List ADT implemented with a Linked List

Problem:

For this assignment you will implement the List ADT from the class demo (List_3358 on the class website) using a doubly linked list. This will make the insert and delete O(1). The class demo is implemented using an array, and the insert and delete are O(n).

Use the following header file: **list_3358_LL.h** (on the class website). The header file contains the datatype and member variables necessary for the doubly linked list. Make a separate implementation file (list_3358_LL.cpp) for the member function definitions.

I recommend writing a good test program (similar to list_test.cpp) to make sure all the functions work properly. Since the array-based implementation of list_3358.h and the linked list implementation of list_3358_LL.h share the same exact interface, you should be able to use the same test program for both of these implementations—and you should get the exact same results! You may want to add some tests to the list_test.cpp program (or write an entirely new one) to sufficiently test your linked list implementation.

NOTES:

- Read the comments in the *.h files carefully (they are the same as in the class demo). They explain what each function needs to do.
- Do not implement the copy constructor until last (you may want to implement it by calling other function(s) in the class).
- Make sure you maintain both the next AND previous pointers (and head AND tail) as you implement the operations. Use NULL for EOL.
- Run your test program on the class demo to get the correct output, then run it on your linked list implementation to validate the output from your implementation.
- The purpose of this assignment is to get experience implementing an ADT in two different ways, to practice with linked lists and pointers, and to get more familiar with the separation of interface and implementation.

Style:

See the Style Guidelines document on the course website. You do not need function description comments this time (they are already in the *.h file).

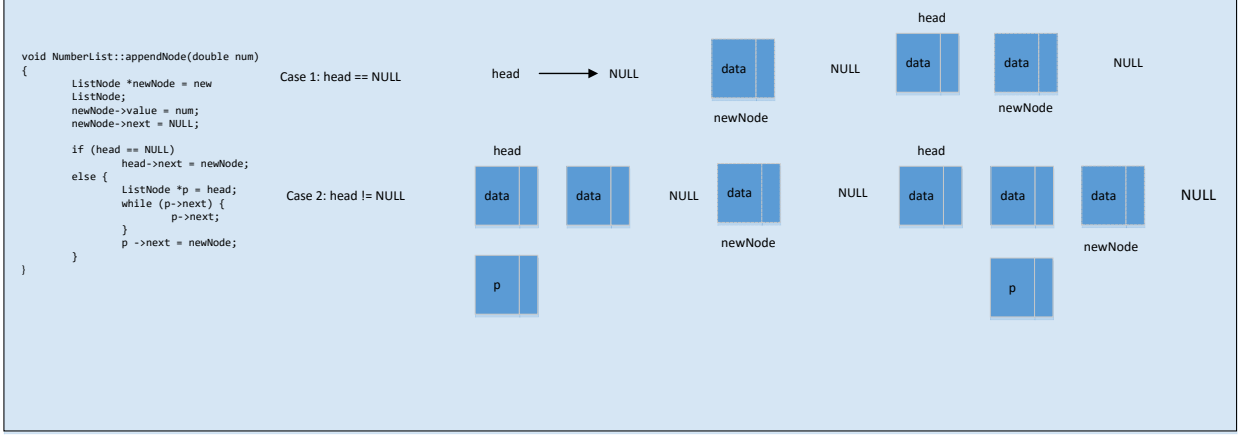
`list_3358_LL.cpp`

We will test it using the original list_3358_LL.h file and our own test driver.

appendNode

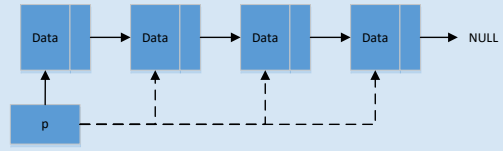
```
void NumberList::appendNode(double num)
{
        ListNode *newNode = new
        ListNode;
        newNode->value = num;
        newNode->next = NULL;

        if (head == NULL)
                head->next = newNode;
        else {
                ListNode *p = head;
                while (p->next) {
                        p->next;
                }
                p ->next = newNode;
        }
}
```



displayList

```
void NumberList::displayList(double num)
{
        ListNode * p = head;
        while (p) {
                cout << p->value << " ";
                p = p->next;
        }
        cout << endl;
}
```



insertNode:
insert a new node into the middle of a list.
Uses two extra pointers: one to point to node before insertion point
- one to point to node before the insertion point [this one is optional]
- one to point to the node after the insertion point

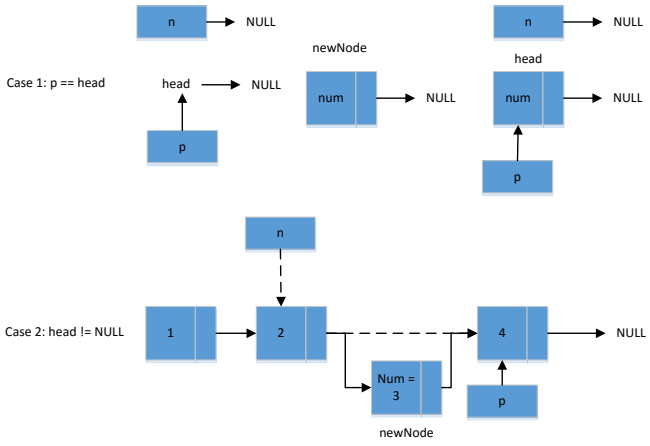
Create the new node, store the data in it
Use pointer p to traverse the list,
until it points to: node after insertion point or NULL
--as p is advancing, make n point to the node before
if p points to first node (p is head, n was not set)
make head point to new node
make new node point to p's node
else
make n's node point to new node

```
void NumberList::insertNode(double num) {

    ListNode *newNode = new ListNode;
    newNode->value = num;

    ListNode *p = head;
    ListNode *n = NULL;
    while (p && p->value < num) {
        n = p;
        p = p->next;
    }

    if (p==head) {
        head = newNode;
        newNode->next = p;
    } else {
        n->next = newNode;
        newNode->next = p;
    }
}
```



deleteNode:
use p to traverse the list, until it points to num or NULL == as p is advancing, make n point to the node before it.

use p to traverse the list, until it points to num or NULL
--as p is advancing, make n point to the node before it
if (p is not NULL) //found!
	if (p==head) //it's the first node, and n is garbage
		make head point to the second element
		delete p's node (the first node)
	else
		make n's node point to what p's node points to
		delete p's node
else: . . . p is NULL, not found do nothing

```
void NumberList::deleteNode(double num )
{
	ListNode *p = head;
	ListNode *n = NULL;

	while (p && p->value < num){
		n = p;
		p = p->next;
	}

	if (p) // p is not NULL so found
	{
		if (p == head) {
			head = head->next;
			delete p;
		} else {
			n->next = p->next;
			delete p;
		}
	}
}
```