Doubly-linked list implementation

Objectives

- Experience writing and interacting with C++ classes for the implementation of dynamic structures

Grading breakdown

Points	
100	Runs correctly
100	**MAX TOTAL POINTS**

Overview

We introduced singly-linked lists in the lecture and went over their implementation. In this independent homework assignment, you will implement a doubly-linked list.

Recall the difference between a singly-linked list (figure 1) and a doubly-linked list (figure 2):

- A singly-linked list only contains a pointer to the next node in the list
- A doubly-linked list contains pointers to the next and previous node in the list

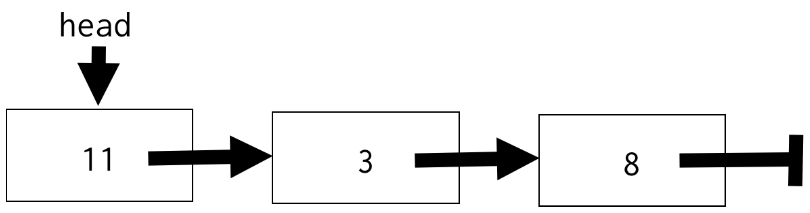

Figure 1. Singly-linked list.

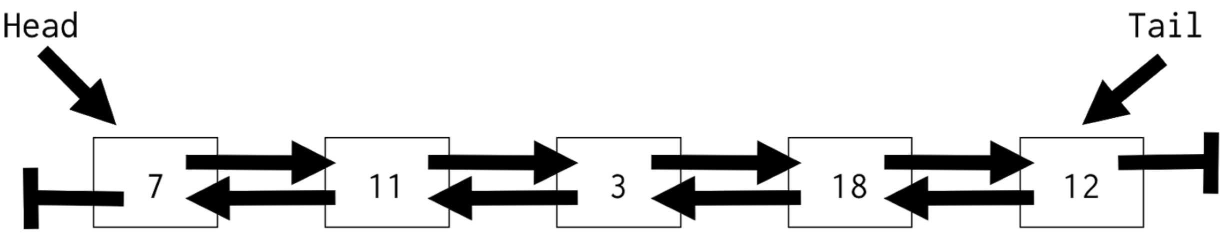

Figure 2. Doubly-linked list.

Requirements & Roadmap

Starter code

Get Starter Code

Test cases

You will need to design different test cases in order to test your implementation of a doubly-linked list. You should do this first.

Implementation

Submit only Node.h and DoublyLinkedList.h to Mimir.

You will first implement the following struct template for a Node<T>:

struct Node<T>
+ data : T
+ next : Node<T>*
+ prev : Node<T>*
+ Node(T)

The doubly-linked attributes and behaviors will be encapsulated together in a class template as follows:

class DoublyLinkedList<T>
- head : Node<T>* Stores a pointer to the first node (front) of the doubly-linked list.
- tail : Node<T>* Stores a pointer to the last node (back) of the doubly-linked list.
- sz : unsigned int Stores the number of nodes in the doubly-linked list (i.e., its size).
+ DoublyLinkedList()

	Default constructor: initializes the head and tail with nullptr, size to 0.
+	`DoublyLinkedList(T data)` The parameterized constructor initializes head and tail to nullptr, size to 0. Constructs a new node on freestore with data, assigns the address of this newly created node to both head and tail and increments size (sz) by 1.
+	`DoublyLinkedList(const DoublyLinkedList<T>&)` Copy constructor initializes one DoublyLinkedList with another; performs a deep copy of the doubly-linked list bound to the function's parameter.
+	`operator=(const DoublyLinkedList<T>&) : DoublyLinkedList<T>&` Copy assignment operator performs a deep copy of the doubly-linked list bound to the function's parameter.
+	`~DoublyLinkedList()` Destructor calls clear to destruct a doubly-linked list.
+	`front() : T&` Returns a reference to the first node's data in the doubly-linked list. You may assume that the list is not empty. (The appropriate way of addressing this would be to use C++ exceptions, but we are not covering them this semester.)
+	`front() const : const T&` Returns a constant reference to the first node's (head's) data in the doubly-linked list. You may assume that the list is not empty.
+	`back() : T&` Returns a reference to the last node's (tail's) data in the doubly-linked list. You may assume that the list is not empty.
+	`back() const : const T&` Returns a constant reference to the last node's (tail's) data in the doubly-linked list. You may assume that the list is not empty.
+	`size() const : unsigned int` Returns the value stored in the `sz` data member.

+	`clear() : void`	Deallocates memory for each node in the doubly-linked list. Afterward, assigns nullptr to head and tail, assigns 0 to sz.
+	`insert(T data, unsigned int idx) : void`	Inserts a node with data at “index” idx in the doubly-linked list. If idx >= sz, immediately return from the function. Otherwise, ensure that the node is correctly inserted into the list at the specified position. This operation increases the size (sz) of the list by one.
+	`erase(unsigned int idx) : void`	Removes the node at “index” idx from the doubly-linked list. This operation decreases the size (sz) of the list by one.
+	`push_back(T data) : void`	Adds a node with data to the end of the doubly-linked list. This operation effectively increases the size (sz) of the list by one and makes an assignment to tail.
+	`pop_back() : void`	Removes the last node (i.e., node residing at the tail position) from the linked list and deallocates its memory. This operation effectively reduces the size of the list by one and makes an assignment to tail.
+	`push_front(T data) : void`	Adds a node with data to the front of the doubly-linked list. This operation effectively increases the size (sz) of the list by one and makes an assignment to head.
+	`pop_front() : void`	Removes the first node (i.e., node residing at the head position) from the linked list and deallocates its memory. This operation effectively reduces the size of the list by one and makes an assignment to head.
+	`to_str() : std::string`	Provided. Returns an std::string showing the nodes accessible from head.

Along with the following non-member helper functions

`operator==(const DoublyLinkedList<T> &lhs, const DoublyLinkedList<T> &rhs)  : bool` Performs an element-wise comparison of the lhs the rhs. This function will return true only if the doubly-linked lists are the same size and each corresponding node has the same value.
`operator!=(const DoublyLinkedList<T> &lhs, const DoublyLinkedList<T> &rhs)  : bool` Performs an element-wise comparison of the lhs the rhs. This function will return false only if the doubly-linked lists are the same size and each corresponding node has the same value.
`operator==(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data == rhs.data`).
`operator!=(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data != rhs.data`).
`operator<(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data < rhs.data`).
`operator<=(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data <= rhs.data`).
`operator>(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data > rhs.data`).
`operator>=(const Node<T> &lhs, const Node<T> &rhs)  : bool` Returns true only if (`lhs.data >= rhs.data`).
`operator<<(std::ostream& os, const Node<T> &rhs) : std::ostream` Inserts the node's data into the stream (`os << rhs.data << std::endl`).
`operator<<(std::ostream& os, const DoublyLinkedList<T> &rhs) :: std::ostream&` Provided. Uses to_str() to insert the DoublyLinkedList<T> object to the std::ostream.