node to next node when a node was not deleting, but that didn’t work which I think is because of how the deleteItem method is implemented. I ultimately was able to fix the issue by setting the node to the head every time a node was deleted so the it is forced to traverse through every node again until it reaches another bigger string (in this case the node is reset to head again and the list will be traversed) or until it reaches the end of the list (as denoted by nullptr).

Testing:

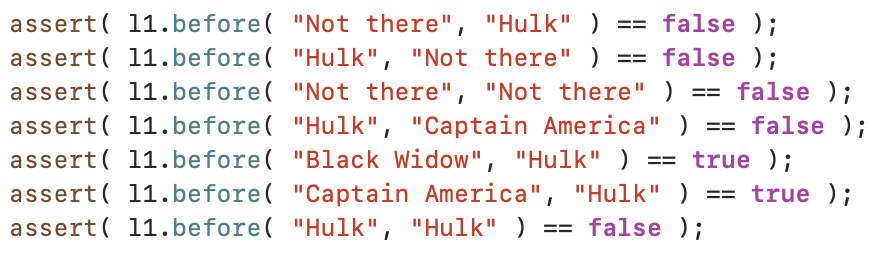
Testing the size method: All test cases for the size function can be generalized into two categories. Either they can be empty in which the size function will return 0, or they have some amount of elements, in which the size function will have to iterate through the list and return a value.



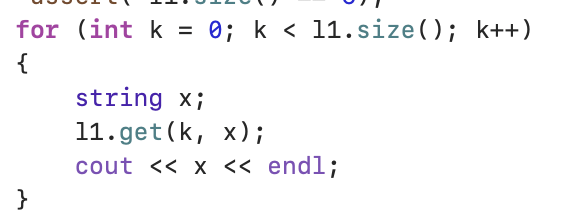
Testing the position method: For the position method, I tested 4 different types of positions: 0th position, last position, the position of a string that is not in the list, and a position of a string that is in the middle of the list.

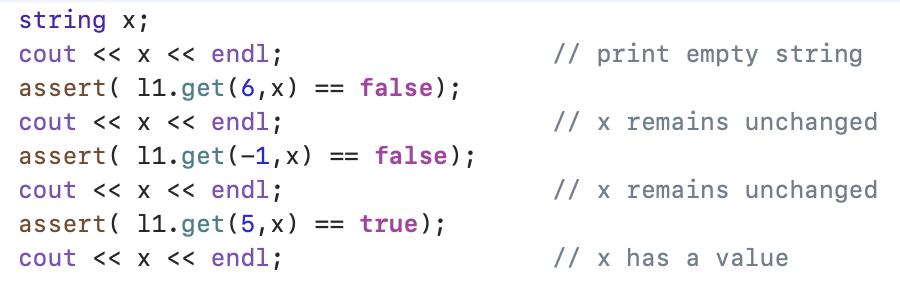


Testing the before method: For the before function, I tested 5 cases: If the first string is before the second string (returns true), if the first string is after the second string (returns false), if one of the strings is out of bounds (returns true), if both of the strings are out of bounds (returns false), if both of the strings are the same (returns false)



Testing the get method: I tested get on a linked list using a for loop to iterate through each node and print it as shown below. I also tested the method for an index that was out of bounds of the linked list. In this case, the string should remain and empty string and the method should return false





Testing the min method: When the list is empty, the min function should return an empty string so I did a test case for that. I also did a test case for a list with only one node in it. In this case, the min method should return that string. I added a few more strings to the list and tested the function with that as well.



Testing the removeAllBiggerThan method: For the removeAllBiggerThan, I tested 3 cases: a case where the data parameter is smaller than every element (all elements should be removed), a case where the data parameter is larger than every element (no elements should be removed), and a case where the data parameter is somewhere in the middle.

