Lab Report: Linked List

Introduction:  
 A doubly linked list is a list that has pointers going to the previous as well as the next node in the list. The data inserted can be placed anywhere the programmer wishes. I have gone with putting the new data into the front of the list, with any following data inserted into the end. This linked list contains the operations that can insert, remove, or find a node in the list. The insert function allows you to add a node into the list. The remove function will delete a node from the list and the find function will loop around the list until it finds the node you have entered.

Programmers Guide:

For my program, I first created a LinkedList class. After that, I created a dataHolder class and classified it as a friend class, so I could access my LinkedList class. I then implemented a template with the data type “T” to accommodate all data types. My next and previousData variables were then set to NULL to start with. I used shared pointers to create the double linked list.

The insert function will let you enter in any data type into the list. The new data will be inserted at the end of the list, making it endData. Entering another node will make it equal to nextData. It works like “Enter a number: “ and the number will be added to the list. Of course, I used all data types, so anything will work.

The find function will loop through the current list and will keep going until it finds the specified data. If none of the nodes are equal to the data entered, the data will come back as NULL. Entering a variable in this function will find the node you want.

The remove function works pretty similar to the find function, except it will remove the node it finds, rather than just display it. This one was also difficult for me to understand, so it took a while. So, what I did is that to remove the middle node, I had to have the previous pointer of the current node point to the previous node, and that node will point to the node After the current node. So, lets say you have 3 numbers in the list, 5 6 7. Entering 6 into the node will have 6 point to 5, then five will skip 6 and point to 7. With nothing pointing to 6 anymore, 6 is removed from the memory address.

I ran into a problem here as well, because I realized that if I didn’t have a previous or next node, the program will just point to 0000000, because there’s pretty much nothing there. What I did was “If previous/nextData is NULL, nextData will be the frontNode”.

The minimum function compares the values that is entered and the nodes in the list. If the node in the list is less than that of the value entered, it will actually change the variable to that node, and then keep looping through the list to guarantee it is the lowest possible value. The maximum function works almost exactly the same way, except with the change of greater than.

I had to add another function, the display function, to show that my program works. This basically just couts the nodes until it gets to the end of the list.