* Import LinkedList, DoublyLinkedList, LinkedStack, ListStack, LinkedQueue, and ListQueue.

from linkedlist import LinkedList from doubly\_linkedlist import DoublyLinkedList from stacks import LinkedStack, ListStack from queues import LinkedQueue, ListQueue

You **must** use the provided data structures to solve the following problems. You will automatically receive a **zero** for a problem if you do not use those provided data structures to solve the problem.

* **Add the following methods to *yourLastName*\_*yourFirstName*\_assignment4.py**.
* **Test each method with the provided test case.**

1. Add a method: **def diff(stack1: LinkedStack, stack2: LinkedStack) -> LinkedStack:**

diff() method accepts two LinkedStacks and returns a LinkedStack with the union of elements in two stacks. Test case:

stack1: 1, 2, 3, 4, 5 stack2: 3, 4, 5, 6, 7

This method should return a stack with elements 1, 2, 3, 4, 5, 6, 7

1. Add a method: **def remove\_max(stack: ListStack) -> ListStack:**

remove\_max() finds and removes the maximum element in a linked stack of integer, and returns the resulting stack.

Test case:

The given stack is 2, 1, 9, 8, 7, 4, 5, 6

Top element

This method should return the stack 2, 1, 8, 7, 4, 5, 6 after the maximum element 9 is removed

Top element

1. Add a method: **def copy\_queue(queue: LinkedQueue) -> ListQueue:**

copy\_queue() uses the elements in linked queue to create a list queue.

Test case:

The given linked queue is 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'

Front Rear

The return list queue should be 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'

Front Rear

1. Add a method: **def swap(stack: ListQueue) -> ListQueue:** swap() swaps the front and rear elements in a queue.

Test case:

The given queue is 'A' → 'B' → 'C' → 'D' → 'E' → 'F'

front element rear element

This method should return 'F' → 'B' → 'C' → 'D' → 'E' → 'A'

Swap

1. Add a method: **def compare(queue1: LinkedQueue, queue2:LinkedQueue) -> bool:**

compare() compares two LinkedQueues and returns a boolean to indicate whether they have the same set of elements or not. Assume all the elements are unique in a list.

Test case:

queue1: 1→2→3→4→5 queue2: 3→4→5→1→2

This method should return True

1. Add a method: **def merge(q1: ListQueue, q2: ListQueue) -> ListQueue:**

merge() merges two sorted ListQueues of Integer together into a new sorted LinkedQueue and returns it. Test case:

Before merging, q1 is 2, 11, 19, 21, 23, 24

q2 is 3, 9, 15, 16, 22

After merging, the merged queue is 2, 3, 9, 11, 15, 16, 19, 21, 22, 23, 24