CS 271 Project 0 – Doubly Linked List

Instructor: Flannery Currin

Due: September 5th, 2025 by 9:00 AM

1 Learning Goals

Welcome back! It has been at least a few months since you took CS 173, so this project is meant to get your rusty C++ skills back up to speed and build upon them. Specifically, after working on this project, you should:

- Be able to implement a template class in C++ (why write 5 or 6 classes when you could write just one?)
- Be able to navigate and manipulate a doubly linked list efficiently, understanding how it differs from a singly linked list
- Know what private members of a doubly linked list allow for this implementation of the list ADT
- Manage dynamic data appropriately
- Be able to instantiate C++ objects and structs
- Test your implementation thoroughly, using multiple datatypes for elements of the list and thinking through expected outputs for a variety of edge cases

2 Project Overview

This is an individual project. You may discuss this assignment with the course TA or instructor, but the work you submit must be your own work. You may find it useful to reference your linked list stop-gap project from CS 173.

Before we get to more complex data structures, let's revisit our old friend the list. In CS 173, you implemented a list ADT using an array and a singly linked list. The singly linked list can only be navigated in one direction: starting at the head and traversing through each subsequent node. Your task now is to implement the list ADT with a doubly linked list which can be traversed both from head to tail and tail to head. To do this:

- Each `DoublyLinkedList` object needs a pointer to the head (first node in the list) and tail (last node in the list).

- Each node contains three members: a value, a pointer to the next node in the list (or `nullptr` for the tail), and a pointer to the previous node in the list (or `nullptr` for the head).

`DoublyLinkedList.h` contains the public interface for this class – **do not modify the public method declarations or the `to_string` method definition**. The minimum required private members are also provided: the Node struct and head and tail pointers. You should not modify these, but you can add other private members as needed for your implementation.

Instead of overloading `operator<<`, this class implements a `to_string` method which returns a string representation of the list. This allows the unit tests in `test_dll_example.cpp` to compare the actual contents of a list object to the expected contents of the list object.

Your task is to:

- Create `DoublyLinkedList.cpp` and define the methods (except `to_string`) declared in `DoublyLinkedList.h`. The Linked List ADT provides descriptions of what each of these methods should do if you need a refresher. Include appropriate documentation – see the style guide on Canvas for a refresher.

- Use `test_dll_example.cpp` to create and run unit tests for your implementation. There are some basic tests included already as an example to get you started, **but these tests are insufficient**. You should test your implementation much more rigorously: adding tests that cover more cases, *using datatypes other than int for T*, and overall creating a test suite that could catch errors your classmates might make.

Note: The tests assume that out-of-bounds indices are handled by raising a runtime exception. For the purposes of this project, you can use the `runtime_error` class defined in `<stdexcept>`. Example usage:

```
throw std::runtime_error("Bad non-descriptive error message!");
```

You should focus on creating an implementation that is correct first. Test often as you go, and comment out sections that are not yet implemented. **To receive credit for the out-of-class component of the project, your submission must compile on the linux machines**. Test on the linux machines (review the guides on Canvas if you need). If you have trouble with a specific method but want to receive partial credit for completed work, **comment out** (do not delete) the incomplete method definitions, declarations, and tests before submitting.

To receive full efficiency points, you should focus on the time complexity of your implementation of methods. This may require you to add additional private members to the class. Can we make `length` $\Theta(1)$ somehow? What was the time complexity of `append` with a singly linked list? Can we do better with a doubly linked list?

If you have a complete, correct, efficient, thoroughly tested project and you are feeling jazzed or wanting to get a head start on project 1, implement a private `swap` helper method that takes pointers to two nodes in a DoublyLinkedList and swaps the values contained in those nodes. What could project 1 be?

3 Project Submission

On Canvas, before the deadline, you will submit **a zip** containing:

- `DoublyLinkedList.h` with any updates to private members you made
- `DoublyLinkedList.cpp` with your method implementations
- `test_dll_example.cpp` with your complete set of unit tests

You do not need to submit the makefile. For future projects, you should be able to create your own makefile if you want or compile your projects with `g++`.

Late submissions will not be accepted, but you can submit updated versions of your project as you go up until the deadline. If you submit a version on Thursday night and an updated version Friday morning before the deadline, the most recent submission will be graded.

In class on the due date, there will be a brief written component of the project (for example, you may need to implement a method similar to but not identical to a method you implemented outside of class). This component will be focused on the learning goals outlined in Section 1.

4 Grading Scheme

The out-of-class submission will be graded using the following scheme:

Criteria	**Description**	**Points**
Completeness	Meets submission requirements (files, documentation, etc.)	2
Correctness	Passes Dr. Currin's extended tests and follows specs (no singly linked lists)	4
Efficiency	Solutions will be clocked to check time complexity	1
Testing	Tests should cover a range of types and cases, and logic should be sound	3
Total		**10**

The in-class component is worth 20 points. You should not need to study extra or memorize your out-of-class work for the in-class component. Working on the out-of-class component with the learning goals in mind is your best preparation. The in-class project component lets you test how prepared you are to apply the concepts covered by the project to a new scenario.