- This lab will cover Linked Lists.
- It is assumed that you have reviewed chapter 6 & 7 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time. Ideally you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins

1. During lecture you learned about the different methods of a doubly linked list.

   Provide the following worst-case runtime for those methods:

```
a.   def __len__(self):


b.   def is_empty(self):

c.   def _insert_between(self, e, predecessor,
successor):

D.   def _delete_node(self, node):
```

2. What is this function doing to the linked list example above?

   Draw the result of the doubly linked list structure after executing the function using the doubly linked list example from question 1. Give the function a name.

```
def mystery(dll):

     dll._header.next.prev = dll._trailer.prev
     dll._trailer.prev.next = dll._header.next

     dll._header.prev = dll._trailer
     dll._trailer.next = dll._header

mystery(dll)
```

3. Trace the following function. What is the output of the following code using the doubly linked list from question 3? Give mystery an appropriate name.

```
#dll = Doubly Linked List
def mystery(dll):

     if len(dll) >= 2:
          node = dll._trailer.prev.prev
          node.prev.next = node.next
          node.next.prev = node.prev

          node.next = None
          node.prev = None
          return node

     else:
          raise Exception("dll must have length of 2 of
          greater")

print(mystery(dll))
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

Download the **DoublyLinkedList.py** file attached with HW6 on NYU Classes

1. In class, we defined the `stack ADT` using a dynamic array as the underlying data structure. Because of the resizing, the ArrayStack run-time for its operations is not exactly in Θ(1). Instead, the cost is Θ(1) amortized. (10 minutes)

   Define the stack ADT that guarantees each method to always run in Θ(1) **worst case**.

```
class LinkedStack:

    def __ init__(self):
          …

    def __len__(self):
    ''' Returns the number of elements in the stack. '''


    def is_empty(self):
    ''' Returns true if the stack is empty,false otherwise.
    '''


    def push(self, e):
    ''' Adds an element, e, to the top of the stack. '''


    def top(self):
    ''' Returns the element at the top of the stack.
        An exception is raised if the stack is empty. '''


    def pop(self):
    ''' Removes and returns the element at the top of the
    stack.
        An exception is raised if the stack is empty. '''
```

2. You will create the MidStack using a **Doubly Linked List** with Θ(1) extra space. All methods of this MidStack should have a Θ(1) run-time.

   MidStack has this feature where we can use the function mid_push to add an element to the middle of the stack.

   The middle is defined as the $(n+1)//2\ th$ element, where n is the number of elements in the stack.

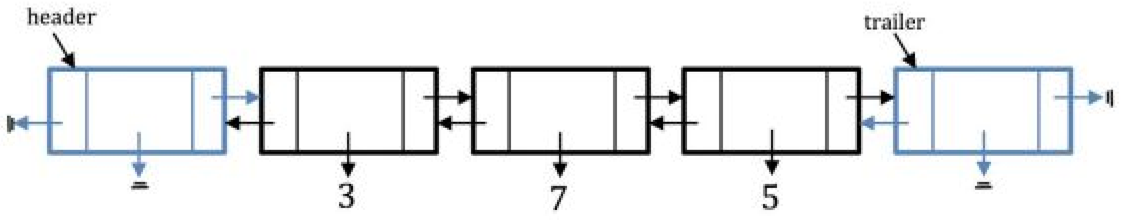
   Hint: To access the middle of the stack in constant time, you may want to define an additional data member to reference the middle of the Doubly Linked List.

```
class MidStack:

    def __ init__(self):
         self.data = DoublyLinkedList( )
         …

    def __len__(self):
    ''' Returns the number of elements in the stack. '''


    def is_empty(self):
    ''' Returns true if stack is empty and false otherwise.
    '''


    def push(self, e):
    ''' Adds an element, e, to the top of the stack. '''


    def top(self):
    ''' Returns the element at the top of the stack.
        An exception is raised if the stack is empty. '''


    def pop(self):
    ''' Removes and returns the element at the top of the
    stack.
        An exception is raised if the stack is empty. '''
```

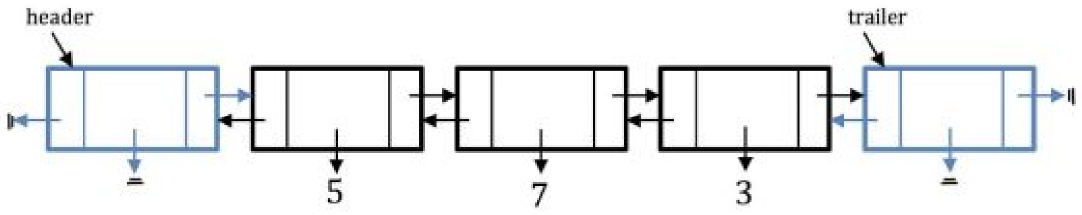
```
def mid_push(self, e):
''' Adds an element, e, to the middle of the stack.
    An exception is raised if the stack is empty. '''
```

3. Implement a method to reverse a doubly linked list. This method should be non-recursive and done **in-place** (do not return a new linked list).

   For example if your list looks like:



   After calling the method on it, it will look like:



   You will implement the reversal in two ways:

   a. First implement a function which reverses the data in the list, but does not move any nodes. Instead, change the value at each node so that the order is reversed.

```
def reverse_dll_by_data(dll):
''' Reverses the linked list '''
```

   b. Next, implement a function which reverses the order of the nodes in the list. That is, you should move the nodes objects around, without changing their data value and without creating any new node objects.

```
def reverse_dll_by_node(dll):
''' Reverses the linked list '''
```