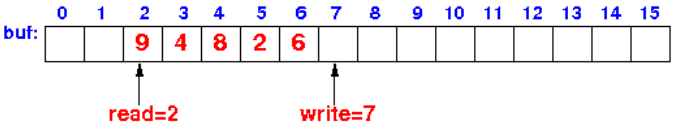
Lab Week 3: Queue implementations

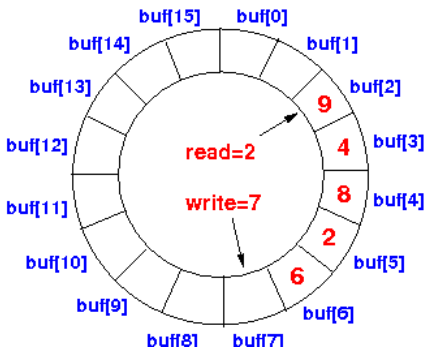
Goal:

- **Implement a Queue class using a circular array**

Before doing this lab make sure to read the class notes and review the class slides carefully.

- The implementation will use a circular array for storing the items in the queue. There are different implementations but they share the idea presented in the picture. Namely elements are added to the rear of the queue using the next "free entry" in an array. (rear may be the index of the current last index storing an item in the queue or the index beyond it Elements are removed from the front(read in the picture) of the queue. Elements are added to the queue at the rear (write) of the queue. The indices in front and rear are incremented as elements are added and deleted from the queue. All arithmetic is **modulo the size of the array structure allocated to store the queue.** Care must be taken in distinguishing a full queue from an empty queue and this can be done in different ways. The specification below enables you do to this through the `num_in_queue` field. **Understand the implications of not having this field!**
- Note that in some implementations that maximum number of items the queue can hold is one less than the size of the array-like structure allocated. When a user of your Queue class specifies its capacity, you may have to take this into account so that the queue can actually store capacity items.





Use the following specification.

Make sure that each of your functions has a docstring that describes its purpose.

```
class Queue:
    def __init__(self, capacity):
      self.capacity = capacity
                  # the maximum number of items that can be stored in queue
      self.num_in_queue = 0
                  # the number of items stored in queue
      self.front = ??
      self.rear = ???

    def is_empty(self):

    def is_full(self):

    def enqueue(self, item):

    def dequeue(self):

    def num_in_queue(self):
    “””returns the number of items in the queue”””
```

- The class **Queue** should raise an **IndexError** exception if a user of your implementation attempts to enqueue an item when the queue is full or dequeue an item with the queue is empty.

Submit the following two files to Canvas:

- **queue_nodelist.py**: Contains a linked based implementation of the **Queue** class
- **queue_tests.py:** Contains comprehensive tests to ensure your implementation of Queue workscorrectly.