- This lab will cover Linked Lists and Binary Trees
- It is assumed that you have reviewed chapter 6, 7 & 8 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!
- Your TAs are available to answer questions in the lab, and during office hours.

Part A
Linked Lists
(2:00 PM - 4:20 PM)

Vitamins (20 minutes)

---

1. Draw the memory image of the linked list object as the following code executes. (7 minutes)

```
from Lab5_DoublyLinkedList import DoublyLinkedList


dll = DoublyLinkedList()
dll.add_first(1)
dll.add_last(3)
dll.add_last(5)
dll.add_after(dll.header.next, 2)
dll.add_before(dll.trailer.prev, 4)
dll.delete_node(dll.trailer.prev)
dll.add_first(0)

print(dll)
```

What is the output of the code?

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

   Provide the following worst-case runtime for those methods (5 minutes)

   a. `def __len__(self):`

   b. `def is_empty(self):`

   c. `def add_after(self, node, data):`

   d. `def add_first(self, data):`

   e. `def add_last(self, data):`

   f. `def add_before(self, node, data):`

   g. `def delete_node(self, node):`

   h. `def delete_first(self):`

   i. `def delete_last(self):`

3. Trace the following function. What is the output of the following code? Give mystery an appropriate name (8 minutes)

```
#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 (80 minutes)

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

Download the **Lab5_DoublyLinkedList.py** file on Brightspace

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.

   Define the stack ADT that guarantees each method to always run in Θ(1) **worst case** (20 minutes).

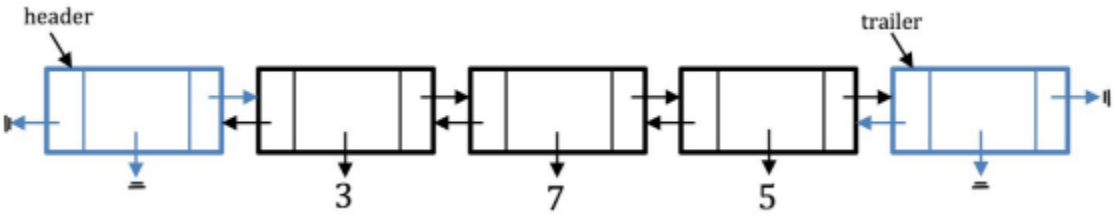
```
class LinkedStack:

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

    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. Implement the following method for the `DoublyLinkedList` class. The get item operator takes in an index, i, and returns the value at the ith node of the Doubly Linked List.

   You only need to support non-negative indices. Your solution should try to optimize the get item method. This means that you should decide whether to iterate from either the header or trailer based on whichever is closer to i. Index should start at 0. (30 minutes)

```
def __getitem__(self, i):
    '''Return the value at the ith node. If i is out of range,
    an IndexError is raised'''
```

   For example, if your Doubly Linked List looks like this:




```
print(dll[0]) # 3 (should iterate from the header)

print(dll[1]) # 7 (either way works)

print(dll[2]) # 5 (should iterate from the trailer)

print(dll[3]) # IndexError
```

   What is the worst-case run-time of the getitem operator? Can we do better?

3. In homework 3, you are asked to implement a `MidStack` ADT using an ArrayStack and an ArrayDeque. This time, 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.

   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. (30 minutes)

```
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. '''
```

Part B
Binary Trees
(4:30 PM - 5:50 PM)

Vitamins (20 minutes)

---

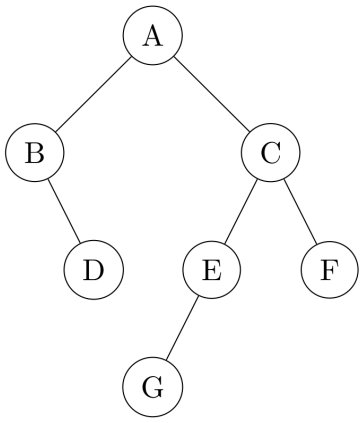
1. Draw the expression tree for the following expressions (given in prefix, postfix, or infix): Remember that the numbers should be leaf nodes. (7 minutes)

    a. 3 4 - 2 + 5 *

    b. (3 * 2) + (4 / 6)

    c. / + 9 9 2

2. Given the following binary tree, complete the following (5 minutes):



a. Give the preorder, inorder, and postorder traversals of the tree:

b. Give the level order traversal (Breadth-First) of the tree:

c. What is the height of the tree?

d. What are the depths of nodes E, B, and G?

3. Draw the binary tree given the following traversals (8 minutes):

preorder : 11, 6, 4, 5, 8, 10, 19, 17, 43, 31, 49

inorder : 4, 5, 6, 8, 10, 11, 17, 19, 31, 43, 49

Is it possible to draw a unique binary tree given only its preorder and postorder? If not, draw two trees with the same preorder and postorder traversal.

Coding (40 minutes)

In this section, it is strongly recommended that you solve the problem on paper before writing code. For each problem, **you may not call any methods defined in the `LinkedBinaryTree` class. Specifically, you should manually traverse the tree in your function. Each node of the tree contains the following references:** `data, left, right, parent`.

Download the **Lab5_LinkedBinaryTree.py** file on Brightspace.

1. Write a ***recursive*** function that returns the sum of all even integers in a `LinkedBinaryTree`. Your function should take one parameter, *root*, which is the root node of the binary tree. You may assume that the tree only contains integers. (10 minutes)

```
def bt_even_sum(root):
    ''' Returns the sum of all even integers in the binary
    tree'''
```

2. Write a **recursive** function that determines whether or not a value exists in a `LinkedBinaryTree`. Your function should take two parameters, a *root* node and *val*, and return True if *val* exists or False if not. (10 minutes)

```
def bt_contains(root, val):
    ''' Returns True if val exists in the binary tree and
    false if not'''
```

3. A **perfect binary tree** is a **binary tree** in which every interior node has 2 children and every leaf node has the same depth.

Implement the following **recursive** function, which takes in the root node of a `LinkedBinaryTree`. The function will be given a parameter, *root*, which is the root node of the binary tree. (20 minutes)

Hint: As long as your answer is a component of your function return then it is fine (Eg. Tuple with multiple values and one of the values being a boolean value).

```
def is_perfect(root):
    ''' Returns True if the Binary Tree is perfect and
    false if not using recursion'''
```

ex)