1 Goals

- Additional practice implementing linked lists;
- Practice with abstract classes and template (generic) types;
- Practice implementing copy and move constructors, copy and move assignment operators;
- More practice with unit testing;
- Running and analyzing performance tests.

Note that you may use whatever environment you like for this class, but your programs must be able to compile and run on ada (which is running Ubuntu) using g++, cmake, make, valgrind, and gdb. It is highly recommended, however, that you use the Department-supplied virtual machine (VM) on your computer for this class (which is also based on Ubuntu).

2 Instructions

1. Accept the GitHub classroom repo for HW-2, and clone it to your local environment. See piazza for the classroom link.

2. Read through the unit tests provided in hw2.test.cpp. You will need to understand and explain the basic idea of each test in your assignment write up (see below). Note that the unit tests can also help with questions regarding the functions defined in Sequence and LinkedSeq.

3. Implement the LinkedSeq functions. See below for additional details.

4. Ensure your LinkedSeq function implementations pass all provided unit tests. (As before, it is good to work incrementally one function at a time, while checking your work against the unit tests.) You will not be required to write additional unit tests for this assignment. Note that you are not allowed to modify any of the unit tests provided. You are free to add additional tests as needed, but you will not be graded on them. Note that the unit tests provided are not comprehensive, i.e., even if all unit tests for your implementation pass, there still could be issues (which may reveal themselves in the performance tests).

5. Ensure your implementation does not have memory issues as reported by valgrind. To run valgrind, use the command: valgrind ./hw2.test. This will run the tool over the unit tests, and will report any memory leaks or other memory issues in your implementation.

6. Run the performance tests via the hw2.perf executable. As in HW-1, you will need to redirect the output of the tests to a file output.dat. Once generated, use the provided gnuplot script to create graphs from your results. For this assignment, three graphs will be
created. One graph is for testing the erase function, one for the insert function, and one for updating values in the linked sequence. Each operation (erase, insert, update) includes a comparison between your LinkedSeq implementation and the standard C++ linked list (i.e., std::list) implementation over three general cases (beginning, middle, and end of sequence) and increasingly larger sequence sizes.

7. Create an assignment write up that includes the three performance graphs, an explanation of the results reported in the graphs (similar to HW-1), and an explanation of each unit test provided in hw2.test.cpp. Your explanations should not just restate the unit tests, but instead should explain the purpose of the test (i.e., what aspect of LinkedSeq is being tested) and a high-level of description what each test does. In some cases, the tests are straightforward and need little explanation. In other cases, the tests are more involved and need greater explanation. Your job is to demonstrate to me that you looked at and understand the tests. Finally, your write up should also contain a brief paragraph on any implementation issues and/or challenges you ran into and how you addressed them (if applicable).

3 Additional Details and Requirements

**Stream Insertion Operator.** You must implement the string insertion operator for LinkedSeq. See the corresponding unit test for how to properly format the output of LinkedSeq objects.

**Constructors.** Since we are using C++ member variable initialization, the default constructor is empty. However, you will need to implement both a copy and move constructor. Your copy constructor should call the overloaded copy assignment operator. Similarly, your move constructor should call your overloaded move assignment operator (similar to the copy case but by also calling std::move()).

**Assignment Operators.** You must implement both a copy and move assignment operator. Your copy assignment operator must make a complete copy of the incoming linked list. This should be done without calling any LinkedSeq functions except for make_empty(). The general algorithm is to “empty” the current object and then traverse the incoming linked list nodes, making corresponding copies of each node within the current object as you are navigating. Your move assignment operator should also call make_empty(), should not call any other LinkedSeq functions, should have the current link list refer to the incoming linked list, and should “null out” the incoming structure. Be sure to also correctly set the node counts of the current and incoming objects.

**Destructor.** You must implement a destructor. Your destructor can reuse your make_empty() function.

**Bounds checking.** Your index operator, insert, and remove functions must check that valid indexes are provided. If a valid index is not provided, your functions must throw a std::out_of_range exception. The corresponding message for the out of range exception should be the qualified name of the function, e.g., "LinkedSeq<T>::operator[]", "LinkedSeq<T>::insert()", etc.
**Insert and access special cases.** Inserting at the end of a `LinkedSeq` is a special case, which must be handled by using the `LinkedSeq`'s `tail` pointer. Similarly, returning or updating the last element of a `LinkedSeq` (via the appropriate index operator) is a special case, which must be handled by using the `tail` pointer. Using the `tail` pointer in this way will avoid having to traverse the entire linked list for insertion and element access, which means faster execution compared to doing the same operation on an index somewhere “in the middle” of the sequence.

**The sort function.** We are including a non-working `sort()` function in HW-2, which we will implement in a later assignment. For now, you only need to leave the `sort()` function empty.