COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 6

- **Sequence class** (using linked list)

In this project, first complete the implementation of linked list toolkit functions. Then complete the implementation of the sequence class. Please note that we do not use all of these functions for the implementation of the sequence class. For example, `list_remove_dups()` and `list_detect_loop()` are not used by the sequence class. You can use `node_test.cpp` to test the operation of toolkit functions.

The **sequence** class stores items in a forward linked list, *in their sequence order*. The "current" item on the list can be maintained by a member variable that points to the node that contains the current item. When the `start` function is activated, we set this "current pointer" to point to the first node of the linked list. When `advance` is activated, we move the "current pointer" to the next node on the linked list.

We propose five private member variables for the new sequence class. The first variable, `many_nodes`, keeps track of the number of nodes in the list. The other four member variables are node pointers:

- `head_ptr` and `tail_ptr:` The head and tail pointers of the linked list. If the sequence has no items, then these pointers are both NULL. The reason for the tail pointer is the `attach` function. Normally this function adds a new item immediately after the current node. But if there is no current node, then attach places its new item at the tail of the list, so it makes sense to keep a tail pointer around.
- `cursor:` Points to the node with the current item (or NULL if there is no current item).
- `precursor`: Points to the node before current item (or NULL if there is no current item or if the current item is the first node). Can you figure out why we propose a `precursor`? The answer is the `insert` function, which normally adds a new item immediately before the current node. But the linked-list functions have no way of inserting a new node before a specified node. We can only add new nodes after a specified node. Therefore, the `insert` function will work by adding the new item after the `precursor` node, which is also just before the `cursor` node.

Keep in mind the four rules for a class that uses dynamic memory:

- Some of your member variables are pointers. In fact, for your sequence class, four member variables are pointers.
- Member functions allocate and release memory as needed. Don't forget to write documentation indicating which member functions allocate dynamic memory so that experienced programmers can deal with failures.
- You must override the automatic copy constructor and the automatic assignment operator. Otherwise two different sequences end up with pointers to the same linked list.
- The class requires a destructor, which is responsible for returning all dynamic memory

to the heap.

The **value semantics** of your new sequence class consists of a copy constructor and an assignment operator.
The primary job of both these functions is to make one sequence equal to a new copy of another. The sequence that you are copying is called the "`source`," and we suggest that you handle the copying in these cases:

1. If the `source` sequence has no current item, then simply copy the source's linked list with `list_copy.` Then set both `precursor` and `cursor` to the null pointer.
2. If the `current` item of the `source` sequence is its first item, then copy the source's linked list with `list_copy.` Then set `precursor` to NULL, and set `cursor` to point to the head node of the newly created linked list.
3. If the current item of the source sequence is after its first item, then copy the `source's` linked list in two pieces using `list_piece`. The first piece that you copy goes from the head pointer to the `precursor`; the second piece goes from the `cursor` to the tail pointer. Put these two pieces together by making the link field of the `precursor` node point to the cursor node. The reason for copying in two separate pieces is to easily set the `precursor` and `cursor`.

Given files:

- `node.h` - Header file for the node class with descriptions for each function.
- `node.cpp` - Incomplete implementation file for the node class.
- `sequence.h` – Header file for the sequence class with descriptions for each function.
- Test files for the node and sequence classes