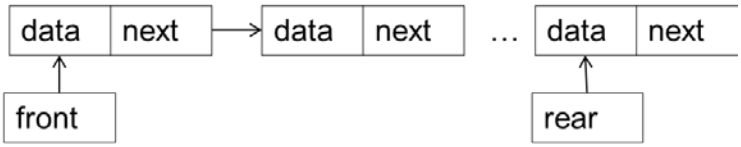
Lab Week 3: Queue implementations

Goal:

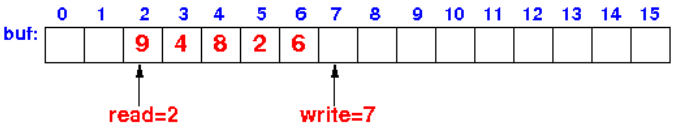
- **Implement a Queue class using a circular array: Class QueueArray**
- **Implement a Queue class using a linked data structure: Class QueueLinked**

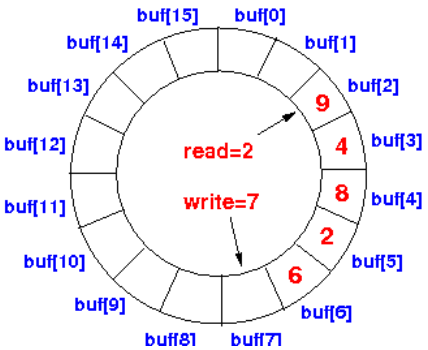
Before doing this lab make sure to read over section 3.10, 3.11, 3.12 carefully. You will not be doing the implementation as described in section 3.12, but it will help you understand the underlying concepts.

- The first implementation you will use a linked structure similar to the linked structure used in implementing the Stack ADT. In this case, there must be a way to add items to the back of the list and remove items from the front of the list. See picture.



- The second implementation will use a circular array for storing the items in the queue. There are different ways of doing this 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 one 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 from an empty queue and this can be done in different ways. You should follow your instructor's advice on the best way to do this. (Why is this better than what the text does?)
- 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.

Submit the following two files to PolyLearn:

- **queues.py** containing a list based implementation of queue and a linked implementation of queue. The classes must be called: **QueueArray** and **QueueLinked** . Both implementations should follow the specification and be thoroughly tested.
- Raise an IndexError exception if a user of your implementation attempts to enqueue an item when the queue is full or dequeue and item with the queue is empty.
- **test_queues.py** contains your set of tests to ensure you classes work correctly

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

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

    def is_empty(self):

    def is_full(self):

    def enqueue(self, item):

    def dequeue(self):

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