Data Structures (810:052) Name:_____

1. The book's LinkedList interface had both operations: appendNode and insertNode. Why would we not want both operations?

2. We were in the middle of writing the insertNode operation for a **sorted linked list** (similar to the LinkedList Template Version 2). To write correct code, you **should** start by considering all "special cases" that might exist when the insertNode operation is called:

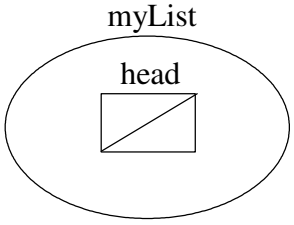
- general case - inserting in the middle of the list, i.e., between two nodes
- inserting the very first item into an empty list
- inserting at the end of a non-empty list
- inserting at the beginning of a non-emtpy list

We had developed the following code which handles the general case, but alas does not handle all the cases.

```
template <class T>
void LinkedList<T>::insertNode(T newValue) {
      ListNode<T> *current;              // To traverse the list
      ListNode<T> *previous = NULL;    // The previous node


      current = head;
      previous = NULL;

      while (current != NULL && current->value < newValue) {
            previous = current;
            current = current->next;
      } // end while

      ListNode<T> *newNode = new ListNode<T>(newValue);

      newNode->next = current;
      previous->next = newNode;









}
```

a) How could you detect the case: "inserting the very first item into an empty list"?

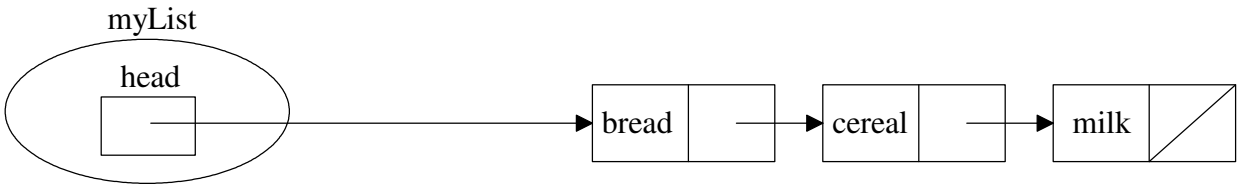
b) Modify the empty-list picture to show the steps needed to insert the very first item. (number the steps)



c) Modify the above code to handle the "inserting the very first item into an empty list" case.

d) How could you detect the case: "inserting at the beginning of a non-emtpy list"?

e) Modify the non-empty list picture to show the steps needed to insert an item at the beginning.



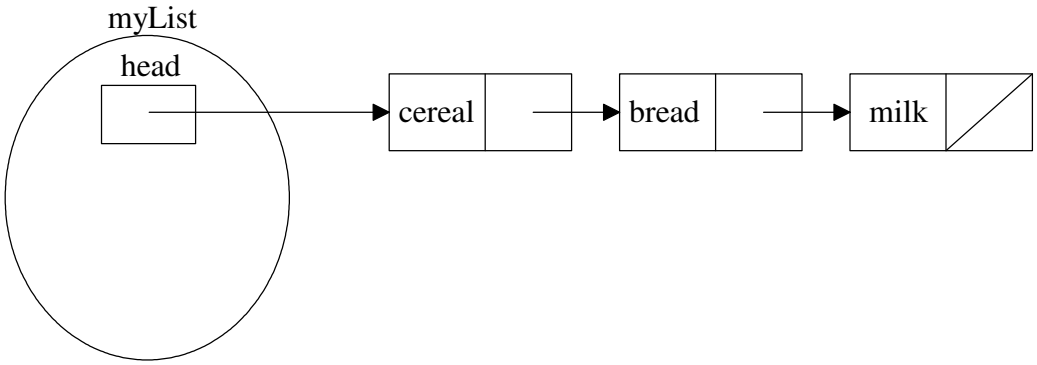
f) Modify the above code to handle the "inserting at the beginning of a non-empty list" case.

3. Suppose we had an **unsorted** singly-linked list API that only added items using either appendEnd or appendFront operations. Plus, it has a size operation to return the number of items in the list.

a) If the list has only a "head" pointer in its representation (as shown above), what would be the theta notation of each operations? (Assume the list currently has "n" items in it)

- appendEnd -
- appendFront -
- size -

b) What would you add to the list representation (i.e., new data members) to facilitate these operations?



c) With this new representation, what would be the theta notation of each operations? (Assume "n" list items)

- appendEnd -
- appendFront -
- size -

d) With your new representation, suppose we wanted to add a new operation appendList which would be called like: "myList.appendList(anotherList);" This call would append all of the anotherList items to the end of the original myList items leaving the anotherList empty. Draw: (1) the representations of the two lists: myList: <1, 5, 2> and anotherList <3, 1, 8, 9> before the call, and (2) show the (numbered) changes after the appendList call.