Project 2B: Extracting Sorted Sublists

For this project we will work with the `List` class, the doubly-linked list discussed during lecture. You will find the `Node` and `List` classes on Blackboard under Course Materials / Project2.

First you **must read the `List` interface** and understand how it works. You will need to know how to use `List` objects by understanding its interface.

You are asked to add to the List class. The added methods are for the purpose of this project only and not general enough to be part of a general List ADT, rather good algorithmic/programming exercise.

Note: exam questions will be directly based on projects and ADTs/classes discussed in lecture. Reading and understanding the List class will be fundamental to your success on the exams as well as on the project.

Implementation:

You will modify the `List` class to add the one **public** member function

```
/**
@pre assumes position is valid, if position is > item_count_ it returns an
empty List, also assumes that operators <= and >= are defined on type T
@param position contained in the sorted/increasing (first <= position <=
last) sublist to be generated
@return a sublist containing the item at position consisting of sorted/
increasing items (first <= position <= last)
*/
List<T> scanSublist(size_t position);
```

This function takes a position as parameter, and returns a sublist of the calling List that is sorted in increasing order and it contains the item at position.

For example, if the original list contains integers [3,4,5,6,3,4,5,9,5,4,3,2,5,7,4], a call to `scanSublist(5)` will return [3,4,5,9], `scanSublist(3)` will return [3,4,5,6] and `scanSublist(9)` will return [4].

The goal is to take advantage of the multi-directionality of the doubly-linked list and to produce the sublist by moving backwards and forward from position using the previous_ and next_ pointers. Your algorithm should do something like:

- *start at position*
- *move left (previous) until you find an item that is >= or you have reached the beginning of the list*
- *Now start moving right (next) adding items to the sublist until you find an item that is <= or you reach the end of the list*

In this way, **the amount of "work" necessary to obtain the sublist should be proportional to the size of the sublist, not to the size of the original list.**

IMPORTANT: You may not use vectors or other containers to do the work. You are expected to obtain the sublist by direct manipulation of the original List object without using intermediate containers to hold the List items. OTHER CONTAINERS SHOULD NOT APPEAR IN COMMENTS EITHER, so if you use them for testing (you shouldn't need to) you must delete them before submitting.

Extra Credit:

Overload the <= and >= operators for the class `CourseMember` to observe the behavior of `scanList` on `List<CourseMember>` to extract in lexicographical order sublists of CourseMebers with the same last name initial (considering only first two letters of last name) as that found at position. If the list is sorted by last name, it will extract all CourseMembers with same last name initial as that in position.

```
/**compares first two letters of lastname to determine relationship
@param lhs rhs the two CourseMember objects to be compared
@return true if lhs.last_name_ and rhs.last_name start with same character
and <= holds for second character
//e.g. Alemani <=  Laranga <= Limoni <= Ligorio <= Zion
// where Limoni <= Ligorio (==) and Laranga <= Ligorio (<)
 */
friend bool operator<=(const CourseMember& lhs, const CourseMember& rhs);


/**compares first two letters of lastname to determine relationship
@param lhs rhs the two CourseMember objects to be compared
@return true if lhs.last_name_ and rhs.last_name start with same character
and >= holds for second character
//e.g. Zion >= Ligorio >= Limoni >= Laranga >= Aleman
// and Limoni >= Ligorio (==) and Ligorio >= Laranga (>)
*/
friend bool operator>=(const CourseMember& lhs, const CourseMember& rhs);
```

Testing:

Gradescope will generate bags of integers and test your functions for correctness. You should do the same. To help you with testing I provided a List::traverse function that will traverse and print every item in the List. Again, this method is for project purposes only, it is not general and it expects that the type T the List holds can be sent to standard output (std::cout << defined for type T). For this project we will be testing with integers, so it will work. For testing the extra credit read below.

How to test your function:

You do not submit your test code, but as your projects become more complex **it is of utmost importance that you test your code incrementally and thoroughly** before submitting and that you **develop the skill and habit** to do so.

Write a `main()` function that instantiates a `List<int>` object, populates it with integers and calls `scanSublist()` with different test cases, e.g.

```
List<int> test_list;
// add integers to test_list;
List<int> sub_list = test_list.scanSublist(pos);//for some integer pos
```

Make sure you test different cases by carefully crafting the order of the items in your list and calling `scanSublist` with different positions.

Edge-cases include:

- `pos = 0`
- `pos = test_list.getLength()`
- `pos = x for 0 < x < test_list.getLength()`
- The sublist includes the first element of the original list
  e.g. `test_list=[3,4,5,6,3,4,5,9,5,4,3,2,5,7,4]` and `sub_list=[3,4,5,6]`
- The sublist includes the last element of the original list
  e.g. `test_list=[3,4,5,6,3,4,5,9,5,4,3,7,5,7,9]` and `sub_list=[5,7,9]`
- Test position past the end of the list (`scanList` should return an empty list not crash)
- Test a case where `sub_list.getLength() = 1`

Testing the Extra Credit:

You can adapt some of the code from Project1C to read in CourseMembers into a List<CourseMember> from roster.csv and test `scanSublist.` Another function can display the names of the CourseMembers in your sublist to check that you are extracting all CourseMembers with same last name initial.
These can be helper functions in your main.cpp.

```
List<CourseMember> createListFromInput(std::string input_file)

void displayCourseMembersInList(const List<CourseMember>& cm_list)
```

Submission:

For the regular submission you must submit the implementation of List with your added function only - **1 file: List.cpp**
For the extra credit submit **CourseMember.cpp** and re-submit **List.cpp** (2 files).

NOTE: If for any reason we look at **your code** and it is **not commented** (your name added in the top preamble stating your modifications and the functions you add thoroughly commented), **you will lose 10 points**.

Your project must be submitted on Gradescope. The due date is Friday March 8 by 6pm. No late submissions will be accepted.

Have Fun!!!!!