- sThis lab will cover Queues.
- It is assumed that you have reviewed chapters 6 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!
- If you finish early, you may leave early after showing the TA your work. Or you may stay and 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. What is the output of the following code?

```
q = ArrayQueue()
i = 2

q.enqueue(1)
q.enqueue(2)
q.enqueue(4)
q.enqueue(8)

i += q.first()
q.enqueue(i)
q.dequeue()
q.dequeue()
print(i) #3
print(q.first()) #4
```

2. Describe what the following function does and give it an appropriate name. Trace the function with a queue of integers

```
def remove_evens_and_reverse_odds(q):
    if (q.is_empty()):
        return

    else:
        val = q.dequeue()
        remove_evens_and_reverse_odds(q)
        if val % 2 != 0:
            q.enqueue(val)
```

3. Trace the following function, which takes in a stack of integers. Describe what the function does, and give a meaningful name to the function:

```
def is_palindrome(input_str):
    s = ArrayStack()
    q = ArrayQueue()

    for char in input_str:
        s.push(char)
        q.enqueue(char)

    while not s.is_empty():
        if s.pop() != q.dequeue():
            return False

    return True
```

4. Fill out the prefix, infix, postfix table below:

Prefix	Infix	Postfix	Value
- * 3 4 10	3 * 4 - 10	3 4 * 10 -	2
+ * 5 5 / 10 2	(5 * 5) + (10 / 2)	5 5 * 10 2 / +	30
+ / - 10 2 4 8	(10 - 2) / 4 + 8	10 2 - 4 / 8 +	10
+ * 6 3 * 8 4	(6 * 3) + (8 * 4)	6 3 * 8 4 * +	50
- + * 8 2 4 + 3 6	(8 * 2) + 4 - (3 + 6)	8 2 * 4 + 3 6 + -	11

5. Consider the "circular" array implementation of a queue, similar to `ArrayQueue` that we studied in class, where the only difference is that the initial capacity is set to 4

```
class ArrayQueue:

    INITIAL_CAPACITY = 4

    def __init__(self):
```

```
        self.data_arr= make_array(ArrayQueue.INITIAL_CAPACITY)
        self.num_of_elems = 0
        self.front_ind = None
    def __len__(self): ...
    def is_empty(self): ...
    def enqueue(self, elem): ...
    def dequeue(self): ...
    def first(self): ...
    def resize(self, new_cap): ...
```

Show the values of the data members: `front_ind`, `num_of_elems`, and the contents of each `data_arr[i]` after each of the following operations. If you need to increase the capacity of `data_arr`, add extra slots as described in class.

operation	front_ind	num_of_elems	data_arr
q=ArrayQueue()	None	0	[None, None, None, None]
q.enqueue('A')			
q.enqueue('B)			
q.dequeue()			
q.enqueue('C')			
q.dequeue()			
q.enqueue('D')			
q.enqueue('E')			
q.enqueue('F')			
q.enqueue('G')			
q.enqueue('H')			

operation	front_ind	num_of_elems	data_arr
q = ArrayQueue()	None	0	[None, None, None, None]
q.enqueue('A')	0	1	['A', None, None, None]
q.enqueue('B')	0	2	['A', 'B', None, None]

q.dequeue()	1	1	[None, 'B', None, None]
q.enqueue('C')	1	2	[None, 'B', 'C', None]
q.dequeue()	2	1	[None, None, 'C', None]
q.enqueue('D')	2	2	[None, None, 'C', 'D']
q.enqueue('E')	2	3	['E', None, 'C', 'D']
q.enqueue('F')	2	4	['E', 'F', 'C', 'D']
q.enqueue('G')	0	5	['C', 'D', 'E', 'F', 'G', None, None, None]
q.enqueue('H')	0	6	['C', 'D', 'E', 'F', 'G', ‘H’, None, None]