- This lab will cover Linked Lists, Stacks, and Queues.
- 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 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.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (maximum 20 minutes)

---

1. Draw the state of the deque object (memory image) and show how the values are placed in the circular array structure as the following code executes.

```
from ArrayDeque import *

dq = ArrayDeque()

dq.add_first(3)
dq.add_last(4)
dq.add_first(2)

dq.add_last(5)
dq.add_first(1)

dq.delete_last()
dq.add_first(0)
```

2. Draw the state of the linked list object as the following code executes:

```
from DoublyLinkedList import DoublyLinkedList


dll = DoublyLinkedList()
dll.add_first(1)
dll.add_last(3)
dll.add_last(5)
dll.add_after(dll.first_node(), 2)
dll.add_before(dll.last_node(), 4)
dll.delete_node(dll.last_node())
dll.add_first(0)

print(dll)
```

What is the output of the code?

3. 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 2. 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)
```

4. Trace the output of the following function call:

```
print(calculate_mystery("( ( ( 60 + 40 ) / 50 ) * ( 16 - 4 ) )"))
```

```
def calculate_mystery(string_input):
    numbers = ArrayStack()
    operators = ArrayStack()
    symbols = {"+", "-", "*", "/"}
    expression_list = string_input.split()

    for item in expression_list:

        if item.isdigit():
            numbers.push(int(item))

        elif item in symbols:
            operators.push(item)

        elif item == ")":           # evaluate
            rhs = numbers.pop()
            lhs = numbers.pop()
            operation = operators.pop()

            if operation == "+":
                res = lhs + rhs
            elif operation == "-":
                res = lhs - rhs
            elif operation == "*":
                res = lhs * rhs
            elif operation == "/":
                res = lhs / rhs

            numbers.push(res)
            print(res)

    return numbers.top()
```

Coding

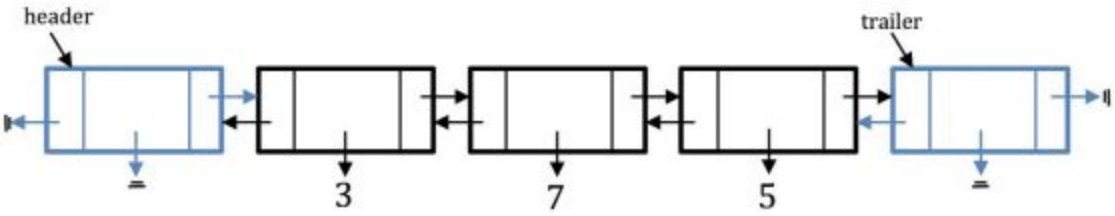
In this section, it is strongly recommended that you solve the problem on paper before writing code.

1. Implement a function to **recursively** sum the values in a doubly linked list. You can assume that the elements of the list are numbers. You may define a helper function with additional parameters that will compute the sum in O(n) run-time.

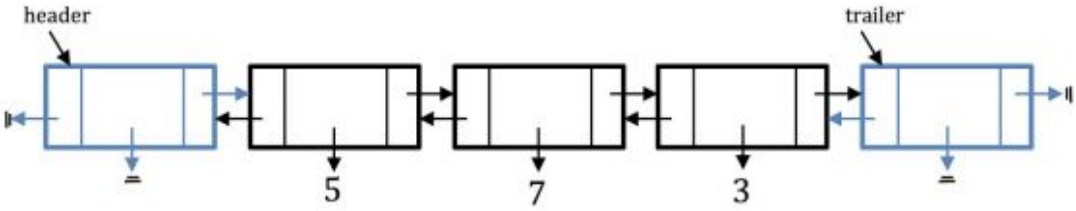
```
def sum_dll(dll):
    …
    return sum_dll_helper( … )


def sum_dll_helper( … ):
    '''Return the sum of the values in the linked list'''
```

2. 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 '''
```

3. In homework 5, you were asked to implement a `MidStack` ADT using an ArrayStack and an ArrayDeque. This time, you will create the MidStack using **Doubly Linked List** with Θ(1) extra space. All methods of this MidStack should have a O(1) run-time.

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

   Hint: To have access to 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 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. '''
```

4. In homework 1, you implemented a generator function that produces the fibonacci sequence. The sequence is as followed 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … , with every subsequent number being the sum of the previous 2 numbers.

   For this question, you will implement a generator function for the n - bonacci sequence. Every n - bonacci sequence starts off with n 1's, and every subsequent number in the sequence is the sum of the previous n numbers. The function will take in two parameters, n for the bonacci value, and k, indicating the first k values in that sequence.

   Note that for n = 2, you will get the fibonacci sequence.

```
def n_bonacci(n, k):
    """
    : n, k type: int
    : yield type: int
    """
```

   ex) 5 - bonacci would look like this:

```
for i in n_bonacci(5, 9):       #first 9 values
    print(i, end = " ")   #1, 1, 1, 1, 1, 5, 9, 17, 33

for i in n_bonacci(5, 2):  #first 2 values
    print(i, end = " ")   #1, 1
```

   ex) 2 - bonacci (fibonacci) would look like this:

```
for i in n_bonacci(2, 9):  #first 9 values
    print(i, end = " ")   #1, 1, 2, 3, 5, 8, 13, 21, 34

for i in n_bonacci(2, 1):  #first value
    print(i, end = " ")   #1
```

   You may only use an **ArrayQueue** for your solution. To save space, you may want to only hold up to n values at any time.

   Give the worst-case run-time and extra space complexity of the following in terms of n and k:

```
for i in n_bonacci(n, k):
    print(i, end = " ")
```

5. Write a function that **flattens** a nested **Doubly Linked List**. Each node of the Doubly Linked List may contain either an integer or a nested Doubly Linked List of integers. Your functions should return a new doubly linked list with the values. (You should not do this in-place.)

```
def flatten_dll(dll):
    """
    : dll type: DoublyLinkedList
    : return type: DoublyLinkedList
    """
```

Write two implementations of this method, <u>one recursive and one non-recursive.</u>
For the *recursive* implementation, you may define a helper function with your own choice of parameters.

```
def flatten_dll_helper( … ):
```

<u>Hint:</u> You may want to use an ArrayStack for the *non-recursive* function.

ex)

```
#dll is a Doubly Linked List that already has values in it

print(dll)
[1 <--> 2 <--> [3 <--> 4 <--> [5 <--> 6] ] <-->7]

dll = flatten_dll(dll)
print(dll)

[1 <--> 2 <--> 3 <--> 4 <--> 5 <--> 6 <-->7]
```